Training of Hidden Markov models as an instance of the expectation maximization algorithm

Bachelorarbeit
zur Erlangung des Hochschulgrades
Bachelor of Science

vorgelegt von
Dipl.-Phys. Stefan Majewsky
geboren am 06.11.1989 in Schwerin

Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Grundlagen der Programmierung

Betreuer: Dipl.-Inf. Kilian Gebhardt
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Dr. h.c./Univ. Szeged Heiko Vogler

Source code at: `https://github.com/majewsky/bachelor-thesis/`

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Stefan Majewsky
Dresden, 13. Juli 2017

Aufgabenstellung für die Bachelorarbeit

„Training of hidden Markov models as an instance of the expectation maximization algorithm“

Technische Universität Dresden
Fakultät Informatik

Student:	Stefan Majewsky
Geburtsdatum:	6. November 1989
Matrikelnummer:	3319755
Studiengang:	Bachelor Informatik
Immatrikulationsjahr:	2006
Studienleistung:	Bachelorarbeit
Beginn am:	0. Mai 2017
Einzureichen am:	0. August 2017
Verantw. Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Dipl.-Inf. Kilian Gebhardt

Im Forschungsfeld der *Natürlichen Sprachverarbeitung* (NLP) werden Probleme häufig in folgenden drei Schritten bearbeitet [Lop08]:

1. Modellierung des Problems
2. Training der Modellparameter
3. Testen des Modells

Für das Training von strukturierten probabilistischen Modellen finden häufig Ausprägungen des *expectation maximization* (EM) Algorithmus [DLR77; Pre05] Verwendung. Der EM Algorithmus berechnet, ausgehend von einem Korpus über unvollständigen Daten und einer initialen Wahrscheinlichkeitsverteilung über vollständigen Daten, eine Folge von Wahrscheinlichkeitsverteilungen. Dazu wird wiederholt das Korpus über unvollständigen Daten zu einem Korpus über vollständigen Daten erweitert (expectation) und

anschließend das *maximum likelihood estimate* über dem vervollständigten Korpus berechnet (maximization). Die Folge der Wahrscheinlichkeitsverteilungen konvergiert zu einem lokalen Maximum der *likelihood*.

Beispiele für die Verwendungen des EM Algorithmus in der NLP sind u.a. das Training des Wörterbuchs im IBM Modell 1 [Bro+93], das unüberwachte Training einer probabilistischen kontextfreien Grammatik [LY90] und das Training eines extended top-down tree transducer [GK04]. Auch wenn es offensichtliche Ähnlichkeiten zwischen dem EM Algorithmus von [DLR77] und diesen Trainingsalgorithmen gibt, wurde selten ein formaler Beweis erbracht, dass diese Algorithmen tatsächlich EM Algorithmen sind und folgerichtig die gleichen Konvergenzeigenschaften gelten.

Büchse, Stüber und Vogler [BSV15] haben ein generisches Framework für EM Algorithmen entwickelt und formal gezeigt, dass es sich bei ihren prototypischen Algorithmen *corpus-based EM algorithm*, *simple-counting EM algorithm* und *inside-outside EM algorithm* tatsächlich um EM Algorithmen handelt. Außerdem geben sie an, wie sich einige der vorgenannten Trainingsalgorithmen in ihr Framework einbetten lassen, und erbringen dadurch den formalen Beweis, dass diese EM Algorithmen sind.

Hidden Markov Modelle (HMM) werden in der NLP zur Modellierung von Sequenzen eingesetzt, u.a. als Sprachmodelle. Beim Training der Übergangswahrscheinlichkeiten zwischen den Zuständen findet der Baum-Welch Algorithmus [Bau+70] Verwendung, welcher üblicherweise zu den EM Algorithmen gezählt wird.

Aufgabe In seiner Bachelorarbeit soll Stefan Majewsky zeigen, dass sich der Baum-Welch Algorithmus zum Training von HMM als ein inside-outside EM Algorithmus aus dem Framework von [BSV15] darstellen lässt. Dazu sollen folgende Teilaufgaben bearbeitet werden.

1. Der Baum-Welch Algorithmus soll geschlossen dargestellt werden [vgl. auch Nel13], wobei ein hohes Maß an Übereinstimmung mit der Primärliteratur [Bau+70] anzustreben ist.

2. Anschließend sollen die Mengen (z.B. X, Y, Z, A, B, C) und Abbildungen (z.B. q, π_1, π_2, K, H), die im Kontext eines inside-outside EM Algorithmus relevant sind, passend instanziiert werden. Dabei soll formal gezeigt werden, dass die Anforderungen, welche an letztere Objekte im Framework von [BSV15] gestellt werden, erfüllt sind.

3. Es soll identifiziert werden, welche Teile des Baum-Welch Algorithmus welchen Hilfsgrößen (z.B. $\chi, \alpha, \beta, (\!|\mu|\!)_{\mathrm{io}}$) eines inside-outside EM Algorithmus entsprechen.

4. Zuletzt soll bewiesen werden, dass der Baum-Welch Algorithmus und die konstruierte Instanz eines inside-outside EM Algorithmus das selbe Ergebnis liefern.

Die Arbeit muss den üblichen Standards wie folgt genügen. Die Arbeit muss in sich abgeschlossen sein und alle nötigen Definitionen und Referenzen enthalten. Die Urheberschaft von Inhalten – auch die eigene – muss klar erkennbar sein. Fremde

Inhalte, z.B. Algorithmen, Konstruktionen, Definitionen und Ideen, müssen durch genaue Verweise auf die entsprechende Literatur kenntlich gemacht werden. Lange wörtliche Zitate sollen vermieden werden. Gegebenenfalls muss erläutert werden, inwieweit und zu welchem Zweck fremde Inhalte modifiziert wurden. Die Struktur der Arbeit muss klar erkenntlich sein, und der Leser soll gut durch die Arbeit geführt werden. Die Darstellung aller Begriffe und Verfahren soll mathematisch formal fundiert sein. Für jeden wichtigen Begriff sollen Erläuterungen und Beispiele angegeben werden, ebenso für die Abläufe der beschriebenen Verfahren sowie Konstruktionen. Wo es angemessen ist, sollen Illustrationen die Darstellung vervollständigen. Bei Diagrammen, die Phänomene von Experimenten beschreiben, muss deutlich erläutert werden, welche Werte auf den einzelnen Achsen aufgetragen sind, und beschrieben werden, welche Abhängigkeit unter den Werten der verschiedenen Achsen dargestellt ist. Schließlich sollen alle Lemmata und Sätze möglichst lückenlos bewiesen werden. Die Beweise sollen leicht nachvollziehbar dokumentiert sein.

Dresden, 11. April 2017

Unterschrift von Heiko Vogler

Unterschrift von Stefan Majewsky

Literatur

[Bau+70] L.E. Baum, T. Petrie, G. Soules und N. Weiss. „A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains“. In: *The annals of mathematical statistics* (1970), S. 164–171.

[Bro+93] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra und Robert L. Mercer. „The mathematics of statistical machine translation: parameter estimation“. In: *Computational Linguistics* 19 (2 Juni 1993), S. 263–311. ISSN: 0891-2017.

[BSV15] Matthias Büchse, Torsten Stüber und Heiko Vogler. „A Generic Inside-Outside EM Algorithm“. Unpublished Manuscript. 2015.

[DLR77] Arthur P. Dempster, Nan M. Laird und Donald B. Rubin. „Maximum Likelihood from Incomplete Data via the EM Algorithm“. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), S. 1–38. ISSN: 00359246.

[GK04] Jonathan Graehl und Kevin Knight. „Training Tree Transducers“. In: *HLT-NAACL 2004: Main Proceedings*. Hrsg. von Daniel Marcu Susan Dumais und Salim Roukos. Boston, Massachusetts, USA: Association for Computational Linguistics, 2004, S. 105–112.

[Lop08] A. Lopez. „Statistical Machine Translation“. In: *ACM Computing Surveys* 40(3) (2008), 8:1–8:9.

[LY90] Karim Lari und Steve J. Young. „The estimation of stochastic context-free grammars using the Inside-Outside algorithm“. In: *Computer Speech and Language* 4 (1990), S. 35–56.

[Nel13] Tobias Nelsen. „Hidden Markov models“. B.S. Thesis. Technische Universität Dresden, 2013.

[Pre05] Detlef Prescher. „A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilisitic Context-Free Grammars“. Tutorial. 2005.

Contents

1 Introduction **1**

1.1 N-gram models . . . 2

1.2 Hidden Markov model . . . 3

2 Expectation-maximization algorithms **5**

2.1 Preliminaries . . . 5

2.2 Algorithmic skeleton . . . 6

2.3 Corpus-based step mapping . . . 8

2.4 Simple counting step mapping . . . 9

2.5 Regular tree grammars . . . 11

2.6 Inside-outside step mapping . . . 17

2.7 Review . . . 19

3 The Hidden Markov model **21**

3.1 Forward and backward algorithms . . . 22

3.2 The Baum-Welch algorithm . . . 24

3.3 Deriving the Baum-Welch algorithm . . . 26

3.3.1 Model parameter and countable events . . . 27

3.3.2 Tree-shaped hidden information . . . 28

3.3.3 Complete-data corpus . . . 32

3.3.4 Inside weights . . . 33

3.3.5 Outside weights . . . 34

3.3.6 Complete-data corpus (cont.) . . . 36

3.3.7 Step mapping . . . 38

3.4 Review . . . 39

Appendix **41**

A Elided proofs from Chapter 3 . . . 41

A.1 Proof of Lemma 3.8 . . . 42

A.2 Proof of Lemma 3.9 . . . 43

B Formulary for Chapter 3 . . . 47

Bibliography **49**

1 Introduction

In natural language processing, it is frequently necessary to judge the correctness of sentences generated by some algorithm. For example, a speech-to-text translator that only recognizes individual words might produce the following two candidate sentences for the same input audio:

He ate soup. He aid soup.

Both sentences sound the same, but the second candidate sentence should be discarded because it is syntactically wrong. As another example, a translation algorithm that translates from another language to English might generate the following two candidate sentences for some input:

This is a small red ball. This is a red small ball.

Both sentences are syntactically correct, but the first sentence should be preferred because it conforms to the customary rules for adjective ordering in the English language. Finally, consider a spelling correction process that is applied to the following sentence:

The design **an** construction of the system will take more than a year. [Kuk92]

Possible replacements for the syntactically wrong word "an" include "a" and "and". In all these situations, a *language model* can be employed to choose the best result from a set of candidates. [S$^+$02, YEG$^+$05] A language model assigns a probability $p_\omega(v)$ to each sentence $v \in V^*$ (with words from the set V), such that correct sentences receive a higher probability than incorrect sentences, e. g.,

$$1 > p_\omega(\text{He ate soup.}) > p_\omega(\text{He aid soup.}) > p_\omega(\text{He He He soup.}) > 0.$$

The probability distribution p_ω is determined by a *model parameter* $\omega \in \Omega$. The language model defines Ω and describes how to obtain p_ω for any model parameter ω. The model parameter is chosen by a suitable *training algorithm* using a training corpus c containing known good sentences, such that ω maximizes the likelihood of this corpus, i. e.,

$$\prod_{v \in c} p_\omega(v).$$

Many training algorithms for language models are instances of the *expectation-maximization (EM) algorithm* [DLR77]. Chapter 2 will introduce a general framework for EM algorithms that first appeared in [BSV15].

1.1 N-gram models

One of the simplest language models is the *bigram model*. [JM09] It interprets the generation of a sentence as a stochastic process, wherein each word is chosen with a probability conditional on the word that appeared before it: (see Figure 1.1)

$$p(\text{He ate soup.}) = b(\text{He}|\#) \cdot b(\text{ate}|\text{He}) \cdot b(\text{soup}|\text{ate}) \cdot b(\#|\text{soup}).$$

The model parameter is b, a conditional probability distribution of $V \cup \{\#\}$ given $V \cup \{\#\}$, where # is a placeholder word that stands in for the start and end of the sentence.
The model parameter can be chosen by a very simple training algorithm [JM09, pp. 123]: All bigrams, i. e., pairs of words occurring one after the other, are counted across the corpus. This includes bigrams containing the sentence delimiter word #. Then, the counts are normalized into a conditional probability distribution by setting

$$b(w|w') := \frac{\text{count}(w'w)}{\sum_{w'' \in V \cup \{\#\}} \text{count}(w'w'')} \quad \forall w, w' \in V \cup \{\#\}\,.$$

The bigram model can be generalized to the *N-gram model*, wherein the next word is chosen with a probability conditional on the $n-1$ words before it. Training then counts n-grams, i. e., sequences of n words, hence the name *N-gram model*. The bigram model is recovered for $n = 2$.
Besides the already mentioned applications where language models augment translators and autocorrect systems, N-gram models are useful in *augmentive communication*: Virtual keyboards on smartphones and tablets predict the next word by looking at the previous n words, thus improving typing speed and accuracy. [HH04] A similar assistive word prediction system is part of speech synthesis programs used by disabled persons such as the physicist Stephen Hawking. [NLH98]

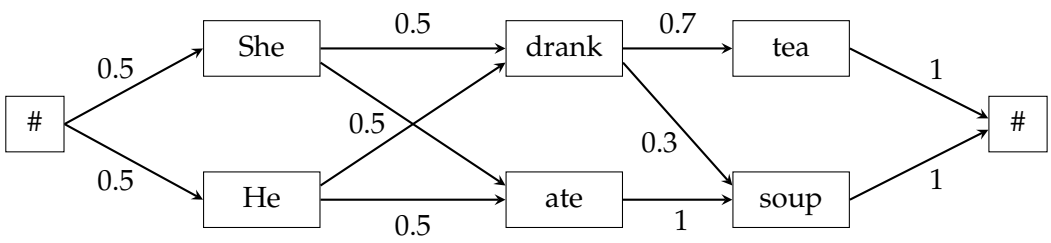


Figure 1.1: Graphic representation of an example instance of the bigram model. The nodes are words from the vocabulary $V \cup \{\#\}$. The start/end marker # is shown twice for readability's sake. Arrows $v \to v'$ represent a nonzero bigram probability $b(v'|v)$.

1.2 Hidden Markov model

A further generalization of N-gram models leads to the *Hidden Markov model*. In this model, the emission probability of a word depends not on the previous words, but on the progression of a state machine that cannot be directly deduced from the emitted words. Every time a word needs to be emitted, the state machine progresses to a new state according to a transition probability distribution t dependent on the previous state, and the next word is predicted by an emission probability distribution e dependent on the new state. The symbol # is used as the initial and final state of the state machine.

For example, if the state sequence that generated the sentence "He ate soup" was "#-noun-verb-noun-#", then the probability of that sentence would be

$$\begin{aligned} p(v|q) = t(\text{noun}|\#) \cdot e(\text{he}|\text{noun}) \cdot t(\text{verb}|\text{noun}) \cdot e(\text{ate}|\text{verb}) \\ \cdot t(\text{noun}|\text{verb}) \cdot e(\text{soup}|\text{noun}) \cdot t(\#|\text{noun}). \end{aligned}$$

Since the state sequence is typically not known, a sum over all possible state sequences q must be computed to obtain $p(v)$ for a sentence $v \in V^*$.

N-gram models can be interpreted a special case of the Hidden Markov model by encoding the current N-gram in the state ($Q = V^n$) and defining the transmission and emission probability in terms of the N-gram probability. For example, for bigrams,

$$t(v_1v_2|v_1'v_2') := \begin{cases} b(v_2|v_1) & \text{if } v_1 = v_2', \\ 0 & \text{otherwise,} \end{cases} \qquad e(v|v_1v_2) := \begin{cases} 1 & \text{if } v = v_2, \\ 0 & \text{otherwise.} \end{cases}$$

Training for the Hidden Markov model is not as straightforward as for the bigram model, since the training corpus typically only contains the observed sentences, not the state sequences that

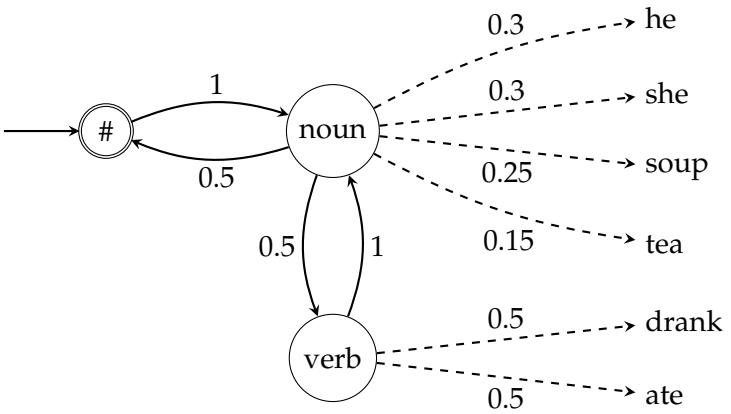


Figure 1.2: Graphic representation of an example instance of the Hidden Markov model, with hidden states "noun" and "verb"; adapted from [Nel13]. Arrows $q \to q'$ and dashed arrows $q \dashrightarrow v$ represent nonzero transition and emission probabilities $t(q'|q)$ and $e(v|q)$, respectively.

produced them. The standard training algorithm for Hidden Markov models is the *Baum-Welch algorithm*. [BPSW70, Bau72]
Chapter 3 will define Hidden Markov models more formally, and discuss algorithms that act on them. Finally, we will show that the Baum-Welch algorithm is an instance of the general EM algorithm laid out in Chapter 2, from which follows that it exhibits the same beneficial convergence properties.

2 Expectation-maximization algorithms

The generic framework for expectation-maximization (EM) algorithms introduced by [BSV15] applies to several kinds of probability models, e. g., language models, translation models, and parsing models. This chapter will introduce terminology and notation from [BSV15] as far as is necessary to apply this framework to language models and, specifically, to the Hidden Markov model in Chapter 3.

2.1 Preliminaries

The set $\{0, 1, 2, \ldots\}$ of non-negative integers and the set of non-negative reals shall be denoted by $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$, respectively. We assume that

$$0^0 := 1, \qquad \log 0 := -\infty, \qquad 0 \cdot (-\infty) = \log 0^0 = 0.$$

Definition 2.1. Given a countable set X, a mapping $c\colon X \to \mathbb{R}_{\geq 0}$ is called an *X-corpus*. Its *size* and *support* are defined as

$$|c| := \sum_{x \in X} c(x) \qquad \text{and} \qquad \operatorname{supp}(c) := \{x \in X\colon c(x) \neq 0\},$$

respectively. The corpus c is called *empty* if $|c| = 0$, *finite* if $|c| < \infty$, and *probability distribution of* X if $|c| = 1$. The set of all probability distributions of X is denoted by $\mathcal{M}(X)$. □

When used as input for a language model's training algorithm, X is the set of all sentences consisting of words from the language in question, and $c(x)$ describes how often a sentence $x \in X$ occurs in the corpus c. Probability distributions can be derived from corpora as follows.

Definition 2.2. Given a non-empty and finite X-corpus c, the *empirical probability distribution* $\tilde{c}$ is defined as

$$\tilde{c}(x) := \frac{c(x)}{|c|}.$$ □

This expression is not well-defined for $|c| = 0$ or $|c| = \infty$, hence the requirement for c to be non-empty and finite.

Definition 2.3. Given an X-corpus c and $p \in \mathcal{M}(X)$, the *likelihood of c under p* is

$$p(c) := \prod_{x \in X} p(x)^{c(x)}.$$ □

The likelihood describes the probability of observing the sentences from the corpus c when sentences occur with the probability distribution described by p. A training algorithm will take c as an input, and seek to find an admissible p such that $p(c)$ is maximized. If any p is admissible, then for non-empty and finite c, the optimal choice is $p = \tilde{c}$ because of the following lemma.

Lemma 2.4. Let c be a non-empty and finite X-corpus. Then $\tilde{c}(c) \geq p(c)$ for every $p \in \mathcal{M}(X)$.

Proof. It suffices to show that $\log \tilde{c}(c) \geq \log p(c)$, since log is monotone. Using Gibbs' inequality,

$$\begin{aligned}\log \tilde{c}(c) &= \sum_{x \in X} c(x) \cdot \log \tilde{c}(x) = |c| \cdot \sum_{x \in X} \tilde{c}(x) \cdot \log \tilde{c}(x) \\ &\geq |c| \cdot \sum_{x \in X} \tilde{c}(x) \cdot \log p(x) = \sum_{x \in X} c(x) \cdot \log p(x) = p(c)\end{aligned}$$ □

However, using $p = \tilde{c}$ directly is not useful because this probability distribution is grossly overfitted: It will assign zero probability to any sentence not in the observed corpus. A useful language model thus limits the set of admissible p by describing the probability distribution in terms of *model parameters* $\omega \in \Omega$.

Definition 2.5. Given a set Ω, an Ω-*probability model for* X is a mapping $p \colon \Omega \to \mathcal{M}(X)$. □

Instead of $p(\omega)$, we write p_ω. Training shall then find $\omega \in \Omega$ such that $p_\omega(c)$ is maximized.

Definition 2.6. Let $f \colon A \to B$ be a mapping from sets A to B. The *argmax* of f is the set

$$\operatorname*{argmax}_{a \in A} f(a) := \left\{ a \in A | \forall a' \in A : f(a) \geq f(a') \right\}. \qquad \square$$

Definition 2.7. Given an Ω-probability model p for X, the *maximum likelihood estimator* for p is the mapping

$$\mathrm{mle}_p \colon \mathbb{R}_{\geq 0}^X \to \mathcal{P}(\Omega), \quad c \mapsto \operatorname*{argmax}_{\omega \in \Omega} p_\omega(c). \qquad \square$$

$\mathrm{mle}_p(c)$ is the set of all ω with maximal likelihood, but training only needs to find a single $\hat{\omega} \in \mathrm{mle}_p(c)$. Computing $\mathrm{mle}_p(c)$ by brute force is typically not tractable because the set Ω is usually countably infinite. However, there is one easily solvable special case.

Lemma 2.8. Let c be a nonempty and finite X-corpus, and p a Ω-probability model for X. If there exists $\hat{\omega} \in \Omega$ such that $p_{\hat{\omega}} = \tilde{c}$, then $\hat{\omega} \in \mathrm{mle}_p(c)$.

Proof. By Lemma 2.4, $p_{\hat{\omega}}(c) = \tilde{c}(c) \geq p_\omega(c)$ for every $\omega \in \Omega$, and thus $\hat{\omega} \in \mathrm{mle}_p(c)$. □

2.2 Algorithmic skeleton

When training a language model, most of the time, not all required data is present in the corpus. For example, a probabilistic context-free grammar assigns a probability distribution to its

Algorithm 2.1 Algorithmic skeleton for EM of language models according to [BSV15]

Input: X-corpus c
Ω-probability model p for $Y \times X$
some initial parameter $\omega_0 \in \Omega_0$ where $\Omega_0 := \{\omega \in \Omega : p_\omega(c) \neq 0\}$
Implicit: step mapping $\psi\colon \Omega_0 \to \mathcal{P}(\Omega)$
such that $\omega' \in \psi(\omega)$ implies $p_\omega(c) \leq p_{\omega'}(c)$ (*nondecreasing*)
Output: sequence $\omega_1, \omega_2, \ldots \in \Omega_0$
such that $p_{\omega_0}(c), p_{\omega_1}(c), p_{\omega_2}(c), \ldots$ nondecreasing

1: $i \leftarrow 0$
2: **while** not converged **do**
3: $\omega_{i+1} \leftarrow$ select a member of $\psi(\omega_i)$
4: output ω_{i+1}
5: $i \leftarrow i + 1$
6: **end while**

derivation rules. [LY90] When the training data consists of full parse trees (*supervised training*), the optimal probability distribution can be found by simply counting how many times each rule is used across all these parse trees, and then computing the empirical probability distribution for this corpus. Most of the times, however, the training data will consist only of sentences (*unsupervised training*). The information about how to parse the sentences is hidden.
A similar problem arises with the Hidden Markov model: When training data is not already annotated with state information, the information about which states correspond to which words from the training data remains hidden. *Expectation-maximization algorithms* can be used when parts of the training data are hidden in such a way. For the remainder, let

- X and Y be countable sets,
- Ω be a set,
- c be a finite X-corpus and
- p be a Ω-probability model for $Y \times X$.

The corpus c represents the set of training data. Each $x \in \operatorname{supp}(c)$ is an *observation*. To judge its probability under any p_ω, additional *hidden information* $y \in Y$ is required. For notational convenience, we define

$$p_\omega(c) := \prod_{x \in X} p_\omega(x)^{c(x)}, \qquad \text{where } p_\omega(x) := \sum_{y \in Y} p_\omega(y, x).$$

That is, even though p_ω is a probability distribution over $Y \times X$, we allow to take the likelihood of the X-corpus c under p by aggregating the probabilities for all hidden information y that lead to a certain observation x according to the law of total probability.
The basic pattern for expectation maximization is outlined in Algorithm 2.1. The algorithm starts with an initial ω_0 such that $p_{\omega_0}(c) \neq 0$. It then iteratively employs a step mapping to choose the next ω_i with a higher (or at least equal) likelihood than the one that came before.

Definition 2.9. Let $\Omega_0 := \{\omega \in \Omega : p_\omega(c) \neq 0\}$. A *step mapping* is a mapping $\psi\colon \Omega_0 \to \mathcal{P}(\Omega)$ which is nondecreasing in the following manner:

$$\forall \omega \in \Omega_0 : \forall \omega' \in \psi(\omega) : p_\omega(c) \leq p_{\omega'}(c). \qquad \square$$

The step mappings that we will consider consist of two steps:

1. *Expectation:* The training data c is converted into a *complete-data corpus*. Using ω_i from the previous iteration, the complete-data corpus estimates how hidden information contributes to the observations in the original corpus.

2. *Maximization:* A suitable maximum-likelihood estimator is applied to the complete-data corpus to choose ω_{i+1}.

This back and forth of using the current ω to enrich the training data and using the enriched data to find a better ω will converge towards a local maximum or saddle point of likelihood. [Wu83, Thm. 2] The iteration is usually aborted after the desired running time has been exceeded, or after the changes of $p_{\omega_i}(c)$ per iteration have become smaller than some threshold.
[BSV15] introduce three types of step mappings that build on each other, each one more specific than the one before it. Since the training of Hidden Markov models will be identified as an instance of the most specific step mapping, the inside-outside step mapping, the remainder of this chapter will introduce all these step mappings in order.

2.3 Corpus-based step mapping

The most general type of complete-data corpus can be obtained by distributing $c(x)$ among the hidden information y according to the probability distribution p_ω:

$$c\langle\omega, p\rangle(y, x) := \begin{cases} c(x) \cdot \frac{p_\omega(y,x)}{p_\omega(x)} & \text{if } p_\omega(x) \neq 0, \\ 0 & \text{if } p_\omega(x) = 0. \end{cases}$$

Recall that $p_\omega(x) = \sum_y p_\omega(y, x)$. The corpus $c\langle\omega, p\rangle$ has the correct structure for applying mle_p, yielding the *corpus-based step mapping*[1]

$$(\!|p|\!)_{\mathrm{cb}}\colon \Omega_0 \to \mathcal{P}(\Omega), \quad \omega \mapsto \mathrm{mle}_p\big(c\langle\omega, p\rangle\big) = \operatorname*{argmax}_{\omega'} p_{\omega'}\big(c\langle\omega, p\rangle\big).$$

It is usually intractable to evaluate this argmax due to the large size of Ω. To apply Lemma 2.8, $\hat{\omega}$ needs to be found such that $p_{\hat{\omega}} = \widetilde{c\langle\omega, p\rangle}$. This operation is typically hard, which is why a more specific step mapping is helpful.

[1] The proof that this step mapping is nondecreasing can be found in [BSV15, pp. 10].

2.4 Simple counting step mapping

The next such step mapping requires the language model to be described by a *counting information*. Before defining this term, some additional notation needs to be introduced.

Definition 2.10. Let A and B be sets. A mapping $p\colon B \to \mathcal{M}(A)$ is called *conditional probability distribution of A given B*. The set of all such mappings is denoted by $\mathcal{M}(A|B)$. □

To simplify notation, we define $p(a|b) := \big(p(b)\big)(a)$.

Definition 2.11. Let $C \subseteq A \times B$ be a set. Then

$$\mathcal{M}_C(A|B) := \big\{p \in \mathcal{M}(A|B) \,\big|\, \forall (a,b) \in A \times B\colon (a,b) \notin C \Rightarrow p(a|b) = 0\big\}$$

is the *set of all conditional probability distributions of A given B constrained to C*. □

Definition 2.12. Let c be an $A \times B$-corpus c. For any $a \in A$ and $b \in B$, we define the A-corpus c_b by $c_b(a) := c(a,b)$, and the *empirical conditional probability distribution* $\tilde{c} \in \mathcal{M}(A|B)$ by

$$\tilde{c}(a|b) := \widetilde{c_b}(a) = \frac{c_b(a)}{|c_b|} = \frac{c(a,b)}{\sum_{a' \in A} c(a',b)}.$$ □

Definition 2.13. Given a set Ω, a *conditional Ω-probability model for A and B (constrained to C)* is a mapping $q\colon \Omega \to \mathcal{M}(A|B)$ (or $q\colon \Omega \to \mathcal{M}_C(A|B)$). □

When talking about counting informations (and inside-outside informations in the next section), we assume the previously established requirements for X, Y, Ω, c and p (see page 7). Furthermore, we require that X and Y both contain a special symbol $\perp$ such that $c(\perp) = 0$. $\perp$ can easily be added to any previously defined X and Y without affecting the requirement for countability. The notation $U_{\not\perp} := U \setminus \{\perp\}$ shall be defined for any set U.

Definition 2.14. Let A, B and C be sets such that $C \subseteq A \times B$. A *counting information* is a triple $\varkappa = (q, \lambda, \pi)$ such that

$$q\colon \Omega \to \mathcal{M}_C(A|B), \qquad \lambda\colon X_{\not\perp} \times Y_{\not\perp} \to [0,1], \qquad \pi\colon X_{\not\perp} \times Y_{\not\perp} \to \mathbb{R}^C_{\geq 0}.$$ □

The motivation for this definition is to model hidden information y as consisting of countable events $c \in C$. The mapping λ describes whether (and, possibly, with what probability[2]) a certain y can be the cause for a certain observation. For $\lambda(x,y) > 0$, the C-corpus $\pi(x,y)$ describes how often each countable event occurs in this hidden information.
Following the assumption that the countable events C fully encode the hidden information Y, we can use these intuitive notions to describe the original probability model p in terms of the counting information.

[2]Most instances of λ use only integer images, i. e. $\lambda(X_{\not\perp} \times Y_{\not\perp}) = \{0,1\}$, thus following this intuitive notion. However, the possibility of using fractional values for $\lambda(x,y)$ is occasionally useful, e. g., to define a counting information for the IBM Model 1 in [BSV15, pp. 23].

Definition 2.15. Given $\varkappa = (q, \lambda, \pi)$, the *induced model* $\varkappa^\flat \colon \Omega \to \mathbb{R}^{Y \times X}$ is given by

$$(\varkappa^\flat)_\omega(y,x) := \begin{cases} \lambda(x,y) \cdot q_\omega\big(\pi(x,y)\big) & \text{if } x, y \neq \bot, \\ 1 - \sum_{x',y' \neq \bot} \lambda(x',y') \cdot q_\omega\big(\pi(x',y')\big) & \text{if } x = y = \bot, \\ 0 & \text{otherwise.} \end{cases}$$ □

This shows why the introduction of $\bot$ into X and Y was useful. The definition ensures that $|(\varkappa^\flat)_\omega| = 1$. Therefore, $\varkappa^\flat$ is an Ω-probability model for $Y \times X$ iff $(\varkappa^\flat)_\omega(\bot, \bot) \geq 0$. We call $\varkappa$ *proper* in this case.
Since we now have a probability model $p = \varkappa^\flat$ as required by the corpus-based step mapping, we can lift its complete-data corpus into the domain of the counting information, obtaining a new complete-data corpus

$$c\langle\omega, \varkappa\rangle \colon C \to \mathbb{R}_{\geq 0}, \quad (a,b) \mapsto \sum_{x,y} c\langle\omega, \varkappa^\flat\rangle(y,x) \cdot \big(\pi(x,y)\big)(a,b).$$

Definition 2.16. Given a set Ω and a conditional Ω-probability model q for A and B, the *conditional maximum likelihood estimator* for q is the mapping

$$\mathrm{cmle}_q \colon \mathbb{R}_{\geq 0}^{A \times B} \to \mathcal{P}(\Omega), \quad c \mapsto \operatorname*{argmax}_\omega q_\omega(c).$$ □

Using this definition, the *simple counting step mapping*[3] for $\varkappa$ is

$$(\!|\varkappa|\!)_{\mathrm{sc}} \colon \Omega_0 \to \mathcal{P}(\Omega), \quad \omega \mapsto \mathrm{cmle}_q\big(c\langle\omega, \varkappa\rangle\big) = \operatorname*{argmax}_{\omega'} q_{\omega'}\big(c\langle\omega, \varkappa\rangle\big).$$

The simple counting step mapping has two advantages over $(\!|\cdot|\!)_{\mathrm{cb}}$: First, many language models can be described in terms of countable events only, such that $\Omega = C$. In this case, Lemma 2.8 can be extended to solve the argmax by computing the empirical probability distribution of $c\langle\omega, \varkappa\rangle$.

Lemma 2.17. Let c be a non-empty and finite $A \times B$-corpus, and q a Ω-probability model for A given B. If there exists $\hat{\omega} \in \Omega$ such that $q_{\hat{\omega}} = \tilde{c}$, then $\hat{\omega} \in \mathrm{cmle}_p(c)$.

Proof. Let $b \in B$. Then $q^{(b)}$, defined by $q^{(b)}_\omega := q_\omega(b)$, is an Ω-probability model for A. Since $q_{\hat{\omega}} = \tilde{c}$, we have $q^{(b)}_{\hat{\omega}}(a) = q_{\hat{\omega}}(a|b) = \widetilde{c_b}(a)$. From this follows $\hat{\omega} \in \mathrm{mle}_{q^{(b)}}(c_b)$ because of Lemma 2.8. This means that $q^{(b)}_{\hat{\omega}}(c_b) \geq q^{(b)}_\omega(c_b)$ for any $\omega \in \Omega$. Hence, we have, for any $\omega \in \Omega$,

$$q_{\hat{\omega}}(c) = \prod_{b \in B} q_{\hat{\omega}}(b)(c_b) \geq \prod_{b \in B} q_\omega(b)(c_b) = q_\omega(c).$$ □

Second, even if this lemma is not applicable, the set of countable events C is usually much smaller than the set of all observations X or hidden information Y, making the evaluation of the argmax

[3] [BSV15, p. 13] show that this step mapping is equivalent to $(\!|\varkappa^\flat|\!)_{\mathrm{cb}}$, and thus also nondecreasing.

more tractable than for the corpus-based step mapping. For example, when considering probabilistic context-free grammars, X (the set of all sentences) and Y (the set of all parse trees) are both countably infinite, but C (the set of all derivation rules) is finite.

2.5 Regular tree grammars

The third and most specific type of step mapping applies to language models whose hidden information can be described as trees, such that the countable events are labels in these trees' nodes. We therefore need to introduce some terminology regarding alphabets and trees first.

Definition 2.18. An *alphabet* is a finite set. Its elements are called *letters*. □

Terminology and notation regarding alphabets will be useful both for the definition of trees in this chapter and for the discussion of the Hidden Markov model in the next chapter.[4]

Definition 2.19. Let Σ be an alphabet. For any $n \in \mathbb{N}$, a sequence $\sigma = \sigma_1 \cdots \sigma_n$ of letters $\sigma_1, \ldots, \sigma_n \in \Sigma$ is called a *word* over Σ with *length* $|\sigma| = n$. For $n = 0$, the *empty word* is denoted by ε. Given two words $\sigma = \sigma_1 \cdots \sigma_n$ and $\tau = \tau_1 \cdots \tau_m$, their *concatenation* is the word

$$\sigma\tau := \sigma_1 \cdots \sigma_n \tau_1 \cdots \tau_m,$$

and especially $\sigma\varepsilon := \sigma$ and $\varepsilon\tau := \tau$. The concatenation of two sets S and T of words over Σ is

$$S \cdot T := \{\sigma\tau | \sigma \in S, \tau \in T\}. \qquad \square$$

In this regard, the alphabet Σ can be considered to be a set of words since letters are isomorphic to one-letter words.

Definition 2.20. Let Σ be an alphabet. The *Kleene star* of Σ is the set of all words over Σ, i. e.,

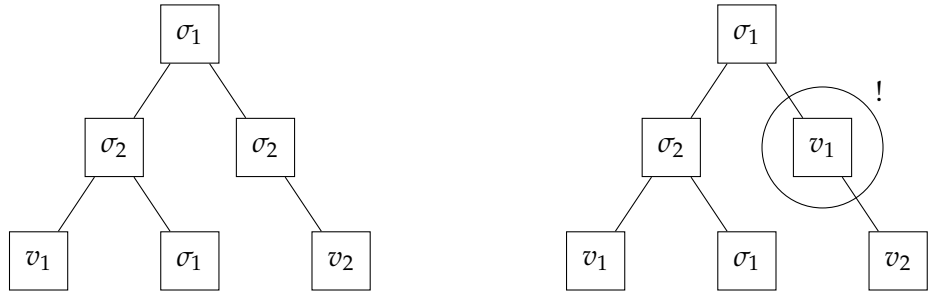
$$\Sigma^* := \bigcup_{i=0}^{\infty} \Sigma^i, \quad \text{where } \Sigma^i := \begin{cases} \{\varepsilon\} & \text{if } i = 0, \\ \Sigma^{i-1} \cdot \Sigma & \text{otherwise.} \end{cases} \qquad \square$$

We also use $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$ to refer to the set of *nonempty words* over Σ.

Definition 2.21. Let Σ be an alphabet and V be a set such that $\Sigma \cap V = \emptyset$. The *set* $U_\Sigma(V)$ *of unranked trees over* Σ *indexed by* V is the smallest set T such that

$$V \subseteq T \quad \text{and} \quad \forall k \in \mathbb{N} \colon \forall \sigma \in \Sigma, t_1, \ldots, t_k \in T \colon \sigma(t_1, \ldots, t_k) \in T. \qquad \square$$

[4] The terms "alphabet", "letters" and "words" are commonly used in formal language theory to describe a set of base symbols, its elements and strings of them. However, in natural language processing, the individual symbols are words rather than letters, and their strings are sentences rather than words. These terms will consequently be applied in Chapter 3.



$\sigma_1\big(\sigma_2(v_1,\sigma_1),\sigma_2(v_2)\big) \in U_\Sigma(V)$ $\qquad$ $\sigma_1\big(\sigma_2(v_1,\sigma_1),v_1(v_2)\big) \notin U_\Sigma(V)$

Figure 2.1: Example (left) and counter-example (right) for trees from $U_\Sigma(V)$, where $\Sigma = \{\sigma_1, \sigma_2\}$ and $V = \{v_1, v_2\}$.

The notation $\sigma(t_1, \ldots, t_k)$ refers to the tree which has the label σ at its root and the subtrees $t_1, \ldots, t_k$ in that order. For $k = 0$, we write σ instead of $\sigma()$. For $V = \emptyset$, we write U_Σ instead of $U_\Sigma(\emptyset)$. Trees from $U_\Sigma(V)$ have labels from $\Sigma \cup V$ at each node, but labels from V are only permitted at leaves (see Figure 2.1). We refer to positions in the tree using *Gorn notation*.

Definition 2.22. Given $t \in U_\Sigma(V)$, the set of *positions* in t is defined recursively by

$$\mathrm{pos}(t) := \begin{cases} \{\varepsilon\} & \text{if } t = v \in V, \\ \{\varepsilon\} \cup \bigcup_{i=1}^{k} \{i\} \cdot \mathrm{pos}(t_i) & \text{if } t = \sigma(t_1, \ldots, t_k). \end{cases}$$ □

Positions are words from $\mathbb{N}^*$. Figure 2.2 illustrates the set of positions in a tree, and some of the operations on positions as defined below.

Definition 2.23. Given $t \in U_\Sigma(V)$ and $w \in \mathrm{pos}(t)$, $t(w)$ denotes the *label* in t at position w, i. e.,

$$t(w) := \begin{cases} v & \text{if } t = v \in V \text{ and } w = \varepsilon, \\ \sigma & \text{if } t = \sigma(t_1, \ldots, t_k) \text{ and } w = \varepsilon, \\ t_i(w') & \text{if } t = \sigma(t_1, \ldots, t_k) \text{ and } w = iw' \text{ where } i \in \mathbb{N}, w' \in \mathrm{pos}(t_i). \end{cases}$$

Moreover, $t|_w$ denotes the *subtree* in t at position w, i. e.,

$$t|_w := \begin{cases} t & \text{if } w = \varepsilon, \\ t_i|_{w'} & \text{if } t = \sigma(t_1, \ldots, t_k) \text{ and } w = iw', \end{cases}$$

and for any $t' \in U_\Sigma(V)$, $t[t']_w$ denotes the tree that results from replacing $t|_w$ by t', i. e.,

$$t[t']_w := \begin{cases} t' & \text{if } w = \varepsilon, \\ \sigma(t_1, \ldots, t_{i-1}, t_i[t']_{w'}, t_{i+1}, t_k) & \text{if } t = \sigma(t_1, \ldots, t_k) \text{ and } w = iw' \\ & \text{where } i \in \mathbb{N}, w' \in \mathrm{pos}(t_i). \end{cases}$$

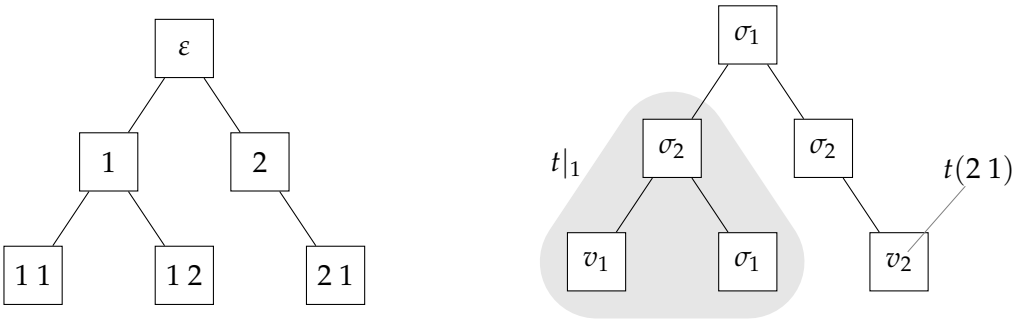


Figure 2.2: Left: A tree from $U_{\mathbb{N}^*}$ in which every node is labeled with its position within the tree. Right: Illustration of the operations $t(w)$ and $t|_w$ for a tree t and positions $w \in \mathrm{pos}(t)$.

Finally, $\mathrm{succ}(t)$ denotes the set of *successors* of t, i. e.,

$$\mathrm{succ}(t) := \begin{cases} \emptyset & \text{if } t = v \in V, \\ \{t_1, \ldots, t_k\} & \text{if } t = \sigma(t_1, \ldots, t_k). \end{cases}$$ □

Definition 2.24. Let Σ be an alphabet and $\square \notin \Sigma$ be some symbol. A *1-context over* Σ is a tree $t \in U_\Sigma(\{\square\})$ such that $\square$ appears at exactly one position in t. The set of all such 1-contexts is denoted by C_Σ. □

1-contexts are used to describe trees that are not fully known yet: The symbol $\square$ is a placeholder for a missing subtree. Given a 1-context $t \in C_\Sigma$ and a tree $t' \in U_\Sigma(V)$, we abbreviate $t[t'] := t[t']_w$ where $w \in \mathrm{pos}(t)$ such that $t(w) = \square$. In other words, $t[t']$ is the tree that results from t when the node labeled with $\square$ is replaced by t'.

Definition 2.25. A *ranked alphabet* is a pair (R, rk) where R is an alphabet and $\mathrm{rk}\colon R \to \mathbb{N}$ is a mapping. rk is said to assign a *rank* or *arity* to each symbol in R. □

A ranked alphabet is usually denoted only by R. The existence of rk is implied.

Definition 2.26. Let R be a ranked alphabet and V be a set. The *set* $T_R(V)$ *of ranked trees over* R *indexed by* V is defined by

$$T_R(V) := \big\{ t \in U_R(V) \;\big|\; \forall w \in \mathrm{pos}(t) : t(w) \in R \Rightarrow \mathrm{rk}\big(t(w)\big) = \big|\mathrm{succ}\big(t|_w\big)\big| \big\}.$$ □

In other words, the number of subtrees of each position in t with a label from R must be equal to the rank of that label.

Definition 2.27. Given an alphabet Σ, a *regular tree grammar (RTG) over* Σ is a triple $\mathcal{G} = (Q, q_0, R)$ where Q is a nonempty alphabet (of *grammar states*), $q_0 \in Q$ is an *initial state*, and $R \subset Q^* \times \Sigma \times Q$ is a finite ranked alphabet (of *rules*) such that

$$\forall \rho = \big((q_1, \ldots, q_k), \sigma, q\big) \text{ in } R\colon\ \mathrm{rk}(\rho) = k.$$ □

We will write $\big((q_1, \ldots, q_k), \sigma, q\big)$ as $q \to \sigma(q_1, \ldots, q_k)$.

Definition 2.28. Let $\mathcal{G} = (Q, q_0, R)$ be an RTG. The *family* $\big(D^q(\mathcal{G}) \,\big|\, q \in Q\big)$ *of partial abstract syntax trees of* $\mathcal{G}$ is the smallest Q-indexed family $(D^q | q \in Q)$ such that, for all $q \in Q$,

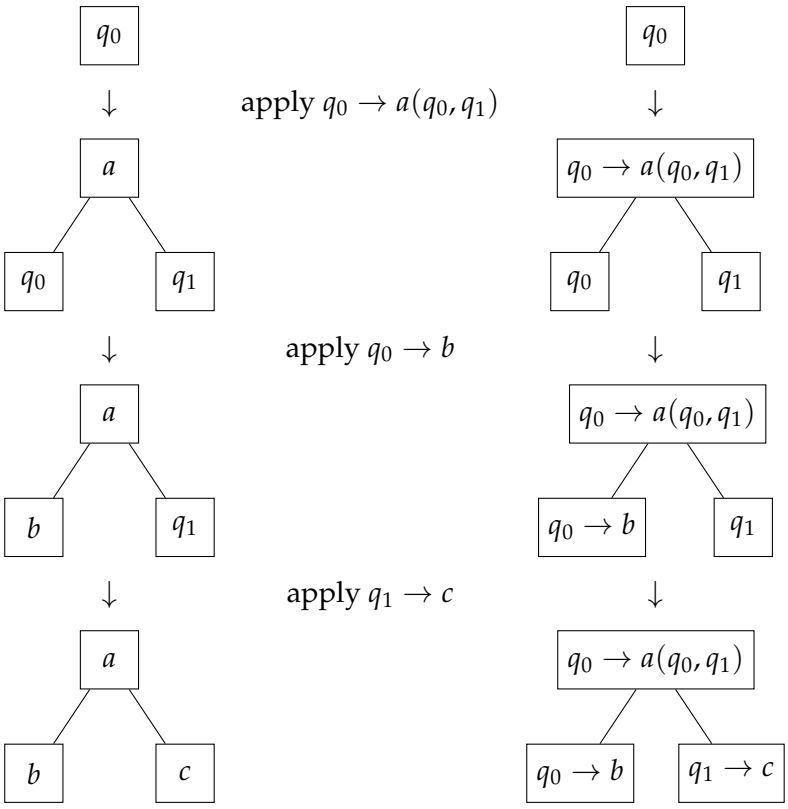
$$q \in D^q \quad \text{and} \quad \forall \rho = \big(q \to \sigma(q_1, \ldots, q_k)\big) \text{ in } R, d_1 \in D^{q_1}, \ldots, d_k \in D^{q_k} \colon \rho(d_1, \ldots, d_k) \in D^q.$$

An *abstract syntax tree* is a partial abstract syntax tree $d \in D^{q_0}(\mathcal{G})$ that does not have any labels from Q. □

An RTG generates a language (i. e., a countable set) of trees. Trees are derived by starting with a tree containing only a root node with the label q_0, then successively replacing nodes with labels from Q according to the RTG's rule set until no such nodes are left. The resulting tree is in U_Σ and its abstract syntax tree is in T_R. Partial abstract syntax trees are in $T_R(Q)$. For example, consider the RTG $\mathcal{G} = \big(\{q_0, q_1\}, q_0, R\big)$ where $\Sigma = \{a, b, c\}$ and

$$R = \{q_0 \to a(q_0, q_1), \quad q_0 \to b, \quad q_1 \to c\}.$$

The tree $a(b, c)$ is derived like this: (Each row shows the partial tree from $U_\Sigma(Q)$ on the left and the partial abstract syntax tree from $T_R(Q)$ on the right.)



From the (partial) abstract syntax tree on the right, the tree on the left can be derived by substituting each label $\rho = \big(q = \sigma(q_1, \ldots, q_k)\big)$ in R with the symbol $\sigma \in \Sigma$ that it contains, without

changing labels from Q. We shall call this projection $\pi_\Sigma : T_R(Q) \to U_\Sigma(Q)$.

Definition 2.29. Let $\mathcal{G}$ be an RTG over Σ. The *language of* $\mathcal{G}$ is the set

$$[\![\mathcal{G}]\!] := \{t \in U_\Sigma | \exists d \in D^{q_0}(\mathcal{G}) \colon t = \pi_\Sigma(d)\}.$$

A language $L \subseteq U_\Sigma$ is *regular* if there exists an RTG $\mathcal{G}$ over Σ such that $[\![\mathcal{G}]\!] = L$. □

Definition 2.30. An RTG $\mathcal{G}$ over Σ is called *deterministic* if, for any $(q_1, \ldots, q_k) \in Q^*$ and $\sigma \in \Sigma$, there is at most one $q \in Q$ such that $\big(q \to \sigma(q_1, \ldots, q_k)\big) \in R$. A language $L \subseteq U_\Sigma$ is called *deterministic* if there exists an RTG $\mathcal{G}$ with $[\![\mathcal{G}]\!] = L$. □

Definition 2.31. An RTG $\mathcal{G}$ over Σ is called *unambiguous* if, for every tree $t \in [\![\mathcal{G}]\!]$, there exists exactly one abstract syntax tree $d \in D^{q_0}(\mathcal{G})$ such that $\pi_\Sigma(d) = t$. The set of all unambiguous RTG over Σ shall be denoted as $\mathcal{R}(\Sigma)$. □

From determinism results unambiguity. To see why, recall the derivation of $t = a(b, c)$ above. Since $\mathcal{G}$ in this example is deterministic, the abstract syntax tree can be recovered from t by traversing the nodes from the bottom up and assigning the rules that are used at that position. At each position, we know the symbol σ at this position in the tree and the states $q_1, \ldots, q_k$ from the left sides of the rules used for the child nodes. Therefore, the state q (and therefore the rule ρ) for this position can be chosen deterministically.

Definition 2.32. A *probabilistic regular tree grammar (PRTG) over* Σ is a pair $(\mathcal{G}, p)$ of an RTG $\mathcal{G} = (Q, q_0, R)$ over Σ and a mapping $p : Q \to \mathbb{R}_{\geq 0}^{Q^* \times \Sigma}$ constrained to R in the following way:

$$\forall q \in Q, u \in Q^* \times \Sigma : \big(p(q)\big)(u) \neq 0 \Rightarrow (u, q) \in R.$$

$(\mathcal{G}, p)$ is called *proper* if $p \in \mathcal{M}_R(Q^* \times \Sigma | Q)$. The *meaning* of $(\mathcal{G}, p)$ is the mapping

$$[\![(\mathcal{G}, p)]\!] \colon U_\Sigma \to \mathbb{R}_0, \quad t \mapsto \sum_{d \in D^{q_0}(\mathcal{G}) : \pi_\Sigma(d) = t} p(d).$$

Herein, $p(d) := p\big(\pi(d)\big)$, where $\pi(d)$ is an R-corpus with

$$\big(\pi(d)\big)(p) := \big|\{w \in \mathrm{pos}(d) : d(w) = p\}\big|.$$ □

We usually write a PRTG $(\mathcal{G}, p)$ as just $\mathcal{G}$ and imply the existence of p. The meaning function assigns a probability to a tree from $[\![\mathcal{G}]\!]$ by summing the probability of all derivations resulting in that tree, where the probability of a derivation is the product of the probability of each rule occurring in it. This implies that rule applications are statistically independent from each other. When describing hidden information in terms of PRTG in the next section, the notion of *inside and outside weights* will be useful to judge, broadly speaking, how much a certain state contributes to the derivations of a certain observation.

Definition 2.33. Let $\mathcal{G} = (Q, q_0, R)$ be a PRTG. The *inside weight* of a state $q \in Q$ is given by

$$\beta(q) := \sum_{d \in D^q(\mathcal{G}) \cap T_R} p(d). \qquad \square$$

The sum goes over all complete abstract syntax trees rooted at q. The inside weight thus measures the collective probability of all derivations starting at q. Inside weights are usually calculated by noting that each $d \in D^q(\mathcal{G}) \cap T_R$ must have a rule of the form $q \to \dots$ at its root. The remaining probabilities can then be expressed as the inside weights of the states that emerge from this rule application, giving

$$\beta(q) = \sum_{q_1,\dots,q_k,\sigma} p\big(q \to \sigma(q_1,\dots,q_k)\big) \cdot \beta(q_1) \cdots \beta(q_k). \tag{2.1}$$

The set of all such equations is a non-linear equation system in the variables $\beta(q)$ for $q \in Q$. In some cases, this system can be solved intuitively by starting with those equations where no $\beta(q_i)$ occurs on the right-hand side, then substituting the obtained value in the other equations until they are all solved. This is not possible, however, if states derive other states in a cyclic way, e. g.,

$$R = \left\{ \begin{aligned} &\rho_1 = q_0 \to \sigma_1(q_1, q_1), \\ &\rho_2 = q_1 \to \sigma_2(q_0), \\ &\rho_3 = q_1 \to \sigma_3 \end{aligned} \right\} \quad \rightsquigarrow \quad \begin{aligned} \beta(q_0) &= p(\rho_1) \cdot \beta(q_1)^2 \\ \beta(q_1) &= p(\rho_2) \cdot \beta(q_0) + p(\rho_3) \end{aligned}$$

In the general case, [BSV15, pp. 6] show that the inside weights β are the least fixpoint of the mapping $F : (\mathbb{R}_{\geq 0} \cup \{\infty\})^Q \to (\mathbb{R}_{\geq 0} \cup \{\infty\})^Q$, given by

$$\big(F(u)\big)(q) := \sum_{q_1,\dots,q_k,\sigma} p\big(q \to \sigma(q_1,\dots,q_k)\big) \cdot u(q_1) \cdots u(q_k).$$

Therefore, $\beta = \lim_{n\to\infty} F^n(u_0)$ where $u_0(q) := 0 \ \forall q \in Q$ can be approximated by performing as many iterations of F as desired.

Definition 2.34. Let $\mathcal{G} = (Q, q_0, R)$ be a PRTG. The *outside weight* of a state $q \in Q$ is given by

$$\alpha(q) := \sum_{d \in D^{q_0}(\mathcal{G}) \colon \exists d' \in C_R \colon d = d'[q]} p(d). \qquad \square$$

The sum goes over all partial abstract syntax trees starting from q_0, where only one unexpanded state is left, and that state is q. The outside weight measures the collective probability of all derivations that use q, but without considering derivations at or below q. Similar to what we did with inside weights, we can expand this definition into

$$\alpha(q) = \delta^q_{q_0} + \sum_{q', q_1,\dots,q_k,\sigma,m \colon q_m = q} \alpha(q') \cdot p\big(q' \to \sigma(q_1,\dots,q_k)\big) \cdot \prod_{l \neq m} \beta(q_l). \tag{2.2}$$

In this formulation, the sum goes over all rules that derive q, that is, q is among the states $q_1, \ldots, q_k$ on the right hand side of the rule, at index m. The outside weight $\alpha(q)$ considers the outside weight $\alpha(q')$ of the previous state and the inside weight of all states adjacent to q, but not the inside weight of q itself. Finally, we need to account for $q = q_0$, in which case the trivial tree d containing only a q_0-labeled root node must be considered. Since $p(d) = 1$ for this tree, it is accounted for in the above formula through the Kronecker symbol

$$\delta_{q_0}^{q} := \begin{cases} 1 & \text{if } q_0 = q, \\ 0 & \text{otherwise.} \end{cases}$$

Assuming that the inside weights $\beta(q)$ have already been calculated, the equations for $\alpha(q)$ form a linear equation system that can be solved efficiently with the standard algorithms for linear equation systems.

2.6 Inside-outside step mapping

We can now resume the task of describing hidden information in terms of trees, such that the countable events, as introduced by the counting information, are labels in these trees' nodes. Similar to how a language model can be described as a counting information for use with the simple counting step mapping, a language model may also be described as an *inside-outside information*, for use with the inside-outside step mapping.
Again, we assume the previous definitions of X, Y, c, A, B and C throughout this section.

Definition 2.35. An *inside-outside (IO) information* is a quadruple[5] $\mu = (q, \pi_1, K, H)$ where

- $C \subseteq A \times B$ is a ranked alphabet such that $Y_{\not\perp} \subseteq T_C$ contains ranked trees over C,
- $q\colon \Omega \to \mathcal{M}_C(A|B)$ as for counting informations,
- $\pi_1\colon Y_{\not\perp} \to X_{\not\perp}$ maps hidden information to observations,
- $K \in \mathcal{R}(C)$ is a unambiguous RTG with $[\![K]\!] = Y_{\not\perp}$ and
- $H\colon X_{\not\perp} \to \mathcal{R}(C)$ assigns an unambiguous RTG to every observation such that

$$\forall x \in X_{\not\perp}\colon [\![H(x)]\!] = \pi_1^{-1}(x).$$

Furthermore, we require that, for every $\omega \in \Omega$, the PRTG (K, p'_ω) is proper, where

$$p'_\omega(\rho) := \begin{cases} q_\omega(a|b) & \text{if } \rho = \big(q \to (a,b)(q_1, \ldots, q_k)\big) \text{ in } R(K), \\ 0 & \text{otherwise.} \end{cases}$$

□

[5]The definitions in this chapter have previously diverged from [BSV15] by omitting the set Z, which is only required for translation models, and otherwise set to $Z = \{\emptyset\}$ for language models. For IO informations, this leads to additional divergence from the definition in [BSV15], where μ is introduced as a quintuple $\mu = (q, \pi_1, \pi_2, K, H)$. Both definitions are congruent for language models by letting $\pi_2(y) := \emptyset$.

Each hidden information y is a tree from T_C, and a conditional probability $q_\omega(a|b)$ can be assigned to each countable event $c = (a, b)$ that occurs in the tree. Each hidden information y belongs to exactly one observation x. The set of all hidden informations $Y_{\not\perp}$ is generated by the PRTG (K, p'_ω), and for each observation x, the set of all hidden informations leading to this observation is generated by the PRTG $\big(H(x), p'_\omega\big)$.
The description of a language model using an IO information is a special case of the description via counting informations, since an IO information induces a counting information.

Definition 2.36. Let $\mu = (q, \pi_1, K, H)$ be an IO information. The *induced counting information* $\mu^\flat = (q, \lambda, \pi)$ is given by

$$\lambda(x, y) := \begin{cases} 1 & \text{if } \pi_1(y) = x, \\ 0 & \text{otherwise,} \end{cases}$$

$$\big(\pi(x, y)\big)(a, b) := \begin{cases} \big|\{w \in \mathrm{pos}(y) : y(w) = (a, b)\}\big| & \text{if } \pi_1(y) = x, \\ 0 & \text{otherwise.} \end{cases}$$

□

The induced counting information $\mu^\flat$ is always proper. [BSV15, p. 15] Using μ, we can construct a suitable complete-data corpus:

$$c\langle\omega, \mu\rangle : C \to \mathbb{R}_{\geq 0}, \quad (a, b) \mapsto \sum_x c(x) \cdot \chi_{\omega,x}(a, b).$$

Unlike $c\langle\omega, \varkappa\rangle$, this definition uses the original corpus c instead of relying on the complete-data corpus of the previous step mapping. In this expression, $(\chi_{\omega,x} | \omega \in \Omega, x \in X_{\not\perp})$ is a family of C-corpora, defined by

$$\chi_{\omega,x}(a, b) := \beta_x(q_0)^{-1} \cdot \sum_{\rho=(q\to(a,b)(q_1,\ldots,q_k)) \text{ in } R} \alpha_x(q) \cdot p'_\omega(\rho) \cdot \beta_x(q_1) \cdots \beta_x(q_k), \tag{2.3}$$

where (Q, q_0, R) is the PRTG $H(x)$ with rule probabilities p'_ω as defined above, and α_x and β_x are the outside and inside weights of $H(x)$, respectively.
To understand the formula for $\chi_{\omega,x}(a, b)$, recall that $H(x)$ is an RTG such that the set of its complete abstract syntax trees is $\pi_1^{-1}(x)$, the set of all hidden informations that lead to the observation x. Each of these derivations d has a probability $p(d)$, and the sum of all these probabilities is the inside weight $\beta_x(q_0)$. The term $\chi_{\omega,x}(a, b)$ measures the contribution of rules containing the countable event (a, b) to this total weight (such that multiple occurrences of (a, b) in a single derivation are counted multiple times). For every such derivation, we have the part that leads up to the state q (described by the outside weight $\alpha_x(q)$), the rule ρ that produces (a, b) with probability $p'_\omega(\rho) = q_\omega(a|b)$, and the derivations that occur below this rule application (described by the inside weights $\beta_x(q_i)$ of the produced states).

Given the complete-data corpus, we can once more apply a maximum-likelihood estimator for q to arrive at the *inside-outside step mapping*

$$(\!|\mu|\!)_{\mathrm{io}} \colon \Omega_0 \to \mathcal{P}(\Omega), \quad \omega \mapsto \mathrm{cmle}_q\big(c\langle\omega, \mu\rangle\big) = \operatorname*{argmax}_{\omega'} q_{\omega'}\big(c\langle\omega, \mu\rangle\big).$$

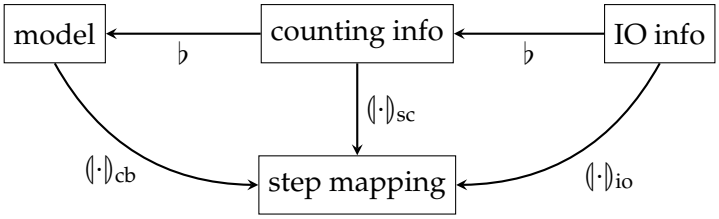
[BSV15, pp. 16] shows that $(\!|\mu|\!)_{\mathrm{io}} = (\!|\mu^\flat|\!)_{\mathrm{sc}}$ for all IO informations μ, from which it follows that $(\!|\cdot|\!)_{\mathrm{io}}$ is nondecreasing.

2.7 Review

This chapter introduced the three formalizations of language models proposed by [BSV15]:

1. models $p \colon \Omega \to \mathcal{M}(Y \times X)$,
2. counting informations $\varkappa = (q, \lambda, \pi)$,
3. inside-outside informations $\mu = (q, \pi_1, K, H)$.

Each formalization is more specific than the one before it, as evidenced by the $\flat$ operator that converts each formalization into the previous one. A step mapping can be defined for each of these formalizations. [BSV15] illustrate these relationships as follows:



In the definition of both $\varkappa = (q, \lambda, \pi)$ and $\mu = (q, \pi_1, K, H)$, the only part that depends on ω is the probability model q. All other parts are fixed as soon as the language model is instantiated, and are therefore not subject to training. For many language models, it is sufficient to choose $\Omega := \mathcal{M}_C(A|B)$ and $q = \mathrm{id}$ (i. e., $q_\omega = \omega$ for all ω). Applying Lemma 2.17, we see that the empirical probability distribution for the complete-data corpus is an element of $(\!|\mu|\!)_{\mathrm{io}}(\omega)$ (and similar for the simple counting step mapping). Since the EM algorithm only needs one element from $(\!|\mu|\!)_{\mathrm{io}}(\omega)$, we can use

$$(\!|\mu|\!)'_{\mathrm{io}}(\omega) := \{\widetilde{c\langle\omega, \mu\rangle}\}$$

instead, which is much less expensive to compute than the full argmax.

3 The Hidden Markov model

This chapter is accompanied by a formulary, which can be found in Appendix B on page 47. The definition of the Hidden Markov model, the forward algorithm, backward algorithm and Baum-Welch algorithm in this chapter are based on [JM09, pp. 210], except where otherwise noted.

Definition 3.1. A *Hidden Markov model* (HMM) is a quintuple[1] $\mathcal{H} = (Q, V, \#, t, e)$ such that

- Q is a non-empty alphabet (*states*),
- V is a non-empty alphabet (*words*),
- $\# \notin Q \cup V$ (*initial* and *final state*),
- $t \in \mathcal{M}(Q \cup \{\#\} | Q \cup \{\#\})$ (*transition probability distribution*), and
- $e \in \mathcal{M}(V|Q)$ (*emission probability distribution*). □

From here on, we will abbreviate $Q \cup \{\#\}$ as $Q_\#$.
The Hidden Markov model describes a sentence as being the result of the progression of a probabilistic state machine that starts out in #, traverses states from Q, and in the end reaches # again. Each time a state from Q is reached, a word from V is emitted. The sequence of all these emitted words is the sentence that is observed.
When a sentence $v = v_1 \cdots v_n$ in V^* is observed, is must have been caused by a certain sequence of states $q = q_1 \cdots q_n$ in Q^*, but it is not known which one it was, only that the lengths of both sequences agree. Therefore, by the law of total probability,

$$P(v) = \sum_{q \in Q^n} P(v, q) = \sum_{q \in Q^n} P(v|q) \cdot P(q).$$

Definition 3.2. A stochastic process is said to have the *Markov property* if the conditional probability distribution of the next state of the process only depends on the present state, not on the states before it. □

A Hidden Markov model exhibits the Markov property in two separate ways: First, the conditional probability distribution of each state depends only on the state directly preceding it. Second, the conditional probability distribution of each emitted word depends only on the state that was inhabited at the time of emission. These two conditional probability distributions are called t and e, and are part of the quintuple $\mathcal{H} = (Q, V, \#, t, e)$ as defined above.

[1] This definition diverges in structure from [JM09] in one significant way: It uses a single state # instead of a pair of initial state q_0 and final state q_F. This allows us to use a single probability distribution t to describe all state transitions, instead of the triple of matrices $(a_{0i})_i$, $(a_{ij})_{(i,j)}$, and $(a_{iF})_i$ that appear in [JM09].

The progression of the probabilistic state machine of $\mathcal{H}$ through the state sequence $q = q_1 \cdots q_n$ involves several separate stochastic events: entering each state $q_1, \ldots, q_n$ in that order, then entering the state # after n other states. Therefore, by the chain rule,

$$P(q_1 \cdots q_n) = P(q_1, \ldots, q_n, n) = P(q_1) \cdot P(q_2|q_1) \cdots P(q_n|q_1, \ldots, q_{n-1}) \cdot P(n|q_1, \ldots, q_n).$$

The last factor, $P(n|q_1, \ldots, q_n)$ is the probability of the state sequence having length n if $q_1, \ldots, q_n$ are known or, in other words, the probability of the state sequence terminating (by the state machine coming back to #) after these n states. Since, by the first Markov property, each state only depends on the one directly preceding it, we can reformulate each factor in terms of the transition probability distribution t, i. e.,

$$\left.\begin{aligned} P(q_1) &= t(q_1|\#), \\ P(q_i|q_1, \ldots, q_{i-1}) &= t(q_i|q_{i-1}), \\ P(n|q_1, \ldots, q_n) &= t(\#|q_n) \end{aligned}\right\} \quad \Rightarrow \quad P(q_1 \cdots q_n) = t(q_1|\#) \cdot t(q_2|q_1) \cdots t(q_n|q_{n-1}) \cdot t(\#|q_n).$$

In a similar way, we can rewrite $P(v|q)$ as

$$P(v_1 \cdots v_n|q_1 \cdots q_n) = \prod_{i=1}^{n} P(v_i|q_1 \cdots q_n) = \prod_{i=1}^{n} e(v_i|q_i)$$

because of the second Markov property. Putting all these results into the original equation for $P(v)$, we obtain

$$P(v = v_1 \cdots v_n) = \sum_{q_1,\ldots,q_n \in Q} t(q_1|\#) \cdot e(v_1|q_1) \cdot \prod_{i=2}^{n} \big[t(q_i|q_{i-1}) \cdot e(v_i|q_i)\big] \cdot t(\#|q_n). \tag{3.1}$$

As a special case, for the empty sentence $v = \varepsilon$, we have

$$P(v = \varepsilon) = t(\#|\#) \tag{3.2}$$

since the state machine goes from the initial directly into the final state, without ever visiting a state that emits a word.

3.1 Forward and backward algorithms

When $P(v)$ is computed in this manner, the computation takes an exponential amount of time in the sentence length n since $|Q|^n$ summands need to be evaluated. However, the expressions for $P(q) \cdot P(v|q)$ for similar state sequences q share several common factors. By following a *dynamic programming* approach, i. e., by storing common subterms in a tabular memory for later re-use, the computational effort can be reduced significantly. There are two well-known schemes for dividing $P(v)$ into subterms, which lead to the *forward* and *backward algorithm*, respectively.

Definition 3.3. Let $\mathcal{H} = (Q, V, \#, t, e)$ be an HMM, $q \in Q$ be a state, $v = v_1 \cdots v_n$ in V^+ be a *nonempty* sentence, and $i \in \{1, \ldots, n\}$.[2] The *forward weight*[3] $T_v(i, q)$ is the probability of the HMM being in state q after having emitted the first i words of v, i. e.,

$$T_v(i, q) := P(v_1, \ldots, v_i, q_i = q).$$

The *backward weight* $S_v(i, q)$ is the probability of the HMM generating v when the first i words of v have already been emitted and the HMM is in state q after that many words, i. e.,

$$S_v(i, q) := P(v_{i+1}, \ldots, v_n, n | q_i = q). \qquad \square$$

The forward weight can be calculated by following the same methods as in the previous section for $P(v)$. For $i = 1$, the forward weight describes the transition from the initial state # into q, and the emission of v_1 in that state, i. e.,

$$T_v(1, q) = P(v_1, q_1 = q) = P(q_1 = q) \cdot P(v_1 | q_1 = q) = t(q|\#) \cdot e(v_1|q). \tag{3.3}$$

For $i \geq 2$, the forward weight can be calculated iteratively by first obtaining the forward weights $T_v(i-1, q')$ for any possible previous state $q' \in Q$, because

$$T_v(i, q) = P(v_1, \ldots, v_i, q_i = q) = \sum_{q' \in Q} P(v_1, \ldots, v_i, q_{i-1} = q', q_i = q)$$

by the law of total probability, and then, by the chain rule and the Markov properties of $\mathcal{H}$,

$$\begin{aligned} T_v(i, q) &= \sum_{q' \in Q} P(v_1, \ldots, v_{i-1}, q_{i-1} = q') \cdot P(q_i = q | q_{i-1} = q') \cdot P(v_i | q_i = q) \\ &= \sum_{q' \in Q} T_v(i-1, q') \cdot t(q|q') \cdot e(v_i|q) \\ &= e(v_i|q) \cdot \sum_{q' \in Q} T_v(i-1, q') \cdot t(q|q'). \end{aligned} \tag{3.4}$$

The probability $P(v)$ can then be computed in a similar way as

$$P(v) = P(v_1, \ldots, v_n, n) = \sum_{q \in Q} P(v_1, \ldots, v_n, q_n = q) \cdot P(n | q_n = q) = \sum_{q \in Q} T_v(n, q) \cdot t(\#|q). \tag{3.5}$$

The forward weights $T_v(i, q)$ exhibit a topological ordering: To compute $T_v(i, q)$ with $i > 1$, all $T_v(i-1, q')$ need to be computed first. Computing the full matrix $\big(T_v(i, q)\big)_{i,q}$ in that order, in order to finally obtain $P(v)$, yields the *forward algorithm*.

[2] [JM09] uses the term "time" and the symbol t for this index. We avoid the symbol t because it is already used for the transition probability distribution.

[3] The names "forward/backward weight" have been chosen deliberately, because we will see in the next section that these values correspond to the inside and outside weight.

Each forward weight can be computed in $O(|Q|)$ time because $|Q|$ summands need to be added. Since there are $n \cdot |Q|$ forward weights for each v, the forward algorithm runs in $O(n \cdot |Q|^2)$ time, which is much better than the exponential time required for the initial formula for $P(v)$.
The same time complexity arises when $P(v)$ is being restated in terms of backward weights. Backward weights can be computed in a similar manner to forward weights, with the difference of iterating in opposite temporal order.

$$P(v) = \sum_{q \in Q} t(q|\#) \cdot e(v_1|q) \cdot S_v(1, q) \tag{3.6}$$

$$\text{where } S_v(i,q) = \begin{cases} t(\#|q) & \text{if } i = n, \\ \sum_{q' \in Q} t(q'|q) \cdot e(v_{i+1}|q') \cdot S_v(i+1, q') & \text{otherwise.} \end{cases} \tag{3.7}$$

The *backward algorithm* works analogously to the forward algorithm: It computes the matrix $\big(S_v(i,q)\big)_{i,q}$ in *decreasing* order of i to obtain $P(v)$.

3.2 The Baum-Welch algorithm

The Baum-Welch algorithm is first stated in [BPSW70], but since notational conventions have changed considerably since then, we are using a contemporary formulation in [JM09] as a reference (see Algorithm 3.1 on page 25).
The algorithm uses two terms that have not yet been introduced: $U_v(i, q, q')$ is defined as the probability of the HMM progressing from state q at time i into state q' at time $i+1$ while generating the sentence v. $R_v(i,q)$ is the probability of the HMM being in state q at time i while generating the sentence v.

$$\begin{aligned} U_v(i,q,q') &:= P(q_i = q, q_{i+1} = q'|v) && \text{for } i \in \{1, \ldots, |v|-1\}, q, q' \in Q \\ R_v(i,q) &:= P(q_i = q|v) && \text{for } i \in \{1, \ldots, |v|\}, q \in Q \end{aligned}$$

Both U_v and R_v can be expressed in terms of the forward and backward weights T_v and S_v, by applying the same calculation rules for probabilities that were already used for deriving formulas for $P(v)$, T_v and S_v.

$$\begin{aligned} R_v(i,q) = P(q_i = q|v) &= \frac{P(v, q_i = q)}{P(v)} = \frac{P(v_1, \ldots, v_n, n, q_i = q)}{P(v)} \\ &= \frac{P(v_1, \ldots, v_i, q_i = q) \cdot P(v_{i+1}, \ldots, v_n, n|q_i = q)}{P(v)} \\ &= \frac{T_v(i,q) \cdot S_v(i,q)}{P(v)}. \end{aligned} \tag{3.8}$$

Algorithm 3.1 Baum-Welch algorithm, based on [JM09, p. 226]. To reach a local maximum (or saddle point) for the corpus likelihood $p(c)$, the outermost loop needs to be executed until (t, e) stop changing, possibly infinitely long. The loop condition is stated as "not converged" to describe that the loop is typically aborted once the changes to (t, e) per iteration fall below some manually chosen threshold.

The formulation of the algorithm has been altered from [JM09] to also train the transition probabilities for the initial and final state, and to support a corpus with multiple nonempty sentences of different length (by taking sums over the time index i in the E-step rather than in the M-step). These alterations have previously been applied successfully to an implementation of HMM in [Nel13]. We will demonstrate in section 3.3.6 that this algorithm can be extended to accept a V^*-corpus instead of V^+-corpus.

Input: HMM $\mathcal{H}_0 = (Q, V, \#, t_0, e_0)$; V^+-corpus h
Variables: $t \in \mathcal{M}(Q_\#|Q_\#)$, $e \in \mathcal{M}(V|Q)$
$\mathrm{count}_{\mathrm{tr}} \colon Q_\# \times Q_\# \to \mathbb{R}_{\geq 0}$
$\mathrm{count}_{\mathrm{em}} \colon V \times Q \to \mathbb{R}_{\geq 0}$
Output: sequence of HMM $\mathcal{H}_i$ over Q and V
such that $p_{\mathcal{H}_0}(c) \leq p_{\mathcal{H}_1}(c) \leq p_{\mathcal{H}_2}(c) \leq \ldots$

$(t, e) \leftarrow (t_0, e_0)$
while not converged **do**
  consider the HMM $(Q, V, \#, t, e)$
  $\mathrm{count}_{\mathrm{tr}}(q', q) \leftarrow 0$ **for every** $q, q' \in Q_\#$
  $\mathrm{count}_{\mathrm{em}}(w, q) \leftarrow 0$ **for every** $q \in Q$ and $w \in V$
  for $v = v_1 \cdots v_n$ in $\mathrm{supp}(h)$ **do**
    calculate all forward weights $T_v(i, q)$ and backward weights $S_v(i, q)$
    for $i \in \{1, 2, \ldots, n-1\}$ **do**
      for $q, q' \in Q$ **do**
        $\mathrm{count}_{\mathrm{tr}}(q', q) \leftarrow \mathrm{count}_{\mathrm{tr}}(q', q) + h(v) \cdot U_v(i, q, q')$
      end for
    end for
    for $i \in \{1, 2, \ldots, n\}$ **do**
      for $q \in Q$ **do**
        $\mathrm{count}_{\mathrm{em}}(v_i, q) \leftarrow \mathrm{count}_{\mathrm{tr}}(v_i, q) + h(v) \cdot R_v(i, q)$
      end for
    end for
    for $q \in Q$ **do**
      $\mathrm{count}_{\mathrm{tr}}(q, \#) \leftarrow \mathrm{count}_{\mathrm{tr}}(q, \#) + h(v) \cdot R_v(1, q)$
      $\mathrm{count}_{\mathrm{tr}}(\#, q) \leftarrow \mathrm{count}_{\mathrm{tr}}(\#, q) + h(v) \cdot R_v(n, q)$
    end for
  end for
  for $q, q' \in Q_\#$ **do**
    $t(q'|q) \leftarrow \frac{\mathrm{count}_{\mathrm{tr}}(q', q)}{\sum_{q'' \in Q_\#} \mathrm{count}_{\mathrm{tr}}(q'', q)}$
  end for
  for $q \in Q$ and $w \in V$ **do**
    $e(w|q) \leftarrow \frac{\mathrm{count}_{\mathrm{em}}(w, q)}{\sum_{w' \in V} \mathrm{count}_{\mathrm{em}}(w', q)}$
  end for
  output $(Q, V, \#, t, e)$
end while

And analogously,

$$\begin{aligned} U_v(i,q,q') &= P(q_i = q, q_{i+1} = q' | v) = \frac{P(v_1,\ldots,v_n,n,q_i=q,q_{i+1}=q')}{P(v)} \\ &= \frac{1}{P(v)} \cdot \begin{bmatrix} P(v_1,\ldots,v_i,q_i=q) \cdot P(q_{i+1}=q'|q_i=q) \\ \cdot\, P(v_{i+1}|q_{i+1}=q') \cdot P(v_{i+2},\ldots,v_n,n|q_{i+1}=q') \end{bmatrix} \\ &= \frac{T_v(i,q) \cdot t(q'|q) \cdot e(v_{i+1}|q') \cdot S_v(i+1,q')}{P(v)}. \end{aligned} \tag{3.9}$$

With these definitions, we can observe the basic motivation and method of the Baum-Welch algorithm: Given a previous estimate for t and e, the algorithm estimates how often each state is visited and how often each state transition occurs throughout the corpus, and normalizes these counts to obtain better estimates for t and e.
In particular, the algorithm employs counter variables $\text{count}_{\text{tr}}(q,q')$ and $\text{count}_{\text{em}}(w,q)$, which are reset in lines 4–5 and then computed in lines 6–22 by summing terms of the form $h(v) \cdot R_v(i,q)$ and $h(v) \cdot U_v(i,q,q')$ in a particular way. By rewriting the nested loops as closed formulas, we see that, for any $q, q' \in Q$, lines 4 and 8–12 result in

$$\text{count}_{\text{tr}}(q',q) = \sum_{v \in \text{supp}(h)} \sum_{i=1}^{|v|-1} h(v) \cdot U_x(i,q,q') \tag{3.10}$$

after line 22, since no other statements modify $\text{count}_{\text{tr}}(q',q)$. Analogously, for any $q \in Q$, lines 4 and 18–21 result in

$$\text{count}_{\text{tr}}(\#,q) = \sum_{v \in \text{supp}(h)} h(v) \cdot R_v\big(|v|,q\big), \tag{3.11}$$

$$\text{count}_{\text{tr}}(q,\#) = \sum_{v \in \text{supp}(h)} h(v) \cdot R_v(1,q). \tag{3.12}$$

Finally, because of lines 5 and 13–17, we also have, for any $q \in Q$ and $w \in V$,

$$\text{count}_{\text{em}}(w,q) = \sum_{v \in \text{supp}(h)} \sum_{\substack{i=1 \\ v_i = w}}^{|v|} h(v) \cdot R_v(i,q). \tag{3.13}$$

3.3 Deriving the Baum-Welch algorithm

[JM09] describes Baum-Welch as an instance of the EM algorithm. And indeed, the basic structure of the algorithm looks similar to the types of EM algorithms that we have introduced in Chapter 2 in several ways:

- The conditional probability distributions t and e are the model parameters that are iteratively optimized.

- The counter variables count_{tr} and count_{em} act like complete-data corpora. They are computed using the previous model parameters, then new model parameters are obtained by taking the empirical probability distribution of these corpora, which is the efficient solution to the conditional maximum-likelihood estimator that is suggested by Lemma 2.17.
- The hidden information is the state sequence q that produces the sentence v from the corpus. It can be decomposed into countable events in several ways (e. g., states only, pairs of time and states, or pairs of subsequent states).
- The way that forward weights and backward weights appear in the computation of count_{tr} and count_{em} is similar to how inside and outside weights appear in the computation of the inside-outside complete-data corpus.

Therefore, in the remainder of this chapter, we will show that the Baum-Welch algorithm can be obtained from the generic EM algorithm from Chapter 2 through suitable instantiation of the inside-outside step mapping, from which follows that its convergence properties also apply to the Baum-Welch algorithm.

3.3.1 Model parameter and countable events

For the remainder of this section, let $\mathcal{H} = (Q, V, \#, t, e)$ be an HMM. Without loss of generality, we require $Q \cap V = \emptyset$. Observations are sentences with words from V, i. e.,

$$X = V^* \cup \{\bot\}.$$

An IO information contains only one component which can be subject to training, the model parameter ω which chooses the conditional probability distribution $q_\omega \in \mathcal{M}_C(A|B)$. For a HMM, q_ω must describe both the transition and emission probability distribution. We therefore choose

$$\Omega := \mathcal{M}(Q_\#|Q_\#) \times \mathcal{M}(V|Q)$$

such that every model parameter $\omega = (t, e)$ is a pair of transition and emission probability distribution for $\mathcal{H}$. Moreover, we choose

$$\begin{aligned}
A &:= Q_\# \cup V, \\
B &:= Q_\# \times \{T\} \cup Q \times \{E\}, \\
C &:= \big\{(q', (q, T)) \mid q, q' \in Q_\#\big\} \cup \big\{(v, (q, E)) \mid v \in V, q \in Q\big\}, \\
q_{\omega=(t,e)}\big(a \,\big|\, (q, s)\big) &:= \begin{cases} t(a|q) & \text{if } s = T \text{ and } a, q \in Q_\#, \\ e(a|q) & \text{if } s = E, a \in V, \text{ and } q \in Q, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Herein, E and T are symbols such that $E, T \notin Q \cup V$ that denote if a countable event $c \in C$ is a

transition event

$$c = \big(q', (q, T)\big) \text{ with probability } q_{(t,e)}(c) = t(q'|q),$$

or an emission event

$$c = \big(v, (q, E)\big) \text{ with probability } q_{(t,e)}(c) = e(v|q).$$

Lemma 3.4. With the definitions shown above, $q_\omega \in \mathcal{M}_C(A|B)$ for all $\omega = (t, e)$.

Proof. Let $\omega = (t, e)$ in Ω. It is easy to see that $q_\omega\big(a|(q, s)\big) = 0$ for every $\big(a, (q, s)\big) \notin C$ because q_ω was defined in that way. According to Definitions 2.10 and 2.11, what remains to be shown is that $q_\omega\big((q, s)\big) \in \mathcal{M}(A)$ for every $(q, s) \in B$. For $q \in Q_\#$ and $s = T$,

$$\sum_{a\in A} q_{(t,e)}\big(a \,\big|\, (q, T)\big) = \sum_{q'\in Q_\#} \underbrace{q_{(t,e)}\big(q' \,\big|\, (q, T)\big)}_{=t(q'|q)} + \sum_{a\in A\setminus Q_\#} \underbrace{q_{(t,e)}\big(a \,\big|\, (q, T)\big)}_{=0} = \sum_{q'\in Q_\#} t(q'|q) = 1$$

because $t \in \mathcal{M}(Q_\#|Q_\#)$. Analogously, for $q \in Q$ and $s = E$, we have

$$\sum_{a\in A} q_{(t,e)}\big(a \,\big|\, (q, E)\big) = \sum_{v\in V} \underbrace{q_{(t,e)}\big(v \,\big|\, (q, E)\big)}_{=e(v|q)} + \sum_{a\in A\setminus V} \underbrace{q_{(t,e)}\big(a \,\big|\, (q, E)\big)}_{=0} = \sum_{v\in V} e(v|q) = 1$$

because $e \in \mathcal{M}(V|Q)$. □

3.3.2 Tree-shaped hidden information

In order to describe hidden information $y \in Y_{\not\perp}$ as a ranked tree of countable events from C, we assign a rank to each $c = \big(a, (q, s)\big)$ in C by

$$\mathrm{rk}\Big(\big(a, (q, s)\big)\Big) := \begin{cases} 0 & \text{if } s = E, \\ 2 & \text{if } s = T \text{ and } a \neq \#, \\ 0 & \text{if } s = T \text{ and } a = \#. \end{cases}$$

The intuition for this choice is that each transition event causes two further events: the emission event in the state that was entered, and the transition event into the state after that. Emission events do not result in further events because of the Markov property, and a transition event into the # state marks the end of the stochastic process after which no further events occur. This rank assignment results in trees such as the one in Figure 3.1.
The trees $y \in Y_{\not\perp}$ are generated by the tree grammars $H(x)$ and K. We define[4]

$$K := \big\{Q_K, (T, \#), R_K\big\} \quad \text{where} \quad Q_K := \{T\} \times Q_\# \cup \{E\} \times Q$$

[4]The set of grammar states Q_K is isomorphic to B, but the components of each pair are swapped: We write $(s, q) \in Q_K$, but $(q, s) \in B$. This deliberate choice provides a visual cue for distinguishing grammar states from countable events.

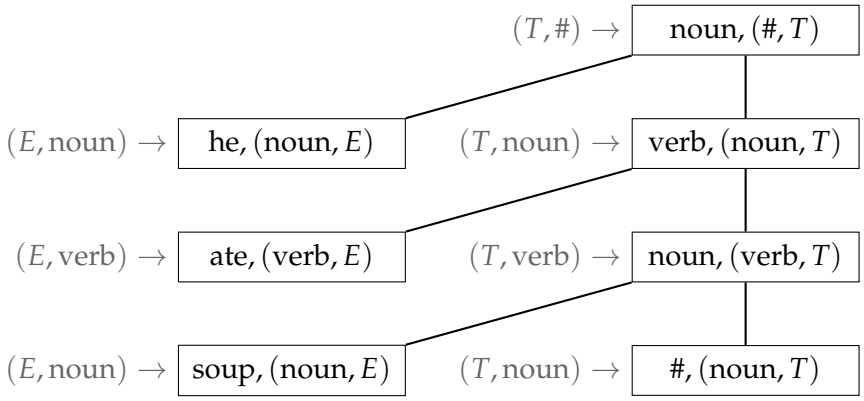


Figure 3.1: Example for a hidden information $y \in Y_{\not\downarrow} \subseteq T_C$ corresponding to the observation $x =$ "He ate soup." and the state sequence "noun-verb-noun". The label to the left of each node shows which grammar state produces this particular subtree when y is generated by K.

and R_K contains the following rules:

$$\begin{aligned}
&(T,q) \to \big(q',(q,T)\big)\big((E,q'),(T,q')\big) && \forall q \in Q_{\#} \text{ and } q' \in Q,\\
&(T,q) \to \big(\#,(q,T)\big) && \forall q \in Q_{\#},\\
&(E,q) \to \big(v,(q,E)\big) && \forall q \in Q \text{ and } v \in V.
\end{aligned}$$

The grammar K is an RTG since each rule produces as much subtrees as the rank of its label (a,b). The IO information requires that K have certain properties: First, it must be unambiguous. Because each $c \in C$ is produced by at most one rule from R_K, K is deterministic and, hence, unambiguous (see page 15). Furthermore, we need to show the following lemma.

Lemma 3.5. For every $\omega \in \Omega$, the PRTG (K, p'_ω) is proper, where

$$p'_\omega(\rho) := \begin{cases} q_\omega(a|b) & \text{if } \rho = \big(q \to (a,b)(q_1,\dots,q_k)\big) \text{ in } R_K,\\ 0 & \text{if } \rho \notin R_K. \end{cases}$$

Proof. The restriction of p'_ω to R_K, i. e., that $p'_\omega(\rho) = 0$ for all $\rho \notin R_K$, follows directly from the definition above. It remains to be shown that $p'_\omega \in \mathcal{M}(Q_K^* \times C | Q_K)$. For any $q \in Q_{\#}$, we have

$$\begin{aligned}
&\sum_{q_1\cdots q_k \in Q^*,(a,b)\in C} \underbrace{p'_\omega\Big((q_1\cdots q_k,(a,b)\,\Big|\,(T,q)\Big)}_{= p'_\omega((T,q)\to(a,b)(q_1,\dots,q_k))}\\
&= p'_\omega\Big((T,q) \to \big(\#,(q,T)\big)\Big) + \sum_{q'\in Q} p'_\omega\Big((T,q) \to \big(q',(q,T)\big)\big((E,q'),(T,q')\big)\Big)\\
&= q_\omega\big(\#,(q,T)\big) + \sum_{q'\in Q} q_\omega\big(q',(q,T)\big) = t(\#|q) + \sum_{q'\in Q} t(q'|q) = \sum_{q'\in Q_{\#}} t(q'|q) = 1
\end{aligned}$$

because $t \in \mathcal{M}(Q_\# | Q_\#)$. Analogously, for any $q \in Q$, it follows from $e \in \mathcal{M}(V|Q)$ that

$$\sum_{q_1 \cdots q_k \in Q^*, (a,b) \in C} p'_\omega\big((E,q) \to (a,b)(q_1,\ldots,q_k)\big) = \sum_{v \in V} p'_\omega\Big((E,q) \to \big(v,(q,E)\big)\Big)$$
$$= \sum_{v \in V} q_\omega\big(v,(q,E)\big) = \sum_{v \in V} e(v|q) = 1. \qquad \square$$

Using K, we can define the set Y of hidden information as

$$Y := [\![K]\!] \cup \{\bot\}.$$

Furthermore, we introduce a mapping $\pi_X : T_C \to V^*$ that reads the generated sentence from a tree $t \in T_C$ (see Figure 3.2), i. e.,

$$\pi_X(t) := \begin{cases} \varepsilon & \text{if } t = \big(\#,(q,T)\big) \text{ or } t = \big(\#,(\#,E)\big), \\ v & \text{if } t = \big(v,(q,E)\big), \\ \pi_X(t_1)\pi_X(t_2) & \text{if } t = \big(q',(q,T)\big)(t_1,t_2). \end{cases}$$

From π_X, we derive the mapping $\pi_1 : Y_{\not\perp} \to X_{\not\perp}$ that the IO information requires by restricting the domain of π_X to $Y_{\not\perp}$[5], i. e.,

$$\pi_1(y) := \pi_X(y) \quad \forall y \in Y_{\not\perp}.$$

Finally, the IO information requires a mapping $H : X_{\not\perp} \to \mathcal{R}(C)$ such that $H(x)$ generates all y with $\pi_1(y) = x$. We define $H(x)$ by modifying the states and rules of K such that only the sentence x can be generated.

Definition 3.6. Let $x \in V^*$ be a sentence. We denote by $\mathrm{suff}(x)$ its *set of suffixes*, including x itself, but excluding the empty sentence, i. e.,

$$\mathrm{suff}(x) := \begin{cases} \emptyset & \text{if } x = \varepsilon, \\ \{x\} \cup \mathrm{suff}(x') & \text{if } x = vx' \text{ with } v \in V \text{ and } x' \in V^*. \end{cases} \qquad \square$$

Definition 3.7. The operators $\mathrm{head}\colon V^+ \to V$ and $\mathrm{tail}\colon V^+ \to V^*$ decompose a nonempty sentence into its leading word and the rest of the sentence, i. e., for any $x = v_1 \cdots v_n$ in V^+,

$$\mathrm{head}(x) := v_1 \quad \text{and} \quad \mathrm{tail}(x) := v_2 \cdots v_n. \qquad \square$$

For example, for $V = \{a,b,c\}$ and $v = bbcb$, we have

$$\mathrm{suff}(bbcb) = \{bbcb, bcb, cb, b\}, \qquad \mathrm{head}(bbcb) = b, \qquad \mathrm{tail}(bbcb) = bcb.$$

[5] We cannot use the recursive definition of π_X to define π_1 directly, since the recursion goes over arguments from T_C that are not all in $Y_{\not\perp}$.

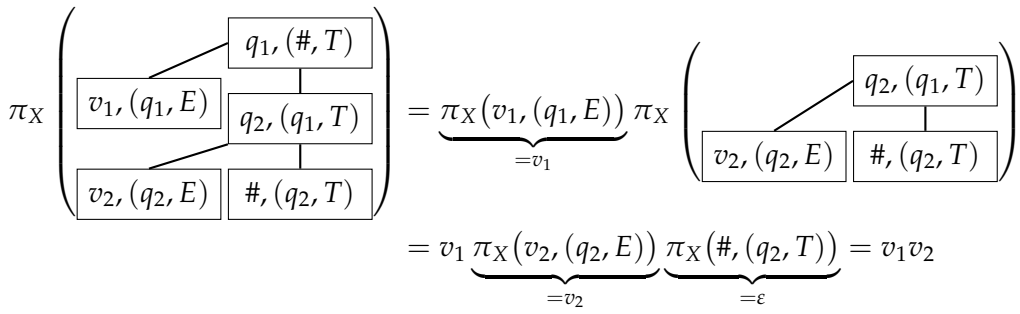


Figure 3.2: Example for how π_X can be used to read the generated sentence from a tree $y \in Y_{\not\perp}$, where the HMM has states $Q = \{q_1, q_2, \ldots\}$ and words $V = \{v_1, v_2, \ldots\}$.

Using these operators, we define, for any $x \in X_{\not\perp} = V^*$,

$$H(x) := \big\{Q_x, (T, \#, x), R_x\big\} \quad \text{where} \quad \begin{aligned} Q_x := {} & \{T\} \times Q_\# \times \big(\text{suff}(x) \cup \{\varepsilon\}\big) \\ & \cup \{E\} \times Q \times \text{suff}(x) \end{aligned}$$

and R_x contains the following rules:

$$\begin{aligned}
&(T, q, x') \to \big(q', (q, T)\big)\big((E, q', x'), (T, q', \text{tail}(x'))\big) && \forall q \in Q_\#, q' \in Q, \text{ and } x' \in \text{suff}(x), \\
&(T, q, \varepsilon) \to \big(\#, (q, T)\big) && \forall q \in Q_\#, \\
&(E, q, x') \to \big(\text{head}(x'), (q, E)\big) && \forall q \in Q \text{ and } x' \in \text{suff}(x).
\end{aligned}$$

A grammar state $q \in Q_x$ tracks both the state from $Q_\#$ that the HMM is currently in, and the suffix of x that has not yet been generated yet. Emission rules are restricted such that only the leading word from that suffix can be generated.
For the empty sentence $x = \varepsilon$, the initial grammar state $(T, \#, \varepsilon)$ can only be expanded using a rule of the second kind without producing any subsequent states. We therefore have

$$[\![H(\varepsilon)]\!] = \Big\{\big(\#, (\#, T)\big)\Big\}.$$

Using the same reasoning as above for K, we see that each $H(x)$ is deterministic and, therefore, unambiguous. For (q, π_1, K, H) to be an IO information, it remains to be shown that

$$\forall x \in X_{\not\perp} \colon \ [\![H(x)]\!] = \pi_1^{-1}(x).$$

We decompose the proof for this equivalence into two lemmas.

Lemma 3.8. With the definitions given above,

$$\forall x \in X_{\not\perp} \colon \ \forall y \in [\![H(x)]\!] \colon \ \pi_1(y) = x.$$

Lemma 3.9. With the definitions given above,

$$[\![K]\!] = \bigcup_{x \in X_{\not\perp}} [\![H(x)]\!].$$

The proofs for these lemmas can be found in Appendix A. For each $x \in X_{\not\perp}$, we then have

$$\pi_1^{-1}(x) = \{y \in \underbrace{Y_{\not\perp}}_{=[\![K]\!]} \mid \pi_1(y) = x\} = \bigcup_{x' \in X_{\not\perp}} \{y \in [\![H(x')]\!] \mid \pi_1(y) = x\}$$

by Lemma 3.9. Because of Lemma 3.8, only $x' = x$ contributes a nonempty set to this union, yielding

$$\pi_1^{-1}(x) = \{y \in [\![H(x)]\!] \mid \pi_1(y) = x\} = [\![H(x)]\!]$$

as required by the definition of the IO information. By all the considerations in this subsection, the tuple $\mu_{\mathcal{H}} := (q, \pi_1, K, H)$ therefore fulfils Definition 2.35 and is, in fact, an IO information.

3.3.3 Complete-data corpus

For the remainder of this chapter, let c be a V^*-corpus, $\omega = (t, e)$ in Ω, and $\mu := \mu_{\mathcal{H}}$. Our goal is to instantiate the inside-outside step mapping $(\!|\mu|\!)_{\mathrm{io}}$. The E-step of $(\!|\mu|\!)_{\mathrm{io}}$ involves the computation of the complete-data corpus $c\langle\omega, \mu\rangle$ according to

$$c\langle\omega, \mu\rangle(a, b) = \sum_{x \in X_{\not\perp} = V^*} c(x) \cdot \chi_{\omega,x}(a, b) \tag{3.14}$$

for any $(a, b) \in C$. We therefore need to derive expressions for all such $\chi_{\omega,x}(a, b)$ first. Following the definition of $\chi_{\omega,x}(a, b)$ in (2.3) on page 18, we have, for every $(a, b) \in C$ and $x \in V^*$,

$$\chi_{\omega,x}(a, b) = \beta_x\big((T, \#, x)\big)^{-1} \cdot \sum_{(q \to (a,b)(q_1,\ldots,q_k)) \in R_x} \alpha_x(q) \cdot q_\omega(a|b) \cdot \beta_x(q_1) \cdots \beta_x(q_k). \tag{3.15}$$

The inside and outside weights are defined by (2.1) and (2.2), i. e., for any $x \in V^*$ and $q \in Q_x$,

$$\beta_x(q) = \sum_{(q \to (a,b)(q_1,\ldots,q_k)) \in R_x} q_\omega(a|b) \cdot \beta_x(q_1) \cdots \beta_x(q_k), \tag{3.16}$$

$$\alpha_x(q) = \delta^q_{(T,\#,x)} + \sum_{(q' \to (a,b)(q_1,\ldots,q_k)) \in R_x,\, m \in \{1,\ldots,k\}:\, q_m = q} \alpha_x(q') \cdot q_\omega(a|b) \cdot \prod_{l \in \{1,\ldots,k\}:\, l \neq m} \beta_x(q_l). \tag{3.17}$$

In computing the inside weights, outside weights, and complete-data corpus contributions, we start with the simple case of $x = \varepsilon$. The RTG $H(\varepsilon)$ has only one rule,

$$R_x = \Big\{(T, \#, \varepsilon) \to \big(\#, (\#, T)\big)\Big\}.$$

Therefore, all inside weights $\beta_\varepsilon(q)$ vanish, except for

$$\beta_\varepsilon\big((T,\#,\varepsilon)\big) = q_\omega\big(\# \,\big|\, (\#,T)\big) = t(\#|\#).$$

Furthermore, because no rule produces any further grammar states, all outside weights $\alpha_\varepsilon(q)$ vanish, except for $\alpha_\varepsilon\big((T,\#,\varepsilon)\big) = 1$ since $(T,\#,\varepsilon)$ is the initial state of $H(\varepsilon)$. We therefore have

$$\chi_{\omega,\varepsilon}\big(\#,(\#,T)\big) = \underbrace{\beta_x\big((T,\#,\varepsilon)\big)^{-1}}_{=t(\#|\#)^{-1}} \cdot \underbrace{\alpha_x\big((T,\#,\varepsilon)\big)}_{=1} \cdot \underbrace{q_\omega\big(\# \,\big|\, (\#,T)\big)}_{=t(\#|\#)} = 1$$

and all other $\chi_{\omega,\varepsilon}(a,b)$ vanish. The grammar state $(T,\#,\varepsilon)$ is unreachable in all other $H(x)$, i. e., it is neither the initial state nor produced by any rule $\rho \in R_x$. We therefore have $\alpha_x(T,\#,\varepsilon) = 0$ and thus $\chi_{\omega,x}\big(\#,(\#,T)\big) = 0$ for $x \neq \varepsilon$, from which follows that

$$c\langle\omega,\mu\rangle\big(\#,(\#,T)\big) = c(\varepsilon) \cdot \underbrace{\chi_{\omega,\varepsilon}\big(\#,(\#,T)\big)}_{=1} + \sum_{x\neq\varepsilon} c(x) \cdot \underbrace{\chi_{\omega,x}\big(\#,(\#,T)\big)}_{=0} = c(\varepsilon).$$

After having considered the case that $x = \varepsilon$, it remains to calculate the inside weights, outside weights, and complete-data corpus contributions for all $x = x_1 \cdots x_n$ in V^+. We begin by noting that any $x' \in \mathrm{suff}(x)$ has the form $x' = x_i \cdots x_n$ for some $i \in \{1,\ldots,n\}$, and we have

$$\mathrm{head}(x_i \cdots x_n) = x_i \quad \text{and} \quad \mathrm{tail}(x_i \cdots x_n) = x_{i+1} \cdots x_n,$$

if we define $x_{n+1} \cdots x_n := \varepsilon$ as a corner case. We are going to use this representation of x' multiple times in the following sections.

3.3.4 Inside weights

We start with emission states (E,q,x'). For any $q \in Q$ and $x' \in \mathrm{suff}(x)$, we initially have

$$\beta_x\big((E,q,x')\big) = q_\omega\big(\mathrm{head}(x') \,\big|\, (q,E)\big) = e\big(\mathrm{head}(x') \,\big|\, q\big).$$

Replacing x' by $x_i \cdots x_n$ results in

$$\beta_x\big((E,q,x_i \cdots x_n)\big) = e(x_i|q)$$

for any $i \in \{1,\ldots,n\}$. For transmission states (T,q,x'), (3.16) leads to

$$\begin{aligned}
\beta_x\big((T,q,\varepsilon)\big) &= q_\omega\big(\# \,\big|\, (q,T)\big) = t(\#|q), \\
\beta_x\big((T,q,x')\big) &= \sum_{q'\in Q} \underbrace{q_\omega\big(q' \,\big|\, (q,T)\big)}_{=t(q'|q)} \cdot \underbrace{\beta_x\big((E,q',x')\big)}_{=e(\mathrm{head}(x')|q')} \cdot \beta_x\Big(\big(T,q',\mathrm{tail}(x')\big)\Big)
\end{aligned}$$

for all $q \in Q_\#$ and $x' \in \mathrm{suff}(x)$. Again, we insert $x' = x_i \cdots x_n$, and obtain

$$\begin{aligned}\beta_x\big((T, q, x_{n+1} \cdots x_n)\big) &= t(\#|q),\\ \beta_x\big((T, q, x_i \cdots x_n)\big) &= \sum_{q' \in Q} t(q'|q) \cdot e(x_i|q') \cdot \beta_x\big((T, q', x_{i+1} \cdots x_n)\big)\end{aligned}$$

for all $q \in Q_\#$ and $i \in \{1, \ldots, n\}$. Comparison of this set of equations with (3.7) on page 24 shows that, for all $q \in Q$ (specifically, $q \neq \#$) and $i \in \{2, \ldots, n+1\}$ (specifically, $i \neq 1$), the inside weights recover the backward weights, i. e.,

$$\beta_x\big((T, q, x_i \cdots x_n)\big) = S_x(i-1, q).$$

It remains to consider grammar states (T, q, x') where either $q = \#$ or $x' = x$. Only one such state is reachable, the initial grammar state $(T, \#, x)$ with

$$\beta_x\big((T, \#, x)\big) = \beta_x\big((T, \#, x_1 \cdots x_n)\big) = \sum_{q' \in Q} t(q'|q) \cdot e(x_1|q') \cdot \underbrace{\beta_x\big((T, q', x_2 \cdots x_n)\big)}_{= S_x(1,q)} = P(x)$$

because of (3.6). We technically also have nonzero inside weights for grammar states (T, q, x) with $q \in Q$, and grammar states $(T, \#, x')$ with $x' \in \mathrm{suff}(x) \setminus \{x\}$. But those grammar states are not reachable, which is why their exact inside weights bear no relevance for the following.

3.3.5 Outside weights

Whereas we started with emission states (E, q, x') in the previous section, it will now prove useful to consider the transmission states (T, q, x') first. We begin by noting that

$$\alpha_x\big((T, \#, x)\big) = 1$$

for the initial grammar state, since it is not produced by any rule. By expanding (3.17) for $H(x)$ with $x = x_1 \cdots x_n$ in V^+, we initially obtain

$$\alpha_x\Big(\big(T, q, \mathrm{tail}(x')\big)\Big) = \sum_{q' \in Q_\#} \alpha_x\big((T, q', x')\big) \cdot \underbrace{q_\omega\big(q \,\big|\, (q', T)\big)}_{= t(q|q')} \cdot \underbrace{\beta_x\big((E, q, x')\big)}_{= e(\mathrm{head}(x')|q)}$$

for all $q \in Q$ and $x' \in \mathrm{suff}(x)$. Replacing x' by $x_i \cdots x_n$ yields

$$\alpha_x\big((T, q, x_{i+1} \cdots x_n)\big) = \sum_{q' \in Q_\#} \alpha_x\big((T, q', x_i \cdots x_n)\big) \cdot t(q|q') \cdot e(x_i|q)$$

for all $q \in Q$ and $i \in \{1, \ldots, n\}$. The grammar states (T, q, x) with $q \neq \#$ are not produced by any rule in R_x and therefore unreachable, which means that their inside weights vanish. For $i = 1$,

we therefore have

$$\alpha_x\big((T,q,x_2\cdots x_n)\big) = \sum_{q'\in Q_\#} \underbrace{\alpha_x\big((T,q',x_1\cdots x_n)\big)}_{1 \text{ for } q'=\#,\, 0 \text{ otherwise}} \cdot t(q|q') \cdot e(x_1|q) = t(q|\#)\cdot e(x_1|q) = T_x(1,q).$$

For $x' \neq x$, we can likewise observe that the grammar states $(T,\#,x')$ are unreachable, hence in $\alpha_x\big((T,q,x_{i+1}\cdots x_n)\big)$ for $i \in \{2,\dots,n\}$, it suffices to sum over $q' \in Q$ only. We therefore have

$$\begin{aligned}\alpha_x\big((T,q,x_2\cdots x_n)\big) &= T_x(1,q),\\ \alpha_x\big((T,q,x_{i+1}\cdots x_n)\big) &= \sum_{q'\in Q} \alpha_x\big((T,q',x_i\cdots x_n)\big)\cdot t(q|q')\cdot e(x_i|q)\end{aligned}$$

for any $q \in Q$ and $i \in \{2,\dots,n\}$. Comparison of this set of equations with (3.3) and (3.4) shows that the outside weights recover the inside weights, i. e.,

$$\alpha_x\big((T,q,x_{i+1}\cdots x_n)\big) = T_x(i,q)$$

for all $i \in \{1,\dots,n\}$ and $q \in Q$.
It remains to consider emission states (E,q,x'). For those states, (3.17) leads to

$$\alpha_x\big((E,q,x')\big) = \sum_{q'\in Q_\#} \alpha_x\big((T,q',x')\big)\cdot \underbrace{q_\omega\big(q \,|\, (q',T)\big)}_{=t(q|q')} \cdot \beta_x\Big(\big(T,q,\mathrm{tail}(x')\big)\Big)$$

for any $q \in Q$ and $x' \in \mathrm{suff}(x)$ or, in other words,

$$\alpha_x\big((E,q,x_i\cdots x_n)\big) = \sum_{q'\in Q_\#} \alpha_x\big((T,q',x_i\cdots x_n)\big)\cdot t(q|q')\cdot \underbrace{\beta_x\Big(\big(T,q,x_{i+1}\cdots x_n\big)\Big)}_{=S_x(i,q)}.$$

for $q \in Q$ and $i \in \{1,\dots,n\}$. We need to consider two subcases. For $i = 1$, we obtain

$$\begin{aligned}\alpha_x\big((E,q,x_1\cdots x_n)\big) &= \sum_{q'\in Q_\#} \underbrace{\alpha_x\big((T,q',x_1\cdots x_n)\big)}_{1 \text{ for } q'=\#,\, 0 \text{ otherwise}} \cdot t(q|q')\cdot S_x(1,q)\\ &= t(q|\#)\cdot S_x(1,q) = \frac{T_x(1,q)\cdot S_x(1,q)}{e(x_1|q)},\end{aligned}$$

and for $i \in \{2,\dots,n\}$, we likewise have

$$\begin{aligned}\alpha_x\big((E,q,x_i\cdots x_n)\big) &= \sum_{q'\in Q_\#} \underbrace{\alpha_x\big((T,q',x_i\cdots x_n)\big)}_{0 \text{ for } q'=\#} \cdot t(q|q')\cdot S_x(i,q)\\ &= \sum_{q'\in Q} T_x(i-1,q')\cdot t(q|q')\cdot S_x(i,q) = \frac{T_x(i,q)\cdot S_x(i,q)}{e(x_i|q)}.\end{aligned}$$

3.3.6 Complete-data corpus (cont.)

In the previous two sections, we have derived expressions for all relevant inside and outside weights[6] of $H(x)$ for any $x \in X_{\not\perp} = V^*$. In this section, we will compute the complete-data corpus by expanding the equations (3.14) and (3.15). The results of this computation will be used in the following section to instantiate the inside-outside step mapping $(\!|\mu|\!)_{\mathrm{io}}$ for $\mu = \mu_{\mathcal{H}}$. In section 3.3.3, we already considered the case that $x = \varepsilon$, and obtained

$$c\langle\omega,\mu\rangle\big(\#,(\#,T)\big) = c(\varepsilon).$$

Therefore, as in the previous sections, it remains to consider $x = x_1 \cdots x_n$ in V^+. For emission events $\big(v,(q,E)\big)$ with $v \in V$ and $q \in Q$, (3.15) leads to

$$\begin{aligned}\chi_{\omega,x}\big(v,(q,E)\big) &= P(x)^{-1} \cdot \sum_{i\in\{1,\ldots,n\}:\, x_i=v} \underbrace{\alpha_x\big((E,q,x_i\cdots x_n)\big)}_{=T_x(i,q)\cdot S_x(i,q)\cdot e(x_i|q)^{-1}} \cdot \underbrace{q_\omega\big(v\,\big|\,(q,E)\big)}_{=e(v|q)} \\ &= \sum_{i\in\{1,\ldots,n\}:\, x_i=v} \frac{T_x(i,q)\cdot S_x(i,q)}{P(x)} = \sum_{i\in\{1,\ldots,n\}:\, x_i=v} R_x(i,q).\end{aligned}$$

For transition events into and out of the # state, we obtain, for every $q \in Q$,

$$\chi_{\omega,x}\big(\#,(q,T)\big) = P(x)^{-1} \cdot \underbrace{\alpha_x\big((T,q,\varepsilon)\big)}_{=T_x(n,q)} \cdot \underbrace{q_\omega\big(\#\,\big|\,(q,T)\big)}_{=t(\#|q)=S_x(n,q)} = R_x(n,q),$$

$$\begin{aligned}\chi_{\omega,x}\big(q,(\#,T)\big) &= P(x)^{-1} \cdot \sum_{x'\in\mathrm{suff}(x)} \underbrace{\alpha_x\big((T,\#,x')\big)}_{=0 \text{ for } x'\neq x} \cdot \underbrace{q_\omega\big(q\,\big|\,(\#,T)\big)}_{=t(q|\#)} \cdot \beta_x\big((E,q,x')\big) \cdot \beta_x\Big(\big(T,q,\mathrm{tail}(x')\big)\Big) \\ &= P(x)^{-1} \cdot \underbrace{\alpha_x\big((T,\#,x)\big)}_{=1} \cdot t(q|\#) \cdot \underbrace{\beta_x\big((E,q,x)\big)}_{=e(x_1|q)} \cdot \underbrace{\beta_x\Big(\big(T,q,\mathrm{tail}(x)\big)\Big)}_{=S_x(1,q)} \\ &= P(x)^{-1} \cdot \underbrace{t(q|\#)\cdot e(x_1|q)}_{=T_x(1,q)} \cdot S_x(1,q) = R_x(1,q).\end{aligned}$$

For transition events from $q \in Q$ to $q' \in Q$, we initially get

$$\chi_{\omega,x}\big(q',(q,T)\big) = P(x)^{-1} \cdot \sum_{x'\in\mathrm{suff}(x)} \alpha_x\big((T,q,x')\big) \cdot \underbrace{q_\omega\big(q'\,\big|\,(q,T)\big)}_{=t(q'|q)} \cdot \underbrace{\beta_x\big((E,q',x')\big)}_{=e(\mathrm{head}(x')|q')} \cdot \beta_x\Big(\big(T,q',\mathrm{tail}(x')\big)\Big).$$

[6] All these inside and outside weights are assembled in the table at the end of Appendix B for quick referencing.

To simplify this expression, we replace x' by $x_i \cdots x_n$ once more.

$$\begin{aligned}
\chi_{\omega,x}\big(q',(q,T)\big) &= P(x)^{-1} \cdot \sum_{i=1}^{n} \underbrace{\alpha_x\big((T,q,x_i\cdots x_n)\big)}_{=0 \text{ for } i=1} \cdot t(q'|q) \cdot e(x_i|q') \cdot \underbrace{\beta_x\Big(\big(T,q',x_{i+1}\cdots x_n\big)\Big)}_{=S_x(i,q')} \\
&= P(x)^{-1} \cdot \sum_{i=2}^{n} \underbrace{\alpha_x\big((T,q,x_i\cdots x_n)\big)}_{=T_x(i-1,q)} \cdot t(q'|q) \cdot e(x_i|q') \cdot S_x(i,q') \\
&= \sum_{i=2}^{n} U_x(i-1,q,q') = \sum_{i=1}^{n-1} U_x(i,q,q').
\end{aligned}$$

When we put these expressions into the definition of $c\langle\omega,\mu\rangle$, we initially sum over over all $x \in V^+ = X_{\not\perp}$. However, we can exclude $x = \varepsilon$ from the sum since, as was discussed in section 3.3.3, all $\chi_{\omega,\varepsilon}(a,b)$ vanish except for $(a,b) = \big(\#,(\#,T)\big)$. Furthermore, the factor $c(x)$ makes every contribution from outside $\operatorname{supp}(c)$ vanish. It therefore suffices to sum over $\operatorname{supp}(c) \setminus \{\varepsilon\}$ only, which we shall denote by X_c. For any $q,q' \in Q$ and $v \in V$, we finally have

$$\begin{aligned}
&c\langle\omega,\mu\rangle\big(\#,(\#,T)\big) = c(\varepsilon), \qquad (3.18)\\
&c\langle\omega,\mu\rangle\big(\#,(q,T)\big) = \sum_{x\in X_c} c(x)\cdot R_x\big(|x|,q\big), \quad c\langle\omega,\mu\rangle\big(q',(q,T)\big) = \sum_{x\in X_c} c(x)\cdot \sum_{i=1}^{|x|-1} U_x(i,q,q'),\\
&c\langle\omega,\mu\rangle\big(q,(\#,T)\big) = \sum_{x\in X_c} c(x)\cdot R_x(1,q), \qquad c\langle\omega,\mu\rangle\big(v,(q,E)\big) = \sum_{x\in X_c} c(x)\cdot \sum_{\substack{i=1\\ x_i=v}}^{|x|} R_x(i,q).
\end{aligned}$$

These equations are similar to (3.10) through (3.13) aside from variable names and term ordering, with one exception: Algorithm 3.1 operates on a V^+-corpus rather than a V^*-corpus, so the empty sentence ε is not accounted for. Given the V^+-corpus h, we can define the equivalent V^*-corpus c_h by

$$c_h(x) := \begin{cases} h(x) & \text{if } x \in V^+,\\ 0 & \text{if } x = \varepsilon. \end{cases}$$

By using c_h instead of c in the equations (3.18), and by noting that

$$X_{c_h} = \operatorname{supp}(c_h) \setminus \{\varepsilon\} = \operatorname{supp}(h) \setminus \{\varepsilon\} = \operatorname{supp}(h),$$

we find that (3.10)–(3.13) are equivalent to (3.18). Therefore, after line 22 of Algorithm 3.1,

$$\operatorname{count}_{\text{tr}}(q',q) = c_h\langle\omega,\mu\rangle\big(q',(q,T)\big) \quad \text{and} \quad \operatorname{count}_{\text{em}}(v,q) = c_h\langle\omega,\mu\rangle\big(v,(q,E)\big) \qquad (3.19)$$

for all $(q',q) \in Q_\# \times Q_\#$ and $(v,q) \in V \times Q$, respectively. This equivalence also shows how the Baum-Welch algorithm can be extended to allow for a V^*-corpus instead of a V^+-corpus. The only difference is that $\operatorname{count}_{\text{tr}}(\#,\#)$ has to be set to $c(\varepsilon)$ between lines 4 and 22.

3.3.7 Step mapping

Having calculated the complete-data corpus, we can apply the inside-outside step mapping,

$$(\!|\mu|\!)_{\mathrm{io}}(\omega) := \mathrm{cmle}_q\big(c\langle\omega,\mu\rangle\big) = \operatorname*{argmax}_{\omega'} q_{\omega'}\big(c\langle\omega,\mu\rangle\big),$$

then use it to choose a new model parameter $\hat{\omega}$ from this set. According to Lemma 2.17 (see page 10), we can choose $\hat{\omega}$ such that $q_{\hat{\omega}} = \widetilde{c\langle\omega,\mu\rangle}$ if such a $\hat{\omega}$ exists.

Lemma 3.10. For any $p \in \mathcal{M}_C(A|B)$, there exists $\omega = (t,e)$ in Ω such that $q_\omega = p$.

Proof. The $\omega = (t,e)$ in question is given by

$$\begin{aligned} t(q'|q) &:= p\big(q' \,\big|\, (q,T)\big) && \forall q,q' \in Q_\#, \\ e(v|q) &:= p\big(v \,\big|\, (q,E)\big) && \forall v \in V, q \in Q. \end{aligned}$$

We need to show that $\omega \in \Omega$ and $q_\omega = p$. Since $p \in \mathcal{M}(A|B)$, it holds for any $q \in Q_\#$ that

$$\sum_{q' \in Q_\#} t(q'|q) = \sum_{q' \in Q_\#} p\big(q' \,\big|\, (q,T)\big) = 1.$$

Therefore, $t \in \mathcal{M}(Q_\#|Q_\#)$. By an analogous argument, we have $e \in \mathcal{M}(V|Q)$, from which follows $\omega \in \Omega$. Moreover, for any $(a,b) \in A \times B$, we have

$$q_\omega(a|b) = \begin{cases} p(a|b) & \text{if } (a,b) \in C, \\ 0 & \text{otherwise} \end{cases}$$

because of how q is defined. This is equivalent to saying that $q_\omega = p$ since $p \in \mathcal{M}_C(A|B)$ is restricted to C. □

Using this lemma, we obtain the final and principal result of this thesis.

Theorem 3.11. Algorithm 3.1 (see page 25) is an instance of Algorithm 2.1 (see page 7).

Proof. Since $c\langle\omega,\mu\rangle$ is zero-valued anywhere outside C, we have $\widetilde{c\langle\omega,\mu\rangle} \in \mathcal{M}_C(A|B)$. We can therefore apply the construction for (t,e) from this proof to obtain $\hat{\omega} = (\hat{t},\hat{e})$ which is in $(\!|\mu|\!)_{\mathrm{io}}(\omega)$.

$$\begin{aligned} \hat{t}(q'|q) &:= \widetilde{c\langle\omega,\mu\rangle}\big(q' \,\big|\, (q,T)\big) && \forall q,q' \in Q_\#, \\ \hat{e}(v|q) &:= \widetilde{c\langle\omega,\mu\rangle}\big(v \,\big|\, (q,E)\big) && \forall v \in V, q \in Q. \end{aligned}$$

In the case of Algorithm 3.1, we have $c = c_h$, and the equations (3.19) apply. We therefore have, for every $q, q' \in Q_\#$ and $v \in V$,

$$\hat{t}(q'|q) = \widetilde{c\langle\omega,\mu\rangle}\big(q' \,\big|\, (q,T)\big) = \frac{c\langle\omega,\mu\rangle\big(q' \,\big|\, (q,T)\big)}{\sum_{q''\in Q_\#} c\langle\omega,\mu\rangle\big(q'' \,\big|\, (q,T)\big)} \overset{(3.19)}{=} \frac{\mathrm{count}_{\mathrm{tr}}(q',q)}{\sum_{q''\in Q_\#} \mathrm{count}_{\mathrm{tr}}(q'',q)},$$

$$\hat{e}(v|q) = \widetilde{c\langle\omega,\mu\rangle}\big(v \,\big|\, (q,E)\big) = \frac{c\langle\omega,\mu\rangle\big(v \,\big|\, (q,E)\big)}{\sum_{v'\in V} c\langle\omega,\mu\rangle\big(v' \,\big|\, (q,E)\big)} \overset{(3.19)}{=} \frac{\mathrm{count}_{\mathrm{em}}(v,q)}{\sum_{v'\in V} \mathrm{count}_{\mathrm{em}}(v',q)}.$$

Comparison with Algorithm 3.1 shows that this is exactly the same computation carried out by lines 23–28. In total, we see that each iteration of the main loop of this algorithm takes in a previous $\omega = (t, e)$ and computes $\hat{\omega} = (\hat{t}, \hat{e})$ as shown above, such that $\hat{\omega} \in (\!|\mu|\!)_{\mathrm{io}}(\omega)$. This is the same behavior that we described more abstractly in Algorithm 2.1. □

3.4 Review

The principal insights of this chapter, and this thesis overall, are twofold:

1. We have shown that any Hidden Markov model $\mathcal{H}$ can be described as an inside-outside information $\mu_{\mathcal{H}}$.

2. Using the step mapping $(\!|\mu_{\mathcal{H}}|\!)_{\mathrm{io}}$, we have instantiated the generic EM algorithm from [BSV15], and shown that the algorithm thus obtained is identical to the well-known Baum-Welch algorithm.

From this follows that all theorems regarding the generic EM algorithm, esp. concering its convergence properties, apply to the Baum-Welch algorithm as well.

Appendix

A Elided proofs from Chapter 3

This appendix contains the proofs for Lemmas 3.8 and 3.9. We imply that $\mathcal{H} = (Q, V, \#, t, e)$ is an HMM. Moreover, we imply all the definitions introduced in Section 3.3 to describe $\mathcal{H}$ in terms of the IO information $\mu_{\mathcal{H}}$. As a prerequisite for both proofs, we shall begin by writing down the languages of all the grammar states of $H(x)$ and K.

Definition A.1. Let $\mathcal{G} = (Q_{\mathcal{G}}, q_0, R_{\mathcal{G}})$ be an RTG over C. For any grammar state $q \in Q_{\mathcal{G}}$, the *language of q generated by $\mathcal{G}$* is given by

$$[\![\mathcal{G}]\!]_q := \pi_C\big(D^q(\mathcal{G}) \cap T_{R_{\mathcal{G}}}\big). \qquad \square$$

The mapping $\pi_C \colon T_{R_{\mathcal{G}}}(Q_{\mathcal{G}}) \to U_C(Q_{\mathcal{G}})$ was introduced as π_Σ on page 15. The language $[\![\mathcal{G}]\!]_q$ contains all trees from U_C that can be generated by applying the grammar's rules, starting from the state q. It especially holds that

$$[\![\mathcal{G}]\!] = [\![\mathcal{G}]\!]_{q_0}. \tag{A.1}$$

To find $D^q(\mathcal{G}) \cap T_{R_{\mathcal{G}}}$, we first observe that, according to Definition 2.28, each $d \in D^q(\mathcal{G})$ must have one of two forms,

$$d = q \quad \text{or} \quad d = \rho(d_1, \ldots, d_k),$$

where $\rho = \big(q \to c(q_1, \ldots, q_k)\big)$ in $R_{\mathcal{G}}$ such that $\mathrm{rk}(c) = k$, and each $d_i \in D^{q_i}(\mathcal{G})$. The first form, $d = q$, is not allowed in $[\![\mathcal{G}]\!]_q$ since $q \notin T_{R_{\mathcal{G}}}$. The second form is admissible, but because $d \in T_{R_{\mathcal{G}}}$, the subtrees d_i may not have positions labeled with a state $q \in Q_{\mathcal{G}}$ either. We therefore have

$$D^q(\mathcal{G}) \cap T_{R_{\mathcal{G}}} = \bigcup_{\rho = (q \to c(q_1,\ldots,q_k)) \text{ in } R_{\mathcal{G}}} \big\{\rho(d_1, \ldots, d_k) \,\big|\, d_1 \in D^{q_1}(\mathcal{G}) \cap T_{R_{\mathcal{G}}}, \ldots, d_k \in D^{q_k}(\mathcal{G}) \cap T_{R_{\mathcal{G}}}\big\},$$

from which follows

$$[\![\mathcal{G}]\!]_q = \bigcup_{\rho = (q \to c(q_1,\ldots,q_k)) \text{ in } R_{\mathcal{G}}} \big\{c(t_1, \ldots, t_k) \,\big|\, t_1 \in [\![\mathcal{G}]\!]_{q_1}, \ldots, t_k \in [\![\mathcal{G}]\!]_{q_k}\big\}, \tag{A.2}$$

when π_C is applied on both sides. We can thus describe the languages $[\![K]\!]$ and $[\![H(x)]\!]$ recursively, in terms of the languages generated from their states.

Let $x \in V^*$. Application of (A.2) to all reachable states in $H(x)$ yields

$$\begin{aligned}\llbracket H(x) \rrbracket_{(T,q,x')} &= \bigcup_{q' \in Q} \left\{ \big(q', (q,T)\big)(t_1, t_2) \,\middle|\, t_1 \in \llbracket H(x) \rrbracket_{(E,q',x')}, t_2 \in \llbracket H(x) \rrbracket_{(T,q',\mathrm{tail}(x'))} \right\}, \\ \llbracket H(x) \rrbracket_{(T,q,\varepsilon)} &= \big\{ \big(\#, (q,T)\big) \big\}\end{aligned}$$

for all $q \in Q_\#$ and $x' \in \mathrm{suff}(x)$, and

$$\llbracket H(x) \rrbracket_{(E,q,x')} = \big\{ \big(\mathrm{head}(x'), (q,E)\big) \big\}$$

for all $q \in Q$ and $x' \in \mathrm{suff}(x)$. Inserting the last equation into the one above it, we can describe $\llbracket H(x) \rrbracket = \llbracket H(x) \rrbracket_{(T,\#,x)}$ recursively by

$$\llbracket H(x) \rrbracket_{(T,q,x')} = \bigcup_{q' \in Q} \left\{ \big(q', (q,T)\big) \Big(\big(\mathrm{head}(x'), (q',E)\big), t \Big) \,\middle|\, t \in \llbracket H(x) \rrbracket_{(T,q',\mathrm{tail}(x'))} \right\}, \tag{A.3a}$$

$$\llbracket H(x) \rrbracket_{(T,q,\varepsilon)} = \big\{ \big(\#, (q,T)\big) \big\} \tag{A.3b}$$

for all $q \in Q_\#$. Analogous application of (A.2) to all states in K, followed by insertion of the expression for $\llbracket K \rrbracket_{(E,q)}$ into that of $\llbracket K \rrbracket_{(T,q)}$, results in

$$\llbracket K \rrbracket_{(T,q)} = \big\{ \big(\#, (q,T)\big) \big\} \cup \bigcup_{q' \in Q} \bigcup_{v \in V} \left\{ \big(q', (q,T)\big) \Big(\big(v, (q',E)\big), t \Big) \,\middle|\, t \in \llbracket K \rrbracket_{(T,q')} \right\} \tag{A.4}$$

for any $q \in Q_\#$. An important observation can be made from the structure of (A.3).

$$\forall x \in V^+, q \in Q\colon\ \llbracket H(x) \rrbracket_{(T,q,\mathrm{tail}(x))} = \llbracket H\big(\mathrm{tail}(x)\big) \rrbracket_{(T,q,\mathrm{tail}(x))}. \tag{A.5}$$

This equivalence holds because $Q_{\mathrm{tail}(x)}$ and $R_{\mathrm{tail}(x)}$ are strict subsets of Q_x and R_x, respectively, and furthermore, because R_x does not contain any rules not in $R_{\mathrm{tail}(x)}$ that expand grammar states from $Q_{\mathrm{tail}(x)}$. Therefore, we have

$$D^q\big(H(x)\big) = D^q\Big(H\big(\mathrm{tail}(x)\big)\Big) \quad \forall q \in Q_{\mathrm{tail}(x)},$$

from which follows (A.5).

A.1 Proof of Lemma 3.8

Lemma 3.8. With the definitions from Section 3.3,

$$\forall x \in X_{\not\perp}\colon\ \forall y \in \llbracket H(x) \rrbracket\colon\ \pi_1(y) = x.$$

Proof. Let $x \in X_{\not\perp} = V^*$. We are going to show that

$$\forall x' \in \operatorname{suff}(x) \cup \{\varepsilon\} : \ \forall q \in Q_{\#} \colon \ \pi_X\big(\llbracket H(x) \rrbracket_{(T,q,x')}\big) = \{x'\}\,. \tag{A.6}$$

by induction over the length of x'. The lemma follows from this statement because

$$\pi_1\big(\llbracket H(x) \rrbracket\big) = \pi_X\big(\llbracket H(x) \rrbracket\big) = \pi_X\big(\llbracket H(x) \rrbracket_{(T,\#,x)}\big) \overset{\text{(A.6)}}{=} \{x\}\,.$$

For the base case, let $x' = \varepsilon$. Then (A.6) holds because of

$$\pi_X\big(\llbracket H(x) \rrbracket_{(T,q,\varepsilon)}\big) \overset{\text{(A.3b)}}{=} \big\{\pi_X\big(\#, (q, T)\big)\big\} = \{\varepsilon\}$$

for any $q \in Q_{\#}$. For the induction step, let $x' \in \operatorname{suff}(x)$. Since $|\operatorname{tail}(x')| < |x'|$, it can be assumed that $\operatorname{tail}(x')$ satisfies (A.6) (*induction hypothesis*). Application of π_X to both sides of (A.3a) yields

$$\begin{aligned}
\pi_X\big(\llbracket H(x) \rrbracket_{(T,q,x')}\big) &= \bigcup_{q' \in Q} \Big\{ \pi_X\big(\operatorname{head}(x'), (q', E)\big)\, \pi_X(t) \,\Big|\, t \in \llbracket H(x) \rrbracket_{(T,q',\operatorname{tail}(x'))} \Big\} \\
&= \big\{\operatorname{head}(x')\big\} \cdot \bigcup_{q' \in Q} \Big\{ \pi_X(t) \,\Big|\, t \in \llbracket H(x) \rrbracket_{(T,q',\operatorname{tail}(x'))} \Big\} \\
&= \big\{\operatorname{head}(x')\big\} \cdot \bigcup_{q' \in Q} \underbrace{\pi_X\big(\llbracket H(x) \rrbracket_{(T,q',\operatorname{tail}(x'))}\big)}_{= \{\operatorname{tail}(x')\} \text{ by induction hypothesis}} \\
&= \big\{\operatorname{head}(x')\, \operatorname{tail}(x')\big\} = \{x'\}.
\end{aligned}$$

□

A.2 Proof of Lemma 3.9

Lemma 3.9. With the definitions from Section 3.3,

$$\llbracket K \rrbracket = \bigcup_{x \in X_{\not\perp}} \llbracket H(x) \rrbracket.$$

Proof. Let $\pi_Q \colon T_C \to Q_{\#}$ be the mapping defined by

$$\pi_Q(y) := q \text{ such that } y(\varepsilon) = \big(a, (q, b)\big).$$

That is, π_Q extracts the state q from the label $\big(a, (q, b)\big)$ of the root position of any y. As a prerequisite for the actual proof, we are going to show that

$$\forall y \in T_C \colon \ \Big(y \in \llbracket K \rrbracket_{(T,\pi_Q(y))}\Big) \text{ iff } \Big(y \in \llbracket H\big(\pi_X(y)\big) \rrbracket_{(T,\pi_Q(y),\pi_X(y))}\Big), \tag{A.7}$$

by structural induction on y. According to the definition of the ranked alphabet C, each $y \in T_C$

can have one of three forms,

$$\begin{aligned} y_1(v,q) &:= \big(v,(q,E)\big) && \text{where } v \in V \text{ and } q \in Q, \\ y_2(q) &:= \big(\#,(q,T)\big) && \text{where } q \in Q_\#, \\ y_3(q,q',y_e,y_t) &:= \big(q',(q,T)\big)(y_e,y_t) && \text{where } q \in Q_\#, q' \in Q, \text{ and } y_e, y_t \in T_C. \end{aligned}$$

The first two forms are the base cases for the structural induction on y, and the last form represents the inductive step.

Base cases

For any $y = y_1(v,q)$, we have $\pi_Q(y) = q$ and $\pi_X(y) = v$. We see that (A.7) holds because

$$\big(v,(q,E)\big) \notin [\![K]\!]_{(T,q)} \quad \text{and} \quad \big(v,(q,E)\big) \notin [\![H(v)]\!]_{(T,q,v)}.$$

For any $y = y_2(q)$, we have $\pi_Q(y) = q$ and $\pi_X(y) = \varepsilon$, and (A.7) is satisfied because

$$\big(\#,(q,T)\big) \in [\![K]\!]_{(T,q)} \quad \text{and} \quad \big(\#,(q,T)\big) \in [\![H(\varepsilon)]\!]_{(T,q,\varepsilon)}.$$

Induction step: "⇒" direction

Consider any

$$y = y_3(q,q',y_e,y_t) = \big(q',(q,T)\big)(y_e,y_t).$$

such that $y \in [\![K]\!]_{(T,q)}$, and let $x := \pi_X(y)$. Note that $\pi_Q(y) = q$. We see from (A.4) that

$$y_t \in [\![K]\!]_{(T,q')} \quad \text{and} \quad y_e = \big(v,(q',E)\big) \quad \text{where} \quad q' \in Q \text{ and } v \in V.$$

From this follows $x = \pi_X(y) = v\pi_X(y_t)$ and, therefore, $\mathrm{head}(x) = v$ and $\mathrm{tail}(x) = \pi_X(y_t)$. Since y_t is a substructure of y, it can be assumed to satisfy (A.7) (*induction hypothesis*). Since $y_t \in [\![K]\!]_{(T,q')}$, we can see from (A.4) that only $\pi_Q(y_t) = q'$ is possible. Therefore, by the induction hypothesis,

$$y_t \in [\![H\big(\pi_X(y_t)\big)]\!]_{(T,q',\pi_X(y_t))} = [\![H\big(\mathrm{tail}(x)\big)]\!]_{(T,q',\mathrm{tail}(x))}.$$

Because of (A.5), this is equivalent to saying that $y_t \in [\![H(x)]\!]_{(T,q',\mathrm{tail}(x))}$. We therefore have

$$y = \big(q',(q,T)\big)\Big(\big(\mathrm{head}(x),(q',E)\big),y_t\Big)$$

with $q \in Q_\#$, $q' \in Q$, and $y_t \in [\![H(x)]\!]_{(T,q',\mathrm{tail}(x))}$, from which follows

$$y \in [\![H(x)]\!]_{(T,q,x)}$$

because of (A.3a).

Induction step: "⇐" direction

Consider any $y = y_3(q, q', y_e, y_t)$ such that $y \in [\![H(x)]\!]_{(T,\pi_Q(y),x)}$, wherein $x := \pi_X(y)$. Since $\pi_Q(y) = q$, it follows from (A.3) that $y_e = \big(\mathrm{head}(x), (q', E)\big)$ and $y_t \in [\![H(x)]\!]_{(T,q',\mathrm{tail}(x))}$. Because of (A.5), the second statement is equivalent to

$$y_t \in [\![H\big(\mathrm{tail}(x)\big)]\!]_{(T,q',\mathrm{tail}(x))}.$$

From (A.3), we see that $\pi_Q(y_t)$ must be q'. Since y_t is a substructure of y, it can be assumed to satisfy (A.7), from which follows that $y_t \in [\![K]\!]_{(T,q')}$. According to (A.4), it therefore holds that $y \in [\![K]\!]_{(T,q)}$. The proof for (A.7) is thereby complete.

Conclusion of the proof

To finally derive Lemma 3.9, we first note that

$$\pi_Q\big([\![K]\!]_{(T,\#)}\big) = \{\#\} \quad \text{and} \quad \pi_Q\big([\![H(x)]\!]_{(T,\#,x)}\big) = \{\#\} \tag{A.8}$$

for every $x \in X_{\not\perp}$, which follows from (A.4) and (A.3), respectively. We can thus expand $[\![K]\!]$ into

$$\begin{aligned}[\![K]\!] = [\![K]\!]_{(T,\#)} &= \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![K]\!]_{(T,\#)}\big\} \\ &= \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![K]\!]_{(T,\pi_Q(y))}\big\}.\end{aligned}$$

By using (A.7), we obtain

$$\begin{aligned}[\![K]\!] &= \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![H\big(\pi_X(y)\big)]\!]_{(T,\pi_Q(y),\pi_X(y))}\big\} \\ &= \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![H\big(\pi_X(y)\big)]\!]_{(T,\#,\pi_X(y))}\big\}.\end{aligned}$$

To obtain the desired structure, we split this set into one set per yield $x = \pi_X(y)$.

$$\begin{aligned}[\![K]\!] &= \bigcup_{x \in X_{\not\perp}} \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![H\big(\pi_X(y)\big)]\!]_{(T,\#,\pi_X(y))}, x = \pi_X(y)\big\} \\ &= \bigcup_{x \in X_{\not\perp}} \big\{y \in \pi_Q^{-1}(\#) \,\big|\, y \in [\![H(x)]\!]_{(T,\#,x)}, x = \pi_X(y)\big\}.\end{aligned}$$

Because of (A.8), each $H(x)_{(T,\#,x)}$ is a subset of $\pi_Q^{-1}(\#)$. We can therefore simplify this to

$$[\![K]\!] = \bigcup_{x \in X_{\not\perp}} \big\{y \in [\![H(x)]\!]_{(T,\#,x)} \,\big|\, x = \pi_X(y)\big\} = \bigcup_{x \in X_{\not\perp}} \big\{y \in [\![H(x)]\!] \,\big|\, x = \pi_X(y)\big\},$$

which is equivalent to Lemma 3.9 because of Lemma 3.8. □

B Formulary for Chapter 3

This appendix assembles most of the formulas derived in Chapter 3. It serves as a quick reference for the observant reader who wants to follow the derivations without having to flip pages excessively. For most equations, the tag near the right margin refers to the page where the equation was first introduced.

We imply that $\mathcal{H} = (Q, V, \#, t, e)$ is an HMM. Let $v \in V^*$ be an observation.

$$P(v = v_1 \cdots v_n) = \sum_{q_1,\ldots,q_n \in Q} t(q_1|\#) \cdot e(v_1|q_1) \cdot \prod_{i=2}^{n} \big[t(q_i|q_{i-1}) \cdot e(v_i|q_i)\big] \cdot t(\#|q_n), \tag{3.1, p. 22}$$

$$P(v = \varepsilon) = t(\#|\#). \tag{3.2, p. 22}$$

In terms of the forward weight $T_v(i,q)$ for $v = v_1 \cdots v_n$ in V^+, $i \in \{1,\ldots,n\}$ and $q \in Q$:

$$P(v) = \sum_{q \in Q} T_v(n,q) \cdot t(\#|q), \tag{3.5, p. 23}$$

$$T_v(i,q) = \begin{cases} e(v_1|q) \cdot t(q|\#) & \text{if } i = 1, \\ e(v_i|q) \cdot \sum_{q' \in Q} T_v(i-1,q') \cdot t(q|q') & \text{if } i \in \{2,\ldots,n\}. \end{cases} \tag{3.4, p. 23}$$

In terms of the backward weight $S_v(i,q)$ for $v = v_1 \cdots v_n$ in V^+, $i \in \{1,\ldots,n\}$ and $q \in Q$:

$$P(v) = \sum_{q \in Q} t(q|\#) \cdot e(v_1|q) \cdot S_v(1,q), \tag{3.6, p. 24}$$

$$S_v(i,q) = \begin{cases} t(\#|q) & \text{if } i = n, \\ \sum_{q' \in Q} t(q'|q) \cdot e(v_{i+1}|q') \cdot S_v(i+1,q') & \text{if } i \in \{1,\ldots,n-1\}. \end{cases} \tag{3.7, p. 24}$$

Expressions from the Baum-Welch algorithm, for $v = v_1 \cdots v_n$ in V^+, $i \in \{1,\ldots,n\}$ and $q, q' \in Q$:

$$R_v(i,q) = \frac{T_v(i,q) \cdot S_v(i,q)}{P(v)}, \tag{3.8, p. 24}$$

$$U_v(i,q,q') = \frac{T_v(i,q) \cdot t(q'|q) \cdot e(v_{i+1}|q') \cdot S_v(i+1,q')}{P(v)}. \tag{3.9, p. 26}$$

The IO information $\mu_{\mathcal{H}}$ has $X_{\not\perp} = V^*$ and $\Omega = \mathcal{M}(Q_\#|Q_\#) \times \mathcal{M}(V|Q)$. For any $\omega = (t,e)$ in Ω, hidden information is structured according to

$$A = Q_\# \cup V, \quad B = Q_\# \times \{T\} \cup Q \times \{E\}, \tag{p. 27}$$

$$C = \underbrace{\big\{(q',(q,T)) \mid q,q' \in Q_\#\big\}}_{\substack{q_\omega(q'|(q,T)) = t(q'|q) \\ \text{rank 0 for } q' = \#,\ 2 \text{ otherwise}}} \cup \underbrace{\big\{(v,(q,E)) \mid v \in V, q \in Q\big\}}_{\substack{q_\omega(v|(q,E)) = e(v|q) \\ \text{rank 0}}}. \tag{p. 27}$$

The grammar K has states $Q_K = \{T\} \times Q_\# \cup \{E\} \times Q$, the initial state $(T, \#)$ and the rules

$$\begin{aligned}
&(T,q) \to \big(q',(q,T)\big)\big((E,q'),(T,q')\big) && \forall q \in Q_\# \text{ and } q' \in Q,\\
&(T,q) \to \big(\#,(q,T)\big) && \forall q \in Q_\#,\\
&(E,q) \to \big(v,(q,E)\big) && \forall q \in Q \text{ and } v \in V. \qquad \text{(p. 29)}
\end{aligned}$$

Each $H(x)$ has states $Q_x = \{T\} \times Q_\# \times \big(\text{suff}(x) \cup \{\varepsilon\}\big) \cup \{E\} \times Q \times \text{suff}(x)$, the initial state $(T, \#, x)$ and the rules

$$\begin{aligned}
&(T,q,x') \to \big(q',(q,T)\big)\big((E,q',x'),(T,q',\text{tail}(x'))\big) && \forall q \in Q_\#, q' \in Q, \text{ and } x' \in \text{suff}(x),\\
&(T,q,\varepsilon) \to \big(\#,(q,T)\big) && \forall q \in Q_\#,\\
&(E,q,x') \to \big(\text{head}(x'),(q,E)\big) && \forall q \in Q \text{ and } x' \in \text{suff}(x). \qquad \text{(p. 31)}
\end{aligned}$$

For $x = \varepsilon$, only the initial state $(T, \#, \varepsilon)$ is reachable and the only nonzero inside and outside weights are

$$\beta_\varepsilon(T,\#,\varepsilon) = t(\#|\#) = P(\varepsilon), \qquad \text{(p. 33)}$$

$$\alpha_\varepsilon(T,\#,\varepsilon) = 1. \qquad \text{(p. 33)}$$

The following table shows inside and outside weights for $H(x)$ with $x = x_1 \cdots x_n$ in V^+ nonempty and rule probabilities according to $q_\omega(a|b)$ of the produced symbol $(a,b) \in C$. Unreachable grammar states are shown greyed out. All statements apply for all $q \in Q$ and $i \in \{1,\ldots,n\}$ unless otherwise noted.

Grammar state q_g	Reachable?	Inside weight $\beta_x(q_g)$	Outside weight $\alpha_x(q_g)$
$(T,\#,\varepsilon)$	no	not relevant	0
$(T,\#,x)$	yes (initial)	$P(x)$	1
$(T,\#,x_i \cdots x_n)$	no	not relevant	0
(T,q,ε)	yes	$t(\#\|q) = S_x(n,q)$	$T_x(n,q)$
(T,q,x)	no	not relevant	0
$(T,q,x_i \cdots x_n)$	for $i > 1$	$S_x(i-1,q)$ for $i > 1$	$T_x(i-1,q)$ for $i > 1$
(E,q,x)	yes	$e(x_1\|q)$	$\dfrac{T_x(1,q) \cdot S_x(1,q)}{e(x_1\|q)}$
$(E,q,x_i \cdots x_n)$	yes	$e(x_i\|q)$	$\dfrac{T_x(i,q) \cdot S_x(i,q)}{e(x_i\|q)}$

Bibliography

[Bau72] Leonard E Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[BPSW70] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[BSV15] Matthias Büchse, Torsten Stüber, and Heiko Vogler. A generic inside-outside EM algorithm. Unpublished Manuscript, 2015.

[DLR77] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[HH04] Saša Hasan and Karin Harbusch. N-best hidden markov model supertagging to improve typing on an ambiguous keyboard. In *Proceedings of Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 24–31, 2004.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson International Edition, 2nd edition, 2009.

[Kuk92] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439, 1992.

[LY90] Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[Nel13] Tobias Nelsen. Hidden markov models. B. Sc. thesis, Technische Universität Dresden, 2013. Available online: `https://forschungsinfo.tu-dresden.de/detail/abschlussarbeit/28412`.

[NLH98] Alan Newell, Stefan Langer, and Marianne Hickey. The role of natural language processing in alternative and augmentative communication. *Natural Language Engineering*, 4(1):1–16, 1998.

[S+02] Andreas Stolcke et al. SRILM – an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002, 2002.

[Wu83] C. F. Jeff Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, pages 95–103, 1983.

[YEG+05] Steve Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, P. C. Woodland, et al. *The HTK Book (for HTK Version. 3.3)*. Cambridge University Engineering Department, 2005.