INFERENCE OF STRING MAPPINGS FOR LANGUAGE TECHNOLOGY

DISSERTATION

Presented in Partial Fulfillment of the Requirements
for the Degree Doctor of Philosophy
in the Graduate School of The Ohio State University

By
Martin Jansche, M.A.

* * * * *

The Ohio State University
2003

Dissertation Committee:

Chris Brew, Adviser
W. Detmar Meurers
Gerald Penn
Richard Sproat

Approved by

Adviser
Department of Linguistics

ABSTRACT

Mappings between formal languages play an important role in speech and language processing. This thesis explores issues related to inductive inference or learning of string-to-string mappings. The kinds of mappings considered fall within the larger class of rational transductions realized by finite state machines. Such mappings have applications in speech synthesis, speech recognition, and information retrieval and extraction. The present work takes its examples from speech synthesis, and is in particular concerned with the task of predicting the pronunciation of words from their spelling. When applied to this task, deterministic mappings are also known as letter-to-sound rules.

The three most commonly used metrics for evaluating letter-to-sound rules are prediction error, which is not generally applicable; string error, which can only distinguish between perfect and flawed pronunciations and is therefore too coarse; and symbol error, which is based on string edit distance and subsumes string error. These three performance measures are independent in the sense that they may prefer different models for the same data set. The use of an evaluation measure based on some version of string edit distance is recommended.

Existing proposals for learning deterministic letter-to-sound rules are systematized and formalized. Most formal problems underlying the learning task are shown to be intractable, even when they are severely restricted. The traditional approaches based on aligned data and prediction error are tractable, but have

other undesirable properties. Approximate and heuristic methods are recommended. The formalization of learning problems also reveals a number of new open problems.

Recent probabilistic approaches based on stochastic transducers are discussed and extended. A simple proposal due to Ristad and Yianilos is reviewed and recast in an algebraic framework for weighted transducers. Simple models based on memoryless transducers are generalized to stochastic finite transducers without any restrictions on their state graphs. Four fundamental problems for stochastic transducers (evaluation, parameter estimation, derivation of marginal and conditional models, and decoding) are identified and discussed for memoryless and unrestricted machines. An empirical evaluation demonstrates that stochastic transducers perform better on a letter-to-sound conversion task than deterministic mappings.

ACKNOWLEDGMENTS

Columbus – Euro trash.
Jerry Seinfeld, in the *Seinfeld* episode *The Library*

This thesis would not exist in its present form, if at all, were it not for the advice, encouragement, and support I received from many individuals. Needless to say, the usual exculpatory remarks are in effect.

I'd first like to thank my thesis committee, Chris Brew, Detmar Meurers, Gerald Penn, and Richard Sproat. Chris has been an outstanding adviser, knowledgeable, generous with his time, pointing out new research directions, and always pushing for clarity. Detmar provided detailed feedback on many empirical and presentational aspects, and pointed out connections to other work that had previously gone unnoticed. Gerald played many crucial roles at various stages of this project: as my mentor at Bell Labs during the summer of 2000 he got me interested in machine learning of letter-to-sound rules, and has provided insightful feedback on many algorithmic and mathematical issues. Richard has always been a source of valuable and virtually instantaneous advice, and many questions discussed here originated in conversations with him.

I'd further like to thank the Department of Linguistics, the Department of Computer and Information Science, and the Graduate School for their support. I am grateful for the liberal and undogmatic environment at Ohio State that allowed me to pursue many diverse interests. The influence of several memorable

courses is visible at many points in this thesis: the seminar on finite state methods in language processing taught by Bob Kasper and Erhard Hinrichs in Sp/99 sparked my interest in finite state language processing; Gerald Baumgartner's course on compiler implementation in Au/99 gave me the background for turning this general interest into working programs; Chris Brew's statistical NLP seminar in Sp/00 got me thinking about machine learning; and Rephael Wenger's courses on computational complexity in Sp/00 and Au/00 provided the background for much of Chapter 3 of this thesis.

A major source of learning and inspiration came from internships at Bell Laboratories and AT&T Labs. I'd like to thank Steve Abney, Michiel Bacchiani, Bob Carpenter, Jennifer Chu-Carroll, Julia Hirschberg, George Kiraz, Gerald Penn, Brian Roark, Chilin Shih, Richard Sproat, and Evelyne Tzoukermann for making my summers in New Jersey educational and enjoyable.

I'm indebted to the following people who answered technical questions or sent me copies of publications: Antal van den Bosch, Alexander Clark, Jason Eisner, Mehryar Mohri, Michael Riley, and Izhak Shafran. There are undoubtedly others who were inadvertently left out. Whoever you might be: please think nothing of it and accept my apologies.

At Ohio State, the help of Chris Brew, Carl Pollard, and Tim Watson in addressing the many bureaucratic challenges was indispensable.

Special thanks go to Shravan Vasishth for many conversations about academic issues, computing, empiricism, life, logic, music, statistics, and the theory of grammar; to Nathan Vaillette for unwittingly being a role model; and to Lijun Feng for her love, support, and understanding.

To Restaurant Japan a final thanks for all the fish.

VITA

1973	Born in Karlsruhe, Germany
1995	*Zwischenprüfung* Chinese Studies, University of Heidelberg
1996	*Zwischenprüfung* Computational Linguistics, University of Heidelberg
1998	M. A. Linguistics, The Ohio State University
1998–2002	Graduate Teaching and Research Associate, The Ohio State University

PUBLICATIONS

1. Martin Jansche. Learning local transductions is hard. *Mathematics of Language 8*, 2003.

2. Martin Jansche and Steven P. Abney. Information extraction from voicemail transcripts. *Empirical Methods in Natural Language Processing*, 2002.

3. Martin Jansche. Information extraction via heuristics for a movie showtime query system. *Eurospeech 7*, 2001.

4. Martin Jansche. Re-engineering letter-to-sound rules. *NAACL 2*, 2001.

FIELDS OF STUDY

Major Field: Linguistics

Area of Specialization: Computational Linguistics

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

> *It was a reaction I learned from my father: have no respect* whatsoever *for authority; forget who said it and instead look at what he starts with, where he ends up, and ask yourself, "Is it reasonable?"*
>
> Richard P. Feynman, *What do* you *care what other people think?*

1.1 Motivation

Many language and speech processing applications can be naturally broken down into a number of common subtasks, such as classification, ranking, clustering, or translation. This thesis focuses on translation tasks which can be conceptualized as relations or mappings between formal languages. More specifically, the mappings considered here are of a very restricted kind, namely those that are at most regular (finite state) or even more restricted. Despite these restrictions, such mappings can play useful roles in several areas, including:

Speech synthesis Every orthographic word must be mapped to a phonemic representation. Frequent and/or exceptional words are typically looked up in a pronunciation dictionary. However, dictionaries are finite, but there seems to be no absolute limit to the number of previously unseen words encountered even after huge amounts of text have been seen [Kornai, 1999, 2002; Baayen, 2001]. A text-to-speech system must assign pronunciations

to all words, including previously unseen ones. For certain highly regular ("shallow") writing systems this is not very problematic (for example all commonly used transcription systems of Mandarin Chinese are essentially phonemic in nature), or it amounts to a classification problem (the standard writing system for the Chinese languages assigns only a few pronunciations to each character; disambiguating which pronunciation is most likely correct in a given context is a classification problem). But for languages like English with rather "deep" orthographies [Sproat, 2000], previously unseen words must be given a pronunciation based mostly on their orthography. The problem is then one of inferring from a pronunciation dictionary a suitable mapping from orthographic strings to phonemic strings, also known as a letter-to-sound mapping.

Speech recognition Automatic speech recognizers are usually trained on a corpus of speech data aligned with their transcriptions. Such corpora are typically constructed by human transcribers in a long and labor-intensive process. There are however situations where both speech data and the corresponding textual transcriptions are available, for example broadcast news with closed-captioning, or name dialing applications where each participant speaks and types in their own name. In such situations one can induce mappings between letters and the phonemes that have been produced by a preliminary acoustic model. Such mappings are useful for segmenting unlabeled data into words, aiding automatic transcription, or for constructing lexical databases for speech recognizers. The learning task is quite different from the one associated with speech synthesis: since both an orthographic and a phonemic representation are available the mediating mapping can

theoretically be simpler (less information is needed to go from letters to phonemes, since the phonemes are already known); on the other hand one has to deal with uncertainty in the phonemic transcription – in fact, multiple competing transcriptions are usually provided when the task of transcribing is carried out automatically. Typically this need for dealing with uncertainty calls for stochastic models of rational transductions.

Information extraction Information extraction often amounts to mapping from linear textual representations to labeled records ("feature structures", "slot and filler" representations, or "frames"). Often the extracted information has to be standardized. This is most obvious for information extraction from speech transcripts [see Jansche and Abney, 2002], but holds as well for information extraction from text: there are several ways to say aloud, or write down, numbers and alphanumeric sequences. For example, an utterance including the words 'fourteen ninety-five' might refer to a monetary amount like $14.95 or to the year 1495. Context may be used to disambiguate, and, depending on what kind of entity 'fourteen ninety-five' is, it may have to be transduced to different representations. Since most numbers that one encounters in text are small, such transductions are, for all practical purposes, rational [Sproat, 2000]. Automatic acquisition of such transductions may be desirable, especially if the number of different entities (years, monetary amounts, telephone numbers, social security numbers, room numbers, door lock combinations, tax form numbers, etc.) is large.

Information retrieval Information retrieval may benefit from a measure of the phonemic similarity between different orthographic forms. A standard example is searching a name dictionary: a user searches for a person named

'McCune' whose name may be recorded in the dictionary as 'Macune', 'McKeown', 'MacEwan', etc. This can be accomplished by a crude phonemic hashing algorithm like Soundex [Russell, 1918], or may call for a mapping between letters and phonemes that assigns scores to likely pronunciations of a given letter string, or to likely transcriptions of a given phoneme string. Stochastic transducers may again be suitable.

Historical linguistics Genetic relationships between languages are established on the basis of a convincing inventory of cognates, i. e., pairs of semantically similar words from both languages whose phonemic representations exhibit a *systematic* relationship. Generally it is the fact that a systematic mapping can be established between reasonably sized subsets of the vocabularies of the two languages that counts as evidence of relatedness, whereas superficial similarities do not. A typical example is Latin 'habere' and German 'haben' (which overlap semantically in the meaning 'to have'): they may look similar, but this superficial similarity does not give rise to a regular, systematic correspondence on a larger scale. Rather, one can observe a systematic correspondence between Latin initial ⟨c⟩ /k/ and German/ English ⟨h⟩ /h/ ('cornu' – 'Horn' – 'horn'; 'cor', 'cordis' – 'Herz' – 'heart'; 'canis' – 'Hund' – 'hound'; 'centum' – 'hundert' – 'hundred'), among other, similarly systematic correspondences summarized by Grimm's Law. The identification of cognates, appropriately constrained, amounts to inference of a phoneme-to-phoneme mapping over a large dictionary of semantically similar words. Systematicity of the mapping can be measured, for example in terms of entropy. This would allow us to *quantify* the overall regularity of sound change, i. e., it allows us to talk about degrees of Neogrammaticality.

The concrete examples in this thesis all pertain to letter-to-sound mappings used in speech synthesis systems. However, as is clear from the discussion of the application areas, the requirements that speech synthesis alone would impose on the learning task are somewhat impoverished: for example, it is not immediately obvious from the task description that speech synthesis would benefit from stochastic transduction models. Considering the broader range of applications discussed above, stochastic models are most widely applicable and can also be beneficially applied to the task of learning letter/phoneme correspondences for speech synthesis.

1.2 Letter-to-Sound Conversion

Letter-to-sound conversion determines the pronunciation of a word on the basis of its orthographic form. For singling out the orthographic level, the following notation is used:

⟨letters⟩

The pronunciation of a word is represented as a string of phonemes. Phoneme symbols are taken from the International Phonetic Alphabet (IPA) [International Phonetic Association, 1999]. Another notational device is used to indicate the phonemic level:

/fonimz/

Klatt [1987] provides an excellent overview of speech synthesis and delineates the role of letter-to-sound conversion in speech synthesis systems. For a longer introduction see Dutoit 1997; for a more detailed description of text analysis see Sproat 1997; Sproat et al. 1998. One of Klatt's [1987] figures is reproduced here as

Figure 1.1 and shows a high-level flowchart for obtaining the pronunciation of an isolated word. Letter-to-sound rules appear under the label 'L-T-S'.

Klatt [1987] clearly did not intend to give a definitive description of the architecture of a speech synthesizer. Many variations are possible, and many aspects are language specific. For example, 'affix stripping' and 'affix reattachment' in Figure 1.1 seem to be biased toward the morphologies of European languages. A more neutral term would be *morphological analysis*, which clearly has some role to play for letter-to-sound conversion [van den Bosch, 1997]. However, we follow Klatt [1987] and view morphological analysis as a separate and distinct processing step [see also van den Bosch, 1997].

The flowchart mentions a few of the modules that can play a role in finding the pronunciation of a word, and other modules are conceivable. The relative importance of these modules will vary from language to language. For example, English has a fair number of homographs whose pronunciation may depend on word sense and/or part of speech. One could argue that letter-to-sound conversion for the Chinese languages [Shih and Sproat, 1996] is predominantly a homograph disambiguation problem, since all standardized character sets used for Chinese are finite and each character has a small, known inventory of possible pronunciations. We ignore the issue of homograph disambiguation in order to keep the problem we address simple and uniform.

We are interested in the letter-to-sound conversion task as an isolated problem, since it is that perspective that makes any techniques developed for it widely applicable. The problem is simply one of mapping from a string of orthographic symbols to a string of phonemic symbols. While additional information such as word sense, part of speech, morphology, or even etymology [Font Llitjós, 2001]

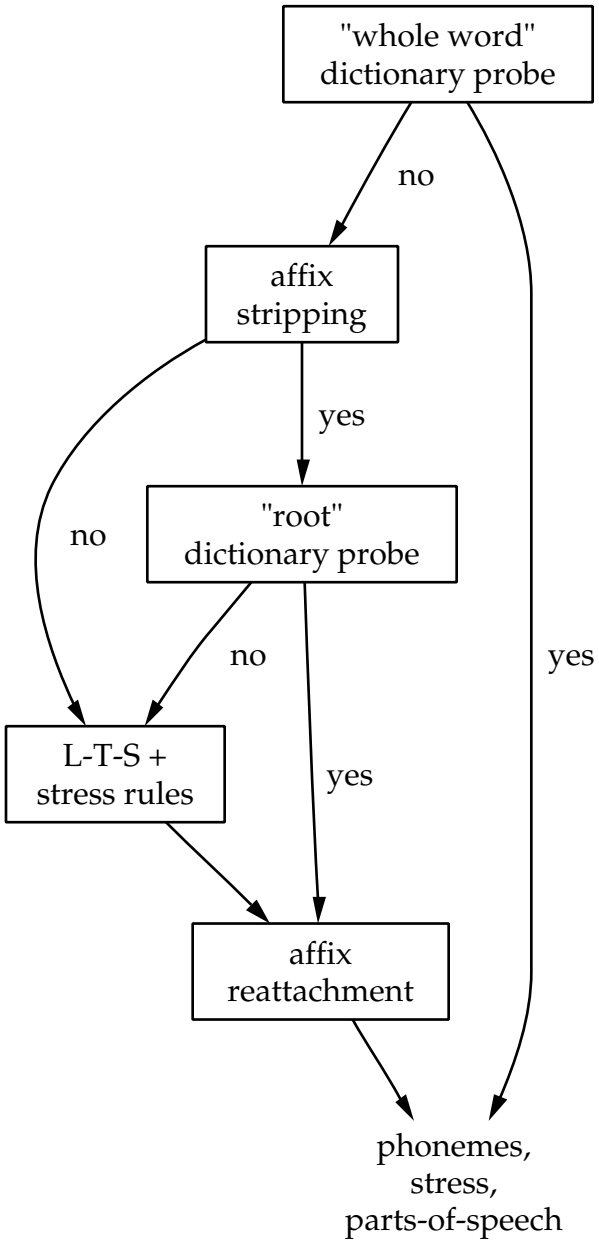

Figure 1.1: The world according to Klatt [1987].

may be helpful in practice, they would only serve to complicate the abstract problem we are concerned with. Note also that Figure 1.1 distinguishes between letter-to-sound rules and stress rules, and we will do the same. Viewed abstractly, the task of predicting stress symbols is very similar to predicting phoneme symbols. We prefer to focus on a single coherent task as an illustration of the general learning problems we are interested in. That task will be letter-to-sound conversion, excluding stress assignment.

Letter-to-sound conversion is often referred to as *grapheme-to-phoneme* conversion [see for example Galescu and Allen, 2001; Rentzepopoulos and Kokkinakis, 1991]. For some authors the terms *letter* and *grapheme* are completely interchangeable. For others they are not. Some would prefer it if English had graphemes like ⟨th⟩, ⟨mb⟩ (as in ⟨thumb⟩), or ⟨bt⟩ (as in ⟨debt⟩). The choice of grapheme inventory is immaterial from a purely formal point of view: a learning algorithm that works for the grapheme inventory $\{a, b, c, \ldots, z\}$ and grapheme strings like ⟨t h u m b⟩ will also work for a grapheme inventory of $\{a, b, c, \ldots, z, th, mb, \ldots\}$ and grapheme strings like ⟨th u mb⟩, since only a change in alphabets is involved.

However, assuming the availability of graphemic strings like ⟨th u mb⟩ begs the question of how the graphemic parse was obtained from the letter string ⟨thumb⟩. A deterministic preprocessing step that does not become a new source of errors is hardly ever possible [van den Bosch, 1997], except of course for writing systems or transcription systems that are close to the phonemic representation by design (for example, the Pinyin transcription of Mandarin Chinese), but in that case there is hardly any need for graphemic parsing.

A major source of concern is the problematic status of a parsed grapheme string as an intermediate representation: how much agreement would one find if one were to ask native speakers to segment orthographic words graphemically? If

there is no commonly agreed upon standard for parsing orthographic words into grapheme strings, we are left with an intermediate representation whose quality can only be evaluated indirectly by its usefulness for any subsequent processing steps. This is undesirable, since it makes separate development of the preprocessing component impossible, and end-to-end training of the overall module is also likely to be hard. Fortunately, it is possible to stay agnostic about the status of graphemes as far as the learning problems are concerned. We will see another example of a problematic intermediate representation later, namely alignments of letter and sounds, which cannot be glossed over or ignored as easily.

The literature on machine learning of letter-to-sound rules falls into roughly two major classes, depending on the kinds of models used: *deterministic* mappings that produce a single pronunciation for a given letter sequence; and *stochastic* nondeterministic mappings that assign probabilities to all conceivable pronunciations of a given letter sequence. All approaches are essentially *symbolic* in nature, in the sense that they map from discrete representations (letter strings) to discrete representations (phoneme strings).

Within the deterministic tradition NETtalk [Sejnowski and Rosenberg, 1987] has sought and received wide attention, and has served as the blueprint or catalyst for many subsequent approaches. The method starts with aligned data and uses a sliding window (which ensures locality of the mapping) to produce training instances. Those training instances are then handed to a classifier learner; in the case of NETtalk artificial neural networks are used, but many other classifier learners have been proposed, including decision trees [Lucassen and Mercer, 1984], memory-based learning [Stanfill and Waltz, 1986], default rules [Hochberg et al., 1991], and transformation-based learning [Huang et al., 1994]. Despite such

superficial diversity, all these approaches share a common shortcoming: they require an aligned training corpus. Aligning a pronunciation dictionary can be done manually or (semi-)automatically, but it has to be kept in mind that this is fundamentally an optimization task: find the best alignment. What is the objective function that is being maximized, i. e., which alignment is best? In this case the best answer is: an alignment that enables the classifier learner to optimally exploit any and all regularities in the data. This suggests that learning deterministic mappings may be quite hard in general, since evaluating the objective function of the alignment step would require another optimization (learning a classifier) to be carried out. This thesis gives a formal characterization of the hardness of a few variants of this problem. It turns out that even simple variants are intractable in the general case, even if we only require approximate solutions.

Stochastic approaches often suffer from the same problem of requiring aligned data [for example Rentzepopoulos et al., 1993; Minker, 1996]. This is unfortunate, since it is in fact possible to formulate parameter estimation procedures based on the EM algorithm that do not require an explicit alignment of the data. Such an algorithm was described by Ristad and Yianilos [1998] for a very simple class of stochastic transducers, although it seems to have been known since much earlier. (This situation is reminiscent of the formulation of the EM algorithm itself [Dempster et al., 1977].) This thesis shows how a parameter estimation algorithm for memoryless transducers [Ristad and Yianilos, 1998] can be generalized and applied to general stochastic finite transducers.

1.3 Overview of the Thesis

Chapter 2 begins with an overview of evaluation measures and data sets that could be used for evaluating a letter-to-sound component. Several metrics have been proposed for measuring the quality of the pronunciations predicted by a letter-to-sound component. The simplest one is called string error and is analogous to misclassification loss for classification tasks. However, this analogy is not particularly good, since unlike in ordinary classification tasks the predictions have internal structure. A more fine grained measure is symbol error, which is based on weighted string edit distance. A third measure, prediction error, is of limited generality and is closely tied to traditional approaches to learning letter-to-sound rules, which are discussed in Chapter 3.

Section 2.4 studies the relationships between the three main loss functions discussed in Section 2.3. Using realistic, but biased, excerpts from existing pronunciation dictionaries, we show that the three evaluation measures give rise to independent optimization problems: for example, minimizing string error often leads to an increase in symbol error, and vice versa. We argue that string error is a very crude performance measure and is of limited practical use, since it provides very little information that can be used to compare different approaches.

Chapter 3 is concerned with on deterministic approaches to learning letter-to-sound rules, which includes many traditional proposals found in the literature. In Section 3.2 an approach based on grammatical inference is reviewed and several desiderata for an ideal algorithm are identified. Ideally, an algorithm should be able to take a standard pronunciation dictionary as training data. In that sense, the traditional approaches described in Section 3.3 are far from ideal, since they require input data in a specific format, which we refer to as *aligned* data. Formally,

all that is required is that letter strings and corresponding phoneme strings are of the same length, so that there is a natural correspondence between the nth letter of a word and the nth phoneme. This requirement, discussed in Section 3.3.2, holds for the traditional approaches which reduce the letter-to-sound problem to a classification problem. The key insight is to treat a letter plus some fixed amount of surrounding context (and potentially other features like the part-of-speech of the word) as an instance that is labeled with a phoneme or other transcription symbol. The induced classifier predicts these labels, and one can then apply it to each letter of a word in succession and string together the predicted labels to form the predicted pronunciation. During supervised training the label assigned to a letter must be known, hence the need for aligned data. This requirement is problematic because pronunciation dictionaries do not normally have this same-length property, nor can alignments be naturally observed. Moreover, the quality of the alignment determines the quality of the classifier induced on the basis of aligned data, but virtually all traditional approaches start with one alignment that is never modified or influenced by the induced classifier. The traditional approaches suffer from another shortcoming, namely the evaluation measure that serves as the optimization objective during their training is prediction error, which is highly task specific.

In light of these issues, we ask two more or less orthogonal questions: First, what happens if one tries to learn from unaligned data, discovering simple alignments automatically as an integral part of learning? Second, what are the consequences of using other evaluation measures as optimization objectives? In order to simplify the discussion, Section 3.4 reduces the traditional approaches to a function learning problem involving two related and highly restricted classes of functions. The properties of the formal learning problem associated with such

function are discussed in Section 3.5. We show in Section 3.5.1 that automatically discovering alignments while searching for a consistent hypothesis is a hard problem under many reasonable optimization objectives. But even if we do not require consistent hypotheses and do not need to discover alignments as part of learning, finding an optimal hypothesis is also difficult under many reasonable loss functions, as discussed in Section 3.5.2. These results provide some new justification for the traditional approaches, which have managed to avoid a number of computationally challenging problems. Showing that a problem is principle very hard also provides a challenge to come up with approximate or heuristic solutions. However, there are more opportunities for heuristics and simplifying assumptions in a nondeterministic probabilistic scenario.

Such a setting is the topic of Chapter 4. It reviews an existing approach that uses a very simple class of weighted nondeterministic translation devices known as memoryless transducers. These are weighted finite state machines with a fixed, trivial state graph. There are four fundamental problems associated with weighted and/or stochastic finite transducers. The problem of computing the weight or probability they assign to a given pair of strings is discussed in Section 4.2. The second problem, parameter estimation, discussed in Section 4.3, is about adjusting the parameters of a stochastic transducer on the basis of training data. The third problem, appearing in Section 4.4, is the task of deriving a transducer for the conditional distribution given a joint distribution represented by another stochastic transducer. The fourth problem, in Section 4.5, is about using a transducer for prediction: given an input string, what is the best output string? While reviewing some of the solutions that have been proposed for memoryless stochastic transducers, Chapter 4 casts the problems in terms of operations on weighted

transducers and provides some background on the algebraic and other concepts used in conjunction with weighted finite state machines.

The perspective developed in Chapter 4 is used in Chapter 5 to generalize the approach based on memoryless transducers. Memoryless transducers are of limited expressivity, but have the advantage that many algorithms are particularly straightforward in this special case. Many of the concepts developed for the memoryless case carry over to the more general setting, but the algorithmic issues are more intricate. The same four fundamental problems that provided the structure for Chapter 4 apply unchanged, but the answers given in Chapter 5 are different. The algorithms for computing the probability of a pair of strings (Section 5.2) depend partly on the graph structure of the transducer. In the worst case, an all-pairs algebraic path algorithm can be used, after some modifications that make it applicable to stochastic transducers. Parameter estimates (Section 5.3) can be derived along the same lines as for memoryless transducers, and the algorithm we provide is less costly, though less general, than an alternative from the recent literature. Deriving conditional transducers (Section 5.4) is a bit more involved than for the memoryless case, and involves several operations on weighted transducers. Among those is ϵ-removal, which requires some corrections to an existing algorithm to work properly. Finding the best output (Section 5.5) is essentially the same as for memoryless machines. However, we also discuss a different notion of "best" in terms of the recommended evaluation measure from Chapter 2.

Chapter 6 focuses on empirical and practical issues. The relationship between prediction error and symbol error is often complex in practice (Section 6.2): the numbers for raw prediction error and raw symbol error tend to be similar, so that minimizing prediction error can be seen as a heuristic for minimizing symbol error. Direct minimization of string error is explored in Section 6.3 and Section 6.4,

first using exhaustive search, then in terms of greedy local hill-climbing, but ultimately it has to be abandoned. Section 6.5 asks how well approaches based on classifier learning can ultimately perform: the state of the art is still quite a bit away from its upper performance bound, and perfection seems to be out of reach. In Section 6.6 we investigate whether cutting corners during parameter estimation and decoding for stochastic transducers is permissible, before comparing our approach to letter-to-sound conversion, which is based on stochastic transducers, with a traditional approach based on classifiers in Section 6.7. Finally in Section 6.8 we point toward fundamental statistical modeling issues surrounding stochastic transducers that need to be addressed in future work.

CHAPTER 2

EVALUATION METRICS AND DATA

2.1 Introduction

The goal of evaluating a letter-to-sound module is to arrive at an assessment of its performance on a set of test data and an estimate of its expected performance on unseen future data.

Not too long ago, Divay and Vitale [1997, p. 516] rightly complained that 'no standardized tests exist for evaluating letter-to-sound systems'. The situation remains essentially unchanged, even though in general more emphasis is now being placed on systems evaluation [Hirschman and Thompson, 1997], and generic recommendations have been formulated for evaluating TTS systems [Pols, 1997] and their letter-to-sound components [Gibbon et al., 1997, pp. 513–515].

The perceived quality of a letter-to-sound component can depend on many factors and should ideally be assessed in the context of a full TTS system evaluation [Gibbon et al., 1997, ch. 12]. For example, the letter-to-sound component might predict an "incorrect" phoneme string that happens to coincide with an attested pronunciation variant; or a small segmental error might be introduced without any confusion because the word form could already be determined unambiguously before the error occurred; or a correct segment could have been replaced by

a phonetically very similar segment without leading to confusion, e. g. /wæbɪt/ may be close enough (thanks partly to Mel Blanc) to the pronunciation of ⟨rabbit⟩ to not cause any major confusion. On the other hand, in densely populated lexical neighborhoods even the slightest error might cause confusion. Furthermore, it is possible that a later module renders an otherwise good prediction by the letter-to-sound component unintelligible, for example, when a concatenative synthesizer chooses a bad unit or has to modify and distort a poorly matching unit.

One kind of evaluation would involve human listeners who would be asked to judge the intelligibility of synthetic speech where all aspects of the system were held constant except for the letter-to-sound component. However, that kind of evaluation is beyond the scope of the present work, as it would involve a full TTS system. Moreover the results of such an end-to-end evaluation would depend on the quality of those components that feed into the letter-to-sound module and those that consume its output. In order to do well on an end-to-end evaluation a letter-to-sound module should have been optimized using end-to-end training. End-to-end evaluation would then assess the contribution of a letter-to-sound module to an entire speech synthesizer, which could be larger than if the letter-to-sound module was run in isolation (when the preceding text analysis makes mistakes that the letter-to-sound module can detect and correct), or lesser (when subsequent components introduce mistakes that degrade intelligibility).

In practice it is much easier and less costly to assess the performance of a letter-to-sound module separately and in isolation from any other components (unit testing). This avoids the dependencies that can arise in end-to-end evaluation and is thus more suitable for initially comparing different techniques for letter-to-sound conversion. Typically this is done by comparing the output of the

module on isolated words with the corresponding entries in a pronunciation lexicon or similar reference standard ("gold standard") [Gibbon et al., 1997, pp. 513–515]. Such comparisons have recently been carried out for a small number of languages and TTS systems [Pols, 1997; Yvon et al., 1998; Damper et al., 1999]. While these comparisons provide valuable insights into the relative performance of existing letter-to-sound components, many of the systems that were evaluated had been built without reference to the evaluation metrics used for comparison – or, in a few cases, without any consideration for rigorous performance evaluation [Damper et al., 1999, pp. 162–163]. A modern development approach should establish an evaluation metric from the outset, so that this metric can be incorporated into the objective functions for any optimization steps during development, so that, ideally, the common situation of development using one criterion and evaluation a different criterion can be avoided.

As some evaluation metrics depend on alignment information, which is absent from many data sets, we begin with a review in Section 2.2 of lexical resources that one might use for development or evaluation. Section 2.3 describes the three most commonly used evaluation metrics, which measure various deviations from reference pronunciations recorded in a data set. All of these metrics take on nonnegative values and are therefore suitable as loss functions or similar objective functions. They are compared and contrasted in Section 2.4, where it is demonstrated that they may select different optimal models on the same data set. Section 2.5 defines a corresponding accuracy measure for each error measure and discusses the differences between maximizing accuracy vs. minimizing error, and also reiterates the point that the evaluation metric should ideally also be the objective optimized during model training.

2.2 Data Sets

The availability of certain evaluation metric is dependent on properties of the data sets used for development or evaluation. We will briefly describe two pronunciation dictionaries for American English that are available as text files.

The number of speech corpora and lexical databases that can be used to as a reference standard against which letter-to-sound systems can be measured is growing steadily [Gibbon et al., 1997, ch. 6]. For English (both British and American English) the following resources exist and have been used to evaluate the text analysis components of TTS systems:

CELEX [Baayen et al., 1993, 1996]. The English part contains 160,595 word form entries (including multi-word units) and their British English pronunciations, plus information about syllable structure, morphology, frequency, etc.

CUVOALD Computer-Usable Version of the Oxford Advanced Learner's Dictionary [Mitton, 1992]. This version contains 70,646 words with their British English pronunciations plus part-of-speech tags, rudimentary frequency information, and syllable counts.

CMUdict The Carnegie Mellon Pronouncing Dictionary [Weide, 1998]. It contains 127,069 entries of words and their American English pronunciations. Lower in quality than the other dictionaries, it has quite a few errors and inconsistencies. More details are provided below.

PRONLEX [Kingsbury et al., 1997]. It contains 90,988 entries of words with American English pronunciations, and was designed to cover the vocabulary of two large speech corpora distributed by the Linguistic Data Consortium.

NETtalk data set [Sejnowski, 1988]. Based on Webster's Pocket Dictionary, it contains 20,008 word forms aligned with their American English pronunciations. More details are provided below.

HML Hoosier Mental Lexicon [Nusbaum et al., 1984]. Also based on Webster's Pocket Dictionary, it contains 19,321 word forms together with their American English pronunciations, parts of speech, syllable structures, familiarity ratings, etc. The selection of entries is somewhat peculiar: for example, it contains an entry for ⟨yes⟩, but not for ⟨no⟩; there are entries for ⟨Monday⟩ through ⟨Saturday⟩, but not for ⟨Sunday⟩, though there is one for ⟨sundae⟩; and the digits from zero to nine are ⟨zero⟩, ⟨one⟩, ⟨three⟩, ⟨five⟩, ⟨six⟩, ⟨seven⟩, and ⟨nine⟩, i. e., the positive integral powers of two are missing.

Any approach based on mixing and matching different resources is bound to run into serious difficulties. Constructing a unified dictionary of words common to all of them, merging two dictionaries, or training a model on data drawn from one dictionary and evaluating on data from a different dictionary are all hindered by the fact that these resources represent different dialects, which are transcribed at different levels of detail. For example, ⟨audible⟩ is transcribed in CUVOALD as /ɔdəbl/, but in the NETtalk data set as /ɔdəbl̩/ with a syllabic /l̩/ – a contrast that is absent from the transcription system used by CUVOALD. The distinction between plain /l/ and syllabic /l̩/ is made in HML, which on average appears to include more phonetic details than the other dictionaries. For instance, ⟨audible⟩ is transcribed as /ɔdɨbl̩/ in HML, which distinguishes between a near-close near-front unrounded vowel /ɪ/ and a close central unrounded vowel /ɨ/. This distinction is not made by CUVOALD, the NETtalk data set, or CMUdict, where ⟨audible⟩ is transcribed as /ɑdəbəl/. However, while HML's /ɨ/ often corresponds to /ɪ/ or

/ə/ in the other dictionaries, the correspondences are idiosyncratic. For example, when the past tense morpheme ⟨ed⟩ is transcribed as /ɨd/ in HML, it corresponds to /əd/ in the NETtalk data, but on the other hand ⟨liquid⟩ is /lɪkwɨd/ in HML and /lɪkwɪd/ in the NETtalk data set. Damper et al. [1999, sec. 5.2 and appendix B] discuss post hoc 'harmonization' of different phoneme inventories in the context of an empirical comparison.

It is therefore generally best to use a single lexical resource for training and evaluation. Among the available resources, we primarily use the NETtalk data and CMUdict. The NETtalk data set, which will be described in Section 2.2.2, is unique in that it contains alignment information. CMUdict is the largest American English dictionary in the above list and is available without any practical restrictions ("prescriptions").

2.2.1 The CMU Pronouncing Dictionary

The Carnegie Mellon Pronouncing Dictionary [Weide, 1998], henceforth CMUdict, is a large data set of American English pronunciations. An entry in CMUdict consists of a letter string and a sequence of phoneme-plus-stress symbols. For example:

```
SLAUGHTERHOUSE  S L AO1 T ER0 HH AW2 S
```

The phonetic transcriptions include lexical stress information on vowel symbols: in this example, `AO1` carries primary stress, `AW2` has secondary stress, and `ER0` is unstressed. Predicting stress patterns is beyond the scope of the present work, and so we will ignore most stress information. For greater clarity we want to avoid the use of idiosyncratic transcription systems as much as possible, and prefer the International Phonetic Alphabet (IPA) [International Phonetic Association, 1999].

aa	/ɑ/	k	/k/
ae	/æ/	l	/l/
ah0	/ə/	m	/m/
ah1, ah2	/ʌ/	n	/n/
ao	/ɔ/	ng	/ŋ/
aw	/a͡ʊ/	ow	/o/
ay	/a͡ɪ/	oy	/ɔ͡ɪ/
b	/b/	p	/p/
ch	/ʧ/	r	/ɹ/
d	/d/	s	/s/
dh	/ð/	sh	/ʃ/
eh	/ɛ/	t	/t/
er	/ɚ/	th	/θ/
ey	/e/	uh	/ʊ/
f	/f/	uw	/u/
g	/g/	v	/v/
hh	/h/	w	/w/
ih	/ɪ/	y	/j/
iy	/i/	z	/z/
jh	/ʤ/	zh	/ʒ/

Figure 2.1: CMUdict transcription symbols and IPA correspondences.

The pronunciation of ⟨slaughterhouse⟩ recorded in CMUdict can be rendered in IPA as /slɔtɚha͡ʊs/. A full list of IPA equivalents of the transcription symbols used by CMUdict appears in Figure 2.1. We have taken the liberty to use lowercase letters and stripped off most indicators of lexical stress, with the exception of ah, which is rendered in IPA as /ə/ when unstressed and as /ʌ/ when stressed. The symbols listed in Figure 2.1 are treated as units: for diphthongs like ay /a͡ɪ/ this is indicated by a tie bar, and the indivisible nature of the affricates ch /ʧ/ and jh /ʤ/ is represented typographically by the use of ligatures.

The dictionary may contain multiple entries with a common orthographic string. For example, there are three entries for the letter string ⟨minute⟩:

```
MINUTE  M IH1 N AH0 T
MINUTE(2)  M AY0 N UW1 T
MINUTE(3)  M AY0 N Y UW1 T
```

Parenthesized numbers indicate the presence of multiple entries. No rationale for the presence of multiple entries is given, the order in which multiple entries appear seems arbitrary, and their status is never clarified. For example, the first entry for ⟨minute⟩ lists the pronunciation /mɪnət/, so we know it to be the entry for the noun *minute*, whereas the second and third entry correspond to an adjectival usage. These last two entries are minor pronunciation variants and have the same meaning and part of speech, which is distinct from the nominal sense of *minute*. However, the nature of these distinctions is not made explicit in the dictionary. When using CMUdict for development or evaluation of a letter-to-sound component, we may have to treat such words specially.

Simply selecting the first of multiple entries for a word will not do, due to inconsistencies in the numbering of entries. For example, the morpheme ⟨day⟩ in the names of the days of the week is either pronounced /de/ (as in the isolated word ⟨day⟩, only unstressed) or /di/ (so that ⟨Thursday⟩ sounds almost like ⟨thirsty⟩). This variation is reflected in CMUdict, but not consistently. If the first entry for each day of the week is selected, a week sounds like this: /mʌndi/, /tuzdi/, /wɛnzdi/, /θɚzde/, /fɹa͡ɪdi/, /sætɚdi/, /sʌnde/. For no apparent reason, the pronunciations of ⟨Thursday⟩ and ⟨Sunday⟩ are treated as exceptional.

CMUdict also contains a fair number of abbreviations, some Internet domain names, and other "nonstandard words" [Sproat et al., 2001]. For example, ⟨mph⟩

has two entries, one with pronunciation /ɛmpiɛʧ/ (spelling it), and one with pronunciation /maɪlzpɚaʊɚ/ (expanding it into ⟨miles per hour⟩). Neither pronunciation is particularly relevant for the problem of learning letter/phoneme correspondences: pronouncing a word by concatenating the names of its letters, i. e., spelling it, is easy and does not require automatic learning methods; on the other hand, expanding abbreviations into the words they abbreviate is hard without any knowledge of likely expansions, so the practical solution is often to put abbreviations in an exception dictionary. In other words, the letter string ⟨mph⟩ is only interesting for the learning task when it represents the interjection /m̩f/, which is not included in CMUdict. A good indicator that an entry might be an abbreviation is the difference in length between its letter string and phoneme string: if the phoneme string is longer, there is a good chance that it is an abbreviation, though this also catches ordinary words like ⟨cubism⟩ /kjubɪzəm/, and erroneous entries like ⟨disparages⟩ /dɪspɛɹɪʤəsɹʌptɪv/.

2.2.2 The NETtalk Data Set

The NETtalk data set [Sejnowski, 1988] is much smaller than CMUdict, but by no means a subset, as more than one fifth of its entries are not found in CMUdict. An entry consists of a letter string, transcription string, a stress and syllabification pattern, and an indicator of foreign or "irregular" word forms. The dictionary is aligned because in each entry the three strings (letters, transcription, stress) have the same length. For example:

```
fuchsia  fYS--x-  >1>>>0<  1
```

This is the entry for ⟨fuchsia⟩. The number `1` in the fourth column indicates that this is not a foreign word (which is, strictly speaking, wrong), but has an "irregular" pronunciation, which is given as `fYS--x-`. We again prefer the use of the standard IPA [International Phonetic Association, 1999], and this allows us to represent the pronunciation of ⟨fuchsia⟩ as /fjuʃə/. The IPA correspondences of the NETtalk transcription symbols are listed in Figure 2.2. This inventory list differs from all others in the surveyed literature – for example Stanfill and Waltz, 1986, p. 1226; Bakiri and Dietterich, 2001, p. 29; and Sejnowski, 1988, the primary description of the NETtalk data set – in that it is correct and minimally complete. Note that the NETtalk transcription system uses uppercase and lowercase letters, as well as non-alphabetic symbols. As before, phonemic units are indicated typographically: diphthongs like `A` /a͡ɪ/ have a tie bar, and the affricates `C` /t͡ʃ/ and `J` /d͡ʒ/ are represented by ligatures. Note especially that a few NETtalk symbols denote two phonemes, as indicated by an absence of tie bars or ligatures. They are: `K` /kʃ/, `X` /ks/, `Y` /ju/, `!` /ts/, `#` /gz/, and `+` /wɑ/. The special symbol `-` denotes zero phonemes, i. e., it is silent. When transliterating NETtalk transcriptions into IPA we may occasionally want to preserve the alignment information; we then use the symbol /-/ even in IPA transcriptions.

The NETtalk transcription system contains more than just phonemic information, since the dashes are meaningful, although they are phonetically empty. We can see this more easily if we line up the letters, transcription symbols, and the stress/syllable pattern vertically, like this:

```
fuchsia
fYS--x-
>1>>>0<
```

a	/ɑ/	C	/ʧ/
b	/b/	D	/ð/
c	/ɔ/	E	/ɛ/
d	/d/	G	/ŋ/
e	/e/	I	/ɪ/
f	/f/	J	/ʤ/
g	/g/	L	/l̩/
h	/h/	M	/m̩/
i	/i/	N	/n̩/
k	/k/	O	/ɔɪ/
l	/l/	R	/ɚ/
m	/m/	S	/ʃ/
n	/n/	T	/θ/
o	/o/	U	/ʊ/
p	/p/	W	/a͡ʊ/
r	/ɹ/	Z	/ʒ/
s	/s/	@	/æ/
t	/t/	*	/ʍ/, /ʌ/
u	/u/	^	/ʌ/
v	/v/	K	/kʃ/
w	/w/	X	/ks/
x	/ə/	Y	/ju/
y	/j/	!	/ts/
z	/z/	#	/gz/
A	/a͡ɪ/	+	/wɑ/

Figure 2.2: NETtalk transcription symbols and IPA correspondences.

This makes it visually clear that the letter ⟨f⟩ is aligned with the transcription symbol `f` (and hence the phoneme /f/) and occurs in the onset (denoted by >) of a syllable; the letter ⟨u⟩ is aligned with the transcription symbol `Y` (and hence the two phonemes /ju/) and is stressed; the letter ⟨c⟩ is aligned with the symbol `S` (and hence the phoneme /ʃ/) and is part of the onset of the second syllable; the letter ⟨h⟩ is aligned with the transcription symbol `-` and is therefore silent, but nevertheless part of the (astoundingly big) onset of the second syllable; and so forth until ⟨a⟩, which is silent and part of the coda (denoted by <) of the last syllable.

We do not question the wisdom of this alignment, and simply note that the representation of stress/syllable information depends too much on the letters. Say the alignment was slightly different (and arguably more reasonable), like this:

```
fuchsia
fY--S-x
>1<<>>0
```

Then the syllabic status of the letter ⟨h⟩, for example, would appear to be different. However, the issue of whether it is part of the coda of the first syllable or the onset of the second syllable seems irrelevant, given that it is silent in both cases. It is worth noting that predicting stress/syllable patterns seems to be a harder task than predicting phonemes, as evidenced by the generally lower performance scores [see for example Bakiri and Dietterich, 2001]. Church [1985] goes even further when he states that it 'is impossible to determine the stress of a word by looking through a five or six character window', which is what many traditional approaches (Section 3.3) do. On the other hand, Bakiri and Dietterich

[2001] use a window that is three times larger than that, which improves the quality of stress assignment considerably. It also makes it possible to treat the task of predicting stress labels as fundamentally the same as the task of predicting transcription symbols, regardless of whether stress is predicted separately or as a combination of transcription-plus-stress symbols [van den Bosch, 1997]. In that sense the approach taken by Bakiri and Dietterich [2001] does not add any new kinds of challenges beyond the basic task of predicting transcriptions, but has to make a number adjustments in order to cope with NETtalk's arguably unreasonable representation of stress and syllabification. Since we are primarily interested in general techniques for learning string-to-string mappings, for which letter-to-sound rules are already a good test case, stress assignment is beyond the scope of the present endeavor. Consequently we will omit all stress and syllabification information from here on forward.

The notion of alignment implicit in the NETtalk data is somewhat special: only the phonemic representations are padded with special symbols to make them the same length as the corresponding letter strings. Letter strings are not padded. If a letter corresponds to more than one phoneme, one of the special multi-phoneme symbols is used. This way, a letter-to-sound component can scan its input letter by letter (possibly including a certain amount of context) and predict transcription symbol for each letter position. If the letter strings were padded with special symbols, resulting, for example, in

```
f-uchsia
fyu--S-x
```

then the letter-to-sound component would have to treat insertion points specially, e. g. by predicting their occurrence, or by assuming that they occur regularly

throughout the letter string [Jansche, 2001]. Riley [1991] discusses several options for dealing with insertions in the context of phoneme-to-phone conversion, or pronunciation modeling.

The presence of multi-phoneme symbols begs the question of how they ended up where they are. On what basis would one decide when to use a multi-phoneme symbol instead of several single-phoneme symbols? The NETtalk data set contains minimal pairs like the following:

```
inflection    inflexion
InflEkS-xn    InflEK-xn
```

By consulting Figure 2.2 we can see that the symbols `kS` denote the same phoneme string as the symbol `K`, and the two entries are therefore homophonous. Using the multi-phoneme symbol `K` in the transcription of ⟨inflexion⟩ seems to be motivated exclusively by considerations of systematicity, a desire to pick a transcription symbol corresponding to the letter ⟨x⟩ analogous to other entries.

The desire to impose systematicity in the transcriptions and alignments goes so far that incorrect or impossible transcription strings are occasionally included. In his description of the NETtalk data set Sejnowski [1988] explicitly mentions the case of ⟨eighth⟩, which is transcribed as if it were pronounced /eθ/ instead of /etθ/. Another systematic anomaly, one which he fails to mention, concerns the morpheme ⟨one⟩, occurring on its own or in compounds like ⟨oneself⟩. One might think that ⟨one⟩ is homophonous with ⟨won⟩, but not so in the NETtalk data set, where the situation is like this:

```
won    one
w^n    wn-
```

Apparently the experts who created that data set felt that it was important that the letter ⟨n⟩ in ⟨one⟩ was aligned with the phoneme /n/, even when that meant either introducing another two-phoneme symbol to represent /wʌ/ (which was not done), or transcribing ⟨one⟩ incorrectly as /wn/.

Constructing an aligned dictionary like the NETtalk data set from an ordinary unaligned pronunciation dictionary like CMUdict is therefore not simply a matter of aligning the phonemes with respect to the letters of each word, but must also take into account the conversion of two or more phonemes into special multi-phoneme symbols. All of this must be carried out in a uniform and systematic way, and seems to have occasionally gone wrong in the NETtalk data set. Inconsistencies can be illustrated using such minimal pairs as

```
guy     buy
g-A     bA-
```

where the letter substring ⟨uy⟩ is uniformly pronounced /a͡ɪ/, yet the alignments differ. A similar example is

```
tong    tongue
taG-    t^-G--
```

in which ⟨ng⟩ is pronounced /ŋ/, despite differences in the alignments. We will see yet another example later on in Figure 2.5. One could justifiably count as a mistake any situation in which fragments of words share the same letter sequence and the same phoneme sequence (ignoring padding symbols and mapping multi-phoneme symbols to phonemes) but exhibit differences in alignment. Such alignment inconsistencies could be detected using a variant of the technique described by Dickinson and Meurers [2003].

It is not clear what standard should be used in producing alignments, perhaps because there is no objective measure of the goodness of an alignment, other than the internal consistency of an aligned dictionary. For example, one principle behind the NETtalk alignments seems to be to align /ɚ/ with an ⟨r⟩ rather than a "vowel letter" in words like ⟨liar⟩ /laɪ-ɚ/, ⟨diner⟩ /daɪn-ɚ/, or ⟨color⟩ /kʌl-ɚ/. Another principle seems to require aligning a phoneme that corresponds to a longer sequence of letters with the first letter in that group. This seems to be behind the alignment of ⟨colonel⟩ /kɚ--n-ɫ/, where /ɚ/ corresponds to the first occurrence of ⟨o⟩. Not only is this a rare alignment – in fact it occurs precisely once – more importantly, other than a general tie breaking principle there seems to be no way to decide which alignment is "right", or even whether ⟨colonel⟩ /k-ɚ-n-ɫ/ would have been "better".

While an aligned dictionary contains potentially useful information, the alignments themselves may become sources of uncertainty and inconsistency. When using the NETtalk dictionary for development or evaluation of a letter-to-sound component, we may have to take this into account as another potential source of errors.

2.3 Evaluation Metrics

Evaluation metrics are used to compare predicted pronunciations against reference pronunciations, which usually come from a held-out evaluation data set. In this section we will define several evaluation metrics, conceptualized as loss functions, that have been used in the literature. A comparison of these metrics will follow in Section 2.4.

The only formal requirement placed on an evaluation metric is that it maps from pairs of objects under comparison to the nonnegative real numbers. An evaluation metric need not be a metric in the technical sense of satisfying the triangle inequality and symmetry, and taking on the value zero iff its arguments are identical, though many evaluation metrics do in fact have some or all of these properties.

Evaluation metrics obviously play an important role in evaluating and comparing the letter-to-sound components of a TTS system. As noted above, such evaluations often take on the form of unit testing, for practical reasons. Ideally, for an evaluation metric used during unit testing to be considered effective, it should be able to accurately predict an upper bound on the error introduced by the letter-to-sound component in an end-to-end evaluation of an entire TTS system. In reality, the extent of such a correlation has not been investigated for the various evaluation metrics that have been proposed in the literature. Since we are not concerned with end-to-end evaluation, we will not investigate this issue further.

The main evaluation metric used for evaluation should ideally be optimized during development. In other words, it should contribute to the objective function of any optimizations carried out during development. Typically this means minimizing the expected loss (risk), or an upper bound or approximation of it (such as the empirical risk). The loss functions discussed below differ greatly in terms of the optimization problems they give rise to.

The principal definition of loss is in terms of errors. Errors can be made at different levels, as will be discussed in the rest of this section. In the speech recognition literature, a certain terminology is already established, but it does not carry over easily to the letter-to-sound setting without the potential for confusion. In

	Speech recognition	*Letter-to-sound*	*Generic term*
Basic unit	word	phon(em)e	symbol
Sequence	sentence	word	string
Meta-Sequence	text	sentence	

Figure 2.3: Terminology used in conjunction with evaluation metrics.

many speech recognition applications the basic units are words, and performance is usually reported in terms of word error rate. The next larger level at which errors are tallied in speech recognition is the sentence or utterance, i. e., a sequence of words. For letter-to-sound applications, the basic units are phonemes and the next larger units are words, viewed as sequences of phonemes. Unless we make it clear which application domain we are talking about, the term *word error rate* is somewhat confusing, since it may refer to basic units in speech recognition and language modeling, or to larger units in letter-to-sound conversion. To avoid confusion we use the term *symbol* (and *symbol error rate* etc.) when referring to basic units, and *string* when referring to a sequence of basic units. A summary of this terminology is displayed in Figure 2.3.

2.3.1 String Error

The recommendation of Damper et al. [1999] seems attractive at first, because of its simplicity: they use a 0-1-loss function that counts the pronunciation of an entire word as correct (zero loss) just in case it matches the reference pronunciation perfectly. This measure, which we refer to generically as *string error* is simple, rigorous, and conservative, since it arguably underestimates the perceived quality of

a TTS system, which may tolerate a certain amount of errors. On the other hand, optimization under 0-1-loss usually creates integer programming problems that we cannot even expect to approximate efficiently, let alone solve exactly; further details can be found in Section 3.5 in the next chapter.

A related high-level 0-1-loss function is *sentence error*, as included in the results listed by Yvon et al. [1998], which counts an entire sentence as correct just in case its pronunciation was predicted correctly. This is an informative evaluation measure especially for languages with non-trivial phonemic effects that cross word boundaries (e. g. *liaison* in French). Despite their similar nature, there are several important differences between string (word) error and sentence error:

- On the one hand sentence error appears to be an extremely conservative measure, since a single pronunciation error can render an otherwise impeccable sentence wrong. On the other hand, a system could theoretically render a small number of sentences utterly incomprehensible and receive a relatively high sentence error score. Word error has similar shortcomings (more on this below), but they are arguably less severe.

- Word error can be used as an optimization objective (see Section 3.5), although this seems only feasible for very simple models. Optimizing sentence error is quantitatively a much harder problem, not only because the average sentence length measured in letters or phonemes is much higher than the average word length, but also because simple models are no longer applicable.

  A simple deterministic model might take little or no context into account for computing a letter-to-phoneme mapping. This is only a minor problem

within a word, since it is quite likely that a given letter (plus context) appears at most once within a word. If a letter-in-context appears more than once there is a possibility that the word is exceptional in the sense that there is no way to assign the same pronunciation to all occurrences of that letter while correctly predicting the pronunciation of the word. For the very simple deterministic model whose training is discussed in Section 6.3, roughly one third of the (aligned) training data are not utilized because they contain multiple occurrences of the same letter with different pronunciations. (On the other hand, using just a single letter of context would virtually eliminate such exceptional words.) The same model cannot be applied to sentences instead of words, since the chances of finding the same letter with different pronunciation more than once are much higher, which would mean that most of the training data would be useless, since a single guaranteed mistake in a sentence means that it need not be taken into account when optimizing sentence error.

- What sets word error apart from all other evaluation criteria discussed here is the fact that one can calculate the expected number of previously unseen word types in a given amount of word tokens (for example, how many previously unseen words we expect to find in the next million words of newspaper text, after having encountered a few million words already). This number is nonzero even if the number of previously seen words is large [Baayen, 2001]. By contrast, the orthographic and phonemic inventories are always fixed and assumed to be known. If a letter-to-sound component is only used for out-of-dictionary words, one can then calculate the expected

coverage for various dictionary sizes [Damper et al., 1999]. It is not clear how such a calculation could be carried out for sentences instead of words.

We use the term *string error* to refer to the absolute number of words in an evaluation data set whose predicted pronunciations differ from their reference pronunciations. When we speak of *string error rate* we mean the relative fraction of words in an evaluation data set whose predicted pronunciations deviated from the reference standard.

2.3.2 Prediction Error

Within the long tradition of approaches that rely on aligned data, perhaps the most prominent example is the NETtalk system [Sejnowski and Rosenberg, 1987] and its aligned data set [Sejnowski, 1988; Blake and Merz, 1998] discussed in Section 2.2.2.

The advantage of working with aligned data is that a letter sequence and corresponding transcription sequence are of the same length. Learning same-length transducers is conceptually not very different from learning acceptors. Usually, the problem is restricted much further, so that the letter-to-sound conversion problem can be formulated in terms of a simpler classification problem. For each letter in a word there is a corresponding symbol in the aligned phonemic transcription, and since the inventory of transcription symbols is finite, and in fact very small, one can perform letter-to-sound conversion by applying a classifier to each letter position and concatenating the individual predictions. Dietterich [2002] gives an excellent overview of this paradigm and related approaches, and we discuss its application to letter-to-sound transductions further in Section 3.3.

Among the features one can use for predictions are such diverse information sources as the letter context, the one-sided phoneme context, as well as global properties of the word such as its contextual part of speech or its linguistic origin. It has been shown empirically [Lucassen and Mercer, 1984] that non-local context does not carry much information about the pronunciation of a given letter. Many approaches, including NETtalk, rely exclusively on features of the local letter context for prediction. As discussed in Section 3.4, such approaches essentially boil down to learning alphabetic substitutions.

For any approach that reduces the letter-to-sound conversion task to separate tasks of classifying each letter (plus context), a natural evaluation metric is classification error or *prediction error* . This is simply the number of mistakes the underlying classifier makes when predicting transcription symbols. Since the predicted transcription string and the reference transcription string are of equal length, prediction error is the same as the Hamming distance between the two transcription strings. The *prediction error rate* is just the empirical error of the underlying classifier on the evaluation data set.

The performance figures reported by most approaches that require aligned training data – this includes NETtalk [Sejnowski and Rosenberg, 1987] and other approaches trained primarily on the NETtalk data [for example Stanfill and Waltz, 1986; Stanfill, 1987; Bakiri and Dietterich, 1993, 2001], but applies also more generally to many approaches based on simple classifier learning [for example van den Bosch and Daelemans, 1993] – are usually the prediction error rate (or prediction accuracy) of the underlying classifiers. Note that figures for prediction error (rate) are crucially dependent on the inventory of transcription symbols used by the evaluation data set, since the underlying classifiers predict transcription symbols, not phonemes.

2.3.3 Symbol Error

Another measure of performance that is sometimes used is based on the string edit distance [Wagner and Fischer, 1974; Kruskal, 1983] between the predicted pronunciation and the reference pronunciation of a word. Because symbol error is defined in terms of the predicted phoneme string, which does not use padding symbols and in which all multi-phoneme transcription symbols have been mapped to the phonemes they represent, it is conceptually similar to string error, and unlike prediction error, which is defined in terms of transcription symbols and alignments. The effects of this conceptual difference are discussed in Section 2.4.1.

In any concrete evaluation, several details need to be spelled out, including the allowable edit operations and their costs used for computing the string edit distance, as well as what kind of normalization was used in order to obtain a relative error rate from raw edit distance.

In the simplest case, we can use the Levenshtein distance (see page 121), which has uniform costs for insertions, deletions and substitutions. The *symbol error* of a word is then the minimal number of insertions, deletions and substitutions required to transform the predicted symbol (phoneme) string into the reference symbol string.

A relative measure of error is less easily defined, since it is not clear what we should divide the absolute edit distance by. In the Automatic Speech Recognition (ASR) literature, the analogous notion of word error rate is obtained by dividing (normalizing) the edit distance by the number of words in the reference transcription. We will speak generically of *symbol error rate* defined as raw edit distance divided by the length of the reference string. Per our terminological remarks in

conjunction with Figure 2.3, this is meant to encompass word error rate of ASR as well as what we may call *phoneme error rate* of letter-to-sound modules. It is essentially the same as what Fisher [1999] calls *phone error rate*, but others [for example Luk and Damper, 1996] also speak of it more neutrally in terms of symbols.

A desirable property of this definition of symbol error rate is that it is proportional to the absolute edit distance (raw symbol error), since the normalizing constant depends only on the fixed reference string. A minor drawback is the somewhat unintuitive result that the error rate can exceed 1.0 when the predicted string is longer than the reference string. An example of this appears in Figure 2.14. Symbol error (rate) can be extended to sets of string pairs by adding the individual edit distances for each individual string (and dividing by the total length of all reference strings).

Crucially, string edit distance need not be parameterized using costs which are uniformly either zero or one. In fact, there can be separate costs for each basic edit operation. For example, substituting the phoneme /æ/ for the phoneme /ɛ/ could have much lower cost than replacing it with /ʧ/. The loss $L(x, x')$ incurred by predicting a certain pronunciation x when the reference pronunciation is x' need not be symmetric, i. e., generally $L(x, x') \neq L(x', x)$. For example, omitting a final /t/, especially after another stop consonant as in ⟨strict⟩ /stɹɪkt/, may generate a small loss, but adding a final /t/ where the reference standard does not have one would be worse, since that would be like saying ⟨start⟩ instead of ⟨star⟩.

While weighted edit distance can easily accommodate more phonetically realistic versions of symbol error, the necessary empirical work in this area has not been done. The phonetics literature contains confusion studies that assess the similarity of consonants or of vowels, but in the most general case the parameters

for weighted edit distance have to provide costs for all possible combinations, including comparisons between vowels and consonants, plus provide insertion and deletion costs, for which empirical data are very hard to find.

Perhaps a better approach would be to view the loss $L(x, x')$ as a word confusion probability, which in the simplest case can be computed exactly like string edit distance (this will be the topic of Section 4.2). Moreover, the parameters of such a probability distribution could be estimated from a list of confusable phonemic forms (see Section 4.3). This would potentially simplify experiments, since the auditory confusability of entire words could be tested, instead of testing the confusability of specific phones, which requires controlling for context. A loss function based on probabilities also has some technical advantages over weighted string edit distance that will be discussed in Section 5.5. However, there does not appear to be much empirical work that could be used to derive a loss function in terms of confusion probabilities.

In the rest of this chapter and throughout Chapter 3 we therefore assume that symbol error is defined as ordinary Levenshtein distance, which simplifies the presentation of examples in the next section. It should be clear, though, that for letter-to-sound applications a more realistic assignment of edit costs based on some notion of phonetic similarity should be preferred.

2.4 Interconnections

As there are major conceptual differences between the loss functions introduced in the previous sections, one should suspect that minimizing any one loss function will generally not result in minimization of the other metrics. We illustrate this point for pairs of the major loss functions discussed previously.

Orthography	s	l	a	u	g	h	t	e	r	h	o	u	s	e	*PrER*	*SymER*	*StrER*
Reference alignment	s	l	ɔ	-	-	-	t	-	ɚ	h	a͡ʊ	-	s	-			
Classifier #1 predictions	s	l	-	ɔ	-	-	t	ɚ	-	h	-	a͡ʊ	s	-	6/14	0/8	0/1
Classifier #2 predictions	-	-	-	s	l	ɔ	-	t	-	-	ɚ	h	a͡ʊ	s	14/14	0/8	0/1

Figure 2.4: Examples of predicted pronunciations, contrasting prediction error rate and symbol error rate.

2.4.1 Prediction Error vs. Symbol Error

As mentioned before, prediction error rate is only meaningful in approaches that use pre-aligned evaluation data. Symbol error rate – in our case, phoneme error rate – on the other hand is based on the minimal number of edit operations (insertions, deletions and substitutions) necessary to transform the predicted string into the reference string, ignoring the reference alignment. An optimal sequence of edit operations defines an alignment between the predicted string of phonemes (and hence the string of letters used for prediction) and the reference string, and this alignment may differ from the reference alignment.

It is easy to see that one can keep symbol error low and make the prediction error arbitrary large in theory. Consider the word ⟨slaughterhouse⟩ and its reference pronunciation /slɔtɚha͡ʊs/. A reference alignment, obtained by concatenating the alignments for ⟨slaughter⟩ and ⟨house⟩ from the NETtalk data set, is shown in Figure 2.4.

Now suppose that a classifier, call it Classifier #1, has been learned on the basis of an aligned data set and that it makes reasonable predictions on input ⟨slaughterhouse⟩ shown on the third line of Figure 2.4. The prediction error rate of

Classifier #1 is $6/14 \approx 0.43$, whereas the phoneme error rate is $0/8 = 0$. The point is that the classifier implicitly produces an alignment differing from the reference standard and is penalized for that, even though the differences do not matter in terms of phoneme error or word error.

A worse case arises for another hypothetical classifier, call it Classifier #2, which predicts the wildly unreasonable, though theoretically possible, results on the last line of Figure 2.4. Every single prediction of Classifier #2 turns out to be wrong compared with the reference alignment. Its prediction error rate is therefore $14/14 = 1$, while the phoneme error rate is still 0, the same as for Classifier #1.

These examples illustrate that prediction error rate can distinguish between different classification results that are indistinguishable on the basis of symbol error rate. It is not clear, however, that the power to make such distinctions is necessary or desirable. There don't seem to be many practical applications that would make it necessary to know the exact alignment that gave rise to a particular output string; letter-to-phoneme conversion is arguably not one of them. In other words, using prediction error rate may result in spurious distinctions, which, combined with the fact that it only applies in settings involving aligned data, makes this a performance measure of very limited utility.

We now exhibit two exemplary situations where minimizing symbol error rate does not necessarily minimize prediction error rate, and conversely.

The first situation arises when ties among fairly arbitrary alignment decisions are broken non-uniformly. We again use the aligned NETtalk data set [Sejnowski, 1988] as a source of examples. Consider the words ⟨neural⟩ and ⟨rheumatic⟩. In both instances the letters ⟨eu⟩ are pronounced /ʊ/, but the alignments differ; see the reference alignments displayed in Figure 2.5.

	Reference	*Predicted*	*PrE*	*SymE*
⟨amateur⟩	/æmət--ɚ/	/æmæt--ɹ/	2	2
⟨neural⟩	/nʊ-ɹ-ˌl/	/n--ɹæˌl/	2	2
⟨rheumatic⟩	/ɹ--ʊmætɪk/	/ɹ---mætɪk/	1	1
		Totals	5	5

Figure 2.5: Example data set with inconsistent alignments. The predicted pronunciations were produced by majority classifiers and minimize prediction error.

Assume a classification setting in which phonemes, including the pseudo-null phoneme /-/, are predicted for each letter position in a word without using any context, i. e., as a function of individual letters only. The learning task can be decomposed into inference of up to 26 independent classifiers, one for each letter of the alphabet. Since no other information is available, for each of these independent classifiers the optimal choice that minimizes classification error is a simple majority classifier that predicts the phoneme most frequently associated with the given letter. In the case of Figure 2.5 the choices are unique, as can be seen from Figure 2.6. We can obtain the total prediction error from the same figure by adding up all occurrence counts of the minority phonemes. The predictions of the majority classifiers are displayed in Figure 2.5, together with the number of prediction errors (column *PrE*) and phoneme/symbol errors (column *SymE*). Overall prediction error rate is $5/22 \approx 0.23$, and symbol error rate is $5/16 \approx 0.31$.

This symbol error rate is not optimal for the present classification setting, although the prediction error rate is in fact optimal. Because the choice of alignment

Letter	*Phonemes and occurrence counts*
⟨a⟩	/æ/ 2, /ə/ 1, /-/ 1
⟨c⟩	/k/ 1
⟨e⟩	/-/ 2, /u/ 1
⟨h⟩	/-/ 1
⟨i⟩	/ɪ/ 1
⟨l⟩	/l̩/ 1
⟨m⟩	/m/ 2
⟨n⟩	/n/ 1
⟨r⟩	/ɹ/ 2, /ɚ/ 1
⟨t⟩	/t/ 2
⟨u⟩	/-/ 2, /u/ 1

Figure 2.6: Letters and corresponding phonemes among the data in Figure 2.5.

of /ʊ/ with one of the letters ⟨eu⟩ is different for ⟨neural⟩ and ⟨rheumatic⟩, the phoneme most frequently associated with the letter ⟨e⟩ is /-/. If the reference alignment had been more consistent, we might have found that the phoneme /ʊ/ was the best prediction for the letter ⟨e⟩. The consequences of predicting /ʊ/ instead of /-/ for the letter ⟨e⟩ are shown in Figure 2.7. This slightly modified mapping is one that minimizes symbol error (and, incidentally, also string error), which is less sensitive to spurious alignment distinctions than prediction error. Compared with Figure 2.5, the prediction error rate of $6/22 \approx 0.27$ is higher, but the symbol error rate of $4/16 = 0.25$ is lower, and in fact optimal.

The second example of differing results for minimizing prediction error vs. symbol error involves the special multi-phoneme symbols in the NETtalk data set. The use of multi-phoneme symbols is not limited to the NETtalk data set; other

	Reference	*Predicted*	*PrE*	*SymE*
⟨amateur⟩	/æmət--ɚ/	/æmætʊ-ɹ/	3	3
⟨neural⟩	/nʊ-ɹ-ļ/	/nʊ-ɹæļ/	1	1
⟨rheumatic⟩	/ɹ--ʊmætɪk/	/ɹ-ʊ-mætɪk/	2	0
		Totals	6	4

Figure 2.7: Example data set with non-uniformly broken ties. The predicted pronunciations minimize symbol error.

approaches also use a fixed, language-specific inventory of multi-phoneme symbols [Minker, 1996], or generate multi-phoneme symbols automatically as needed [Sproat, 2000]. Recall that in the NETtalk transcription system the symbol X represents the phoneme sequence /ks/, and the symbols K stands for the phoneme sequence /kʃ/. Consider the set of words plus reference alignments shown in Figure 2.8. For each word, the reference transcription and predicted transcription are shown. Since prediction error (*PrE*) is defined in terms of transcription symbols, it appears on the same line as the transcription string. Underneath each transcription string is the corresponding string of phonemes, on the same line as the values for symbol error (*SymE*) and string error (*StrE*), which are defined in terms of phonemes rather than transcription symbols.

Assume the same classification scenario as before: for each letter predict a phoneme (possibly null) without reference to any features other than the identity of the letter. Figure 2.9 shows the phoneme symbols (using NETtalk transcription) associated with each letter, including occurrence counts. In order to minimize prediction error, we have to choose for each letter the symbol most frequently associated with it. The choices are unique. The predictions under the majority

	Reference	*Predicted*	*PrE*	*SymE*	*StrE*
⟨flexure⟩	`flEK-R-`	`fl-z-r-`	3		
	/flɛkʃɚ/	/flzɹ/		4	1
⟨inflexion⟩	`InflEK-xn`	`Infl-zIxn`	3		
	/ɪnflɛkʃən/	/ɪnflzɪən/		3	1
⟨lynx⟩	`lIGX`	`lAnz`	3		
	/lɪŋks/	/laɪ̯nz/		4	1
⟨prefix⟩	`prifIX`	`fr-fIz`	3		
	/pɹifɪks/	/fɹfɪz/		4	1
⟨xenophobe⟩	`zEnxf-ob-`	`z-nxf-xb-`	2		
	/zɛnəfob/	/znəfəb/		2	1
⟨xerophyte⟩	`zIrxf-At-`	`z-rxf-At-`	1		
	/zɪɹəfaɪ̯t/	/zɹəfaɪ̯t/		1	1
⟨xylophone⟩	`zAlxf-on-`	`zAlxf-xn-`	1		
	/zaɪ̯ləfon/	/zaɪ̯ləfən/		1	1
		Totals	16	19	7

Figure 2.8: Example data set with multi-phoneme symbols. The predicted pronunciations were produced by majority classifiers and minimize prediction error.

classifiers are shown in Figure 2.8. The column labeled *PrE* shows the absolute number of prediction errors, column *SymE* shows symbol errors, and column *StrE* string errors. The overall prediction error rate is $16/53 \approx 0.30$, the symbol (phoneme) error rate is $19/48 \approx 0.40$, and the string (word) error rate is $7/7 = 1$.

Let's take another look at the phoneme symbols corresponding to the letter ⟨x⟩ shown in Figure 2.9. The symbol z, which stands for the phoneme /z/, is the symbol most frequently associated with the letter ⟨x⟩ in this biased sample, but the runners-up `K` and `X` are almost as frequent. Crucially, if we look at the phoneme sequences that `K` and `X` represent, namely /kʃ/ and /ks/, we see that the phoneme /k/ occurs a total of four times in the pronunciation associated with ⟨x⟩.

Letter	*Phonemes and occurrence counts*
⟨b⟩	b 1
⟨e⟩	- 4, E 3, i 1, I 1
⟨f⟩	f 3
⟨h⟩	- 3
⟨i⟩	I 2, - 1
⟨l⟩	l 4
⟨n⟩	n 4, G 1
⟨o⟩	x 4, o 2
⟨p⟩	f 3, p 1
⟨r⟩	r 2, R 1
⟨t⟩	t 1
⟨u⟩	- 1
⟨x⟩	z 3, K 2, X 2
⟨y⟩	A 2, I 1

Figure 2.9: Letters and corresponding phonemes among the data in Figure 2.8.

If phoneme error is measured as Levenshtein distance with uniform costs for all insertions, deletions and substitutions, we must choose /k/ as the pronunciation of ⟨x⟩ in order to minimize symbol (phoneme) error.

The consequences of this change appear in Figure 2.10. The only difference compared with Figure 2.8 is that the prediction for ⟨x⟩ is now k. This increases the absolute number of prediction errors by three, since now all occurrences of the letter ⟨x⟩ are sources of errors, whereas previously the symbol z was correctly predicted in three cases. The overall prediction error rate is $19/53 \approx 0.36$. On the other hand, the absolute number of insertions, deletions and substitutions of phonemes was decreased by one: previously, the prediction of /z/ where /ks/ or /kʃ/ were expected resulted in four substitutions and four deletions; the current prediction of /k/ still requires four deletions where /ks/ or /kʃ/ were expected,

	Reference	*Predicted*	*PrE*	*SymE*	*StrE*
⟨flexure⟩	`flEK-R-`	`fl-k-r-`	3		
	/flɛkʃɚ/	/flkɹ/		3	1
⟨inflexion⟩	`InflEK-xn`	`Infl-kIxn`	3		
	/ɪnflɛkʃən/	/ɪnflkɪən/		2	1
⟨lynx⟩	`lIGX`	`lAnk`	3		
	/lɪŋks/	/laɪ̯nk/		3	1
⟨prefix⟩	`prifIX`	`fr-fIk`	3		
	/pɹifɪks/	/fɹfɪk/		3	1
⟨xenophobe⟩	`zEnxf-ob-`	`k-nxf-xb-`	3		
	/zɛnəfob/	/knəfəb/		3	1
⟨xerophyte⟩	`zIrxf-At-`	`k-rxf-At-`	2		
	/zɪɹəfaɪ̯t/	/kɹəfaɪ̯t/		2	1
⟨xylophone⟩	`zAlxf-on-`	`kAlxf-xn-`	2		
	/zaɪ̯ləfon/	/kaɪ̯ləfən/		2	1
		Totals	19	18	7

Figure 2.10: Example data set with multi-phoneme symbols. The predicted pronunciations minimize symbol error.

but only three substitutions when the reference pronunciation is /z/. The overall symbol (phoneme) error rate is $18/48 \approx 0.38$, and the overall string (word) error rate is unchanged at $7/7 = 1$ as in the previous example.

Incidentally, the optimal string error rate is $5/7 \approx 0.71$, since it is possible for a classifier to completely match the pronunciation of two words. Since by assumption the classifier in this toy example cannot take any letter context into account, it is difficult or impossible for it to correctly predict the pronunciation of words in which a given letter occurs more than once with different pronunciations. For example, the correct pronunciation of ⟨flexure⟩ could only be captured by assuming that both occurrences of the letter ⟨e⟩ are "silent", that ⟨x⟩ maps to /e/, and

that ⟨u⟩ is pronounced /kʃ/. However, such a mapping would not be able to account for the pronunciation of any of the other words. Similarly unreasonable assumptions would have to be made to correctly capture the pronunciation of ⟨xylophone⟩ due to the two occurrences of ⟨o⟩. The words ⟨inflexion⟩, ⟨xenophobe⟩ and ⟨xerophyte⟩ cannot be accommodated at all by this simple model, because of multiple occurrences of ⟨i⟩, ⟨o⟩ and/or ⟨e⟩.

Since the five words discussed so fare are all very hard or impossible to fit into the current classification model without automatically resulting in errors, we can ignore them completely if the goal is to minimize the number of words whose pronunciations are mispredicted. Among the remaining two words, ⟨lynx⟩ and ⟨prefix⟩, the only letter occurring more than once is ⟨x⟩, but its pronunciation is the same at both occurrences. We can therefore construct a mapping that will correctly predict the pronunciations of those two words. The consequences of such a mapping are shown in Figure 2.11. Notice in particular that we had to choose minority symbols as the pronunciations associated with the letters ⟨e⟩, ⟨n⟩, ⟨p⟩, ⟨x⟩, and ⟨y⟩. The overall string (word) error rate of $5/7 \approx 0.71$ is optimal, but the prediction error rate of $26/53 \approx 0.49$ as well as the symbol (phoneme) error rate of $29/48 \approx 0.60$ are much higher than before. We will have more to say about minimizing string error rate in the next two sections.

The discussion of the two example scenarios demonstrated that minimizing prediction error is generally independent from minimizing symbol error. However, there are situations where the two notions converge. For example, if the number of letters in a word equals the number of phonemes, and if no multi-phoneme symbols are used, then prediction error and symbol error coincide for that word, since any mispredicted phoneme will count as one substitution and there are no insertions or deletions. Also, if we managed to reduce prediction

	Reference	*Predicted*	*PrE*	*SymE*	*StrE*
⟨flexure⟩	`flEK-R-`	`fliX-ri`	4		
	/flɛkʃɚ/	/fliksɹi/		4	1
⟨inflexion⟩	`InflEK-xn`	`IGfliXIxG`	5		
	/ɪnflɛkʃən/	/ɪŋfliksɪəŋ/		5	1
⟨lynx⟩	`lIGX`	`lIGX`	0		
	/lɪŋks/	/lɪŋks/		0	0
⟨prefix⟩	`prifIX`	`prifIX`	0		
	/pɹifɪks/	/pɹifɪks/		0	0
⟨xenophobe⟩	`zEnxf-ob-`	`XiGxp-xbi`	6		
	/zɛnəfob/	/ksiŋəpəbi/		7	1
⟨xerophyte⟩	`zIrxf-At-`	`Xirxp-Iti`	5		
	/zɪɹəfa͡ɪt/	/ksiɹəpɪti/		6	1
⟨xylophone⟩	`zAlxf-on-`	`XIlxp-xGi`	6		
	/za͡ɪləfon/	/ksɪləpəŋi/		7	1
		Totals	26	29	5

Figure 2.11: Example data set with multi-phoneme symbols. The predicted pronunciations minimize string error.

error to its optimum of zero, then all other measures of error would have to be at their optima, including a symbol error of identically zero.

Prediction error is sensitive to differences in alignment, and we have seen an example where every single prediction by a classifier was wrong, yet the symbol error rate was zero. We explore related situations in the next section.

2.4.2 Prediction Error vs. String Error

In the previous scenario involving multi-phoneme symbols we had already seen an example where minimizing prediction error conflicts with minimizing string

	Reference	*Predicted*	*PrE*	*StrE*
⟨hat⟩	/hæt/	/hæt/	0	0
⟨hit⟩	/hɪt/	/hɪt/	0	0
⟨hut⟩	/hʌt/	/hʌt/	0	0
⟨kin⟩	/kɪn/	/kɪŋ/	1	1
⟨thank⟩	/θ-æŋk/	/thæŋk/	2	1
⟨think⟩	/θ-ɪŋk/	/thɪŋk/	2	1
		Totals	5	3

Figure 2.12: Example data set. The predicted pronunciations were produced by majority classifiers and minimize prediction error.

error. The same conflict can be seen on a more realistic sample, again taken from the NETtalk data set, that does not involve any multi-phoneme symbols.

The main point, illustrated in Figure 2.12 and Figure 2.13, is this: if the goal is to minimize string error, one need not worry about words that cannot be accommodated by the classification model; the string error criterion cannot distinguish between predicted pronunciations with one minor flaw on the one hand, and complete gibberish on the other. So a good strategy for minimizing string error is to identify and eliminate hopeless cases.

We assume the same classification setting as before: predict phonemes (possibly null) on the basis of isolated letters. The majority classifications, which minimize prediction error, can be seen in Figure 2.12. The key point is that ⟨n⟩ must be mapped to /ŋ/, because that combination occurs twice, not to /n/, which occurs only once. However, all the words that contain ⟨n⟩ paired with /ŋ/, namely ⟨thank⟩ and ⟨think⟩, are flawed beyond repair. In order to minimize string error, we can trade off a higher misprediction penalty on those two words for a lower

	Reference	*Predicted*	*PrE*	*StrE*
⟨hat⟩	/hæt/	/hæt/	0	0
⟨hit⟩	/hɪt/	/hɪt/	0	0
⟨hut⟩	/hʌt/	/hʌt/	0	0
⟨kin⟩	/kɪn/	/kɪn/	0	0
⟨thank⟩	/θ-æŋk/	/thænk/	3	1
⟨think⟩	/θ-ɪŋk/	/thɪnk/	3	1
		Totals	6	2

Figure 2.13: Example data set. The predicted pronunciations minimize string error.

string error, because it is possible to correctly predict the pronunciation of ⟨kin⟩ if the letter ⟨n⟩ is mapped to the minority symbol /n/ (we also see that the sample is biased in this regard, since overall /n/ would be more frequent than /ŋ/). The predicted pronunciations are shown in Figure 2.13. It is easy to see that a string error rate of 2/6 is indeed optimal: ⟨t⟩ must be pronounced as /t/, since any other choice would eliminate three words; for the same reason ⟨h⟩ must be pronounced as /h/. But that renders ⟨thank⟩ and ⟨think⟩ hopeless, and so these two words have no further role to play. Only ⟨kin⟩ remains, and it causes no additional errors.

In the discussion of Figure 2.4 above, we had noted that prediction error is sensitive to spurious distinctions. Every prediction of a hypothetical mapping, which we had called Classifier #2, was wrong, yet the concatenation of all individual prediction results matched the reference pronunciation. For convenience, this is repeated in Figure 2.14: the column labeled *PrER* shows prediction error rates, *StrER* string error rates, and *SymER* symbol error rates.

Orthography	s l a u g h t e r h o u s e	*PrER*	*SymER*	*StrER*
Reference alignment	s l ɔ - - - t - ɚ h a͡ʊ - s -			
Classifier #2 predictions	- - - s l ɔ - t - - ɚ h a͡ʊ s	14/14	0/8	0/1
Classifier #3 predictions	s l ɔ - - - t - ɚ h a͡ʊ - z -	1/14	1/8	1/1
Classifier #4 predictions	t o t ˌ ʌ t ɚ ʤ ɪ b ə ɹ ɪ ʃ	14/14	12/8	1/1

Figure 2.14: Examples of predicted pronunciations, contrasting prediction error rate and string error rate.

Whereas prediction error rate is too sensitive to differences in the predicted pronunciation from the reference pronunciation, string error rate is too oblivious. Consider another hypothetical classifier, Classifier #3, whose output is shown in Figure 2.14. Its prediction is slightly flawed, since the last segment was incorrectly predicted to be a /z/ instead of an /s/. Incidentally, this would be correct if the word was ⟨house⟩ and used as verb, but it does not generalize to ⟨slaughterhouse⟩. Prediction error rate is only $1/14 \approx 0.07$, but word error rate is 1 because the predicted pronunciation is wrong. Compare this with the output of Classifier #4, which is total utter gibberish. String error rate cannot distinguish between the massive problems of Classifier #4 and the minor flaw of Classifier #3, since neither prediction matches the reference pronunciation. On the other hand, Classifier #4 is equivalent to Classifier #2 in terms of prediction error rate, but the pronunciations they predict are radically different.

In practice, prediction error and string error are correlated. If prediction errors occur independently of one another, and if spurious prediction errors due to alignment differences are rare, then string (word) error rate can be estimated as

$1 - (1 - r)^n$ where r is the prediction error rate (so $1 - r$ is prediction accuracy; see also Section 2.5), and n is the average number of letters in a word.

2.4.3 Symbol Error vs. String Error

The previous discussion surrounding Figure 2.12 and Figure 2.13 exhibited a conflict between minimizing prediction error and minimizing string error. A similar conflict exists between minimizing symbol error and minimizing string error, and the exact same examples can be used to illustrate this.

An important property shared by symbol error and string error is their universal applicability. Unlike prediction error, which only applies when the overall task is decomposed into separate classification problems, symbol error and string error can be computed for any method that predicts pronunciations.

Unfortunately, many authors do not report symbol accuracy (or some variation thereof) for their approaches. Prediction error/accuracy is often used without being properly identified as such. For example, the seminal NETtalk journal paper talks about '[t]he percentage of correct phonemes' [Sejnowski and Rosenberg, 1987, p. 153], but it is clear that what is being measured is prediction accuracy of transcription symbols. In the NETtalk transcription system, a transcription symbol may correspond to zero, one, or two phonemes, and therefore speaking of the 'percentage of correct phonemes' is somewhat misleading.

The desire for a universally applicable evaluation metric prompted Damper et al. [1999] to recommend string (word) accuracy as the main criterion for comparing different approaches and systems. While Damper et al. [1999, p. 164] indicate that they 'generally favour [...] string-edit distance', they (correctly) characterize word accuracy as a simple metric that is 'more stringent and sensitive

than symbols correct scores'. Word accuracy is also the 'ultimate measure of performance' for Galescu and Allen [2001]. However, as the previous discussions have shown, word accuracy (resp. word error rate) is too fussy and conservative: by treating all errors alike, it cannot distinguish between minor flaws and major problems in the output of different schemes or classifiers.

We therefore recommend against using either string (word) error or prediction error as the sole universal evaluation metric. As Figure 2.14 illustrates, symbol (phoneme) error rate based on (some version of) string edit distance is preferable, because it is able to make more distinctions than string error without being sensitive to spurious alignment issues. Because symbol error rate is applicable in all situations, we recommend that it be used as the main evaluation metric.

2.5 Accuracy, Optimization and Approximations

Sometimes it makes sense to talk about accuracy instead of error rate. If the error rate of a classifier is r, its accuracy is defined to be $1 - r$.

Ideally, whatever metric one decides to use for evaluation should also be (a part of) the objective function that is optimized during model training. In practice this is often not the case. For example, Damper et al. [1999] compare several 'automatic-phenomization [sic] techniques' using string error rate as the main evaluation criterion. A NETtalk-like system is included in their comparison, which was trained on an aligned dictionary. However, the objective minimized during training of this system is prediction error, which prefers different models than string error. Even if a globally optimal model that minimizes prediction error could be found during training, there is no guarantee that this model will be optimal when evaluating based on a different criterion.

When using any of the above evaluation metrics as (part of) an optimization objective, we have a choice: we could either minimize error rate, or maximize accuracy. If we can find globally optimal solutions, this amounts to the same thing. But if we consider approximations to within relative bounds, the two optimization problems are generally different [Ausiello et al., 1999; Hromkovič, 2001].

Suppose that the true global optimum for a 1,000 word problem is a mapping which correctly predicts the pronunciation of 900 words and makes one or more mistakes on 100 words. Further assume that we have two approximation algorithms with an approximation ratio of 1.2, one of which is a maximization algorithm and the other one a minimization algorithm. If we use the maximization algorithm to maximize string (word) accuracy, then the 1.2 approximation ratio means that the algorithm is guaranteed to find a feasible solution that results in the correct pronunciation of at least $900/1.2 = 750$ words. In the worst case, this means an empirical word error rate of 25%. On the other hand, if we minimize word error rate with the 1.2-approximate minimization algorithm, we are guaranteed to find a solution that results in the incorrect pronunciation of at most $100 \times 1.2 = 120$ words. The worst-case word error rate is then only 12%.

2.6 Conclusion

This chapter reviewed the most commonly used evaluation metrics for letter-to-sound rules. Prediction error is typically used by approaches based on aligned data. However, it is only applicable for aligned data, and because it takes alignments into account, it is sensitive to spurious alignment differences. Both properties are undesirable, and prediction error should not be used as a general evaluation criterion. By contrast, string error is universally applicable and oblivious

to alignment information. It is, however, also oblivious to subtle differences between predicted strings and reference strings, since it is a 0-1-loss function that can only distinguish between perfectly correct predictions vs. flawed predictions. While the boundary could be redrawn so that perfectly correct and slightly flawed predictions incur a loss of zero and everything else a loss of one, this still does not change the fact that string error provides only a single bit of information. String error may exaggerate the differences between classifiers, for example, if one classifier makes many perfect predictions, and the other is not far off but its predictions contain minor mistakes. At the same time, important differences between classifiers may be obscured by string error, like when one classifier's predictions contain subtle flaws and the other one's major blunders, since both would count as equally wrong. Comparisons based on string error, such as Damper et al.'s [1999], may therefore not be very trustworthy. While Damper et al. [1999] are right that string error rate is a very 'stringent' criterion (no pun intended on their part), it is precisely because of this very quality that string error rate and string accuracy should *not* be used.

Instead, we recommend the use of some variant of what we have referred to as symbol error. Called word error (rate) in speech recognition, symbol error is widely used in that area. When symbol error is defined as ordinary Levenshtein edit distance, it can be used to calculate string error, since a string is perfectly correct iff its edit distance to the reference string is zero. In other words, there is a forgetful mapping that converts edit distance into string error, returning a string error of zero for an edit distance of zero, and a string error of one otherwise. Despite this, we saw that optimizing edit distance may lead to different results than optimizing string error. Because edit distance makes more distinctions than string error, it provides more information about the differences between classifiers, and

its use should therefore be preferred over string error. However, for technical reasons that will be discussed in Section 5.5, Levenshtein distance may sometimes be a bit cumbersome to work with, in which case it will be better to define symbol error in terms of the probability of one string being mistaken for another.

The key result of this chapter is the demonstration that the three different evaluation measures discussed here lead to distinct optimization problems. Which metric to use as an optimization objective or loss function during model training is an important question. Although the choice may depend on technical considerations discussed in the following chapters, ideally it should match the measure used for empirical evaluation. This is a general concern that goes beyond the particular task of learning and evaluating letter-to-sound mappings discussed here. Similar issues arise in machine learning of information extraction components, where it is often the case that some variant of prediction error specific to a particular approach is minimized during model training, whereas the quality of the trained components is measured in terms of (the harmonic mean of) precision and recall. We suspect that these different criteria are also pulling in different directions, just like the tug-of-war described in this chapter.

CHAPTER 3

LEARNING DETERMINISTIC TRANSDUCERS

3.1 Introduction

Much of the traditional literature on learning letter-to-sound rules has focused, often implicitly, on deterministic finite transducers. Deterministic transducers compute functions from a formal language $L_1 \subseteq \Sigma^*$ to another formal language $L_2 \subseteq \Gamma^*$, and for finite transducers these languages are regular. For example L_1 might be the set of French orthographic words and L_2 the set of valid French phoneme strings. We will omit all references to L_1 and L_2 and generally speak of partial functions from Σ^* to Γ^*, where Σ would, for example, be the set of letters of the Roman alphabet, and Γ a set of transcription symbols or phoneme symbols.

Learning letter-to-sound rules could be viewed directly as learning certain kinds of deterministic finite transducers. The so-called subsequential transducers are a very general class of deterministic finite transducers, and this class can be learned (in a specific sense) from positive data, which is the topic of Section 3.2. Most other machine learning approaches have treated the overall learning task as classifier learning (under a very different definition of learning), by reducing the overall prediction task to a classification task. Section 3.3 shows that the mappings computed by these approaches fall within a restrictive class of deterministic

finite transductions, so-called local transductions. A local transductions is in turn equivalent to a deterministic preprocessing step followed by a very simple mapping characterized completely by how it maps symbols in Σ to strings in Γ^*. This is the topic of Section 3.4. The goal is to restrict the class of mappings that have to be learned, so that one can concentrate on the essence of the learning problem. In Section 3.5 the complexity of several restricted learning problems is investigated, providing some formal backing for the main point Chapter 2 that different loss functions give rise to very different learning problems.[1]

3.2 Subsequential Transducers

Oncina et al. [1993] presented an algorithm (OSTIA) that can infer subsequential transducers from positive samples, where inference is understood as identification in the limit [Gold, 1967]. The limit identification paradigm features a learner that receives an infinite stream of samples, and for each sample it reads it outputs a hypothesis consistent with all samples seen up to that point. The learner identifies the target concept that generated the samples in the limit, if the stream of hypotheses it generates has the target concept as a fixed point. This view of learning has many unrealistic aspects, some of which will be discussed in terms of the specific learning algorithm OSTIA.

Subsequential transducers are like sequential transducers, or (deterministic) generalized sequential machines [Eilenberg, 1974, ch. XI], but have designated final states with an associated string output function. OSTIA was subsequently applied to various natural language learning tasks by some of the the authors of the original OSTIA paper and their colleagues. One of the first attempts (and

[1] Some of the material presented in this chapter appears in Jansche 2003, which is copyright © 2003 by the present author.

apparently the only one so far) to use OSTIA for phonemic modeling (letter to sound rules, post-lexical rules, pronunciation modeling) is the work of Gildea and Jurafsky [1994, 1996].

Some important characteristics of OSTIA can be summarized as follows:

Brittleness OSTIA cannot deal with noisy data containing imperfect generalizations or genuine exceptions. If there is any evidence in the training samples that the resulting machine cannot be subsequential, the learning algorithm will abort.

Because a typical pronunciation dictionary will contain homographs, which share a common orthographic form but differ in pronunciation, for example ⟨associate⟩ /əsosiət/ (noun) vs. /əsosiet/ (verb), one cannot use such a pronunciation dictionary directly as training data for OSTIA. Even if homographs were removed, there is no guarantee that OSTIA will terminate normally, since there can easily be less obvious cues for the non-subsequentiality of the data.

Out-of-class behavior On the other hand, there are situations where the training data are actually generated from a non-subsequential transducer, but the learning algorithm fails to realize that, in the sense that it would fail to identify that relation in the limit. Limit-identification does not require that the algorithm terminates once it has seen a characteristic sample of the target relation, so it could consume more and more training data without ever converging to a fixed point.

Many definitions of learning, including limit identification and classical PAC learning, assume that the target concept falls within the concept class that the learner can identify. In practice, this assumption is often unrealistic, and

it is often sufficient that a learner finds the best hypothesis for a set of observations, regardless of whether the true concept falls within the concept class of the learner. Such a learner is usually called *agnostic*, because it makes no assumptions about the data it sees.

The previous point already demonstrated that OSTIA is not agnostic, since it terminates abnormally on some out-of-class samples. (Open problem: Can there be an agnostic version of OSTIA? If so, should it be called OSTIA-GNC?) For other out-of-class samples OSTIA is more reasonable. Consider the following set of samples:

a	a
aa	ba
aaa	aba
$aaaa$	$baba$

Suppose these data were generated by the nondeterministic (and hence non-sequential) transducer shown in Figure 3.1. This is not a completely artificial example, as transductions of this form do actually arise in natural phonological systems [Jansche, 1998]. Observe that the transducer in Figure 3.1 is unambiguous, and that the transduction it computes is therefore a rational function [see for example Roche and Schabes, 1997b, sec. 1.3.6 and references cited therein], in fact a total function $f : \{a\}^* \to \{a, b\}^*$ defined by $f(a^{2n}) = (ba)^n$, and $f(a^{2n+1}) = a(ba)^n$ for all $n \in \mathbb{N}$. Crucially, any sample of the behavior of f, i. e. a finite subset of the set $\{\langle x, y\rangle \mid f(x) = y\}$ (called the *graph* of f; see Definition 3.1), can be represented by a subsequential transducer.

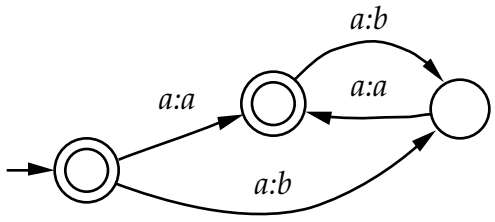

Figure 3.1: An unambiguous but nondeterministic transducer realizing the rational function $\{\langle a, a\rangle, \langle \epsilon, \epsilon\rangle\}\{\langle aa, ba\rangle\}^*$.

As more and longer strings are fed to OSTIA, the automaton it learns will grow accordingly, but the learning procedure will never converge, because at each point only a finite amount of data has been seen and there clearly exist subsequential transducers consistent with those data. For any non-trivial sample, the transducer more or less memorizes by rote all strings up to the longest string in the training sample, and any generalizations beyond the longest string will be wrong.

In other words, we have an example of an out-of-class target function for which OSTIA will neither converge (in the limit) nor discover the fact that the target concept does not fall within the learnable concept class. Although it will terminate normally on any finite set of samples, it returns a hypothesis that is guaranteed to be incorrect.

Partial mapping There is no guarantee that OSTIA always finds a subsequential transducer which accepts all possible input strings. Determining whether

this is the case is easy: check if the complement (with respect to Σ^*) of the first projection of the transducer is empty. However, it is not clear what to do if the function computed by the inferred transducer is not total: we still need to find a way to determine the pronunciations of words that the transducer rejects, so we are back to where we started.

Wrong hypothesis space In practice, OSTIA tends to converge slowly and the generalizations it obtains seem unnatural [Gildea and Jurafsky, 1994, 1996]. (This may very well be a consequence of the first two points discussed here.) Gildea and Jurafsky [1994] propose ways to add bias in the form of constraints on the alignment of symbols on the input and output tape. This seems entirely justified for phonemic modeling, but the fact that it was necessary to add this bias suggests that OSTIA was not the right tool to use in the first place, since it does not make use of the locality of information inherent in many phonemic generalizations (see the discussion of local letter context in Section 2.3.2).

Data requirements On the positive side, the input to OSTIA is a set of pairs of input/output strings (training samples). Unlike in many other approaches, there is no requirement that the two strings in a sample must be of equal length (aligned).

In sum, the OSTIA algorithm has many properties that make it unsuitable for the task of learning letter-to-sound transducers. If one is only interested in "off-the-shelf" solutions, OSTIA should not be the first choice, and probably not even the second. However, a desirable property of OSTIA is the fact that it works with unaligned data. Still, we prefer algorithms that are robust to irregularities ("noise") in their training data, degrade gracefully on out-of-class input, and

whose outputs are total functions of a restricted class appropriate for the letter-to-sound task. The traditional approaches discussed in the next section are diametrically opposed in terms of the properties discussed here, as they avoid many of the shortcomings of OSTIA, but require aligned training data. An ideal learning algorithm should retain the desirable characteristics of the traditional approaches and of OSTIA. The algorithms in Chapter 5 arguably come closest to that ideal.

3.3 Strictly Local Transducers

The approach to learning letter-to-sound mappings used by NETtalk [Sejnowski and Rosenberg, 1987] has become well known, perhaps partly due to the attention paid to NETtalk within the context of the so-called symbolism vs. connectionism debate. The basic technique, which we had briefly touched on in Section 2.3.2, did not originate with NETtalk; in fact, Sejnowski and Rosenberg [1987] refer to prior work by Lucassen and Mercer [1984], and the general idea may have much earlier origins.

The key aspect is the reduction of the transducer learning problem to a well understood classifier learning problem. The fact that NETtalk employed artificial neural network classifiers is at most of secondary interest here, though the choice of classifier learner has sparked a separate, more focused debate [see for example Stanfill and Waltz, 1986; Dietterich et al., 1995; Daelemans et al., 1999], distinct from the connectionism debate. Many kinds of classifiers and classifier learners have been used in the literature, including artificial neural networks [Sejnowski and Rosenberg, 1987], memory-based learning [Stanfill and Waltz, 1986; Stanfill, 1987], rules [Hochberg et al., 1991; Fisher, 1999], and decision trees [Lucassen and Mercer, 1984; Riley, 1991; Daelemans and van den Bosch, 1997; Jiang et al., 1997;

Black et al., 1998; Chotimongkol and Black, 2000; Jansche, 2001; Sproat, 2001]. In addition, closely related approaches have employed iterated version spaces [Hamilton and Zhang, 1994], transformation-based learning [Huang et al., 1994; Bouma, 2000], and Markov models [Minker, 1996], which have also been used for the inverse task of phoneme-to-letter conversion [Rentzepopoulos et al., 1993; Rentzepopoulos and Kokkinakis, 1996]. Eager learners that infer structurally simple classifiers, typically rules or decision trees, have received additional attention. For example, decision trees can be compiled into finite state automata, either directly [Sproat and Riley, 1996] or indirectly [Sproat, 2001] by providing an implementation of an abstract data type for finite state machines [Mohri et al., 2000].

A common observation is that overly eager learners that aim to produce concise descriptions are at a disadvantage. For example, Dietterich et al. [1995] and Bakiri and Dietterich [1993, 2001] use the decision tree learner ID3 [Quinlan, 1986] without pruning or early stopping. The IGTREE algorithm formulated and used by Daelemans and van den Bosch [1997] explicitly eschews the use of a pruning step. Later, Daelemans et al. [1999] argue that removing rare or exceptional instances is generally harmful.

For classifier-based approaches two assumptions are required. First, letter strings must be of the same length as the corresponding phoneme strings. Since this is not normally the case for ordinary pronunciation dictionaries, one must somehow transform the dictionary, for example by padding shorter strings or contracting adjacent symbols into multi-symbol units in order to shorten longer strings. The NETtalk data set (Section 2.2.2) is the end result of such a transformation. It satisfies the first requirement, namely in all of its entries the orthographic form has the same length as its transcription. Second, we assume locality in the sense that each output symbol (phoneme) can be predicted based on its aligned

input symbol (letter) plus a fixed amount of surrounding input context. The second, more specific assumption (essentially a Markov assumption) is the topic of the next subsection; the aligned data requirement is discussed after that in Section 3.3.2.

3.3.1 Locality Assumption

Let us look again at the word ⟨slaughterhouse⟩ and its reference alignment from Figure 2.4 or Figure 2.14. Suppose that each symbol of the reference alignment can be predicted from the corresponding letter plus two letters of context on each side. This means that we need to examine all substrings of length 5 and predict a corresponding phoneme. Special treatment is required for the edges of the string. To keep things simple, we pad the letter string on both sides with placeholder symbols ⟨-⟩. The approach can then be visualized as sliding a fixed window of size 5 across the padded letter string, outputting a transcription symbol at each window position. This is shown in Figure 3.2; letters outside the window are show in a lighter shade. This visualization happens to include the inessential property that the fixed window slides smoothly (or at least as smoothly as is possible for a discrete object) across the input string from left to right, when in fact any permutation of the window positions would do, as long as the outputs are concatenated in the order corresponding to the window positions.

Each line of Figure 3.2 can be understood as pairing an input string of length 5 with an output symbol, in this case a symbol from Figure 2.2 or the symbol /-/. In other words, under the assumption of locality the transduction from letter strings to phoneme strings is completely characterized by a function from letter windows

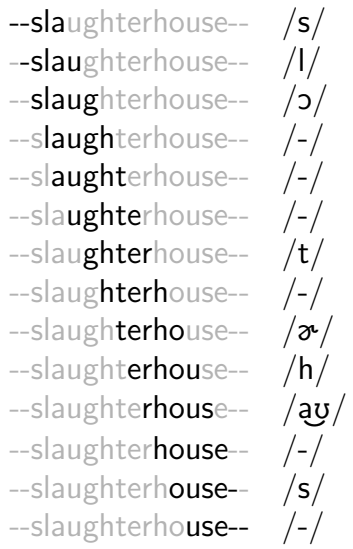

Figure 3.2: An illustration of strict locality in terms of a symmetric sliding window of size 5.

to transcription symbols. So the problem of inferring a local transducer from positive samples has been reduced to the problem of learning a function onto a finite codomain, which is precisely a classifier learning problem. The training instances for a supervised classifier learner then consist of the input (letter) windows, suitably encoded, and the output (transcription) symbols are the class labels that we want to predict.

For lack of a better term we shall speak of *strictly k-local same-length transductions* by analogy with the strictly k-testable languages [McNaughton and Papert,

1972]. Moreover, strictly k-testable languages are the languages accepted by scanner ("sliding window") automata (compiler implementers would speak of "peephole optimizers"), and strictly k-local same-length transductions are transductions computed by scanner transducers (the sliding window process described above).

The strictly 2-testable languages are often referred to as *local languages* [McNaughton and Papert, 1972]. A local language over a finite nonempty alphabet Σ is characterized by its permissible prefixes $P \subseteq \Sigma$, suffixes $S \subseteq \Sigma$, and substrings (factors) $F \subseteq \Sigma\Sigma$. Let $G = \Sigma\Sigma - F$, then the local language characterized by P, S, and F can be defined by the following star-free expression, where the complement $\overline{A}$ of a set A is taken with respect to Σ^* (so in particular $\overline{\{\}} = \Sigma^*$):

$$P\overline{\{\}} \cap \overline{\{\}}S \cap \overline{\overline{\{\}}\, G\, \overline{\{\}}} \tag{3.1}$$

This can be extended to transductions from an input alphabet Σ^* to an output alphabet Γ^*. First, we need a few auxiliary definitions:

Definition 3.1 (Graph of a relation). Given a relation $R : A \to B$ where A and B are sets, define $\#(R)$, the *graph of* R, to be the set $\{\langle a, b\rangle \in (A \times B) \mid aRb\}$. Note that this includes the case of R being a function.

Definition 3.2 (Zip). Given two same-length sequences $u = \langle u_1, \ldots, u_n\rangle$ and $w = \langle w_1, \ldots, w_n\rangle$ for some integer $n \geq 0$, define $\mathrm{zip}(u, w) = \langle\langle u_1, w_1\rangle, \ldots, \langle u_n, w_n\rangle\rangle$. Extend this operation to sets $S \subseteq \bigcup_{n\in\mathbb{N}} \Sigma^n \times \Gamma^n$ and define $\mathrm{zip}(S) = \{\mathrm{zip}(u, w) \mid \langle u, w\rangle \in S\}$.

Local transductions can now be defined in terms of two functions $p : \Sigma \to \Gamma$ and $f : \Sigma^2 \to \Gamma$. Define the relation $F : \Sigma^2 \to \Gamma^2$ for which uFw holds just in case

$w = ab$ for some $a \in \Gamma$ and $b = f(u)$ (if f is defined for u). Let $P = \#(p)$ and $G = (\Sigma \times \Gamma)^2 - \mathrm{zip}(\#(F))$, and define deterministic local same-length transductions in terms of the following star-free expression (complements are now taken with respect to $(\Sigma \times \Gamma)^*$):

$$P\overline{\{\}} \cap \overline{\overline{\{\}}\, G\, \overline{\{\}}} \tag{3.2}$$

This definition is possible because same-length rational transductions are in many respects equivalent to regular languages over symbol pairs, and so they are closed under intersection and complementation. If the functions p and f are total, so is the corresponding local transduction.

The generalizations to strictly k-testable languages, resp. strictly k-local transductions are straightforward.

Definition 3.3 (Strictly k-testable language). A strictly k-testable language for fixed $k > 0$ over a finite alphabet Σ is characterized by a tuple $\langle T, P, S, F\rangle$ consisting of permissible short strings $T \in \bigcup_{n=0}^{k-2} \Sigma^n$, prefixes and suffixes $P, S \in \Sigma^{k-1}$, and substrings $F \in \Sigma^k$. Let $G = \Sigma^k - F$. The language is then defined as

$$T \cup \left(P\overline{\{\}} \cap \overline{\{\}}S \cap \overline{\overline{\{\}}\, G\, \overline{\{\}}}\right)$$

A language is said to be strictly locally testable if it is strictly k-testable for some integer k. Our definition of strictly k-testable languages differs slightly from various definitions given in the literature [McNaughton and Papert, 1972; Zalcstein, 1972; García and Vidal, 1990; Yokomori and Kobayashi, 1998], but our class of strictly locally testable languages includes all analogous classes defined in the literature.

Definition 3.4 (Strictly k-local deterministic same-length transduction). For a fixed $k > 0$, a strictly k-local deterministic same-length transduction over finite alphabets Σ and Γ is characterized by a tuple $\langle t, p, f \rangle$ where $t \in \bigcup_{n=0}^{k-2} (\Gamma^n)^{\Sigma^n}$ is a length-preserving mapping from $\bigcup_{n=0}^{k-2} \Sigma^n$ into Γ^*; and where p is a function $\Sigma^{k-1} \to \Gamma^{k-1}$; and f is a function $\Sigma^k \to \Gamma$. Define $F : \Sigma^k \to \Gamma^k$ to be the relation for which uFw holds just in case there exists a string $v \in \Gamma^{k-1}$ and a symbol $f(u) \in \Gamma$ such that $w = va$ and $a = f(u)$. Let $T = \text{zip}(\#(t))$, $P = \text{zip}(\#(p))$, and $G = (\Sigma \times \Gamma)^k - \text{zip}(\#(F))$. The transduction is then defined as

$$T \cup \left(P\overline{\{\}} \cap \overline{\overline{\{\}}\, G\, \overline{\{\}}} \right)$$

Strict locality plays an important role in many linguistic domains. It has often been argued that local letter context provides most of the information for letter-to-sound conversion in languages like English [Lucassen and Mercer, 1984] or French [Laporte, 1997]. In the analysis of lexical tone, the mapping between underlying tones and surface tones is often local: for example the tone systems studied by Gibbon [1987, 2001] are strictly local, but apparent exceptions can also be found [Jansche, 1998]. The kinds of n-gram language models often used in speech recognition are essentially stochastic versions of the strictly n-testable languages [García and Vidal, 1990; Torres and Varona, 2001]. Word n-gram models are clearly only an approximation to the syntax of natural languages [Brill et al., 1998]; however, Kornai [1985] has argued that natural language syntax is non-counting and regular, which would mean that formal descriptions would fall into the class of star-free languages.

Phonological theory has long been concerned with strict locality in the sense that segmental processes operate exclusively on adjacent segments. If we assume

that phonemic systems of natural languages are always regular, then strict locality holds more or less trivially because of the morphic generator representation theorem for regular languages [Eilenberg, 1974, p. 27; Salomaa, 1981, p. 97f.], which states that every regular language is the homomorphic image of a local language under an alphabetic substitution. Put somewhat overly simply, as long as the alphabet can be enriched, every regular language can be represented by a local language. However, enriching the inventory of phonemes is precisely the solution often adopted. For example, a language may be described in terms of a process whereby nasalization spreads transparently through laryngeals, e. g., all vowels following a nasal consonant become nasalized if there is no intervening non-nasal consonant, the exception being /h/ which is "transparent" to the spreading of nasalization. The well-formed phoneme strings of such a natural language cannot be described by a local formal language. However, if we introduce a new symbol /h̃/ (for a "nasalized /h/"), then nasalization spreading is local if /h̃/ is restricted to occur in nasalized contexts and /h/ occurs only in non-nasalized contexts. An alphabetic substitution – perhaps best thought of as a phoneme to phone mapping – then maps both plain /h/ as well as nasalized /h̃/ to the phone [h]. In other words, as long as the phoneme inventory can be enriched, all regular phonemic systems can be described in terms of local languages, and theoretical claims about phonological locality are therefore empirically vacuous.

An important consequence of strict locality is that identical substrings of sufficient length will be mapped to the same output no matter where they are located within a larger string. This is desirable for codes because it limits the effects of errors [Winograd, 1964], and is also useful for letter-to-sound transductions. Take the example of ⟨slaughterhouse⟩ again and compare it with ⟨laughterhouse⟩. Both are compound nouns containing the noun ⟨house⟩ as the second component, and

it receives the same pronunciation in both words, no matter what preceded it. On the other hand, we need to make sure that we choose a large enough window, otherwise the substring ⟨laughter⟩ would, incorrectly, be pronounced the same in ⟨slaughterhouse⟩ and ⟨laughterhouse⟩. Letter-to-sound applications use fairly large window sizes: NETtalk [Sejnowski and Rosenberg, 1987] initially used a symmetric seven-letter window, but Sejnowski [1988] mentions subsequent experiments with 11 letters. Other researchers have experimented with window sizes up to 15 letters: Bakiri and Dietterich [2001] use symmetric windows with 7 to 15 letters of context; Stanfill and Waltz [1986] employ an asymmetric window with four letters preceding the central letter and ten letters following it.

The transductions commonly used in practice differ from the idealized definition of strictly k-local transductions given above. On that definition, a transducer with $k = 15$ would treat words of length 13 or less specially, which would affect more than 99% of the NETtalk data set, whose average word length is 7.3 letters. In practice, if the window has l letters of context preceding the central letter and r letters of context following it, the input string is padded with l dummy symbols on the left and r dummy symbols on the right and the padded string is then scanned and processed. Setting $k = 15$ only means that up to 15 letters of context *may* be used, and one often has to back off to much shorter conditioning contexts. On the other hand, using ten letters of lookahead like Stanfill and Waltz [1986] makes it theoretically possible to perfectly predict the pronunciations of all words (ignoring homographs) of length 10 or less, which together make up more than 89% or the NETtalk data.

If we contrast the current classifier-based approach with OSTIA discussed in Section 3.2, we see that many of the deficiencies of OSTIA are absent or greatly reduced.

Robustness to variation Brittleness is generally not an issue, since most classifier learners can deal with conflicting labeling arising from substrings like ⟨aughter⟩ being pronounced differently in ⟨slaughter⟩ vs. ⟨laughter⟩. Moreover, no serious classifier learner would terminate abnormally just because it was presented with two or more identical instances bearing conflicting labels.

Robustness to out-of-class data The current approach is well behaved on out-of-class training data, i. e., it is agnostic in the same sense OSTIA is not. By examining substrings of length k it constructs a k-local approximation to whatever target function the training data were actually sampled from. If the target concept happens to be k-local, it can be inferred exactly if the training data are a characteristic sample. Moreover, there exist polynomial-time algorithms that determine whether a regular language is k-testable and which find an approximate minimum value of k for which a language is k-testable [Kim and McNaughton, 1994].

Total mapping Under some reasonable assumptions it is easy to ensure that the inferred transducers realize total functions: even though there may be letter sequences of length k that are not found in the training data, one can back off to shorter contexts, and in the worst case only the central letter of a window is used for prediction. Assuming all letters of the alphabet have been seen in the training data, one can easily guarantee that the classifier and, therefore, the inferred transducer correspond to total functions on their respective domains.

More appropriate hypothesis space The strictly local deterministic same-length transductions are a proper subclass of the subsequential transductions, yet

by focusing on the more restrictive class of concepts we do not seem to lose anything (this is hard to quantify, since our attempts to use OSTIA were unsuccessful for the reasons described in Section 3.2). As mentioned above, strict locality has been observed and quantified [Lucassen and Mercer, 1984] for many linguistic domains, especially for phonemic modeling.

Data requirements On the downside, the classifier-based approach described here requires same-length data, which typically means aligned pronunciation dictionaries. If aligned data are not available, an alignment procedure [see for example Daelemans and van den Bosch, 1997; Sproat, 2001] must first be carried out, which can be a source of mistakes. But even if aligned data like the NETtalk dictionary are used, there is no guarantee that the alignments are consistent or helpful.

3.3.2 Aligned Data Requirement

The requirement of aligned data is troubling, for several reasons. First, it means that in general training data will have to be preprocessed by an alignment procedure. There are many possible alignments of two strings – often exponentially many, depending on the alignment model – from which an alignment procedure has to pick one. Moreover, when an alignment procedure is applied to a whole pronunciation dictionary, it would not suffice if it were to randomly pick an alignment for each dictionary entry. Whatever criteria are applied to pick an alignment between a letter string and a phoneme string must be applied consistently across the dictionary.

We can view this as a combinatorial optimization problem, if we can supply an objective function that quantifies the overall quality of an aligned dictionary.

It should be kept in mind that the overall goal goes beyond the production of aligned dictionaries. In fact, we do not regard the result of the alignment step as an essential level of representation, we have not defined any evaluation criteria that would apply to aligned dictionaries, and we doubt that there exist useful independent objective criteria. We see an aligned dictionary solely as a practical requirement for the classifier-based approach to learning letter-to-sound transducers, and therefore the quality of an aligned dictionary can only be quantified in terms of its utility and contribution to the overall learning task.

It is then easy to define an objective function for the alignment procedure: the quality of an aligned dictionary is defined as the quality of the optimal classifier inferred on the basis of the aligned dictionary. It is not clear that there are efficient algorithms for an alignment procedure with this objective. For one thing, optimal classifier learning can be a hard problem [Hyafil and Rivest, 1976, see for example]. But even if we assume the presence of an oracle that can tell us the classification performance of an optimal classifier trained on a given aligned dictionary, formulating efficient algorithms for the alignment step does not seem straightforward. While the problem of aligning two strings can be decomposed and solved efficiently by a dynamic programming algorithm [Wagner and Fischer, 1974], the problem we are faced with is finding optimal alignments over an entire collection of string pairs, for which an equally simple decomposition does not seem possible in general. The problem is that we are not simply interested in the least-cost alignment of two strings given a fixed cost function. Rather, the global cost function has to be treated as variable and linked to the quality of a classifier trained on the aligned data, so that the cost function and alignments are mutually dependent: alignments are chosen to minimize cost, and the cost function depends on the performance of a classifier trained on aligned data.

In practice, alignment procedures sometimes use ad hoc solutions [Daelemans and van den Bosch, 1997], or optimize alignments under a very simple classification model while actually using much more sophisticated classifiers [Sproat, 2001]. In this last approach, the classification model used for computing alignments is essentially the same as in most of the toy examples in Section 2.4, namely phonemes are predicted based on a window of size one, i. e. without using any context preceding or following the central letter. This means that the optimal classifier that minimizes prediction error is the one that picks for each letter the phoneme most frequently associated with it. Sproat [2001] uses a relatively simple non-contextual model like this to obtain an initial alignment, but the decision tree classifiers inferred on the basis of the aligned data do use contextual information from a symmetric seven-letter window. The overall approach proposed by Sproat [2001] is unique in that it is iterative: alignments are updated and recomputed based on the predictions of the richer decision tree classifier. However, the improvements from the iterative updates are comparatively small, which would suggest that the overall performance depends fairly crucially on the quality of the initial alignment and the model used to obtain that alignment.

Starting from aligned data also makes the overall learning task less interesting if the goal of learning is to discover interesting regularities in the data. For example, Rentzepopoulos et al. [1993, p. 324] describe their segmentation and alignment procedure for the phoneme-to-letter task and conclude:

> [This] procedure has to be done off-line by hand in order to produce a set of rules for the segmentation of the input speech into symbols. This is the only part of the algorithm which is language-specific and requires knowledge of the spelling of the language.

On this view, which is entirely justifiable, discovering key aspects of the spelling system of a language is outside the scope of the learning algorithm proper, and

has been moved into the typically marginalized area of data preparation. Similar criticism could be leveled against approaches to letter-to-sound that use a fixed inventory of multi-phoneme symbols [Sejnowski, 1988; Minker, 1996]: the fact that in the NETtalk dictionary the letter ⟨x⟩ is predominantly aligned with multi-phoneme symbols that stand for the phoneme strings /ks/, /kʃ/, or /gz/ makes it unnecessary to discover that ⟨x⟩ is somehow special in English orthography.

Ideally, we would prefer algorithms that work with unaligned data and integrate the discovery of alignments, if necessary, tightly with the discovery of letter/phoneme correspondences. The approaches discussed here either require aligned data, or use a simple alignment procedure that is usually (Sproat 2001 is the exception) decoupled from the classifier inference procedure.

3.4 Morphisms of Free Monoids

Computational aspects of the classifier-based approach can be studied abstractly in terms of inferring certain morphisms of free semigroups or free monoids. Recall that a nonempty string over a fixed finite alphabet Σ can be viewed as an element of the free semigroup generated by Σ, which is usually written Σ^+. Formally, a semigroup is a tuple $\langle A, \cdot \rangle$ where $\cdot$ is an associative binary operation on A, called multiplication, and A is closed under this multiplication operation. A semigroup $\langle A, \cdot \rangle$ is free just in case every element has a unique factorization, and it is freely generated by a subset $B \subseteq A$ if every element has a unique factorization in terms of elements of B. If a semigroup lacks a multiplicative identity element, it is always possible to add one to turn the semigroup into a monoid. For example, $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$ is the monoid freely generated by Σ, where we have added ϵ, the empty string, as the identity element.

A semigroup morphism $f : A \to B$ preserves the structure of semigroups, i. e., $f(x \cdot y) = f(x) \cdot f(y)$ for all $x, y \in A$. A monoid morphism also maps the unit element of A onto the unit element of B. A semigroup morphism whose domain is a free semigroup Σ^+ is completely characterized, due to unique factorization, by the images it assigns to the generators of the semigroup, i. e. the elements of Σ: if the factorization of $w \in \Sigma^+$ is $w_1 \cdots w_n$, then $f(w) = f(w_1) \cdots f(w_n)$ with $w_i \in \Sigma$ for $1 \leq i \leq n$. The same holds for monoid morphisms.

Another property of free monoids and free semigroups is that any function $f : \Sigma \to A$ from a finite set Σ onto a monoid (resp. semigroup) A can be uniquely extended to a monoid morphism $\Sigma^* \to A$ (resp. semigroup morphism $\Sigma^+ \to A$), which we call f^*. To evaluate $f^*(w)$ when f is known, either $w = \epsilon$ and so $f^*(w) = \epsilon$ by definition, or else $w \in \Sigma^+$ can be uniquely factorized into $w_1 \cdots w_n$ for some $n > 0$ where $w_i \in \Sigma$ $(1 \leq i \leq n)$, and $f^*(w) = f(w_1) \cdots f(w_n)$.

Sequential string-to-string transductions can be viewed as generalizations of morphisms of free monoids [Eilenberg, 1974]. Conversely, morphisms of free monoids are precisely those mappings that can be computed by generalized sequential machines with a trivial one-state topology [Eilenberg, 1974, p. 299]. We are especially interested in the following two classes of morphisms between free monoids [Eilenberg, 1974, p. 6]:

Definition 3.5 (Fine morphism). A morphism $f : \Sigma^* \to \Gamma^*$ between free monoids is called a *fine morphism* just in case $f(x) \in \Gamma \cup \{\epsilon\}$ for all $x \in \Sigma$.

Definition 3.6 (Very fine morphism). A morphism $f : \Sigma^* \to \Gamma^*$ between free monoids is called a *very fine morphism*, or an *alphabetic substitution*, if $f(x) \in \Gamma$ for all $x \in \Sigma$.

The learning problem associated with the classifier-based approach can now be formulated in terms of inference of a very fine morphism: if we use a window of size one, then the classifier we want to learn is formally a very fine morphism. This gives us a different perspective on the learning problem, as we can now ask what would happen if we generalized the hypothesis space to fine morphisms. That would mean that we no longer require that letter strings and phoneme strings are of the same length, since a fine morphism is allowed to erase certain letters. So an approach based on inference of fine morphisms can be applied to pairs of letter strings and phoneme strings provided the length of each phoneme string is less than or equal to the length of the corresponding letter string. This is the case for more than 98% of all entries in CMUdict, and also among its 112,108 purely alphabetic entries (whereas only about 24% of the CMUdict entries have letter strings and phoneme strings of equal lengths). Inference of a fine morphism thus includes the computation of alignments for the vast majority of the entries of an ordinary unaligned pronunciation dictionary.

Limiting our attention to one-letter windows does not imply a loss of generality. We can easily incorporate more context by a combination of deterministic preprocessing steps ("shingling"). The idea is to cram the desired context into the one-letter window by enriching the input alphabet. If we ultimately want to use windows of size k for prediction, we choose a new input alphabet that slightly extends Σ^k. Because the original alphabet Σ is finite, the new alphabet too will be finite (and in fact only polynomially larger), and so we stay within the realm of finite automata.

Input (letter) strings can be augmented straightforwardly to include predetermined amounts of left context (history) or right context (lookahead). We illustrate the two simplest cases, in which we want to include one letter of context at each position in the input string. Introducing one letter of left context means deterministically transforming a length n input string of the form

$$a_1\, a_2\, \cdots\, a_n$$

into the string (also of length n)

$$\langle \text{-}, a_1 \rangle\, \langle a_1, a_2 \rangle\, \cdots\, \langle a_{n-1}, a_n \rangle$$

where $\langle \text{-} \rangle$ is a padding symbol introduced to represent the imaginary left context at the left edge of the original string. If the original alphabet was Σ, the new alphabet is $(\Sigma \cup \{\text{-}\}) \times \Sigma$.

Mappings of this sort can be computed by strictly k-local same-length transducers. A concrete example of a strictly 2-local transducer with input alphabet $\Sigma = \{a, b\}$ is shown in Figure 3.3.

Adding one letter of right context means replacing an input string of the form

$$a_1\, a_2\, \cdots\, a_n$$

by the string

$$\langle a_1, a_2 \rangle\, \langle a_2, a_3 \rangle\, \cdots\, \langle a_n, \text{-} \rangle$$

where $\langle \text{-} \rangle$ represents the imaginary context beyond the right edge of the original string. If the original alphabet was Σ, the new alphabet is now $\Sigma \times (\Sigma \cup \{\text{-}\})$.

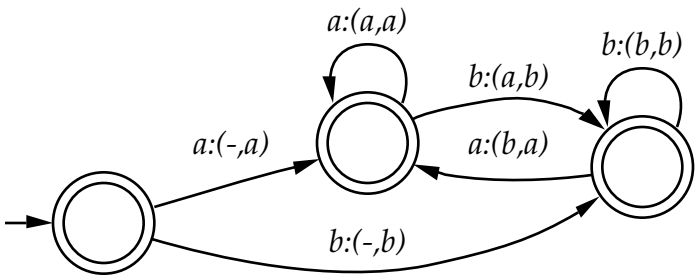

Figure 3.3: A local sequential transducer that introduces left context.

This mapping can be computed by an augmented version of a strictly k-local same-length transducer which adds a subsequential output function that associates final states with final output strings. Figure 3.4 shows a concrete example of such a transducer over the input alphabet $\Sigma = \{a, b\}$. Two states have a non-empty final output, which is displayed inside the circles representing the states.

Both transducers define bijections between plain and augmented strings; their inverses are very fine morphisms that forget the added context. These techniques generalize straightforwardly to larger amounts of context, and they can be combined to produce the kinds of windows discussed earlier and illustrated in Figure 3.2.

Note especially that for present purposes the elements of the new alphabet are viewed as unanalyzable units. Any practical classifier would still be at liberty to analyze the internal structure of these symbols before deciding which output symbol to produce. Reducing the learning problem to inference of (very) fine morphisms of free monoids abstracts away from most details of local transducers

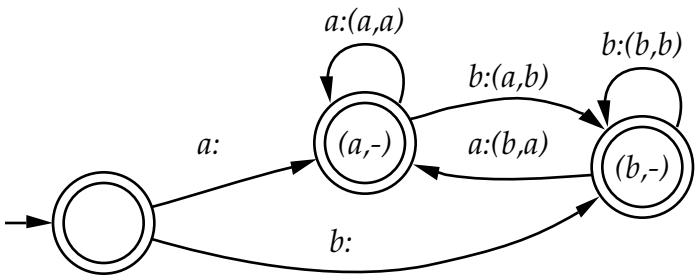

Figure 3.4: A local subsequential transducer that introduces right context.

and allows us to concentrate on morphisms and their associated learning problems. One encouraging aspect of this reduction is that the hypothesis space has become finite: for finite alphabets Σ and Γ there are only finitely many (very) fine morphisms from Σ^* to Γ^*. For the more general case of fine morphisms, the hypothesis space H is the set of all functions from Σ to $\Gamma \cup \{\epsilon\}$, for which $\ln |H| = |\Sigma| \ln(|\Gamma| + 1)$. Because of the way sample complexity in the PAC learning model [Valiant, 1984] is defined in terms of $\ln |H|$ [Kearns and Vazirani, 1994, sec. 3], this means that the problem of learning fine morphisms has polynomial sample complexity. If there were an efficient algorithm that outputs a consistent hypothesis for a given sample, fine morphisms might be efficiently PAC learnable. Unfortunately, as the next section will show, this is almost certainly not the case.

3.5 Learning Tasks and their Complexity

This section formulates a number of formal problems related to learning fine morphisms and very fine morphisms. At the core of these problems are the question of whether there is a morphisms which is consistent with a set of data and the task of finding a morphism which optimizes a fixed loss function. Most of these problems turn out to be quite hard. This is not very surprising in light of other studies of the complexity of natural language processing and machine learning tasks [for example Hyafil and Rivest, 1976; Barton et al., 1987; Brew, 1992; Knight, 1999; Casacuberta and de la Higuera, 2000; de la Higuera and Casacuberta, 2000].

The previous section showed that any strictly local transduction corresponding to the approach outlined in Section 3.3 can be viewed as a morphism of free monoids that gets applied to deterministically transformed input strings (that may incorporate a fixed amount of local context). The transformations that accumulate local context are generally held constant, in other words, a maximum window size is selected initially and not subject to change during the learning phase. So we can view the learning task abstractly as inductive inference of morphisms of free monoids.

The domain of the morphisms we want to infer is some fixed finite alphabet Σ. This may be the result of a preprocessing transformation and correspond to n-tuples of letters, but the present formulation of the problem allows us to abstract away from such details. We therefore speak abstractly of *input symbols* when referring to the elements of Σ.

We focus on the two kinds of morphisms singled out above. As noted there, inference of very fine morphisms corresponds to working with aligned data; and

inference of fine morphisms corresponds to discovering alignments, subject to the length constraints discussed earlier.

The model of learning used here is not limit-identification [Gold, 1967] as for OSTIA and other grammatical inference procedures. Instead we focus initially on the problem of finding consistent hypotheses, which is relevant for PAC learning, and more generally on empirical risk minimization. Although empirical risk minimization has drawbacks [Minka, 2000], particularly if the training data are not representative of the distribution of future data, it is used by most practical approaches to learning deterministic letter-to-sound rules.

There are three main notions of empirical risk, corresponding to the three kinds of loss functions defined in Section 2.3. As noted there, prediction error is only applicable with aligned data, whereas symbol error and string error are universally applicable. So we obtain the following five empirical risk minimization problems:

1. Given a finite aligned dictionary $D \subseteq \bigcup_{n\in\mathbb{N}} \Sigma^n \times \Gamma^n$ find a very fine morphism with minimal prediction error (rate) on D.

2. Given a finite dictionary $D \subseteq \Sigma^* \times \Gamma^*$ find a very fine morphism with minimal string error (rate) on D.

3. Given a finite dictionary $D \subseteq \Sigma^* \times \Gamma^*$ find a very fine morphism with minimal symbol error (rate) on D.

4. Given a finite dictionary $D \subseteq \Sigma^* \times \Gamma^*$ find a fine morphism with minimal string error (rate) on D.

5. Given a finite dictionary $D \subseteq \Sigma^* \times \Gamma^*$ find a fine morphism with minimal symbol error (rate) on D.

Note that there is no problem asking for a fine morphism that minimizes prediction error rate on an aligned dictionary, since such a morphism would necessarily be very fine (except in some trivial and unimportant cases).

The first problem of finding a very fine morphism that minimizes prediction error rate is very easy to solve. It was formulated in such a way as to make it appear similar to the remaining four problems, but this superficial similarity soon disappears: observe that the loss function is defined in terms of individual symbol occurrences, not in terms of whole strings. So there is no need to distinguish the elements of D. In fact, we can assume w. l. o. g. that the cardinality of D is one, since the elements of D can be concatenated without affecting the total or average loss. For each element σ of Σ occurring in D do the following: keep count of how often it is aligned in D with a symbol from Γ (this only requires counters, and can therefore be done in logarithmic space); then determine the element γ of Γ with the highest count (breaking any ties arbitrarily), and set $f(\sigma) \leftarrow \gamma$. Finally, return f. Observe that f is a partial function $\Sigma \to \Gamma$ that can be extended to a very fine partial morphism $f^* : \Sigma^* \to \Gamma^*$, since $\Gamma \subseteq \Gamma^*$ by minor abuse of notation. Correctness is obvious, since choosing majority labels minimizes prediction error. The overall procedure requires logarithmic space, and time linear in the total length of all strings in D.

In reality, the problem is somewhat harder, since only a small fraction of Σ may be represented in D, especially if the elements of Σ have internal structure due to a preprocessing step. Practical approaches need to extend the morphism so that it also applies to elements of Σ not encountered in D, but this is generally only possible if the elements of Σ can be decomposed and analyzed.

The remaining four problems are intuitively much harder. Since symbol error and string error are not formulated in terms of individual symbols but in terms of

whole string pairs, minimizing either loss function appears to be a fairly intricate combinatorial optimization problem, involving tradeoffs of the sort seen in some of the examples from Section 2.4.

3.5.1 Exact Solutions

In this subsection we focus on the optimization problems corresponding to empirical risk minimization. We had formulated two versions of the problem of learning a fine morphism, namely finding a fine morphism that minimizes the string error rate, resp. the symbol error rate, on the training data D. Since we are interested in exact solutions, we could also maximize string accuracy, resp. symbol accuracy.

The problem of finding a function $f : \Sigma \to \Gamma \cup \{\epsilon\}$ such that the empirical risk of f^* is minimal is fundamentally a combinatorial optimization problem. Like all such problems it can be stated formally in different ways [Papadimitriou and Steiglitz, 1998, p. 345f.]: the optimization version asks for the optimal f for a given set of samples $D \subseteq \Sigma^* \times \Gamma^*$; the evaluation version asks for the total loss incurred on D by the optimal f^*; and the decision version asks whether there exists an f^* such that the total loss incurred by it on D is less than or equal to a given budget k. A solution to the optimization version could be used to construct an answer to the evaluation version, which in turn could be used to solve the decision version. Contrapositively, if the decision version is hard to solve, so are the other two versions.

The optimization problem corresponding to string error minimization has the following associated decision problem. An analogous problem MIN-VFMC for very fine morphisms could be defined similarly. The decision problem is stated

in a format similar to the one used by Garey and Johnson [1979] and asks whether or not a solution exists within a given budget k:

Problem 3.1 (Fine morphism minimization – MIN-FMC)

Instance: A finite sequence $D = \langle s_1, \ldots, s_n \rangle$ where each $s_i \in \Sigma^* \times \Gamma^*$ for $1 \leq i \leq n$; and a natural number k with $k \leq n$.

Question: Does there exist a fine morphism which is inconsistent with at most k elements of D, i. e., is there a function $f : \Sigma \to \Gamma \cup \{\epsilon\}$ and a length $m = n - k$ unordered subsequence $\langle t_1, \ldots, t_m \rangle$ of D such that $t_i \in \#(f^*)$ for all $1 \leq i \leq m$?

Observe that the string error rate of a particular morphism on a given data set D is zero if and only if its symbol error rate on D is zero. This holds if symbol error rate is defined as Levenshtein distance (see page 121), or under any other assignment of insertion, deletion and substitution costs, provided all matches (substitutions of identical symbols) have zero cost and all other edit operations have strictly positive costs. In other words, there is a common subproblem of the decision version of the fine morphism optimization problem which is independent of the loss function used: the restricted decision version (with budget $k = 0$) asks whether there exists an f^* such that the total loss incurred by it on D is identically zero. We call this the *consistency* problem. Obviously, if the decision version of an optimization problem can be solved efficiently, so can the consistency problem.

An answer to the questions posed in the following two variants of the consistency problem would tell us whether appropriate morphisms exists that reduce the empirical loss to zero.

Problem 3.2 (Very fine morphism consistency – VFMC)

Instance: A finite (multi)set $D \subseteq \Sigma^* \times \Gamma^*$.

Question: Does there exist a very fine morphism consistent with all elements of D, i. e., is there a function $f : \Sigma \to \Gamma$ such that $D \subseteq \#(f^*)$?

Problem 3.3 (Fine morphism consistency – FMC)

Instance: A finite (multi)set $D \subseteq \Sigma^* \times \Gamma^*$.

Question: Does there exist a fine morphism consistent with all elements of D, i. e., is there a function $f : \Sigma \to \Gamma \cup \{\epsilon\}$ such that $D \subseteq \#(f^*)$?

Clearly FMC is a special case of MIN-FMC. If there were a polynomial time algorithm for solving the latter, then FMC could be solved in polynomial time, by calling the hypothetical MIN-FMC algorithm with the budget k set to zero.

Of the two consistency problems formulated here, FMC is intuitively more difficult than VFMC, since one has to decide which input symbols are mapped to the empty string, or equivalently, how the output strings should be aligned relative to the inputs. This issue does not arise with VFMC since only strings of equal length need to be considered (if D contains a pair of strings with different lengths, then no very fine morphism can be consistent with D). The remainder of this subsection makes these intuitions about the difficulty of FMC more precise.

The size of an instance of one of these problems is the total length of all strings in the dictionary D:

Definition 3.7 (Dictionary size). Define the size $\|D\|$ of a dictionary $D \subseteq \Sigma^* \times \Gamma^*$ as

$$\|D\| = \sum_{\langle x,y\rangle \in D} |x| + |y|$$

where $|w|$ is the length of string w.

We will prove that problem FMC is complete for the complexity class **NP** [see for example Garey and Johnson 1979 for a definition]. Membership of FMC in **NP** can be established straightforwardly:

Theorem 3.1. *Problem* FMC *has succinct certificates that can be verified in polynomial time.*

Proof. A certificate for FMC is a partial function $f : \Sigma \to \Gamma \cup \{\epsilon\}$, which can be represented in space linear in $\|D\|$ (because w. l. o. g. f only mentions elements of Σ that occur in D). Verification amounts to applying f^* to each input string in D and comparing the results to the corresponding reference output contained in D. The verification procedure, shown in Figure 3.5, runs in time linear in $\|D\|$. □

On the other hand, problem VFMC for very fine morphisms can be solved efficiently in linear time and space by the following procedure: iterate over D, and any time a previously unseen input symbol σ is encountered, set $f(\sigma) \leftarrow \gamma$ where γ is the output symbol aligned with σ; run the verification algorithm from Figure 3.5 on D and f and return its answer.

NP-hardness of FMC is established by a reduction from 3SAT, the decision problem asking whether there is a satisfying truth assignment for a set of disjunctive clauses with at most three literals each. We first define the construction and then prove that it correctly preserves the structure of 3SAT.

Definition 3.8 (Boolean variable gadget). For any Boolean variable v, the set $\mathscr{V}(v)$ contains the following pairs (a_v and b_v are new symbols dependent on v):

$$\langle a_v\, v\, \overline{v}\, b_v, FTF \rangle$$

$$\langle a_v\, b_v, F \rangle$$

```
1:  // Input: instance D of FMC, certificate f
2:  for each ⟨x, y⟩ ∈ D do
3:      a_1 ··· a_n ← x
4:      b_1 ··· b_m ← y
5:      j ← 1
6:      for i ← 1 to n do
7:          if f(a_i) ≠ ϵ then
8:              if j > m then
9:                  return false
10:             else if f(a_i) ≠ b_j then
11:                 return false
12:             else // f(a_i) matches b_j
13:                 j ← j + 1
14:             end if
15:         end if
16:     end for
17:     if j ≠ m + 1 then
18:         return false
19:     end if
20: end for
21: return true
```

Figure 3.5: Certificate verification algorithm for FMC.

The Boolean variable gadget encodes the fact that a variable occurring in a 3CNF formula can take on only the values T (true) and F (false). It consists of two entries that will become part of a larger dictionary constructed by the reduction. A fine morphism that is consistent with that dictionary can map the symbols v and $\overline{g}$ only to T or F, and maps v to T iff it maps $\overline{v}$ to F. To see why this is the case, consider the first tuple. It is clear that exactly one of the input symbols must be mapped to the empty string $\epsilon$by a consistent fine morphism, since the output string FTF is one symbol shorter. If either v or $\overline{v}$ were mapped to ϵ, then both a_v and b_v would get mapped to F. But such a mapping would be inconsistent with the second tuple, since it would wrongly predict an output string of FF instead of F. Therefore the only choice is to map either a_v to F and b_v to ϵ, or the other way round. To ensure that a_v and b_v do not appear in any gadgets for other, unrelated variables, now concrete symbols a_v and b_v have to be chosen for each variable v.

Definition 3.9 (3SAT clause gadget). For any 3SAT clause C_i of the form $(l_{i1} \vee l_{i2} \vee l_{i3})$ (where each l_{ij} is a literal of the form v or $\overline{v}$) the set $\mathscr{C}(C_i)$ contains the following pairs (all of c_{ij}, d_{ij}, e_i and f_i are new symbols dependent on i):

$$\langle c_{i1}\, l_{i1}\, d_{i1}, FT\rangle$$

$$\langle c_{i2}\, l_{i2}\, d_{i2}, FT\rangle$$

$$\langle c_{i3}\, l_{i3}\, d_{i3}, FT\rangle$$

$$\langle d_{i1}\, d_{i2}\, d_{i3}\, e_i\, f_i, TT\rangle$$

The 3SAT clause gadget represents the constraint that in a clause C of the form $(l_1 \vee l_2 \vee l_3)$ at least one literal l_j must be true for the overall formula to be satisfied. A literal is a plain or negated variable, which can also be viewed as an input

symbol in the dictionary built by the reduction. Each literal l_j is represented as an input string $c_j\, l_j\, d_j$ that must be correctly mapped to the output string FT by a consistent fine morphism. The Boolean variable gadgets described previously ensure that a plain or negated variable like l_j can only be mapped to T or F by a consistent morphism. But this means that d_j can only be mapped to T or ϵ. The last pair, however, is $\langle d_1\, d_2\, d_3\, e\, f,\ TT\rangle$, so at most two symbols among d_1, d_2 and d_3 get mapped to T by a consistent fine morphism. In other words, at least one d_j must map to the empty string ϵ, which means that the corresponding l_j must map to T, thus making the clause C true. As in the Boolean variable gadget, the symbols other than those corresponding to literals or variables of the original formula must be unique for each clause, so that there are no additional constrains between clauses not present in the 3CNF formula.

Definition 3.10 (Reduction from 3SAT). Given an instance $\phi = \bigwedge_{i=1}^{n} C_i$ of 3SAT, define $\mathscr{D}(\phi)$ as the collection $\bigcup_{i=1}^{n} \mathscr{C}(C_i) \ \cup\ \bigcup\{\mathscr{V}(v) \mid \text{variable } v \text{ occurs in } \phi\}$.

For example, if $\phi = (x \vee \overline{x} \vee y)$, the dictionary $\mathscr{D}(\phi)$ defined by this reduction contains the following entries:

$$\begin{array}{cc} \langle ax\overline{x}b, FTF\rangle & \langle exf, FT\rangle \\ \langle ab, F\rangle & \langle g\overline{x}h, FT\rangle \\ \langle cy\overline{y}d, FTF\rangle & \langle iyj, FT\rangle \\ \langle cd, F\rangle & \langle fhjkl, TT\rangle \end{array}$$

Theorem 3.2. *The reduction from* 3SAT *to* FMC *can be computed in logarithmic space and creates an instance whose size is polynomial in the size of the original instance.*

Proof. The reduction $\mathscr{D}$, which can be made to run in linear time, builds a collection $\mathscr{D}(\phi)$ with the following properties: let m be the number of distinct variables

of ϕ (so $m \leq 3\,n$); then $\|\mathscr{D}(\phi)\| = 10\,m + 22\,n \leq 52\,n$, $|\mathscr{D}(\phi)| = 2\,m + 4\,n \leq 10\,n$, $|\Sigma| = 4\,m + 8\,n \leq 20\,n$, and $|\Gamma| = 2$. Only counters need to be stored for computing the reduction (in order to keep track of clauses and variables represented by integers), which requires logarithmic space. □

Theorem 3.3. *Problem* FMC *is* ***NP****-hard.*

Proof. We show that $\phi = \bigwedge_{i=1}^{n} C_i$ is satisfiable iff there exists a fine morphism f^* consistent with $\mathscr{D}(\phi)$. It will be convenient to let V denote the set of distinct variables of ϕ.

($\Rightarrow$) Assume that ϕ is satisfiable, i. e., there exists a satisfying assignment $\tau : V \to \{T, F\}$. Incrementally define a fine morphism f^* consistent with $\mathscr{D}(\phi)$ as follows: for all $v \in V$, let $f(v) = \tau(v)$ and $f(\overline{v}) = \overline{\tau(v)}$. If $\tau(v) = T$, let $f(a_v) = F$ and $f(b_v) = \epsilon$, which makes f^* consistent with $\mathscr{V}(v)$; otherwise, if $\tau(v) = F$, let $f(a_v) = \epsilon$ and $f(b_v) = F$ to make f^* consistent with $\mathscr{V}(v)$. In either case f^* can be made consistent with $\mathscr{V}(v)$, and because a_v and b_v do not occur outside the gadget for v, f^* can be made consistent with all variable gadgets.

The fact that τ is a satisfying assignment means that in each clause C_i at least one literal is made true by τ. So f will map at most two d_{ij} in $\mathscr{C}(C_i)$ to T, and therefore the definition of f^* can always be extended to make it consistent with the fourth pair in $\mathscr{C}(C_i)$ and hence consistent with the entire clause gadget for C_i. Since all symbols in a gadget other than literals of ϕ occur only in that gadget, the definition of f^* can be extended to make it consistent with all gadgets and therefore consistent with $\mathscr{D}(\phi)$. Hence there exists a consistent fine morphism f^* constructible from τ.

($\Leftarrow$) Conversely, assume that a fine morphism g consistent with $\mathscr{D}(\phi)$ exists. We show that $g|_V$, i. e. g restricted to the variables of ϕ, is a satisfying truth assignment for ϕ. Obviously g being consistent with $\mathscr{D}(\phi)$ means that g is consistent with all variable gadgets and all clause gadgets.

Pick any variable gadget $\mathscr{V}(v)$. Then, because of the second pair in $\mathscr{V}(v)$, g must map exactly one of a_v and b_v to F: if $g(a_v) = F$ then $g(b_v) = \epsilon$, and for the first pair $g(v) = T$ and $g(\overline{v}) = F$; otherwise if $g(b_v) = F$, then $g(a_v) = \epsilon$, $g(v) = F$, and $g(\overline{v}) = T$. Note in particular that $g|_V(v) \in \{T, F\}$, so $g|_V$ is formally a truth assignment.

Now pick any clause gadget $\mathscr{C}(C_i)$ and suppose that g maps no l_{ij} in $\mathscr{C}(C_i)$ to T. Then all d_{ij} in $\mathscr{C}(C_i)$ are mapped to T because of the first three pairs in that clause gadget. But this would make g inconsistent with the fourth pair, contradicting the assumption that g is consistent with all clause gadgets. So g must map at least one l_{ij} in $\mathscr{C}(C_i)$ to T, which means that $g|_V$ makes the clause C_i true, and is therefore a satisfying truth assignment for ϕ. □

The preceding three theorems together imply that the consistency problem FMC is **NP**-complete. The existence of efficient deterministic algorithms for solving FMC is therefore unlikely. Since FMC is a subproblem of empirical risk minimization, the decision version MIN-FMC of this optimization problem is also **NP**-hard. Showing that the decision versions of the optimization problems are in **NP** is straightforward, but requires separate proofs depending on which loss function is used. If the loss function is string error (MIN-FMC) only a few minor modifications to the certificate verification algorithm in Figure 3.5 are required, which now has to aggregate the number of mistakes and compare it to the budget k. For loss based on edit distance, using the standard dynamic programming algorithm to

compute edit distances [Wagner and Fischer, 1974] ensures that certificates can be verified in polynomial time. In other words, the decision versions of all empirical risk minimization problems for fine morphisms are **NP**-complete, because they contain the **NP**-complete consistency problem FMC as a subproblem.

The optimization version of empirical risk minimization, which asks for a function f such that the associated fine morphism f^* minimizes string error on D, is a function problem [Papadimitriou, 1994, sec. 10.3]. As such it is **FNP**-hard because the decision version MIN-FMC is **NP**-hard, but the optimization version is probably not a member of **FNP**, since that would require concise certificates which can attest to the optimality of f. We conjecture that the optimization version of MIN-FMC is in fact $\mathbf{FP}^{\mathbf{NP}}$-complete, just like the traveling salesperson problem (TSP) [Papadimitriou, 1994, pp. 418ff.].

3.5.2 Approximate Solutions

The remaining problems concern inference of very fine morphisms. They are treated separately, because it is possible to show stronger results than in the preceding subsection about the existence of nonexistence of approximation algorithms.

It was shown earlier that the simple consistency problem VFMC can be solved very efficiently. An interesting related problem is MAX-VFMC, which asks for a very fine morphism that maximizes string accuracy. Let us define MAX-VFMC formally:

Problem 3.4 (Very fine morphism maximization – MAX-VFMC)

Instance: A finite sequence $D = \langle s_1, \ldots, s_n \rangle$ where each $s_i \in \bigcup_{j \in \mathbb{N}} \Sigma^j \times \Gamma^j$ for

$1 \leq i \leq n$; and a natural number k with $k \leq n$.

Question: Does there exist a very fine morphism consistent with at least k elements of D, i. e., is there a function $f : \Sigma \to \Gamma$ and a length k unordered subsequence $\langle t_1, \ldots, t_k \rangle$ of D such that $t_i \in \#(f^*)$ for all $1 \leq i \leq k$?

It will become clear that MAX-VFMC is actually a hard problem, which happens to have an efficiently solvable subproblem, namely VFMC. The situation is vaguely reminiscent of the case of 2SAT. The decision version of 2SAT is a restricted version of the satisfiability problem in which each clause consists of at most two disjunctive literals. Although 2SAT can be decided efficiently (in nondeterministic log-space [Papadimitriou, 1994, pp. 184f.] and therefore in polynomial time), the corresponding optimization problem MAX-2SAT is **NP**-complete [Garey and Johnson, 1979; Papadimitriou, 1994; Ausiello et al., 1999].

We establish a slightly stronger result about MAX-VFMC by showing that it is hard to approximate in polynomial time (we follow the terminology of Ausiello et al. [1999]):

Theorem 3.4. *Problem* MAX-VFMC *is* ***APX****-hard.*

Proof. Show this by exhibiting an **AP**-reduction from an **APX**-complete problem. It suffices to show that MAX-k-CSP is L-reducible [Papadimitriou, 1994, pp. 309ff.] to MAX-VFMC. MAX-k-CSP is a constraint satisfaction problem with conjunctive constraints containing at most k literals [see for example Khanna et al., 1997b; Ausiello et al., 1999].

Given an instance $C = \langle (l_{11} \wedge \cdots \wedge l_{1k}), \ldots, (l_{n1} \wedge \cdots \wedge l_{nk}) \rangle$ of MAX-k-CSP, construct an instance of MAX-VFMC by mapping the ith constraint $(l_{i1} \wedge \cdots \wedge l_{ik})$ to the pair $\langle l_{i1}\overline{l_{i1}} \cdots l_{ik}\overline{l_{ik}}, TF \cdots TF \rangle$ to form D (if a literal l is negative, i. e. is of the

form $\overline{v}$, then $\bar{l}$ is simply v). So Σ consist of the negated and unnegated variables of C, and $\Gamma = \{T, F\}$.

This construction ensures that there is a truth assignment τ that makes exactly m constraints of C true iff there exists a very fine morphism f^* which is consistent with exactly m elements of D. One can construct f from τ (and vice versa) by letting $f(v) = \tau(v)$ and $f(\overline{v}) = \overline{\tau(v)}$, where v is a variable occurring in C. □

This means that there is no polynomial-time approximation scheme (PTAS) for MAX-VFMC (unless **P** = **NP**). The best we can hope for would be to show that MAX-VFMC belongs to **APX**, which would mean that there is a polynomial-time algorithm with an approximation ratio bounded by some constant $r > 1$. While this seems likely, we have no definitive answer either way, and so this is left as an open problem.

Instead of maximizing string accuracy we can minimize string error rate. As discussed in Section 2.5, these two problems are generally distinct when approximation algorithms are involved. The minimization problem can be formulated like this:

Problem 3.5 (Very fine morphism minimization – MIN-VFMC)

Instance: A finite sequence $D = \langle s_1, \ldots, s_n \rangle$ where each $s_i \in \bigcup_{j \in \mathbb{N}} \Sigma^j \times \Gamma^j$ for $1 \leq i \leq n$; and a natural number k' with $k' \leq n$.

Question: Does there exist a very fine morphism inconsistent with at most k' elements of D , i. e., is there a function $f : \Sigma \to \Gamma$ and a length $k = n - k'$ unordered subsequence $\langle t_1, \ldots, t_k \rangle$ of D such that $t_i \in \#(f^*)$ for all $1 \leq i \leq k$?

It is clear from Section 3.5.1 that VFMC is no harder than 2SAT; in fact, both problems are in **P**. The point of the following reduction is that it applies both to

1: // Input: instance D of VFMC
2: $V \leftarrow \{\}$
3: $\phi \leftarrow \langle\,\rangle$
4: **for each** $\sigma \in \Sigma$ **do**
5: **for each** $\gamma \in \Gamma$ **do**
6: $V \leftarrow V \cup \{x_{\sigma,\gamma}\}$
7: **for each** $\gamma' \in \Gamma$ **do**
8: **if** $\gamma \neq \gamma'$ **then**
9: **for** $i \leftarrow 1$ **to** $\text{len}(D)$ **do**
10: $\phi \leftarrow \phi \cdot \langle(\overline{x_{\sigma,\gamma}} \vee \overline{x_{\sigma,\gamma'}})\rangle$
11: **end for**
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: **for** $i \leftarrow 1$ **to** $\text{len}(D)$ **do**
17: $\langle u, w\rangle \leftarrow D[i]$
18: $V \leftarrow V \cup \{y_i\}$
19: $\phi \leftarrow \phi \cdot \langle(y_i)\rangle$
20: **for** $j \leftarrow 1$ **to** $\text{len}(u)$ **do**
21: $\phi \leftarrow \phi \cdot \langle(x_{u[j],w[j]} \vee \overline{y_i})\rangle$
22: **end for**
23: **end for**
24: **return** $\langle V, \phi\rangle$

Figure 3.6: Reduction from MIN-VFMC to MIN-2SAT.

the simple decision problems, as well as the decision versions of the full optimization problems. This will establish that MIN-VFMC is MIN-2SAT-easy.

Definition 3.11 (Reduction of (MIN-)VFMC to (MIN-)2SAT). Given an instance D of VFMC with alphabets Σ and Γ, construct a set of variables V and a sequence ϕ of 2-CNF clauses using the algorithm in Figure 3.6.

It is obvious from this statement of the reduction that the computation can be carried out in polynomial time. It creates new variables of the form $x_{\sigma,\gamma}$ which encode the mapping performed by the very fine morphism: there exists a very fine morphism f^* consistent with D such that f is defined for a and $f(a) = b$ iff a satisfying assignment τ for ϕ can be found such that $\tau(x_{a,b}) = T$ and $\tau(x_{a,\gamma}) = F$ for all $\gamma \neq b$. The constraints involving two x variables (line 10 of Figure 3.6) ensure that it is impossible for $x_{a,b}$ and $x_{a,c}$ ($b \neq c$) to both be true. It is however possible that $x_{a,\gamma}$ is false for all $\gamma \in \Gamma$, in which case f is not defined for a (to ensure that f is total one would have to add a clause $\bigvee_{\gamma\in\Gamma} x_{a,\gamma}$ with the result that ϕ would not necessarily be a 2-CNF formula but rather a $|\Gamma|$-CNF formula).

A variable of the form y_i encodes the requirement that the very fine morphism f^* be consistent with the ith sample in D (see line 19 of Figure 3.6). Suppose $D[i]$, the ith element of D, is $\langle a_1 \cdots a_n, b_1 \cdots b_n \rangle$. For all j ($1 \leq j \leq n$), if $f(a_j) = b_j$ then it is possible to find a truth assignment that makes x_{a_j,b_j} true and so the clause $(x_{a_j,b_j} \vee \overline{y_i})$ (line 21) is satisfied no matter what truth value is assigned to y_i. Therefore if f^* is consistent with $D[i]$ it is possible to set y_i to true to satisfy the clause involving only y_i. Conversely assume that there is a satisfying assignment τ for ϕ. Then $\tau(y_i) = T$ because of the clause consisting only of y_i (line 19), and so each $x_{\sigma,\gamma}$ occurring in a clause with $\overline{y_i}$ (line 21) must be true, which means that a very fine morphism f^* can be found by setting $f(\sigma) = \gamma$ iff $\tau(x_{\sigma,\gamma}) = T$ such that f^* is consistent with $D[i]$.

The discussion up to this point establishes that VFMC reduces to 2SAT, which we already knew.

Now suppose f^* is inconsistent with $D[i] = \langle a_1 \cdots a_n, b_1 \cdots b_n \rangle$. This means that there exists a symbol a_j ($1 \leq j \leq n$) for which $f(a_j) \neq b_j$. Assuming all clauses of the form $(\overline{x_{a,b}} \vee \overline{x_{a,c}})$ in ϕ are satisfied (it is always possible to satisfy the subset

consisting of clauses of that form), this means that x_{a_j,b_j} must be false. In this case it is possible to set y_i to false, which makes all clauses $(x_{a_j,b_j} \vee \overline{y_i})$ true ($1 \leq j \leq n$). In other words, it is always possible to find a truth assignment for which at most one clause involving y_i is not satisfied. We have just shown that if an optimal very fine morphism (one that is inconsistent with a minimal number of samples in D) is inconsistent with m elements of D, then an optimal truth assignment for ϕ leaves at most m clauses unsatisfied.

Going in the opposite direction, suppose that τ is a truth assignment for ϕ that leaves m clauses unsatisfied. We can assume w. l. o. g. that any τ satisfies all clauses of the form $(\overline{x_{a,b}} \vee \overline{x_{a,c}})$, since if it did not we could change it to do so without increasing the number of unsatisfied clauses. Let $f(\sigma) = \gamma$ just in case $\tau(x_{\sigma,\gamma}) = T$. Because all clauses of the form $(\overline{x_{a,b}} \vee \overline{x_{a,c}})$ are satisfied, f is a partial function $\Sigma \to \Gamma$. As in the previous discussion the number of unsatisfied clauses in ϕ can be said w. l. o. g. to be equal to the number of variables y_i for which $\tau(y_i) = F$. It follows from the definition of the reduction that f^* will leave the same number of clauses unsatisfied.

We have just proved the following result:

Theorem 3.5. *Problem* MIN-VFMC *L-reduces to* MIN-2SAT.

The name MIN-2SAT follows the terminology of Khanna et al. [1997a]; the problem also goes by the name MIN-2CNF-DELETION. Polynomial-time algorithms with constant approximation ratios are not known for MIN-2SAT. At this point the best known polynomial-time algorithms have poly-logarithmic approximation bounds [Klein et al., 1997]. It is not clear that a modification of the approximation algorithm for MIN-2SAT is effective for solving practical instances of MIN-VFMC. A formulation of such an algorithm remains an open problem. A precise

characterization of the hardness of MIN-VFMC is another open problem [but see Khanna et al., 1997a].

Minimizing symbol error rate appears to be an altogether more intricate problem. If it can be expressed in terms of constraint satisfaction at all, the constraints involved are likely going to be considerably more complex than those encountered so far, since they would have to encode the fact that string edit distance itself is based on an (easy) optimization. We conjecture that the decision version of the optimization problem involving minimization of symbol error rate is also **NP**-complete. This is the only problem of the five empirical risk minimization problems introduced at the beginning of this section whose complexity has not been fully established.

All other problems involving minimization of string or symbol error rate have been shown to be quite hard. Efficient algorithms for exact solutions are not forthcoming, and the existence of good approximation algorithms seems also unlikely. In some cases we can turn to inefficient algorithms or efficient heuristics without any performance guarantees. For small problem instances, exhaustive search is an option, and greedy heuristics based on local search seem to work well when exhaustive search is too expensive. The traditional approach based on classifier learning can also be seen as a heuristic solution: in a preprocessing step an aligned dictionary is produced, which is then used as an instance of the only empirical risk minimization problem studied here that is known to be efficiently solvable, namely minimization of prediction error on an aligned dictionary.

3.6 Conclusion

This chapter discussed machine learning of deterministic transducers. The grammatical inference algorithm OSTIA has a very inclusive concept class, but suffers from many practical drawbacks. Traditional approaches to learning letter-to-sound rules can be seen as inference procedures for same-length local transducers. They have many advantages compared with OSTIA, but are restricted to same-length (aligned) data.

We showed that the traditional approaches are in a certain specific sense equivalent to learning alphabetic substitutions using prediction error as loss. All formal problems associated with those approaches have efficient solutions and reasonable sample complexity, but the problem does not fall into the classical PAC learning paradigm, since it generally requires agnostic learners. We believe that efficient agnostic PAC learnability [Kearns et al., 1992] could be demonstrated.

By contrast, working with loss functions other than prediction error and/or asking the learner to automatically discover alignments leads to difficult problems. Automatically discovering alignments has been conceptualized as learning fine morphisms (alphabetic substitutions that may erase symbols), but the formal question whether consistent hypotheses exist cannot be answered efficiently, which means that neither PAC learning nor empirical risk minimization can be made efficient. Agnostic learning of very fine morphisms under global loss functions (not prediction error) is also difficult, but potentially amenable to approximations.

One can define a third class of morphisms of free monoids that extends the very fine morphisms in another direction. These morphisms are of the form $f^* : \Sigma^* \to \Gamma^*$ where f is a function $\Sigma \to \Gamma \cup \Gamma^2$, i. e., each input symbols is mapped to

one or two output symbols. This would be useful for automatically discovering multi-phoneme symbols [Sejnowski, 1988; Sproat, 2001]. The formal properties of the learning tasks associated with this new class of morphisms appear to be very similar to those of the fine morphisms. In particular, we suspect that the analogous consistency problem is also **NP**-complete.

To our knowledge, a formalization of the learning problems for the specific application of letter-to-sound rules has not been attempted before. Without the background of the present chapter, the traditional approaches based on classifier learning may have seemed like ad hoc solutions, and while they are not always very elegant, they have managed to concentrate on tractable and well understood problems. In fact, the results from this chapter could even be viewed as arguments in favor of the traditional approaches, since there are no obvious efficient alternatives, and in some cases there probably never will be. Our discussion also revealed a number of interesting open problems, which may yet lead to efficient approximation algorithms. The most interesting problem is MIN-FMC using symbol error as loss, which would be able to automatically discover alignments and which employs a reasonable loss function. While MIN-FMC is difficult to solve exactly, it would be worth while to investigate efficient approximation algorithms and/or heuristics.

CHAPTER 4

LEARNING MEMORYLESS STOCHASTIC TRANSDUCERS

4.1 Introduction

This chapter discusses a very restrictive class of stochastic transductions, so-called *memoryless* transductions, which do not take any letter or phoneme context into account. Memoryless transductions are important because they are closely related to string edit distance as well as morphisms of free monoids. Several key algorithmic tasks related to working with memoryless transducers, most notably parameter estimation, were discussed by Ristad and Yianilos [1996, 1998]. The goal of this chapter is to explain, correct and extend their results, and also to introduce the algebraic framework used for the generalizations discussed in Chapter 5.

4.1.1 Stochastic Transducers

Stochastic transducers are instances of weighted finite transducers [Berstel and Reutenauer, 1988] where weights are interpreted as probabilities. Ordinary transducers realize rational relations, which can be thought of in terms of their characteristic functions $\Sigma^* \times \Gamma^* \to \{0,1\}$, where Σ and Γ are finite sets and whose codomain is discrete. If R is the characteristic function of a rational relation, then

$R(x, y) = 1$ is taken to mean that strings $x \in \Sigma^*$ and $y \in \Gamma^*$ are related. The deterministic transductions considered in Chapter 3 are special in the sense that for all letter strings $x \in \Sigma^*$ there is at most one phoneme string $y \in \Gamma^*$ such that $R(x, y) = 1$. In other words, they are essentially functions from Σ^* to Γ^*, which justifies the slightly different notation used earlier.

By contrast, stochastic transductions are best viewed as functions belonging to type $\Sigma^* \times \Gamma^* \to [0; 1]$, i. e. as having a continuous codomain. If P is a function of this type, then call it a *joint distribution* over Σ^* and Γ^* if

$$\sum_{\langle x,y\rangle \in \Sigma^* \times \Gamma^*} P(x, y) = 1.$$

Given a joint distribution P the value $P(x, y)$ is the probability of strings $x \in \Sigma^*$ and $y \in \Gamma^*$ being generated simultaneously, since P is a distribution over pairs of strings. The joint distribution can potentially model all characteristics of x and y.

Call P a *conditional distribution* over Σ^* provided

$$\forall y \in \Gamma^* \colon \sum_{x \in \Sigma^*} P(x, y) = 1.$$

As usual, the fact that P is a conditional distribution will be indicated by the notation $P(x \mid y)$ instead of $P(x, y)$. Given such a conditional distribution P the value $P(x \mid y)$ is the probability of string $x \in \Sigma^*$ being generated from string $y \in \Gamma^*$. Characteristics of y are not modeled by the conditional distribution P. The symmetric case where P is conditional on $x \in \Sigma^*$ instead of $y \in \Gamma^*$ can be defined analogously.

Stochastic transductions that model the relationship between strings of letters and strings of phonemes can be used for predicting a phoneme string corresponding to a given letter string. The overall approach and the roles that joint and conditional models play in it are introduced in the following subsection. It provides some of the background not only for the present chapter, but also for Chapter 5.

4.1.2 The "Noisy Channel" Metaphor

The deterministic transduction models from Section 3.3 share a deficiency whose effects are most visible for very simple models that use a small context window: the information about the output string comes exclusively from scanning the input string. Intrinsic patterns in the output language (the second projection of the transducer) are modeled only very indirectly. In the context of letter-to-sound conversion, take a word like ⟨committed⟩ /kəmɪtəd/ and assume a very simple deterministic transduction model that employs a one-letter window. Such a model is clearly too simple, as it cannot possibly map the orthographic form ⟨committed⟩ to its reference pronunciation. The best output – in the sense of minimizing phoneme error rate (see Section 2.3.3) – this model could ever produce is /kəmmɪttəd/, which is highly improbable as a phonemic form of a single English word. The only way to improve the deterministic transduction models considered so far is to use more input context for prediction. Using a sufficient amount of input context is certainly a good idea; however, one can also look for more direct ways of ensuring that the predicted output falls within the range of what is usual, customary and reasonable. Phonemic strings like /kəmmɪttəd/ are arguably unusual and should not be predicted if better alternatives are available.

The deterministic transduction model can be augmented straightforwardly as follows: instead of predicting a single transcription symbol for each input window the scanner examines, it predicts several weighted alternatives. In general, the output the transducer produces on a given input is no longer a single string, but a finite language (more precisely, if the alternatives have weights the output is a finite string-to-weight mapping). For the example ⟨committed⟩ it would be reasonable to assume that the one-letter input window ⟨m⟩ can be mapped either to /m/ or to the empty string, and similarly for ⟨t⟩. The output of the transducer would then be the set {/kəmmɪttəd/, /kəmɪttəd/, /kəmmɪtəd/, /kəmɪtəd/}. An additional subsequent component could then pick the most plausible of these alternatives, which would presumably be the reference pronunciation /kəmɪtəd/. In general, if the transducer produces a weighted set of alternatives, a subsequent component would *rescore* those alternatives on the basis of their existing weights and some intrinsic measure of goodness. The overall output string is the one with the "best" weight after rescoring. Rescoring has been used by a few authors, including Jiang et al. [1997] and Chotimongkol and Black [2000], and typically involves phonemic n-gram models, also known as n-phone models.

We write $T(s; l)$ for the score the letter-to-sound transducer assigns to the output string s on input l. In practice the rescoring approach often takes the following form:

$$s^\star = \operatorname*{argmax}_s \; T(s; l) \times R(s).$$

$R(s)$ is the score assigned to the phoneme string s by the rescoring component. We assume w. l. o. g. that selecting the overall output amounts to maximizing (rather than minimizing) the combined score. When an n-gram model is used for rescoring, then R is in fact a probability distribution over phoneme strings, though this

is not required in general. If the letter-to-sound transducer is based on a classifier that outputs a probability distribution over the set of classes instead of the single most likely class, then T too is a probability distribution (conditional on l) over phoneme strings. However, it should be clear that the rescoring approach is an ad hoc solution even if both T and R are probability distributions, since $s \mapsto T(s; l) \times R(s)$ is generally not a proper probability distribution.

A solution that has clear interpretation in terms of probabilities can be obtained by interpreting T not as a probability distribution over s conditional on l, but as probability distribution over l conditional on s. Then T can be reinterpreted as a model of a noisy channel, and R as a model of the source that generated the signal transmitted through the channel. A slight change of notation might be helpful: let P_{src} (replacing R) represent the source model and P_{chn} (replacing T) the source-conditional channel model. In other words, we want to maximize the value of $P_{\text{chn}}(l \mid s) \times P_{\text{src}}(s)$ across different phonemic strings s. By Bayes' Theorem, the above expression gives rise to a proper joint probability distribution, unlike the rescoring approach. This joint distribution corresponds to a generative process whereby first a phoneme string s is generated by sampling from the distribution P_{src}, and then a letter string l is generated by sampling from the distribution $l \mapsto P_{\text{chn}}(l \mid s)$. Using the terminology of "Information" Theory [Shannon, 1948], one could say that l is the distorted result of s being transmitted through a noisy channel. Assigning a pronunciation to a letter string l can then be seen as recovering ("decoding") the most plausible phoneme string s that gave rise to l. We shall assume for the moment that the most plausible phoneme string is given by

$$s^\star = \operatorname*{argmax}_s \; P_{\text{chn}}(l \mid s) \times P_{\text{src}}(s).$$

Call the joint distribution $\langle l, s\rangle \mapsto P_{\text{chn}}(l \mid s) \times P_{\text{src}}(s)$ over pairs of letter and phoneme strings P_2. When P_2 can be realized by a finite transducer, we call it a stochastic string-to-string transduction. Moreover:

$$\begin{aligned} s^\star &= \operatorname*{argmax}_s \; P_{\text{chn}}(l \mid s) \times P_{\text{src}}(s) \\ &= \operatorname*{argmax}_s \; P_2(l, s) \\ &= \operatorname*{argmax}_s \; P_2(l, s) \; \frac{1}{\sum_s P_2(l, s)} \\ &= \operatorname*{argmax}_s \; P_2(s \mid l). \end{aligned} \tag{4.1}$$

Finding the optimal $s^\star$ is often referred to as maximum a posteriori (MAP) decoding, i. e., locating a mode of the "posterior" distribution obtained from a "prior" density P_{src} and "likelihood function" P_{chn}. (This terminology is confusing, because prior and posterior densities are usually distributions over parameters.)

We focus on stochastic rational transductions $P_2(x, y)$ that do not impose restrictions on the lengths of x and y. Otherwise, say if $x = \langle x_1, \ldots, x_n\rangle$ and $y = \langle y_1, \ldots, y_n\rangle$ always had equal length n, it would be easy to define a joint model P_2 over a pair of strings

$$\langle\langle x_1, \ldots, x_n\rangle, \langle y_1, \ldots, y_n\rangle\rangle$$

in terms of a simpler model P_{pairs} over a single sequence of symbol pairs:

$$P_2(\langle x_1, \ldots, x_n\rangle, \langle y_1, \ldots, y_n\rangle) = P_{\text{pairs}}(\langle\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle\rangle).$$

This is possible because the two representations are isomorphic under the function zip from Definition 3.2.

When working with aligned data, each letter string has by definition the same length as the corresponding phoneme string. While it is possible to apply stochastic transducers (Markov models) in such a setting [Rentzepopoulos et al., 1993; Rentzepopoulos and Kokkinakis, 1996; Minker, 1996, see for example], it also reintroduces the need to first obtain aligned data (see Section 3.3.2). Approaches that involve training Markov models on aligned data are therefore similar to classifier-based approaches, with which they have been grouped in Section 3.3. We would like to reduce or eliminate the restriction that requires aligned training data. While the stochastic approach allows that fairly straightforwardly, it raises additional problems for parameter estimation.

There are several other fundamental issues one has to address before adopting an approach based on stochastic transductions. The following problems extend the three Ferguson–Rabiner problems for Hidden Markov Model (HMMs) [Rabiner, 1989]:

1. First of all, the general structure of the model should be determined. All models considered here can be represented by stochastic rational transducers. In the remainder of this chapter, we describe Ristad and Yianilos's [1998] memoryless transducers, whose topology is fixed a priori. In Chapter 5 the state graphs of the transducers can take on any shape, but we will assume that once a particular topology has been chosen it remains fixed. In other words, we are not concerned with model induction, selection or merging [Stolcke and Omohundro, 1993].

2. We are especially interested in joint models P_2 (or conditional models P_{chn}) whose parameters can be estimated from unaligned data. The approaches we will discuss involve (more or less explicitly) a joint model P_3 on triples

of the form $\langle a, x, y \rangle$ where x is a letter sequence, y a phoneme sequence, and a an alignment of x and y. The joint probability $P(x, y)$ of a particular letter sequence x corresponding to a particular phoneme sequence y then arises by marginalization $P_2(x, y) = \sum_a P_3(a, x, y)$, involving a summation over all alignments of x and y if we want to evaluate the joint mass function P_2. How can this computation be done efficiently? We call this question **Problem 2**, because its solutions appear in Section 4.2 and Section 5.2, for memoryless and general stochastic transducers, respectively.

3. The models should be parameterized in such a way as to allow for efficient estimation of their parameters. This is generally difficult for conditional models, and so we will focus on joint models. The parameter estimation problem will be known as **Problem 3**, as it is discussed in Section 4.3 for memoryless transductions and in Section 5.3 for the general case.

4. The previous problem raises an immediate follow-up question: how do we turn a joint letter/sound model P_2 into a conditional model P_{chn}? Since a conditional probability can be expressed as a joint probability divided by a marginal probability like, for example, in equation (4.1), deriving a conditional model from a joint model usually requires first deriving a marginal model. This task is known as **Problem 4**, since it appears in Section 4.4 and Section 5.4.

5. Finally, approaches based on stochastic transduction must somehow address the decoding problem, which we refer to as **Problem 5**. For memoryless transductions this is discussed in Section 4.5, and the general case is the topic of Section 5.5.

4.1.3 Memoryless Stochastic Transducers

In the rest of this chapter we summarize and extend the approach laid out by Ristad and Yianilos [1996, 1998] and develop the terminology and background for Chapter 5. The key simplifying assumption of the present approach is that the stochastic transducer that realizes the joint distribution on letter strings and phoneme strings is completely memoryless, i. e., has a trivial one-state topology. In this respect it is very similar to transducers that realize morphisms of free monoids (see Section 3.4), but it is nondeterministic and allows the empty input string to be mapped to non-empty output strings. Since the transducer topology is fixed and trivial, we could in principle omit any references to transducer states; however, as the next chapter generalizes this approach to non-trivial topologies, the additional dependency on machine states should be kept in mind.

A joint letter/sound model P_2 assigns probability $P_2(x, y)$ to a letter string x and a phoneme string y. The fact that P_2 is memoryless means that no letter-context or phoneme-context is taken into account, which makes this special case similar to the one-letter local transducers of the sort discussed in Section 3.3.

Ristad and Yianilos [1998] formulate a joint model corresponding to the following memoryless generative process. Given two finite nonempty sets (alphabets) Σ and Γ, define a set $\Omega = ((\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})) - \{\langle \epsilon, \epsilon \rangle\}$ of basic edit operations. Generate $x \in \Sigma^*$ and $y \in \Gamma^*$ in parallel, starting initially with $l \leftarrow \epsilon$ and $s \leftarrow \epsilon$. Repeat the following process: (*) Draw ω from the set $\Omega \cup \{\#\}$ with probability θ. If $\omega = \#$, stop; i. e., # is a terminating symbol, and we must require $\theta(\#) > 0$ to ensure that the process terminates eventually. Otherwise it must be the case that $\omega = \langle \sigma, \gamma \rangle$ for some $\sigma \in \Sigma \cup \{\epsilon\}$ and $\gamma \in \Gamma \cup \{\epsilon\}$. Append σ to x and γ to y, and repeat from (*).

Equivalently, one can view the generative process as producing an *alignment* $a \in \Omega^*$ from which the strings x and y are projected. Following Mohri [2002a], one can formally define the notion of alignment in terms of a sequence $a \in \Omega^*$ of basic edit operations (insertions, deletions, substitutions). Let π_i be the ith projection of an n-tuple $z = \langle z_1, \ldots, z_n \rangle$, i. e., $\pi_i(z) = z_i$. Note that π_1 gives rise to a fine morphism $\pi_1^* : \Omega^* \to \Sigma^*$ (see page 79), and similarly for π_2.

Definition 4.1 (Alignment). An alignment a of two strings $x \in \Sigma^*$ and $y \in \Gamma^*$ is an element of $\Omega^* = (((\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})) - \{\langle \epsilon, \epsilon \rangle\})^*$ such that $\pi_1^*(a) = x$ and $\pi_2^*(a) = y$. Moreover, let $h : \Omega^* \to \Sigma^* \times \Gamma^*$ be the function $a \mapsto \langle \pi_1^*(a), \pi_2^*(a) \rangle$, i. e., $h(a) = \langle x, y \rangle$ whenever a is an alignment of x and y.

A memoryless model P_1 over Ω^* corresponding to the generative process described before is easy to define. Let $a = \langle a_1, \ldots, a_n \rangle$ be an element of Ω^*. The generative process produces the alignment a with probability $P_1(a \mid \theta)$, which is defined to be (note the presence of the terminating symbol #)

$$P_1(a \mid \theta) = \left(\prod_{i=1}^{n} \theta(a_i) \right) \theta(\#). \tag{4.2}$$

The $((|\Sigma| + 1) \times (|\Gamma| + 1))$-dimensional parameter vector θ is treated as a function $\Omega \cup \{\#\} \to [0; 1]$ throughout this discussion. We require $\theta(\#) > 0$ and $\sum_{\omega \in \Omega \cup \{\#\}} \theta(\omega) = 1$.

Since generating an alignment a implicitly generates a pair of strings $h(a)$, one can make this explicit by defining a model P_3 over $\Omega^* \times \Sigma^* \times \Gamma^*$ as follows:

$$P_3(a, x, y \mid \theta) = \begin{cases} P_1(a \mid \theta) & \text{if } h(a) = \langle x, y \rangle \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The model P_2 on $\Sigma^* \times \Gamma^*$ which we are ultimately interested in arises through marginalization of P_3 by summing over all alignments of x and y:

$$P_2(x, y \mid \theta) = \sum_{a \in \Omega^*} P_3(a, x, y \mid \theta) \tag{4.4}$$

In other words, the probability of a string x corresponding to a string y is the total probability of all alignments of x and y. The basic model P_1 (or, equivalently, P_3) defines the probability of one alignment. However, unlike in computational biology [Durbin et al., 1998], we are rarely interested in knowing the probability of a single alignment, which could be obtained by evaluating P_1 or P_3. Instead we may want to evaluate $P_2(x, y)$ for given strings x and y, which involves computing a sum over exponentially many alignments in the worst case. Evaluating the mass function P_2 is analogous to the first of the Ferguson–Rabiner problems for HMMs [Rabiner, 1989]. We address this problem in Section 4.2.

The situation is equally tricky for parameter estimation, which is similar to Rabiner's [1989] Problem 3. The problem is that we want to work with unaligned training data, i. e., we can only observe examples of co-occurring $\langle x, y \rangle$ pairs, but not their alignments. This means that we have to estimate the parameters θ – the probabilities of edit operations, which naturally belong to P_1 – in terms of observations that we assume were generated by the model P_2 derived from P_1. Parameter estimation under these circumstances is discussed in Section 4.3.

Rabiner's [1989] Problem 2 concerns decoding, in this case, finding the most plausible alignment and/or the best output string. We discuss issues concerning decoding in Section 4.5. The models used during decoding are often conditional models, or are joint models derived from a conditional channel model and a univariate source model, as described above. We are therefore faced with a fourth

problem that has no direct analog in the HMM literature, namely obtaining conditional and/or marginal models from a joint model. The reason this question never arises in conjunction with HMMs is that HMMs are usually not conceptualized as transducers realizing joint models [there are exceptions, however, most notably Jelinek, 1997 and Manning and Schütze, 1999]. Marginalization and conditionalization of joint models are discussed in Section 4.4.

4.2 Evaluating the Mass Function of a Joint Model

This section discusses the issue of computing the value $P_2(x, y)$ of the joint mass function P_2 for a specific pair of strings $\langle x, y \rangle$. As mentioned above, this requires a summation over exponentially many alignments of x and y in the worst case. The exact number of possible alignments of two strings x and y as defined in Definition 4.1 is given by the Delannoy number $D(|x|, |y|)$. The Delannoy numbers [Sloane, 1996–2003, A8288; see also Banderier and Schwer, 2002 for an overview and history] are defined in terms of the recurrence

$$\begin{aligned} D(a, 0) &= 1, \\ D(0, b) &= 1, \\ D(a, b) &= D(a-1, b) + D(a, b-1) + D(a-1, b-1). \end{aligned} \tag{4.5}$$

This recurrence resembles Pascal's formula for the binomial coefficients, except for the presence of the second additive term. Efficient algorithms for evaluating recurrences of this sort involve the use of dynamic programming [see for example Cormen et al., 1990, ch. 16]. In fact, the straightforward dynamic programming algorithm for evaluating the Delannoy number $D(|x|, |y|)$ can be used as the basis

for the algorithms that sum over the $D(|x|, |y|)$ many alignments of the strings x and y in the computation of $P_2(x, y)$.

4.2.1 The Generic Forward Algorithm

The classical dynamic programming algorithm for computing $P_2(x, y)$ is analogous the Forward step of the Forward-Backward algorithm for Hidden Markov Models (HMMs) [see Jelinek, 1997; Durbin et al., 1998]. We present the algorithm for the Forward step for memoryless transducers in some detail here, for two main reasons. First, although Ristad and Yianilos [1998, sec. 2.2] gave an explicit algorithm, which they call FORWARD-EVALUATE and which our algorithm is based on, their presentation contains mistakes. Second, our version of the Forward algorithm abstracts away from certain details of the algebraic operations on edit costs or probabilities. We use this opportunity to introduce an algebraic framework for weighted graphs and automata that will be used throughout the remainder of this chapter.

We call the algorithm presented in this section the generic Forward algorithm. It is a generalization both of Ristad and Yianilos's [1998] algorithm FORWARD-EVALUATE and of the standard dynamic programming algorithm for computing string edit distances, including the Levenshtein distance [Wagner and Fischer, 1974; Kruskal, 1983].

Pseudocode for the generic Forward algorithm appears in Figure 4.1. Observe that it mentions two binary operations $\oplus$ and $\otimes$, and two constants $\bar{0}$ and $\bar{1}$. Replacing the abstract binary operations by addition and multiplication on the non-negative real numbers, and the abstract constants by the numbers zero and one, respectively, yields the standard Forward algorithm.

```
1:  // Input: strings x ∈ Σ*, y ∈ Γ*, cost function θ
2:  ⟨x_1, ..., x_T⟩ ← x
3:  ⟨y_1, ..., y_V⟩ ← y
4:  α[0, 0] ← 1̄
5:  for t ← 0 to T do
6:      for v ← 0 to V do
7:          if t > 0 ∨ v > 0 then
8:              α[t, v] ← 0̄
9:          end if
10:         if t > 0 then // deletion
11:             α[t, v] ← α[t, v] ⊕ α[t − 1, v] ⊗ θ(x_t, ϵ)
12:         end if
13:         if v > 0 then // insertion
14:             α[t, v] ← α[t, v] ⊕ α[t, v − 1] ⊗ θ(ϵ, y_v)
15:         end if
16:         if t > 0 ∧ v > 0 then // substitution
17:             α[t, v] ← α[t, v] ⊕ α[t − 1, v − 1] ⊗ θ(x_t, y_v)
18:         end if
19:     end for
20: end for
21: α[T, V] ← α[T, V] ⊗ θ(#) // termination
22: return α
```

Figure 4.1: The generic Forward algorithm.

In general, however, the generic operations mentioned in Figure 4.1 can be quite different from ordinary addition and multiplication. In fact, they only need to obey a few axioms, namely those which characterize semiring algebras [Golan, 1992, 1999; Kuich, 1997; Głazek, 2002]. Semirings have been used in Natural Language Processing in the work of Mohri, Pereira and Riley [for example Mohri, 1997; Pereira and Riley, 1997; Mohri et al., 2000] on weighted finite state machines, and they also play a role in context-free parsing algorithms [Goodman, 1998, 1999].

Formally, a semiring is an algebraic structure over a carrier set $\mathbb{K}$, which is the codomain of the parameter (cost) function $\theta : \Omega \cup \{\#\} \to \mathbb{K}$ in the general case. Semirings are defined as follows.

Definition 4.2 (Semiring). A semiring is an algebraic structure $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ with the following properties:

1. $\langle \mathbb{K}, \oplus, \bar{0} \rangle$ is a commutative monoid;
2. $\langle \mathbb{K}, \otimes, \bar{1} \rangle$ is a monoid;
3. $\bar{0}$ is a two-sided annihiliator for $\otimes$, in other words, $a \times \bar{0} = \bar{0} = \bar{0} \times a$ for all $a \in \mathbb{K}$;
4. $\otimes$ distributes over $\oplus$ on both sides, in other words, we have both $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$ for all $a, b, c \in \mathbb{K}$.

(The name *semiring* is due to the fact that this structure resembles a unit ring, except that the presence of additive inverses is not required. Because of that, semirings are also known as "rigs", a pun on "rings without negation".)

The multiplication symbol $\otimes$ may be omitted if no confusion can arise. Furthermore, define $a^0 = \bar{1}$ and $a^n = a \otimes a^{n-1}$ for $n > 0$. As usual, $(\cdot)^n$ has the highest precedence, and $\otimes$ has precedence over $\oplus$.

Intuitively, a semiring allows us to summarize the weights of the edges along a path by $\otimes$-multiplying them. Since the edges of a path appear in a certain order, $\otimes$ is associative (there is no tree structure on the edges), but not commutative (order of appearance matters). Also, finite sets of paths with common end points can be summarized by $\oplus$-adding together individual paths. As the members of a set are unordered, the $\oplus$-addition operation is associative and commutative. Distributivity of $\otimes$-multiplication over $\oplus$-addition means that a finite set of paths can be extended on either side by a path, and the effect is the same as extending each path in the set. When used to summarize weighted graphs, semirings are also known as path algebras.

Any (associative) unit ring is also a semiring, for example, the set $\mathbb{Z}$ of integers under addition and multiplication. An even simpler example of a semiring is the set $\mathbb{N}$ of natural numbers under addition and multiplication, which is not a ring due to the lack of additive inverses. Two other semirings need to be explicitly defined at this point.

1. The *nonnegative real semiring* $\mathbb{R}^{\geq 0}_{+,\times} = \langle \{x \in \mathbb{R} \mid x \geq 0\}, +, \times, 0, 1 \rangle$ extends the natural numbers and provides a framework for manipulating probabilities [Mohri, 1997 mentions the real semiring without restriction to the nonnegative reals; Goodman, 1999 defines his "inside semiring" over the nonnegative reals plus ∞, which we will discuss later on]. To repeat what was pointed out above, using the newly introduced terminology: the standard Forward algorithm is the instantiation of the generic Forward algorithm in the real semiring $\mathbb{R}^{\geq 0}_{+,\times}$.

To compute the probability $P_2(x, y \mid \theta)$, apply the standard Forward algorithm to x, y and θ. The algorithm returns the $(|x| + 1) \times (|y| + 1)$ matrix α (with indices starting at 0), whose last entry $\alpha[|x|, |y|]$ equals $P_2(x, y \mid \theta)$. More generally, $\alpha[m, n]$ is the probability (summed over all alignments, but ignoring proper termination) of the prefixes $\langle x_1, \ldots, x_m \rangle$ and $\langle y_1, \ldots, y_n \rangle$.

The algorithm is not restricted to values from the interval $[0; 1]$. For example, if $\theta(\omega) = 1$ for all $\omega \in \Omega \cup \{\#\}$ then on termination $\alpha[a, b]$ contains the Delannoy number $D(a, b)$.

2. The structure $\mathbb{R}_{\min,+} = \langle \mathbb{R} \cup \{+\infty, -\infty\}, \min, +, +\infty, 0 \rangle$, called the min–plus or (real) *tropical semiring*, is used in the calculation of shortest paths in weighted graphs, where the weight of a path is the sum of the weights of its edges [see for example Mohri, 1997 for further applications in language and speech processing]. Instantiating the generic Forward algorithm in this semiring yields the classical dynamic programming algorithm for computing the string edit distance [see Kruskal, 1983, sec. 5 for an overview and a discussion of the 'remarkable history of multiple independent discovery' – at least nine times – of this algorithm; Mohri, 2002a shows how to compute the Levenshtein distance slightly less naturally in the real semiring].

   The algorithm computes the Levenshtein distance between strings $x \in \Sigma^*$ and $y \in \Sigma^*$ (note that the two alphabets are required to be identical in this special case) when $\theta(s, \epsilon) = \theta(\epsilon, s) = \theta(s, t) = 1$ and $\theta(s, s) = \theta(\#) = 0$ for all $s, t \in \Sigma^*$ such that $s \neq t$, i. e., insertions, deletions, and substitutions have unit cost and exact matches have zero cost. When applying the algorithm to x, y and this θ, the Levenshtein distance can be found in $\alpha[|x|, |y|]$ on termination.

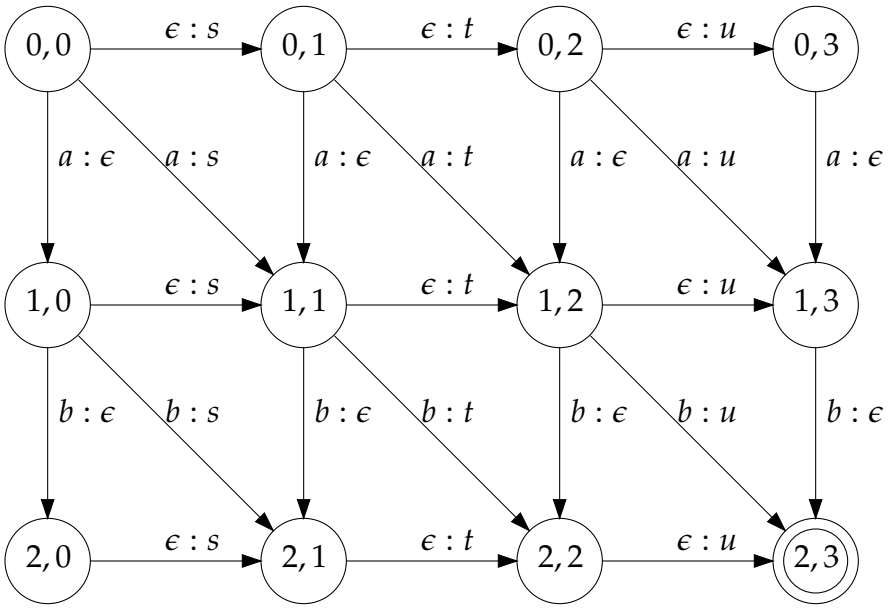

Figure 4.2: Example of an alignment trellis for the string pair $\langle ab, stu \rangle$.

The similarity between the Forward algorithm for HMMs and the dynamic programming algorithm for calculating the Levenshtein distance is pointed out repeatedly in the literature [for example, Ristad and Yianilos, 1998; Jurafsky et al., 2000; Clark, 2001]. However, it is the algebraic perspective that allows us to make precise the intuition that these very similar algorithms are in fact instances of just one generic procedure.

The generic Forward algorithm fills in a matrix α corresponding to a graph ("trellis") compactly representing all alignments of two strings. An example of the alignment trellis for the string pair $\langle ab, stu \rangle$ is shown in Figure 4.2. The vertices of this graph are labeled with indices into α, and its edges are labeled with

edit operations: horizontal edges correspond to deletions, vertical edges to insertions, and diagonal edges to substitutions. The key loop invariant of the generic Forward algorithm is this: when $\alpha[t, v]$ is computed, then $\alpha[t-1, v]$, $\alpha[t, v-1]$, and $\alpha[t-1, v-1]$ already contain the respective $\oplus$-sums over all paths originating from $(0, 0)$. The body of the inner loop then computes $\alpha[t, v]$ in terms of these three values and the weights of its three incoming edges. The invariant holds because the order in which the vertices of Figure 4.2 are explored by the generic Forward algorithm is a topological ordering.

Note that there are only quadratically many vertices and edges in the dynamic programming trellis for strings x and y with lengths $|x|$ and $|y|$. The number of vertices is simply $(|x|+1)(|y|+1)$. The number of edges $T(|x|, |y|)$ is given by the recurrence [Sloane, 1996–2003, A83381]

$$\begin{aligned} T(a, 0) &= a, \\ T(0, b) &= b, \\ T(a, b) &= T(a-1, b) + T(a, b-1) - T(a-1, b-1) + 3, \end{aligned}$$

which is similar to the Delannoy recurrence (4.5), but which can be expressed analytically by the simple formula

$$T(a, b) = 3\,a\,b + a + b.$$

Since the generic Forward algorithm examines every edge and every vertex exactly once, its running time is $O(|x|\ |y|)$ [Wagner and Fischer, 1974, p. 172].

As Ristad and Yianilos [1998] point out, the space requirements of the Forward algorithm can be reduced from $O(|x|\ |y|)$ to $O(\min(|x|, |y|))$ when it is used

to evaluate $P_2(x, y)$ and the intermediate values of the matrix α do not matter. Observe that the matrix α is filled in row by row, and column by column within each row (this corresponds to one topological ordering of the trellis). But $\alpha[t, v]$ only depends on $\alpha[t, v-1]$ from within the same row, and $\alpha[t-1, v]$ and $\alpha[t-1, v-1]$ from the preceding row. Crucially, there is no dependency on any rows before the immediately preceding row. This means that only two row vectors need to be stored at any point. Secondly, if rows are bigger than columns, the matrix can be transposed, i. e., explored column by column (this still corresponds to a topological ordering of the trellis). So if all we care about is the value of $P_2(x, y)$, we do not need to store the entire Forward matrix α. However, access to the full matrix is helpful for parameter estimation, which is discussed in Section 4.3.

4.2.2 The Generic Backward Algorithm

The Forward algorithm calculates $P_2(x, y)$ by examining all combinations of prefixes of x and y. A similar algorithm – the so-called Backward algorithm – applies to the symmetric case involving all suffixes of x and y. A generic version of the Backward algorithm appears in Figure 4.3; the only difference compared with Ristad and Yianilos's [1998] algorithm BACKWARD-EVALUATE is that concrete operations on reals have been replaced with abstract semiring operations. As before, strings x and y are viewed as tuples whose indices start at 1, and the result β is a $(|x|+1) \times (|y|+1)$ matrix with indices starting at 0.

The generic Backward algorithm employs a similar dynamic programming scheme as the generic Forward algorithm, but explores the trellis in reverse. The

```
1:  // Input: strings x ∈ Σ*, y ∈ Γ*, cost function θ
2:  ⟨x_1, ..., x_T⟩ ← x
3:  ⟨y_1, ..., y_V⟩ ← y
4:  β[T, V] ← θ(#) // termination
5:  for t ← T to 0 step −1 do
6:    for v ← V to 0 step −1 do
7:      if t < T ∨ v < V then
8:        θ ← 0̄
9:      end if
10:     if t < T then // deletion
11:       β[t, v] ← β[t, v] ⊕ θ(x_{t+1}, ϵ) ⊗ β[t + 1, v]
12:     end if
13:     if v < V then // insertion
14:       β[t, v] ← β[t, v] ⊕ θ(ϵ, y_{v+1}) ⊗ β[t, v + 1]
15:     end if
16:     if t < T ∧ v < V then // substitution
17:       β[t, v] ← β[t, v] ⊕ θ(x_{t+1}, y_{v+1}) ⊗ β[t + 1, v + 1]
18:     end if
19:   end for
20: end for
21: return β
```

Figure 4.3: The generic Backward algorithm.

Backward trellis is identical to the corresponding Forward trellis – see the example in Figure 4.2 – except that the directions of all edges are reversed (corresponding Forward and Backward trellises are in fact isomorphic as graphs). Likewise, a topological ordering of the Backward trellis can be obtained by reversing a topological ordering of the Forward trellis. The generic Backward algorithm uses such a topological ordering to fill in the Backward matrix β: it explores the Backward trellis in reverse row-by-row order, and reverse column-by-column order within each row. Correctness of the generic Backward algorithm also mirrors the analogous arguments for the generic Forward algorithm.

In the nonnegative real semiring the Backward algorithm can be used to calculate suffix probabilities, including $\beta[0,0] = P_2(x,y)$. When all we are interested in is calculating $P_2(x,y)$, the Forward and the Backward algorithm are equally suitable. They do, however, have uses beyond evaluating the probability mass function P_2, as they can be combined to compute the conditional probability of an edit operation occurring in an alignment of two strings, which plays an important role for parameter estimation. We turn to this in the following section.

4.3 Estimating the Parameters of a Joint Model

So far we have identified the joint distribution P_2 (4.4) as our primary working model and shown how to compute $P_2(x,y \mid \theta)$ for a given pair of strings $\langle x,y \rangle$ and a given parameter vector θ. We now turn to the question of finding a parameter vector $\hat{\theta}$ which maximizes the quantity $P_2(x,y \mid \theta)$ for a given pair of strings $\langle x,y \rangle$. In other words, we want to maximize the likelihood function

$$\theta \mapsto P_2(x,y \mid \theta)$$

in order to find the maximum likelihood estimate (MLE) $\hat{\theta}$.

Parameter estimation is complicated by the fact that P_2 derives from the basic underlying model P_1 (4.2), which is stated in terms of alignments. But alignments are generally not observable. When an unaligned pair of strings $\langle x, y \rangle$ is observed, its sampling distribution is P_2, not P_1. It is clear that an alignment $a \in \Omega^*$ contains more information than a corresponding tuple of strings $\langle x, y \rangle \in \Sigma^* \times \Gamma^*$, as evidenced by the fact that the function $h : \Omega^* \to \Sigma^* \times \Gamma^*$ introduced in Definition 4.1 is many-to-one, since it forgets alignment details and only retains the identities of the aligned strings. Therefore a sample $\langle x, y \rangle \in \Sigma^* \times \Gamma^*$ from P_2 generally corresponds to many samples $a \in \Omega^*$ from P_1, namely the set $\{a \in \Omega^* \mid h(a) = \langle x, y \rangle\}$. On the other hand, maximum likelihood parameter estimation for P_1 is straightforward.

In sum, we have the following situation: parameter estimation for P_2 would be easy if alignments could be observed, but alignments generated by P_1 are purposely obliterated in the derivation of P_2. It is useful to think of this as a so-called *missing data problem* and to use the Expectation Maximization (EM) algorithm [Dempster et al., 1977; McLachlan and Krishnan, 1997] for maximum likelihood estimation. In the terminology of the EM algorithm, P_1 is the *complete-data specification* and P_2 the *incomplete-data specification*. They are directly related through the function h from Definition 4.1, and indirectly through P_3, as follows:

$$P_2(x, y \mid \theta) = \sum_{\substack{a \in \Omega^* \\ h(a) = \langle x, y \rangle}} P_1(a \mid \theta) = \sum_{a \in \Omega^*} P_3(a, x, y \mid \theta)$$

The EM algorithm is a general method for maximizing the incomplete-data likelihood. It is most useful in situations where maximizing the complete-data likelihood is comparatively easy, as is the case here.

4.3.1 Derivation of EM Updates

The key contribution of Ristad and Yianilos [1998] is the explicit formulation of a parameter estimation algorithm for memoryless stochastic transducers, which is as an instance of the EM algorithm. However, Ristad and Yianilos [1998] do not explicitly derive their parameter estimation algorithm from the generic EM algorithm. The goal of this section is to fill in some of the details of this derivation, which closely resembles the derivation of the Forward-Backward algorithm [Jelinek, 1997; Durbin et al., 1998].

Ristad and Yianilos [1998, p. 524] themselves point out that the 'applicability of EM to the problem of optimizing the parameters of a memoryless stochastic transducer was first noted by Bahl, Jelinek and Mercer' in 1975. Algorithms for solving this problem may have been part of the community folklore for quite some time, as earlier authors who worked with stochastic transducers – for example Parfitt and Sharman [1991] or Gilloux [1991]; see also the references cited by Clark [2001, sec. 6.5.1] – must have had some way of obtaining parameter values for their models. For example, Parfitt and Sharman [1991] mention that they used the Forward-Backward algorithm during training, but do not provide further details. Gilloux [1991] briefly discusses the computation of optimal paths and mentions within the same paragraph that 'the learning algorithm applicable to general [Markov] models remains valid', which suggests that he may have used Viterbi training or EM (Forward-Backward) training.

EM parameter re-estimation involves maximizing the expected complete-data log-likelihood. Given a single observation $\langle x, y \rangle$, the EM algorithm iteratively re-estimates the parameter vector θ, constructing a sequence $\theta^{(0)}, \theta^{(1)}, \ldots$ that converges to a stationary point – usually a mode of the likelihood – under very general conditions.

A single re-estimation step of EM maximizes (M) the expected (E) log-likelihood of the complete-data specification, where the expectation is taken over the conditional distribution of the missing data. Let y stand for the observed "incomplete" data and z the unobserved "missing" data. Together they make up the complete data. In general (but shown only for the discrete case here), an EM re-estimation step finds the parameter values which maximize a certain variant of the complete-data likelihood. The complete-data log-likelihood will be referred to as L, and the function being maximized is the conditional expectation of L, usually called Q. A single re-estimation step takes the form

$$\theta^{(n+1)} = \operatorname*{argmax}_{\theta} Q(\theta) = \operatorname*{argmax}_{\theta} \sum_{z} P(z \mid y, \theta^{(n)}) \log P(y, z \mid \theta). \tag{4.6}$$

The EM algorithm, shown in Figure 4.4, simply iterates these updates until some convergence criterion is met.

In the specific present case, the complete data are alignments a. The conditional probability of the missing data is expressed in the usual way as a fraction of the complete-data probability divided by the observed-data probability. An observed sample is a string pair $\langle x, y \rangle$, whose sampling distribution is P_2. The re-estimation step in this specific case becomes

$$\theta^{(n+1)} = \operatorname*{argmax}_{\theta} \sum_{a \in \Omega^*} \frac{P_3(a, x, y \mid \theta^{(n)})}{P_2(x, y \mid \theta^{(n)})} \log P_1(a \mid \theta).$$

1: $\theta^{(0)} \leftarrow$ initial guess of parameter values
2: $n \leftarrow 0$
3: **repeat**
4: calculate the fixed parameters of function Q // so-called E step
5: $\theta^{(n+1)} \leftarrow \operatorname{argmax}_\theta Q(\theta)$ // M step
6: $n \leftarrow n + 1$
7: **until** convergence

Figure 4.4: The EM algorithm.

The conditional probability $P_3(a, x, y \mid \theta^{(n)}) / P_2(x, y \mid \theta^{(n)})$ which the expectation is taken over is abbreviated (by a slight abuse of notation) as $P(a \mid x, y, \theta^{(n)})$. With all conventions in place, finding $\theta^{(n+1)}$ means maximizing the function Q given by

$$Q(\theta) = \sum_{a \in \Omega^*} P(a \mid x, y, \theta^{(n)})\ L(\theta).$$

Suppose $a = \langle a_1, \ldots, a_n \rangle$ is a sequence of edit operations, i. e. an element of Ω^*. Let $\mathrm{C}(\omega, a)$ denote the number of times the edit operation ω occurs in a, and extend this notation to include $\mathrm{C}(\#, a) = 1$ for all $a \in \Omega^*$. The complete-data log-likelihood L can now be rewritten like this:

$$\begin{aligned} L(\theta) &= \log P_1(a \mid \theta) \\ &= \log \left(\theta(\#) \prod_{i=1}^{n} \theta(a_i) \right) \\ &= \log \prod_{\omega \in \Omega \cup \{\#\}} \theta(\omega)^{\mathrm{C}(\omega, a)} \\ &= \sum_{\omega \in \Omega \cup \{\#\}} \mathrm{C}(\omega, a)\ \log \theta(\omega) \end{aligned}$$

Note that this generalizes easily to more sophisticated distributions over Ω^*, especially if $P_1(a \mid \theta)$ can be expressed as a simple product, which is the case for Markov chains.

Incorporating the rearranged form of the complete-data log-likelihood into the conditional expectation Q, the latter can then be simplified as follows:

$$\begin{aligned} Q(\theta) &= \sum_{a \in \Omega^*} P(a \mid x, y, \theta^{(n)}) \ \log L(\theta) \\ &= \sum_{a \in \Omega^*} P(a \mid x, y, \theta^{(n)}) \sum_{\omega \in \Omega \cup \{\#\}} \mathrm{C}(\omega, a) \ \log \theta(\omega) \\ &= \sum_{\omega \in \Omega \cup \{\#\}} \sum_{a \in \Omega^*} P(a \mid x, y, \theta^{(n)}) \ \mathrm{C}(\omega, a) \ \log \theta(\omega) \\ &= \sum_{\omega \in \Omega \cup \{\#\}} \gamma(\omega) \ \log \theta(\omega) \end{aligned}$$

A new piece of notation, $\gamma(\omega)$, was introduced on the last line using the substitution

$$\gamma(\omega) = \sum_{a \in \Omega^*} P(a \mid x, y, \theta^{(n)}) \ \mathrm{C}(\omega, a) \tag{4.7}$$

to group together all terms that do not depend on θ. This move is not just an arbitrary simplifying device, because $\gamma(\omega)$ can be interpreted as the expected number of times ω occurs in an alignment of x and y. Moreover, this is the only term that depends on the sample $\langle x, y \rangle$ and can therefore hide the additional complexity introduced when using multiple independent and identically distributed samples. Furthermore, parameter estimation extends straightforwardly to general stochastic transducers provided γ can be calculated efficiently, which will be the topic of Section 5.3.

Setting aside for the moment the issue of how to compute γ even for the simple case of memoryless transducers, let's turn to the problem of maximizing the expected log-likelihood Q. We still wish to find

$$\theta^{(n+1)} = \operatorname*{argmax}_{\theta} \sum_{\omega \in \Omega \cup \{\#\}} \gamma(\omega) \ \log \theta(\omega)$$

subject to the normalization constraint $\sum_{\omega \in \Omega \cup \{\#\}} \theta(\omega) = 1$ and keeping in mind that θ is really a finite-dimensional vector. This constrained optimization problem can be transformed into an unconstrained problem using the method of Lagrange multipliers. Introduce an auxiliary function

$$\begin{aligned} Q_{\text{aux}}(\theta) &= \sum_{\omega \in \Omega \cup \{\#\}} \gamma(\omega) \log \theta(\omega) + \lambda \left(1 - \sum_{\omega \in \Omega \cup \{\#\}} \theta(\omega) \right) \\ &= \lambda + \sum_{\omega \in \Omega \cup \{\#\}} \left(\gamma(\omega) \log \theta(\omega) - \lambda \, \theta(\omega) \right) \end{aligned}$$

and solve the system of equations (note that $\theta(\omega)$ should be viewed as one component of the finite-dimensional parameter vector θ)

$$\frac{\partial}{\partial \, \theta(\omega)} Q_{\text{aux}} = \gamma(\omega) \frac{1}{\theta(\omega)} - \lambda = 0$$

to obtain the maximizing $\theta^{(n+1)}(\omega)$ for all $\omega \in \Omega \cup \{\#\}$. The result is still dependent on λ, which can be eliminated straightforwardly, since the only constraint is proper normalization of $\theta^{(n+1)}$. This leaves us with the following solution, which forms the core of Ristad and Yianilos's [1998] algorithm MAXIMIZATION-STEP:

$$\theta^{(n+1)}(\omega) = \frac{\gamma(\omega)}{\lambda} = \frac{\gamma(\omega)}{\sum_{\omega \in \Omega \cup \{\#\}} \gamma(\omega)} \tag{4.8}$$

Using EM for parameter estimation is an elegant solution when alignments are not known, because it deals properly with the uncertainty about the contribution of an individual alignment to the total probability assigned to a pair of strings. If alignments are known, parameter estimation is very straightforward, because no random variables are involved and one can simply count how often a particular edit operation occurs in an alignment. In practice, it may be tempting to cut corners and pretend that alignments are known even when they are not. Instead of computing the expectation of the log likelihood over all possible values of the missing data z (the alignments), however improbable some of them may be, one could concentrate on the k most probable values and treat them as though they were part of the observed data (denoted by y in the EM re-estimation formula). For $k = 1$ this is known as *Viterbi training* [see Bridle 1997 for a high-level discussion of optimization methods for maximum likelihood training]. Compare the following parameter update for Viterbi training with equation (4.6) for EM:

$$\theta^{(n+1)} = \operatorname*{argmax}_{\theta} \log P(y, \hat{z} \mid \theta) \quad \text{where} \quad \hat{z} = \operatorname*{argmax}_{z} P(z \mid y, \theta^{(n)}).$$

The imputed information $\hat{z}$ is treated as observed along with the regular observed data y, with the effect that the parameter update on the left is now very straightforward. For the concrete application of stochastic memoryless transducers this means finding the best alignment $\hat{a}$ of two strings, which is very similar to the decoding problem discussed in Section 4.5. Whether or not Viterbi training is a useful approximation depends on the concrete circumstances. This issue will be investigated empirically in Section 6.6.

4.3.2 Calculating Expected Counts

We still need to address the issue of computing the expected parameter counts γ, which is part of the so-called E step of the EM algorithm. This may at first seem to involve summing over an infinite number of alignments $a \in \Omega^*$ according to equation (4.7). However, since $P_3(a, x, y)$ is zero unless a is an alignment of x and y, the summation is really only over the set $h^{-1}(x, y) = \{a \in \Omega^* \mid h(a) = \langle x, y \rangle\}$ of all alignments of x and y. So the equation for γ simplifies to

$$\gamma(\omega) = \frac{1}{P_2(x, y)} \sum_{a \in h^{-1}(x,y)} P_1(a)\, \mathrm{C}(\omega, a). \tag{4.9}$$

Calculating $\gamma(\omega)$ simultaneously for all $\omega \in \Omega \cup \{\#\}$ involves a dynamic programming scheme very similar to the Forward or Backward algorithm. The key insight is to rearrange the computation to obtain the contribution of each edit operation in its specific context. For a concrete example, consider again the trellis graph from Figure 4.2, and suppose we want to compute $\gamma(\epsilon : t)$. We had already seen in the previous section how to calculate the normalizing term $P_2(ab, stu)$ of the conditional distribution. All that remains to do is to calculate the unconditional expectation of the edit operation $\epsilon : t$. There are three edges labeled $\epsilon : t$ in the trellis which contribute to some of the alignments of ab and stu. Their contributions can be considered in isolation and accumulated by the algorithm. In other words, the unconditional expectation can be expressed in terms of the expected counts of edges, rather than the expected counts of edit operations. This is easy to do in an acyclic graph, since each edge occurs at most once in any successful path, so its expected count is simply the total probability of all paths it is a part of. For instance, the total probability of all paths the go through the edge labeled $\epsilon : t$

calculate α using the Forward algorithm
calculate β using the Backward algorithm
$p \leftarrow \beta[0,0]$ // normalizing term, assumed to be nonzero
for each $\omega \in \Omega$ **do**
  $\gamma(\omega) \leftarrow 0$
end for
$\gamma(\#) \leftarrow 1$
for each edge e in the Forward/Backward trellis **do**
  $\omega \leftarrow$ the label of e
  $s \leftarrow$ the source state of e
  $t \leftarrow$ the target state of e
  $\gamma(\omega) \leftarrow \gamma(\omega) + \dfrac{\alpha[s]\,\theta(\omega)\,\beta[t]}{p}$
end for

Figure 4.5: Calculating expected counts of basic edit operations for memoryless transducers.

from $(1,1)$ to $(1,2)$ is the probability of reaching the vertex labeled $(1,1)$ through any path, then traversing the edge in question, and then going from $(1,2)$ to the final state via any path. But the prefix probability of reaching $(1,1)$ from the start state $(0,0)$ is just $\alpha[1,1]$ calculated by the Forward algorithm, and the suffix probability of reaching the final state from $(1,2)$ is $\beta[1,2]$ calculated by the Backward algorithm.

The calculation of expected counts of edit operations in $\Omega \cup \{\#\}$ is sketched in Figure 4.5. Note that the expected count of the terminating symbol # is necessarily one, since it occurs precisely once in every alignment generated by P_1 (4.2). The algorithm simply explores all edges in any order and accumulates the expected counts of their labels, which are edit operations in Ω.

Ristad and Yianilos's [1998] algorithm EXPECTATION-STEP iterates over edges in a specific order, namely in topological order of their source vertices. It is therefore possible, as they note, to combine this algorithm with the Forward or Backward algorithm. We do not include another version of their algorithm here, since no changes to Ristad and Yianilos's [1998] presentation are necessary, nor are generalizations possible, as there is no meaningful notion of expected values in arbitrary semirings. Instead, the natural place for expected values may be in a semimodule over the nonnegative real semiring [Eisner, 2001]. We will return to this issue in Section 5.3, when we consider parameter estimation for general stochastic transducers. It will become clear there that the present method of calculating expectations is applicable in the general case too.

In conclusion, the parameter of memoryless stochastic transducers can be estimated by the following instance of the EM algorithm: in the so-called E step, the expected counts $\gamma(\omega)$ of the edit operations $\omega \in \Omega \cup \{\#\}$ are computed using the algorithm in Figure 4.5; in the M step, these expected counts are normalized and become the re-estimated parameter values, as motivated by equation (4.8). While Ristad and Yianilos [1998] give explicit algorithms for EM-style parameter estimation, they do not provide much of an explanation or an explicit derivation of their algorithms from the abstract formulation of EM. The present section filled in those details, and also forms the background for Section 5.3.

4.4 Obtaining Conditional Models

We have seen procedures for evaluating the probability mass function of a joint model and for estimating its parameters. But in many applications we would

sometimes like to work with conditional models instead of joint models. For example, the abstract channel model P_{chn} discussed on page 109 is a conditional model which can be combined with a source model P_{src} to form a joint model. For letter-to-sound conversion, we might want to combine a conditional channel model with a phonotactic source model, such as an n-phone model.

Depending on the application, directly estimating the parameters of a conditional model may be desirable; however, discriminative training of conditional models is much less straightforward than parameter estimation of joint models, and we do not propose to deal with it here. Instead we focus on the simpler problem of conditionalizing a given joint model. In the specific case of memoryless transductions this means, given a stochastic transducer realizing $P_2 : \Sigma^* \times \Gamma^* \to \mathbb{R}^{\geq 0}$, how does one obtain a stochastic transducer realizing the function $\langle x, y \rangle \mapsto P_2(x, y) / \sum_{x'} P_2(x', y)$?

Ristad and Yianilos [1998] never even mention this question. This is not surprising, as it does not seem to be discussed in the traditional HMM literature either. The present section shows that it is always possible to derive a memoryless conditional model from a memoryless joint model. The analogous statement for general stochastic transducers is false, as discussed in Section 5.4. As a side benefit, we also obtain the corresponding marginal distributions.

4.4.1 Evaluating the Mass Function of a Conditional Model

While the parameter vector θ of a joint model is itself a probability distribution on $\Omega \cup \{\epsilon\}$, naively renormalizing it such that $\sum_\sigma \theta(\sigma, \gamma) = 1$ for all $\gamma \in \Gamma \cup \{\epsilon\}$ would not give rise to the desired conditional distribution on $\Sigma^* \times \Gamma^*$, because of the special status of insertion and deletion operations.

Let's start with the simpler problem of evaluating the conditional mass function, i. e., calculating the value of the following expression for fixed x and y:

$$\frac{P_2(x, y \mid \theta)}{\sum_{x' \in \Sigma^*} P_2(x', y \mid \theta)} \tag{4.10}$$

Computing the numerator of this fraction was discussed in Section 4.2 and will be very familiar by now. Evaluating the denominator involves a summation over the infinite set Σ^* and therefore deserves special attention.

The following concrete example illustrates the general computation. Suppose $\Sigma = \{s\}$ and $\Gamma = \{f, g\}$, and θ is given as

$$\begin{aligned} \theta(s, f) &= 0.2, & \theta(\epsilon, f) &= 0.3, \\ \theta(s, g) &= 0.1, & \theta(\epsilon, g) &= 0.2, \\ \theta(s, \epsilon) &= 0.1, & \theta(\#) &= 0.1. \end{aligned}$$

Say we want to evaluate the denominator for $y = fg$, i. e., we need to compute $\sum_{x'} P_2(x', fg \mid \theta)$. This means evaluating P_2 for every pair of strings in the set $\{\langle x', fg \rangle \mid x' \in \Sigma^*\}$, or finding the total probability mass contained by P_2 restricted to this set. In the concrete example at hand we want to compute the infinite sum

$$P_2(\epsilon, fg) + P_2(s, fg) + P_2(ss, fg) + P_2(sss, fg) + P_2(ssss, fg) + \cdots .$$

The stochastic transducer that accepts precisely the strings occurring in the terms of this sum appears in Figure 4.6. (It is the result of composing the single-state

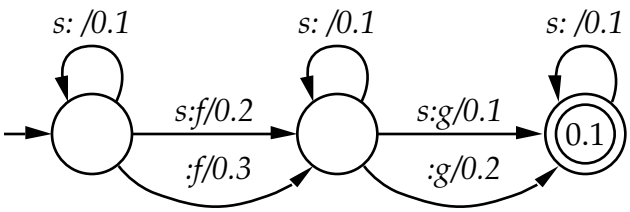

Figure 4.6: A stochastic transducer realizing P_2 restricted to $\{\langle x', fg\rangle \mid x' \in \Sigma^*\}$.

transducer realizing P_2 with the identity transducer representing the string fg. More on this in Chapter 5.) By consulting Figure 4.6 it is easy to check that

$$\begin{aligned}
P_2(\epsilon, fg) &= 0.3 \times 0.2 \times 0.1 = 0.006, \\
P_2(s, fg) &= ((0.2 \times 0.2) + (0.3 \times 0.1) + 3(0.1 \times 0.3 \times 0.2)) \times 0.1 = 0.0088, \\
P_2(ss, fg) &= 0.00446, \\
P_2(sss, fg) &= 0.00108, \\
P_2(ssss, fg) &= 0.000199.
\end{aligned}$$

The probability for longer strings x' decreases despite a concomitant increase in the number of paths that contribute to it. The sum must converge, since the set of strings that contribute to it is a subset of $\Sigma^* \times \Gamma^*$, across which P_2 sums to unity. Therefore the infinite sum in this example must be greater, but not much greater, than 0.020539.

The exact value of the infinite sum can be computed recursively. But first, note that the edge labels are not important, since we need to sum over all paths without

attention to their labels. After removing edge labels and replacing multiple edges between the same pair of source and target vertices we get the simple graph in Figure 4.7. The sum of all paths through that graph is, however, still the same as the sum of all paths through the transducer in Figure 4.6.

Computing this sum would be easy if the graph were acyclic. Whereas the graph does have cycles, the only cycles it has are loops. Call such a graph *almost-acyclic*. Note that the transducer representing the marginal probability at a single point y is necessarily almost-acyclic: the string y is accepted by an acyclic automaton, and its composition with the memoryless transducer realizing P_2 is acyclic except for the presence of arcs labeled with deletion operations, whose presence is not affected by y. But since the memoryless transducer has a trivial one-state topology, the only deletion arcs are loops, corresponding to the deletion loops of the memoryless transducer.

Theorem 4.1. *A graph is almost-acyclic iff it is isomorphic to a graph with an upper triangular adjacency matrix.*

This is a trivial generalization of the observation that a graph is acyclic iff its states can be renumbered in such a way as to make its adjacency matrix strictly upper triangular. We may therefore assume w. l. o. g. that a weighted directed almost-acyclic graph is always represented by an upper triangular matrix.

In cases like the present example in Figure 4.7 the weight of a loop cannot be unity, since such a loop corresponds to one or more deletion operations whose total probability must be less than one because $\theta(\#) > 0$. Let M be the upper triangular adjacency matrix of such a graph, and I the identity matrix with compatible dimensions. Then $I - M$ is triangular with no zeroes in its diagonal, and therefore $\det(I - M) \neq 0$, meaning $I - M$ is invertible. The matrix inverse $(I - M)^{-1}$ could

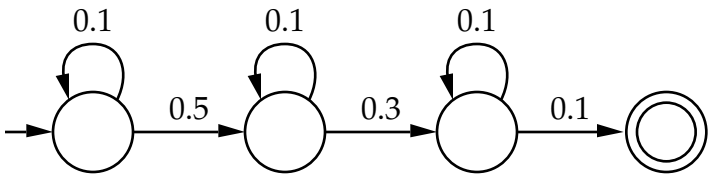

Figure 4.7: An unlabeled directed weighted graph that abstracts away from irrelevant details included in Figure 4.6.

be computed very easily using LU decomposition [Press et al., 1992, sec. 2.3] without the need to explicitly compute the LU decomposition for the triangular matrix $I - M$. But $(I - M)^{-1} = M^*$ [Fink, 1992, sec. 2.5], the closure of M, and so the sum of all paths from the unique (w. l. o. g.) source state s to the unique (w. l. o. g.) target state t is found in $M^*[s, t]$. However, real matrix inversion is a cubic time algorithm whose full generality is not needed in this special case, though it will be when we revisit these issues in Chapter 5.

While the single-source algebraic path algorithm for directed acyclic weighted graphs (DAWGs) [Cormen et al., 1990, sec. 25.4] cannot be used directly for almost-acyclic graphs like the one in Figure 4.7, it can easily be modified to accommodate them, provided the infinite sums corresponding to the loops exist. The advantage of this algorithm is that it runs in time linear in the number of vertices and edges of the graph, like the Forward algorithm, which is in fact a special instance of the algebraic path algorithm for DAWGs, as we shall see later. The vertices are examined in topological order, i. e., from left to right for the graph in Figure 4.7.

Suppose w is the total weight of all paths through the graph excluding the loop on the first state. Since that loop can be taken any number of times, the infinite sum can be expressed in terms of w as

$$\begin{aligned}\sum_{x'} P_2(x', fg) &= w + 0.1\, w + 0.1^2\, w + 0.1^3\, w + \cdots \\ &= w \sum_{i=0}^{\infty} 0.1^i \\ &= w \frac{1}{1 - 0.1} = \frac{10}{9}\, w.\end{aligned}$$

The geometric series on the second line converges and can be expressed in closed form as the fraction shown on the third line. This convergence result holds in general, because the weight of the loop is always less than one, as noted above.

Moving on to the next state, we can write $w = 0.5\, w'$ where w' is the total weight of all paths starting from the second state and including the loop there. Proceed recursively to bring the infinite sum into closed form, arriving ultimately at

$$\begin{aligned}\sum_{x'} P_2(x', fg) &= \frac{1}{1 - 0.1} \times 0.5 \times \frac{1}{1 - 0.1} \times 0.3 \times \frac{1}{1 - 0.1} \times 0.1 \\ &= \frac{5}{243} \\ &\approx 0.020576.\end{aligned} \tag{4.11}$$

We have evaluated the denominator of (4.10) for $y = fg$. Using the joint probabilities computed earlier, which occur in the numerator of (4.10), the corresponding conditional probabilities are as follows:

$$\begin{aligned}
P_2(\epsilon \mid fg) &= 0.006000 \, \frac{243}{5} = 0.2916, \\
P_2(s \mid fg) &= 0.008800 \, \frac{243}{5} = 0.427680, \\
P_2(ss \mid fg) &= 0.004460 \, \frac{243}{5} = 0.216756, \\
P_2(sss \mid fg) &= 0.001080 \, \frac{243}{5} = 0.052488, \\
P_2(ssss \mid fg) &= 0.000199 \, \frac{243}{5} = 0.0096714.
\end{aligned}$$

We can see that the probability that one of these five strings from Σ^* corresponds to, or was generated by, the string $fg \in \Gamma^*$ is more than 99.8%. The most likely string in Σ^* corresponding to fg is s, by a wide margin.

A generic algorithm that computes marginal weights in arbitrary semirings must rely on being able to compute infinite summations like the geometric series in the nonnegative real semiring. In an arbitrary semiring, a unary partial operation $(\cdot)^*$ called *closure* may be defined for certain elements $a \in \mathbb{K}$. When defined for a, closure obeys the axiom

$$a^* = \bar{1} \oplus (a \otimes a^*) = \bar{1} \oplus (a^* \otimes a). \tag{4.12}$$

Recursively expanding the right-hand side gives us

$$a^* = a^0 \oplus a^1 \oplus a^2 \oplus \cdots = \bigoplus_{i=0}^{\infty} a^i.$$

Associativity and distributivity are required to hold for all sums, including any infinite sums that may exist. We say that a semiring is *closed* if a^* is defined for all $a \in \mathbb{K}$. The nonnegative real semiring is not closed, since $(\cdot)^*$, the limit of the infinite geometric series, is a partial operation $a \mapsto 1/(1-a)$ defined only on the half-open interval $[0;1)$.

The generic single-source algebraic path algorithm for directed almost-acyclic weighted graphs shown in Figure 4.8 is a straightforward generalization of algorithm DAG-SHORTEST-PATHS presented by Cormen et al. [1990, sec. 25.4]. It computes the matrix closure M^* of an upper triangular adjacency matrix M over a semiring in which the semiring closure operation $(\cdot)^*$ is defined for all diagonal elements of M. Using an adjacency matrix representation makes the presentation of the algorithm slightly easier. However, changing the representation of the graph to employ adjacency lists – perhaps with a separate list of loops – is straightforward and results in a running time linear in the number of vertices and edges.

Everything is in place now for computing the marginal probability

$$\sum_{x' \in \Sigma^*} P_2(x', y)$$

for a fixed string $y \in \Gamma^*$. All we have to do is apply the single-source algebraic path algorithm from Figure 4.8 to a simple graph corresponding to transducers of the form shown in Figure 4.7. The marginal probability can then be found in $d[t]$, where t is the unique (w. l. o. g.) target state of the graph. Since we now can evaluate the marginal probability, and we already knew how to evaluate the joint probability, computing conditional probabilities of the form (4.10) is easy.

1: // Input: $n \times n$ adjacency matrix M of a weighted graph
Require: M is upper triangular
2: **for** $i \leftarrow 1$ **to** n **do**
3: $\quad d[i] \leftarrow \bar{0}$
4: **end for**
5: $d[1] \leftarrow \bar{1}$
6: **for** $i \leftarrow 1$ **to** n **do**
7: $\quad d[i] \leftarrow d[i] \otimes (M[i,i])^*$
8: $\quad$ **for** $j \leftarrow i+1$ **to** n **do**
9: $\quad\quad d[j] \leftarrow d[j] \oplus d[i] \otimes M[i,j]$
10: $\quad$ **end for**
11: **end for**
12: **return** d
Ensure: $d[i]$ is the algebraic distance between vertex 1 and vertex i

Figure 4.8: The generic single-source algebraic path algorithm for almost-acyclic weighted directed graphs.

4.4.2 Marginal Automata

In general, being able to evaluate the conditional probability for a fixed pair of strings is not enough. We are looking for a way to represent the conditional distribution

$$\langle x, y \rangle \mapsto P_2(x, y) \times \frac{1}{\sum_{x' \in \Sigma^*} P_2(x', y)} \tag{4.13}$$

(call it C) as a stochastic transducer. This would be possible if there was a function R, represented by a stochastic automaton, for the reciprocal of the marginal density, i. e.:

$$R(z, y) = \begin{cases} \dfrac{1}{\sum_{x' \in \Sigma^*} P_2(x', y)} & \text{if } z = y \\ 0 & \text{otherwise} \end{cases}$$

The conditional distribution C (4.13) could then be expressed as $C = P_2 \circ R$, where $\circ$ denotes composition of weighted transducers [Mohri et al., 1996; Pereira and Riley, 1997] (see Chapter 5 for further details). The reciprocal marginal R can in turn be computed from an unambiguous stochastic transducer representing the marginal density by setting its parameters to their reciprocal values. For example, if there was a stochastic automaton with parameter function $\delta : \Gamma \cup \{\#\} \to [0;1]$ such that

$$\sum_{x'} P_2(x', \langle y_1, \ldots, y_n \rangle) = \delta(\#) \prod_{i=1}^{n} \delta(y_i)$$

for all $\langle y_1, \ldots, y_n \rangle \in \Gamma^*$, then replacing $\delta(\gamma)$ with $1/\delta(\gamma)$ for all $\gamma \in \Gamma \cup \{\#\}$ would yield a stochastic automaton representing R.

As the discussion surrounding equation (4.11) has shown, the marginal probability, even though it is expressed as a sum over products, can generally be represented as a product. Observe that each state (except the added sink state) in an automaton like the one in Figure 4.7 representing a marginal probability has a loop whose weight is the sum of the weights of all deletion operations. That means, each non-loop edge in that graph is preceded by a loop, and hence the loops can be eliminated by pre-$\otimes$-multiplying the parameters for the insertion and substitution operations with the semiring closure of the total deletion weight.

This leads directly to an algorithm for constructing a stochastic automaton realizing the marginal density $y \mapsto \sum_{x'} P_2(x', y)$ from a given transducer representation of P_2. The pseudo-code listing appears in Figure 4.9. The algorithm works in any semiring, provided the semiring closure operation is defined for the total

```
// Input: parameter function θ of a joint memoryless transducer
d ← 0̄    // total deletion weight
for each σ ∈ Σ do
    d ← d ⊕ θ(σ, ϵ)
end for
for each γ ∈ Γ do
    δ(γ) ← θ(ϵ, γ)    // marginal weight of γ
    for each σ ∈ Σ do
        δ(γ) ← δ(γ) ⊕ θ(σ, γ)
    end for
    δ(γ) ← d* ⊗ δ(γ)    // pre-multiply with the weight of the deletion loop
end for
δ(#) ← d* ⊗ θ(#)
return δ
```

Figure 4.9: The generic marginalization algorithm for memoryless transducers.

deletion weight d. Applying this algorithm to our running example yields the following parameter function for the marginal automaton:

$$\delta(\#) = 1/9,$$

$$\delta(f) = 5/9,$$

$$\delta(g) = 3/9.$$

We can use this construction to remove the loops from the almost-acyclic graph in Figure 4.7. The result, which appears in Figure 4.10, is a DAWG that is equivalent in the sense that the sum over all paths weights, which is $4^2/9^3 = 16/729$, is the same as the value calculated previously in equation (4.11) for the almost-acyclic graph.

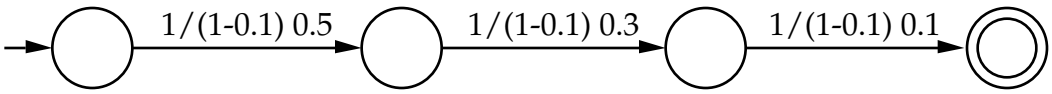

Figure 4.10: An acyclic graph which is equivalent to the almost-acyclic graph from Figure 4.7.

It is easy to see that the parameter function δ of the marginal automaton is a proper probability distribution. In general, suppose θ is a probability distribution over a set of basic events partitioned into two disjoint sets C and D. Let d denote the total probability mass of D. The total probability mass of C is therefore $1 - d$, which when multiplied with $d^* = 1/(1-d)$ equals unity. In other words, θ can be turned into a probability distribution over C rather than over $C \cup D$ by multiplying the probabilities of all basic events in C with $1/(1-d)$. In the marginalization algorithm, θ is defined over the set $\Omega \cup \{\#\}$, D is the set of all deletion operations, C is the set of all other operations (whose second projection is the domain of δ), and d appears verbatim in the algorithm.

4.4.3 Conditional Stochastic Transducers

The marginal automaton is unambiguous: for each string $y \in \Gamma^*$ there is at most one accepting path. Since the weight of a single path is the product of its edges, the reciprocal weight is computed by the same automaton with all its weights replaced by their reciprocals. Composing P_2 with its reciprocal marginal automaton finally gives us a transducer realizing the conditional distribution C (4.13). Our conditionalization algorithm , which carries out these operations implicitly,

```
// Input: parameter function θ of a joint memoryless transducer
d ← 0    // total deletion weight
for each σ ∈ Σ do
    d ← d + θ(σ, ϵ)
end for
r ← 1 − d    // r = 1/d*
for each γ ∈ Γ do
    t ← θ(ϵ, γ)    // marginal weight of γ
    for each σ ∈ Σ do
        t ← t + θ(σ, γ)
    end for
    n ← r/t    // n = 1/(d* t), the reciprocal of the marginal weight
    // conditionalize θ
    for each σ ∈ Σ ∪ {ϵ} do
        θ(σ, γ) ← θ(σ, γ) × n
    end for
end for
θ(#) ← r
return θ
```

Figure 4.11: The conditionalization algorithm for memoryless stochastic transducers.

is a straightforward extension of the marginalization algorithm. However, it crucially relies on the presence of multiplicative inverses, and therefore cannot be stated generically in terms of semirings. Attempts to generalize to semifields or division semirings would seem pointless, since the concept of a conditional distribution is closely tied to the nonnegative real semiring, which happens to be a commutative semifield. The conditionalization algorithm in Figure 4.11 is concrete and does not aim to generalize beyond the nonnegative reals.

We finish our running example by noting that the conditional model corresponding to the earlier joint and marginal models has the following parameter values:

$$\theta(s, f) = 0.36, \qquad \theta(\epsilon, f) = 0.54,$$
$$\theta(s, g) = 0.3, \qquad \theta(\epsilon, g) = 0.6,$$
$$\theta(s, \epsilon) = 0.1, \qquad \theta(\#) = 0.9.$$

If we plug these parameter values into Figure 4.6 in place of the values of the joint model, we can now calculate the conditional probability of any string $x \in \{s\}^*$ given fg directly without first computing the marginal probability of fg. The conditional distribution $\langle x, y \rangle \mapsto P_2(x \mid y, \theta)$, where θ is the conditionalized parameter function, can be evaluated for a specific pair of strings by the Forward algorithm, just like the joint distribution P_2. The conditional probabilities of the first five strings in $\{s\}^*$ (ordered lexicographically) are the same as those calculated previously on page 143.

Furthermore, it is easy to check that the distribution $x \mapsto P_2(x \mid y)$ sums to one for every choice of y: simply apply the marginalization algorithm from Figure 4.9 to the conditional transducer. The resulting marginal automaton then represents the function $y \mapsto \sum_{x'} P_2(x' \mid y)$ which is equal to the constant function $y \mapsto 1$ because all parameters of the marginal automaton are set to unity by the marginalization algorithm.

In conclusion, we have seen how to obtain conditional memoryless transducers from joint memoryless transducers. Conditional models play an important part as channel models that can be combined with source models to form new joint models. The importance of this question is not adequately reflected in the

HMM literature. Ristad and Yianilos [1998, p. 526] mention in passing that conditional distributions are needed for string classification, but they do not provide any discussion of conditionalization and/or marginalization of stochastic transducers. The problem of obtaining conditional from joint models should in fact be considered as an important fourth problem in addition to the three Ferguson–Rabiner problems [Rabiner, 1989] for HMMs. Of those, the one remaining problem we have not discussed yet concerns decoding, which we turn to next.

4.5 Using a Joint Model for Prediction

The problem of predicting a string $x \in \Sigma^*$ for a given string $y \in \Gamma^*$ had been defined earlier as the optimization problem looking for the "best" string

$$x^\star = \operatorname*{argmax}_x \; P_2(x, y) = \operatorname*{argmax}_x \; P_2(x \mid y).$$

This is also known as maximum a posteriori (MAP) decoding, especially in the speech recognition literature [Rabiner, 1989; Jelinek, 1997]. While it is possible to adapt the techniques used in speech recognition and specialize them to the present case [Luk and Damper, 1998], the following discussion again employs the very general perspective of weighted automata and graphs.

Consider again the transducer in Figure 4.6. It compactly represents all strings $x \in \Sigma^*$ corresponding to the given string $fg \in \Gamma^*$, together with their joint probabilities. For the purpose of decoding, the given string fg can be ignored, as one focuses on the first component of the labels. Finding the best *string* $x \in \Sigma^*$ is a hard problem: although the probability of a single string can be computed efficiently by the Forward algorithm, in general one might have to examine exponentially many strings in order to find the best hypothesis. Finding the most likely string of

an HMM is known to be **NP**-hard [Goodman, 1998; Casacuberta and de la Higuera, 2000], and the proof carries over essentially unchanged to stochastic transducers.

In a certain sense, the present situation poses a dilemma not unlike the problem of parameter estimation: there (Section 4.3) we wanted to estimate the parameters of the sampling distribution P_2, but could evaluate their likelihood most easily in terms of P_1, which required information about alignments. Here, it is easy to find best paths, corresponding to single alignments, but we would really like to find best strings, corresponding to sums over potentially exponentially many alignments. This would be possible if these sums were taken over just a single alignment, which is the case if the transducer is unambiguous, i. e., there is at most one successful path for each string. One way to ensure that this is the case involves determinization, but since determinization of automata through subset construction is an exponential time algorithm in the worst case, this may not be feasible for large machines. Moreover, not all weighted automata are determinizable [Mohri, 1997].

In practice, the decoding problem is therefore usually solved heuristically, often by finding the n best paths without determinization and then aggregating the probabilities of paths with identical string labels [Mohri and Riley, 2002]. In the simplest case, known as the Viterbi approximation, we have $n = 1$ (however, larger values of n do not make the problem much harder), so the problem boils down to finding the single most likely path, whose label is then used as the best string hypothesis.

The problem of finding the most likely path can be expressed most naturally in the max–times semiring over the nonnegative real numbers, which is the structure $\mathbb{R}^{\geq 0}_{\max,\times} = \langle \{x \in \mathbb{R} \mid x \geq 0\} \cup \{\infty\}, \max, \times, 0, 1 \rangle$ (this is similar to Goodman's [1999] "Viterbi semiring", although the carrier sets are different). The key

difference compared with the nonnegative real semiring is that $\oplus$-addition now selects the greater of two numbers, rather than adding them. Since this operation is used to summarize sets of paths with common end points, it finds the most likely path in the set. Note that the nonnegative max–times semiring is closed , with $a^* = \infty$ for $a > 1$, and $a^* = 1$ for $0 \leq a \leq 1$. If weights are probabilities, this last condition is always true, and therefore the most likely path is both finite and simple. The nonnegative max–times semiring is isomorphic to the max–plus semiring $\mathbb{R}_{\max,+} = \langle \mathbb{R} \cup \{+\infty, -\infty\}, \max, +, -\infty, 0 \rangle$ (also known as the *polar* semiring), as well as the tropical semiring, by taking (negative) logarithms under the additional convention that $\log 0 = -\infty$ and $\log(+\infty) = +\infty$.

As pointed out on page 140, the automata representing marginal weights of memoryless weighted transducers are necessarily almost-acyclic. This means that we can find the weight of the most likely path of weighted machines like the example from Figure 4.6 by computing the algebraic path from the start state to the unique (w. l. o. g.) sink state using an instantiation of the generic single-source algebraic path algorithm for almost-acyclic graphs (Figure 4.8) in the max–times semiring (or a semiring isomorphic to it, after a suitable transformation of weights). Even simpler, since weights are probabilities, all closures are equal to 1, which means that loops have weight $\bar{1}$ and can therefore be completely ignored. But removing loops from an almost-acyclic graph results in an acyclic graph, which means that the standard DAWG shortest path algorithm [Cormen et al., 1990, sec. 25.4] is applicable. (This simplification is only possible when searching for the single most likely path; if we are interested in the n best paths for $n > 1$ the algorithm in Figure 4.8 can still be used.) The max–times semiring has the additional property that $a \oplus b \in \{a, b\}$, which makes it possible to reconstruct the most likely path in addition to computing its weight (if all we were

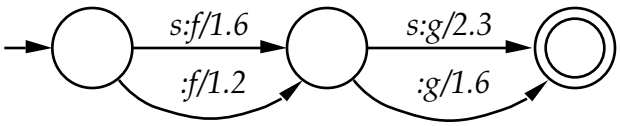

Figure 4.12: A DAWG resulting from the neg-log transform of the transducer displayed in Figure 4.6.

interested in was the probability of the most likely path we could have also used the generic marginalization algorithm from Figure 4.9 instantiated in the max–times semiring).

In general, any other applicable shortest path algorithm can be used for performing a blind search, and more sophisticated algorithms for guided search have also been formulated [see Jelinek, 1997]. Since the semirings used by some shortest path algorithms are more restrictive [Fink, 1992; Mohri, 2002c] than those defined here, the path weights usually have to be transformed and lifted into different semirings before decoding. For example, Dijkstra's algorithm [Cormen et al., 1990, sec. 25.2] requires nonnegative additive weights and a semiring with additional properties [Fink, 1992]. When starting from probabilities this typically means working with their negative logarithms. This is not strictly required in the present case, since, as mentioned above, the relevant graphs can always be considered to be acyclic, but it is useful for numerical stability and is required in the general setting discussed in Section 5.5. Applying the negative log transformation to Figure 4.6 results in the graph shown in Figure 4.12.

The transformed weights are understood to belong to the tropical semiring $\mathbb{R}^{\geq 0}_{\min,+}$. For the same reasons discussed before, the loops present in Figure 4.6 are absent in Figure 4.12, since in all cases we consider shortest paths are necessarily simple paths. Similarly absent is the final weight of the unique final state, since its presence or absence does not influence the search results.

Observe that the shortest path has labels $\langle \epsilon, f \rangle \langle \epsilon, g \rangle$, which means that the heuristically best hypothesis in $\{s\}^*$ is the empty string ϵ. However, the earlier discussions surrounding Figure 4.6 (see page 143 in particular) had established that s is the most likely string. This illustrates the fact that the heuristic solution which examines only the single most likely path may return suboptimal hypotheses. Clark [2001, fig. 6.5] provides another example in support of the same point.

In fact, there is a systematic problem here: the most likely path is necessarily simple and its edge labels either mention a symbol or the empty string, which means that the heuristic hypothesis is a string that cannot be longer than the longest (in terms of edges, not of edge weights) simple path. In other words, when sampling only the single most likely path, the heuristically decoded string is at most as long as the corresponding given string. This problem never surfaces in applications of HMMs such as part-of-speech assignment, since there the decoded sequence (say, part-of-speech labels) is precisely as long as the given input sequence (for example, words without part-of-speech labels). Nor is it a major problem for letter-to-sound rules, since relatively few phoneme strings are longer than their corresponding letter strings.

However, it is not difficult to construct examples that illustrate the general problem. Consider the following parameter vector for a joint model:

$$\begin{aligned} \theta(s, f) &= 0.04, & \theta(\epsilon, f) &= 0.01, \\ \theta(s, g) &= 0.04, & \theta(\epsilon, g) &= 0.01, \\ \theta(s, \epsilon) &= 0.8 & \theta(\#) &= 0.1. \end{aligned}$$

The rest of the task remains unchanged: we still want to find the most likely string in Σ^* that corresponds to $fg \in \Gamma^*$. As before, the longest simple path has length 2, but now the most likely string (by a small margin) is s^9 of length 9. In a situation like this where the deletion probability $\theta(s, \epsilon)$ dominates, one must sample a fair number of paths in order to find the most likely string. Fortunately, letter-to-sound rules are well-behaved in this regard, and we will either use the most likely path as a heuristic solution, or employ determinization in order to find the most likely string directly.

Summing up, we have seen that decoding with memoryless transducers poses a few difficult challenges, despite the apparent simplicity of the models. Decoding should properly be viewed as finding the most likely string, but as there do not seem to be efficient algorithms for this one has to resort to efficient heuristics in practice. The simplicity of memoryless transducers does not appear to make the MOST-LIKELY-STRING problem, which is **NP**-hard for HMMs and hence for general stochastic transducers, any easier for this special case. In light of the last example presented in this section, we conjecture that the MOST-LIKELY-STRING problem remains **NP**-hard for memoryless transducers, but we do not have a proof of this claim.

4.6 Conclusion

This chapter proposed the use of stochastic rational transductions for modeling letter-to-sound and other correspondences. Stochastic transductions are generally not limited to same-length relations and are therefore suitable for working with unaligned data. If alignments are not known, one is forced to average or sum over all alignment possibilities, for example when computing the probability of two strings corresponding to each other or during parameter estimation. Doing this is fairly straightforward for memoryless transductions in the model due to Ristad and Yianilos [1998]. We reviewed their approach and positioned it within a unifying algebraic framework. While much of our discussion was a review of existing results, every section also contained extensions that Ristad and Yianilos [1998] had only hinted at or completely ignored.

In Section 4.2 the Forward and Backward algorithms for memoryless stochastic transducer were presented (correcting a few mistakes in Ristad and Yianilos's [1998] presentation) in an algebraic framework in which they turned out to be abstractly identical to the classic dynamic programming algorithm for computing string edit distances [Wagner and Fischer, 1974].

The key contribution of Ristad and Yianilos [1998] was to point out that parameter estimation for memoryless stochastic transducers can be viewed as an instance of the EM algorithm. However, they did not provide a derivation of their algorithm from the EM theorem. Section 4.3 filled in the missing details in a way that can easily be generalized, which will be the topic of Section 5.3.

In addition to the well-known Ferguson–Rabiner Problems for HMMs (evaluation, decoding, estimation), an additional fourth problem – deriving marginal

and/or conditional models from a joint model – was introduced, and corresponding algorithms for memoryless transducers were presented in Section 4.4. These algorithms are a novel extension to Ristad and Yianilos's [1998] approach.

In Section 4.5 we discussed decoding for memoryless stochastic transducers and pointed to a an open issue: whereas the MOST-LIKELY-STRING problem for HMMs is known to be **NP**-hard [Goodman, 1998; Casacuberta and de la Higuera, 2000], is it also **NP**-hard for memoryless stochastic transducers? An affirmative answer seems likely. For letter-to-sound rules a simple approximate decoding scheme (so-called Viterbi decoding) may be an option, but it has a systematic weakness and more robust approximations should be used for other applications in which the decoded output strings are expected to be much longer than the corresponding input strings.

We had pointed out throughout this chapter that many of the algorithms presented here are not necessarily restricted to memoryless transducers. Ristad and Yianilos [1998] too had hinted at possible generalizations to richer transducer topologies. The next chapter extends the current approach to the class of stochastic rational transducers.

CHAPTER 5

LEARNING GENERAL STOCHASTIC TRANSDUCERS

5.1 Introduction

Working within the framework developed in Chapter 4 for memoryless transductions, this chapter generalizes the previous approach and applies it to stochastic rational transducers with arbitrary state graphs. The discussion of simple memoryless transductions in the preceding chapter introduced four fundamental problems for stochastic transducers, namely evaluation, estimation, conditionalization, and decoding. The structure of the present chapter closely resembles that of Chapter 4, as we address each of these four problems for general stochastic transducers in turn.

Evaluation, decoding and estimation are closely related to analogous problems for Hidden Markov Models, which are usually presented as the Three Fundamental Problems for HMMs, following Ferguson and Rabiner [Rabiner, 1989]. Evaluation, marginalization and decoding for stochastic transducers are also familiar from approaches to natural language processing based on weighted finite automata [Mohri et al., 1996; Mohri, 1997; Pereira and Riley, 1997]. The previous sections introduced some of the algebraic techniques for working with weighted

automata, and we saw generic algorithms that apply across many different semirings and can be specialized in the real semiring, which is the natural place for manipulating probabilities.

The problem of parameter estimation for stochastic transducers has received much less attention. As mentioned at the beginning of this chapter, the applicability of the EM algorithm may have been known for quite some time and is perhaps implicitly present in the work of Gilloux [1991] and Parfitt and Sharman [1991]. Ristad and Yianilos [1998] provided concrete algorithms for the special case of memoryless transducers, which have a one-state topology, including a parameter estimation algorithm which they identified as an instance of the EM algorithm. The precise relationship, which they did not provide details on, was discussed in Section 4.3. At around the same time, Durbin et al. [1998, ch. 4] introduced another special case of stochastic transducers, which they refer to as Pair HMMs, with a slightly richer three-state topology (which, incidentally, resembles the filter transducer used by Mohri et al., 1996). Also Durbin et al. [1998] make it clear how their algorithms derive from the EM algorithm. More general Pair HMMs were applied to natural language processing tasks by Clark [2001, ch. 6].

Parameter estimation algorithms for general stochastic transducers have only begun to emerge fairly recently [Clark, 2001; Eisner, 2001, 2002], and many details still remain to be worked out. What makes working with general stochastic transducers conceptually harder is the fact that not only is the state sequence the machine passes through "hidden" as in HMMs, even if the identity of the states is not in question there is an additional layer of unobserved information in the form of alignments. For example, memoryless transducers have precisely one state, so there is never any doubt about what state the machine is in, but the precise sequence of edges the machines passes through when it generates a pair of strings

is nevertheless hidden. It is this dependency on two kinds of hidden information that makes formulating algorithms for stochastic transducers challenging. For example, the algorithm for calculating expected edge counts given by Clark [2001, sec. 6.4.3] looks rather daunting at first sight, due to multiple dependencies on alignments, states, and symbols. By contrast, the algebraic framework used by Eisner [2001, 2002] is very elegant and allows for high-level abstractions, but may also obscure the true complexity of his algorithms.

Like Eisner, we view stochastic transducers as special cases of weighted finite transducers. Weighted machines will be described in the algebraic framework of closed semirings used earlier in this chapter. This allows for many useful generalizations [Mohri et al., 1996; Goodman, 1999] and the formulation of generic algorithms that have uses beyond the present application. Weighted automata generalize both stochastic automata, as well as traditional unweighted machines.

Stochastic generalizations of automata have been studied since the early days of formal language theory [Rabin, 1964]. The move from traditional unweighted automata to weighted automata is fairly straightforward, as "unweighted" automata can be seen as weighted automata over the Boolean semiring. The Boolean semiring is simply a two-element Boolean algebra, which, like all distributive lattices, satisfies the semiring axioms (Definition 4.2). Many classical algorithms for manipulating automata that were stated in terms of Boolean operations can be generalized to arbitrary closed semirings. However, there are exceptions for which generalizations are not straightforward or do not extend all the way to arbitrary semirings, but only to restricted subclasses.

When the weights of an automaton represent probabilities, they are treated as belonging to the nonnegative real semiring. This leads to a few complications, because a number of generic algorithms [see for example Cormen et al., 1990,

sec. 26.4] require semirings which are closed and whose $\oplus$-addition operation is idempotent, i. e., it has to be the case that $a \oplus a = a$ for all $a \in \mathbb{K}$. However, the nonnegative real semiring is not closed (see page 144) and addition of real numbers is not idempotent. There is not much one can do about the latter, though it turns out that generic algorithms exist for our purposes that do not require $\oplus$-additive idempotence. Extending the nonnegative real semiring with a new element ∞ yields an extension which is formally closed. Fletcher [1980] defines such a structure, which we refer to as the *closed nonnegative real semiring*, as follows:

$$\mathbb{K} = \mathbb{R}^{\geq 0} \cup \{\infty\} \qquad a^* = \begin{cases} 1/(1-a) & \text{if } a < 1 \\ \infty & \text{if } a \geq 1 \end{cases}$$

$$\oplus = + \qquad \bar{0} = 0$$

$$\otimes = \times \quad (\text{with } 0 \times \infty = \infty \times 0 = 0) \qquad \bar{1} = 1$$

This is similar to Goodman's [1999] "inside semiring", except that Goodman requires all semirings to be ω-continuous [see Kuich, 1997], which may have implications for the definition of closure that he does not discuss.

For practical applications it is often advantageous to work with logarithmically transformed numbers, since weights representing probabilities can be very close to zero and underflow their floating point representations [see Durbin et al., 1998, sec 3.6 for practical advice].

The following structure, which is called the *log semiring* is isomorphic to the closed nonnegative real semiring by taking negative logarithms:

$$\mathbb{K} = \mathbb{R} \cup \{+\infty, -\infty\} \qquad a^* = \begin{cases} \log(1 - \exp(-a)) & \text{if } a > 0 \\ -\infty & \text{if } a \leq 0 \end{cases}$$

$$a \oplus b = -\log(\exp(-a) + \exp(-b)) \qquad \bar{0} = +\infty$$

$$\otimes = + \qquad \bar{1} = 0$$

Note that one must require $(+\infty) + (-\infty) = (-\infty) + (+\infty) = +\infty$.

The $\oplus$-operation of the log semiring is somewhat expensive to compute [see however Durbin et al., 1998, sec. 3.6], and it is often reasonable to compute $a \oplus b$ as $\min(a, b)$, especially if $a \ll b$ or $b \ll a$. If we set $\oplus = \min$ we obtain the tropical semiring, which we had encountered for the first time on page 121. Observe that the tropical semiring is closed, as closure can be defined as $a^* = 0$ if $a \geq 0$, and $a^* = -\infty$ if $a < 0$.

The plan for the rest of this chapter is as follows. In Section 5.2 the problem of evaluating the mass function of general stochastic transducers is discussed. This is a generalization of the problem from Section 4.2 to arbitrary transducer topologies. Section 5.3 concern parameter estimation, which, as pointed out earlier in Section 4.3, is mostly about computing expected edge counts. Eisner's [2002] approach to computing these expectations is reviewed and compared with our approach in Section 5.3. Marginalization and decoding would both benefit from determinization, which is not always possible in general weighted automata. A necessary precondition for determinization which can always be satisfied is that

a given machine be ϵ-free. A large portion of Section 5.4 is therefore about ϵ-removal. Finally, Section 5.5 concerns decoding, but unlike Section 4.5 it discusses the concept of minimizing the risk or expected loss of a hypothesis.

5.2 Evaluating the Mass Function of a Joint Model

The generic Forward algorithm (Figure 4.1) from Section 4.2 was used to evaluate the probability mass function of a memoryless joint distribution. The simple joint models used up to this point could all be represented by weighted transducers with a one-state topology. But this restriction can easily be removed, and we will now consider general stochastic transducers with arbitrary topologies, represented as weighted finite state transducers whose weights are taken in the closed nonnegative real semiring (or, equivalently, the log semiring). We begin with a review of weighted finite transducers using notation and terminology commonly used in language and speech processing [Mohri et al., 1996; Mohri, 1997; Pereira and Riley, 1997].

Definition 5.1 (Weighted transducer). A weighted finite transducer over a closed semiring $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ is a tuple $\langle Q, \Sigma, \Gamma, q_s, F, E, \rho \rangle$, where Q is a finite set of states or vertices, Σ and Γ are finite nonempty sets (alphabets), $q_s \in Q$ is the initial state or start state, $F \subseteq Q$ is the set of final or accepting states, $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ is a finite set of edges or transitions, and $\rho : F \to \mathbb{K}$ is the final weight function.

The following notational conventions will be useful: given a transition $e = \langle q, \sigma, \gamma, k, q' \rangle$ we call q the source state of e and write $s(e) = q$; similarly, $t(e) = q'$ is the target state of e; we say that e bears the label $\langle \sigma, \gamma \rangle$ and denote this by $h(e) = \langle \sigma, \gamma \rangle$, as well as $h_1(e) = \sigma$ and $h_2(e) = \gamma$; finally, e has weight $w(e) = k$.

We assume w. l. o. g. that $\mathrm{w}(e) \neq \bar{0}$, since otherwise this edge could be removed without changing the behavior of the transducer.

A path a of length $|a| = n$ (possibly zero) is an alternating sequence of $n+1$ states and n edges $\langle q_1, e_1, \ldots, q_n, e_n, q_{n+1} \rangle$ such that $\mathrm{s}(e_i) = q_i$ and $\mathrm{t}(e_i) = q_{i+1}$ for all i ($1 \leq i \leq n$). Abusing notation, we write $\mathrm{s}(a) = q_1$ for the source vertex of the path, and $\mathrm{t}(a) = q_{n+1}$ for its target vertex; the label of a is defined to be $\mathrm{h}(a) = \langle \mathrm{h}_1(a), \mathrm{h}_2(a) \rangle$, where $\mathrm{h}_j(a) = \mathrm{h}_j(e_1) \cdots \mathrm{h}_j(e_n)$ for $j \in \{1, 2\}$; and the weight of a is defined to be $\mathrm{w}(a) = \bigotimes_{i=1}^{n} \mathrm{w}(e_i)$. Note that the empty path at state q is $a = \langle q \rangle$, and therefore $|a| = 0$, $\mathrm{s}(a) = \mathrm{t}(a) = q$, $\mathrm{h}(a) = \langle \epsilon, \epsilon \rangle$, and $\mathrm{w}(a) = \bar{1}$.

Furthermore, some of this notation is extended to sets of paths from a common source vertex q to a common target vertex q' (not necessarily distinct). If A is such a set, we write $\mathrm{s}(A) = q$, $\mathrm{t}(A) = q'$, and define its weight to be $\mathrm{w}(A) = \bigoplus_{a \in A} \mathrm{w}(a)$.

One can assume w. l. o. g. that the number of final states of a transducer is at most one. If it exceeds one, add a new state $q_\mathrm{t} \notin Q$ to Q, for each $q \in F$ add an edge $\langle q, \epsilon, \epsilon, \rho(q), q_\mathrm{t} \rangle$ to E, and then set $F = \{q_\mathrm{t}\}$ and $\rho(q_\mathrm{t}) = \bar{1}$. The previous notation can then be extended to apply to a transducer $\mathscr{T}$: let $\mathrm{s}(\mathscr{T})$ denote the initial state of $\mathscr{T}$, and $\mathrm{t}(\mathscr{T})$ its unique final state (transducers without final states can be ignored, since they all realize a trivial empty transduction).

A weighted transducer can be viewed as a weighted graph $\mathscr{G}$ by ignoring all labels. In this case it makes sense to talk about the total weight of all paths from a vertex q to another vertex q' (not necessarily distinct): if A is that set, then $\mathrm{w}(q, q')$ stands for $\mathrm{w}(A)$. Furthermore, the weight of the entire weighted graph $\mathscr{G}$ is then defined to be the total weight of all paths from its unique initial vertex to its unique (w. l. o. g.) final vertex, namely $\mathrm{w}(\mathscr{G}) = \mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{t}(\mathscr{G}))$.

The behavior of a weighted transducer is best conceptualized as a function $\Sigma^* \times \Gamma^* \to \mathbb{K}$ where $\mathbb{K}$ is the carrier set of the semiring algebra that provides the operations on the transducer's weights. A transducer $\mathscr{T}$ assigns a weight $\mathscr{T}(x, y)$ to a pair of strings $\langle x, y \rangle \in \Sigma^* \times \Gamma^*$, which is the total weight of all accepting paths (from the initial state to the final state) labeled with $\langle x, y \rangle$, formally:

$$\mathscr{T}(x, y) = \bigoplus_{\substack{a \\ \mathrm{s}(a)=\mathrm{s}(\mathscr{T}) \\ \mathrm{t}(a)=\mathrm{t}(\mathscr{T}) \\ \mathrm{h}(a)=\langle x,y \rangle}} \bigotimes_{i=1}^{|a|} a_i$$

One way of computing this would be to construct a weighted graph $\mathscr{G}$ whose paths coincided precisely with the accepting paths labeled with $\langle x, y \rangle$ of $\mathscr{T}$, and then to compute $\mathrm{w}(\mathscr{G})$. In order to carry out this construction, we need to introduce the concept of transducer composition.

Composition of weighted finite transducers is an operation which combines two weighted transducers $\mathscr{T}$ and $\mathscr{U}$ over the same semiring into a new transducer $\mathscr{T} \circ \mathscr{U}$ such that

$$[\mathscr{T} \circ \mathscr{U}](x, z) = \bigoplus_{y} \mathscr{T}(x, y) \otimes \mathscr{U}(y, z). \tag{5.1}$$

This is reminiscent of matrix multiplication, and generalizes relational composition. Algorithms for constructing a transducer realizing $\mathscr{T} \circ \mathscr{U}$ from transducers $\mathscr{T}$ and $\mathscr{U}$ are well known [Mohri et al., 1996; Pereira and Riley, 1997; Mohri et al., 2000], and are not completely trivial in the general case of semirings with non-idempotent $\oplus$-addition.

The transducers considered below are typically of the form $\mathscr{X} \circ \mathscr{T} \circ \mathscr{Y}$, where $\mathscr{X}$ is a transducer that recognizes only a single string, and so is $\mathscr{Y}$. More precisely,

such a transducer recognizes the string x if it maps the pair $\langle x, x \rangle$ to $\bar{1}$ and maps all other pairs to $\bar{0}$. We write $\mathrm{Str}(x)$ to denote a transducer with that behavior. Given a string $x \in \Sigma^*$, the transducer $\mathrm{Str}(x)$ can be constructed as follows: the set of states Q is the set of all prefixes of x; the two alphabets are identical, i. e., $\Sigma = \Gamma$; the initial state is the empty prefix ϵ; the final state is the trivial prefix x; the set of transitions is the set

$$\{\langle q, \sigma, \sigma, \bar{1}, q' \rangle \mid q, q' \in Q \wedge \sigma \in \Sigma \wedge q\sigma = q'\};$$

and the final weight of the final state x is $\rho(x) = \bar{1}$.

A memoryless transducer M_θ with parameter vector θ of the form considered earlier in this chapter takes the following form: the set of states Q is the singleton set $\{q\}$; the initial state is q, obviously; q is also a final state, i. e., $F = Q$; the set of transitions is the set

$$\{\langle q, \sigma, \gamma, \theta(\sigma, \gamma), q \rangle \mid \sigma \in (\Sigma \cup \{\epsilon\}) \wedge \gamma \in (\Gamma \cup \{\epsilon\})\};$$

and the final weight of q is $\rho(q) = \theta(\#)$.

5.2.1 Reconstruction of the Forward Algorithm

Transducers of the form $\mathrm{Str}(x) \circ M_\theta \circ \mathrm{Str}(y)$ correspond precisely to the Forward trellises encountered in Section 4.2. For example, the specific trellis in Figure 4.2 can be viewed as the result of transducer composition, namely as the transducer $\mathrm{Str}(ab) \circ M_\theta \circ \mathrm{Str}(stu)$ (only edge labels are shown in Figure 4.2; the weight of an edge labeled $\langle \sigma, \gamma \rangle$ is $\theta(\sigma, \gamma)$). To see why this is the case consider the behavior of

the transducer $\text{Str}(x) \circ \mathscr{T}$ for any transducer $\mathscr{T}$. By the definitions of composition and of $\text{Str}(x)$,

$$[\text{Str}(x) \circ \mathscr{T}](x', z) = \bigoplus_y \text{Str}(x)(x', y) \otimes \mathscr{T}(y, z) = \begin{cases} \mathscr{T}(x, z) & \text{if } x = x' \\ \bar{0} & \text{otherwise} \end{cases}$$

Suppose that $x = x'$. Then $\text{Str}(x)(x', y) = \bar{0}$ for all $y \neq x'$, so the $\oplus$-sum over all y has only one non-$\bar{0}$-zero term when $x = x' = y$, namely $\text{Str}(x)(x, x) \otimes \mathscr{T}(x, z)$, which is identical to $\mathscr{T}(x, z)$ since by definition $\text{Str}(x)(x, x) = \bar{1}$. Otherwise (if $x \neq x'$) it is always the case that $\text{Str}(x)(x', y) = \bar{0}$. We can use this argument twice to show that

$$[\text{Str}(x) \circ \mathscr{T} \circ \text{Str}(y)](x', y') = \begin{cases} \mathscr{T}(x, y) & \text{if } x = x' \wedge y = y' \\ \bar{0} & \text{otherwise} \end{cases}$$

The definition of a transducer's behavior entails that all accepting paths of a transducer realizing the composition $\text{Str}(x) \circ \mathscr{T} \circ \text{Str}(y)$ are labeled with $\langle x, y \rangle$. Furthermore, one can assume w. l. o. g. that these are the only paths through the transducer, since it assigns $\bar{0}$ to all other pairs of strings, which is the default if there are no accepting paths. But this means that we have restricted our attention to precisely the paths labeled $\langle x, y \rangle$, so that we can now ignore all labels and treat the weighted transducer $\text{Str}(x) \circ \mathscr{T} \circ \text{Str}(y)$ as a weighted graph $\mathscr{G}$ with the property that $\text{w}(\mathscr{G}) = \mathscr{T}(x, y)$.

If $\mathscr{T}$ is a memoryless transducer M_θ, then $\mathscr{G}$ has a trellis topology, as previously discussed. This means that the generic Forward algorithm can be seen as performing the following computations (note the similarity to Eisner's [2001] slogan "compose + minimize"):

1. Given strings x and y and a (memoryless) transducer $\mathscr{T}$, construct the underlying weighted graph $\mathscr{G}$ of the composed transducer $\mathrm{Str}(x) \circ \mathscr{T} \circ \mathrm{Str}(y)$;

2. Compute and return the vector of weights ("algebraic distances") of the form $\mathrm{w}(\mathrm{s}(\mathscr{G}), q)$ for all vertices q of $\mathscr{G}$.

The generic Forward algorithm is very efficient because the composition in the first step does not have to be performed explicitly as the topology of $\mathscr{T}$ is known, and the single-source algebraic path problem at the core of the second step can be solved in linear time on acyclic graphs like the Forward trellises. Note that the vertices of $\mathscr{G}$ are triples $\langle x', q, y' \rangle$ where x' is a prefix of x, q is the single state of the memoryless transducer $\mathscr{T}$, and y' is a prefix of y. By mapping such a triple to a pair $\langle |x'|, |y'| \rangle$, we obtain the indices of the matrix α that the generic Forward algorithm (Figure 4.1) constructs. In this sense, the distance vector introduced here contains the same information as the Forward matrix α.

5.2.2 Computing Forward and Backward Probabilities

The specific assumptions of the Forward algorithm are now easy to remove. For one thing, the topology of $\mathscr{T}$ may be arbitrary. If no restrictions are placed on $\mathscr{T}$, then the composition step generally has to be carried out explicitly. Furthermore, the transducers that $\mathscr{T}$ is composed with on either side do not have to be of the form $\mathrm{Str}(x)$. In an extreme case, they may be completely absent: for example, computing the total weight of $\mathscr{T} \circ \mathrm{Str}(y)$ when $\mathscr{T}$ is a joint stochastic transducer yields the marginal probability of the string y. In general the composed transducer may be of the form $\mathscr{X} \circ \mathscr{T} \circ \mathscr{Y}$ without any further restrictions. This is useful if we are interested in computing probabilities for non-basic events like the string x corresponding to either y or y'.

To compute such probabilities, we need to determine the total weight of the weighted graph $\mathscr{G}$ corresponding to the composed transducer. We cannot generally compute the weight of a graph by naive enumeration of all paths, since there are infinitely many paths if the graph has cycles. If the graph is acyclic, the number of paths is obviously finite, but it may still be exponential in the size of the graph, as we had seen earlier. We need to decompose the problem and exploit the algebraic properties of the semiring from which the weights are drawn. There are several ways of computing the total weight of the graph $\mathscr{G}$ in the second step, depending on what is known about $\mathscr{G}$ and/or its weight semiring. The following cases are important:

1. Acyclic graphs with a known topology. This is the case for the graphs constructed implicitly by the Forward and Backward algorithms. Since $\mathscr{X}$ and $\mathscr{Y}$ are acyclic and have a known topology and $\mathscr{T}$ has a trivial one-state topology, $\mathscr{X} \circ \mathscr{T} \circ \mathscr{Y}$ is acyclic and has a predictable trellis topology, which can then be left implicit. Moreover, topological orderings of the vertices of $\mathscr{G}$ are known a priori. The Forward and Backward algorithms explore the edges of $\mathscr{G}$ in a known topological order, but neither $\mathscr{G}$ nor a topological ordering have to be constructed explicitly. Because $\mathscr{G}$ is acyclic, any semiring can be used, as closure is not required. It is clear, based in part on the discussion in Section 4.2, that the overall running time is linear in the size of $\mathscr{G}$.

   Though slightly more complex, the topologies of the Pair HMMs studied by Durbin et al. [1998, ch. 4] (see also the related works cited by Clark [2001, sec. 6.5.1]) are fixed and known a priori too, so their algorithms also fall in this class.

2. Acyclic graphs whose topology is not known a priori. The main difference compared with the Forward algorithm is that implicit representations can no longer be used. Composition has to be performed explicitly, and the vertices of $\mathscr{G}$ must be topologically sorted, as the single-source algebraic path for DAWGs explores the vertices of $\mathscr{G}$ in topological order. The asymptotic running time is still linear in the size of $\mathscr{G}$ [Cormen et al., 1990, sec. 25.4], but the proportional overhead is slightly larger compared with the first case.

3. Almost-acyclic graphs. We had seen these in Section 4.4, arising in the computation of marginal weights. More precisely, we had argued there, although not in these exact terms, that the composition $M_\theta \circ \mathrm{Str}(y)$ of a memoryless transducer M_θ with a transducer $\mathrm{Str}(y)$ accepting a single string is necessarily almost-acyclic. The algorithm in Figure 4.8 can be used to compute the weight of $\mathscr{G}$, but it assumes that $\mathscr{G}$ is represented by a triangular adjacency matrix. This representation can be obtained by topologically sorting the vertices of $\mathscr{G}$ while ignoring any loops. Due to the presence of loops, the semirings should be closed, or at the very least the closure operation must be defined for the weights of all loops of $\mathscr{G}$.

4. Arbitrary graphs over k-closed semirings, for which a generic single-source algebraic path algorithm is applicable [Mohri, 1998, 2002c]. In the framework developed by Mohri, a semiring is k-closed if $\bigoplus_{i=1}^{k+1} a^i = \bigoplus_{i=1}^{k} a^i$ for all $a \in \mathbb{K}$. Intuitively, this means that the weight of a cycle can be determined by traversing it k times, but traversing it more often does not change the total weight. The min–plus semiring over the nonnegative reals is 0-closed, because $a^0 \oplus a^1 = a^1$ is the same as $\bar{1} \oplus a = \bar{1}$, which in this specific semiring means $\min(0, a) = 0$. This is always true, because a is nonnegative.

The closed nonnegative real semiring, however, is not k-closed for any natural number k, because generally $\sum_{i=1}^{k+1} a^i \neq \sum_{i=1}^{k} a^i$ for $a \in \mathbb{R}^{\geq 0}$. However, for $a \in [0;1)$ and sufficiently large k, one may say that $\sum_{i=1}^{k+1} a^i \approx \sum_{i=1}^{k} a^i$, and Mohri's algorithm can therefore be used as an approximation when the weights of cycles represent probabilities, which fall into the interval $[0;1)$.

5. Arbitrary graphs over arbitrary closed semirings. In this most general case, the graphs need not be acyclic and the closed semirings need not be k-closed or additively idempotent. In fact, the semirings need not even be closed, as long as the closure operation is defined for the weights of all cycles.

   No single-source algebraic path algorithms are known. However, certain variants of the Floyd–Warshall all-pairs shortest-path algorithm [Warshall, 1962; Floyd, 1962] are applicable. Generalizations that work with idempotent semirings are well known [Aho et al., 1974; Cormen et al., 1990], and the requirement for an idempotent $\oplus$-operation can in fact be dropped [Fletcher, 1980].

   The generalized Floyd–Warshall all-pairs algebraic path algorithm solves many related graph problems, all of which ask for the weight of a graph. The interpretation of that weight depends on the semiring used. At least the following specializations of this algorithm and corresponding problems (and semirings) are known:

   (a) Kleene's algorithm for converting a finite automaton to a regular expression (semiring of regular languages over a fixed fixed alphabet Σ: let $\mathbb{K}$ be the set of regular languages over Σ, $\oplus$ is set union, $\otimes$ is concatenation of languages, $\bar{0}$ is the empty set, and $\bar{1}$ is the set $\{\epsilon\}$);

(b) Warshall's algorithm [Warshall, 1962] for computing the transitive closure of graphs (Boolean semiring);

(c) Floyd's algorithm [Floyd, 1962] for computing all-pairs shortest paths (tropical semiring);

(d) The Gauss–Jordan algorithm for matrix inversion with pivoting (real semiring).

Since we are dealing with non-k-closed non-idempotent semirings over potentially cyclic graphs, we present the generalized Gauss–Jordan–Kleene–Floyd–Warshall–Aho–Hopcroft–Ullman–Lehmann–Fletcher all-pairs algebraic path algorithm in some detail. Unfortunately, many standard presentations of this algorithm [Aho et al., 1974; Cormen et al., 1990] are too specific. For one thing, they assume [Cormen et al., 1990, p. 570] that the $\oplus$-operation of the semiring is additively idempotent, but this assumption can easily be dropped [Fletcher, 1980] without major modifications to the algorithm. They also assume that the semiring is closed, which requires the closure operation of the semiring to be a total function; but in fact one can work with a partial closure operation, as long as all closures that arise during the execution of the algebraic path computation are well-defined.

A version of the generalized Floyd–Warshall algorithm that does not require additive idempotence of the weight semiring is shown in Figure 5.1. As Fletcher [1980] points out, this algorithm differs from the version that requires an idempotent semiring only in terms of when the identity matrix is added: lines 10–12 in Figure 5.1 compute $L^{(n)} \leftarrow L^{(n)} \oplus \bar{I}$ (where $\bar{I}$ is the $n \times n$ identity matrix

1: // Input: $n \times n$ adjacency matrix M of a weighted graph
2: $L^{(0)} \leftarrow M$ // CLR: $L^{(0)} \leftarrow M \oplus \bar{I}$
3: **for** $k \leftarrow 1$ **to** n **do**
4: **for** $i \leftarrow 1$ **to** n **do**
5: **for** $j \leftarrow 1$ **to** n **do**
6: $L^{(k)}[i,j] \leftarrow L^{(k-1)}[i,j] \oplus \left(L^{(k-1)}[i,k] \otimes (L^{(k-1)}[k,k])^* \otimes L^{(k-1)}[k,j]\right)$
7: **end for**
8: **end for**
9: **end for**
10: **for** $i \leftarrow 1$ **to** n **do**
11: $L^{(n)}[i,i] \leftarrow L^{(n)}[i,i] \oplus \bar{1}$
12: **end for**
13: **return** $L^{(n)}$

Figure 5.1: The generalized Floyd–Warshall all-pairs algebraic path algorithm.

with $\bar{1}$ along the main diagonal and $\bar{0}$ everywhere else), whereas the COMPUTE-SUMMARIES algorithm from [Cormen, Leiserson, and Rivest, 1990, p. 575] instead adds the identity matrix to $L^{(0)}$ on line 2 (see the comment 'CLR' in Figure 5.1).

As a result, the COMPUTE-SUMMARIES algorithm computes a slightly different distance matrix, which coincides with the true distance matrix on the additional assumption of idempotence. Consider the following example:

$$\begin{pmatrix} a & b \\ \bar{0} & c \end{pmatrix}$$

In a weighted graph with this adjacency matrix, the generalized distance between vertex 1 and vertex 2 is $a^* \otimes b \otimes c^*$. However, the CLR algorithm computes this

quantity as $(a \oplus \bar{1})^* \otimes b \otimes (c \oplus \bar{1})^*$, which is equal to the true distance if the semiring is idempotent (in order to prove the equality $(a \oplus \bar{1})^* = a^*$, show that $\bigoplus_{i=1}^{n} (a \oplus \bar{1})^i = \bigoplus_{i=1}^{n} a_i$ by induction on n, using the lemma $(a \oplus \bar{1})^n = \bigoplus_{i=0}^{n} a^i$, which can also be proved by induction). Fletcher's algorithm, on the other hand, computes the correct distance directly without relying on idempotence to save the day.

The running time of the generalized Floyd–Warshall algorithm is cubic in the number of vertices. More precisely if $T_\oplus$ is the running time of the $\oplus$-addition operation, $T_\otimes$ the running time of the $\otimes$-multiplication operation, and $T_{(\cdot)^*}$ the running time of the closure operation, then running time of the overall algorithm is $\Theta(n^3 (T_\oplus + T_\otimes) + n T_{(\cdot)^*})$. The space requirement is merely quadratic, since at any given point during the execution of the algorithm at most two $n \times n$ matrices are accessed, even though the algorithm constructs n such matrices in total. The space requirement can be further reduced in practice (though not asymptotically) by using a version of the algorithm that computes the closure of the adjacency matrix in place. A variant of the algorithm that uses in-place updates appears in Figure 5.2. Observe that the matrix is accessed one row at a time, so sparse representations like adjacency lists could be used.

Once $L^{(n)}$ has been computed, the vector α of distances from the graph's source vertex $\mathrm{s}(\mathscr{G})$ is the row of $L^{(n)}$ corresponding to the source vertex; and the vector β of distances to the target vertex $\mathrm{t}(\mathscr{G})$ is the column of $L^{(n)}$ corresponding to the graph's target vertex. The weight $\mathrm{w}(\mathscr{G})$ of the graph can be found in the cell of $L^{(n)}$ where the row vector α and the column vector β intersect. There does not appear to be a general algorithm for computing α and β without computing the entire distance matrix $L^{(n)}$. Therefore, when dealing with almost-acyclic graphs (which remain almost-acyclic when all edges are reversed) it is preferable

```
1:  // Input: n × n adjacency matrix L of a weighted graph
2:  for k ← 1 to n do
3:     b ← (L[k, k])*
4:     for i ← 1 to n do
5:        if i ≠ k and L[i, k] ≠ 0̄ then
6:           a ← L[i, k] ⊗ b
7:           for j ← 1 to n do
8:              L[i, j] ← L[i, j] ⊕ (a ⊗ L[k, j])
9:           end for
10:       end if
11:    end for
12:    // finish the case i = k
13:    if L[k, k] ≠ 0̄ then
14:       c ← 1̄ ⊕ (L[k, k] ⊗ b)
15:       for j ← 1 to n do
16:          L[k, j] ← c ⊗ L[k, j]
17:       end for
18:    end if
19:    L[k, k] ← L[k, k] ⊕ 1̄
20: end for
21: return L
```

Figure 5.2: The generalized Floyd–Warshall all-pairs algebraic path algorithm, using in-place updates.

to run the linear time single-source algebraic path algorithm for almost-acyclic paths twice, instead of using the cubic time all-pairs algebraic path algorithm for general graphs.

In conclusion, computing the probability mass that a general stochastic transducer assigns to a single basic event $\langle x, y \rangle$ (or to a set of events that can be described by finite transducers; this includes computing marginal probabilities) can be done by a weighted transducer composition followed by an algebraic distance computation using an appropriate algebraic path algorithm.

We consider this approach to be preferable to others that have been proposed in the literature, mainly because viewing it as a sequence of operations on finite transducers places it among other well understood problems, and it abstracts away from some of the complexities of the underlying problem. For example, we had noted earlier that the vertices of $\mathscr{G}$ resulting from the composition $\mathscr{X} \circ \mathscr{T} \circ \mathscr{Y}$ of three transducers are in general triples. This parallels the observations by Durbin et al. [1998, ch. 4] and Clark [2001, ch. 6] that a three-dimensional array must be used for a direct presentation of the dynamic programming schemes that extend the Forward algorithm for HMMs. The clarity of their presentation of their algorithms is somewhat reduced by the multiple dependencies between states and two strings or sequences that have to be tracked. The classic Forward algorithm and its generalization in Figure 4.1 interleave transducer composition and algebraic path computations, and while this is very elegant for memoryless transducers, it makes direct extensions and generalizations to arbitrary topologies rather cumbersome.

By contrast, the added conceptual complexity has remained hidden in our presentation, as there was never any need to inspect the internal structure of the vertices of $\mathscr{G}$. Moreover, since we separated the composition step cleanly from the algebraic distance computation, our approach can deal with arbitrary compositions, whereas Clark [2001] is restricted to compositions of the form $\mathrm{Str}(x) \circ \mathscr{T} \circ \mathrm{Str}(y)$. This separation into two steps comes with very little overhead, since composition can be carried out "on the fly" [Mohri et al., 2000], so that in practical implementations the two steps may end up interleaved again.

5.3 Estimating the Parameters of a Joint Model

Maximum likelihood parameter estimation for general stochastic transducers proceeds along the same general lines as for memoryless transducers. In our previous discussion in Section 4.3 we had already pointed out the generality of the parameter estimation algorithm presented by Ristad and Yianilos [1998]. As we had seen earlier, the central problem of EM-style parameter estimation is the computation of expected counts of edges, which is the topic of this section for the most general case.

5.3.1 Calculating Expected Counts

We had seen in the memoryless case (Section 4.3) that computing expected edge counts involves computing the Forward probabilities α and Backward probabilities β. In the general case, when a joint stochastic transducer $\mathscr{T}$ is trained on a sample $\langle x, y \rangle$, we can construct the composed transducer $\mathrm{Str}(x) \circ \mathscr{T} \circ \mathrm{Str}(y)$ and compute the distance matrix of its underlying graph $\mathscr{G}$ as in the preceding section, in order to obtain the distance vectors α and β.

Consider an accepting path a through the graph (from the source vertex $\mathrm{s}(\mathscr{G})$ to the target vertex $\mathrm{t}(\mathscr{G})$). Its weight $\mathrm{w}(a)$ can be viewed as the probability that a is the result of a random walk through the transducer, and its contribution to the weight of the graph (which is assumed to be finite) is the conditional probability $\mathrm{w}(a)/\mathrm{w}(\mathscr{G})$. As before, let $\mathrm{C}(e, a)$ denote the number of times that edge e occurs in path a. Recall that in general a need not be a simple path, so it is possible that $\mathrm{C}(e, a) > 1$ if the graph $\mathscr{G}$ has cycles. The expected number of occurrences of edge e, written $\gamma(e)$ following both our earlier usage and traditional HMM terminology, is defined in a manner analogous to equation (4.9) from Section 4.3:

$$\gamma(e) = \frac{1}{\mathrm{w}(\mathscr{G})} \sum_{\substack{a \\ \mathrm{s}(a)=\mathrm{s}(\mathscr{G}) \\ \mathrm{t}(a)=\mathrm{t}(\mathscr{G})}} \mathrm{w}(a)\ \mathrm{C}(e, a)$$

Define $\gamma'(e) = \gamma(e) \times \mathrm{w}(\mathscr{G})$. We know how to compute $\mathrm{w}(\mathscr{G})$ using an algebraic path algorithm as discussed in Section 5.2. The remaining question is how to compute $\gamma'(e)$. It is clear that we only need to consider paths that contain the specific edge e. Consider the set of all paths that contain at least one occurrence of edge e. The paths in this set are of the form depicted in Figure 5.3. In this figure, A stands for the set of all paths that do not include edge e and which originate at the designated source vertex $\mathrm{s}(\mathscr{G})$ (called s in the figure) and end at the vertex $\mathrm{s}(e)$. Edge e is shown in bold, and the dotted line labeled R represents the set (possibly empty) of all paths that do not include edge e and which lead from $\mathrm{t}(e)$ back to $\mathrm{s}(e)$. Finally, B is the set of all paths that do not include e leading from $\mathrm{t}(e)$ to the designated target vertex $\mathrm{t}(\mathscr{G})$ (called t in the figure). Clearly, any successful path through the graph $\mathscr{G}$ that passes along edge e will have a prefix drawn from set A, pass through e at least once, possibly return via a path in R and pass through

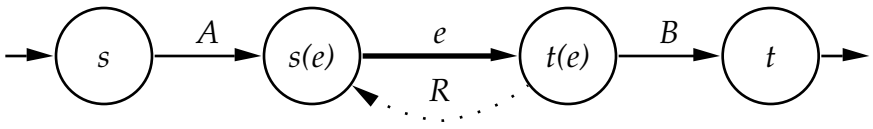

Figure 5.3: Schematic decomposition of paths that traverse edge e.

e again (this may be repeated a finite number of times), before reaching the sink vertex via a path in B.

Since in general it is possible for a path to contain more than one occurrence of edge e, we need to explicitly take the number of occurrences into account:

$$\gamma'(e) = \mathrm{w}(A) \times (1\,\mathrm{w}(e) + 2\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e) + \cdots) \times \mathrm{w}(B)$$

An easy special case arises when the set R is empty. Then $\mathrm{w}(R) = \mathrm{w}(\{\}) = 0$, and therefore

$$\gamma'(e) = \mathrm{w}(A) \times \mathrm{w}(e) \times \mathrm{w}(B).$$

Moreover, $R = \{\}$ implies that no path from the graph's source vertex $\mathrm{s}(\mathscr{G})$ that contains e can reach $\mathrm{s}(e)$ again after having traversed e (note that edge e cannot be a loop in this case, because then R would contain the empty path $\langle e \rangle$). In other words, the set A contains all paths that lead to $\mathrm{s}(e)$ from the graph's source vertex,

and therefore $\mathrm{w}(A) = \mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{s}(e))$. A symmetric argument leads to $\mathrm{w}(B) = \mathrm{w}(\mathrm{t}(e), \mathrm{t}(\mathscr{G}))$, and this allows us to conclude in this special case that

$$\gamma'(e) = \mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{s}(e)) \times \mathrm{w}(e) \times \mathrm{w}(\mathrm{t}(e), \mathrm{t}(\mathscr{G})).$$

Next we consider what happens when $R \neq \{\}$. This is the crucial case that arises only with cyclic transducers. Since we had assumed that edge weights are non-$\bar{0}$-zero, this means that $\mathrm{w}(R) \neq 0$. Therefore it is possible to rearrange the infinite sum corresponding to the $e\,R$ loop as follows:

$$\begin{aligned}
&1\,\mathrm{w}(e) + 2\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e) + 3\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e) + \cdots \\
&= \frac{1}{\mathrm{w}(R)} \times (1\,\mathrm{w}(e)\,\mathrm{w}(R) + 2\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e)\,\mathrm{w}(R) + \cdots) \\
&= \frac{1}{\mathrm{w}(R)} \times \sum_{i=1}^{\infty} i \times (\mathrm{w}(e)\,\mathrm{w}(R))^i \\
&= \frac{\mathrm{w}(e)}{(1 - \mathrm{w}(e)\,\mathrm{w}(R))^2}
\end{aligned}$$

The last equality is due to the fact that, for $|r| < 1$,

$$\sum_{i=1}^{\infty} i \times r^i = \frac{r}{(1-r)^2}.$$

We know that $|\mathrm{w}(e)\,\mathrm{w}(R)| < 1$, because otherwise $\mathrm{w}(\mathscr{G})$ would not be finite. Continuing with $\gamma'(e)$:

$$\begin{aligned}
\gamma'(e) &= \mathrm{w}(A) \times (1\,\mathrm{w}(e) + 2\,\mathrm{w}(e)\,\mathrm{w}(R)\,\mathrm{w}(e) + \cdots) \times \mathrm{w}(B) \\
&= \mathrm{w}(A) \times \frac{\mathrm{w}(e)}{(1 - \mathrm{w}(e)\,\mathrm{w}(R))^2} \times \mathrm{w}(B) \\
&= \mathrm{w}(A) \times \frac{1}{1 - \mathrm{w}(e)\,\mathrm{w}(R)} \times \mathrm{w}(e) \times \frac{1}{1 - \mathrm{w}(R)\,\mathrm{w}(e)} \times \mathrm{w}(B) \\
&= \mathrm{w}(A) \times (\mathrm{w}(e)\,\mathrm{w}(R))^* \times \mathrm{w}(e) \times (\mathrm{w}(R)\,\mathrm{w}(e))^* \times \mathrm{w}(B)
\end{aligned}$$

Because of the decomposability of sets of paths, any path from $\mathrm{s}(\mathscr{G})$ to $\mathrm{s}(e)$ can be spliced together from a prefix included in the set A and any number of continuations that first traverse edge e and then return to $\mathrm{s}(e)$ via a path (possibly an empty path) included in the set R. Because $\mathrm{w}(A)$ is the total probability of all paths that start at $\mathrm{s}(\mathscr{G})$, do not traverse e, and end at $\mathrm{s}(e)$, and $(\mathrm{w}(e) \times \mathrm{w}(R))^*$ is the total probability of all paths that start and end at $\mathrm{s}(e)$, we can say that

$$\mathrm{w}(A) \times (\mathrm{w}(e)\,\mathrm{w}(R))^* = \mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{s}(e)).$$

An analogous argument can be made for $\mathrm{w}(\mathrm{t}(e), \mathrm{t}(\mathscr{G}))$, and therefore $\gamma'(e)$ again simplifies to

$$\gamma'(e) = \mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{s}(e)) \times \mathrm{w}(e) \times \mathrm{w}(\mathrm{t}(e), \mathrm{t}(\mathscr{G})).$$

Observe that $\mathrm{w}(\mathrm{s}(\mathscr{G}), \mathrm{s}(e))$ is precisely the Forward probability $\alpha[\mathrm{s}(e)]$, and $\mathrm{w}(\mathrm{t}(e), \mathrm{t}(\mathscr{G}))$ is the Backward probability $\beta[\mathrm{t}(e)]$. This means that the conditional expectation $\gamma(e)$ can be computed exactly as before in Figure 4.5.

The exact details of parameter estimation depend on the parameterization of the stochastic transducer. One often assumes [Clark, 2001; Eisner, 2002, for example] that parameters are associated with each edge of the transducer $\mathscr{T}$ in such a way that for each state q, the weights of all edges leaving q plus the final weight $\rho(q)$ of q sum to one. Eisner [2002] calls such transducers *Markovian*, and Mohri and Riley [2001] call them *standardized* and show that all deterministic minimal weighted transducers can be standardized. It makes sense to parameterize transducers in a way that ensures standardization, which will then be preserved as an invariant during parameter re-estimation. A very general case involves separate parameter vectors $\theta(q)$ for each state q which consist of parameters for each outgoing edge and for the final weight of q. For notational convenience, we assume that there is a single designated final state q_t, so that the final weight of any other state can be attached to an edge leading to q_t and labeled with $\langle \epsilon, \epsilon \rangle$. Outgoing edges will be identified by triples $\langle \sigma, \gamma, q' \rangle$ consisting of the edge labels and the target state of the edge, i. e., we assume w. l. o. g. that for each label there is at most one edge between any pair of states. This means that parameters have the form $\theta(q)(\sigma, \gamma, q')$, which we abbreviate as $\theta(q)(e)$. For Markovian or standardized transducers we assume that $\sum_e \theta(q)(e) = 1$ for all states q.

Parameter estimation amounts to mapping the expected edge counts to the appropriate parameters. For Markovian transducers, this is shown in Figure 5.4, which constitutes the so-called E step of the present instance of EM and closely resembles the corresponding algorithm for memoryless transducers from Figure 4.5. The maximization step is essentially the same as for memoryless transducers: for each state q, the expected parameter counts $\gamma(q)$ are normalized and become the new parameters $\theta(q)$ for q.

1: calculate the Forward probabilities α
2: calculate the Backward probabilities β
3: $p \leftarrow \mathrm{w}(\mathscr{G})$ // normalizing term, assumed to be nonzero
4: initialize γ to contain only zeroes
5: **for each** edge e in $\mathscr{G}$ **do**
6: $\quad \gamma(\mathrm{s}(e))(e) \leftarrow \gamma(\mathrm{s}(e))(e) + \dfrac{\alpha[\mathrm{s}(e)]\ \mathrm{w}(e)\ \beta[\mathrm{t}(e)]}{p}$
7: **end for**

Figure 5.4: Calculating expected parameter counts for general stochastic transducers.

5.3.2 On So-Called Expectation Semirings

An alternative way to compute expectations is to reformulate the problem in such a way that it can be solved by the generalized Gauss–Jordan–Kleene all-pairs algebraic path algorithm (Figure 5.1). This means defining an appropriate semiring structure such that expected values can be read off the distance matrix returned by the generalized Gauss–Kleene–Warshall algorithm.

Semiring structures for carrying out these kinds of computations are due to Riley [Michael D. Riley, personal communication, July 2002] and Eisner [2001, 2002]. The idea is to enrich the carrier set of the semiring in such a way that expected values can be accumulated. A useful example is Riley's entropy semiring:

$$\mathbb{K} = \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \qquad \langle p, \eta \rangle^* = \langle 1/(1-p),\ \eta/(1-p)^2 \rangle$$
$$\langle p, \eta \rangle \oplus \langle p', \eta' \rangle = \langle p + p',\ \eta + \eta' \rangle \qquad \bar{0} = \langle 0, 0 \rangle$$
$$\langle p, \eta \rangle \otimes \langle p', \eta' \rangle = \langle p'\, p,\ p'\, \eta + p\, \eta' \rangle \qquad \bar{1} = \langle 1, 0 \rangle$$

where $\langle p, \eta \rangle^*$ is undefined for $p \geq 1$. It is used as follows. We are given a weighted graph $\mathscr{G}$ corresponding to a stochastic transducer whose edge weights are probabilities taken from the closed nonnegative real semiring. Replace each edge weight p with a new weight $\langle p, -p \log(p) \rangle$ from the entropy semiring. Compute the generalized all-pairs algebraic distance matrix, and $\oplus$-add (in the entropy semiring) the generalized distances between the machine's start state and each final state. The result is a tuple $\langle P, \mathrm{H} \rangle$ where $P = \mathrm{w}(\mathscr{G})$ is the total probability mass of the stochastic transducer and H is the entropy of the probability distribution represented by the transducer. To see why this is the case, consider what happens when we $\otimes$-multiply two edges. Suppose their original probabilities are p and p', so that their weights in the expectation semiring are $\langle p, -p \log p \rangle$ and $\langle p', -p' \log p' \rangle$, respectively. Now $\otimes$-multiply these values:

$$\begin{aligned} \langle p,\ -p \log p \rangle \otimes \langle p',\ -p' \log p' \rangle &= \langle p\, p',\ p'(-p) \log p + p(-p') \log p' \rangle \\ &= \langle p\, p',\ -p\, p' \,(\log p + \log p') \rangle = \langle p\, p',\ -p\, p'\ \log(p\, p') \rangle \end{aligned}$$

The result is again of the form $\langle p, -p \log p \rangle$. So $\otimes$-multiplication can be used to work out the probability $p = \mathrm{w}(a)$ of each path a, together with its contribution $-p \log p$ to the entropy. By $\oplus$-adding up these values across all paths, we obtain

$$\left\langle \sum_{\substack{a \\ \mathrm{s}(a)=\mathrm{s}(\mathscr{G}) \\ \mathrm{t}(a)=\mathrm{t}(\mathscr{G})}} \mathrm{w}(a),\ -\sum_{\substack{a \\ \mathrm{s}(a)=\mathrm{s}(\mathscr{G}) \\ \mathrm{t}(a)=\mathrm{t}(\mathscr{G})}} \mathrm{w}(a) \log \mathrm{w}(a) \right\rangle = \langle \mathrm{w}(\mathscr{G}),\ \mathrm{H} \rangle.$$

Since the entropy H of a distribution is a special kind of expected value under that distribution, the approach can be generalized to compute expected values of arbitrary sorts. Eisner's [2002] expectation semirings are, not surprisingly, very

similar to Riley's entropy semiring. In fact, suppose Riley's expectation semiring was initialized as follows: replace each edge weight p of a stochastic transducer with a new weight $\langle p, 0\rangle$, and replace the weight p of a designated edge e with the new weight $\langle p, p\rangle$. Then computing the weight of the graph yields the total probability mass of the graph together with the expected number of occurrences of edge e in an accepting path.

Eisner's [2001] proposal is simply to let all ηs be vectors. Formally, Eisner's expectation semirings are defined in terms of certain semimodules over the closed nonnegative real semiring. In general, given a closed semiring $\langle R, \boxplus, \boxtimes, \hat{0}, \hat{1}\rangle$ and an (R, R)-bisemimodule $\langle M, \dot{+}, \cdot, \vec{0}, \vec{1}\rangle$, define the *trivial extension* [Golan, 1992, p. 145, example 13.20; see also example 13.33] of R by M, written $R \ltimes M$, as follows:

$$\mathbb{K} = R \times M \qquad \langle r, m\rangle^* = \langle r^*,\ r^* \cdot m \cdot r^*\rangle$$
$$\langle r, m\rangle \oplus \langle r', m'\rangle = \langle r \boxplus r',\ m \dot{+} m'\rangle \qquad \bar{0} = \langle \hat{0}, \vec{0}\rangle$$
$$\langle r, m\rangle \otimes \langle r', m'\rangle = \langle r \boxtimes r',\ r \cdot m' \dot{+} r' \cdot m\rangle \qquad \bar{1} = \langle \hat{1}, \vec{0}\rangle$$

However, the generality of this definition seems unnecessary, since the notion of expectation does not seem particularly meaningful outside a semiring that can be used for manipulating probabilities. Although in principle any semiring with $\mathbb{K} \subseteq \mathbb{R} \cup \{\infty\}$ and where $\oplus$ is real addition defines a discrete probability distribution on the paths of a graph, provided the weight of the graph is finite, it is an open question whether there are any practically useful semirings in which $\oplus = +$ and $\otimes$ does not essentially involve real multiplication. In practice, the real semiring and the isomorphic log semiring appear to be most useful, and so

expectations generally live in bisemimodules over the real semiring, which in all practical cases turn out to be vector spaces.

In all of Eisner's examples, the underlying semiring of his expectation semirings is in fact the real semiring, and the associated bisemimodules are real vector spaces $\mathbb{R}^n$ where n is the number of distinct kinds of expected values one wants to compute (the number of edges, or more generally the number of parameters, in case of parameter tying). Riley's entropy semiring is a special case where $n = 1$.

Using Eisner's expectation semiring requires the following initialization. Replace the weight p of the ith edge with $\langle p, p \cdot \mathbf{b}_i \rangle$ where $\mathbf{b}_i$ is the ith basis vector of the vector space (all components are zero, except that the ith component is one). The definition of expectation semirings as the trivial extension of the real semiring by a real vector space remains unchanged. With this initialization, computing the weight of the graph in the appropriate expectation semiring effectively computes the expected values of all edges simultaneously. However, the fact that the operations of the bisemimodule (the vector space) manipulate vectors of real numbers affects the space and time complexity of the overall computation.

The worst case space requirement for computing all-pairs algebraic paths in expectation semirings is $O(V^2 \times E)$ in the presence of cyclic graphs and no parameter tying, where V the number of vertices of the graph and E the number of edges, because each entry of the $V \times V$ distance matrix is an E-dimensional vector. If the graph is dense yet simple, $O(V^4)$ space is required. However, the state graph of the transducer need not be simple, as there can be up to $(|\Sigma| + 1)(|\Gamma| + 1)$ edges between any pair of nodes. For example, the memoryless transducers from Chapter 4 have that many loops. So the maximal number of edges if $V^2(|\Sigma| + 1)(|\Gamma| + 1)$.

As noted before, the running time of the all-pairs algebraic path algorithm is $O(V^3\,(T_\oplus + T_\otimes) + V\,T_{(\cdot)^*})$, where $T_\oplus$ etc. is the running time of an isolated $\oplus$ etc. operation. But note that the running times of the semiring operations are not constant in the expectation semiring, because they involve operations on E-dimensional vectors. So the running time of the all-pairs algebraic path algorithm is in fact $O(V^3\,E)$, which is $O(V^5)$ for dense simple graphs. Eisner [2001] is aware this and discusses a more sophisticated implementation that he intends to be used in practice.

By contrast, the approach outlined earlier requires a call to the all-pairs algebraic path algorithm applied to the real semiring, for which we assume, justifiably, that all semiring operations can be carried out in constant time. Therefore the overall running time of our algorithm for computing edge expectations is $O(V^3 + E)$ in the worst case, which is $O(V^3)$ for dense simple graphs. Eisner's algorithm, on the other hand, is extremely general and still applicable in the presence of complex parameter tying. However, some of the scenarios discussed by Eisner [2002] require symbolic representations of edge weights, which means that the basic semiring operations must evaluate and manipulate symbolic expressions, rather than concrete floating point numbers. This makes a naive implementation of Eisner's algorithm very expensive, but it might be the only straightforward choice if one were to use symbolic parameter tying.

In conclusion, we have shown that the parameter estimation algorithm proposed by Ristad and Yianilos [1998] for memoryless stochastic transducers extends more or less straightforwardly to general stochastic transducers, even in the presence of cycles. The running time of the extended parameter estimation algorithm is dominated by the running time of algebraic path algorithm discussed in Section 5.2. The alternative parameter estimation procedure proposed by Eisner

[2001, 2002] is more expensive to carry out, even when the naive implementation is avoided; whether the added flexibility it affords is essential for practical applications has yet to be demonstrated. We will see an example that could benefit from this flexibility in Section 6.8.

5.4 Obtaining Conditional Models

We had already seen in Section 5.2 that the marginal weight a weighted transducer assigns to a particular string can be computed by a variant of the procedure which evaluates the joint mass function for a particular pair of strings. To find the marginal weight of string y assigned by a transducer $\mathscr{T}$, compute the composition $\mathscr{T} \circ \mathrm{Str}(y)$ and determine the weight of its underlying graph, which is equal to $\bigoplus_{x'} \mathscr{T}(x', y)$.

5.4.1 Marginal Automata

As in Section 4.4 before, there may also be the need to construct the marginal automaton of a transducer. In general, the marginal automata are just projections of the original transducer. This means replacing each transducer edge of the form $\langle q, \sigma, \gamma, k, q' \rangle$ with an automaton edge $\langle q, \sigma, k, q' \rangle$ (first projection) or $\langle q, \gamma, k, q' \rangle$ (second projection). The resulting automata may not be very easy to work with, as they may contain ϵ-transitions and are generally not deterministic. As pointed out in Section 4.4, deterministic automata are important for constructing conditional transducers. In general, marginalization comprises the following steps:

1. Project the transducer onto an automaton;

2. Remove ϵ-transitions from the automaton;

3. Determinize the automaton.

The generic marginalization algorithm for memoryless transducers, which appeared in Figure 4.9, implicitly carries out all three steps. The computation of marginal weights on lines 7 and 9 is necessary for projecting a deterministic automaton (the second projection is taken), and $\otimes$-multiplying the weights of non-ϵ-edges with the weight of the source state's deletion loops on line 11 corresponds to ϵ-removal, as it ensures that the weights of ϵ-transitions are correctly preserved.

5.4.2 Conditional Stochastic Transducers

Conditionalization of a stochastic transducers also works as in Section 4.4 before. It only makes sense in the real semiring, as it requires an unambiguous marginal automaton whose weights are replaced by their reciprocal values, and which is then composed with the original transducer [see also Eisner, 2002]. Formally, the following steps are involved:

1. Given a stochastic transducer $\mathscr{T}$, compute the first projection $\pi_1(\mathscr{T})$ (resp. second projection $\pi_2(\mathscr{T})$), remove ϵ-transitions, determinize, and call the resulting automaton $\mathscr{A}$;

2. Replace each edge $\langle q, \sigma, k, q' \rangle$ of $\mathscr{A}$ with an edge $\langle q, \sigma, \sigma, 1/k, q' \rangle$ and call the resulting transducer $\mathscr{R}$;

3. Compute the composition $\mathscr{R} \circ \mathscr{T}$ (resp. $\mathscr{T} \circ \mathscr{R}$).

The first step is the marginalization procedure described above. The second step constructs a transducer $\mathscr{R}$ that is just like the automaton $\mathscr{A}$ except that its weights

are the multiplicative inverses of the corresponding weights of $\mathscr{A}$. While this would in principle work in any division semiring, there do not appear to be any applications outside the real semiring (or the isomorphic log semiring). Suppose we are working with the second projection, as in all previous examples. Marginalization ensures that $\mathscr{A}(y) = \sum_{x'} \mathscr{T}(x', y)$, which together with the fact that $\mathscr{A}$ is deterministic means that the behavior of $\mathscr{R}$ is the following:

$$\mathscr{R}(y', y) = \begin{cases} \dfrac{1}{\sum_{x'} \mathscr{T}(x', y)} & \text{if } y' = y \\ 0 & \text{otherwise} \end{cases}$$

The composition carried out in the third step has the following behavior:

$$\begin{aligned} [\mathscr{T} \circ \mathscr{R}](x, y) &= \sum_{y'} \mathscr{T}(x, y') \circ \mathscr{R}(y', y) \\ &= \frac{\mathscr{T}(x, y)}{\sum_{x'} \mathscr{T}(x', y)} \end{aligned}$$

The last equality is due to the fact that the sum over all y' contains precisely one nonzero term, namely when $y' = y$.

The conditionalization algorithm for memoryless stochastic transducers from Figure 4.11 performs the three general steps described here. The first step (marginalization) was described above, and corresponds to lines 8, 10 and 12 of Figure 4.11. In fact, line 12 takes care of the ϵ-removal step of marginalization, and also computes the reciprocal of the marginal weight. Composition of memoryless transducers, which is almost trivial, corresponds to line 15.

In short, the marginalization and conditionalization algorithms encountered in Section 4.4 are special cases of the general procedures described here, just like

the Forward and Backward algorithms in Section 4.2 turned out to be special cases of the general evaluation procedure in Section 5.2.

The general procedures have the following ingredients: projection, described above; ϵ-removal [Mohri, 2002b]; determinization [Mohri, 1997, sec. 3.3]; and composition, described in Section 5.2. Some further comments on ϵ-removal and determinization are in order. Determinization is not strictly required, as we only need to ensure that an automaton is unambiguous. Disambiguation is possible under more general conditions [Eilenberg, 1974] than determinization [Mohri, 1997, sec. 3.4–5], which is not always possible for weighted automata. However, general weighted disambiguation algorithms do not seem to have received much attention in the literature (Roche and Schabes [1997b, fig. 1.28] give a disambiguation algorithm for the unweighted case), and the automata we have encountered in practice all turned out to be determinizable.

5.4.3 Epsilon-Removal in the Real Semiring

Removing ϵ-transitions is often considered an integral part of determinization, but is conceptually quite distinct. Furthermore, ϵ-removal is always possible, as it only involves algebraic distance computations. The rest of this section concerns the ϵ-removal algorithm for weighted automata [Mohri, 2001, 2002b]. We discuss this algorithm in some detail because of certain flaws in Mohri's presentation. Our primary goal is to obtain a correct, though not necessarily efficient, algorithm for the case of stochastic automata.

Consider the automaton shown in Figure 5.5. It is Markovian, or standardized, in the sense discussed earlier, as the weights of all outgoing transitions (including the final weight) sum to one for each state. However, some of these outgoing

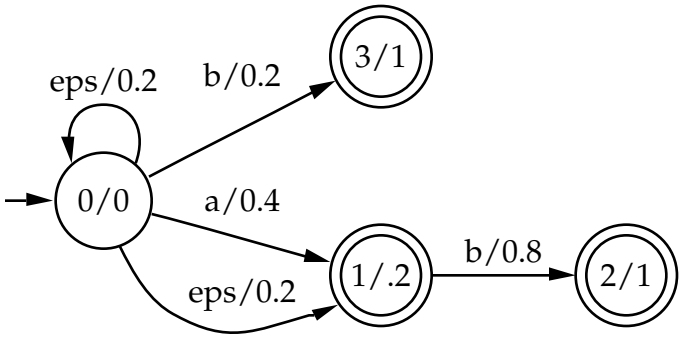

Figure 5.5: A stochastic automaton with ϵ-transitions.

transitions are labeled with the empty string ϵ (called 'eps' in the figure). The goal of ϵ-removal is to produce an equivalent automaton without any ϵ-transitions.

The weighted ϵ-removal algorithm [Mohri, 2001] has two major steps. In a first step, the so-called ϵ-closure is computed. The ϵ-closure of a state p is the set $C[p]$ of states q reachable from p via paths (possibly empty) labeled with ϵ, together with the algebraic distance $d[p,q]$ from p to q, which is the $\oplus$-sum of the weights of all ϵ-paths from p to q. Note that q is unreachable iff $d[p,q] = \bar{0}$. Formally, the ϵ-closure of p is defined like this:

$$C[p] = \{\langle q, w\rangle \in Q \times \mathbb{K} \mid d[p,q] = w \wedge w \neq \bar{0}\}$$

It can be viewed as a sparse row vector of the algebraic distance matrix d.

One can work with the so-called ϵ-subgraph of an automaton, which contains exactly those edges labeled with ϵ, and compute the algebraic distance from p to

q in that subgraph without regard to the path labels. Mohri's approach to computing the ϵ-closure uses his own generic single-source algebraic path algorithm [Mohri, 1998, 2002c]. However, as noted in Section 5.2, Mohri's algebraic path algorithm requires k-closed semirings and therefore does not apply to the case of weighted automata over the real semiring. When working in the real semiring, one can again use the generalized Floyd–Warshall algorithm from Figure 5.1 for computing the algebraic distance matrix in the ϵ-subgraph of the automaton.

The second major step of Mohri's weighted ϵ-removal algorithm adjusts the transitions of a weighted automaton. The adjustment has the effect that a state p has an outgoing transition labeled with symbol $\sigma \in \Sigma$ just in case there is a state q in the original automaton that is reachable from p via zero or more ϵ-transitions, and q has an outgoing transition labeled with σ which has weight w'; the adjusted transition then has weight $w \otimes w'$, where $w = d[p, q]$ is the algebraic distance from p to q in the ϵ-graph, as computed during the first step.

Note that in almost all semirings $p \in C[p]$. For this not to be the case, $d[p, p]$ would have to equal $\bar{0}$. But $d[p, p]$ is always of the form a^*, and so $p \notin C[p]$ would mean that there is some weight a such that $a^* = \bar{0} = d[p, p]$. But that would allow us to conclude:

$$
\begin{aligned}
a^* &= \bar{0} \\
\bar{1} \oplus a \otimes a^* &= \bar{1} \oplus a \otimes \bar{0} \\
a^* &= \bar{1} \\
\bar{0} &= \bar{1} \\
b \otimes \bar{0} &= b \otimes \bar{1} \quad \text{for any } b \\
\bar{0} &= b
\end{aligned}
$$

1: **for each** $p \in Q$ **do**
2: $\quad E'[p] \leftarrow \{\}$
3: $\quad \rho'[p] \leftarrow \bar{0}$
4: $\quad$ **for each** $(q, w) \in C[p]$ **do**
5: $\quad\quad E'[p] \leftarrow E'[p] \cup \{(p, \sigma, w \otimes w', r) \mid (q, \sigma, w', r) \in E[q], \sigma \neq \epsilon\}$
6: $\quad\quad \rho'[p] \leftarrow \rho'[p] \oplus (w \otimes \rho[q])$
7: $\quad$ **end for**
8: **end for**
9: **return** E', ρ'

Figure 5.6: The correct ϵ-removal algorithm.

In other words, $a^* = \bar{0}$ can only happen for the degenerate one-element semiring, an uninteresting case one can easily ignore or explicitly forbid [see for example Golan, 1992, p. 1]. Thus $p \in C[p]$ for all non-trivial semirings.

An edge of a weighted automaton is a quadruple from $Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$, where Q is the set of states, Σ the alphabet, and $\mathbb{K}$ the (carrier set of the) weight semiring. Let $E[p]$ denote the outgoing edges of state p. Let $\rho[p] \in \mathbb{K}$ be the final output function at state p. Then the set F of final states can be defined in terms of Q, $\mathbb{K}$ and ρ as $F = \{q \in Q \mid \rho[q] \neq \bar{0}\}$.

The correct ϵ-removal algorithm appears in Figure 5.6. A proof of its correctness was given by Mohri [2001], though it does not apply to Mohri's own algorithm, which is incorrect because it explicitly assumes that $d[p, p] = \bar{1}$, which need not be the case in general (consider the automaton in Figure 5.5 as an example).

Mohri's algorithm is shown in Figure 5.7. A non-essential difference is that it keeps track of the set F of final states. It contains a glitch in the inner loop body: the update of the outgoing edges E and final weights ρ are wrong when state q has

```
1:  for each p ∈ Q do
2:     E[p] ← {(p, a, w′, r) | (p, a, w′, r) ∈ E[p], a ≠ ϵ}
3:     for each (q, w) ∈ C[p] do
4:        E[p] ← E[p] ∪ {(p, a, w ⊗ w′, r) | (q, a, w′, r) ∈ E[q], a ≠ ϵ}
5:        if q ∈ F then
6:           if p ∉ F then
7:              F ← F ∪ {p}
8:           end if
9:           ρ[p] ← ρ[p] ⊕ (w ⊗ ρ[q])
10:       end if
11:    end for
12: end for
```

Figure 5.7: Mohri's ϵ-removal algorithm.

been processed before state p, since then $E[q]$ already contains updated edges not present in the original automaton. It is obvious that Mohri's algorithm is a special case of the algorithm in Figure 5.6 when $d[p, p] = \bar{1}$ for all states p: since $p \in C[p]$ as discussed earlier, the update performed on line 2 of Mohri's algorithm is just a special case (namely $p = q$) of the update on line 5 of the correct algorithm.

However, the correct algorithm in Figure 5.6 also applies when $d[p, p] \neq \bar{1}$. In particular, it applies to the automaton in Figure 5.5. In that example the non-$\bar{0}$-zero ϵ-distances are $d[0, 0] = 0.2^* = 1.25$ and $d[0, 1] = 0.2^* \times 0.2 = 0.25$. After the application of the ϵ-removal algorithm the weights of the transitions leaving state 0 in the original automaton have been multiplied by $d[0, 0] = 1.25$, a transition labeled with b going from state 0 to state 2 appears whose weight is $d[0, 1] \times 0.8 = 0.2$, and state 0 is now final with final weight $d[0, 1] \times \rho[1] = 0.25 \times 0.2 = 0.05$. Figure 5.8 shows the resulting automaton.

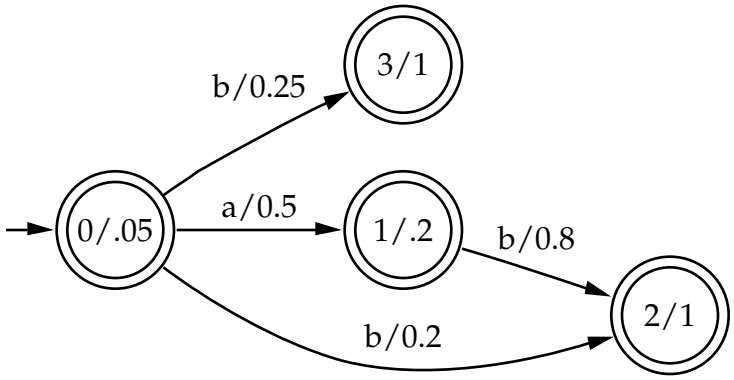

Figure 5.8: An ϵ-free automaton equivalent to the one in Figure 5.5.

In sum, we have seen that the marginalization and conditionalization procedures first introduced in Section 4.4 apply essentially unchanged to general stochastic transducers. Marginal automata arise as projections of transducers. In order to obtain a conditional transducer, the relevant marginal transducer must be made unambiguous. Formulating a general disambiguation algorithm for those weighted transducers that have an unambiguous representation appears to be an open problem. In practice, determinization can be used, since deterministic transducers are necessarily unambiguous. Determinization requires weighted ϵ-removal, which, until recently [Mohri, 2001], was never discussed as an independent operation on weighted machines worthy of detailed attention. However, even the most recent version of Mohri's [2002b] weighted ϵ-removal algorithm contains unnecessary assumptions and flaws, which were corrected in this section.

5.5 Using a Joint Model for Prediction

5.5.1 MAP Decoding

The fundamental issues for MAP decoding discussed in Section 4.5 are the same for general stochastic transducers as they were for memoryless transducers. We had previously discussed the situation where we want to find the most likely string x corresponding to a given string y. In the framework of the current chapter, this can be done as follows:

1. Let $\mathscr{T}$ be a joint stochastic transducer. Construct a transducer $\mathscr{A}$ (over the real semiring) representing the composition $\mathscr{T} \circ \mathrm{Str}(y)$ (the first projection of $\mathscr{A}$ could be taken without affecting the outcome);

2. Ensure that $\mathscr{A}$ (over the real semiring) is unambiguous;

3. Without changing the weights of $\mathscr{A}$, replace its weight semiring with the max–times semiring (or any semiring isomorphic to it, e. g., the tropical semiring, which would require changing the weights using the appropriate isomorphism from the max–times semiring);

4. Find the optimal path of $\mathscr{A}$ (over the max–times semiring) and read off its label.

As discussed in Section 5.2 and Section 5.4, the first step constructs the transducer or automaton representing the marginal distribution of y. In fact, it would be possible to let $\mathscr{A} = \mathscr{T} \circ \mathscr{Y}$ for more general transducers $\mathscr{Y}$, which, for example, might represent a disjunction of multiple strings. The remaining steps are as in Section 4.5. Step 2 ensures that every string accepted by $\mathscr{A}$ corresponds to exactly one successful path of $\mathscr{A}$, so the best string can be obtained by finding the best

path. For the reasons discussed in Section 5.4, step 2 usually means determinizing $\mathscr{A}$, which is not always possible, and even if it is the result may sometimes be very large. If step 2 is left out, the best string may be represented by multiple paths, potentially excluding the best path (we had seen examples of this in Section 4.5). Skipping step 2 results in the so-called Viterbi approximation. Step 3 simply spells out the fact that the search for the best path takes place in a different semiring which, among other things, is idempotent and naturally ordered. Step 4 uses any best path algorithm to search for the optimal path. If the tropical semiring was used in step 3 (which requires taking the negative logarithm of all weights) and the new weights are all nonnegative (this is necessarily the case if $\mathscr{T}$ is a Markovian transducer), then Dijkstra's single-source shortest path algorithm can be used in step 4, just as in Section 4.5 before.

The use of determinization in step 2 is in fact less problematic than it was in the preceding section. Mohri [1997, p. 293] points out that machines representing finite sets can always be determinized. So if it is known that the best string corresponding to a given string y is contained in a finite set X, one can construct an automaton $\mathscr{X}$ accepting X and let $\mathscr{A} = \mathscr{X} \circ \mathscr{T} \circ \mathrm{Str}(y)$ in step 1 above. In step 2, determinization can be performed because $\mathscr{A}$ is guaranteed to be determinizable.

If one is looking for the single best path, it may not matter whether an automaton is determinizable, since it does not have to be expanded in full. When using Dijkstra's algorithm, the automaton (over the tropical semiring) only needs to be expanded until an accepting transition out of a final state is taken. All other partially expanded paths are necessarily longer (less likely), hence the algorithm can stop early, as soon as an accepting transition is encountered.

5.5.2 Minimum Risk Decoding

Even if we can find the mode of a probability distribution efficiently in practice, this may not be what we want. We had argued in Chapter 2 that symbol error should be the preferred evaluation metric, but decoding the most likely string minimizes string error, and does not generally minimize symbol error. The problem of minimizing symbol error has been partly addressed in the recent speech recognition literature.

The quality of the speech transcripts produced by automatic speech recognizers is usually measured in terms of word error rate (WER), which we refer to abstractly (Figure 2.3) as symbol error rate. However, both parameter estimation and decoding for continuous speech recognition typically employ maximum likelihood methods [Bahl et al., 1983]. The same is true of the approaches described here: the parameter estimation procedures described in Section 4.3 and Section 5.3 find (local) maxima of the likelihood, and MAP decoding finds (perhaps to an approximation) the hypothesis with the highest posterior probability. Since all models describe properties of whole strings, these training and decoding methods minimize string error instead of symbol error.

Theoretically, it would be preferable to minimize risk , i. e. expected loss, during parameter estimation and decoding, using an appropriate loss function such as symbol error. While there has been some work on minimum risk parameter estimation for speech recognition under specific loss functions [Bahl et al., 1988; Rahim and Lee, 1997; Saul and Rahim, 2000], the procedures are often expensive and not used much in practice.

Here we focus on minimum risk decoding, which has received more attention lately [see Evermann, 1999, for an overview]. Suppose we are given a distribution

P over a discrete set of hypotheses X. Minimum risk decoding is the task of finding a hypothesis $x^\star = \operatorname{argmin}_x R(x)$ whose risk $R(x^\star)$ under P is minimal among the elements of X. The risk $R(x)$ of a hypothesis is its loss expected under P, where $L(x, x')$ is the loss incurred by selecting hypothesis x when the correct choice would have been x'. Formally:

$$R(x) = \sum_{x' \in X} L(x, x')\, P(x') \tag{5.2}$$

A concrete example may illustrate the relevant concepts. Consider a basic alphabet $\Sigma = \{a, b, c\}$ and let $L(x, x')$ be the Levenshtein distance (see page 121) between x and x'. Furthermore, suppose P is the following distribution over Σ^*:

$$\begin{aligned} P(aa) &= 0.3 \\ P(ab) &= 0.3 \\ P(bc) &= 0.4 \\ P(x) &= 0 \quad \text{for all other strings } x \in \Sigma^* \end{aligned}$$

In this setting the MAP hypothesis is bc. But notice that the closest contenders both favor a as the first symbol; moreover, the disjunction of aa and ab has higher probability than bc. Consider the expected loss $R(x)$ for several hypotheses $x \in$

Σ^*. Let $p = (0.3, 0.3, 0.4)$, and so the vector dot product $p \cdot (0, 1, 2)$, for example, is the risk of the hypothesis aa, since $L(aa, aa) = 0$, $L(aa, ab) = 1$, and $L(aa, bc) = 2$.

$$
\begin{aligned}
R(\epsilon) &= p \cdot (2, 2, 2) = 2.0 \\
R(c) &= p \cdot (2, 2, 1) = 1.6 \\
R(a) &= p \cdot (1, 1, 2) = 1.4 \\
R(b) &= p \cdot (2, 1, 1) = 1.3 \\
R(bc) &= p \cdot (2, 2, 0) = 1.2 \\
R(aa) &= p \cdot (0, 1, 2) = 1.1 \\
R(ab) &= p \cdot (1, 0, 2) = 1.1 \\
R(ac) &= p \cdot (1, 1, 1) = 1.0
\end{aligned}
$$

It is easy to see that the minimum risk hypothesis is ac (all other hypotheses, including infinitely many not shown here due to lack of space, have higher expected loss). Notice that ac has probability 0 under P.

Unfortunately, minimum risk decoding is **NP**-complete, since it is an instance of the so-called MEDIAN-STRING problem [de la Higuera and Casacuberta, 2000]. This is not surprising, because it is known that the intuitively simpler problem MOST-LIKELY-STRING is already **NP**-complete, as mentioned in Section 4.5. Unlike in speech recognition, this is not a major concern for letter-to-sound conversion, as the problem instances are usually quite small in practice. Approximate methods for minimum risk decoding exist that use n-best lists [Stolcke et al., 1997] (which could be made more efficient by using a better algorithm for finding the n best MAP hypotheses [Mohri and Riley, 2002]), or so-called "confusion networks" [Mangu et al., 2000]; Evermann [1999] gives a detailed overview.

An exact algorithm was recently given by Mohri [2002a]. However, Mohri's algorithm may be expensive in practice (it is prohibitively expensive in theory, with a worst case exponential running time), since it employs three separate invocations of weighted determinization. What makes Mohri's algorithm nontrivial is the fact that Levenshtein distance is most naturally defined in terms of the operations of the tropical semiring, whereas the probability distribution P is most naturally defined in the nonnegative real semiring. Although the definition of risk R in equation (5.2) appears to be of the same form as the definition of composition in equation (5.1), an automaton realizing R cannot be constructed naively by composition of a transducer realizing L with an automaton realizing P, when the weights of L are taken in the tropical semiring and those of P in the real semiring, as composition is only defined if the two semirings are identical. Mohri [2002a] shows, among other things, how Levenshtein distance can be computed in the real semiring (or, more precisely, the log semiring).

Whereas word error rate based on Levenshtein distance is the commonly used evaluation metric in speech recognition and should therefore be the loss function minimized during decoding (and, ideally, also during training) by a speech recognizer, the lack of a widely agreed upon evaluation metric for letter-to-sound rules is perhaps a blessing in disguise, since one can look for an evaluation metric that is easy to work with as a loss function for minimum risk decoding. A promising candidate is the notion of *stochastic edit distance*, introduced by Ristad and Yianilos [1998] which is the (negative logarithm of the) sum of the costs of all memoryless alignments of two string, where the cost of an alignment is the product of the costs of its edit operations. If the costs of edit operations are probabilities, as they are for Ristad and Yianilos [1998], the stochastic edit distance is precisely the (negative logarithm of the) probability of two strings being generated by a

memoryless stochastic transducer, as described in Section 4.2. In other words, the stochastic edit distance is computed over the nonnegative real semiring, while ordinary edit distance is computed over the (nonnegative) tropical semiring. Both can be evaluated using the generic Forward algorithm from Figure 4.1.

We propose to use a variant of Ristad and Yianilos's [1998] stochastic edit distance as the loss function for minimum risk decoding. To simplify matters, we use a probability distribution more or less directly, without taking negative logarithms. More precisely, assume that the loss function L is of the form

$$L(x, x') = 1 - P'(x \mid x'),$$

where P' is a probability distribution conditional on x'. Note that L is nonnegative and can be interpreted as the probability of not selecting hypothesis x given that the true hypothesis is x'. Furthermore, as P' is a probability distribution, it can potentially be constructed by training on samples of similar or confusable strings using the methods described in Section 4.3 and Section 4.4. Under these assumptions, risk minimization can be simplified like this:

$$\begin{aligned}
\operatorname*{argmin}_x R(x) &= \operatorname*{argmin}_x \sum_{x'} L(x, x')\, P(x') \\
&= \operatorname*{argmin}_x \sum_{x'} (1 - P'(x \mid x'))\, P(x') \\
&= \operatorname*{argmin}_x \left(\sum_{x'} P(x') - \sum_{x'} P'(x \mid x')\, P(x') \right) \\
&= \operatorname*{argmin}_x \left(1 - [\pi_1(P' \circ P)](x)\right) \\
&= \operatorname*{argmax}_x [\pi_1(P' \circ P)](x)
\end{aligned}$$

Crucially, P' and P are both probability distributions expressed in the real semiring (or, equivalently, the log semiring). Unlike in the situation where L is Levenshtein distance, the simplifying composition $P' \circ P$ is well defined. In fact, if we let $\mathscr{A} = \pi_1(P' \circ P)$ we can continue with steps 2 through 4 of the decoding procedure outlined at the very beginning of this section. Step 2 involves disambiguation or determinization, but this is the only expensive computation involved, whereas Mohri's decoding algorithm under Levenshtein loss uses up to three determinization steps.

Continuing with our earlier example, let the conditional distribution P' be represented by a memoryless stochastic transducer with the following parameters:

$$\begin{aligned}
\theta(\#) &= 0.94 & \theta(b, \epsilon) &= 0.02 \\
\theta(\epsilon, a) &= 0.06 & \theta(b, a) &= 0.27 \\
\theta(\epsilon, b) &= 0.06 & \theta(b, b) &= 0.33 \\
\theta(\epsilon, c) &= 0.06 & \theta(b, c) &= 0.27 \\
\theta(a, \epsilon) &= 0.02 & \theta(c, \epsilon) &= 0.02 \\
\theta(a, a) &= 0.33 & \theta(c, a) &= 0.27 \\
\theta(a, b) &= 0.27 & \theta(c, b) &= 0.33 \\
\theta(a, c) &= 0.27 & \theta(c, c) &= 0.27
\end{aligned}$$

These are approximately the parameters of the conditional transducer (see Section 4.4) corresponding to a joint transducer which generates exact matches with probability 0.11, substitutions with probability 0.09, insertions and deletions with probability 0.02, and which halts with probability 0.01.

A stochastic automaton realizing the posterior distribution P is shown in Figure 5.9. The distribution P is similar to the one occurring in the earlier example, with most of its mass (more than 92%) assigned to the strings aa, ab, and bc (the relevant transitions appear in bold in Figure 5.9), but P also reserves a small portion of the total probability mass for the infinitely many other strings. In particular, P assigns the following probabilities to strings:

$$
\begin{aligned}
P(aa) &= 0.280427 \\
P(ab) &= 0.280427 \\
P(bc) &= 0.366951 \\
P(ac) &= 0.001475 \\
P(x) &\leq 0.01 \qquad \text{for all other strings } x \in \Sigma^*
\end{aligned}
$$

As in the previous example involving Levenshtein loss, the MAP hypothesis under P is the string bc, but the minimum risk hypothesis is the string ac, which is not very probable according to P alone.

In conclusion, we have described exact and approximate maximum a posteriori decoding and minimum risk decoding for stochastic transducers. While the concept of minimum risk decoding is not new per se, it has, to our knowledge, never been applied to letter-to-sound conversion. A novel contribution of this section is the suggestion to employ a loss function that can be naturally expressed in the real semiring, for example, loss defined in terms of conditional probabilities. When both the loss function and the posterior distribution are represented as weighted automata over the real semiring, the risk function can be expressed as their composition. Minimum risk decoding in such a setting is conceptually

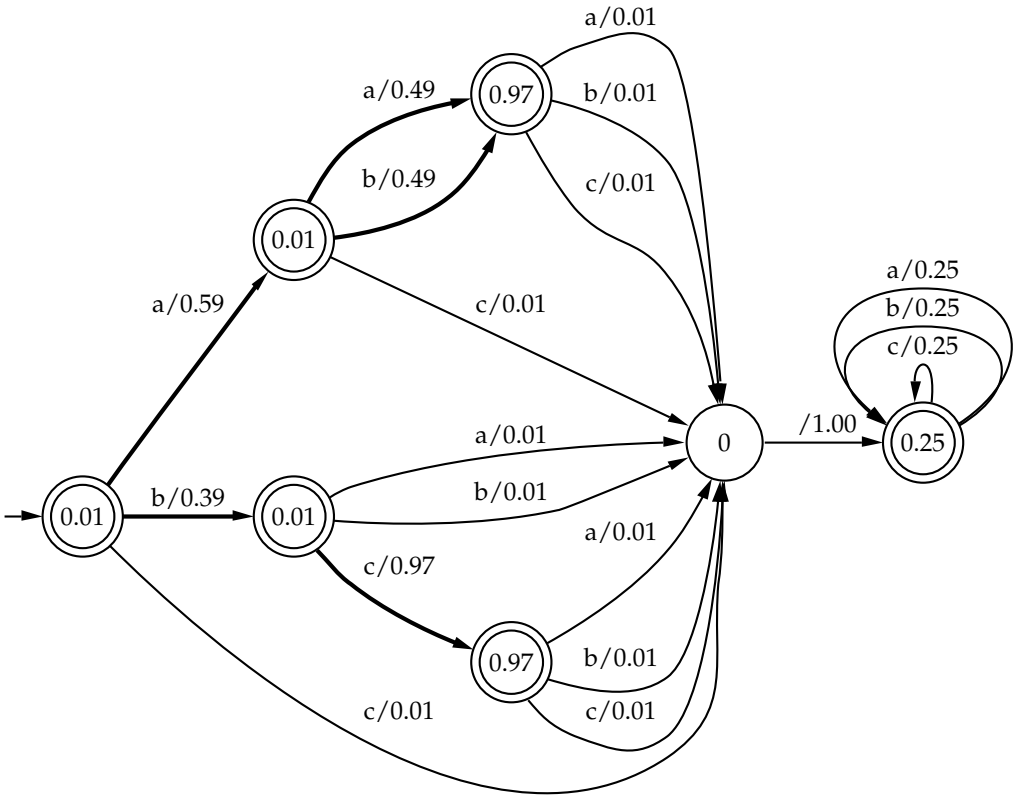

Figure 5.9: A stochastic automaton representing the posterior distribution P used in minimum risk decoding.

much simpler than when traditional Levenshtein distance is used as the loss function. Furthermore, using probabilities (or the so-called stochastic edit distance) instead of Levenshtein distance may potentially give the loss incurred by a pair of strings a natural interpretation, for example, as the probability that the second string is mistaken (misread, misperceived, mispronounced, etc.) as the first string. A similarly natural interpretation of Levenshtein distance does not seem possible.

5.6 Conclusion

This chapter generalized Ristad and Yianilos's [1998] approach described in Chapter 4 to the general case of stochastic rational transducers with arbitrary state graphs. Ristad and Yianilos [1998] had merely hinted at the possibility of such a generalization without providing any details.

The discussion touched on the same fundamental issues that had been introduced and worked out for the memoryless case in Chapter 4. Algorithms for solving the four fundamental problems for stochastic transducers (evaluation, decoding, estimation, conditionalization) were presented within the framework of weighted finite state transducers.

The problem of computing the total weight or probability mass of a transducer was discussed in Section 5.2. Several algorithms may be applicable, depending on the topology of the transducer and on properties of its weight semiring. In the worst case, a cubic-time all-pairs algebraic path algorithm must be used. While this is not an option for large vocabulary continuous speech recognizers [Mohri, 2002c], the transducers used in letter-to-sound applications are usually very compact and therefore not problematic. The worst case can arise quite easily, since

general stochastic transducers may have cycles, and because real numbers representing probabilities have properties that prevent standard textbook algorithms from being applicable. We pointed out that an appropriate version of an all-pairs algebraic path algorithm is applicable and can be used as part of evaluation and parameter estimation.

Parameter estimation may appear to be more complex in the presence of cycles, but Section 5.3 showed that this is not actually the case. The EM algorithm given by Ristad and Yianilos [1998] for memoryless transducers is still applicable, but the computation of expected counts is more involved. Compared with Eisner's [2001] parameter estimation algorithm, the algorithm presented in Section 5.3.1 is cheaper, but slightly less general. Whether this loss of generality plays any role in practice remains to be seen.

Conditionalization and decoding require unambiguous automata, for which the lack of a general weighted disambiguation algorithm was noted. Weighted determinization [Mohri, 1997] is used instead, after ϵ-removal has been carried out. Some glitches in Mohri's [2002b] ϵ-removal algorithm were corrected in Section 5.4.3.

We proposed the use of minimum risk decoding for stochastic transducers and in particular for letter-to-sound applications. Classical Levenshtein distance is not easy to work with when used as a loss function for decoding. A closely related alternative loss function based on stochastic memoryless transducers representing conditional distributions was suggested, which is easier to work with, results in simpler decoding algorithms, and has a more natural interpretation than Levenshtein distance. However, as noted in Section 2.3.3, more empirical work is needed in order to derive a loss function defined in terms of plausible confusion probabilities.

CHAPTER 6

EXPERIMENTS

6.1 Introduction

In the preceding three chapters we had mainly focused on theoretical and algorithmic issues surrounding the classifier-based and the transduction-based approach to letter-to-sound conversion. In this chapter we return to empirical and practical issues. Section 6.2 investigates the relationship between prediction error and symbol error, which are often closely connected in practice. If there is a close enough connection between symbol error and prediction error, then classification-based approaches can be seen as heuristics for minimizing symbol error indirectly, by minimizing prediction error. In Section 6.3 we discuss an exhaustive search strategy that can be used to minimize string error directly for certain small problems. In Section 6.4 exhaustive search is replaced by a greedy heuristic, which can be applied for larger problems. These two sections also address the question when minimizing string error is useful in practice. Section 6.5 establishes best-case performance results for classifier-based approaches. Such best-case bounds are important for deciding whether existing approaches can still be improved, or whether they are already close to optimal on a given data set. Section 6.6 assesses the effect of the Viterbi approximation, which had been discussed at two points in

Chapter 5, on the performance of the transduction-based approach. The best overall training and decoding strategy is from that section is then compared with the classifier-based approach in Section 6.7. Finally, we mention the issue of length (or duration) modeling as an open problem for modeling with stochastic finite state transducers in Section 6.8.

6.2 Is Prediction Error Overly Fussy?

As discussed on page 42, prediction error is sensitive to irrelevant differences between the predicted transcription and the reference transcription of a word. Specifically, when padding symbols are present in the transcription strings, it is possible that the predicted transcription and reference transcription differ only in the placement of padding symbols, resulting in spurious prediction errors, while the corresponding phoneme strings derived from the transcription strings are actually identical.

The following experiment shows that spurious prediction errors do occur regularly in practice. Since prediction error is only meaningful for aligned data, the development and evaluation data were taken from the aligned NETtalk data set (see Section 2.2.2). The presence of spurious prediction errors is indicative of inconsistent alignment choices in the evaluation data.

After removal of stress information, the NETtalk dictionary consists of 19,940 unique entries (19,802 unique words, due to the presence of homographs). In the following experiments, the data are partitioned into two equal parts, each containing 9,970 entries, with one part being used for training and the other for evaluation.

The clearest case of spurious prediction errors is a situation where the prediction error for a given word is nonzero, but the string error is zero (and therefore the symbol error is too). Repeated partitioning (50 times) of the NETtalk data into equal training and evaluation data was carried out. For each partition of the data, a classifier with a seven-letter context window was trained on the development portion and tested on the evaluation portion of the partitioned data, and the number of perfectly predicted words with nonzero prediction error was recorded. On average, 4.3 words ($N = 50$, $\sigma = 2.33$) incurred spurious prediction errors; the per-trial numbers ranged from 1 to 9, i. e., were never zero. Spurious prediction errors in the test data point to inconsistent alignments, which could be corrected by changing the test data, substituting the predicted pronunciations for the reference pronunciation, since they are both correct and differ only in alignment details.

A less clear case can be made by comparing raw prediction error with raw symbol error (Levenshtein distance). One cannot compare prediction error *rate* with symbol error *rate*, since prediction error rate is raw prediction error divided by the number of *letters*, whereas symbol error rate is raw symbol error divided by the number of *reference phonemes* (see Section 2.3). When a transcription string is predicted with a single prediction error, this always corresponds to one or two symbol errors: a single mispredicted transcription symbol corresponds to zero, one, or two phonemes, and therefore to one or two insertions, deletions, or substitutions when comparing the predicted phoneme string to the reference phoneme string, so a raw prediction error of 1 potentially underestimates raw symbol error. If the absolute number of prediction errors is greater than 1, however, prediction error can overestimate symbol error, since two mispredictions may cancel out, as discussed in Section 2.4.1.

In a similar setting as before, the number of words was recorded for which the raw prediction error exceeds the raw symbol error. This was repeated for several 1:1 splits of the NETtalk data. On average, prediction error overestimated symbol error for 77.1 words ($N = 50$, $\sigma = 8.55$). However, the number of words for which symbol error exceeds prediction error was always larger. This suggests that a good heuristic for optimizing symbol error would be to reduce the number of errors made when predicting multi-phone symbols.

In sum, prediction error does diverge from symbol error in practice, but the divergence is not very large and may be tolerable. If a divergence is present it may indicate a problem with the training data, depending on the direction of the divergence. If prediction error exceeds symbol error for one word, there probably are spurious alignment differences that could be removed if changing the test data is an option. The case of symbol error exceeding prediction error can potentially be dealt with by using cost-sensitive training procedures for the underlying classifier, since the misclassification cost for multi-phoneme symbols is clearly higher than for ordinary transcription symbols. An example of a cost matrix for a subset of the NETtalk transcription symbols (see Figure 2.2) appears in Figure 6.1. Each cell holds the Levenshtein distance between the IPA correspondences of the transcription symbols for its row and its column.

The matrix of Levenshtein distances between individual transcription symbols was used as part of a comparison between cost-sensitive training and ordinary training of a classifier-based letter-to-sound converter. As before, the training and evaluation data were obtain by a random 1:1 split of the NETtalk data set, resulting in two disjoint dictionaries containing 9,970 entries each. Throughout this comparison a symmetric three-letter window was used, in other words, only the immediately preceding and the immediately following letter were considered

	k /k/	s /s/	S /ʃ/	K /kʃ/	X /ks/	! /ts/	- / /
k /k/	0	1	1	1	1	2	1
s /s/	1	0	1	2	1	1	1
S /ʃ/	1	1	0	1	2	2	1
K /kʃ/	1	2	1	0	1	2	2
X /ks/	1	1	2	1	0	1	2
! /ts/	2	1	2	2	1	0	2
- / /	1	1	1	2	2	2	0

Figure 6.1: Excerpt from the cost matrix used for cost-sensitive training and evaluation of a classifier on the NETtalk data set.

when predicting the pronunciation of a given letter in context. A backed-off majority classifier provided the baseline for this comparison: during training this classifier stores the labels encountered for all context windows of varying sizes up to the maximal window size (three, in this case), and during evaluation it labels each window with its majority label, provided it had seen that window during training, otherwise backing off to smaller window sizes. This is similar to Daelemans and van den Bosch's [1997] IGTREE algorithm and to other decision tree learners that do not use pruning (see the discussion on page 66). The first non-baseline classifier in this comparison is the decision tree learner C4.5 [Quinlan, 1993], for which we had determined through cross-validation on the training data that disabling pruning and using unpruned trees for classification is indeed advantageous, since the pruned decision trees failed to beat the baseline. The ordinary C4.5 algorithm is not cost-sensitive. However, general techniques exist that turn arbitrary classifier learners into cost-sensitive classifier learners. The

	Pr. err.		*Pr. cost*		*Sym. err.*		*Str. err.*	
	abs.	*rel.*	*sum*	*avg.*	*abs.*	*rel.*	*abs.*	*rel.*
Baseline	12218	0.1673	12579	0.1722	12450	0.1705	7303	0.7325
C4.5	12149	0.1663	12510	0.1713	12395	0.1697	7289	0.7311
C4.5 + MetaCost	12201	0.1671	12554	0.1719	12423	0.1701	7292	0.7314

Figure 6.2: Comparison between ordinary training and cost-sensitive training of classifiers.

second classifier in this comparison is a combination of one such meta-classifier, namely MetaCost [Domingos, 1999], with ordinary C4.5 (using the same settings as before, i. e., without any pruning).

The results of the comparison are shown in Figure 6.2. The four columns indicate, respectively, prediction error, prediction cost, symbol error, and string error. The first thing to note is that both classifiers beat the baseline across all columns, although the improvements over the baseline are small. Crucially, the cost-sensitive classifier based on MetaCost controlling C4.5 performs worse than plain C4.5. This failure of cost-sensitive training does not rule out the possibility that other cost-sensitive learners would be more successful. There is still some advantage in performing cost-sensitive evaluation however: the second column lists the total and average prediction costs, which are only available with cost-sensitive evaluation. Because of the choice of the cost matrix, the total prediction cost is an upper bound for raw symbol error. In other words, in most practical situations raw symbol error will be bounded from below by raw prediction error and from above by the total prediction cost.

In Chapter 2 we had presented general theoretical arguments involving somewhat artificial data sets to show that minimizing prediction error (likewise for prediction cost) does not generally minimize symbol error or string error. The present section discussed several empirical aspects of the discrepancy between prediction error and symbol error in conjunction with a commonly used and more or less natural data set. We saw that in this specific situation raw prediction error and raw symbol error are typically not too far apart, but any observed difference should be analyzed, as it may indicate problems with the data, specifically inconsistent alignments. The answer to the question that forms the title of this section is negative: raw prediction error tends to underestimate raw symbol error in practice. If an upper bound on symbol error is needed one can carry out a cost-sensitive evaluation, where the misclassification costs are edit distances between the phoneme strings represented by the transcription symbols. While minimizing prediction error had some effect on the other evaluation measures, the relationship was indirect; the next section focuses on the direct minimization of an evaluation measure.

6.3 Brute-Force Word Error Minimization

The present section investigates the feasibility of minimizing empirical error directly through combinatorial search. Theoretical results demonstrating the general hardness of a problem, of the sort that we had seen in Chapter 3, may not reveal much about the problem instances one encounters in practice. For example, we have no reason to believe that pronunciation dictionaries for natural languages would correspond to intricate Boolean formulas specifically designed to trigger the worst-case behavior of a blind search algorithm. While blind search

t /t/	282
T /θ/	18
S /ʃ/	9
C /ʧ/	8
-	8
D /ð/	6
Total	331

Figure 6.3: Phonemes corresponding to ⟨t⟩ in the training data.

may be an option in practice, we can do even better if we take into account any inherent structure of the problem.

The specific problem is to find the globally optimal morphism that maximizes string (word) accuracy on the training data. In the following discussion, we use a 2,000 word random sample of the NETtalk data [Sejnowski, 1988] as training data. Since the NETtalk dictionary is aligned, we are dealing with an instance of MAX-VFMC. We decide to skip any preprocessing steps, i. e., to use no input context. Now consider, for example, the phonemes aligned with the letter ⟨t⟩. Figure 6.3 lists those phonemes (in the NETtalk transcription) together with occurrence counts.

The occurrence counts reflect the number of words in which the phoneme occurred together with the letter ⟨t⟩. In fact, we can eliminate words like ⟨that⟩, transcribed as /ð-æt/ or /ð-ət/ (the NETtalk dictionary lists both pronunciations), in which multiple occurrences of a letter, in this case ⟨t⟩, correspond to different phonemes, since no morphism in our search space has any hope of correctly producing the correct pronunciation.

The above table tells us that even in the most optimistic case we are guaranteed to make mistakes: if the optimal morphism were to map the letter ⟨t⟩ to the phoneme /t/, there would be $18+9+8+8+6=49$ words whose pronunciations it would mispredict. In other words, for each choice of how to pronounce the letter ⟨t⟩ we obtain upper bounds on the accuracy of the overall morphism. In this case, we know that the overall accuracy cannot exceed $2,000-49$ words. (Lower bounds are less useful, since it is very easy to find morphisms that mispredict on virtually all of the words.)

This provides us with a recursive scheme for refining the upper bound. We explore partial morphisms and start with an empty morphism and an upper bound of 2,000 words. We first consider all phonemes corresponding to, say, the letter ⟨t⟩. For each choice of phoneme we obtain a new bound and eliminate all training data that are mispredicted by the current choice. We recursively consider all other letters. Doing this naively would mean that we would have to explore approximately 19 trillion morphisms for this data set. But since we keep track of bounds, we can use a branch-and-bound strategy to prune away large subtrees of the search space. The globally optimal morphism must, by definition, be at least as accurate as any other morphism. This means that those parts of the search space can be eliminated for which the accuracy bound falls below the accuracy of the best feasible solution found so far.

A greedy strategy can be used to find an initial feasible solution. In this case, it is the morphism that maximizes prediction accuracy, which maps each letter to the phoneme it most frequently co-occurs with. This morphism is shown in Figure 6.4. It correctly predicts the pronunciation of 65 words in the training set.

This initial solution provided a useful bound for a subsequent branch-and-bound search for the globally optimal solution. The search took less than a minute

⟨a⟩	@	⟨n⟩	n
⟨b⟩	b	⟨o⟩	o
⟨c⟩	k	⟨p⟩	p
⟨d⟩	d	⟨q⟩	k
⟨e⟩	-	⟨r⟩	r
⟨f⟩	f	⟨s⟩	s
⟨g⟩	g	⟨t⟩	t
⟨h⟩	-	⟨u⟩	-
⟨i⟩	I	⟨v⟩	v
⟨j⟩	J	⟨w⟩	-
⟨k⟩	k	⟨x⟩	X
⟨l⟩	l	⟨y⟩	i
⟨m⟩	m	⟨z⟩	z

Figure 6.4: Letter-to-phoneme mapping that optimizes prediction accuracy.

on commodity hardware and is in our experience also feasible on much larger data sets. It revealed two tied solutions that correctly predict 93 words of the training dictionary. The tie was broken on a disjoint 1,000 word held-out data set (the only difference between the two morphisms was the pronunciation assigned to the letter ⟨h⟩). The letters for which the (now unique) globally optimal solution differs from the initial greedy solution are shown in Figure 6.5.

We selected 2,000 unseen words for evaluation and compared the accuracy of the greedy solution and the globally optimal solution. Performance on the unseen evaluation data was only slightly worse than on the training data. The results are summarized in Figure 6.6. More importantly, we wanted to make sure that the classification assigned by the optimal morphism is indeed different from the initial solution, from which it differs in only the six letters shown in Figure 6.5.

⟨e⟩	E	⟨o⟩	a
⟨h⟩	h	⟨u⟩	^
⟨k⟩	-	⟨w⟩	w

Figure 6.5: Letter-to-phoneme mapping that optimizes word accuracy (only showing letters for which it differs from Figure 6.4).

We applied the McNemar test [Dietterich, 1998] to the predictions of the two classifiers on the evaluation data, which confirmed that the differences are clearly significant at the 1% level ($\chi^2 = 10.0119$ at 1 degree of freedom, so $p < 0.00156$, i. e. significance is approaching the 0.1% level).

In sum, we have seen an example for which exhaustive search is both feasible and beneficial, since the error reduction on the training data results in an error reduction on unseen test data. However, since the induced mappings do not use any context for prediction, they are very simple and the search space is comparatively small (a bit less than 20 trillion hypotheses). Using a single letter of context for prediction results in a much larger search space (more than 10^{132} hypotheses) that cannot be searched exhaustively, not even with the minimal guidance provided by branch-and-bound. We noticed that the initial greedy solution can be improved by local hill-climbing search, and for the simple experiment reported above the local optimum it found was confirmed to be identical to a global optimum found by branch-and-bound. It is therefore reasonable to suspect that local search would be useful in situations where exhaustive search is no longer possible. This is the topic of the next section.

	Training		*Evaluation*	
	abs.	*rel.*	*abs.*	*rel.*
Greedy solution	65	3.25%	59	2.95%
Optimal solution	93	4.65%	89	4.45%

Figure 6.6: Comparison of word accuracy between the greedy and the optimal solution.

6.4 Local Search

Local search, or hill-climbing search, is an algorithm design technique with many applications [Aho et al., 1983, sec. 10.5] and is used here as a heuristic method for solving combinatorial optimization problems [Papadimitriou and Steiglitz, 1998; Hromkovič, 2001]. The idea is to define a *local change neighborhood*, a set of transformations of feasible solutions that produce new feasible solutions. During one iteration of local search, the change neighborhood is searched exhaustively, and the change that results in the largest improvement of the objective is applied. This is repeated iteratively until no further improvements can be found.

Local search was used to construct initial feasible solutions for the exhaustive search described in the previous section. Surprisingly, the solutions found by local search were always confirmed to be globally optimal by subsequent exhaustive searches. Since local search is generally more tractable than exhaustive global search, this is a welcome result, since it suggests that local search may suffice even when the optimality of a solution it produces cannot be confirmed by exhaustive search.

The following experimental setup, similar to that of Bakiri and Dietterich [2001] was used. A random 1,000 word subset of the NETtalk data was selected for initial training; another, disjoint 1,000 word subset for local search; and a third, disjoint 1,000 word subset for evaluation. An initial solution that optimizes prediction accuracy was obtained on the basis of the first training set. The classifier that was learned is a simple variation on standard decision trees [Quinlan, 1986], with the additional restriction that the features used for prediction are letters; furthermore, a new letter feature can only be inserted into the tree if it is contiguous to a letter feature already in the tree.

Then local search was carried out on the second training set. The change neighborhood is defined by the non-majority labels at the leaves of the initial decision tree. In other words, if a leaf of the tree was contained several class labels, the initial solution would choose the dominant label, and the runners-up would be added to the local change neighborhood for that leaf. During local search, these alternatives are tentatively inserted into the tree, which is then evaluated on the second training set. The objective during local search was to minimize string error.

Local search was able to reduce string error rate considerably on the training data, from 52.4% to 46% after 56 iterations, as shown in Figure 6.7, while the relative increase of string error and prediction error was much smaller. However, cross-validation on a held-out 1,000 word data set revealed that string error was almost unaffected, fluctuating only slightly around its initial value. The string error on a third set of unseen test data was 78.9% initially, and 79.9% after local search had found an optimum.

There are two possible conclusions one might draw from this. It could plausibly be the case that minimizing empirical string error has no beneficial effect

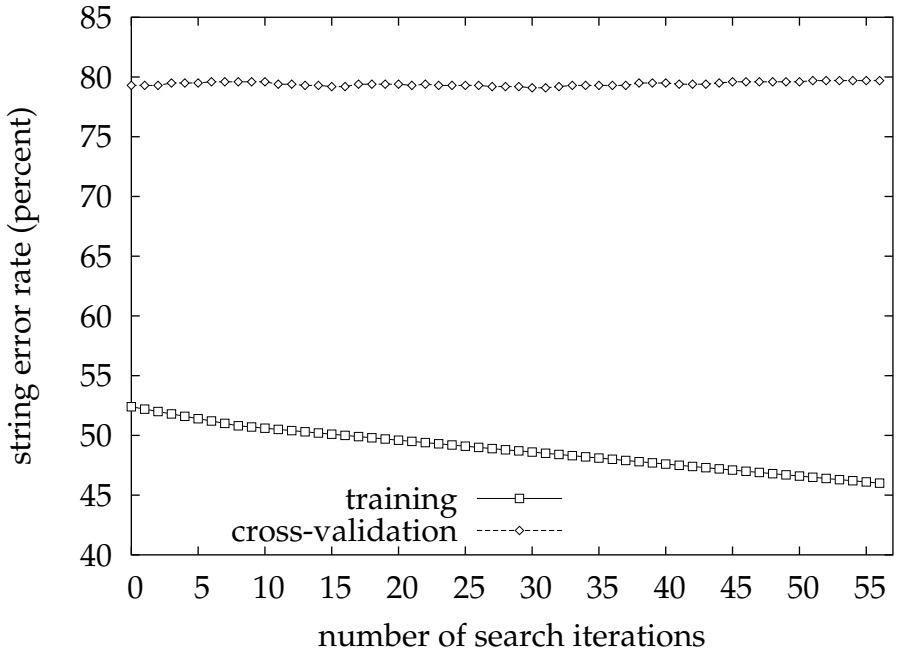

Figure 6.7: Minimization of string error using local search.

on the true error of the trained classifier. As we had seen in Section 2.4, and especially in Figure 2.11, minimizing string error often means making decisions in a way that increases prediction error and symbol error on words whose pronunciation one cannot expect to correctly predict. Such words are then "sacrificed", which potentially opens up new possibilities for mappings that reduce string error on other words. Another explanation for why minimizing empirical string error has little effect in the present case might be that we did not manage to find the true global minimum, because local search is not an admissible search method. This is less likely, but since the search for a better hypothesis than the one found by local search proved to be intractable in practice, there is no guarantee that a better solution, should one exist, would not result in a different performance on an unseen test set. Either way we are forced to conclude that minimizing string error is not useful in this case.

6.5 Bounds on the Performance of Classifier-Based Approaches

The preceding sections painted a rather bleak picture, suggesting that improvements to the quality of classifier-based approaches to learning letter-to-sound converters are somewhat elusive: in Section 6.2 we tried cost-sensitive classification without seeing any improvements in the quality of the predicted pronunciations; in Section 6.3 we saw that exhaustive search for an optimal classifier is only possible for very small problem instances; the local search heuristics described in Section 6.4 can deal with larger problems and managed to reduce empirical error on the training data, but the improvements did not generalize to cross-validation

and evaluation data. We may therefore wonder whether the classifier-based approaches are already performing close to a theoretical optimum, or whether large improvements are still possible. Since purely data-driven methods rely exclusively on their training data for generalization, any information absent from the training data cannot be used to classify future instances. We use this observation to establish empirical performance bounds for classifier-based approaches, asking the question whether information that is necessary for correct classification was completely absent from the training data.

We start out by asking whether the letter–phoneme pairings observed in the training data are in principle sufficient. In other words, are we likely to encounter a significant number of previously unseen letter–phoneme pairs in the future? A quantitative answer to this question immediately translates into a performance bound – though by no means a tight bound – since most classifier-based approaches cannot predict phonemes that were never associated with a particular letter in the training data.

One way to answer this question is to measure classification accuracy for a classifier that predicts a phoneme based solely on the corresponding letter, i.e., uses a sliding window of size 1. To obtain a performance bound, we make use of an oracle during classification (in the speech recognition literature this setup would be called a "cheating experiment"). If at all possible, the oracle magically selects the correct phoneme and the classifier outputs the oracle's choice as its prediction. But the oracle is not omniscient; it only chooses from among the phonemes that are known to have been associated with the given letter in the training data, so any unknown association will result in a classification error.

We again use the NETtalk data set without stress information and after the removal of duplicate entries that are only distinguished by different stress patterns,

leaving us with 19,940 unique entries. Our evaluation method is based on 5-fold cross-validation on the entire data set. After ten iterations of cross-validation we arrived at an estimate of 26.8 classification errors (standard deviation 4.1) for the 146,473 letters in the data, corresponding to a classification accuracy of 99.98%. This means that any realistic classifier (i. e. one that does not consult an oracle) is unlikely to ever perform at 100% accuracy, due to a small but nonzero number of unseen letter–phoneme combinations.

Although this demonstrates that almost all of the relevant pairings have been seen and the correct phoneme could, in principle, be chosen from among those paired with a given letter, we need to specify how exactly that choice should be made. The performance of instance-based learners depends one the size of the neighborhood and the distance metric. Decision trees perform differently at various levels of pruning. Performance of backed-off n-gram models [Fisher, 1999] varies with the choice of back-off strategy. The underlying question is always how much information goes into making the classification decision. For example, if we look at the letter ⟨p⟩ without any information about context, we might predict a phoneme /p/ as its pronunciation. But that guess might be different if we knew that ⟨p⟩ was followed by an ⟨h⟩. And we might revise our guess again if we knew that ⟨p⟩ occurred in the context of ⟨haphazard⟩.

When using a large window size, typically up to 7 as in [Sejnowski and Rosenberg, 1987], but sometimes as large as 15 [Bakiri and Dietterich, 2001], in principle any subsequence of the letters in a window may be used for prediction. Since the number of subsequences is exponential in the window size, many approaches only use contiguous substrings that contain the focus letter. This restriction, which can be justified empirically [van den Bosch and Daelemans, 1993], results in a number of contexts quadratic in the window size. On the other hand,

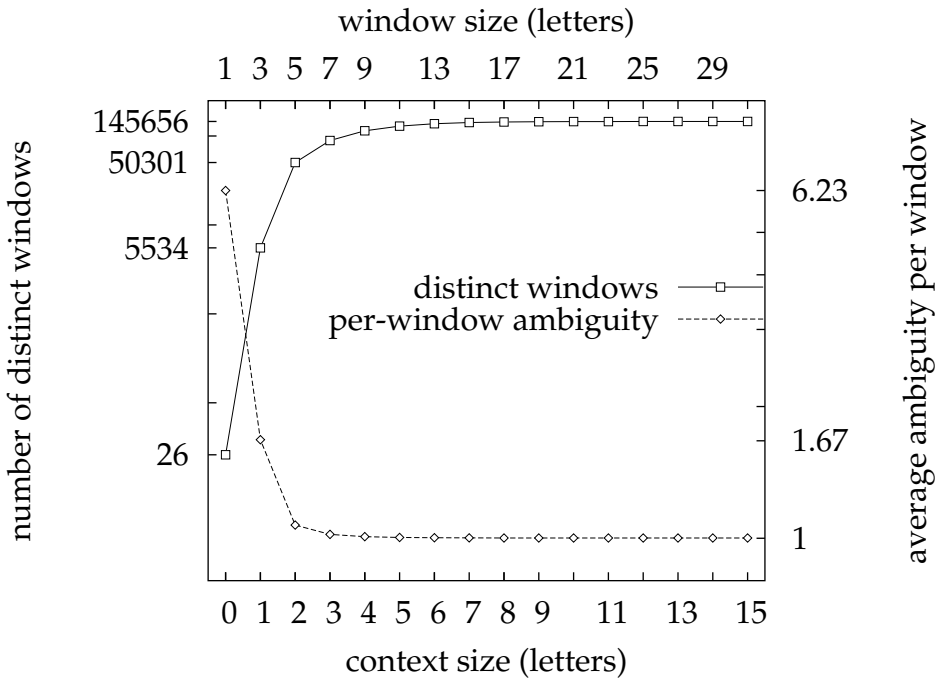

Figure 6.8: Number of distinct windows and average ambiguity per window as functions of context/window size.

the number of distinct windows a classifier has to store or analyze may in theory grow as an exponential function of the maximum window size used by the classifier. However, this number is always bounded by the amount of available training data. Figure 6.8 shows that the number of distinct windows as a function of context size quickly approaches its empirical maximum (the number of letters in the training data). We can also see that the average ambiguity per window approaches unity, which means that for the vast majority of windows there is precisely one label associated with it in the training data.

The setup is now exactly as before: given a maximum allowable window size, the classifier learner scans its training data and stores all context windows and the phonemes associated with them. During classification, an oracle chooses one of the matching stored windows and predicts the phoneme that was most frequently associated with it in the training data. This provides an upper bound on the classification accuracy of decision trees, backed-off n-gram models, instance-based learners, etc.

For a comparison with Bakiri and Dietterich [1993] we replicated their evaluation methodology: many of their experiments are based on training on a 1,000 word dictionary and evaluating on a second, disjoint 1,000 word dictionary. Using an oracle, we obtain a performance bound of 94.64% prediction accuracy and 68.57% string (word) accuracy for a seven-letter window. The baseline system of Bakiri and Dietterich [1993] achieves only 81.3% prediction accuracy (which they refer to as 'phonemes correct'), suggesting that there may be considerable room for improvement. Figure 6.9 shows bounds on prediction error and string error as functions of context size (note that the y-axis is logarithmic and that error bars indicate standard deviation). We can see that prediction error never falls below 5%, which is partly due to the fact that the classifiers were trained on a rather small dictionary comprising only 1,000 words. On the other hand, the maximally possible prediction accuracy exceeds even that of the best classifier built by Bakiri and Dietterich [1993].

Their best-performing system, which uses a 15-letter window, was trained on 19,002 words and achieves 93.7% prediction accuracy [Bakiri and Dietterich, 1993, p. 44]. We trained on 18,940 words (since we ignore stress we use a slightly smaller data set) and obtain an upper bound of 98.2%. This tells us that, in theory, the error of the best system developed by Bakiri and Dietterich [1993] could be

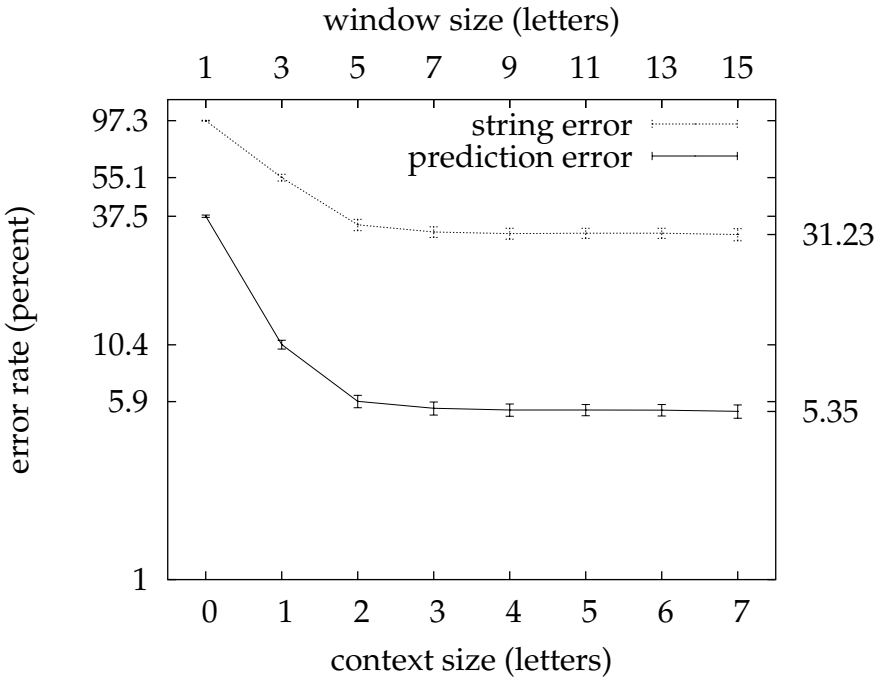

Figure 6.9: Performance bounds as functions of context/window size for 1,000 words of training data.

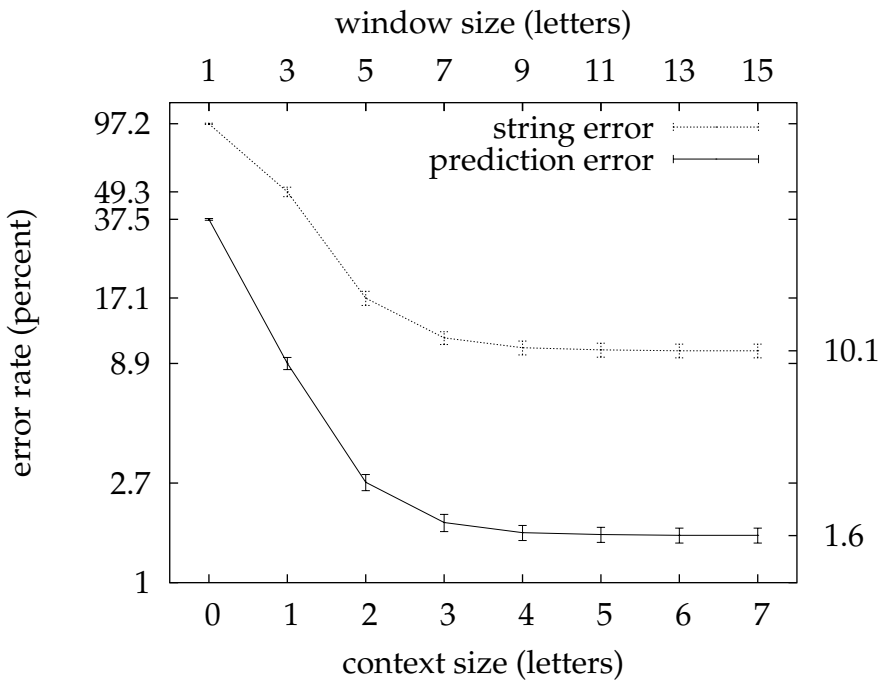

Figure 6.10: Performance bounds as functions of context/window size for approx. 19,000 words of training data.

reduced by a factor of three or four. Figure 6.10 plots bounds on prediction error and string error against context size, this time for a training dictionary with almost 19,000 entries. Compared with Figure 6.9 the maximally possible accuracy is much higher due to the larger training set, yet there clearly still exists a performance ceiling at around 98.5% prediction accuracy and 10% word accuracy. In other words, even if the theoretically best possible performance could be achieved for this class of models, the resulting letter-to-sound converter would still get the pronunciation of one out of every ten words in the dictionary wrong.

The performance of a classifier-based converter depends both on the amount of context used for prediction and on the amount of available training data. Figure 6.9 and Figure 6.10 showed two single-dimensional snapshots of a multidimensional function that depends also on the size of the training set. In Figure 6.11 we plot the lower bound on prediction error against both context size and training set size. The important aspect of this figure is that the lower bound has leveled off for five or more letters of context and 16,000 or more words in the training dictionary, so that prediction error is unlikely to ever drop below 1.5%. But since the average word in the NETtalk data set contains about 7.3 letters, and assuming prediction errors are independent, this means that string (word) error will remain above 10%. In practice performance is likely going to be worse, since these bounds were derived under unrealistic assumptions involving oracles that can make use of atypical and infrequent patterns in the training data.

Now that the best-case performance results of the approach under discussion have been delineated, there are two possible lines of investigation. First, one could strive to refine existing techniques and push them as close as possible to their theoretical limits. The comparison with Bakiri and Dietterich's [1993] work indicates that there is still much room for possible improvements, though this comparison does not directly lead to suggestions regarding where to look for improvements. Second, one can also look for different approaches with better best-case performance bounds, hoping that the practically achievable performance is higher than for the classifier-based approach described here.

The literature contains relatively few attempts to identify rigid performance bounds. For example Brill et al. [1998] conduct experiments to determine what kinds of improvements on n-gram language models are possible, but they not

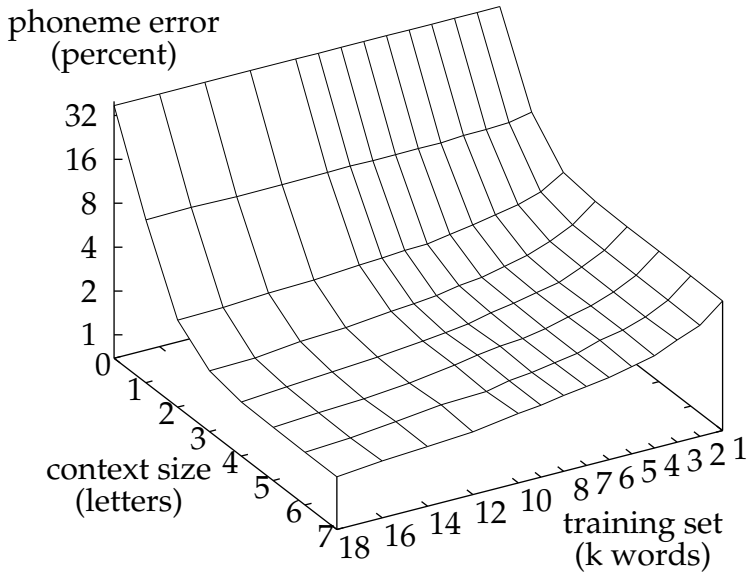

Figure 6.11: Bound on prediction error as a function of both context/window size and training set size.

establish clear upper bounds. Within the area of text-to-speech conversion, Marchand and Damper [2000, sec. 7.1] investigate upper and lower bounds for decoding strategies used in analogical modeling, which only addresses a specialized subproblem of letter-to-sound conversion. Damper [2001b, p. 20] suggests that evaluating a letter-to-sound converter on its training data set can be seen as establishing an upper bound on its performance. However, such a bound is necessarily very optimistic, as it does not take differences between training data and evaluation data into account. But it is precisely the existence of such differences that makes the letter-to-sound conversion problem hard, for both humans and machines, as there are always words whose pronunciations could not have been predicted on the basis of past experience.

Ideally we would also like to establish similar bounds for the transduction-based approach, which we turn to next. However, in that approach one usually models the probability of any conceivable transduction, which means that very unlikely pronunciations that could not have been predicted from any information in the training data are usually still present in hypothesis graphs. If hypothesis graphs are pruned to remove unlikely candidates, the true pronunciation of a word might drop out, which would then contribute to a performance bound, as that would reveal a shortcoming of the model. We will see an example of this later on.

6.6 Effects of Viterbi Training and Decoding

The rest of this chapter is concerned with the transduction-based approach to letter-to-sound conversion developed in Chapter 4 and Chapter 5. We begin with a comparison of various internal issues related to training (parameter estimation)

and decoding of stochastic finite state transducers. A comparison between the transduction-based approach and classifier-based approaches will follow in Section 6.7.

The possibility of taking shortcuts during the training of stochastic transducers had been pointed out at the end of Section 4.3.1. The choice is between proper EM training and a heuristic variant of it, so-called Viterbi training. A similar approximation is possible during decoding, as described in Section 4.5 and Section 5.5, which can either find the best string (MAP decoding) or the label of the best path (Viterbi decoding).

Here we compare the effects of using Viterbi training vs. proper EM training, and of Viterbi decoding vs. MAP decoding. Two disjoint 1,000 word subsets of CMUdict were selected as training and test data. A stochastic transducer with one-letter memory was trained on the training data and evaluated on the test data.

Before training, the model parameters were initialized with uniform random values on the unit interval, normalized, and saved. Then 20 iterations of EM training and Viterbi training were carried out, starting from the same saved stated of the initial model. Figure 6.12 shows the increase in log likelihood during training. Viterbi training is slightly faster and achieves a high relative convergence of log likelihood after fewer iterations than EM training. However, one can see clearly that the likelihood of the model built using Viterbi training is always lower than for EM training. This was confirmed across several repetitions of this procedure, for different random initialization of the model parameters.

A difference in likelihood need not have any effect on the quality of the predictions made by the differently trained models. Each model was evaluated on the unseen test data. Furthermore, two decoding strategies were compared. Viterbi

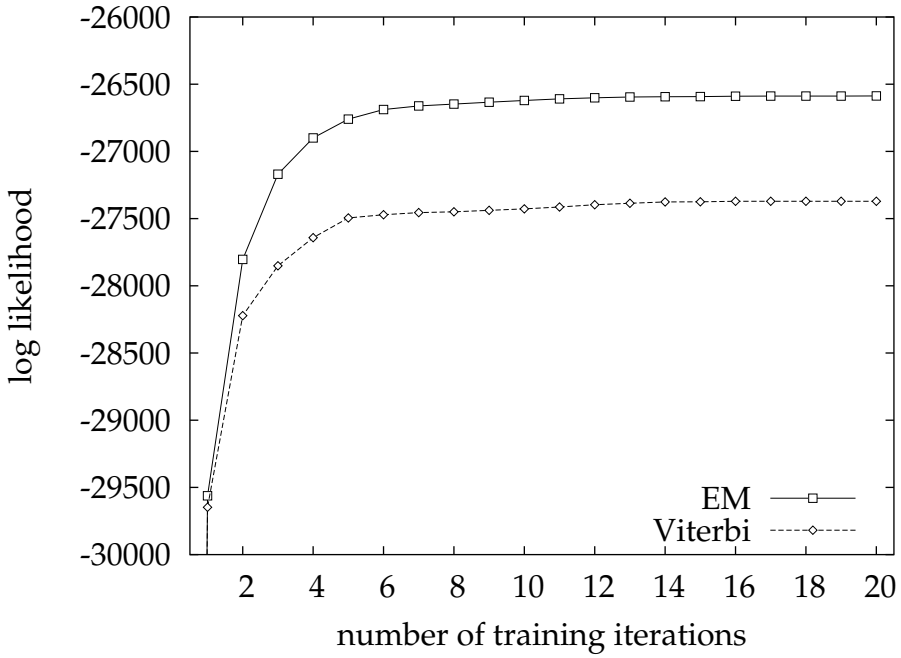

Figure 6.12: Comparison of log likelihood during EM training and Viterbi training.

Training	*Decoding*	*SymER*	*StrER*
Viterbi	Viterbi	36.33%	95.6%
	MAP	36.20%	95.6%
EM	Viterbi	34.51%	92.4%
	MAP	34.33%	92.5%

Figure 6.13: Comparison of errors made by the same transduction model for different training and decoding strategies.

decoding refers to selecting the label of the most probable path of the composed transducer, as described in Section 5.5. What we call MAP decoding here is not an exhaustive search, but refers to finding the most probable phoneme string among all possible phoneme strings up to a fixed length.

Figure 6.13 shows the results of the evaluation. We can see that Viterbi training results in higher symbol error rate (where symbol error is Levenshtein distance) and string error rate than EM training. For each trained model, MAP decoding is also better than Viterbi decoding, though the differences are not as big as the differences arising from the two training methods. This means that proper EM training should always be preferred over Viterbi training. Although it is slightly slower, this should not matter much, since training happens off-line. Full MAP decoding is also better than Viterbi decoding, but is much slower. For practical applications, Viterbi decoding may therefore be a reasonable shortcut to take.

6.7 Classification vs. Transduction

The preceding section demonstrated that the combination of EM training and MAP decoding for stochastic transducers results in higher quality of predicted

pronunciations compared with Viterbi approximations. In this section we therefore primarily use EM training and MAP decoding when working with stochastic transducers, which we are about to compare with a somewhat more traditional approach to letter-to-sound conversion. Among the classifier-based approaches, Sproat's [2001] *Pmtools* stands out in two respects: first, unlike NETtalk and its close relatives, it does not use a fixed inventory of multi-phoneme symbols, instead constructing such special representations as needed, based on the training data; second, unlike all other classifier-based approaches, an iterative training procedure is employed, whereby an induced classifier is used to recompute the best alignment of the training data. In all other respects the *Pmtools* algorithm is comparable to other classifier-based approaches, though it should arguably be preferred because of its distinguishing characteristics. We use it as a representative of the classifier-based approaches and compare it with our transduction-based approach.[1]

The training and evaluation data for the present comparison were obtained in the following way. First, a subset of CMUdict was selected whose entries are purely alphabetic, i.e., which excludes numbers and symbols. The resulting dictionary contains 119,114 word forms. Second, the orthographic words in CMUdict were intersected with those found in the NETtalk data set, resulting in 15,402 unique orthographic words. Third, the alphabetic subset of CMUdict was joined (using the Unix command `join`) with the list of words common to CMUdict and the NETtalk dictionary, yielding a 16,697 word form subset of CMUdict. From this dictionary a subset consisting of 10,000 randomly chosen entries was selected for

[1]Richard Sproat's help with the present comparison is gratefully acknowledged, in particular his role as the user-friendly interface to *Pmtools*.

	SymER	*StrER*
Pmtools	41.56%	92.04%
SFST *Viterbi*	35.39%	93.04%
SFST EM/MAP	34.01%	92.18%
Topline		8.46%

Figure 6.14: Comparison of approaches based on classification vs. transduction.

training, and a disjoint subset of 5,000 randomly chosen entries was set aside for evaluation.

As in the previous section, all models used a two-letter window with one letter of left context. The main comparison is between *Pmtools* and a stochastic finite state transducer (SFST) employing EM training and MAP decoding. More specifically, MAP decoding was approximated using the 2,000 most likely paths (see Section 5.5). While both approaches made at least one mistake on the vast majority of words, their performance differs noticeably when measured in terms of symbol error rate (Levenshtein distance). The stochastic transducer performed at 34% symbol error rate, which is of course comparable to the best result from the previous section (see Figure 6.13). This could not be matched by *Pmtools*, which had a much higher error rate of more than 40%. This even exceeds the symbol error rate of 35.4% for a stochastic transducer for which the Viterbi approximations were used during training and decoding. These results are summarized in Figure 6.14. The label SFST refers to the approaches based on stochastic finite state transducers; the additional label *Viterbi* indicates that Viterbi approximations were used, as opposed to the preferred EM training and MAP decoding. The

line labeled *topline* shows the best hypothetical performance of a transduction-based approach, which was determined by an oracle that can tell whether the correct pronunciation of a word is present among the 2,000 most likely paths used during decoding. If the correct pronunciation could magically be picked out whenever it was present, the overall string error rate would fall below 10%. Unlike in the classifier-based approaches discussed in Section 6.5, this is not an absolute performance bound, since it crucially depends on the size of the hypothesis graph.

The present comparison also provides further backing for our claim that empirical comparisons of letter-to-sound converters should not be based on string error rate alone. The string error rates of *Pmtools* and the stochastic transducer are very close (corresponding to a raw difference of seven words), yet there is a pronounced difference in symbol error rate. Especially when error rates are high, as they are here, real differences can easily remain hidden if one only looks at string error rate. In light of this the view held by Damper et al. [1999] that string error rate should be preferred because it results in higher and therefore more conservative numbers is particularly disturbing.

6.8 Modeling Word Length

Working with joint transducers means working with bivariate models. This leads to issues that may not have been addressed in the traditional univariate HMM literature. The purpose of this section is to draw attention to these problems, specifically the problem of modeling the length of words.

The model used in the preceding section was not properly equipped to deal with differing lengths of letter strings and phoneme strings. It tends to predict

phoneme strings that are about as long or slightly shorter than their corresponding phoneme strings, but it cannot handle the full range of variation in lengths. We have been working with joint models for convenience, since parameter estimation is relatively straightforward. But this means that we are modeling both the distribution of the lengths of letter strings as well as the conditional distribution (regression function) of the lengths of phoneme strings given a particular letter string.

Dealing with univariate length distributions is relatively straightforward. For the purpose this discussion we assume that we are developing a model for the language $\{x\}^*$ that assigns probabilities to strings such as xx or $xxxxx$. Take the orthographic words of the NETtalk data set for example, and ignore the identities of the letters, or replace all letters with x. In the simplest case a memoryless univariate model – a one-state stochastic automaton – gives rise to a geometric distribution. However, a geometric distribution cannot adequately model the observed distribution of word length among the NETtalk data.

Consider Figure 6.15, which shows the empirical distribution of word length in the NETtalk data set, and also the word counts under a fitted geometric model. The geometric distribution is a one-parameter model with a specific fixed shape, and it is obvious that no geometric distribution will even remotely resemble the empirical distribution. The data clearly call for a richer model.

One model that fits the observed data reasonably well is the stochastic transducer shown in Figure 6.16. It has five free parameters. Observe first that it is a two-component mixture with mixing parameter m. The top component has three free parameters, p_1 through p_3, and assigns nonzero probabilities to strings of lengths one through four. The bottom component is a kind of negative binomial distribution with one free parameter p that assigns nonzero probabilities to strings of

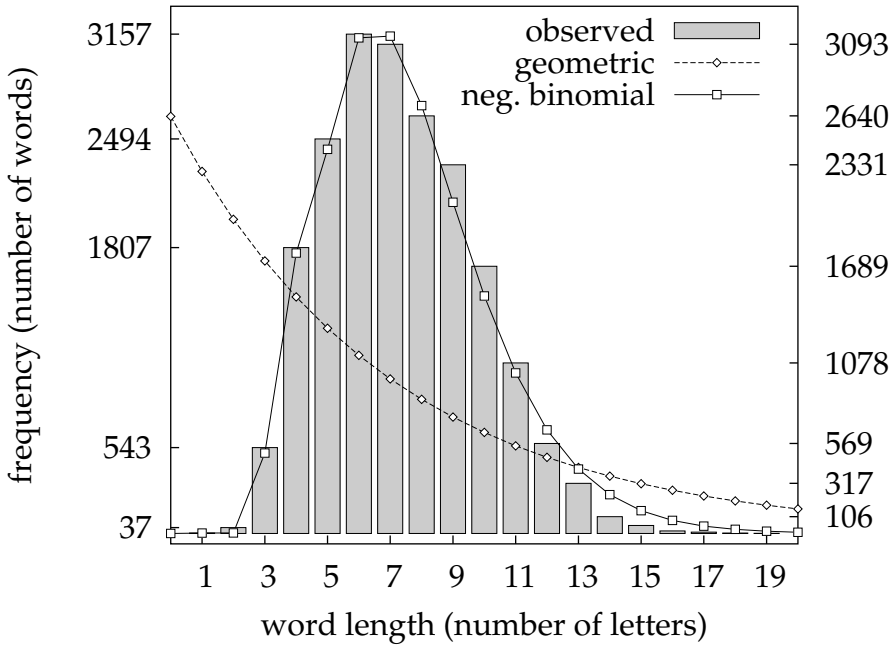

Figure 6.15: Distribution of word length in the NETtalk data set.

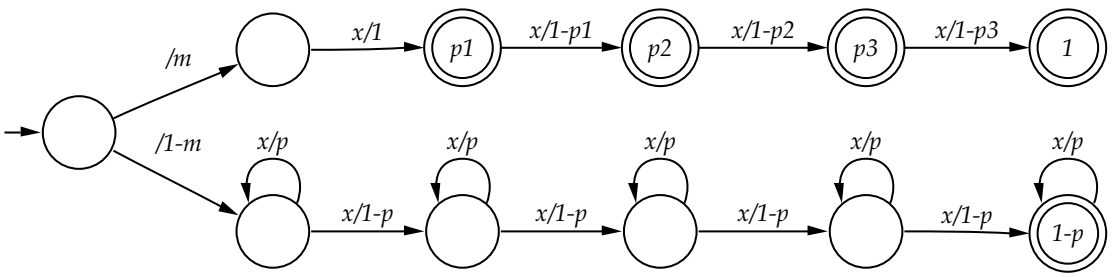

Figure 6.16: A stochastic automaton giving rise to a negative binomial model of string length.

length four or more [see also Durbin et al., 1998, sec. 3.4]. The predicted word counts under a fitted model of this form are shown in Figure 6.15 and should be contrasted with the predictions made by the memoryless geometric model.

Building a language model for the orthographic words of the NETtalk dictionary on the basis of this model of word length is already somewhat challenging, due to the presence of parameter tying in the length model and due to its interaction with the language model proper. This would be a good test case for Eisner's [2002] approach, which was specifically designed for dealing with complex parameter tying. However, it is not clear how any of this carries over straightforward to the bivariate setting of joint stochastic transducer models, where one not only has to account for the variation in length of orthographic and phonemic strings, but also for their covariance. Building discrete bivariate models, preferably with relatively few parameters, of joint length distributions is an important issue that needs to be addressed when one wants to work with pairs of strings of unequal and highly variable lengths.

6.9 Conclusion

In this chapter we looked at a number of empirical issues that corroborate or shed new light on some of the theoretical claims discussed in earlier chapters. There had been a theoretically motivated concern (see page 42) that prediction error may make spurious distinctions and might overestimate symbol error. It turned out that on average the opposite situation holds in practice, namely prediction error tends to underestimate symbol error. Cost-sensitive evaluations can be carried out to determine prediction cost, which in practice provides an upper bound on symbol error. In other words, symbol error is typically sandwiched between prediction error and prediction cost, and if both are minimized then symbol error is indirectly minimized. This gives some new justification to the traditional classifier-based approaches, which generally minimize prediction error.

Section 6.3 showed how string error can be minimized directly using guided branch-and-bound search. While this is feasible and useful for small problem instances, it quickly becomes intractable. A greedy local heuristic was used in Section 6.4, but although it can deal with larger problems the reduction of empirical error did not result in improvements during cross-validation and evaluation. We concluded that trying to minimizing string error directly is not a worthwhile goal. This is an important result because string error rate had, to our knowledge, never been used directly as an optimization criterion. At most it was used indirectly: for example, Bakiri and Dietterich [2001] compare several improvements on the NETtalk classification design and generally prefer feature inventories and decoding strategies that increase string accuracy, sometimes even over other designs that only improve phoneme accuracy.

The practical difficulties associated with minimizing string accuracy are one of the reasons string accuracy should not be used as an evaluation criterion or optimization objective. Another reason was given at the end of Section 6.6: in that section we compared a classifier-based approach to letter-to-sound conversion with our own transducer model. The two approaches were virtually indistinguishable in terms of string error rate, while there was a large difference in terms of symbol error rate. This confirms our suspicion that string error rate is a crude metric that considers highly flawed approaches as equal even though there may exist clear differences in the amounts of symbol-level errors.

Section 6.5 presented lower bounds on the prediction and string error rate of classifier-based approaches trained on the NETtalk data set, or, equivalently, upper bounds on best-case accuracy. It turned out that the current state of the art in classifier-based letter-to-sound conversion is nowhere near its best-case optimum. We also established clearly what previous authors [for example Stanfill and Waltz, 1986] had noted, namely that the letter-to-sound conversion task is inherently difficult if the only information one can consult is local letter context, since a fair number of pronunciations one encounters could not have been predicted on the basis of known letter–phoneme associations. For the NETtalk data set we estimate that the predicted pronunciations of at least 10% of previously unseen entries will necessarily contain mistakes.

For the transduction-based approach to letter-to-sound conversion, we compared the effect of Viterbi training vs. EM training and Viterbi decoding vs. MAP decoding. Not surprisingly, Viterbi decoding is considerably faster than MAP decoding, and it is also slightly less accurate. A less obvious result is that Viterbi training yields noticeably inferior models compared with EM training, without being much faster. We therefore recommend EM training when working with

stochastic transduction models. If speed of decoding is an issue, Viterbi decoding is a useful approximation. Moreover, speed can be traded off for increased accuracy by decoding based on the n most likely paths for $n \geq 1$, where the case of $n = 1$ is regular Viterbi decoding. Training speed is less of an issue, since training happens off-line. Viterbi training tends to converge earlier then EM training and gets stuck without the likelihood increasing to the level reached during EM training.

In Section 6.7 we compared our transduction-based approach with a sophisticated classifier-based approach. This was a balanced comparison, because the two learners had access to exactly the same training and evaluation data and had been configured to use the exact same information sources. The approach developed in Chapter 5 had noticeably lower symbol error rate than the classifier-based converter, even when we used the Viterbi approximations.

Finally, Section 6.8 pointed out modeling issues that are different and potentially more complex for joint transduction models, which realize bivariate distributions, than for univariate HMMs. In order for stochastic finite state transducers to be useful for modeling real-world data, some support for parameter tying and complex parameter interactions appears to be necessary.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The focus of this thesis has been on foundational issues related to modeling rational string-to-string transductions. This modeling task is important for language and speech technology, but is also relevant for other areas, including bioinformatics. A simplified problem that involves same-length transductions has been studied at an abstract level in the machine learning literature, sometimes under the label 'machine learning for sequential data' [Dietterich, 2002]. However, this label is misleading, as sequential data comes in many varieties and is certainly not always adequately modeled by deterministic rational same-length transductions. This is true for many language and speech processing applications. The concrete application that has been used throughout this thesis is the task of predicting the pronunciation of an English word on the basis of its written representation. We saw in Chapter 2 and Chapter 3 that the typical data for this task violate the same-length assumption: the phoneme strings one wants to predict are most often shorter than the letter strings they correspond to. This seems to suggest that the existing machine learning techniques reviewed by Dietterich [2002] are simply not applicable, since they assume same-length transductions. Many authors have therefore tried to alter their data so that existing research on same-length modeling, and especially on classification, would remain applicable. Chapter 3 argued that such approaches do not pay enough attention to their preprocessing

steps in which the primary training data are altered ("aligned") and cannot make any useful guarantees about optimality of the inferred hypotheses. On the other hand, if the alignment step is taken seriously and integrated with the learning task, one is faced with intractable problems for even the simplest learning tasks. By contrast, Chapter 4 addressed the learning problem directly without reducing it to classification, albeit under very simplistic assumptions. The stochastic transduction model due to Ristad and Yianilos [1998] can model memoryless stochastic transductions without the same-length assumption. Many of its simplifying assumptions that render it memoryless can easily be dropped, and such generalizations can be obtained naturally when the memoryless model is presented, as it was, in terms of weighted finite state transducers. Generalizations of the memoryless transduction model were presented in Chapter 5. A number of problems in Chapter 3 through Chapter 5 had to be left open and should be addressed by future research. Among them, the questions of whether decoding for memoryless transducers is **NP**-hard and whether algorithms that currently involve determinization could profitably be reformulated in terms of disambiguation are the most practically relevant issues.

We avoided the question of how the transducer topology should be determined, assuming that it had been fixed in advanced. Inferring a suitable transducer topology from data could proceed along the lines of model selection or model merging [Stolcke and Omohundro, 1993]. Certain classes of topologies may have desirable properties that simplify some of the algorithms that operate on stochastic transducers. A clear example of this are memoryless transducers, for which the Forward algorithm is a specialized procedure that combines transducer composition and algebraic distance computations. However, the class of topologies for which the classical Forward algorithm is applicable appears to be much richer.

For example, the familiar Forward trellis graph (see Figure 4.2) arises from many other specialized topologies, in particular for transducers whose states are fully determined by the labels of the paths that reach them. A detailed discussion of this rather general special case will be the topic of forthcoming work.

The general transduction model developed in Chapter 5 is relevant for many language and speech processing tasks that cannot be modeled naturally in terms of deterministic same-length transductions or in terms of memoryless stochastic transductions. A number of examples had been given in Chapter 1. Among them, information extraction has received an increasing amount of attention recently and is likely to grow in importance in the future. It is clear that information extraction can benefit from, and sometimes virtually requires, the use of transductions. For example, when extracting telephone numbers from speech transcripts, spoken numbers should be converted to digit strings, since the length of the digit string is a much better indicator of whether a spoken number is a phone number than the number of spoken words is [Jansche and Abney, 2002]. In this example, the transduction was modeled using a small hand-crafted finite state transducer without any algorithmic training on empirical data. While this may have been adequate for a small and relatively clearly defined domain like telephone numbers from the North American Numbering Plan, larger and/or more diverse domains could certainly benefit from automatic training methods. The performance of an information extraction component is usually measured in terms of its precision and recall, or rather in terms of their harmonic mean, the so-called F-measure. In general, training an information extraction component should therefore mean optimizing its F-measure. The implications of this simple and straightforward formulation of the learning task have not been discussed in the literature. In fact,

existing papers on information extraction often mention precision, recall, and F-measure only in conjunction with the evaluation of information extraction components, whereas the optimization criterion used during the training of these components is usually not related to the evaluation measures in any direct way and is often left implicit. As in the case of letter-to-sound rules, using the preferred evaluation measures as the optimization objectives during training would very likely render the learning task extremely difficult. In fact, it may be much harder than learning letter-to-sound rules, since essentially two diverging global criteria (precision and recall) are involved. In other words, while the basic modeling techniques may be the same for information extraction and letter-to-sound conversion – stochastic finite transductions seem adequate – it is the difference in optimization criteria that renders the information extraction problem very different from, and intuitively harder than, the letter-to-sound problem. A full characterization of the information extraction learning problem would be a worthwhile topic for future research.

While the task of letter-to-sound conversion served as a source of examples throughout this thesis, specific aspects of this concrete task were discussed in Chapter 2 and Chapter 6. Chapter 2 provided a systematic comparison of evaluation metrics that had been proposed and used for the letter-to-sound task. We argued that the preferred evaluation metric should at the very least be fine-grained and universally applicable. Tallying the number of correctly predicted words, as Damper et al. [1999] do, is universally applicable, but must be dismissed because it is a coarse measure and can easily distort perceived differences. It is subsumed by string edit distance [Wagner and Fischer, 1974; Kruskal, 1983], which is a useful general metric. However, ordinary string edit distance (Levenshtein distance) has two drawbacks. First, because it is additive, combining it with probabilities in

minimum risk decoding (Section 5.5) is challenging [Mohri, 2002a]. Second, Levenshtein distance uses uniform weights that do not correspond to perceived phonetic similarity. The development of a simple measure of phonetic similarity or confusability is a very useful avenue for further research. Such a measure would be beneficial for many other areas, such as phonetics and/or psycholinguistics research or forensic linguistics, plus it could form the basis of lexical confusability measures [Fosler-Lussier et al., 2002]. In order to be applicable, such a measure must be able to quantify the difference between any two phon(em)es, which includes comparisons of vowels with stop consonants etc., which the traditional phonetics literature usually ignores.

In order for reported results on letter-to-sound conversion to be comparable, the field should establish a standardized set of evaluation data. For English alone there appear to be at least a dozen different data sets in use, which include different amounts of annotation, different transcription conventions, different phoneme inventories, etc. Comparisons across different data sets seem utterly futile, unless dictated by necessity [Damper et al., 1999]. In many other subfields, de facto standards for training and evaluation do exist, for example, the use of certain sections of the Penn Treebank in part-of-speech annotation and parsing, or the ongoing debate of evaluation in word sense disambiguation. Lamenting the current state of the field in letter-to-sound conversion evaluation would be about as useless as hastily declaring an existing data set to be the basis of a new standard. However, at least for English a preliminary standard should be established, which should satisfy the following criteria: it should be available at no cost and freely distributable; it should be representative of the general vocabulary of a single dialect; the status of pronunciation variants must be clearly indicated; it must not include alignment information. None of the data sets discussed in Chapter 2 satisfy

these criteria: many data sets are proprietary and have restrictive licenses; some data sets have odd vocabulary gaps, others include inconsistent pronunciations that appear to originate from different dialects; some data sets (especially CMUdict) do not differentiate between dialectal variation (which one may choose to ignore), variation due to sense and/or part-of-speech distinctions (which should be modeled), and variation that is due to speaking rate (which should be kept consistent). Alignment information along the lines of the NETtalk data set should be excluded for pedagogical reasons: the undue influence of NETtalk introduced the letter-to-sound conversion task to the machine learning community, which has access to the NETtalk data set via the UCI Machine Learning Repository [Blake and Merz, 1998]; because the focus has often been exclusively on NETtalk, its approach in terms of aligned data has come to dominate the machine learning approaches to the problem. If a future standard data set for training and evaluating letter-to-sound converters explicitly excluded alignment information, the field as a whole would be forced to think carefully about evaluation measures and we might eventually see the development of novel approaches no longer based on classification.

BIBLIOGRAPHY

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974. 172, 173

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, Reading, MA, 1983. 221

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, Germany, 1999. 56, 97

R. H. Baayen, R. Piepenbrock, and L. Gulikers. CELEX2. CD-ROM, catalog number LDC96L14, Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA, 1996. 19

R. H. Baayen, R. Piepenbrock, and H. van Rijn. The Celex lexical database. CD-ROM, Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA, 1993. 19

R. Harald Baayen. *Word Frequency Distributions*. Number 18 in Text, Speech and Language Technology. Kluwer, Dordrecht, The Netherlands, 2001. 1, 35

L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer. A new algorithm for the estimation of hidden Markov model parameters. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 493–496, 1988. 200

Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 5(2):179–190, March 1983. 200

Ghulum Bakiri and Thomas G. Dietterich. Performance comparison between human engineered and machine learned letter-to-sound rules for English: A machine learning success story. In *18th International Conference on the Applications of Computers and Statistics to Science and Society*, Cairo, Egypt, 1993. 37, 66, 228, 231

Ghulum Bakiri and Thomas G. Dietterich. Constructing high-accuracy letter-to-phoneme rules with machine learning. In Damper [2001a], pages 27–44. 25, 27, 28, 37, 66, 73, 222, 226, 243

Cyril Banderier and Sylviane Schwer. Why Delannoy's numbers? In *5th Lattice Path Combinatorics and Discrete Distributions Conference*, Athens, Greece, June 2002. 116

G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad. *Computational Complexity and Natural Language*. Computational Models of Cognition and Perception. MIT Press, Cambridge, MA, 1987. 84

Jean Berstel and Christophe Reutenauer. *Rational Series and their Languages*. Springer-Verlag, Berlin, Germany, 1988. 105

Alan W. Black, Kevin Lenzo, and Vincent Pagel. Issues in building general letter to sound rules. In *Proceedings of the 3rd ESCA Workshop on Speech Synthesis*, pages 77–80, 1998. 66

C. L. Blake and C. J. Merz. UCI repository of machine learning databases. Electronic document collection, Dept. of Information and Computer Sciences, University of California, Irvine, CA, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. 36, 251, 264

Gosse Bouma. A finite-state and data-oriented method for grapheme to phoneme conversion. In *Proceedings of the First Conference of the North-American Chapter of the Association for Computational Linguistics*, pages 303–310, 2000. 66

Chris Brew. Letting the cat out of the bag: Generation for shake-and-bake MT. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 610–616, Nantes, France, August 1992. 84

John Bridle. Optimization and search in speech and language processing. In Cole et al. [1997], chapter 11.7, pages 365–369. 133

Eric Brill, Radu Florian, John C. Henderson, and Lidia Mangu. Beyond n-grams: Can linguistic sophistication improve language modeling? In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 186–190, Montreal, QC, 1998. 71, 231

Francisco Casacuberta and Colin de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In Oliveira [2000], pages 15–24. 84, 152, 158

Ananlada Chotimongkol and Alan W. Black. Statistically trained orthographic to sound models for Thai. In *Proceedings of the Sixth International Conference on Spoken Language Processing*, Beijing, China, October 2000. 66, 108

Kenneth Church. Stress assignment in letter to sound rules for speech synthesis. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 246–253, Chicago, IL, July 1985. 27

Alexander Simon Clark. *Unsupervised Language Acquisition: Theory and Practice*. PhD thesis, University of Sussex, Brighton, England, September 2001. 122, 128, 155, 160, 161, 170, 177, 178, 183

Ron Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Batista Varile, Annie Zaenen, Antonio Zampolli, and Victor Zue, editors. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press and Giardini, Cambridge, England and Pisa, Italy, 1997. 253, 258, 263, 265

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1st edition, 1990. 116, 141, 144, 153, 154, 161, 171, 172, 173, 174

Walter Daelemans, Antal van den Bosch, and Jakub Zavrel. Forgetting exceptions is harmful in language learning. *Machine Learning*, 34(1):11–41, 1999. 65, 66

Walter M. P. Daelemans and Antal P. J. van den Bosch. Language-independent data-oriented grapheme-to-phoneme conversion. In Jan P. H. van Santen, Richard W. Sproat, Joseph P. Olive, and Julia Hirschberg, editors, *Progress in Speech Synthesis*, pages 77–89. Springer, New York, NY, 1997. 65, 66, 75, 77, 214

R. I. Damper, editor. *Data-Driven Techniques in Speech Synthesis*. Number 9 in Telecommunications Technology and Applications. Kluwer, Boston, MA, 2001a. 253, 254

R. I. Damper, Y. Marchand, M. J. Adamson, and K. Gustafson. Evaluating the pronunciation component of text-to-speech systems for English: A performance comparison of different approaches. *Computer Speech and Language*, 13(2):155–176, April 1999. 18, 21, 33, 36, 54, 55, 57, 239, 249, 250

Robert I. Damper. Learning about speech from data: Beyond NETtalk. In Damper [2001a], pages 1–25. 233

C. de la Higuera and F. Casacuberta. Topology of strings: Median string is NP-complete. *Theoretical Computer Science*, 230(1–2):39–48, January 2000. 84, 202

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Jornal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977. 10, 127

Markus Dickinson and W. Detmar Meurers. Detecting errors in part-of-speech annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 107–114, Budapest, Hungary, 2003. 30

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998. 220

Thomas G. Dietterich. Machine learning for sequential data: A review. In Terry Caelli, Adnan Amin, Robert P. W. Duin, Mohamed S. Kamel, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002*, number 2396 in Lecure Notes in Computer Science. Springer, Berlin, Germany, 2002. 36, 246

Thomas G. Dietterich, Hermann Hild, and Ghulum Bakiri. A comparison of ID3 and backpropagation for English text-to-speech mapping. *Machine Learning*, 18(1):51–80, 1995. 65, 66

Michael Divay and Anthony J. Vitale. Algorithms for grapheme-phoneme translation for english and french: Applications for database searches and speech synthesis. *Computational Linguistics*, 23(4):495–523, 1997. 16

Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, August 1999. 215

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, England, 1998. 115, 117, 128, 160, 162, 163, 170, 177, 242

Thierry Dutoit. *An Introduction to Text-to-Speech Synthesis*. Number 3 in Text, Speech and Language Technology. Kluwer, Dordrecht, The Netherlands, 1997. 5

Samuel Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York, NY, 1974. 60, 72, 79, 192

Jason Eisner. Expectation semirings: Flexible EM for finite-state transducers. In Gertjan van Noord, editor, *Proceedings of the ESSLLI Workshop on Finite-State*

Methods in NLP, Helsinki, Finland, August 2001. 136, 160, 161, 168, 184, 186, 188, 209

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, PA, July 2002. 160, 161, 163, 183, 184, 185, 188, 189, 190, 242

Gunnar Evermann. Minimum word error rate decoding. MPhil thesis, University of Cambridge, Churchill College, Cambridge, England, August 1999. 200, 202

Eugene Fink. A survey of sequential and systolic algorithms for the algebraic path problem. Technical Report CS-92-37, Dept. of Computer Science, University of Waterloo, Waterloo, ON, 1992. 141, 154

William M. Fisher. A statistical text-to-phone function using ngrams and rules. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 649–652, Phoenix, AZ, May 1999. 39, 65, 226

John G. Fletcher. A more general algorithm for computing closed semiring costs between vertices of a directed graph. *Communications of the ACM*, 23(6): 350–351, June 1980. 162, 172, 173

Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6): 345, June 1962. 172, 173

Ariadna Font Llitjós. Improving pronunciation accuracy of proper names with language origin classes. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, August 2001. 6

Eric Fosler-Lussier, Ingunn Amdal, and Hong-Kwang Jeff Kuo. On the road to improved lexical confusability metrics. In *Proceedings of the ISCA Tutorial and Research Workshop on Pronunciation Modeling and Lexicon Adaptation*, Estes Park, CO, September 2002. 250

Lucian Galescu and James F. Allen. Bi-directional conversion between graphemes and phonemes using a joint n-gram model. In *Proceedings of the Fourth ISCA Tutorial and Research Workshop on Speech Synthesis*, Blair Atholl, Scotland, August 2001. 8, 55

G. Gander, E. Krane, G. Pillock, and I. Sacks. *Generalized Node Vortex Grammar*. Fons Niger, Oxford, England, 1985. 256

Pedro García and Enrique Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 12(9):920–925, September 1990. 70, 71

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979. 88, 90, 97

Dafydd Gibbon. Finite state processing of tone systems. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, pages 291–297, Copenhagen, Denmark, 1987. 71

Dafydd Gibbon. Finite state prosodic analysis of African corpus resources. In *Proceedings of the 7th European Conference on Speech Communication and Technology (Eurospeech 2001)*, pages 83–86, Aalborg, Denmark, September 2001. 71

Dafydd Gibbon, Roger Moore, and Richard Winski, editors. *Handbook of Standards and Resources for Spoken Language Systems*. Mouton de Gruyter, Berlin, Germany, 1997. 16, 18, 19

Dan Gildea and Dan Jurafsky. Automatic induction of finite state transducers for simple phonological rules. Technical Report TR-94-052, International Computer Science Institute, Berkeley, CA, October 1994. Also presented at ACL 33 (1995). 61, 64

Daniel Gildea and Daniel Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22(4):497–530, 1996. 61, 64

Michel Gilloux. Automatic learning of word transducers from examples. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 107–112, Berlin, Germany, 1991. 128, 160

Kazimierz Głazek. *A Guide to the Literature on Semirings and their Applications in Mathematics and Information Sciences*. Kluwer, Dordrecht, The Netherlands, 2002. 119

Jonathan S. Golan. *The Theory of Semirings with Applications in Mathematics and Theoretical Computer Science*. Number 54 in Pitman Monographs and Surveys in Pure and Applied Mathematics. Longman and Wiley, Harlow, England and New York, NY, 1992. 119, 186, 195

Jonathan S. Golan. *Semirings and their Applications*. Kluwer, Dordrecht, The Netherlands, 1999. 119

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5): 447–474, 1967. 60, 85

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, December 1999. 119, 120, 152, 161, 162

Joshua T. Goodman. *Parsing Inside-Out*. PhD thesis, Harvard University, Cambridge, MA, May 1998. 119, 152, 158

H. J. Hamilton and J. Zhang. Learning pronunciation rules for English graphemes using the version space algorithm. In *Proceedings of the Seventh Florida Artificial Intelligence Research Symposium (FLAIRS-94)*, pages 76–80, Pensacola, FL, May 1994. 66

Lynette Hirschman and Henry S. Thompson. Overview of evaluation in speech and natural language processing. In Cole et al. [1997], chapter 13.1. 16

J. Hochberg, S. M. Mniszewski, T. Calleja, and G. J. Papcun. A default hierarchy for pronouncing English. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(9):957–964, September 1991. 9, 65

Juraj Hromkovič. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, Berlin, Germany, 1st edition, 2001. 56, 221

C. B. Huang, M. A. Son-Bell, and D. M. Baggett. Generation of pronunciation from orthographies using transformation-based error-driven learning. In *Proceedings of the Third International Conference on Spoken Language Processing*, pages 411–414, Yokohama, Japan, September 1994. 9, 66

Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976. 76, 84

International Phonetic Association. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, Cambridge, England, 1999. 5, 21, 25

Martin Jansche. A two-level take on Tianjin tone. In *Proceedings of the Third ESSLLI Student Session*, pages 162–174, Saarbrücken, Germany, August 1998. 62, 71

Martin Jansche. Re-engineering letter-to-sound rules. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 111–117, Pittsburgh, PA, 2001. 29, 66

Martin Jansche. Learning local transductions is hard. In *Proceedings of Mathematics of Language 8*, pages 81–92, Bloomington, IN, June 2003. 60

Martin Jansche and Steven P. Abney. Information extraction from voicemail transcripts. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 320–327, Philadelphia, PA, July 2002. 3, 248

Frederick Jelinek. *Statistical Methods for Speech Recognition*. Language, Speech and Communication. MIT Press, Cambridge, MA, 1997. 116, 117, 128, 151, 154

Li Jiang, Hsiao-Wuen Hon, and Xuedong Huang. Improvements on a trainable letter-to-sound converter. In *Proceedings of the 5th European Conference on Speech Communication and Technology (Eurospeech '97)*, pages 605–608, Rhodes, Greece, September 1997. 65, 108

Daniel Jurafsky, James H. Martin, Andrew Kehler, Keith Vander Linden, and Nigel Ward. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, 2000. 122

Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 341–352, Philadelphia, PA, 1992. 103

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, 1994. Second printing, 1997. 83

Sanjeev Khanna, Madhu Sudan, and Luca Trevisan. Constraint satisfaction: The approximability of minimization problems. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 282–296, Ulm, Germany, June 1997a. 101, 102

Sanjeev Khanna, Madhu Sudan, and David P. Williamson. A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 11–20, El Paso, TX, 1997b. 97

Sam M. Kim and Robert McNaughton. Computing the order of a locally testable automaton. *SIAM Journal on Computing*, 23(6):1193–1215, 1994. 74

Paul Kingsbury, Stephanie Strassel, Cynthia McLemore, and Robert MacIntyre. CALLHOME American English lexicon (PRONLEX). CD-ROM, catalog number LDC97L20, Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA, 1997. 19

Dennis H. Klatt. Review of text to speech conversion for English. *Journal of the Acoustical Society of America*, 82(3):737–793, 1987. xii, 5, 6, 7

Philip N. Klein, Serge A. Plotkin, Satish Rao, and Éva Tardos. Approximation algorithms for Steiner and directed multicuts. *Journal of Algorithms*, 22(2): 205–220, 1997. 101

Kevin Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, December 1999. 84

Reinhard B. Köhler and Burghard B. Rieger, editors. *Contributions to Quantitative Linguistics: Proceedings of the First International Conference on Quantitative Linguistics, QUALICO, Trier, 1991*. Kluwer, Dordrecht, The Netherlands, 1993. 263

András Kornai. Natural language and the Chomsky hierarchy. In *Proceedings of the Second Conference of the European Chapter of the Association for Computational Linguistics*, pages 1–7, Geneva, Switzerland, 1985. 71

András Kornai. Zipf's law outside the middle range. In *Proceedings of the Sixth Meeting on Mathematics of Language*, pages 347–356, Orlando, FL, 1999. 1

András Kornai. How many words are there? *Glottometrics*, 4:61–86, 2002. 1

Joseph B. Kruskal. An overview of sequence comparison. In Sankoff and Kruskal [1983], pages 1–44. Reissued by CSLI Publications, Stanford, CA, 1999. 38, 117, 121, 249

Werner Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In Rozenberg and Salomaa [1997], pages 609–677. 119, 162

Éric Laporte. Rational transductions for phonetic conversion and phonology. In Roche and Schabes [1997a], pages 407–430. 71

J. M. Lucassen and R. L. Mercer. An information theoretic approach to the automatic determination of phonemic baseforms. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 42.5.1–42.5.4, 1984. 9, 37, 65, 71, 75

R. W. P. Luk and R. I. Damper. Stochastic phonographic transduction for English. *Computer Speech and Language*, 10(2):133–153, April 1996. 39

Robert W. P. Luk and Robert I. Damper. Computational complexity for a fast Viterbi decoding algorithm for stochastic letter-phoneme transduction. *IEEE Transactions on Speech and Audio Processing*, 6(3):217–225, May 1998. 151

Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14(4):373–400, October 2000. 202

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999. 116

Yannick Marchand and Robert I. Damper. A multistrategy approach to improving pronunciation by analogy. *Computational Linguistics*, 26(2):195–219, 2000. 233

Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley Series in Probability and Statistics. Wiley, New York, NY, 1997. 127

Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, Cambridge, MA, 1972. 68, 69, 70

Thomas P. Minka. Empirical risk minimization is an incomplete inductive principle. http://www.stat.cmu.edu/~minka/papers/erm.html, August 2000. 85

Wolfgang Minker. Grapheme-to-phoneme conversion: An approach based on hidden Markov models. Notes et Documents LIMSI 96-04, Laboratoire d'Informatique pour le Mécanique et les Sciences de l'Ingénieur (LIMISI-CNRS), Orsay Cedex, France, January 1996. 10, 45, 66, 78, 111

Roger Mitton. A description of a computer-usable dictionary file based on the Oxford Advanced Learner's Dictionary. Electronic document, Oxford Text Archive, number 0710-2, June 1992. 19

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997. 119, 120, 121, 152, 159, 164, 192, 199, 209

Mehryar Mohri. General algebraic frameworks and algorithms for shortest-distance problems. Technical Memorandum 981210-10TM, AT&T Labs, Florham Park, NJ, June 1998. 62 pages. 171, 194

Mehryar Mohri. Generic epsilon-removal algorithm for weighted automata. In Andrei Păun and Sheng Yu, editors, *Implementation and Application of Automata: 5th International Conference, CIAA 2000, London, Ontario, Canada, July 24–25, 2000: Revised Papers*, number 2088 in Lecure Notes in Computer Science, pages 230–242. Springer, Berlin, Germany, 2001. 192, 193, 195, 197

Mehryar Mohri. Edit-distance of weighted automata. In Jean-Marc Champarnaud and Denis Maurel, editors, *7th International Conference CIAA 2000, Tours, France*. Springer, 2002a. 114, 121, 203, 250

Mehryar Mohri. Generic ϵ-removal and input ϵ-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002b. 192, 197, 209

Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002c. 154, 171, 194, 208

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. In *Proceedings of the ECAI'96 Workshop on Extended Finite State Models of Language*, pages 46–50, Budapest, Hungary, August 1996. 146, 159, 160, 161, 164, 166

Mehryar Mohri, Fernando Pereira, and Michael Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1): 17–32, January 2000. 66, 119, 166, 178

Mehryar Mohri and Michael Riley. A weight pushing algorithm for large vocabulary speech recognition. In *Proceedings of the 7th European Conference on Speech Communication and Technology (Eurospeech 2001)*, pages 1603–1606, Aalborg, Denmark, September 2001. 183

Mehryar Mohri and Michael Riley. An efficient algorithm for the n-best-strings problem. In *Proceedings of the Seventh International Conference on Spoken Language Processing*, Denver, CO, September 2002. 152, 202

Edward F. Moore, editor. *Sequential Machines: Selected Papers*. Addison-Wesley, Reading, MA, 1964. 263

H. C. Nusbaum, D. B. Pisoni, and C. K. Davis. Sizing up the Hoosier mental lexicon: Measuring the familiarity of 20,000 words. Research on Speech Perception Progress Report 10, Indiana University, Bloomington, IN, 1984. 20

Arlindo L. Oliveira, editor. *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 2000, Proceedings*, number 1891 in Lecure Notes in Artificial Intelligence, Berlin, Germany, 2000. Springer. 253

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 15(5):448–458, May 1993. 60

Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994. 96, 97

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY, 1998. Originally published by Prentice Hall, Englewood Cliffs, NJ, 1982. 87, 221

S. H. Parfitt and R. A. Sharman. A bi-directional model of English pronunciation. In *Proceedings of the 2nd European Conference on Speech Communication and Technology (Eurospeech '91)*, pages 801–804, Genova, Italy, September 1991. 128, 160

Fernando C. N. Pereira and Michael Riley. Speech recognition by composition of weighted finite automata. In Roche and Schabes [1997a]. 119, 146, 159, 164, 166

Louis C. W. Pols. Speech synthesis evaluation. In Cole et al. [1997], chapter 13.7, pages 429–430. 16, 18

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 2nd edition, 1992. 141

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. 66, 222

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993. 214

Michael O. Rabin. Probabilistic automata. In Moore [1964], pages 98–114. 161

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989. 111, 115, 151, 159

Mazin Rahim and Chin-Hui Lee. String-based minimum verification error (SB-MVE) training for speech recognition. *Computer Speech and Language*, 11(2): 147–160, April 1997. 200

P. Rentzepopoulos and G. Kokkinakis. Phoneme to grapheme conversion using HMM. In *Proceedings of the 2nd European Conference on Speech Communication and Technology (Eurospeech '91)*, Genova, Italy, September 1991. 8

P. A. Rentzepopoulos, A. E. Tsopanoglou, and G. K. Kokkinakis. A statistical approach for phoneme-to-grapheme conversion. In Köhler and Rieger [1993], pages 319–328. 10, 66, 77, 111

Panagiotis A. Rentzepopoulos and George K. Kokkinakis. Efficient multi-lingual phoneme-to-grapheme conversion based on HMM. *Computational Linguistics*, 22(3):351–376, 1996. 66, 111

Michael D. Riley. A statistical model for generating pronunciation networks. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 737–740, 1991. 29, 65

Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. Technical Report CS-TR-532-96, Dept. of Computer Science, Princeton University, Princeton, NJ, October 1996. 105, 113

Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20(5):522–532, May 1998. 10, 105, 111, 113, 117, 122, 123, 124, 128, 132, 135, 136, 137, 151, 157, 158, 160, 178, 188, 203, 204, 208, 209, 247

Emmanuel Roche and Yves Schabes, editors. *Finite-State Language Processing*. Language, Speech and Communication. MIT Press, Cambridge, MA, 1997a. 260, 263, 264

Emmanuel Roche and Yves Schabes. Introduction. In *Finite-State Language Processing* Roche and Schabes [1997a], pages 1–66. 62, 192

Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 1. Springer, Berlin, Germany, 1997. 260

Robert C. Russell. Improvements in indexes. US Patent 1,261,167, April 1918. 4

Arto Salomaa. *Jewels of Formal Language Theory*. Computer Science Press, Rockville, MD, 1981. 72

David Sankoff and Joseph Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983. Reissued by CSLI Publications, Stanford, CA, 1999. 260

Lawrence K. Saul and Mazin G. Rahim. Maximum likelihood and minimum classification error factor analysis for automatic speech recognition. *IEEE Transactions on Speech and Audio Processing*, 8(2):115–125, March 2000. 200

Terrence J. Sejnowski. The NetTalk corpus: Phonetic transcription of 20,008 English words. Electronic document, Cognitive Science Center, Johns Hopkins University, Baltimore, MD, 1988. Available as part of Blake and Merz [1998]. 20, 24, 25, 29, 36, 42, 73, 78, 104, 217

Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1(1):145–168, 1987. 9, 36, 37, 54, 65, 73, 226

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. 109

Chilin Shih and Richard Sproat. Issues in text-to-speech conversion for Mandarin. *Computational Linguistics and Chinese Language Processing*, 1(1): 37–86, August 1996. 6

N. J. A. Sloane. *The On-Line Encyclopedia of Integer Sequences*. 1996–2003. http://www.research.att.com/~njas/sequences/. 116, 123

Richard Sproat. Text interpretation for TtS synthesis. In Cole et al. [1997], chapter 5.3, pages 175–182. 5

Richard Sproat, editor. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer, Dordrecht, The Netherlands, 1998. 265

Richard Sproat. *A Computational Theory of Writing Systems*. Cambridge University Press, Cambridge, England, 2000. 2, 3, 45

Richard Sproat, Alan W. Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. Normalization of non-standard words. *Computer Speech and Language*, 15(3):287–333, July 2001. 23

Richard Sproat, Bernd Möbius, Kazuaki Maeda, and Evelyne Tzoukermann. Multilingual text analysis. In Sproat [1998], chapter 3, pages 31–87. 5

Richard Sproat and Michael Riley. Compilation of weighted finite-state transducers from decision trees. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 215–222, Santa Cruz, CA, June 1996. 66

Richared Sproat. *Pmtools*: A pronunciation modeling toolkit. In *Proceedings of the Fourth ISCA Tutorial and Research Workshop on Speech Synthesis*, Blair Atholl, Scotland, August 2001. 66, 75, 77, 78, 104, 237

Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986. 9, 25, 37, 65, 73, 244

Craig W. Stanfill. Memory-based reasoning applied to English pronunciation. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 577–581, Seattle, WA, July 1987. 37, 65

Andreas Stolcke, Yochai König, and Mitchel Weintraub. Explicit word error minimization in n-best list rescoring. In *Proceedings of the 5th European Conference on Speech Communication and Technology (Eurospeech '97)*, Rhodes, Greece, September 1997. 202

Andreas Stolcke and Stephen Omohundro. Hidden Markov model induction by Bayesian model merging. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18, San Mateo, CA, 1993. Morgan Kaufmann. 111, 247

I. Torres and A. Varona. *k*-TSS language models in speech recognition systems. *Computer Speech and Language*, 15(2):127–149, April 2001. 71

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, November 1984. 83

Antal van den Bosch and Walter Daelemans. Data-oriented methods for grapheme-to-phoneme conversion. In *Proceedings of the Sixth Conference of European Chapter of the Association for Computational Linguistics*, pages 45–53, Utrecht, The Netherlands, 1993. 37, 226

Antal P. J. van den Bosch. *Learning to Pronounce Written Words: A Study in Inductive Language Learning*. PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, December 1997. 6, 8, 28

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974. 38, 76, 96, 117, 123, 157, 249

Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1): 11–12, January 1962. 172, 173

Robert L. Weide. The Carnegie Mellon pronouncing dictionary version 0.6. Electronic document, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1998. `ftp://ftp.cs.cmu.edu/project/fgdata/dict/`. 19, 21

S. Winograd. Input-error-limiting automata. *Journal of the ACM*, 11(3):338–351, March 1964. 72

Takashi Yokomori and Satoshi Kobayashi. Learning local languages and their application to DNA sequence analysis. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20(10):1067–1079, October 1998. 70

F. Yvon, P. Boula de Mareüil, C. d'Allesandro, V. Aubergé, M. Bagein, G. Bailly, F. Béchet, S. Foukia, J.-F. Goldman, E. Keller, D. O'Shaughnessy, V. Pagel, F. Sannier, J. Véronis, and B. Zellner. Objective evaluation of grapheme to phoneme conversion for text-to-speech synthesis in French. *Computer Speech and Language*, 12(4):393–410, October 1998. 18, 34

Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972. 70

INDEX