Dynamic Language Models for Hybrid Speech Recognition

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultät II
- Physik und Mechatronik -
der Universität des Saarlandes

von

Munir Georges

Saarbrücken
2015

Tag des Kolloquiums:	12.07.2016
Dekan:	Univ.-Prof. Dr.-Ing. Georg Frey
Mitglieder des Prüfungsausschusses:	Univ.-Prof. Dr. Dietrich Klakow
	Univ.-Prof. Dr.-Ing. Chihao Xu
	Dr. rer. nat. Andreas Leschhorn

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Ort, Datum	(Unterschrift)

Abstract

This thesis describes methods and techniques for low resource speech recognition on embedded devices. The speech recognizer is based on a weighted finite-state transducer. A method to estimate dynamic language models is proposed. This method combines grammars and statistical n-gram models. The grammars can be used to represent word sequences with long range dependencies, such as address entries, abbreviations, proper names or stock prices. These sequences are often sparsely represented in the training data. A decoding technique is described which nests transducers at speech recognition time. This enables the recognition of privacy protected data, e.g. contact names from an user's address book. It also enables the use of favorite music titles or medical records to achieve more precise speech recognition. In addition, a novel technique is proposed which allows speech recognition on multiple devices. This technique uses a dynamic language model in combination with acoustic filler models.

All methods and techniques were evaluated on the Wall Street Journal Corpus. The transducer nesting technique was evaluated in an embedded environment. An recognition improvement from 19% word error rate to 13% was achieved. The decoding technique on multiple devices was applied for accurate client-server based speech recognition keeping personal data on the client. The average latency increased by 15% compared to real time recognition, but a word error rate reduction of 17% was achieved.

Deutsche Zusammenfassung

Diese Arbeit beschreibt Methoden und Techniken zur Spracherkennung mit Transduktoren auf eingebetteten Systemen. Insbesondere dynamische Sprachmodelle werden vorgestellt, mit denen statistische n-gram Modelle und Grammatiken kombiniert werden. Die Grammatiken werden dazu verwendet lange Wortfolgen zu beschreiben. Das sind beispielsweise Adressen, Eigennahmen oder Aktienkurse. Varianten dieser Wortfolgen sind im Datensatz selten vertreten. Es wird eine Suchtechnik verwendet, die mit geringer Speicher- und Rechenleistung verschachtelte Transduktoren zur Laufzeit decodiert. Dies ermöglicht die Verwendung von sensiblen Daten zur Verbesserung der Spracherkennung. Das kann ein Adressbuch sein, aber auch eigene Wiedergabelisten oder medizinische Aufzeichnungen. Darüber hinaus wird eine neue Technik vorgestellt, die Spracherkennung auf mehreren Geräten ermöglicht. Hierfür werden dynamische Sprachmodelle mit akustischen Füllermodellen kombiniert.

Die Methoden und Techniken wurden mit dem Wall Street Journal Corpus bewertet. Die Spracherkennung mit verschachtelten Transduktoren wurde auf eingebetteten Geräten evaluiert. Die Erkennung konnte von 19% Wortfehlerrate auf 13% verbessert werden. Diese Arbeit ermöglichte auch eine Client-Server basierte Spracherkennung bei der personenbezogene Daten auf dem Client verbleiben können. Die mittlere Verzögerung stieg um 15% verglichen mit einer Echtzeiterkennung, jedoch verringerte sich die Wortfehlerrate um 17%.

Acknowledgements

First I would like to thank Prof. Dr. Dietrich Klakow and Prof. Dr. Chihao Xu for reviewing my thesis. I am grateful to SVOX and Nuance Communication for giving me the opportunity to research and write this PhD thesis by providing challenging issues and resources for advanced research in automatic speech and language processing.
Particular thanks are due to all my speech colleagues all over the world for many inspiring conversations. Let me specially mention: Stephan Kanthak, Georg Stemmer, Oliver Bender, Josef Anastasiadis, Eduardo Vellasques, Friederike Niedtner, Ute Ziegenhain, Josef Bauer, Joachim Hofer.

This thesis is dedicated to my parents Dr. Elme and Dr. Jawdat Georges.

Contents

1 Thesis Introduction

This thesis introduces dynamic language models for automatic speech recognition. A transducer nesting technique is used to embed grammars in statistical language models, dynamically. This lead to a novel technique for hybrid speech recognition where the processing is distributed over multiple devices. This was achieved by using acoustic filler models within dynamic language models. The technique is used, e.g., for client-server bases speech recognition keeping private data on the client. Next paragraph provides a brief introduction on the background of this thesis.

The work on this thesis started in the ASR research group at SVOX in Munich, Germany. Along the research on efficient decoding methods for embedded devices, a decoder was developed for short message dictation. The company was acquired by Nuance Communications. The work was first followed up in the automotive ASR search group and later in the spoken language understanding group for cloud and hybrid automotive use-cases in Aachen, Germany. The research on this thesis contributed to recognizing heavy content data on embedded devices. Starting from the recognition of named entities in message dictation, methods and techniques were invented for speech recognition with dynamic language models on multiple devices. The work contributes to research and developing efforts for low resource and hybrid speech recognizers at Nuance for automotive and mobile use-cases.

Thesis Introduction

Automatic speech recognition is becoming an essential element in modern human machine interfaces. The areas of use range from applications in automotive, mobile and home automation to applications in the medical field. The recognition accuracy depends among other things on suitable training data for the target domain. Statistical models are used to compute the most probable word sequences given a sequence of speech features extracted from an audio stream. The training data is often derived from field data, e.g. for acoustic modeling, and text corpora, e.g. for language modeling. Scraped text from the World Wide Web is also used. Common words and phrases are well represented and suited to estimate precise statistical models. The picture changes when large structured content needs to be considered, e.g. for recognizing addresses, media or named entities in a natural language context. This is a spares data problem because it is unlikely to gather enough data representing all entities in a natural language context. In addition, these data might be privacy protected. Two questions arise:

(1) How can this data be represented in the speech search space?

(2) How to store and process the search space, e.g. to protect private data?

Both questions are addressed in this thesis. A dynamic language model is proposed. It embeds stochastic grammars into statistical n-gram Markov models during speech decoding. Whereas heavy content is accessible through large databases or collections from human experts, other data is available through the user itself. The grammar is used to incorporate these knowledge, e.g. addresses, contact names or the list of a user's favorite movies and music titles. A speech recognizer is described based on weighted finite state transducer using a nesting technique. This technique allows the use of dynamic language models on embedded devices. Not all the users data is available on a single device. It is also an issue for cloud infrastructures with data centers that are shared over the world. There are various reasons why such data might not be always available. One is a data privacy restriction for certain data, e.g. medical records. Another reason is the increasing costs of server infrastructures dealing with millions of user profiles. Moreover, medical records might be accessible on a user's smart phone, but stock prices, weather records or crowd sourced trends are immediately available in the cloud. This led to the invention of recognizing speech phrases on multiple devices, depending on where the most suitable data is accessible. It uses a dynamic language model which is combined with acoustic fillers. The proposed methods and techniques are described in four chapters as outlined on following page.

Thesis Overview

Chapter 2 introduces weighted finite-state transducers and operators for processing. The processing can change the data structure or the relation between the input and output language. The data structure of a transducer is described together with compression methods. This enables the use on embedded devices with low resources. Such a decoder was built at SVOX for research on embedded dictation.

Chapter 3 introduces automatic speech recognition: Acoustic feature computation, the representation of language as well as some pre- and post-processing of text. The investigation was done on following software: Millennium ASR at LSV, EAR/VSR at SVOX and Vocon and NCS at Nuance. It leads to the patent application of location aware speech recognition [1] and influenced the work on the patent application of processing user input [2]. Part of the introduction were published by Georges [3] and as joined work with Faubel et al. [4], [5].

Chapter 4 introduces dynamic speech decoding with transducers. The preprocessing is described including the compiling and optimization of weighted finite state transducers. The speech decoding is described in three parts: First, transducer based speech decoding for devices with low resources. Second, on–the–fly transducer nesting which enables the use of dynamic language models. Third, speech decoding on multiple devices. These methods and techniques are part of the research and developing efforts for hybrid speech use-cases at Nuance. Parts of this work were developed during discussions with colleagues. The work leads to the patent applications on speech recognition on multiple devices [6], [7] and [8].

Chapter 5 evaluates a selection of the described methods and techniques. For this, an introduction about the evaluation of language applications is given. The speech decoding with dynamic language models is evaluated. The application uses the transducer nesting technique and is applicable on embedded devices. Speech decoding on multiple devices is evaluated on an hybrid set-up. A cloud recognizer is used to recognize general speech, all personal phrases are recognized on the client. Two applications are described in this thesis and were published at INTERSPEECH and ICASSP [9], [10].

2

Weighted Finite-State Transducer

A weighted finite-state transducer is commonly used in language technology. It is a framework to represent and process relations between two languages, e.g. the relation between a sequence of phonemes and a word sequence. The first language is often denoted as input which is consumed during decoding. The second language is denoted as output which is emitted during decoding. An ϵ symbol is used to align phrases from two languages with different number of entities. This symbol is whether consuming nor emitting. A formal definition is given in section 2.1.

The following two operators have an effect on the data structure representing the languages. The languages and the relation between these are retained. The determinization and the minimization operator is described in Section 2.2 and 2.3 respectively. Whereas determinization causes a more efficient decoding, minimization reduces the required memory. The next two operator has an effect on the relation between languages. Section 2.4 describes the composition of two transducers. Section 2.5 introduces an operator which influences the ϵ sequence. It removes all non consuming and emitting, weight neutral paths. A method to compute the topological order is described in Section 2.6. An open vocabulary processing technique is proposed in Section 2.7. A data structure overview is provided in Section 2.8. A compression method is described in Section 2.9. A summary is given in Section 2.10

2.1 Introduction in weighted Finite-State Transducers

A transducer represents a weighted relation between two regular languages. An introduction on formal languages is provided, e.g. by Schönig [11]. Rabin et al.[12] generalized the notation of an automaton to multiple tape which might be the origin of final state transducers. A weighted finite-state transducer extends an automaton by an additional symbol and weight on each transition. One language is often denoted as "input" and is consumed while decoding. The other language is often denoted as "output" and is emitted in the same time the input is consumed. In speech recognition, it is used to define a mapping between phoneme and word sequences. The variability between these two sequences is represented by weights. Hence, alternative word pronunciations can be modeled as well as hidden Markov models. Mohri et al.[13], [14] wrote an overview of weighted finite-state transducer for speech processing.

In this thesis, weighted finite-state transducers are used for speech recognition with dynamic language models. As described in Chapter 4. Let $L_i \subseteq V_i^* \cup \{\epsilon\}$ be the input language over an input vocabulary V_i and let $L_o \subseteq V_o^* \cup \{\epsilon\}$ be the output language over a vocabulary V_o. The operator "*" denotes the Kleene closure and computes all sequences from the given set. An introduction of automaton theory is provided e.g. by Hopcroft [15]. The dedicated symbol ϵ has to be introduced to apply the Kleen closure on languages. It is used to align phrases from two languages with different number of entities. In addition, it can be used to summarize multiple initial or finial states to one initial or final state. This symbol is whether consuming nor emitting.

Transitions of weighted finite-state transducers are defined by triples $(V_i \times V_o \times \sigma)$ with weight σ. Each transition is associated with a source state and a destination state analogous to transitions of automatons. Hence, a transducer can be visualized as a directed graph. In speech recognition the input vocabulary might be a set of phonemes. Let L_i be such a set. The output language might be a set of words. Let L_o be such a set of words. The transducer can be used to translate the L_i to L_o accordingly to a weight, e.g. the combination of an acoustic and a language model.

A weighted finite-state transducer can be formalized in different ways. The definition may variate depending on the application and required operators. In general, it can be defined as tuple:

$$G = (V_i, V_o, Q, T, i, F, \lambda, \psi)$$

with the following definitions:

V_i, V_o	is the input and output vocabulary
Q	is a set of states
T	is the set of transitions
$i \in Q$	is the initial state
$F \subseteq Q$	is the set of final states
λ	is the initial weight
ψ	is the final weight function.

Having just one initial state is no limitation by considering uϵ transition. The set of transitions T is defined as tuples between states:

$$T \subseteq Q \times (V_i \cup \{\epsilon\} \times (V_o \cup \{\epsilon\}) \times \mathbb{K} \times Q,$$

$\mathbb{K}$ defines the semiring which is a generalization of the algebraic ring structure. The algebraic notation of a semiring $\mathbb{K}$ is described, e.g., by Kuich at al. [16]. The Semiring $\mathbb{K}$ defined as $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$, where it is equipped with associative and commutative operator $\oplus$ with identity $\bar{0}$. Also the associative operator $\otimes$ with identity $\bar{1}$ can be defined by $\mathbb{K}$. Those are commutative. A selection of common semirings is given in Table 2.1.

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Table 2.1: *Semiring selection which are commonly used for speech recognition with weighted finite-state transducer as proposed by Mori [17].*

The tropical semiring is used in automatic speech recognition together with the Viterbi best path approximation. The following equation is fulfilled, e.g. for the Boolean and tropical semiring:

$$x \oplus x = x \quad \forall x \in \mathbb{K}.$$

A decoding with weighted finite-state transducers is described in Section 4.5. Using weighted finite-state transducers for speech recognition enables the compiling and optimization of the search space in advance. This is done by applying operators as described in the following sections.

2.2 Determinization

Determinization is a structural operator on transducers. It has no impact on the relation between both languages nor an impact on the languages itself. This operator optimizes the transducer. Many computations become more efficient, e.g., decoding. This remains even though the number of states may increase during conversion from n up to 2^n states. The decoding time will become linear to the length of the input sequence. Mohri et al. also denotes deterministic transducers as sequential ones [18]. Especially speech decoding takes advantage of an efficient decoding, e.g. motivated by Ney et al. in the work of dynamic programming for speech recognition [19]. A description of determinization for language processing was provided by Mohri et al. [13]. The theory of Determinization is described in the following paragraph together with an algorithm proposed by Mohri et al. [18]. Afterwards, the determinization is discussed for an application that should evaluate area codes of phone numbers.

Determinization in theory

Every nondeterministic finite automaton can be converted into an equivalent deterministic automaton. A proof is provided, e.g. by Schöniger [11]. However, this conversion is not always possible for transducers. The set of deterministic transducers is a subset of all transducers.

In general, all acyclic transducers are determinizable and most transducers used in natural language processing are determnizable accordingly to Mohri et al.[18]. For the sake of clarity, the determinization of unweighted finite state transducers is described here. The extension to weighted transducers is described, e.g. by Mohri et al. [14]. The operator is derived from the power set construction for automatons where the power set over states is computed. This was proposed, e.g., by Rabin et al.[12]. In case of transducers, the power set construction is computed over the set of states by considering the input and output sequence of labels. The resulting transducer is deterministic and allows a efficient decoding, e.g. for speech recognition. Let T_1 be a determinizable transducer which is defined by a 7-tuple:

$$T_1 = (V_1, I_1, F_1, A, B^*, \delta_1, \sigma_1)$$

The tupel T_1 defines following entities:

V_1	set of states
$I_1 \subseteq V$	set of initial states
$F_1 \subseteq V$	set of final states
A, B	input and output alphabet
δ_1	state transition function: $V_1 \times A \rightarrow 2^{V_1}$ (power set of V_1)
σ_1	output function $V_1 \times A \times V_1 \rightarrow B^*$

Applying the determinization operator on T_1 results in a deterministic transducer T_2. The result is defined as 8-tupel:

$$T_2 = (V_2, i_2, F_2, A, B^*, \delta_2, \sigma_2, \Phi_2)$$

with the following entries:

V_2	set of states
$i_2 \in V$	initial states
$F_2 \subseteq V$	set of final states
A, B	input and output alphabet
δ_2	state transition function: $V_2 \times A \rightarrow V_2$
σ_2	output function $V_2 \times A \rightarrow B^*$
Φ_2	final output function $F \rightarrow B^*$

Further, let $a \wedge b$ be the longest common prefix of two strings a and b. Removing the prefix a is denoted by $a^{-1}(a\,b) = b$. The determinization is presented in Algorithm 2.1. The Algorithm does also use two methods J_1 and J_2. Let q_2 be a state candidate of T_2 which includes a set of states (q, ω) from T_1. Those are combined in T_2. Let $J_1(a)$ be the set of all $(q, \omega) \in q_2$ of T_2 for which a transition in T_1 exists with the input symbol a. $J_1(a)$ is defined as follows:

$$J_1(a) = \{(q, \omega) | \delta_1(q, a) \wedge (q, \omega) \in q_2\}\}$$

Further, let $J_2(a)$ be the set of all transitions from q to q' of T_1 with the input symbol a where $(q, \omega) \in q_2$. Those transitions of T_1 are combined in T_2. $J_2(a)$ is defined as:

$$J_2(a) = \{(q, \omega, q') | \delta_1(q, a) \wedge (q, \omega) \in q_2 \wedge q' \in \delta_1(q, a)\}$$

Algorithm 2.1 *Pseudo code for determinization of a transducer T_1 [18]*

1: $F_2 \leftarrow \emptyset$ // Initialize final states
2: $i_2 \leftarrow \bigcup_{i \in I_1} \{(i, \epsilon)\}$ // Unify initial states from T_1
3: $Q \leftarrow \{i_2\}$ // set inital state
4: **while** $Q \neq \emptyset$ **do** // Continue until Q is empty
5: $q_2 \leftarrow$head$[Q]$
6: **if** $\exists (q, \omega) \in q_2 : q \in F_1$ **then** // q_2 is final iff it includes $(q_1, \omega) \wedge q_1$ is a final state
7: $F_2 \leftarrow F_2 \cup \{q_2\}$ // Add final state q_2
8: $\Phi_2(q_2) \leftarrow \omega$ // ω is the output for final state q_2
9: **end if**
10: **for** each a such that $(q, \omega) \in q_2 \wedge \delta_1(q, a)$ **do** // construct output $\sigma_2(q_2, a)$
11: $\sigma_2(q_2, a) \bigwedge_{(q,a) \in J_1(a)} \leftarrow \left[\omega \bigwedge_{q' \in \delta_1(q,\omega)} \sigma_1(q, a, q') \right]$ // for transition q_2 to $\delta_2(q_2, a)$
12: // create transition for q_2 with input a
13: $\delta_2(q_2, a) \bigcup_{(q,\omega,q') \in J_2(a)} \leftarrow \left\{ (q', [\sigma_2(q_2, a)]^{-1} \omega \sigma_1(q, a, q') \right\}$
14: **if** $\delta_2(q_2, a)$ a new state **then**
15: ENQUEUE$(Q, \delta_2(q_2, a))$ // Add new state in the queue
16: **end if**
17: **end for**
18: DEQUEUE(Q) // Remove current state from queue
19: **end while**

Other implementations are proposed, e.g. by Van Noord et al. [20]. The determinization of weighted finite-state transducers is described, e.g. by Allauzen et al. [21] or Mohri et al. [14] It is also possible to add new entries incrementally in already deterministic transducers,e.g. described by Bender et al. [22]

Determinization in practice

A deterministic transducer computes the transduction of a sequence $\underline{w} = w_1 ... w_n$ with n being the number of input symbols. The transduction can be computed in linear time $\mathcal{O}(n)$. The operator is extensively used to optimize transducers for speech recognition, where the entire search space is represented as a transducer. An efficient transduction is important to achieve a recognition with low latencies. Weighted transducers are used in speech recognition which increases the computational complexity. A token passing time synchronous Viterbi beam search is often used as described, e.g. by Young et al. [23]. Its efficiency is based on a well constructed and optimized transducer. Next paragraph introduces the construction.

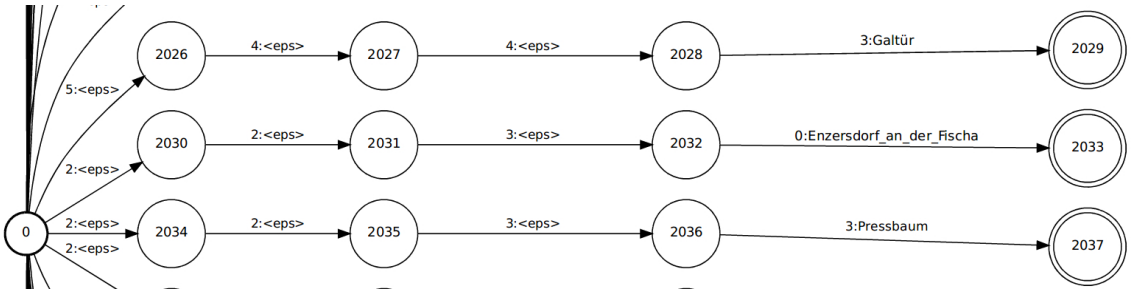


Figure 2.1: *Subset of a non deterministic transducer that decodes Austria area codes. The input sequence is a digit sequence and the output sequence are the names of areas in Austria."<eps>" denotes an empty symbol.*

Let R be the transducer used for speech recognition. It is comprises a transducer L which describes the relation between phonemes and words. Let G be a grammar representing a word sequence. The C and H transducers are used to represent the phoneme context and hidden Markov model, respectively. A detailed construction is described in Section 4.5 together with a decoding technique. R is then compiled by a sequence of transducer operators including Determinization and Minimization. Minimization is denoted by "min" and described in the next section. The operator "$\circ$" denotes the composition of two transducers and is introduces afterwards. R is compiled by:

$$R = \min(H \circ \det(C \circ \det(L \circ G))).$$

As a more general illustration, an application is described that resolves area codes. This can be used on a smart phone. The phone number of an incoming call is evaluated on a mobile phone to display the area on a map. This should increase the readability compared to just displaying the raw number. The application is realized by transducers. Alternatively, a hash table could be used. A transducer that represents the relation between each number and its area can be constructed. Figure 2.1 shows a subset of this nondeterministic transducer for Austria area codes. An equivalent deterministic representation is constructed to enable an efficient time linear decoding of phone numbers. In other words, the input symbols of the final transducer are unique for each outgoing transition at every stage. The result is a transducer that has a tree like structure. An illustration in provided in Figure 2.2.

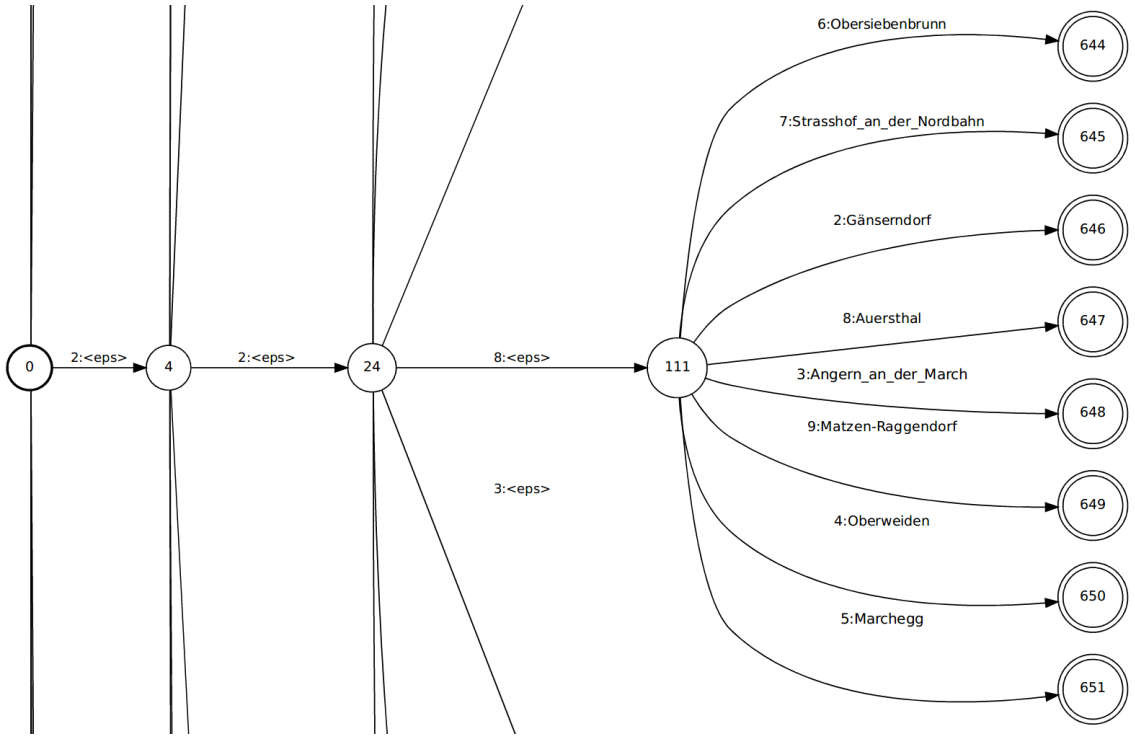


Figure 2.2: *Deterministic transducer derived from the one given in Figure 2.2.*

2.3 Minimization

Minimizing a transducer does not change the relation between both languages. Also the input and output language remain unchanged. Minimization is a structural changing operator. Whereas determinization causes a more efficient computation, minimization leads to reduced memory requirements. It reduces the number of required states. However, this might also have an impact on the processing speed. A reduced number of states may also reduces the time for memory allocations. It can also enable a more efficient use of memory caches, e.g. for decoding on embedded devices.

A pseudo code is given in Algorithm 2.2. A detailed description is provided by Baclet et al. [26]. Let Q be a set of states and let F be the set of final states with F being a subset of Q. Further let Σ be the input alphabet. For the sake of clarity, let "split(B,S,a)" split each partition B if a state in S can be reached with input symbol a. The method iterates over final states and merges all states which are reachable by equal input sequence.

Algorithm 2.2 *Pseudo code of minimizing an deterministic automation derived from Almeida et al. [24] accordingly to Hopcroft [25].*

```
L ← {}
if |F| ≤ |Q − F| then                    // There are non final states
  P ← {Q − F, F}                          // compute the partition
  L ← {F}                                 // Push all final states
else                                      // there are just final states
  P ← {F, Q − F}                          // Initialize partition
  L ← {Q − F}                             // Push all non finial states
end if
while L ≠ ∅ do                            // Iterate over all blocks in L
  S ← extracts(L)                         // Pop an element of L for computation
  for a in ∑ do                           // Iterate over the alphabet
    for B in P do                         // Iterate over all states in P
      (B1, B2) ← split(B, S, a)           // states which are reachable
                                          // from S by consuming a
      P ← P − {B}                         // Remove state from B
      P ← P ∪ {B1}                        // Add state from B1
      P ← P ∪ {B2}                        // Add state from B2
      if |B1| ≤ |B2| then                 // there are less states in B1 as in B2
        L ← L ∪ {B1}                      // add minimal states to L
      else                                // there are less states in B2 as in B1
        L ← L ∪ {B2}                      // add minimal states to L
      end if
    end for
  end for
end while
```

An alternative algorithm was proposed, e.g. by Brzozowski [27]. Also Hopcroft et al. [25], [15] described several algorithms. Especially the minimization of deterministic transducers is important for speech recognition. An algorithm is proposed, e.g. by Mohri et al. [28]. It is desired by many applications to add knowledge continuously with a minimum amount of computational power. This requires on-the-fly minimization to add address-book entries or music titles to the search space. An incremental minimization was proposed, e.g. by Watson et al. [29]. Dobrisek et al. proposed an algorithm for sequential minimization for a pronunciation lexicon [30]. The use of crossword context modeling may increase the accuracy, e.g. for message dictation. But this also increase the transducers significantly. An arc minimization method for crossword modeling was proposed by Zweig et al. [31] Speech can also often be acyclic. A minimization algorithm in linear time was proposed, e.g., by Revuz et al. [32]

2.4 Composition

The composition is a binary operator on weighted finite-state transducers. It computes the intersection of two transducers accordingly to a shared language, e.g. the output from one transducer and the input of the second transducer. The result is a new transducer representing the relation between the input and output of the first and second transducer, respectively. The use of a weighted composition was introduced, e.g. by Fernando et al. [33]. Let "$\circ$" be the composition operator. It computes the relation between P and G given the intersection of W_1 and W_2. Without loss of generality, let P be the language over a sequence of phonemes and G a language grammar. W_1 and W_2 are two languages over a set of words with:

$$W_1 \cap W_2 \neq \emptyset.$$

W_1 is the vocabulary of lexicon R_L as described in Section 4.2. W_2 is the vocabulary used by the language model or grammar. The relation P describes a grapheme to phoneme dictionary. It is defined over sequences of words represented as transducer R_G. A construction of R_G is given in Chapter 4.3. Let R_L and R_G are two binary relations representing the lexicon and grammar, respectively. Both are defined by:

$$R_L \subseteq (P \times W_1)^* \tag{2.1}$$

$$R_G \subseteq (W_2 \times G)^* \tag{2.2}$$

The binary relation $R_L \circ R_G \subseteq P \times G$ is the composition of R_L and R_G. The composition computes the intersection given W_1 and W_2. It is defined as follows:

$$\begin{aligned} R_L \circ R_G = \{&(x,z) \in P \times G \mid \\ &\exists y \in W_1 \cap W_2 : (x,y) \in R_L \wedge (y,z) \in R_G\}. \end{aligned}$$

The result is a transducer that represents the relation between phoneme sequences and words. This transducer is composed with an hidden Markov Model and used for automatic speech recognition. In the following is a description of the operator given together with the use in practice.

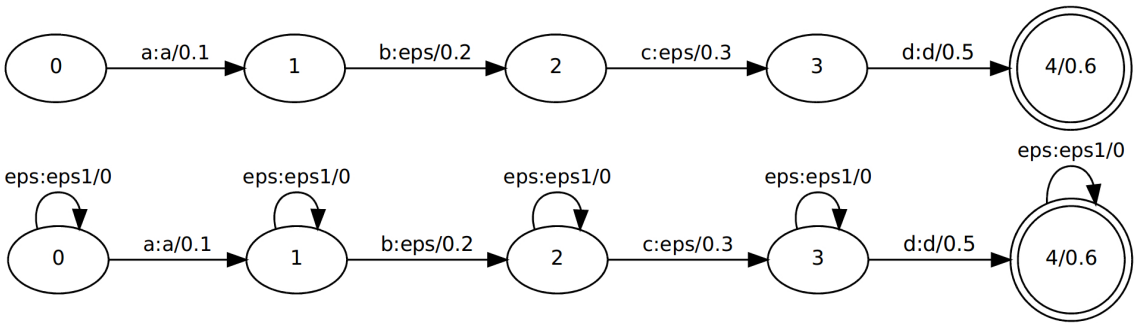


Figure 2.3: *Transducer T_1 on the top. On the bottom T_1' with auxiliary symbols required for ϵ-free composition; The "eps" symbol on the output is replaces by "eps1" and self loops "eps:eps1/0" are introduced*

Composition in theory

Here, the formalism and algorithm proposed by Mohri in [17] is taken up and described together with an example. Let T_1 be a weighted transducer. For the sake of clarity, let T_1 be the one illustrated in Figure 2.3 at the top of the page. It accepts the input language "abcd" and returns "ad" as output sequence, respectively. More generally, T_1 is declared as a 8-tuple:

$$T_1 = (\Sigma^*, \Delta^*, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1)$$

with the following entries:

Σ, Δ	input, output alphabets
Q_1	set of states
I_1	set of initial states
F_1	set of final states
E_1	set of transitions; $Q_1 \times (\Sigma_1 \cup \{\epsilon\}) \times (\Delta_1 \cup \{\epsilon\}) \times \mathbb{K} \times Q_1$
λ	an initial weight function $\lambda : I \to \mathbb{K}$
ρ	a final weight function $\rho : F \to \mathbb{K}$

where $\mathbb{K}$ is a semiring defined by:

$$(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$$

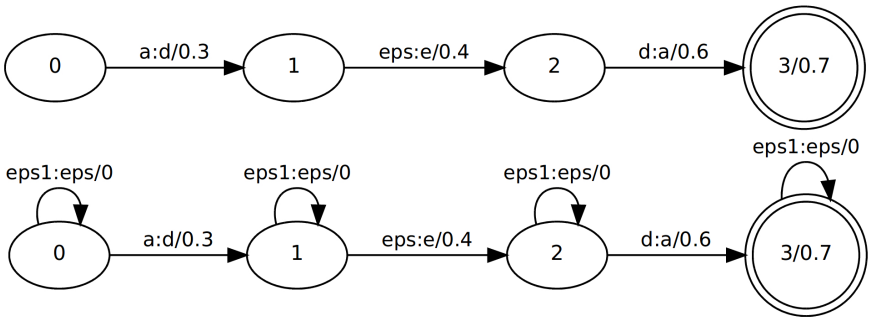


Figure 2.4: *Transducer T_2 on the left. On the right T_2' with auxiliary symbols required for ϵ-free composition; The "eps" symbol on the input is replaced by "eps1" and self loops "eps1:eps/0" are introduced*

The semiring can be chosen from Table 2.1 in the introduction section on weighted finite-state transducers. According to the example, let Σ, Δ be

$$\Sigma = \{a, b, c, d\}$$
$$\Delta = \{a, d\},$$

with initial state "0" $\in I_1$ and final state "4" $\in F_1$. Here, the final state has a weight of 0.6. Further, let T_2 be a weighted transducer. For the sake of clarity, let T_2 be the one illustrated in Figure 2.4 at the top. The input alphabet is $\Delta = \{a, d\}$ and the output one is $\Omega = \{a, d, e\}$. T_2 accepts the input language "ad"$\subseteq \Delta^*$ and returns "dea"$\subseteq \Omega^*$ as output sequence. More generally, T_2 is declared as the following 8-tuple:

$$T_1 = (\Delta^*, \Omega^*, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$$

with the following entries:

Δ, Ω	input,output alphabets
Q_2	set of states
I_2	set of initial states
F_2	set of final states
E_2	set of transitions; $Q_2 \times (\Sigma_2 \cup \{\epsilon\}) \times (\Delta_2 \cup \{\epsilon\}) \times S \times Q_2$
λ	an initial weight function $\lambda : I \rightarrow S$
ρ	a final weight function $\rho : F \rightarrow S$

Note, the input alphabet of T_2 is Δ. It is shared with the input alphabet of T_1.

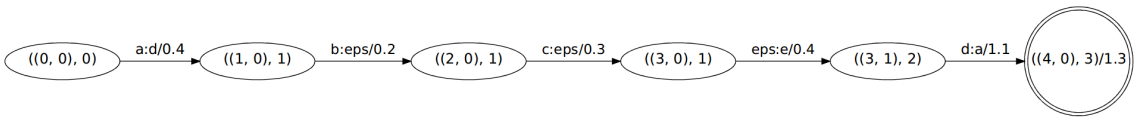


Figure 2.5: *composition of T' and T': $R_{emove\epsilon}((T' \circ \epsilon_1) \circ T')$*

In the example, it is obvious that the output language of T_1 matches the input of T_2. In general, the output language of T_2 has to be a subset of the language T_2 accepts. Here, the composition computes the intersection accordingly to the shared language Δ. The composition of T_1 and T_2 results in T as presented in Figure 2.5. In the example, the following strings are processable:

$$T_1 : \text{"abcd"} \rightarrow \text{"ad"}$$
$$T_2 : \text{"ad"} \rightarrow \text{"dea"}$$
$$T : \text{"abcd"} \rightarrow \text{"dea"}$$

T has the input alphabet $\Sigma = \{a, b, c, d\}$ and the output alphabet $\Omega = \{a, d, e\}$. Given the input string "abcd" it will return "dea". This is the sting T_1 accepts and the output T_2 returns for an appropriated input. More generally, T is declared as 8-Tuple:

$$T_1 \circ T_2 = T = (\Sigma^*, \Omega^*, Q, I, F, E, \lambda, \rho)$$

where

Σ, Ω	input,output alphabets
Q	set of states
I	set of initial states
F	set of final states
E	set of transitions; $Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times S \times Q$
λ	an initial weight function $\lambda : I \rightarrow S$
ρ	a final weight function $\rho : F \rightarrow S$

The Algorithm 2.3 introduces the composition of weighted transducers. The following notation was used for the sake of clarity:

$$E = \biguplus_{\substack{(q_1, a, b, \omega_1, q_2) \in E_1 \\ (q_1', b, c, \omega_2, q_2') \in E_2}} \left\{ \left((q_1, q_1'), a, c, \omega_1 \otimes \omega_2, (q_2, q_2') \right) \right\}$$

Algorithm 2.3 *Pseudo code for composition as published by Mohri [17].*

$Q \leftarrow I_1 \times I_2$ // Add initial states
$\Theta \leftarrow I_1 \times I_2$ // Add initial states
while $\Theta \neq \emptyset$ **do**
  $q = (q_1, q_2) \leftarrow$HEAD($\Theta$) // create new state
  DEQUEUE(Θ) // remove state form the queue
  if $q \in I_1 \times I_2$ **then** // check if new state is an inital one
    $I \leftarrow I \cup \{q\}$ // add new initial state one
    $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
  end if
  if $q \in F_1 \times F_2 the follow ing$: **then** // check if new state is a final one
    $F \leftarrow F \cup \{q\}$ // add new final state
    $\rho(q) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$
  end if
  // walk over all transitions in T_1, T_2
  for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ **do**
    // check matching of out-, input transitions of T_1, T_2
    if $(q' = (n[e_1], n[e_2]) \notin Q)$ **then**
      $Q \leftarrow Q \cup \{q'\}$ // add new state
      ENQUEUE(Θ, q') // append new candidate to the queue
    end if
    $E \leftarrow E \uplus \{(q, i[e_1], o[e_2], \omega[e_1] \otimes \omega[e_2], q')\}$ // add new transition
  end for
end while

The composition is a local operator just considering current states and transitions of both transducers. Further on, it is a ϵ-free composition and treats ϵs as regular symbols of the input and output language. Starting from the initial states $q_1 \in I_1$ and $q_2 \; in I_2$ a new state q is generated. For the sake of clarity, let q be labeled as (q_1, q_2). In a first iteration, q will become an initial state because q_1 and q_2 are initial states, too. The method iterates over all outgoing transitions of q_1 and q_2. A new transition is initialized if the output label from a transition of q_1 matches the input label from an outgoing transition of q_2. In addition, a new state is created which is denoted by the target states. Here, these new states are labeled as tuples from its origin state label. Each target state of T_1 and T_2 is pushed to the queue, e.g. $q_i \in Q_1$ and $q_j \; in Q_2$. These are processed in the next iteration. If both target states $q_i \in I_1$ and $q_j \in I_2$ are initial, the resulting state (q_i, q_j) will be initial, too. If both states $q_i \in F_1$ and $q_j \in F_2$ the resulting state (q_i, q_j) will be final.

The composition does not make a difference between symbols from the regular vocabulary and a ϵ symbol. A preprocessing is required to compose transducers with ϵ-transitions. Let T_1 and T_2 be transducers with such ϵ-transition. T_1 and T_2 are augmented with auxiliary symbols during preprocessing. The result is a transducer T_1' and T_2', respectively. All ϵ-transitions are replaced by "eps1" on the in and output. In addition, a "eps"-self loop is added to each state. All other transitions remain. The transducer T_1' of the example is presented in Figure 2.3 at the bottom. T_2' is presented in 2.4 at the bottom, respectively. Further, a ϵ-filter is applied on T_1'. The one used here is presented in Figure 2.6. The input language matches the output language of T_1. It

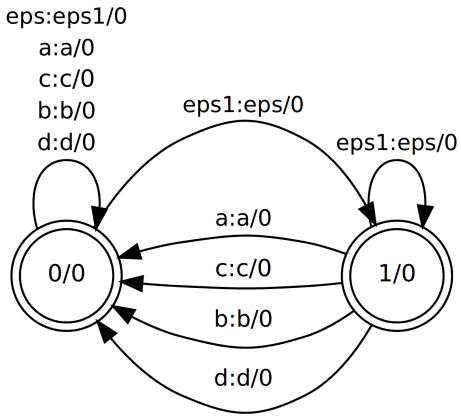


Figure 2.6: *ϵ-Filter for T_1'. In practice, this transducer is built on the fly.*

computes the identity function for all non ϵ-transitions. In contrast, all ϵ-transitions are replaced by "eps1" in the output. Hence, the composition of T_1' with ϵ_1 enables the final composition with T_2'. A ϵ-filter is applied on T_1' as follows:

$$T_1'' = T_1' \circ \epsilon_1$$

The new transducer includes several "eps"-transitions. On the one hand, "eps"-transitions are "empty" symbols and do not change the input or output language. On the other hand, the language is extended by several "eps" when considering these as valid symbols. Here, T_1'' is illustrated in Figure 2.7. In this way, an ϵ-free composition can be used, since all symbols are treated equally. Finally, the composition for T_1 and T_2 is computed the same way as with three single compositions as follows:

$$\begin{aligned} T =& R_{emove\epsilon}(T_1'' \circ T_2') \\ =& R_{emove\epsilon}(T_1' \circ \epsilon_1) \circ T_2') \end{aligned}$$

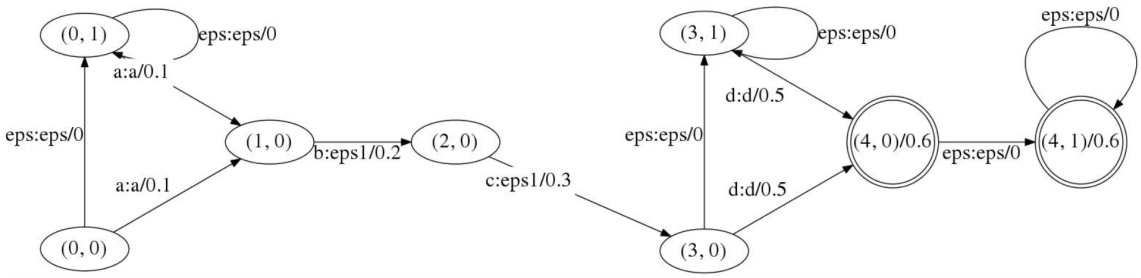


Figure 2.7: $T_1'' = T_1' \circ \epsilon_1$. T_1'' *is used for* ϵ*-free composition with* T_2'.

Note, the result may include several ϵ-Transitions which are removed by $R_{emove\epsilon}$. This operator is described in 2.5 and usually applied together with the composition. Fernando et al. also described the composition for weighted finite-state transducers [33]. The transducer in speech recognition can become huge and it is often not suited to keep a compiled transducer in memory. An on–the–fly computation with considering cross-word contexts was motivated by T. Hori et al. [34], [35] and further improved by J.W. McDonough et al. [36] and C. Allauzen et al. [37], [38]. These methods enables to compose two transducers during run time. This may saves memory because the composition and the operator itself can consume a lot of memory.

Composition in practice

As an illustration, let C be a transducer that stores facts related to contacts on a smart phone, e.g. the birthday of contacts or wedding days. An example transducer is shown in Figure 2.8.

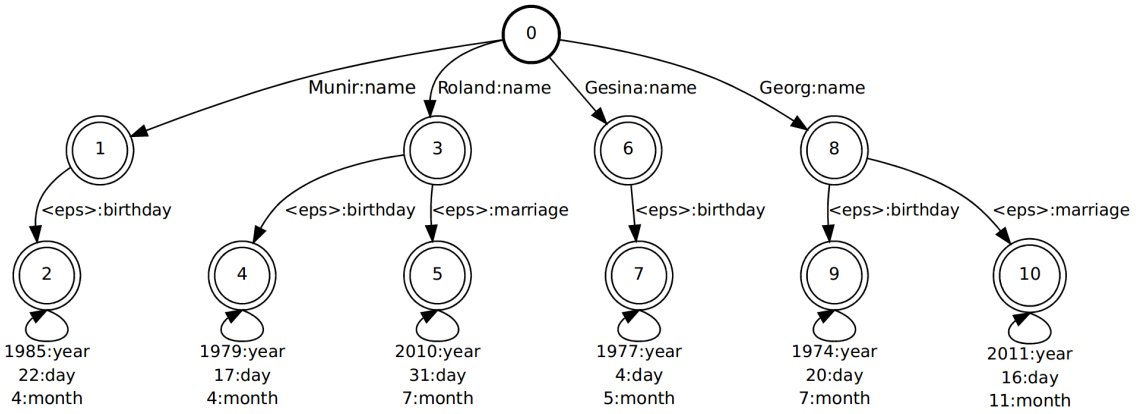


Figure 2.8: *Transducer C that stores personal information, e.g. in an address book.*

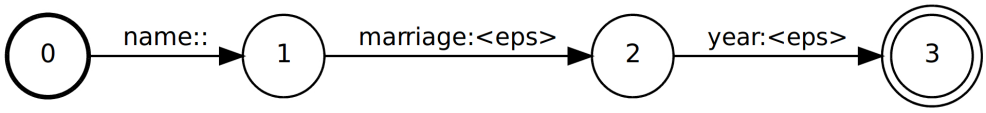


Figure 2.9: *Transducer M_F that is used to retrieve all wedding days. The transducer does also defines the result text formatting: "<name> : <marriage> <year>"*

More precisely, the input symbols are facts whereas the output are semantic tags, e.g. "1985" is a fact and has to be interpreted as "year". This transducer might be available to other applications, too. One of these applications might be an overview application to display all wedding days of the contacts in the contact list. Here, a composition is used to retrieve the desired content and apply some text formatting.

Let M_F be the transducer that is used to retrieve all wedding days out of the content and apply some text formatting. Here, just the year of the wedding is retrieved. The transducer is illustrated in Figure 2.9. The composition M is defined by:

$$M = C \circ M_F$$

The resulting transducer is illustrated in Figure 2.10. An application might treat "<eps>" as ϵ symbol and use M to list all wedding years, e.g.:

Georg:2011
Roland:2010

Other knowledge in C can be retrieved in a similar way, e.g. the birthdays. All the knowledge is already available in C but the formatting information is introduced by the composition. The composition does also choose which content of C will be extracted. Let B_F be the transducer specifying the desired content with a desired text formatting. Here, the output symbol is interpreted as a delimiter symbol, e.g. the date should be formatted with hyphens in the following order: "year - month - day.". This is illustrated

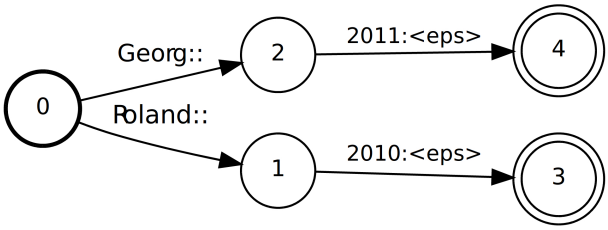


Figure 2.10: *Result transducer M of the composition of the transducers C in Figure 2.8 and M_F in Figure 2.9. Hence, it is computed by $M = C \circ M_F$.*

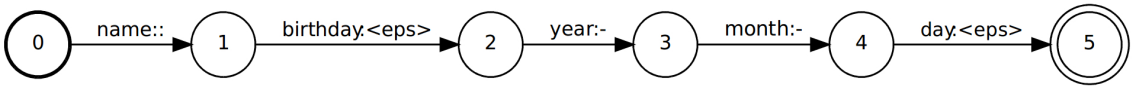


Figure 2.11: *Transducer B_F that is used to retrieve birthdays. The transducer does also defines the result text formatting: "<name> : <birthday> <year>-<month>-<day>"*

in Figure 2.11. The composition $B = C \circ B_F$ is a transducer with contacts and their birthdays. B is illustrated in Figure 2.12. To list all birthdays, one has to go through the transducer in depth first order and just use the output symbols as delimiter for the input symbol. The result would look like:

Georg:1974-7-20
Gesina:1977-4-5
Roland:1979-4-17
Munir:1985-4-22

The composition of transducers is frequently used in language processing. This thesis uses composition to compile the search network for speech recognition starting from a lexicon and grammar. Composition can be seen as a generalization of decoding. The symbol sequence for decoding can be represented as transducer and used for composition. A depth first search in the resulting transducers will result in the same set of symbol sequences as computed by decoding.

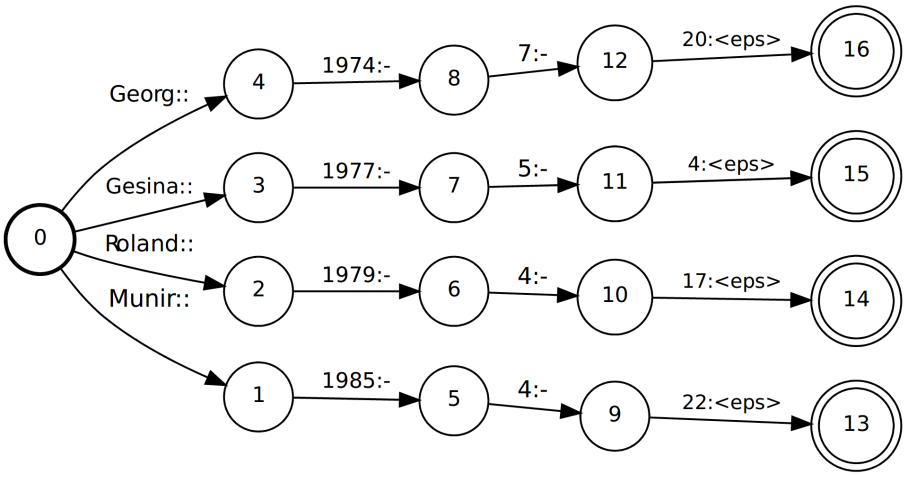


Figure 2.12: *Result transducer B of the composition of the transducers C in Figure 2.8 and B_F in Figure 2.11. Hence, it is computed by $B = C \circ B_F$.*

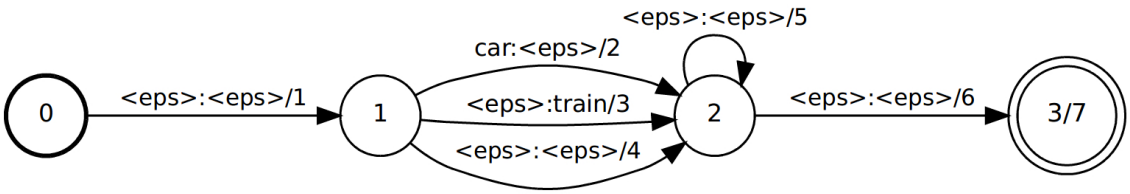


Figure 2.13: *Example transducer with ϵ-transition.*

2.5 Epsilon removal

It might happen that two states are connected by a non consuming and non emitting weight-neutral path. Each transition in such a path has a ϵ label on both, the input and output. The weights along such a path is balanced accordingly to the used semiring. An example transducer is presented in Figure 2.13. Applying the ϵ-removal will result in the transducer illustrated in Figure 2.14.

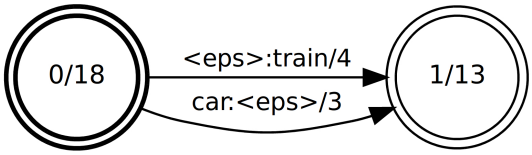


Figure 2.14: *The result of an ϵ-removal on the Transducer shown in Figure 2.13.*

A method for a general ϵ-Removal for automaton was proposed by Mohri et al. [39] and is presented in Algorithm 2.4 For the sake of clarity, let $\text{Trans}_M[s]$ be the set of outgoing transitions of s and let $\text{Next}(t)$ be the destination state of transition t. The label of t is given by $i(t)$ and the weight is given by $w(i)$. If M_i is a weighted automaton with ϵ-transitions, M_o will be an equivalent automaton without. Algorithms for ϵ removal and input ϵ normalization are described by Mohri et al. [40], [40]. A method that extracts ϵ cycles from finite-state transducers was proposed by Kempe [41]. The ϵ removal operator is important for speech recognition. It reduces the memory consumption and speeds up the decoding by removing irrelevant paths from the search network. This operator is often implemented beside others and it is typically not required to be applied.

Algorithm 2.4 *Pseudo code of the ϵ-removal algorithm [39]*

$M_\epsilon \leftarrow M_i|_{\{\epsilon\}}$ // Store all ϵ-transitions of M_i
$M_o \leftarrow M_i|_{\sum^*_{-\{\epsilon\}}}$ // Store all non-ϵ-transitions of M_i
$G_\epsilon \leftarrow$ CLOSURE(M_ϵ) // compute the all-pairs shortest distance allgorithm on M_ϵ
for $p \leftarrow 1$ to $|V|$ **do** // iterate over all states
  for each $e \in \text{Trans}_{G_\epsilon}[p]$ **do** // iterate over all transition
    for each $t \in \text{Trans}_{M_i}[\text{Next}(e)] \wedge i(t) \neq \epsilon$ **do** // iterate over all non ϵ-transitions
      $t' \leftarrow$FindTrans($i(t)$,Next(t),Trans$_{M_o}[p]$) // Create state with label $i(t)$ and weight $w(t)$
      $\omega(t') \leftarrow \omega(t') \oplus \omega(t) \otimes \omega(e)$ // update the weights caused by an ϵ-path
    end for
  end for
end for

2.6 Topological order

Computing the topological order is essential for decoding. A directed graph is used to describe a processing schema. For example, intermediate results are stored in states whereas the computation is described in transitions. This schema is often used, e.g. in parallel computing, project management or to define a "just in time" production. The transducer needs to be acyclic. An example is given in Figure 2.15.

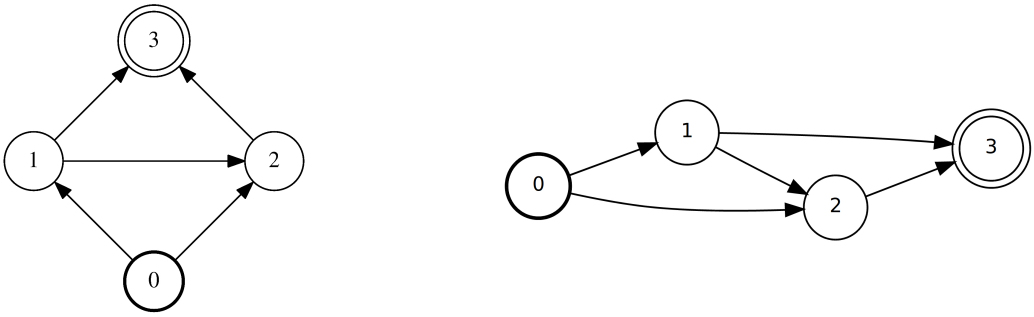


Figure 2.15: *Illustration of a topological sort. The right graph can be processed from the left to the right; The topological sort is "0,1,2,3".*

In speech recognition, the language and acoustical score is stored in each state. The transitions between states denote the phoneme sequence which should be evaluated to recognize a word sequence. Computing the graph in topological order ensures that all intermediate result are computed in time. Hence, all results were computed right before the next state is considered. In some sense, it combines the breadth-first and depth-first search by considering paths. An introduction in breadth- and depth-first search is provided, e.g. by Alsuwaiyel [42]. The graph on the right side is in topologi-

cal order evaluating the states from the left to the right. The computation for time for the topological order of a directed acyclic graph is asymptotically linear in the number of states and edges $\mathcal{O}(|V|+|E|)$. There are various algorithms known to compute the topological order, e.g. by Kahn [43] and Tarjan [44]. Haeupler et al. described an algorithm for fast incremental topological ordering [45]. In this thesis, the order is derived from a depth-first search and is used for speech decoding as described in Section 4.5. Algorithm 2.5 initializes the transducer by marking each state as unprocessed. This

Algorithm 2.5 *Pseudo code of Topological order part 1: initializing the transducer by marking each state as unprocessed. The result state list will be available in L.*

```
∀v ∈ S : v ← white                          // Initialize all states
for v ∈ V do                                // iterate over all states
  if S[v]=white then                        // process only unseen states
    startFromVertex(G, v)                   // Call the method described in Algorithm 2.6
  end if
end for
L                                           // The result is available in the queue L
```

is denoted as "white" in the pseudo code. The method in Algorithm 2.6 is called recursively and processes each white states. If a state is reached for the first time, it is

Algorithm 2.6 *Pseudo code of Topological order part 2: recursively search for the next candidate, result state order in L.*

```
S[v] ←gray                                  // Mark each state as "visited"
for w adjacent to v do                      // Iterate over all following states
  if S[w] = white then                      // process only unseen states
    startFromVertex(G, w)                   // Recursive call
  else if S[v] = gray then                  // process already processed states
    S is cyclic, abroad!                    // Error, topological order is not defined on cyclic graphs
  end if
end for
S[v] ←black                                 // Mark finished states
L.insert(v)                                 // fill L in topological order
```

marked as "gray" and "black" if the state is processed for the very last time. The topological order is given by the order the states are marked as "black".

The topological order is compute at each time a new set of features arrived in speech recognition. Fortunately, it is sufficient to compute the order on all non emitting transitions, e.g. ϵ transitions. The topological order is also used to determine transitions which can be computed in parallel.

2.7 Robust language processing

In language processing, the input vocabulary is often unknown in advance. Additionally, the output vocabulary may also be unknown. This happens, e.g., when a web-scraped text needs to be processed. It is observable that often new words are created or words are written wrongly. A method is proposed which enables a transducer to process an open vocabulary text. This method is also used to extract unknown word sequences from a text. An example is the extraction of named entities. Each name is introduced by "Mr." or "Mrs." in the corpus but the name itself is unknown and needs to be extracted. For this, the functions for input and output symbol handling is modified in the transducer framework. Now, the functions are realized by λ_I, λ_J notations. The function can be defined for each transition and in- and output-label, individually. This enables a robust language processing with transducers. A traditional input handling is realized by the following declaration:

$$\lambda_I\, w.\text{True if } w = \text{"input\_symbol" else False.}$$

where "input_symbol" is the input label assigned to the current transition. In general, the input function is defined over $W \rightarrow \{\text{True}, \text{False}\}$ where W can be an arbitrary vocabulary. It depends only on the used λ_I-method. A path will be followed if the input function evaluates "True". Another example is given in the following:

$$\lambda_I\, w.\text{True if } w \in W \text{ else False.}$$

It passes all words from the vocabulary W whereas all unknown words w are discarded. Hence, it filters a text by known words. If the input evaluates "True" and the output function is evaluated the next state will become active. The output handling was also modified. In general, it is defined over $W \rightarrow W'$ where W depends only on the λ_F-method computing the output symbol. An example looks like the following:

$$\lambda_F\, w.w \text{ if } w \in W' \text{ else } \epsilon.$$

All words which are not in W' are replaced by ϵ. Hence, the words are no longer visible in the resulting transducer. Another example is given in Figure 2.16 which shows a transducer that replaces all unknown words by "<unk>".

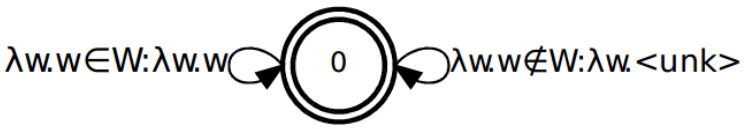


Figure 2.16: *A transducer that uses lambda methods on each transitions. Here, a transducer that replaces all unknown words with "<unk>".*

This transducer modifications are very powerful on the one hand. On the other hand, it introduces also some complexity that might not be required for many applications. However, the technique shows to be helpful for applications discussed in the thesis. It is possible to use a global set of variables, e.g. a list of proper names which should be searched on a web-page. Further, the concept can be used to handle context free languages when a stack memory is introduced and used as global variable for both λ_I and λ_F. However, there are also some disadvantages. First, the transducer construction may need some more attention because the deterministic-assumption is difficult to predict for λ declarations. Second, the transducer minimization, determinization and composition operators do not evaluate any λ declaration and it assumes that all λ declarations are non overlapping and ϵ-free.

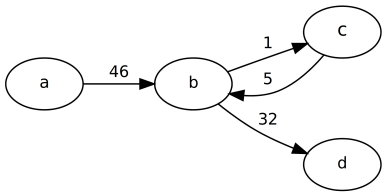
2.8 Data structures

A transducer is treated as directed graph for defining a suitable data structure. The structure has a significant impact on both, the memory consumption and the computational time it takes to apply operators. In speech recognition, the transducer is optimized for efficient decoding on the one hand. On the other hand, the size should not exceed certain limitations, e.g. constant storage, random access and cache sizes. In the following, are common data structures described: adjacency-, incidence matrix and list. The spectral graph theory uses both, adjacency and incidence matrix to close the gap between linear algebra and graph theory. This was introduced, e.g. by Brouwer et al. [46]. A data structure can be optimized for certain operators such as composition, decoding, etc. Let $V = \{v_1, .., v_n\}$ be the set of states and let $E = \{e_1, .., e_m\}$ be the set of transitions for a graph $G = (V, E)$. Note, a transducer is defined by augmenting the states and transitions with some more information, e.g. the marker of a state is initial, final, weighted or the transitions input, output symbol and weight.

An adjacency matrix for G is defined as follows:

$$a_{ij} = \begin{cases} w_{ij} & \text{if } (i,j) \in E \\ 0 & \text{else.} \end{cases}$$

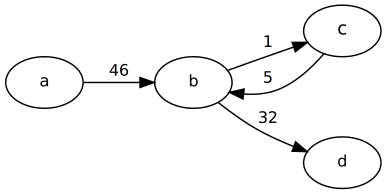
for w_{ij} the label between state i and j. It takes a $\mathcal{O}(n^2)$ memory given n states. An example is shown in Figure 2.17. A undirected graph has a symmetric matrix. Hence, the



$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 46 & 0 & 0 \\ 0 & 0 & 1 & 32 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2.17: *Example graph and its representation with an adjacency matrix.*

matrix has a complete set of real eigenvalues and an orthogonal eigenvector basis. In spectral graph theory, these eigenvalues are the spectrum of the graph, e.g. introduced by Brouwer et al. [46]. A graph with no self-loops has zeros on the main diagonal. In addition, the matrix product A^n for a not weighted graph gives the number of walks between two states of length n. Adjacency matrices are used for various applications, e.g., ranking of web-pages according to hyper-links. In contrast to the adjacency, the incidence matrix stores the transitions for each state. An example is shown in Figure 2.18. It takes $\mathcal{O}(n \cdot m)$ memory where m is the number of transitions in G. This data



$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a,b & b,c & c,b & b,d \end{array} \\ \begin{pmatrix} 46 & 0 & 0 & o \\ 0 & 1 & 0 & 32 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2.18: *Example graph and its representation with an incidency matrix.*

structure is more suitable for graphs with far more states than transitions. Formally, the matrix is defined as following:

$$b_{ij} = \begin{cases} w_{ij} & \text{if } v_i \in e_j \\ 0 & \text{else.} \end{cases}$$

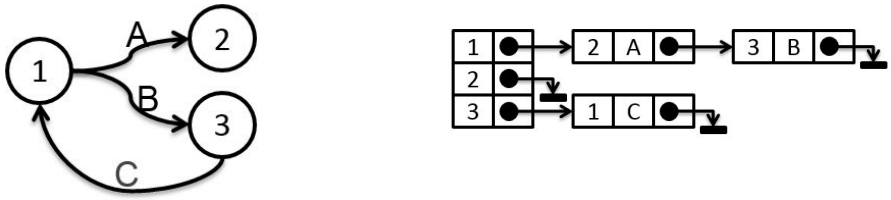


Figure 2.19: *Example graph and its representation with an adjacenc list. The list is a linked list with terminal symbol. State* 1 *has two following states* 2 *and* 3*. State* 2 *has no following states.*

An decoding efficient data structure for graphs is list based. A common representation uses adjacent lists. It is a list of stages with each stage linked to a transition, e.g. let x and y be two stages so that $E \subset V \times V$ and $(x, y) \in E$. For each $x \in E$, a list $A(x)$ of all x-following states is defined as follows:

$$A(x) = \{y \in V : (x, y) \in E\}.$$

The intention is a stage-oriented processing for which each stage can be hashed, efficiently. An example is given in Figure 2.19, where the set of outgoing transitions is represented as linked list. The links are denoted by arrow and terminals are marked by a ground symbol. Usually, the stages are organized in an array whereas the outgoing transitions for each stage are linked lists. In this thesis, an array is also used for transitions. A stage is linked to the set of all its outgoing transitions. This reduces the memory requirement significantly.

The concept of adjacent lists was taken up, e.g. by Goodrich et al. [47] to represent a graph with an incidence list. In contrast to previous structure, it is a list of transitions where each transition points to a source and destination stages The list of transitions can be organized as an array, similar to stages in adjacent lists. Also linked list are often used. Figure 2.20 on next page shows an example and is formalized as:

$$I(x) = \{(x, y) \in E : x \in V\}$$

Both list representations are suited to store weighted finite-state transducers. Each transition stores an output and input symbol together with a weight. In addition, a state needs to be marked as initial or final state. Usually a second array is used to quickly access these states. The data structure is suited for embedded usage because it enables an efficient block decoding by evaluating all outgoing transitions at once.

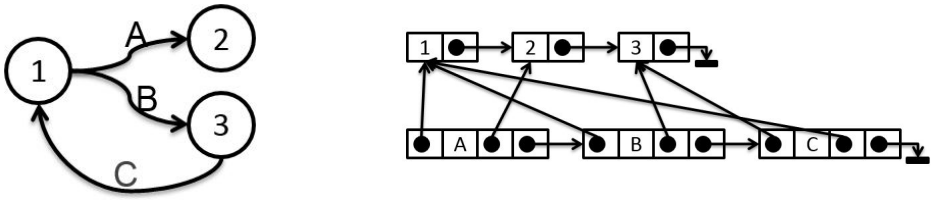


Figure 2.20: *Example graph and its representation with an incidence list. Linked list are used for both, states and transition.*

A garbage collector is not required for not mutable transducers. For other transducers, a concept needs to be implemented that enables to change the transducer without rebuilding it from the very beginning. In practice, the mutable transducer is used during decoding. It is built during decoding and represents the current search sub space. Typically, stages or transitions need to be deleted or the target of a transition changes. In practice, it is also common to use smart pointers. The concept is used in Section 4 where it is used to build the word history tree during decoding. An example implementation for weighted grammars and transducers for language processing is provided by Allauzen et al. [48], [49].

2.9 Compression methods for transducer

Reducing the memory requirement is essential for embedded speech recognition and language processing with transducers. Reducing the static and dynamic memory consumption has a significant impact on the run-time. Aqrawi et al.[50] described the effect of compression on data intensive algorithms. A compact representation of finite-state transducers was proposed by Mohri et al. [51]. Blandford et al. described a compact representation of separable graphs [52]. This section describes the used data structure for efficient decoding for low resource speech recognition.

The computer memory is organized as linear array with a fixed length per entry which is usually one byte. A compressed row and column storage for matrices was proposed by Barrett et al. [53] together with "Block Compressed Row Storage", "Compressed Diagonal Storage" and "Jagged Diagonal Storage". These are powerful data structures used by many libraries for matrix computations, e.g. BLAS. However, these data structures are in general not suitable to represent transducers for language processing and speech recognition. Two memory blocks are required to store all states and transitions as discussed in the previous section 2.8. One is used to store states and

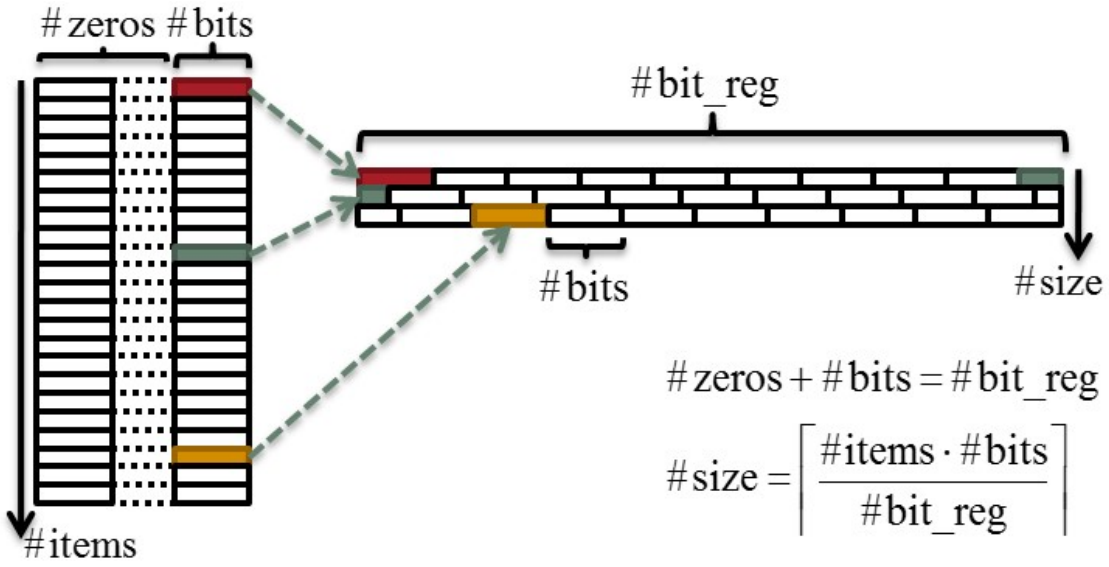


Figure 2.21: *On the left side is a linear organized memory, e.g. storing states. On the right side is a compressed version using a fixed block length.*

the other one is used to store transitions. Each state is addressable and every state has a pointer to a block of outgoing transition. In addition, the number of outgoing transitions is stored per state. This enables a fast iteration over all outgoing transitions.

Here, this weight is also used to identify a final state. The states are directly addressable as long as each state has an identical size, e.g. two byte. Each transition has an input and output symbol, a weight and a destination state address. The outgoing transitions of a state are consecutively stored in the list of transitions as one block. The set of transitions is directly addressable as long as each one has an identical size, e.g. four byte. In this way, a breath-first search can be processed most effectively. An optimized data structure for improved cache performance was proposed by Lam et al. [54] Cache optimized algorithms on graphs are described by Park et al.[55]. This is beneficial for speech decoding to evaluate the next time frame for all consuming transitions. Speech decoding is described in Section 4.5.

The compression of weighted finite-state transducers was proposed by Caseiro et al. [56] and Toivonen et al. [57]. The compression method proposed in this thesis enables arrays with byte unequal entry sizes. Hence, it might be sufficient to store states with six bits and transitions with 12 bits instead of 8 and 16 bits, respectively. This fix block length coding already reduces the memory. An illustration is given in Figure 2.21. Note that each entry is directly addressable. This structure can be implemented efficiently, e.g. using Boolean operators. In addition, parallel programing

can be used, such as the "single instruction, multiple data" code execution extension. Further compression can be realized by a variable block length coding. Each entry can have a different size using a stop-bit at the end of each entry. Searching an entry would requires to go through the entire array. However, using a bit-array as index to store start addresses can speed up the search. In addition, a hierarchical index structure can be used to speed up even the search in the index itself. The result is a direct addressable compression using a variable block length coding as described, e.g., by Brisaboa et al. [58] and Williams et al. [59]. In practice, it is established to use a combination of both methods. The fixed block length is used to store states and transitions, e.g. clustered by the number of bits the entities need. The variable block length coding is used with entries to optimally use the available bits for different data, such as the input, output label, weight, etc. Finally, a fast dictionary based compression method could be used, accordingly to Skibinski et al. [60]. In addition, Whittaker et al. described a quantization approach for n-gram language models [61]. A compression method for language models used for speech recognition was proposed by Olsen et al. [62].

2.10 Summary

This chapter introduced weighted finite-state transducers as a framework for processing regular languages and relations between languages. The application of transducers to human language processing and speech processing was propose, e.g. by Pereira et al. [63] or Mohri et al.[13]. Section 2.1 described the representation of regular languages as states and transitions following the definition of automatons and transducers. In general, it was proposed to consider multi tape automaton, e.g. outlined by Furia [64]. In this thesis, one and two tape automatons are considered which are denoted as automatons and transducers, respectively. There are two classes of operators applicable on automatons and transducers.

The first class which only influences the way a language or the relation between two languages is represented as automaton or transducer, respectively. These operators facilitate the use of transducers on common computational hardware. Two operators are most important for speech recognition. (1.) The Determinization operator described in Section 2.2 optimizes the automaton/transducer for an efficient decoding. (2.) The Minimization operator described in Section 2.3 optimizes the automaton/transducer to fulfill memory requirements.

The second class of operators influences the language or the relation between two languages. For example, the composition operator is used to compile the search space for speech recognizers. This can be done in a very modular way so that parts, e.g. the lexicon transducer, can be used for various projects. Section 2.4 described the composition operator. Another operator of this class is the ϵ-Removal operator. It removes all ϵ-paths from the automaton or transducer, respectively. This operator changes the languages assuming that ϵ is part of the regular vocabulary. However, for speech recognition the ϵ symbol is used to represent relations between languages with different entity length. For example, the entity "u-boat" is related to the sequence of two entities "u" and "boat". In this case, the ϵ-Removal operator will not have an impact on the result. ϵ is treated as a special symbol which is non consuming and non emitting.

Section 2.6 described a method that computes a sequence of states in topological order. Starting the processing at the top of the sequence, ensures that a state is only processed if all previous states are already evaluated that are pointing to it. This method is used in speech decoding to evaluate ϵ-paths. In contrast, the breadth-first search is used to compute emitting transitions. A technique for robust processing was described in Section 2.7. It enables the use of transducers for open vocabulary tasks. This is achieved by defining transition-dependent input- and output functions. Common data structures for automatons and transducers were discussed in Section 2.8. Section 2.9 described the use of compression methods.

Future work may investigate the parallelization of transducer operators. This may not only allow a faster processing but also the use of larger transducers with millions of words. Direct addressable compression methods for huge transducers will also become a challenging research topic. The use of in-memory technology needs to be further investigated and scaled out to large data centers. The demand of a continuous integration of new speech and language features requires an innovative software design. Reducing the resource consumption of cloud based speech services is a main optimization criteria. Although the computational power of embedded devices increased, the provision of advanced speech services remains challenging. Further research on robust parsing techniques using weighted finite-state transducers can improve the language model training by detecting and correcting wrongly written words.

3

Automatic Speech Recognition

Applications for Automatic Speech Recognition are becoming more and more popular. There are applications in the car, e.g. for voice destination entry or point of interest search by voice. Further on, voice commands can be phrased in a natural language. There are also applications in the home automation area such as smart TVs. Speech recognition is used by entertainment systems to query huge databases, e.g. for music titles or movies. A speech recognizer computes the most likely words sequence given a sequence of speech features. For this, speech features are captured and evaluated using an acoustic and a language model. An introduction in automatic speech recognition is provided in Section 3.1. The use of weighted finite-state transducers enables a modular developing process for speech recognizers on the on hand. On the other hand, the optimization of the search network in advance enables recognition on embedded devices. Section 3.2 describes the development process.

The input for any speech recognizer are speech features and a knowledge of word sequences. An introduction in audio signal processing for deriving speech features is given in Section 3.3. The representation of word sequences is described in Section 3.4. Finally, two methods are discussed for word tokenization and text formating in Section 3.5. Whereas the first defines words for recognition, the second prepares word sequences for further processing. The chapter is completed by a summary.

3.1 Conceptual framework

The fundamental formula of speech recognition computes the most likely word sequence $\underline{\hat{w}}$ over vocabulary W given a sequence of speech features $\underline{x}$. An introduction is provided, e.g. by Schukat-Talamazzini [65] or Huang et al. [66]. Let $P(\underline{x}|\underline{w})$ be the acoustic model and let $P(\underline{w})$ be the language model for a sequence of words $\underline{w} \in W^*$. A detailed description of the equation is given in Section 4.5. The speech decoding solves the following equation:

$$\begin{aligned}\underline{\hat{w}} &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} \frac{P(\underline{x}|\underline{w})P(\underline{w})}{P(\underline{x})} \\ &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} P(\underline{x}|\underline{w})P(\underline{w}).\end{aligned}$$

Speech features can be computed on various ways. In this thesis, speech features are derived from audio recordings. In general, it was also proposed to use others, such as visual features. It was also proposed to combine features. A study of audio-visual features for speech recognition is provided, e.g., by Georges [3]. The noisy portion of the audio signal was used to estimate a noise model. Faubel et al. further enhanced the idea by using a "mouth-tracking" method [4], [5]. Lee et al. [67] proposed the AVICAR corpus to evaluate audio-visual speech recognition. This set-up uses video cameras which are mounted in-front of the speaker together with a microphone array under the dash board. The experimental set-up is shown in Figure 3.1 on the left side. On the right side of Figure 3.1, a sample video captured during speech recognition is given. The video signal was used to enhanced voice activity detection. It was possible to estimate a noise profile to improve the noise reduction. It could be shown that the recognition accuracy increased, e.g. by Georges [3] and together with Faubel et al. [4], [5].

However, without loss of generality, acoustic features are used for experiments in this thesis. Today, most speech applications are using speech features derived from audio signals. The feature front end computes the speech features directly on the device. Alternatively, the audio signal is compressed and transferred to the cloud for further processing. Whether the recognizer is hosted on locally or in the cloud depends on the available computational power. The speech is recognized for a certain domain and passed to the device where it is presented to the user in a suitable way. This could be a written text or an executed command, e.g., playing a song or turning the radio on.

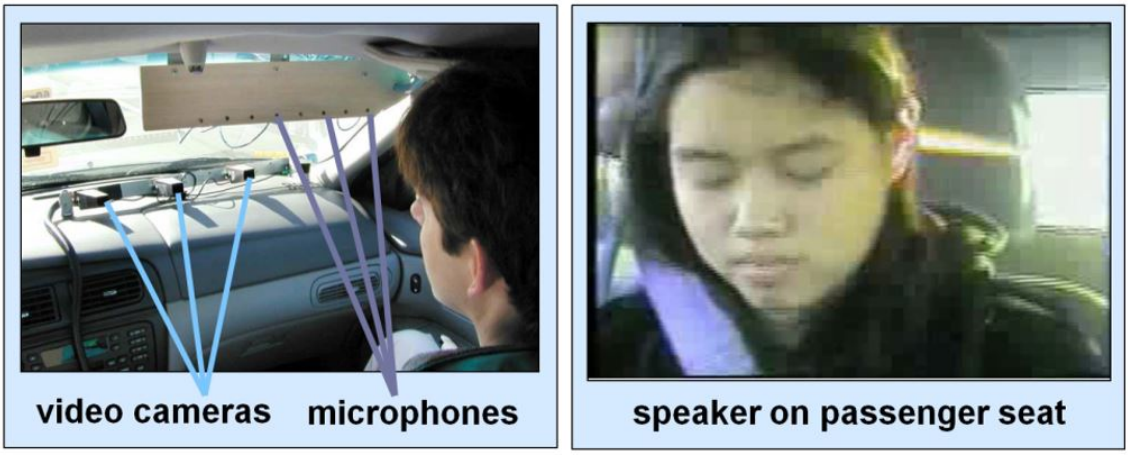


Figure 3.1: *Set-up for an automotive audio visual speech recognizer on the right side. Four video cameras are used and a microphone array with 8 microphones. On the left side is an example video caputure used to improve the speech recognition [5].*

3.2 Developing process

Developing an application for speech recognition can be separated into several modules. An overview of all modules is provided in Figure 3.2, where the process is separated into a preprocessing and a run-time part. Both, the acoustic model (AM) and the language model, represented as weighted finite-state transducer (wFST-LM) are estimated in a preprocessing step and passed to the run-time modules. Speech features are captured and evaluated by the acoustical model $P(\underline{x}|\underline{w})$ for a word sequence $\underline{w}$. It is combined with language model weights $P(\underline{w})$ during decoding. The decoder evaluates all promising word sequences and returns the most likely ones. Both, the acoustic and the language model can be developed independently for each language. This makes the concept of speech recognition suitable for industrial usage. The preprocessing is discussed in this chapter. Chapter 4 describes the run-time part together with the novel decoding technique for nested transducers using dynamic language models.

The output of a speech recognizer can be further processed. An application for message dictation will generate a well readable output. In contrast, a voice and control application will try to understand the spoken utterance. Today, the training data is derived from field data. Especially, cloud recognizers can take advantage of these data. Frequently updates of the acoustic and language model allow to train well suited models. In contrast, an embedded recognizer has quite limited update capabilities. Even though more and more devices are connected, a client update is often difficult and lim-

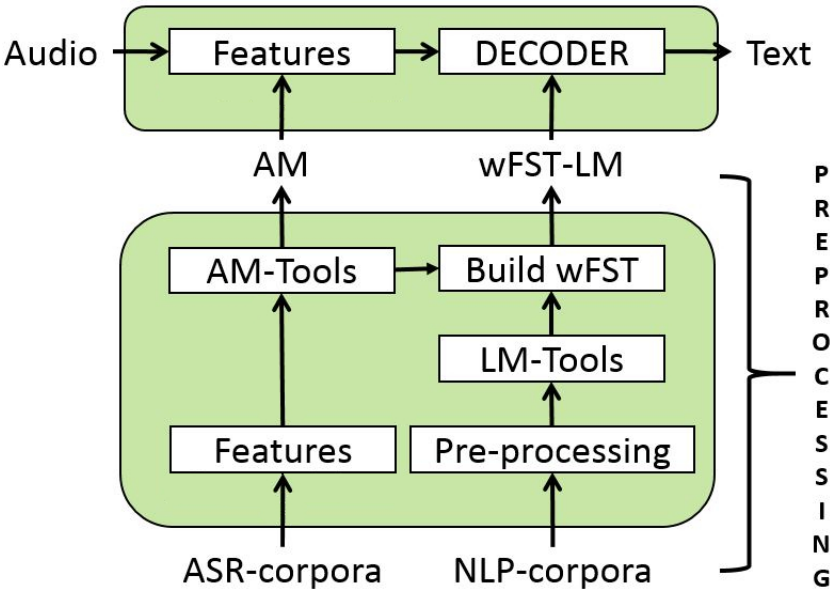


Figure 3.2: *Overview of a speech recognizer using weighted finite-state transducers.*

ited to certain aspects. Thus, the developing process for embedded recognizers differs. In cloud based systems, the task is more how to best learn from observations derived from field data. In embedded systems, the task is more how to best define a solution that matches arbitrary users expectations. In practice, embedded systems are more specified by customers.

3.3 Introduction in acoustic signal processing

Acoustic signal processing enables a robust speech recognition, also in noisy environments or environments with strong reverberation. Computing speech features is a trade-off between enhancing speech features and reducing noise. Section 3.3.1 describes the processing chain of an acoustic feature front end in general. Some portion of the chain is described more detailed afterwards. Section 3.3.2 describes the signal energy which is commonly used for voice activity detection. The MFCC features are described in Section 3.3.3. A feature reduction method is discussed in Section 3.3.4 which enables speech decoding also on embedded devices. Finally an introduction in spectral subtraction and cepstral mean normalization is given in Section 3.3.5.

3.3.1 Feature front end

The feature front-end captures the audio signal and computes a stream of speech features. A voice activity detection is used to pass speech features to the decoder. In general, Mel Frequency Cepstral Coefficients are used which are computed by the following instruction pipeline:

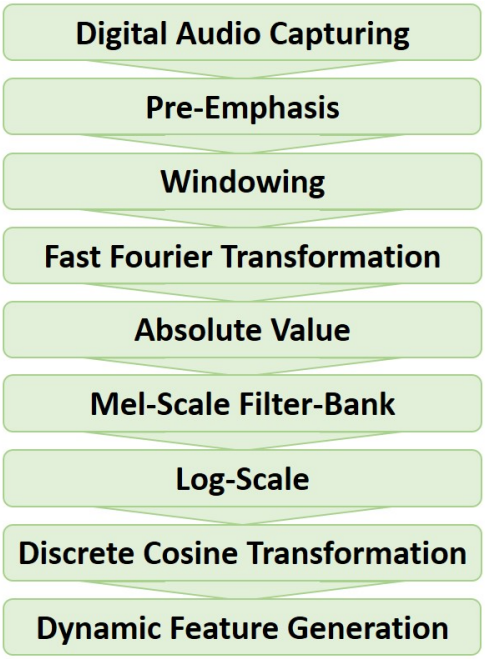


Figure 3.3: *Overview of a speech recognizer using weighted finite-state transducers.*

Hybrid speech recognition denotes a system where the computational effort is shared over one (or more) devices. Today, there are hybrid recognizers computing speech features on the device whereas the decoding happens in the Cloud. This reduces the network load on the one hand. On the other hand, it may makes a continuous integration of new features difficult. It is also common to stream only voiced signals to the cloud, e.g. detected by a local voice activity detection. In addition, the audio signal is compressed, e.g. by Speex or Opus to reduce the network load. The Speex coder is described, e.g. by Valin [68]. Terriberry et al. [69] described the Opus coder. This enables to compute speech features on the cloud.

3.3.2 Signal energy

Voiced sounds typically have a high energy. In fact, this can already be used to detect speech in an audio signal. Especially detecting the beginning and the end of speech is important for real world applications. It saves computational time on the one hand. On the other hand, it reduces the network load in case of cloud based recognition. From a speech feature perspective, the knowledge of speech and non speech sections in a feature stream is used to enhance the noise reduction. The short-time energy also enables a hyphenation as illustrated by Schukat-Talamazzini [65]. In general, the long-time energy of a signal f_n is given by

$$E = \sum_{n=-\infty}^{\infty} |f_n|^2 \, .$$

It derives a short-time energy at sampling time m. It is defined as follows:

$$E^{(m)} = \sum_{n=0}^{N-1} \alpha_n |f_{m+n}|^2 \, ,$$

where $\alpha_n = \omega^2_{-n}$ represents a window function. Typically, a hamming-window is used. It is defined as follows:

$$w(n) = \alpha - \beta \cos \frac{2\pi n}{M-1},$$

with $n = 0..M-1$ being M the window length and n the current index. A Von-Hann window is specified by $\alpha = \beta = 0.5$. Energy-based voice activity detection is realized by a thresholding. Voiced sound is detected when the signal energy E_t at time t exceeds the threshold α. This method is suited for most common speech recognizers. This is formalized as follows:

$$(E_t > \alpha) \, ? \, \text{speech} : \text{non speech} \, .$$

Voice activity detection was also discussed by Rabiner et al. [70] by using zero-crossing rates. Dong et al. proposed an enhanced version using a spectral noise adaptation method [71]. Other methods are proposed, e.g. by Garner et al. [72] Real-time implementations are proposed, e.g. by Moattar et al. [73].

3.3.3 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCC) features are frequently used for speech recognition. The computation is done along several stages. An introduction is provided, e.g. by Davis et al. [74], Schukat-Talamazzini [65] or Huang et al. [66]. The MFCC speech feature was introduced in section 3.3.1. The audio signal is captured and preprocessed following by a discrete Fourier transformation. This is formalized for a windowed short-time signal X on signal x:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi n \frac{k}{N}} \text{ for } 0 \leq k < N \ .$$

Usually a 25ms hamming window is used. This is shifted once in 10ms. The construction of the speech feature is derived from the human audio processing. For example, the human ear has a non-linear pitch perception characteristics. This is technically realized for speech features by a non-linear scaling of the frequency axis. These scales can be applied by a non-linear filter bank. The average spectrum around a center frequency with increasing bandwidth is computed via a filterbank with M filters $m = 1, 2, .., M$. Where filter H_m is triangular and defined as follows:

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m-1] \leq k \leq f[m] \\ \frac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m+1]-f[m])} & f[m] < k \leq f[m+1] \\ 0 & k > f[m+1] \end{cases} .$$

Where $f[m]$ marks the boundaries which are uniformly spaced in the mel-scale. The lowest cut off frequency is typically about $f_l = 300$Hz. The highest cut off frequency of is typically about $f_h = 3400$Hz. Actually, it depends on the sampling rate of the signal. $f[m]$ is given by

$$f[m] = \frac{N}{F_s} B^{-1} \left(B(f_l) + m \frac{B(f_h) - B(f_l)}{M+1} \right) ,$$

where the sampling frequency is denoted by F_s. The discrete Fourier Transformation uses a size of N. The mel-scale B^{-1} is defined as follows:

$$B^{-1}(b) = 700(e^{\frac{b}{1125}} - 1) .$$

The log-energy $S[m]$ at the output of filter m is computed as

$$S[m] = \ln\left[\sum_{k=0}^{N-1} |X_q[k]|^2 H_m[k]\right] \text{ for } 0 \leq m < M .$$

Note that S is even. The mel-frequency cepstrum c is computed via the discrete cosine transformation of the M filter outputs of S multiplied by two and defined as follows:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\pi n \frac{m - \frac{1}{2}}{M}\right) \text{ for } 0 \leq n < M .$$

The Discrete cosine transformation reduces the dimension while decorrelating the components. Typically the first 13 cepstrum coefficients are used for speech recognition. The MFCC transformation is a non homomorphic transformation due to the used log-energy filter $S[m]$. However, using a smoothed transfer S' will result in an approximately homomorphic representation. The disadvantage using S' is an increased sensitivity for noise and spectral estimation errors as discussed, e.g. by Huang et al. [66]. Using S' leads to a homomorphic transformation and is defined as:

$$S'[m] = \sum_{k=0}^{N-1} \ln\left[|X_q[k]|^2 H_m[k]\right] \text{ for } 0 \leq m < M .$$

An illustration in Figure 3.4 of MFCC features for the spoken number sequence 1 to 9 was provided by Georges [3]. The first row shows a plot of the original signal. Followed by its log- and mel-log-spectrum. At the bottom is the final MFCC speech feature stream. This stream is finally used to compute the emission probabilities from the acoustic model. Temporal changes in the cepstra of a speech signal can be used for speech recognition as introduced by Schukat-Talamazzini [65] or Huang et al. [66]. This can be achieved by a derivations of consecutive features or by a linear discriminant analysis as described in the following.

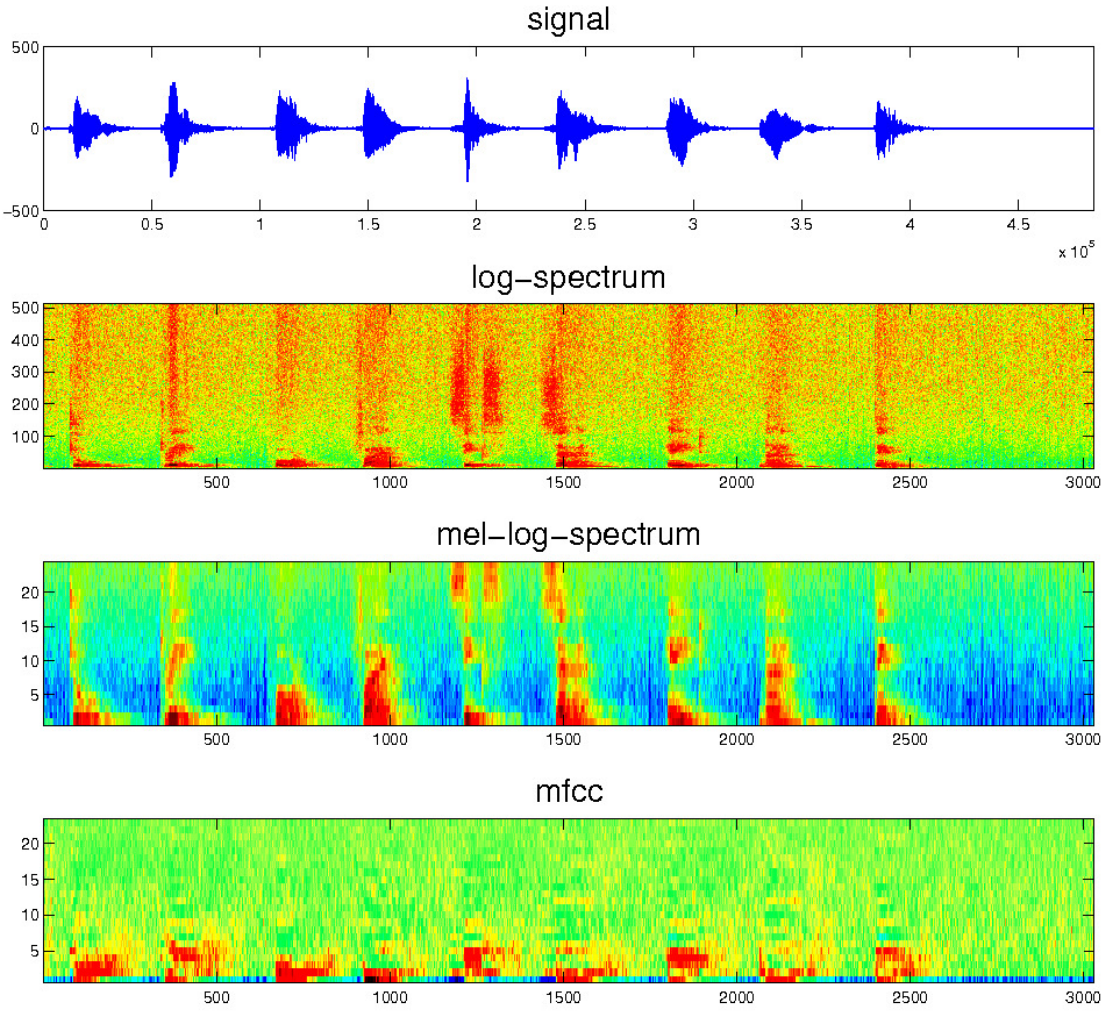


Figure 3.4: *MFCC example for the digits one to nine [3].*

MFCC $+\Delta+\Delta\Delta$

Typically the first and second derivation over a time of 40 ms is used. Δ denotes the first derivative and $\Delta\Delta$ the second derivative. The overall "MFCC $+\Delta+\Delta\Delta$" feature vector has a dimension of 39. A derivative of higher order does not further improves speech recognition, significantly. The derivatives are approximated by a filter kernel F_Δ. An example kernel is defined as follows:

$$F_\Delta = \begin{pmatrix} -2 & -1 & 0 & 1 & 2 \end{pmatrix}$$

MFCC and LDA

The temporal behavior of adjacent MFCC vectors can be trained by a linear discriminant analysis. An introduction of the analysis in general is given in the following section. In contrast to derivative based features, an additional training step is required. During training significant characteristics over adjacent MFCC vectors are learned. Typically, all MFCC features within a 40 ms window length are used.

3.3.4 Linear discriminant analysis

Linear Discriminant Analysis makes use of a class affiliation as proposed by Fisher [75]. An introduction is provided, e.g. by Duda et al. [76]. All classes should have comparable variances. An alternative method is the principle component analysis. Let K be the classes with mean vector $\underline{\mu}_k$ where $k \in K$. Further let $\underline{\mu}$ be the overall mean of the signal. The impact of each class, depending on the amount of available training data is given by p_k. $\underline{\mu}_k$ and p_k is computed for a feature sequence $\underline{x}$ as follows:

$$\underline{\mu}_k = \frac{1}{N_k} \sum_{l=1}^{N_k} x_{i,k} \ ,$$
$$p_k = \frac{N_k}{\sum_{l=1}^{K} N_l} \ .$$

Where N_k denotes the number of features in class k. The between class scatter matrix S_b and the within class scatter matrix S_w are defined as follows:

$$S_b = \sum_{k=1}^{K} p_k (\underline{\mu}_k - \underline{\mu})(\underline{\mu}_k - \underline{\mu})^t \qquad \text{and } S_w = \sum_{k=1}^{K} p_k \Sigma_k$$
$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N_k} (\underline{x}_{i,k} - \underline{\mu}_k)(\underline{x}_{i,k} - \underline{\mu}_k)^t \ .$$

This maximizes the class separability and keeps the class variances roughly constant at the same time. This optimization problem is solved by the following equation:

$$S_b \underline{\phi}_i = \lambda_i S_w \underline{\phi}_i \ .$$

3.3.5 Spectral subtraction and cepstral mean normalization

There are many proposals to reduce noise form the signal and to enhance the speech features at the same time. Here, the spectral subtraction and cepstral mean normalization are discussed. Some more methods, e.g. the vocal tract length normalization is introduced, e.g. by Huang et al. [66].

Spectral subtraction

Spectral subtraction was proposed, e.g. by Boll et al. [77]. It assumes that the desired signal X is distorted by uncorrelated additive noise V captured by microphone m. The distorted power spectral density Y can be approximated by

$$|Y_m(\omega)|^2 \approx |X_m(\omega)|^2 + |V_m(\omega)|^2 \ .$$

The noise power spectrum $\hat{V}_m(k, f)$ can be estimated from the non voiced speech feature stream. A voice detection method is used as described previously in Section 3.3.2 The power spectrum estimation is non-linear. Lockwood et al. [78] proposed a heuristic method. An estimation using a design parameter γ is defined as follows:

$$|\hat{V}_m(\omega)|^2_{NL} = \frac{\max\limits_{\text{over } K \text{ frames}} |\hat{V}_m(\omega)|^2}{1+\gamma SNR(\omega)} \ .$$

where $SNR(\omega)$ denotes the signal to noise ratio of ω. Finally, the desired signal $\hat{X}$ can be estimated by

$$|\hat{X}_m(\omega)|^2 = \max\left\{|Y_m(\omega)|^2 - \alpha|\hat{V}_m(\omega)|^2_{NL}, \beta \geq 0\right\} \ .$$

where α is the overestimation factor. β is a spectral floor of the non-linear spectral subtraction as outlined, e.g. by Wölfel et al. [79].

Cepstral mean normalization

Cepstral mean Normalization is used for compensating channel effects, e.g. outlined by Atal [80]. Different microphones or encodings in the audio processing change the transfer function. Those slight differences in the channel are summarized as channel effects. Even if the same microphone is used, the transfer function varies over time depending on the distance to the microphone and depending on the room acoustics, e.g. described by Huang et al. [66]. Wölfel et al. [79] proposed to use as cepstral mean normalisation to suppress short-time convolutional distortions for distance speech recognition. This method is closely related to spectral subtraction, which subtracts the estimated noise from the signal in the frequency domain. The method has to be applied on the speech feature stream during acoustic model training as well as during recognition in the run-time part. The cepstral mean $\overline{x}$ is computed on the voiced speech feature stream $\underline{x}$ as follows:

$$\overline{x} = \frac{1}{T} \sum_{t=0}^{T-1} x_t \ .$$

The cepstral mean normalized feature vector $\hat{x}$ is finally computed by:

$$\hat{x}_t = x_t - \overline{x} \ .$$

The normalization is applied for each utterance, independently or for each speaker individually. Each utterance should be long enough to capture sufficient phonetic variability and short enough to get along with non-stationary effects of the channel impulse response. The signal should include about two seconds of voices features.

3.4 Language representation

A language can be described as a sequence of concatenated words $\underline{w} = w_1 \cdot w_2 \cdot ... \cdot w_n$, where "·" denotes a delimiter. Usually a whitespace delimiter is used for processing text, e.g. written in German or English. A speech recognizer usually interprets this delimiter as an optional silence phase between spoken words. Let $L_1 \subseteq V^*$ and $L_2 \subseteq V^*$ be two sets of regular languages over vocabulary V. Where "*" denotes the Kleen star operator as introduced, e.g. by Schöniger [11]. V^* denotes the set of all word sequences that can be build using words from the vocabulary V. Further, the union and intersection of two languages are defined as follows:

$$L_1 \cup L_2 = \{\underline{w} | \underline{w} \in L_1 \vee \underline{w} \in L_2\} \qquad \text{union}$$
$$L_1 \cap L_2 = \{\underline{w} | \underline{w} \in L_1 \wedge \underline{w} \in L_2\} \qquad \text{intersection.}$$

The languages L_1, L_2 can be parsed by a finite-state automaton. For example, there is a finite-state automaton which accepts telephone numbers. A telephone number is valid or not and hence a binary decision. Those languages are well representable by grammars. Section 3.4.1 introduces grammars. In general, the assessment whether a word sequence is valid or not depends on various constrains. For example, a phrase can depend on the context and content of the discussion. Also the speakers knowledge, e.g. over the vocabulary can make a difference. In such cases, it is established to use statistical models which are trained on some train data. The model is then used to compute the probability of a word sequence. Here, models based on N-gram Markov language models are discussed. An introduction is provided in Section 3.4.2

The generalization capability of such models have a significant impact on the recognition of sentences which were not seen previously. Section 3.4.3 describes the use of classes to generalize on the one hand. On the other hand, the technique enables speech dictation with large vocabularies on embedded devices. The combination of statistical language models is described in Section 3.4.4. It is used to shift the probability distribution of a language model to the one of a different model, e.g. a uniformly distributed 0-gram. Statistical estimators for N-gram Markov models are introduced in Section 3.4.5. An introduction in back-off language models is given in Section 3.4.6. Finally, an overview of combination methods for grammars and statistical models is given in Section 3.4.7.

3.4.1 Grammars

Whereas grammars were investigated for speech dictation in the past, its usage today focuses mostly on recognizing structured command like utterances. This includes heavy content domains such as voice destination entry. It was proposed to use grammars for error correction. Grammars are also used for part of speech tagging, e.g. proposed by Brill [81]. Furthermore, applications for natural language understanding are using grammars as well as full text search methods. A grammar was also used in this thesis to prepare the train corpora for estimating dynamic language models. Chomsky investigated a formal language theory using grammars, e.g. proposed by Chomsky [82], [83] or Aho et al. [84]. Chomsky defined 4 types of grammars:

- phrase structured grammars with no restrictions
- context sensitive grammars
- context free grammars
- regular grammars

Regular grammars are used for speech recognition although context free grammars were proposed to represent linguistic knowledge. An algorithm is described by Jurafsky et al. for using probabilistic early parser and a stochastic context free grammar to generate word transition probabilities at each frame for a Viterbi decoder[85]. Duckhorn et al. proposed an extension to weighted finite-state transducers in order to enable them to model context-free grammars [86]. Regular grammars have become established for recognizing speech from heavy domains like addresses or media entries, e.g. proposed by Junqua [87].

Let G be a regular grammar. It is defined by a 4-tuple:

$$G = (V, \Sigma, P, S)$$

with following entities:

V	is the set of variables
$S \in V$	are the start variables
Σ	is the set of non-terminals with $V \cap \Sigma = \emptyset$
P	is the set of rules

The set of variables is denoted as vocabulary for natural language processing. The set of rules P is defined as

$$P : (V \cup \Sigma)^{+} \times (V \cup \Sigma)^{*}$$

It describes all possible paths as introduced, e.g. by Schöninger [11]. Grammars can be created by human experts or learned automatically, e.g. by using structured databases. This technique was described in Section 2.4 for constructing a grammar for voice destination entry. It is also common to use grammars for generating syntactic data in an early stage of a data driven model generation process. The data is used to estimate statistical language models including already available field data or data from a project survey. This developing process is established for both, estimating models for natural language understanding and speech recognition.

Many speech applications are using grammars to recognize command like speech, e.g. in car navigation systems, home automation systems or smart televisions. All commands are well known and specified by language experts accordingly to customer's specifications. A set of Voice Commands like "*start navigation*", "*turn the light on*" or "*play radio*" can be directly represented as single rules. Grammars for speech recognition are often characterized by a small set of non-terminal symbols combined with a huge set of terminals. Such a grammar for voice destination entry is presented in Table 3.1. Note that the grammar is simplified. The concrete dependency between streets

Table 3.1: *Overview of grammars in Backus-Naur Form*

⟨*address*⟩	::=	[⟨*number*⟩] ⟨*street*⟩ [[⟨*city*⟩] ⟨*state*⟩] | ⟨*city*⟩ ⟨*state*⟩
⟨*number*⟩	::=	1 | 2 | 3 | ... | 99999
⟨*street*⟩	::=	list of streets
⟨*city*⟩	::=	list of city
⟨*state*⟩	::=	list of states

and cities is not conducted. Voice Destination Entry (VDE) has a manageable structure with just a few non-terminals. For US it is expected that a user utter an address always by almost the same semantic entities:

<number>, <street>, <city>, <state>.

The non-terminal "<street>" can be derived in a huge list of all possible streets in the specified city. Grammars are also used to compactly represent large structured databases. For example, all telephone numbers in the US for Voice Activated Dialing or to represent postal codes as introduced in Section 2.2. Many speech dialog systems are using grammars where rules are activated depending on the current dialog stage. Rule activation is also used to improve speech recognition by reducing the language confusion for certain use cases. A taxi driver will most likely search streets in the same city. Hence, only streets in a dedicated city are active during speech recognition.

A technique is described, e.g. by Angluin [88] to learn regular sets from queries and counter examples. Firoiu et al. proposed to learn regular languages based on positive evidence [89]. There are also proposals to learn non deterministic finite automaton, e.g. by Garcil et al. [90] and for residual finite-state automaton, e.g., by Denis et al. [91]. There are also a set of papers using unsupervised learning methods, e.g., Klein et al. describe an approach to syntactic analyses of natural language text [92]. The use of a history was approached by Feili et al. [93]. Li et al. proposed a transformation-based algorithm for learning complex regular expressions for information extraction [94]. An overview of unsupervised grammar inference systems is provided by Roberts et al. [95] and Clark et al. [96].

3.4.2 N-gram Markov language model

Markov[97] proposed a letter n-gram model that was later used by Shannon et al.[98] to estimate word n-gram models for English. An n-gram language model is an approximation for the probability distribution $P(\underline{w})$ for word sequences $\underline{w} = w_1 ... w_n$. It is used in many language applications such as speech recognition, information retrieval and genre detection or language identification. Examples are summarized, e.g. by Toby Norvig [99]. An historical overview of statistical methods for speech recognition is provided, e.g. by Jelinek [100] or Rosenfeld [101]. Computing the probability of a word w_i in a sequence of words $w_1 ... w_{n-1}$ allows the speech recognizer to decide whether to consider a word for further computation or not. In this way the computational effort is reduced on the one hand. On the other hand it also reduces the confusion between words that are acoustically similar.

Starting from $P(\underline{w})$, a N-gram Markov model can be derived as follows:

$$\begin{aligned}
P(\underline{w}) &= P(w_0 w_1 ... w_n w_{n+1}) \\
&= P(w_0)P(w_1|w_0)P(w_2|w_0 w_1)\cdots P(w_n|w_1 w_2 ... w_{n-1}) \\
&= \prod_{i=1}^{|\underline{w}|} P(w_i|w_1 ... w_{i-1}) \\
&\approx \prod_{i=1}^{|\underline{w}|} P(w_i|w_{i-n+1} ... w_{i-1})
\end{aligned}$$

This approximates a n-gram model. A 2-gram language model is defined as follows:

$$P(\underline{w}) \approx \prod_{i=1}^{|\underline{w}|} P(w_i|w_{i-1})$$

Each sentence $\underline{w} = w_1 ... w_n$ is enclosed by the SGML tag <s> and <\$s$>$. The vocabulary is $W \cup \{<s>, <\backslash s>\}$. This enables the computation of the very first word of each sentence:

$$P(w_i|w_0) = P(w_i|\text{<s>}),$$

as well as the very last word of a sentence:

$$P(w_{n-1}|w_n) = P(<\backslash s>|w_n),$$

On the other hand it ensures that the probability is normalized,

$$\sum_{\underline{w}} P(\underline{w}) = 1.$$

The corpora should be big enough for an reliable estimation of all probabilities. Franz et al.[102] described the 13 unique million n-gram model from a trillion words web collection. The great amount of language variability made speech modeling to a data sparsity problem. In practice, a cut-off frequency is used to reduce wrong estimates due sparse data and noise. The cut-off is applied on n-grams. For example w_i should occur at least one time, a word sequence $w_{i-1} w_i$ should occur at least two times and so on. Handling the data sparsity problem is important.

The language model can be represented as weighted finite-state transducer. This enables to use nesting techniques during decoding as proposed, e.g. by Georges et al. [9]. It is used to combine models from various domains, e.g. to include user content. An overview of combination methods is given in Section 3.4.4. It was also proposed to influence the language model by statistics gathered from the recognized text itself. Such a cache based language model was proposed, e.g. by Kuhn et al. [103]. Guthrie et al.[104] examine the use of Skip-n gram models to overcome this problem. A k-skip n-gram is the set of following word sequences:

$$\{w_{i_1}, w_{i_2}, ..., w_{i_n} | \sum_{j=1}^{n} i_j - i_{j-1} < k\}$$

Information retrieval methods are used to determine suitable in-domain models, e.g., proposed by Mahajan et al. [105]. An introduction in information retrieval with statistical language models is provided, e.g. by Kalt [106], Ponte [107] or Liu et al. [108]. Basically, a statistical language model is estimated for every single document in a collection. During retrieval, the probability of the search term is estimated for each of these models. The document which is assigned to the model that achieved the highest probability on the search term is proposed to be the best result. Overall, the result is a ranked list of retrieved documents according to their achieved probabilities on the search term. The order of words in information retrieval is less important so that most often 1-grams are used together with a Bernoulli or multinomial process to generate the query. An introduction is given, by Salton et al. [109] or Baeza-Yates et al.[110]. A bag of word features are proposed, e.g. by Harris et al.[111]. This can be combined with a vector space model which is commonly used in information retrieval.

Cavnar et al.[112] proposed a text categorization. This approach was later adapted for language identification, e.g. by Vatanen et al. [113]. The factors that determine the performance of text-based language identification was investigated, e.g. by Botha et al.[114]. In principle, a statistical letter model is estimated for each language and evaluated on the test text. The highest probability is most likely achieved by evaluating the test text with the correct language. Using letter n-gram were also proposed for named entity recognition, e.g. by Klein et al. [115]. In general, N-gram models for text genre detection is described, e.g. by Kessler et al. [116].

3.4.3 Class based language models

Linguistic knowledge or statistical clustering methods can be used to group words. Each group is assigned to a class so that the language model is estimated on classes. Instead of estimating the model on words, the class based n-gram model is estimated on word-classes, e.g. a sequence of classes. Each word w_i is assigned to a class C_i with a probability $P(w_i|C_i)$. An N-gram Markov model is then defined as follows:

$$P(w_i|C_{i-n+1}...C_{i-1}) = P(w_i|C_i)P(C_i|C_{i-n+1}...C_{i-1}),$$

where $C_{i-n+1}...C_i$ is an N-gram over seen classes. Each class includes a set of words. The probability $P(\underline{w})$ for a word sequence $\underline{w}$ is computed by:

$$P(\underline{w}) = \sum_{c_i..c_n} \prod_{i=1} P(w_i|c_i)P(C_i|C_{i-n+1}...C_{i-1}).$$

A Bayesian formulation of a many-to-many mapping between words and classes for class based language models is formulated by Su [117]. This can be simplified for many applications when a word is uniquely assigned to one class, e.g. the class for proper names, the class of nouns or the class derived from syntactic-semantic word relations. The equation simplifies with the assumption that each w_i is uniquely assigned:

$$P(\underline{w}) = \prod_{i=1} P(w_i|c_i)P(C_i|C_{i-n+1}...C_{i-1}).$$

The word to class mapping can be learned automatically or derived manually by human expert knowledge. Word class n-gram models are often used in applications such as travel information systems or embedded media search. In general, most heavy content domain recognizers are using classes in the one or the other way. An efficient class based language model for very large vocabularies was proposed, e.g. by Whittaker et al. [118]. In addition to the generalization methods available for word n-gram models, the word class n-gram model can generalize by the content of each class. Especially, it enables to change the content of certain classes on the fly, e.g. to add user content. These models are also used for dialog systems where the classes are filed on-the-fly depending on the current dialog stage. Word-class n-gram models are also used in embedded dictation. Both, estimation and evaluation during speech decoding requires less computational effort compared to word n-gram models. Smoothing, backing-off

and interpolation methods can be applied on class based language models in a similar way. The class based n-gram model is a special case of the dynamic language model proposed in Section 4.3.3. In contrast a dynamic language model can enclose grammatical structures in classes instead of word lists.

A different kind of class n-gram model was proposed by Bahl et al. [119]. It assumes that the word probability of w_i depends on equivalence classes $E(w_1 ... w_{i-1})$ of preceding words $w_1 ... w_{i-1}$. These equivalence classes are derived from decision trees. For long-distances an entropy criterion can be used to develop the tree. In this way, a long-distance n-gram language model can be built as follows:

$$P(\underline{w}) = \prod_{i=1} P(w_i | E(w_1 ... w_{i-1})).$$

An introduction in word class n-gram models and clustering methods is provided e.g., by Brown et al. [120]. Detailed experiments with class based n-gram model were presented, e.g. by Moisa et al. [121] or Jardino [122]. Using corpus-based clustering algorithms to assign word classes was proposed, e.g., Beaujard et al. [123]. It takes a word context similarity criterion into account. Samuelsson et al.[124] used part-of-speech statistics to estimate a class based language model. In this thesis, semantic entities are used to group certain word sequences. For example proper names and identifiers for streets, states or city names are grouped for applications such as voice destination entry. A model M approach with classes was proposed by Ahmad et al. [125]. A hierarchical class n-gram model was proposed by Zitouni et al. [126]. Related is also the work from Yamamoto et al.[127] who used multiple word clusters for a multi class composite n-gram language model.

3.4.4 Interpolation and adaptation of n-grams

A huge amount of text data is required to estimate n-gram probabilities. Interpolating an n-gram language model with an $(n-1)$-gram one assumes that $(n-1)$-grams suffer less from data sparseness. Actually, it is always possible to assume a uniformly distribution for $n = 0$ if no data is available. In this way, interpolation can be used as a smoothing technique, e.g. as proposed by Katz [128], Martin et al. [129]. A summary of interpolating n-gram models is provided by Goodman et al.[130].

Let $P_{li}(w_i|w_{i-1})$ be a combination of a 2-gram language model $P(w_i|w_{i-1})$ and a 1-gram language model $P(w_i)$:

$$P_{li}(w_i|w_{i-1}) = \lambda P(w_i) + (1-\lambda)P(w_i|w_{i-1}),$$

with $0 \leq \lambda_i \leq 1$. A $\lambda = 1$ considers just 1-gram probabilities whereas 2-gram is considered for $\lambda = 0$. This method is often denoted as deleted interpolation. Generally, the interpolation of k probability distributions can be formalized for $0 \leq \lambda_i \leq 1$ as follows:

$$P(w_n|w_{n-k}...w_{n-1}) = \sum_{i=1}^{k} \lambda_i P_i(w_n|w_{n-i}...w_{n-1}).$$

The following criteria ensures that the language model remains normalized:

$$\sum_i \lambda_i = 1.$$

It is also common to define history dependent interpolation weights, such as $\lambda_{w_{i-n+1}...w_{i-1}}$. Usually, these are summarized in classes similar to the history-class-based language model as described in previous Section 3.4.3. A recursive interpolation from higher-order n-gram models to lower-order one can be formulated as follows:

$$\begin{aligned} P_l(w_i|w_{i-n+1}...w_{i-1}) = &\lambda_{w_{i-n+1}...w_{i-1}} P(w_i|w_{i-n+1}...w_{i-1}) \\ &+ (1-\lambda_{w_{i-n+1}...w_{i-1}}) P_l(w_i|w_{i-n+2}...w_{i-1}) \end{aligned}$$

Language model interpolation is an integral part of the developing process. It is used to create models with certain properties. For this, it combines several domain specific language models which were estimated from domain specific text corpora. For example, there might be a corpus for short message dictation and one for search queries. Both can be combined for a voice search with natural queries. Bigi et al.[131] proposed to use information retrieval methods for interpolating various language models. In practice, it is common to determine the weights automatically, e.g. proposed by Jelinek et al. [132]. The expectation maximization algorithm can be used but there are also alternatives, e.g. proposed by Chen et al. who used the Powell's algorithm [133]. The Bayesian interpolation method determines the weights automatically and considers the current history in the word sequence on–the–fly. Interpolation weights λ_i can be adjusted domain dependent, e.g. by human experts.

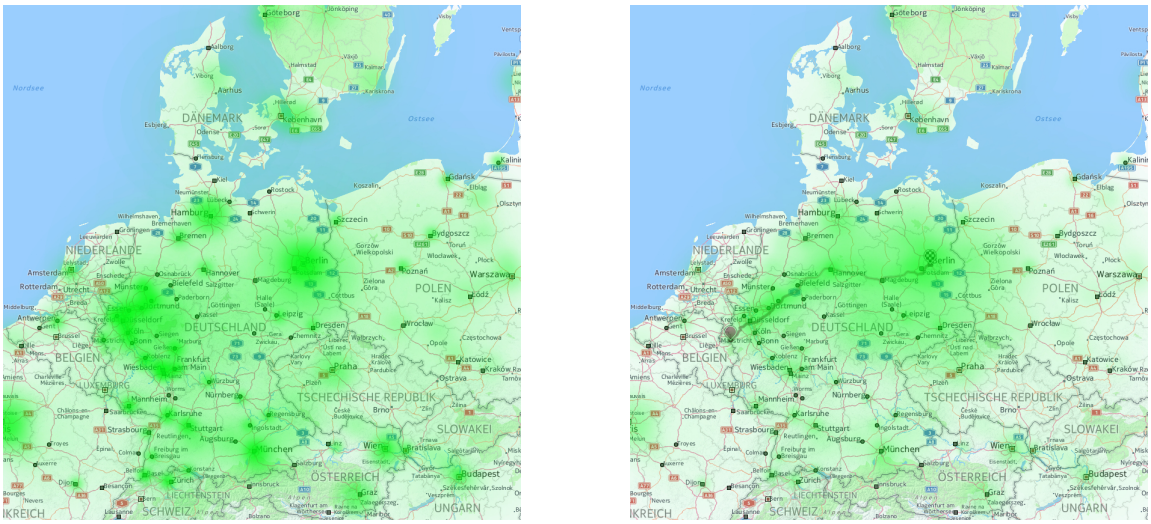


Figure 3.5: *Motion adaptive speech recognition for enhanced voice destination entry. On the left side a schematic example distribution for a traditional language model. On the right side a motion adapted model along a users route.*

Dupont et al.[134] explores a method to interpolate word and class based language models. Klakow[135] proposed a log-linear interpolation for combining models of different context size. Further experiments were reported by Maltese et al. [136]. An overview of major approaches in language model adaptation is provided by Bellegarda et al. [137]. Bilmes et al proposed an online adaptive learning for speech decoding [138]. Various interpolation methods for a variety of recognition tasks was explored by Allauzen et al. [139]. Using grammars for adaptation was also provided by Wu [140]. Methods to combine grammars and statistical language models are summarized in Section 3.4.7. In this thesis, a method is proposed which enables the use of user knowledge during speech decoding. A dynamic language model is used as described in 5.3.2.

Location aware speech recognition

An application for interpolating statistical language models is address entry by voice which is challenging in two aspects: First, the great amount of postal addresses (10M+ for US) and point of interests (3M+ for US) requires a speech recognizer to operate close to the acoustical resolution limit. Second, the sparse amount of training data makes it nearly impossible to estimate reliable statistical models to constrain the search space by incorporating syntactical knowledge. A static system has to make a trade-off between dialog stages and destination coverage to achieve a usable accuracy.

Both have a significant impact on the users experience using voice for destination entry. Continuously interpolation of statistical language models can close the gap by analyzing the motions profile of the speaker as proposed by Georges et al. [1]. For this, statistical language models were estimated for every region. Let $\underline{\lambda}$ be the continuous interpolation weight vector with $\sum_c^{\text{cities}} \lambda_c = 1$. The statistic language model is then defined as follows:

$$P_{\text{poi}}(w_i|w_{i-1}) = \sum_c^{\text{cities}} \lambda_c \cdot P_c(w_i|w_{i-1}),$$

The current state of the art system can be simulated by a uniformly interpolation of all these models. Activation and deactivation capability is achieved by binary interpolation weights, e.g. the current city has a weight of 1 - activated - whereas all others are 0 weighted - disabled:

$$\underline{\lambda} = (0,...,0,1,0,...,0)^T$$

The continuous interpolation weight vector is based on the speakers motion profile. This profile is estimated from various sensors, such as space-based satellite navigation systems, acceleration sensors, speed measurements, altimeters, etc. The information is analyzed together with a default map to generate an aligned weight vector with the set of available statistical models. Figure 3.5 illustrates the use of location aware speech recognition. The following examples shows the advantage and disadvantage of this novel technique. Les "cities" be the set of all cities in Germany and let M be the German city "Munich". Assuming the motion profile indicates a speaker in Munich would result in the following interpolation weight vector:

$$\lambda_{\text{M}} + \sum_{c \neq M}^{\text{cities}} \lambda_c = 1 \text{ with } \forall \lambda_i . i \in \text{cities} \wedge i \neq M \wedge \lambda_i < \lambda_{\text{M}}.$$

Hence, the probability of words $\underline{w}$ related to Munich $P_M(\underline{w})$ is boosted compared to a city-unified model. The perplexity and word error rate will decrease given a correctly assigned interpolation weight vector:

$$\lambda_{\text{M}} \cdot P_M(\underline{w}) + \sum_{c \neq M}^{\text{cities}} \lambda_c \cdot P_c(\underline{w}) >> \sum_c^{\text{cities}} \frac{1}{\#\text{cities}} \cdot P_c(\underline{w})$$

3.4.5 Statistical estimators

The conditional probability can be estimated from a text corpora, e.g. scraped from the world wide web, or transcribed field data by the maximum likelihood estimation [141]. The conditional probability can be estimated from a text corpora, e.g. scraped from the world wide web, or transcribed field data by computing

$$P(w_i|w_{i-n+1}...w_{i-1}) = \frac{P(w_{i-n+1}...w_{i-1}w_i)}{P(w_{i-n+1}...w_{i-1})} \qquad \text{for an n-gram model}$$
$$P(w_i|w_{i-2}w_{i-1}) = \frac{P(w_{i-2}w_{i-1}w_i)}{P(w_{i-2}w_{i-1})} \qquad \text{for a 2-gram model}$$

from a corpora,e.g. by a maximum likelihood estimator as described in Section 3.4.5. The probabilities can be estimated by computing word frequencies:

$$P(w_i|w_{i-n+1}...w_{i-1}) = \frac{F(w_{i-n+1}...w_{i-1}w_i)}{F(w_{i-n+1}...w_{i-1})} \qquad \text{for an n-gram model}$$
$$P(w_i|w_{i-2}w_{i-1}) = \frac{F(w_{i-2}w_{i-1}w_i)}{F(w_{i-2}w_{i-1})} \qquad \text{for a 3-gram model}$$

A class based model is estimated by assigning each word to classes. The word class n-gram model can be estimated by computing the frequency $F(\cdot)$ for words and classes:

$$\begin{aligned} P(w_i|w_{i-n+1}...w_{i-1}) &= P(w_i|C_i)P(C_i|C_{i-n+1}...C_{i-1}) \\ &= \frac{F(w_i)F(C_{i-n+1}...C_i)}{F(C_i)F(C_{i-n+1}...C_{i-1})} && \text{for a n-gram model} \\ &= \frac{F(w_i)F(C_{i-1}C_i)}{F(C_i)F(C_{i-1})} && \text{for a 2-gram model.} \end{aligned}$$

An detailed introduction is also provided, e.g. by Manning et al. [142] together with alternative estimators. However, there are also seldom word sequences which are not belonging to one certain class like media titles. Smoothing techniques are used to enhance the probability of unseen sequences. Goodman [143] and Chen et al.[133] gave an overview on smoothing techniques for language modeling. Jeffreys [144] formalized the add-one smoothing method that was recently revisited by Chopin et al. [145]. This was used, e.g., by Jelinek et al. for speech recognition [146]. n-gram add-one estimator

for a vocabulary size V as described, e.g., by Manning et al. [142]:

$$P_{\text{add-one}}(w_1..w_n) = \frac{F(w_1..w_n)+1}{N+V^n}$$

Lidstone et al. [147] proposed a lambda value instead of just adding one:

$$P_{\text{Lid}}(w_1..w_n) = \frac{F(w_1..w_n)+\lambda}{N+V^n\lambda}$$

where λ is usually smaller 1 and N being the number of overall words. The add-one method is available by $\lambda = 1$. Box et al.[148] proposed the expected likelihood estimation with $\lambda = \frac{1}{2}$. However, this method can be seen as a linear interpolation of a maximum likelihood estimation and a uniform prior as shown by Johnsons et al. [149]:

$$P_{\text{Lid}}(w_1..w_n) = \mu\frac{F(w_1..w_n)}{N} + (1-\mu)\frac{1}{V^n},$$

with μ computed by:

$$\mu = \frac{N}{N+V^n\lambda}.$$

Witten et al. proposed to shift the probability mass depending on the context of the word [150]. Let $T(w_{i-1})$ be the number of different words preceding w_i and $N(w_{i-1})$ its overall count. With $Z(w_{i-1})$ the number of 2-grams in some held-out data the estimator is described as follows:

$$P_{\text{WB}}(w_i|w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N+T(w_{i-1}))} \qquad \text{if F(w\_iw\_i-1)=0}$$

$$P_{\text{WB}}(w_i|w_{i-1}) = \frac{F(w_{i-1})}{N(w_{i-1})+T(w_{i-1})} \qquad \text{if F(w\_iw\_i-1)>0.}$$

This method performs well as long as there is enough data for training available. Church et al. described the Good-Turing estimation [151] There are lot of other language models available like the absolute and linear discounting that was proposed, e.g., by Ney et al. [152], [153].

3.4.6 Back-off n-gram language models

A back-off language model for speech recognition was proposed, e.g. by Katz [128]. The back-off $(n-1)$-gram is used as approximation whenever the n-gram is not reliable, e.g. the count of the n-gram in the corpus is too small for a reliable estimation. Let k be the desired minimal count of an n-gram in the corpus. A Good-Turing estimation can be used to compute the discounting d that reserves some probability mass from the maximum likelihood estimator for the back-off. Alternatively, an absolute discounting could be used. Formally, the Katz back-off model is recursively defined as follows:

$$P_{\text{bo}}(w_i|w_{i-n+1}...w_{i-1}) = \begin{cases} (1-d_{w_{i-n+1}...w_{i-1}})\frac{C(w_{i-n+1}..w_i)}{C(w_{i-n+1}..w_{i-1})} & \text{if } C(w_{i-n+1}..w_i) > k \\ \alpha_{w_{i-n+1}...w_{i-1}} P_{bo}(w_i|w_{i-n+2}..w_{i-1}) & \text{otherwise} \end{cases},$$

where α ensures that the overall model is normalized. Actually, the back-off model is a linear interpolation as introduced in Section 3.4.4 Alternatives are proposed, e.g. by Rosenfeld et al. [154] or Kneser et al. [155]. The latter optimizes $(n-1)$-gram as a significant factor of the back-off combination together with the discounting and normalization weights . This method uses $(n-1)$-gram estimations which are proportional to the number of different words that follow instead of the $(n-1)$-gram counts. A detailed introduction in back-off language models is provided by Huang et al. [66].

3.4.7 Language model combination techniques

It was proposed to interpolate statistical grammars and language models. Kaufmann et al.[156] investigated the usage of rules in statistical language models. Combining stochastic context free grammars with 3-gram models was proposed several times, e.g. by Gillett et al.[157]. Nasr et al.[158] generalized this approach for n-grams. A combination of unified context free grammars and n-gram models was proposed, e.g. by Wang et al.[159] or Martin et al.[160]. In addition, Wang et al.[161] described the combination of statistical and rule-based approaches for language understanding. Vaibhava et al.[162] used a maximum likelihood estimation to improve annotation performance of n-gram models by incorporation stochastic finite-state grammars. Mohri[163] proposed to embed grammars hierarchically. Adaptation techniques for statistical language models using context free grammars were proposed, e.g. by Wu [140]. This thesis introduces a nested of grammars in statistical language models.

3.5 Language processing

There are many applications for language processing. In this thesis, methods are described which are used for speech recognition and related applications. Section 3.5.1 describes a tokenization process. For the sake of clarity, let a token be the desired set of letters. Often, a token is similar to a word. The language model is estimated on these token sequences and the output of the speech decoder is a sequence of tokens. Section 3.5.2 describes a method which interprets the token sequence and generates a formated word sequence. This is then presented to the user. The formatting depends on the context. A medical text has a different format compared to a short message.

3.5.1 Word tokenization

A sequence of letters is separated into a sequence of tokens. This process is called tokenization. White spaces are commonly used to define tokens. A token is often identical to a word. In addition, the tokenization process takes care on well defined tokens, e.g. whether a word is capital or not. It computes also word stems for applications in natural language understanding. The process does also convert written words, e.g.,

111	one one one | one hundred and eleven | hundred and eleven | one eleven
9:30	half past ten | nine thirty.

This process is best suited for a rule-based method. Often, a normalization process is applied which takes care that the generated tokens are distinguishable across the corpora, e.g.,

Ct.|<address>	Connecticut
Ct.|<medical>	computed tomography
Ct.|<musical instruments>	Coil Tap

where "$<\cdot>$" denotes the context of the sentence. Both, tokenization and normalization are most often domain dependent.

In this section, a statistical method is described that identifies co-occurrence of words which can be considered as one token. This has also an impact on the understanding of natural language as described by Georges et al. [2] where the tokenization

is an essential part of a query disambiguation. Word co-occurrences can also be used to improve speech recognition by reducing the language model perplexity. A method is described which uses mutual information, as proposed by Church et al. [164]. A comparative study of joining frequent word sequences to phrases is provided by Klakow [165]. Let "I" be the mutual information that is defined as:

$$\mathrm{I}(w_I, w_J) := \sum_{i,j} P(w_i, w_j) \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)}$$

where the log can be denoted as point-wise mutual information ("PMI") as:

$$\begin{aligned}\mathrm{PMI}(w_i, w_j) &:= \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)} \\ &= \log \frac{p(w_i|w_j)}{p(w_i)} \\ &= \log \frac{p(w_j|w_i)}{p(w_j)} = \mathrm{PMI}(w_j, w_i)\end{aligned}$$

where w_i and w_j are words. This measure can be used to decide whether to join or not both words. A value of 0 indicates independence between w_i and w_j. The "PMI" maximizes for a co-occurrence of w_i and w_j and minimizes if the 2-gram $w_i w_j$ or $w_j w_i$ does not occur in the training corpus. It can be generalized with the chain-rule. Let $w_j w_k = \underline{h}$ be the history of word w_i so that PMI is defined as follows:

$$\begin{aligned}\mathrm{PMI}(w_i, w_j w_k) &= \mathrm{PMI}(w_i, w_j) + \mathrm{PMI}(w_i, w_k|w_j) \\ &= \log \frac{p(w_i, w_j w_k)}{p(w_i)p(w_j w_k)} = \log \frac{p(w_i|w_j w_k)}{p(w_i)} \\ &= \log \frac{p(w_i|\underline{h})}{p(w_i)} = \mathrm{PMI}(w_i, \underline{h})\end{aligned}$$

w_i and $\underline{h}$ are independent if $\mathrm{PMI}(w_i, \underline{h}) = 0$. This generalized variant is best suited because all probabilities are known from the language model developing. Alternatively, Bouma [166] described a variation which is normalized between $[-1, 1]$. The normalized point-wise mutual information is 1 for complete co-occurrence of w_i and $\underline{h}$. In contrast, it is -1 if w_i and $\underline{h}$ are never occurring together. It is computed as follows:

$$-1 \leq \mathrm{PMI}_{normalized}(w_i, \underline{h}) = \frac{\mathrm{PMI}(w_i, \underline{h})}{-\log(p(w_i, \underline{h}))} \leq 1$$

As an example, a 2-gram language model is computed for the Wall Street Journal $5k$ corpus [167] using Kneser-Ney discounting. The 1-gram and 2-gram probabilities are used for the "PMI" computation. A sorted subset is given in the following for common co-occurrences:

$$\begin{aligned}
\text{PMI}(\text{santa}, \text{fe}) &= 13.74529 \\
\text{PMI}(\text{hong}, \text{kong}) &= 12.54758 \\
\text{PMI}(\text{las}, \text{vegas}) &= 12.53122 \\
\text{PMI}(\text{los}, \text{angeles}) &= 11.16868 \\
\text{PMI}(\text{tele}, \text{communications}) &= 8.13129 \\
\text{PMI}(\text{multimillion}, \text{dollar}) &= 8.03381 \\
\text{PMI}(\text{U.}, \text{S.}) &= 7.21532.
\end{aligned}$$

All these 2-grams are suited to be considered as one token. In this way, "los angeles" will become one word "los_angeles" where the word order is given by the origin word sequence. In practice, a threshold has to be defined between number of tokens and a development-set perplexity. A low threshold might also concatenate common phrases, such as:

$$\begin{aligned}
\text{PMI}(\text{accounted}, \text{for}) &= 5.98041 \\
\text{PMI}(\text{according}, \text{to}) &= 5.93769 \\
\text{PMI}(\text{intends}, \text{to}) &= 5.9195 \\
\text{PMI}(\text{able}, \text{to}) &= 5.91757.
\end{aligned}$$

Whereas a PMI $>> 0$ denotes a co-occurrence, a PMI near zero indicates a unlikely word pair in the corpus. These n-grams are most probably independent from each other and not suited to be concatenated. Moreover, it is most likely that these words will never occur consecutively in this corpus. Some examples are:

$$\begin{aligned}
\text{PMI}(\text{manufacturer}, \text{holds}) &= 0.00058 \\
\text{PMI}(\text{stocks}, \text{shows}) &= 0.00013 \\
\text{PMI}(\text{legal}, \text{confrontation}) &= -0.00013 \\
\text{PMI}(\text{pennsylvania}, \text{cleveland}) &= -0.00061.
\end{aligned}$$

In contrast, n-grams with a PMI $<< 0$ do not occur in the corpus at all. The conclusion whether to concatenate or not, is just not possible. For example, there are linguistic reasons such as "the said" or "said the" which are very uncommon on the one hand. On the other hand, there are semantical 2-grams which are obviously not relevant in the corpus as the following example shows:

$$\begin{aligned} \text{PMI}(\text{not}, \text{international}) &= -4.50398 \\ \text{PMI}(\text{two}, \text{interests}) &= -5.00358 \\ \text{PMI}(\text{the}, \text{said}) &= -8.95582. \end{aligned}$$

3.5.2 Text formatting and interpretation

One text formatting example was described in Section 2.4 together with the introduction of composing transducers. Text formatting is a main developing stage for building speech recognizers. Its precision can make the difference whether a speech recognizer is used or not. For example, the text formatting for medical dictation is quit important. In some sense, the text formatting process is an inverse of the tokenization and normalization process described in previous section 3.5.1.

The formatting process is domain dependent and may also vary between user-settings. A typical example is date and time formatting, e.g. whether the user likes "." or "/" delimiter or a 12 hours cycle or 24 hours one. Moreover, text formatting has to interpret an utterance so that it can format it in the appropriated way. A queried time phrase should also be displayed in a similar way, e.g.

half past ten	9:30
nine thirty	9:30

This example can be extended to number formating in general, e.g. for presenting monetary numbers, date and time representation or dosing of medicines. In the following, there are four examples how people might utter "111":

one one one	1:1:1
one hundred and eleven	111
one eleven	1.11
one eleven	one 11.

Which of the following alternatives is the correct one might depend on various aspects, e.g. domain, context, target application. "111" might be different for medical dictation or address entry. A similar example is abbreviation handling, e.g. for address destination:

drive	Dr
court	Ct
place	Pl
road	Rd
lainways	Ln
street	St
avenue	Ave
circle	Cir
terrace	Ter
trail	Trl
way	Way
boulevard	Blvd
loop	Loop
pass	Pass
parkway	Pkwy

Some of these examples are uniquely determinable within the address domain. Other examples are ambiguous, e.g. "St" which could also be "saint" or "state" depending on the context. The abbreviations are ambiguous between domains, e.g., "Dr." could be "drive" or "doctor". Similar for "Ct" which could be "Connecticut", "court", "computed tomography" or "Coil Tap" depending on its context and domain. Thus, text formatting is already a text interpretation stage and may benefit from linguistic knowledge. Abbreviations are also content provider dependent and may also vary between cultures and regions. Formatting does also apply taboo-filtering and may take care on capitalization. Punctuation is part of most text formatting processes.

Common text formatting techniques are based on a mix of rules and statistical models. This was described for speech recognition on mobile devices, e.g. by Schuster [168]. The rules are usually written by human experts for dedicated applications and domains. Some of these rules are adjustable by the user, e.g. time and date representation or whether to use a dot or comma as thousands separator.

3.6 Summary

This chapter introduced automatic speech recognition to translate human speech to text. The set-up and theory is described in Section 3.1. The technique can be separated into several modules. An overview of each module is provided in Section 3.2. The preprocessing module is described in three sections.

Section 3.3 introduces the feature front end. The processing chain is described starting from the audio capturing. Mel Frequency Cepstral Coefficients are described as todays de facto standard features. These features are used to compute emission probabilities according to an acoustic model. Not only the audio-visual speech recognition is still an active research topic. Nearly all topics in speech feature processing is requested. Today, deep neuronal networks are being investigated in combination with bottleneck features. Recurrent neuronal networks and deep believe networks are also commonly investigated for speech and language processing.

Section 3.4 describes the representation of language. In this thesis, recognizers with closed vocabulary are considered. Hence, the recognizer need to have a knowledge of words and word sequences. Both, the grammatical representation as well as statistical models are described. A novel location aware speech recognition system was introduced. It enables a precise recognition of millions of address entries. Currently, future work is focusing on estimating sequence probabilities with neuronal networks. The combination of information retrieval techniques and methods for reasoning opens up new possibilities. Natural language processing and question answering will play an increasingly important role in human machine interfaces. Also the speech to speech translation requires precise language models.

Section 3.5 introduces basic text processing methods. The methods are used during model training on the one hand. On the other hand, it is applicable in the run-time part to display a uttered phrase in an appropriated way to the speaker. A novel technique was proposed to use word co-occurrence for query disambiguation. Future work needs to investigate the use of natural language processing techniques. This is not only important for textual representation but also to model a more humanoid voice in text to speech systems. Prosody and word emphasis is an important factor and analyzing speech and language may solve this open research question.

4

Dynamic Speech Decoding

Weighted finite-state transducers are used for speech recognition. The use of transducers enables to compile and optimize the search space for speech decoding in advance. This enables an efficient speech decoding on the on hand. But on the other hand, it reduces the flexibility to use user-dependent content during the recognition process. Georges et al. [9] propose the use of dynamic language models to combine user-dependent content and transducer based speech recognition. The used transducer nesting technique allows the decoding of multiple transducers simultaneously. In fact, user-dependent content can be added on-the-fly. Moreover, it can be shared over multiple devices and recognized consecutively as described by Georges et al. [10].

Section 4.1 introduces weighted finite-state transducers for speech recognition. The lexicon representation is described in Section 4.2. Section 4.3 describes the representation of three language model types. First, the use of a grammar is described. Second, the representation of a statistical model is introduced. Third, a dynamic language model is described. The developing process to compile a dynamic language model is described in Section 4.4 followed by speech decoding methods for transducers. Speech decoding with transducers is described in Section 4.5 followed by a description of the decoding of nested transducers in Section 4.6. Decoding on multiple devices is introduced in Section 4.7. Finally, the chapter is summarized.

4.1 Finite state transducer for speech recognition

The weighted finite-state transducer for speech recognition is constructed and optimized in advance. Phonetic information from the acoustic model and information about word sequences from the language model are used. Whereas the composition operator for transducers is used for constructing, determinization and minimization is used for optimizing the transducer for efficient decoding. Weighted finite-state transducers are introduced in Chapter 2. Section 2.4 describes the composition operator. Determinization and minimization is described in Section 2.2 and 2.3, respectively.

The search space construction includes several transducers. A lexicon L transducer represents the relation between words $\underline{w} \in W^*$ and its phoneme sequences $\underline{p} \in P^*$. L is described in Section 4.2. The language model is represented in a weighted finite-state transducer G. The weights are used to determine the probability of a word sequence. Hence, all recognizable word sequences are represented in G. A composition is used to compute the relation between phoneme sequences in L and words in G:

$$LG = L \circ G$$

where "$\circ$" denotes the composition operator. G can represent a grammar, a statistical language model or a dynamic language model. The construction is described in Section 4.3. The developing of dynamic language models using finite-state transducers is described in Section 4.4. Speech decoding with transducers is described in Section 4.5.

4.2 Lexicon transducer

A lexicon transducer represents the relation between words $\underline{w} \in W^*$ and their phonetic transcriptions $\underline{\rho} \in P^*$. In fact, the relation $(W \times P)^*$ is represented. A subset of a lexicon transducer L' is shown in Figure 4.1.

The transducer can be seen as a 0-gram word loop model where each word is related to its phonetic transcription. The probabilities are uniformly distributed, usually. Pronunciations variants may use some weighting. The lexicon transducer is optimized by applying determinization and minimization. Both operators, "min" and "det" are

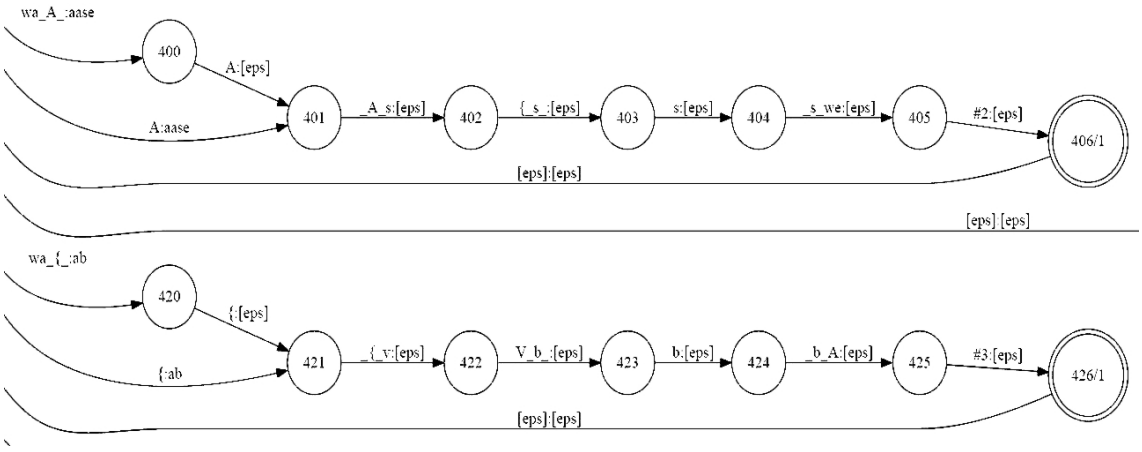


Figure 4.1: *Subset of a lexicon transducer. The input labels are phonemes and the output labes are words. Both, phonemes and words are aligned using ϵ symbols.*

described in Section 2.1 and 2.3. The lexicon L is computed by:

$$L = \min(\det(L'))$$

The phonetic transcription can be created by human experts. Often a grapheme-to-phoneme mapper is used which was trained on some human transcribed training data. Usually, a set of most frequent words of a language is used to train a mapper. The construction of a lexicon transducer is described, e.g. by Mohri et al. [169]. Homophones such as "two", "too", "to" or "read", "red" are sharing the same phonetic transcriptions. An auxiliary symbol "#i" with $i = 0..n$ is used for each n homophones. This auxiliary symbol are removed after speech decoding and not visible to the user. The result is a determinizable transducer. The following two examples illustrate how to augment the phonemes with auxiliary symbols "#":

$$\text{two} =: \text{two\#1}$$
$$\text{too} =: \text{too\#2}$$
$$\text{to} =: \text{to\#3}$$
$$\text{read} =: \text{read\#1}$$
$$\text{res} =: \text{red\#2}$$

4.3 Language transducer

The language transducer G represents the language model. It is used by the decoder to decide whether a word sequence is probable or not. This is a binary decision for grammars. For statistical models, the transducer is used to compute the probability of a word sequence. Command and control tasks are using grammars, typically. A grammar is also used by recognizers for heavy content domains, such as voice destination entry. Statistical models are usually used, e.g. for dictation tasks. Actually, a language model can be represented by a weighted automaton. In practice, a transducer is used which enables the use of the transducer framework. Note, the input and output word sequence of the transducer is identical. The transducer G' is optimized as follows:

$$G = \min(\det(G'))$$

Where "min" and "det" are operators on finite-state transducers as described in Section 2.1 and 2.3. The optimization might happen on-the-fly when the transducer is compiled from the language model, e.g. for heavy content domains. The construction of G' using grammars is described in Section 3.4.1. The representation of a statistical language model as transducer is described in Section 4.3.2. Dynamic language model representation is described in Section 5.3.2.

4.3.1 Grammatical text representation

A grammar is s compact representation for regular languages. A structured model was proposed, e.g. by Chelba [170] or Rastrow et al. [171]. Kaudmann proposed a rule based language model for speech recognition [156]. Duckhorn et al. described a method to use context free grammars for embedded speech recognition [86].

Table 4.1 illustrates a grammar used for voice dialing. Its representing transducer

Table 4.1: *Hand written example egrammar for en-US VDE*

⟨*number*⟩	::=	[<s>] ⟨*num*⟩
⟨*num*⟩	::=	eins | zwei | drei | vier | fuenf | sechs | sieben | acht | neun | null

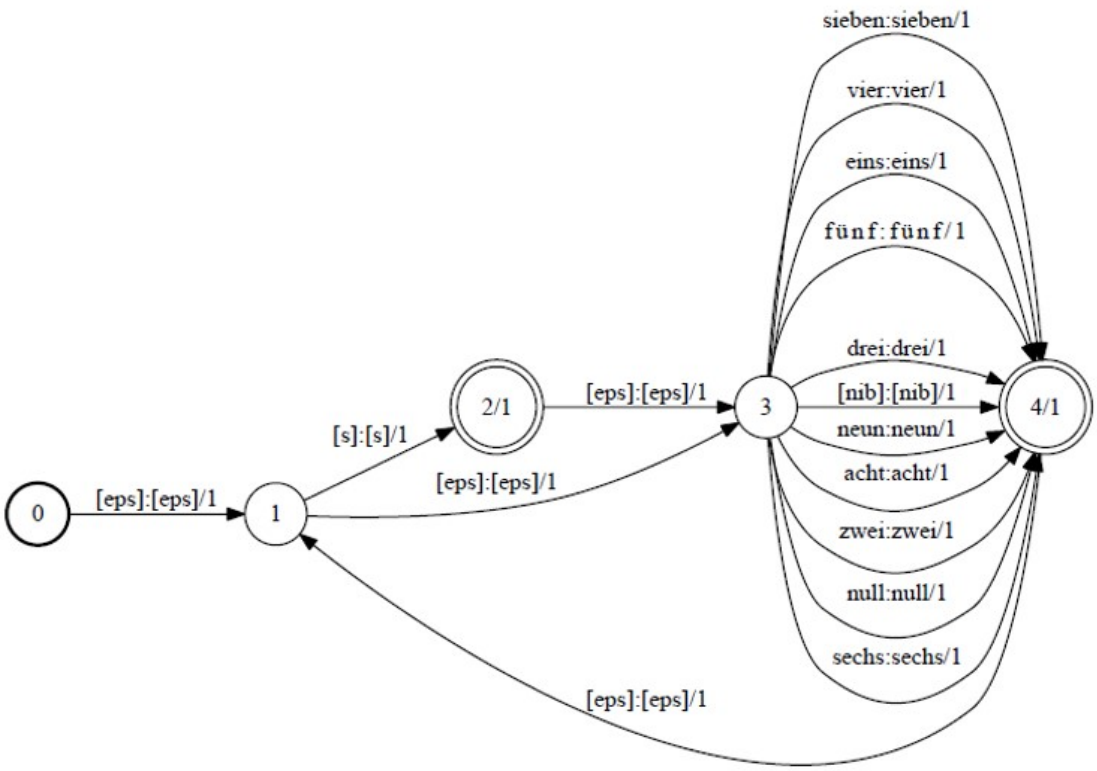


Figure 4.2: *Grammar for digit recognition. The silence "[s]" is optional between digits.*

is shown in Figure 4.2. The "[s]" label denotes an optional silence phase in the feature stream. The transducer also introduces a "[nib]" label which is used as garbage model to increase the robustness, e.g. for voiced background noise. ϵ-transitions are denoted by "[eps]". All digit sequences are recognizable. Reducing the search space on valid telephone numbers would increase the search space, significantly. However, it is still possible to add some constrains as trade-off between sharp search space and its memory requirement. For example, the sequence has to start with an area code or a certain number of digits needed to be recognized. Typically, grammars are hand written, e.g. for command and control applications such as controlling smart TVs by voice. It is also possible to create such grammars automatically. An introduction of grammars for speech recognition is given in Section 3.4.1.

In the following, the construction of a grammar for voice destination entry is described. The starting point is a relational database of entities for addresses. With out loss of generality, there are 3 columns in the database. One column includes street names, one column city names and a third one denotes all states, e.g. for the US. An address is given in each row. There are about 10 million rows for a US address database. Similar databases are available for media domains, air and train timetables etc. A lan-

Table 4.2: *Hand written example egrammar for en-US VDE*

⟨*address*⟩ ::= [<house number>] <street> [[<city>] <state>] |
<city> <state> |
<city> |
<state>

guage expert identifies the order of columns a user might utter their entities. The order of columns is written as grammar. An example grammar for US voice destination entry is given in Table 4.2. The user will be able to utter addresses by denoting the house number, street, city and state. Alternatively, there are partials defines such as street, city, state or city, state etc. This grammar is translated into an acceptor over the vocabulary {"city", "street", "state"}. The result is a transducer S representing the human written grammar. An example transducer is shown in Figure 4.3. For the sake of clarity,

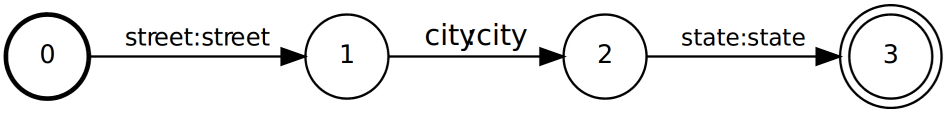


Figure 4.3: *Transcuder **S** representing a semantic relation for an US VDE grammar.*

house numbers are not presented. The database itself is translated into a finite-state transducer P. A subset is illustrated in Figure 4.4. The input vocabulary is given by entries of each column. Each entry is assigned with its column as output symbol over vocabulary {"city", "street", "state"}. Moreover, all possible permutations of columns are represented in the transducer. This way, the relational database is translated into a

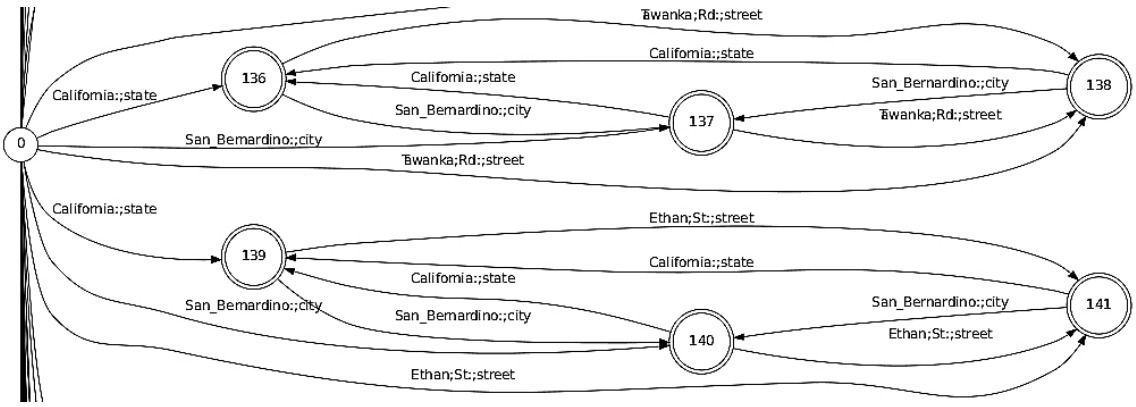


Figure 4.4: *Subsection of transducer **P** representing all permutations which are derived from a relational address database.*

transducer, directly. The composition of P and S will create the desired transducer as follows:

$$V' = \min(\det(P \circ S)), \tag{4.1}$$

where "$\circ$" denotes the composition operator and "min", "det" the operators to determinize and minimize the transducer. The operators are introduced in Section 2.4, 2.2 and 2.3. V' represents the relation between word sequences and columns identifiers. Here, it is the relation between street, cities and states names and the sequence of tags denoted by "street", "city" and "state". For speech recognition, the tags are not required. Those are removed from the recognized word sequence. However, the tags can be passed through the recognizer and evaluated afterwards for a database look-up, e.g. by the application. Figure 4.5 shows the final automaton. Note that no weights are used in the grammar. In general, it would be possible to add some weights, e.g. derived by some popularity measurements from the World Wide Web.

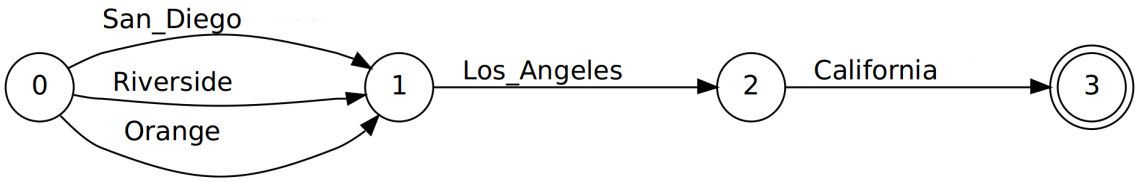


Figure 4.5: *Final recognition grammar for addresses.*

A similar construction can be done for all kind of data that is represented as relational database. It is also possible to use an on-the-fly composition technique to add custom dependent constrains in a language transducer. Adding content, e.g. new cities, is just an additional path in P. It is also possible to describe the transducer as word-class grammar analogous to a uniformly distributed class based language model described in Section 3.4.3.

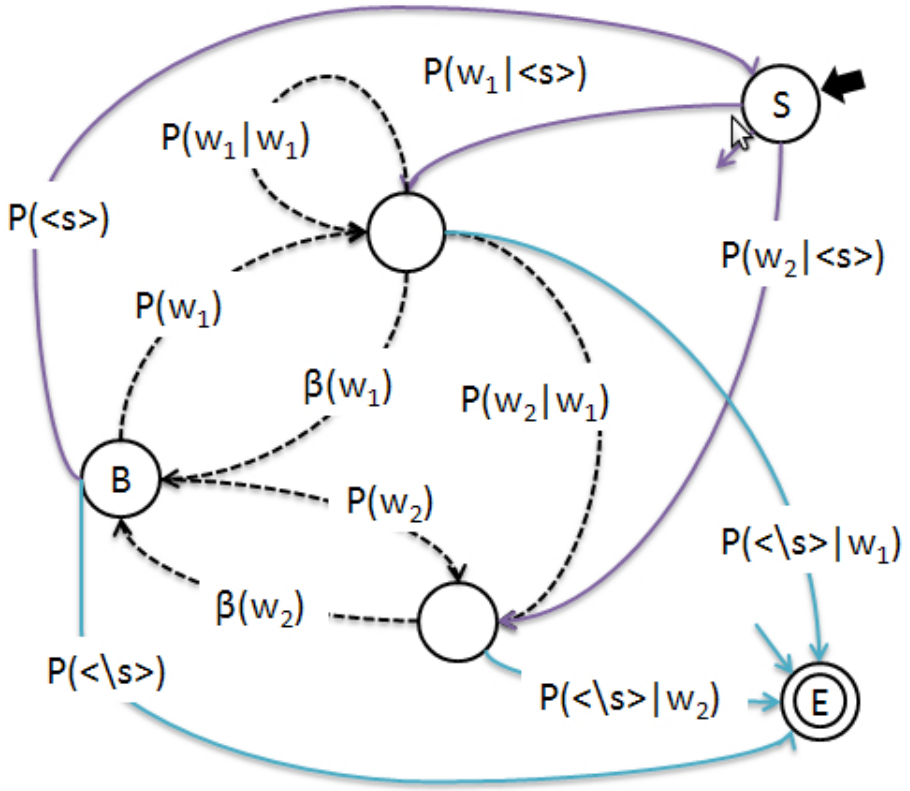


Figure 4.6: *Example representation of a 2-gram back-off Markov language model.*

4.3.2 Statistical Language Modeling

Statistical language models are used, e.g. for speech dictation or applications for natural language understanding. A back-off n-gram Markov model is commonly used. The model is estimated from text corpora, e.g. using web scraper or transcribed field data. N-gram language models are introduced in Section 3.4.6. Figure 4.6 illustrates an automaton for a 2-gram language model. It can compute probabilities for any combination of words w_1 and w_2 as follows:

$$<\text{s}>[w_1, w_2]^*<\backslash\text{s}>$$

where "<s>" and "<\s>" are denoting the begin and end of each sentence. In speech recognition, the start and end of a sentence is typically recognized by a significant silence phase in the feature stream. Note, the probability $P(w_2|w_1)$ for the word sequence $w_1\,w_2$ was estimated from some training data. There is a directed path $P(w_2|w_1)$ in the transducer. Note, the path over the back-off weight $\beta(w_1)P(w_2)$ for the word se-

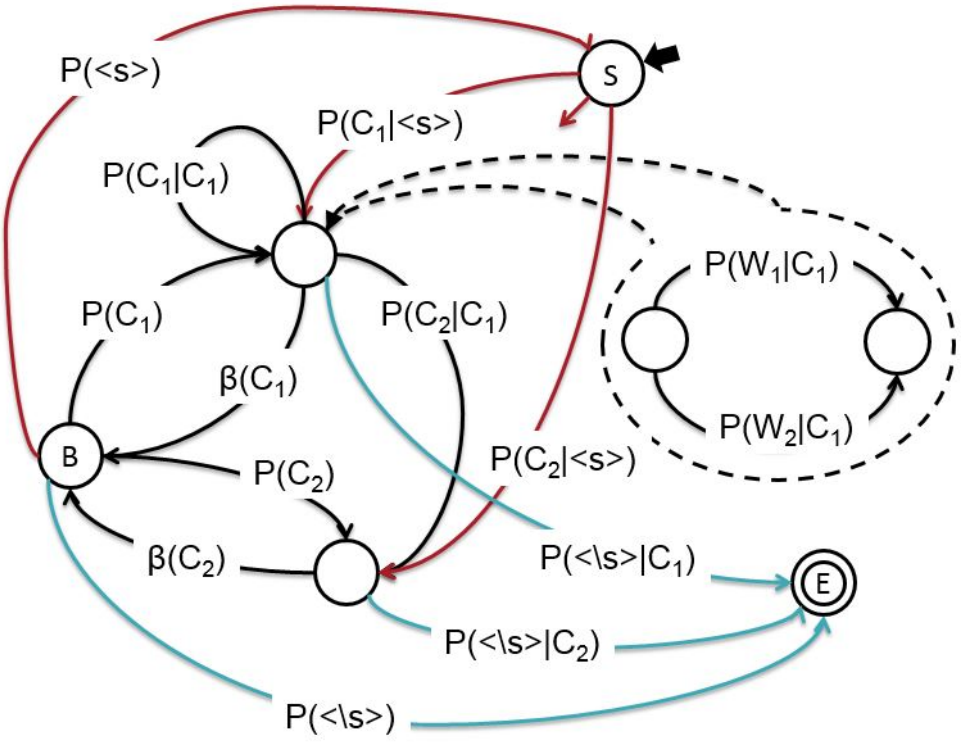


Figure 4.7: *Example representation of a word-class 2-gram back-off language model.*

quence $w_1 w_2$. This is usually used when the probability for $w_1 w_2$ could not been estimated from the training data. In this representation, there are always two paths. This does not influence the decoding, because the following equation is fulfilled for most statistical language models:

$$P(w_2|w_1) > \beta(w_1)P(w_2)$$

Word class n-gram models are representable in a similar way. An example is illustrated in Figure 4.7. This kind of class based language model was introduced in Section 3.4.3. Here, a 2-gram Markov model on word classes is described. It is very similar to the word n-gram model described previously. Either there are transitions between stages for each word w_i in a class $C(w_i)$ as follows:

$$P(w_2|C(w_2))P(C(w_2)|C(w_1)).$$

Or each transition $P(C(w_2)|C(w_1))$ is assigned to a lexical tree computing,

$$P(w_2|C(w_2)).$$

Note, β(<s>) and β(<\s>) are omitted for a better overview. An efficient class based language model was proposed, e.g. by Whittaker et al. [118]. Georges et al. generalized this approach for dynamic language models as described in Section 4.3.3.

In this section, the representation of statistical language model as finite-state automatons was described. It was illustrated by a word and a word-class 2-gram language model example. Whereas the word model becomes the baseline experiment, the word-class model will be extended to a dynamic language model in the following section.

4.3.3 Dynamic Language Modeling

A dynamic language model enables the use of content from various sources. For example, it enables a recognizer to consider just local available user knowledge. Dynamic language models can be seen as generalized class based models. In this thesis, a dynamic language model nests grammars into statistical language models. To create a dynamic language model, a preprocessing step is used to replace grammatical word sequences form a text corpus with its corresponding rule name.

Let D be a rule for any dates and let "22.04.1985" be one instance of that rule. All such grammatical word sequences are given by S_K and used by function R to replace them in any sentences. For example, the sentence

"My birthday is 22.04.1985"

will become

"My birthday is <D>" with <D> generalizing "22.04.1985"

after applying R. A set of sentences $\mathbb{C}'$ is computed by applying the function R on each sentence of $\mathbb{C}$. $\mathbb{C}$ denotes as text corpus which is used for model training, e.g. a web scraped text. This is formalized by:

$$\mathbb{C}' = \{\underline{w} \in ((W \cup T)^* \backslash S_K)^* | \exists \underline{w}' \in \mathbb{C} : \underline{w} = R(\underline{w}')\}.$$

Let S be a set of tuples (U, t) which associates each set $U \subseteq S_K$ with a corresponding grammar tag $t \in T$. The rule for any dates is assigned with the tag <D> as follows:

$$S = \{(U, t) | \exists k : U = S_k \wedge t = t_k\}$$

Further, each sentence in S_K is unique and fulfills:

$$|S_K| = \sum_k |S_k|.$$

Nevertheless, there could still be ambiguity between partial sentences. In this case, we choose the longest word sequence which can be computed by:

$$\underline{\hat{w}} = \min_{k \leq K}(\operatorname*{argmax}_{\underline{w} \in S_k}(\underline{w}))$$

Where max/min is the sequence with the maximal/minimal number of words. R can be computed by common string search methods using regular expressions. In this thesis a method based on finite-state transducer is used and described in next Section 4.4.

Jurafsky et al. proposed the use of stochastic grammars as language model for speech recognition [85]. A combination method of stochastic grammars and n-gram language was presented, e.g. by Gillett et al. [157]. Nasr et al. described a method to combine n-gram models with stochastic automatons [158]. Several contributions were also proposed by Wang et al., e.g., [159], [161] or [172]. Section 3.4.7 gives an overview of language model combination methods. Usually, a class includes a list of words. An evaluation of class based language models was provided, e.g. by Beaujard et al. [123]. Maltese et al. [136] made a comparative study of combined word and word-class models for several languages and clustering techniques. Determining word classes by the use of part of speech statistics was provided by Samuelsson et al. [124]

4.4 Creating a dynamic Language model

A users address book or favorite music titles are dynamic parts of a natural language. These parts can be modeled separately using a dynamic language model. A dynamic language model consists of, at least, 2 models. The dynamic content is represented in one model and nested in another one during speech decoding. In general, the dynamic content could be represented as grammar or statistical model. For the sake of clarity, let the dynamic content be a grammar. Typically, the dynamic language model is spare represented in the training data. For example, dynamic content could be user's medical records or favorite point of interests on the one hand. On the other hand, it could be sparse represented word sequences with long range dependencies, such as telephone numbers or stock prices.

Each model in a dynamic language model is trained separately. Section 4.4.1 outlines method and techniques to tag word clusters in a text. Theses word clusters can be considered as dynamic content. A grammatical tagger is described in Section 4.4.2. This method is used in this thesis to find proper names, abbreviations and stock prices. The output is used to train a model for the dynamic content. In fact, the derived information is used to populate some generalized dynamic content data, e.g. by human experts. Section 4.4.3 describes a technique to replace grammatical represented word sequences in a text. The processed text is used to train the model in which the dynamic content is nested during speech decoding.

4.4.1 Word sequence clustering

Word classes have a significant impact on the recognition performance. Both the accuracy and the computational effort can be influenced by the class definition. Yu et al.[173] improved name recognition by introducing a user model. The impact of word clusters for text classification was investigated by Noam et al. [174]. All kind of structured content is suited to be represented in word classes. Examples are named entities such as proper names or business names.

A general approach was proposed by Martin et al.[175] using n-gram word clusters. Lin [176] proposed an automatic retrieval and clustering of similar words. Wiebe et al.[177] described WordNet sense tagging in the Wall Street Journal. Mutual information was used similar to the technique described in Section 3.5.1. Chunyu et al.[178] de-

scribed a method for abbreviation recognition using maximum entropy models. Roche et al.[179] described a deterministic part of speech tagging with finite-state transducers. Kit et al.[178] proposed a more general abbreviation recognition with a maximum entropy model.

A dynamic language model extends the concept of word classes by introducing classes as container for grammatical given word sequences. A tagging method was used on the basis of transducers as described in the following section. There are various named entity taggers available. Chrupala et al.[180] proposed one for German and a series of named entity taggers is also implemented in NLTK [181]. In this thesis, named entities are tagged by an introducing word 'Mr.', 'Mrs.' etc. A grammatical extraction method was used and is described in the following sections 4.4.2 and 4.4.3.

4.4.2 Extraction of grammatical structures

A rule based part of speech tagger was proposed by Brill [81]. A grammar can be used to represent regular languages in a compact way. This was introduced in Section 3.4.1. Grammars can also be used to filter an arbitrary text. The output will be a set of sentences which are represented in the grammar and given in the text. This way, statistics of certain word sequences are derivable from an arbitrary text. For example, the method is used to extract the abbreviations from the Wall Street Journal [39]. Following sentences is an example:

- about half these managers are in the u. s.

A grammar is used which interprets every single letter with a consecutive dot "." as abbreviation. The output of the filter is a list of all abbreviations in the input text. The result for the example above is:

- u. s.

The output can be used to estimate statistics, e.g. for a language model adaptation. It is also used to train a model for similar dynamic content, such as abbreviations, stock prices, etc. Note that the filtered data can be used as seed data to adapt statistical models.

The technique is described in the following. Let W be the vocabulary, e.g. for English. Let $C \subseteq W_c^*$ be a corpus of sentences $I \subseteq C$ with vocabulary $W_c \subseteq W$ e.g. the WSJ Corpus [167]. Further there is a set of word sequences S. This set of word sequences are

representable as grammar. The grammar is written by human experts. It could be a list of proper names or a set of number representations. In this thesis, regular grammars are used but the method can be extended to context free grammars. The grammar is translated into a transducer T' where the input language is W^*. The transducer generates an identical output for all input sentences $S \subseteq W^*$. All other input sequences are non emitting. Hence, it filters an input sequence by a grammar S.

This can be formalized as composition. Each sentence $I \in C$ can be represented as automaton. As described in Section 2.4, a composition can be used to compute the intersection of two languages. Here, the composition of I and T' will result in the desired word sequence $\underline{w}$:

$$\underline{w} = \mathrm{LS}(I \circ T'),$$

where $\mathrm{LS}(\cdot)$ generates the longest word sequence. For sentences with equal length, random sampling is used to restrict to one solution. This can be motivated by a human user who is only interested in one solution, typically.

In practice, the vocabulary W is unknown, e.g. when processing text from web pages. Let T be the transducer over an input vocabulary $S \cup \{\epsilon\}$. In addition, a mapping $\mathrm{RV}(\cdot)$ is applied on the input sentence I which maps each unknown word to ϵ. The final method is formalized as follows:

$$\underline{w} = \mathrm{LS}(\mathrm{RV}(I) \circ T),$$

The mapping $\mathrm{RV}(\cdot)$ can also be done on-the-fly. A robust input and output handling was introduced in Section 2.7. An example transducer using the λ notation on transitions is shown in Figure 4.8. It is used to filter a text for abbreviations. An abbreviations is defined as a word with 2 letters where the second one is a dot ".".

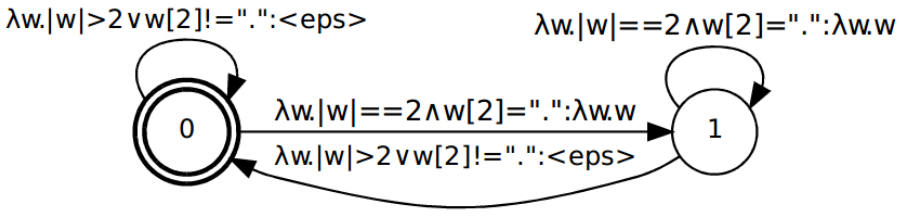


Figure 4.8: *Transducer for extracting abbreviations from a training corpus.*

4.4.3 Grammatical structure replacement

Looking for certain word sequences in an arbitrary text was discussed in the previous section. In this section, a method is described which replaces word sequences by a different sequence, e.g. tags. Similar to the retrieval task, a grammar is used to represent the word sequences in a compact way. Applications are taboo-filtering and text normalization on the on hand. On the other hand, it is used to pre-process a training text for dynamic language model training. For example, the technique is used to identify time, date or media search phrases in a training corpus. An example is selected from the Wall Street Journal [39]:

- it had sales of ninety point five million dollars in nineteen eighty six third quarter

Note, the number representation are spares in the training data. For example, there are no phrases denoting a year of the new millennium. In general, the "five million dollars" are not related to "nineteen eighty six". The idea is to replace these phrases so that the sentence will look like:

- it had sales of <NUMBER-in-DOLLAR> in the <YEAR> third quarter

The phrases "<NUMBER-in-DOLLAR>" and "<YEAR>" are considered as dynamic content during speech decoding with dynamic language models. Hence, the number representation is generalized. During speech decoding a model for uniformed numbers is entered which enables to recognize numbers. These where not present in the training corpus. The method is used as corpus preprocessing to estimate a dynamic language model. Let T be a transducer over vocabulary $W \times W \cup \{R\}$ with R being the set of tag symbols:

$$W \cap R = \emptyset$$

In the example above $R = \{$<NUMBER-in-DOLLAR>,<YEAR>$\}$. Tag-labels are not part of the recognition vocabulary. R is the set of tag-labels where each label is associated with one grammar, e.g. the grammar for time and date representations. The transducer T can be constructed from a grammar. Karttunen [182] proposed a replace operator. A replacing method R is applied for each sentence in $\mathbb{C}$. This is defined recursively:

$$R : \underline{w} \mapsto R'(\emptyset, 1, 1),$$

where R' computes the replacing for a word sequence $\underline{w}$ starting at position 1, 1 in the word sequence. It is defined as follows:

$$R'(o,i,j) = \begin{cases} R'(o\,w_{i:j}, i+j, 1) & \exists v \in W^+ : w_{i:j}\,v \notin S_K \\ R'(o,i,j+1) & \exists v \in W^+ : w_{i:i+j}\,v \in S_K \\ R'(o\,t, i+j, 1) & \begin{cases} \exists (U,t) \in S : w_{i:i+j-1} \in U \\ w_{i+j} \notin U \end{cases} \\ o\,w_{i,|\underline{w}|} & \text{else.} \end{cases}$$

The computation starts with an empty result word sequence, the word w_1 at position 1 and a step size of 1. Four cases have to be considered. First, no initial part of any word sequence in S_K was observed. Second, a word sequence in S_K was partly observed. Third, a complete word sequence in S_K was observed and can be replaced with its corresponding tag symbol. Fourth, the remaining word sequence will not match any word sequence in S_K. R is applied for each sentence in C. The result in C' is a preprocessed text with all sentences in S_K replaced by tag symbols. $\mathbb{C}'$ is then used for N-gram language model learning.

The desired word sequence $\underline{w}$ from a sentence I can be computed as a composition. This is formalized as follows:

$$\underline{w} = \mathrm{LS}(I \circ T),$$

Let LS$(\cdot)$ be a function that generates the longest word sequence. The automaton I is derived from the source sentence.

This method requires to know the vocabulary W in advance. However, this is not the case for many applications, e.g. to process web-scraped text. A robust extension for open vocabulary transducers is introduced in Section 5.2.2. The input and output handling is replaced by λ notations. An example is shown in Figure 4.9, it replaces

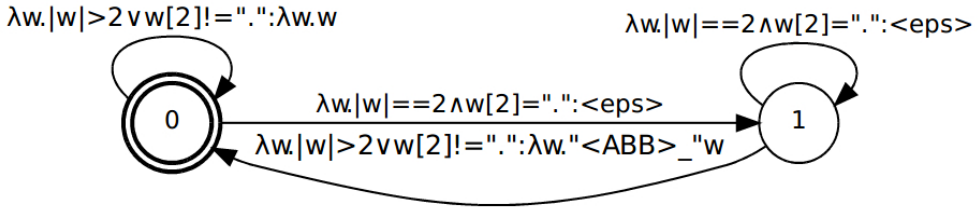


Figure 4.9: *Example transducer that replaces abbreviations in a training corpus.*

abbreviations. Given the example from previous section, the transducer will replace all single letter words with a preceding dot by "<ABB>":

- about half these managers are in the u. s.

The abbreviation "u. s." is replaces by "<ABB>" so that the sentence will become:

- about half these managers are in the <ABB>

Note that the transducer does not know anything else than the structure of an abbreviation. It is also possible to add a list of abbreviations, e.g. "SVOX", "BMW".

4.5 Speech decoding with transducer

Starting from the fundamental formula of speech recognition, an equation is derived for speech decoding. Let $\underline{\hat{w}}$ be the most likely word sequence according to the probability of words given a sequence of features $\underline{x}$ This can be formalized as follows:

$$\begin{aligned}\underline{\hat{w}} &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} P(\underline{w}|\underline{x}) \\ &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} \frac{P(\underline{x}|\underline{w})P(\underline{w})}{P(\underline{x})} \\ &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} P(\underline{x}|\underline{w})P(\underline{w})\end{aligned}$$

where $P(\underline{x}|\underline{w})$ denotes the acoustic model and $P(\underline{w})$ denotes the language model. The acoustic model considers word sequences as sequences of phonemes. In practice, the word sequences are considered as phoneme sequences. These phonemes can be further segmented into hidden Markov states. In fact, the most likely sequence is computed by computing all possible state sequences $\underline{s}$ as follows:

$$\begin{aligned}\underline{\hat{w}} &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} P(\underline{x}|\underline{w})P(\underline{w}) \\ &= \underset{\underline{w}\in W^*}{\operatorname{argmax}} P(\underline{w}) \sum_{\forall \underline{s}} P(\underline{x},\underline{s}|\underline{w})\end{aligned}$$

The last sum is not practical, especially for recognizing large vocabularies on embedded devices.

A "Viterbi approximation" is used to compute the most likely state sequence instead of the most likely word sequence, e.g. outlined by Huang et al. [66]. The equation becomes:

$$\underset{\underline{w}\in W*}{\operatorname{argmax}} P(\underline{w}) \sum_{\forall \underline{s}} P(\underline{x}, \underline{s}|\underline{w}) \approx \underset{\underline{w}\in W*}{\operatorname{argmax}} \left(P(\underline{w}) \max_{\underline{s}} P(\underline{x}, \underline{s}|\underline{w}) \right)$$

A language model weight is used: $P(\underline{w})^{\lambda}$ with λ being the language weight. This was described, e.g, by Brown [183]. It should balances the acoustic and language model distribution. The equation becomes:

$$\underline{\hat{w}} \approx \underset{\underline{w}\in W*}{\operatorname{argmax}} \left(P(\underline{w})^{\lambda} \max_{\underline{s}} P(\underline{x}, \underline{s}|\underline{w}) \right)$$

In addition, a word transition penalty ρ^{λ_w} is introduced. This factor is used for a word length modulation. Here, it is a word independent constant and multiplied to the word probability whenever a word starts. The factor λ_w is tuned on some developing data together with the language model weight λ_{LM}. The equation becomes:

$$\underline{\hat{w}} \approx \underset{\underline{w}\in W*}{\operatorname{argmax}} \left(P(\underline{w})^{\lambda} \rho^{\lambda_w} \max_{\underline{s}} P(\underline{x}, \underline{s}|\underline{w}) \right)$$

In practice, the inverse of the Bayes' posterior probability is used together with logarithms to avoid multiplications during speech decoding. Finally, the equation used for speech decoding is:

$$\underline{\hat{w}} \approx \underset{\underline{w}\in W*}{\operatorname{argmin}} \left(\log(P(\underline{w}))\lambda + \log(\rho)\lambda_w + \log(\max_{\underline{s}} P(\underline{x}, \underline{s}|\underline{w})) \right)$$

Where the language $P(\underline{w})$ and acoustic model $P(\underline{x}, \underline{s}|\underline{w})$ is used to compute the most probable state sequences for an observation sequence to retrieve the best suited word sequence. Whereas $P(\underline{w})$ is used to be a statistical model. It is also common to use stochastic grammars, e.g. for recognizing named entities.

In practice, it is established to use unweighted grammars, e.g. for digit recognition. Also heavy content recognition uses unweighted grammars, since there is just no suitable distribution available. An example is voice destination entry where all destinations are equally probably voice queries. A study using finite state graphs for speech recognition was provided, e.g., by Ou et al. [184].

Sturtevant [185] proposed a stack decoding. Here, a token passing time synchronous Viterbi beam search is used as proposed, e.g. by Young et al. [23]. A one pass decoder design was described, e.g. by Odell et al. [186]. An introduction in dynamic programming search for continuous speech recognition is provided by Ney et al. [19]. This thesis uses finite state transducers as motivated, e.g., by Mohri et al. [169]. An overview of decoding techniques for large continuous speech recognition was provided, e.g. by Aubert [187] and Saon et al. [188]. Here, a speech decoder on weighted finite state transducer is used. The transducer is compiled and optimized in advanced. It starts by composing the grammar with the lexicon transducer:

$$\mathrm{GL} = \det(L \circ G)$$

where L is the lexicon and $P(\underline{w})$ is represented by G. This was introduced in Section 4.2 and 4.3, respectively. The entire recognition search-space is modeled as transducer. It is a composition of the lexicon, language, phoneme context and hidden Markov Model transducer. The phoneme context is a transducer which models 2 to 4-gram phoneme features. Usually, the inter and intra word phoneme sequences are differed whereas cross-word modeling is neglected to fulfill memory requirements. It is possible to compute cross-word models on-the-fly. Here, the final transducer is computed by:

$$\begin{aligned} R =& \min(H \circ \det(C \circ GL)) \\ =& \min(H \circ \det(C \circ \det(L \circ G))). \end{aligned}$$

For the sake of clarity, the ϵ-removal operators is omitted in the above equations. Also all non valid paths are removed from R. A non valid paths is a path that will never reach a final state or, which cannot be derived starting from an initial state.

Often H and C is composed in advanced. The composition of G and L is done on-the-fly. This reduces the memory requirements on the one hand. On the other hand, it increases the computational effort during run-time. Techniques are described, e.g. by Hori et al. [34], [35]. Caseiro et al. [189] or McDonough et al.[36]. A more generalized approach was described by Allauzen et al. [37], [38].

Figure 4.10 illustrates the structure of a speech decoder. Features are computed by the acoustic front end. Further, the probability for each feature vector is computed accordingly to the acoustic model. The feature front end was introduced in Section 3.3. The decoder loads a finite-state transducer either completely or in pieces using an on demand loading method. A technique was described,e.g, by Kanthak et al. [190]

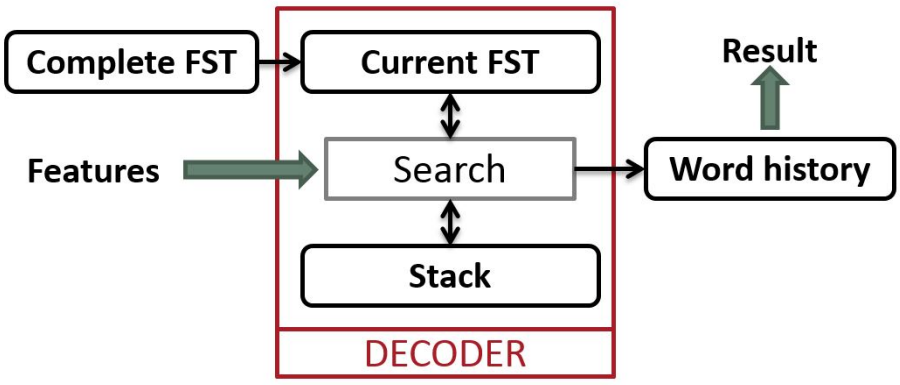


Figure 4.10: *Overview of a decoder used for speech recognition on embedded devices.*

On demand loading is commonly used on embedded devices to reduce the run-time memory requirements. Only a small portion of the transducer has to be available in random-access memory. Note, the caches on embedded systems are limited. The small portion is stored as "Current FST". However, the search algorithm can remain untouched. In addition, the decoder uses a stack which stores intermediate results, e.g. the list of active states. In the following, the stack consists of two queues Q_s and Q_t. The decoding result is stored as word history, representing most likely word sequences occurring. Note, only the word sequence is stored. In contrast, it would also be possible to store the entire state sequence including silent ϵ-transitions.

The speech decoder alternately computes a topological order and a Viterbi beam search. Two queues Q_t and Q_s are used which should be kept in the buffer for fast access. Q_t is filled by the search method and is input for the topological ordering. The result of the ordering, Q_s, is the input for the beam search. The topological sort is described in section 2.6.

Algorithm 4.1 outline the beam search method. All outgoing transitions for each node in Q_s are considered. The computation for each transition has always two stages. First, the new score is computed. For emitting transitions, both the language model score and the acoustic score is considered. A special treatment for self-loops might be required depending on the used acoustic and language model. Second, the score is evaluated to decide whether to keep the target stage or to prune it from the list. If the target stage will be considered, the state is stored in the second queue. In addition, the output label of the transition is computed. It is checked whether there is an output word or not. In case a word should be omitted, it will be added to the word history. ϵ-labels are usually not omitted. Finally, the next time step can be computed starting again with the topological order on Q_t.

Algorithm 4.1 *Pseudo code of speech decoding using a Viterbi beam search*

```
Q_s ← topo(Q_t)                                  // topological sort of active states
while ∀q ∈ Q do                                  // viter over actie states
  while ∀e ∈ q do                                // iter over all outgoing transitions
    q.score + t.score                            // compute new score
    if e not emitting then
      if e not seen yet ∨ e not pruned then
        if output label then
          add output
        end if
      end if
    else                                         // e emitting
      add emission score
      if e not seen yet ∨ e not pruned then
        if output label then
          add output
        end if
      end if
    end if
    Compute the current best score
    Q_t.push(q.nextstate)                        // add new state to the list if not yet in Q
  end while
  reset q
  ENEQUEUE q
end while
switch active states to non active list.
```

The word history can be stored in various ways. In practice, a word lattice is established which enables some powerful re-scoring methods. Multiple-pass search strategies are proposed, e.g. by Schwartz et al. [191]. For the sake of clarity, a tree based history is presented. This can be used for high speed embedded decoders. An illustration is provided in Figure 4.11.

The tree is stored as a hash array using smart pointers as described, e.g., by Andrei [192]. The use of smart pointers ensures an always updated memory without dedicated garbage collection. Each entity stores its usage by storing the number of references pointing to it. Whenever the counter becomes zero, the entity is tiered up. The tree is oriented to the root word w_0. Hence, all pointers are in root direction which makes it fast to retrieve the best suited word sequence. Each state in the search space, given by Q_s is pointing to the word history, respectively. Figure 4.11 shows an example tree. The reference counting is shown in small boxes next to each tree node. The one with zero will be removed automatically.

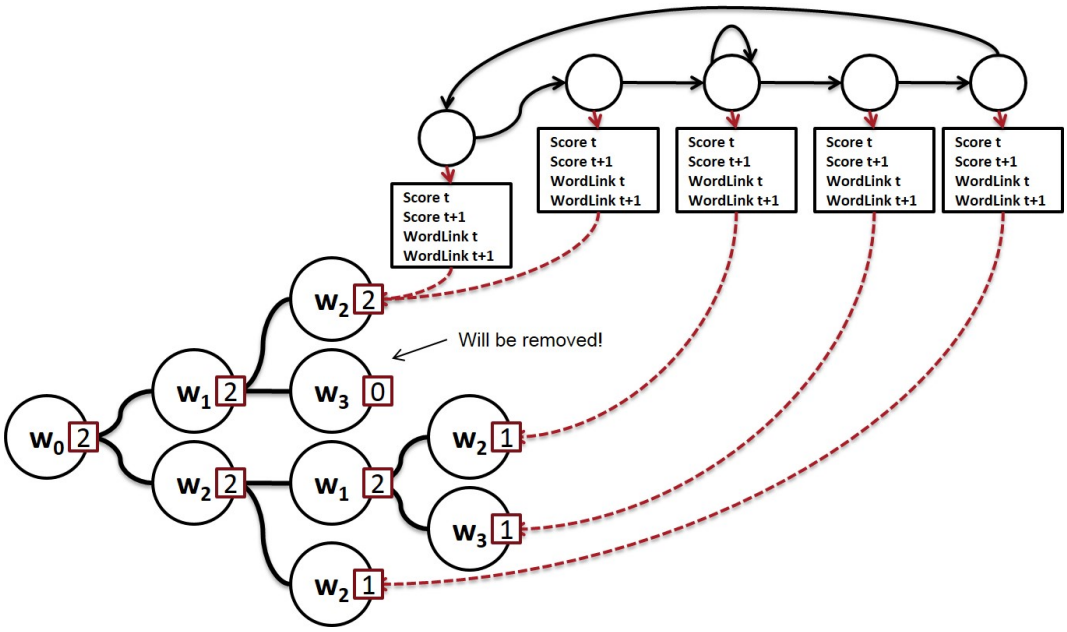


Figure 4.11: *Example search space on the top. The word history is shown on the bottom using smart pointers for automatic garbage collection.*

Figure 4.12 on the next page shows an example tree recording from a digit number recognition task. The used grammar was introduced in Figure 4.1. It is a German gender independent grammar with silence "[s]" between digits. There are 14 paths presented. One is "[s] sechs [s] zwei" and all others are initialized by "[s] sechs [s] eins". The number of pointers pointing to each node is denoted by "Ref = n", where n is the number of references.

In this section, a speech decoding technique was described. Starting from the fundamental formula of speech recognition, the construction of the preprocessed search space was presented. The preprocessing ensures an efficient speech decoding using weighed finite-state transducers. Together with the decoder overview an algorithm for a Viterbi beam search was described. The word history was stored in a tree based data structure.

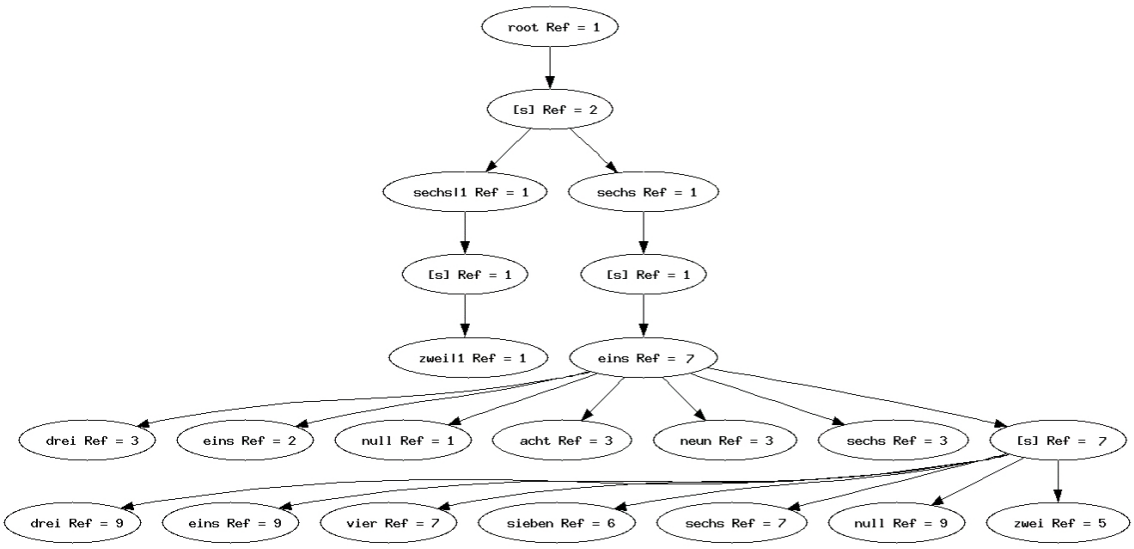


Figure 4.12: *Subset of a real word histroy result of a digit recognition task.*

4.6 Speech decoding of nested transducers

Using dynamic grammars in finite-sate transducers based speech recognition was proposed, e.g. by Schalkwyk et al. [193] Mohri proposes an algorithm to use local grammars in transducers [163]. In contrast, techniques are proposed to add vocabularies to word class models, e.g. by Dixon et al. [194]. A technique to add grammatical knowledge for improved speech recognition was proposed by Georges et al. [9]. A transducer nesting technique is described in this section which is applicable on embedded devices. This technique can be used to add user knowledge.

An n-gram language model $P(R(\underline{w}))$ is used which is represented as transducer G. Where R replaces all domain specific content by tag-symbols. This tag symbol is trapped during speech decoding to nest a second transducer. The construction of G is described in Section 4.3. The R function is discussed in Section 4.4.3. G is composed with a lexicon L. Note, the tag-symbol has to be added to the lexicon. The phoneme context and the hidden Markov model is composed and the entire search network is optimized. The construction is described in previous section 4.5.

Here, the search network G' is built by applying a set of transducer operators. The composition is denoted by $\circ$ as introduced in Section 2.4. The "det" and "min" operator is described in Section 2.2 and 2.3, respectively. G is computed as follows:

$$G' = \min(H \circ \det(C \circ \det(L \circ G)))$$

Furthermore, there is a transducer G'_k for each grammar G_k. Each G'_k has to be built in similar way to G'. It represents the domain specific content such as a user's address book or the user's favorite music titles. Here, the construction uses the same C and H transducer so that the same acoustic model can be used. In general, it is also possible to denote a different acoustic model. For the sake of clarity, one acoustical model is used and the same lexicon L. In practice, L is computed on the device using a comparable set of phonemes together with a grapheme-to-phoneme conversion tool. This enables to use user knowledge which is not available when the embedded system is deployed. The transducer for each G'_k is built as following:

$$G'_k = \min(H \circ \det(C \circ \det(L \circ G_k)))$$

The speech decoding schema is comparable to the one described in Section 4.5. In fact, there are no structural changes required. Even the computation is identical by alternatively computing the topological order and the beam search. A decoding based on the token passing time synchronous Viterbi beam search is used as described, e.g., by Young et al. [23]. For nesting a transducer on the fly, an additional "if" condition is included. It introduces a nested transducer whenever an output label denotes one. More detailed, every time a t_k output word in G' is reached, it continues by evaluating the initial state of the G'_k transducer. This nesting can be achieved by using the replacement operator for weighted finite-state transducers as a preprocessing step. The technique is described by Mohri et al. [169]. Alternatively a concept described by Schalkwyk for adding dynamic grammars can be used [193], as also described, e.g. by Mohri [163]. The proposed technique in this thesis nests a transducer G'_k during speech decoding in G' with negligible additional computational cost. An evaluation is provided by Georges et al. [9]. The decoding schema is outlined in Figure 4.13.

G' is represented by green states S^1 to S^6 which could represent an N-gram language model. The states in blue are representing the nested transducers G'_T and G'_E. When state S^2 is reached during decoding, the transducer G'_T is embedded instead of the "$\delta : T/0$" transition between state S^2 and S^5. State S^2 is connected with an $\epsilon : \epsilon/0$ transition with the initial state T^1 of G'_T. An "$\epsilon : \epsilon/0$" transition connects the final state T^2 with S^5. Hence, a δ transition between S^2 and S^5 is required to protect S^5 during determinization and minimization, analogously. G'_E is embedded between state S^3 and S^6 when S^3 is reached. Each transition with a δ input is omitted by the Viterbi search. All weights are zero in this example. In general, weights are used and individually scaled for each nested transducer.

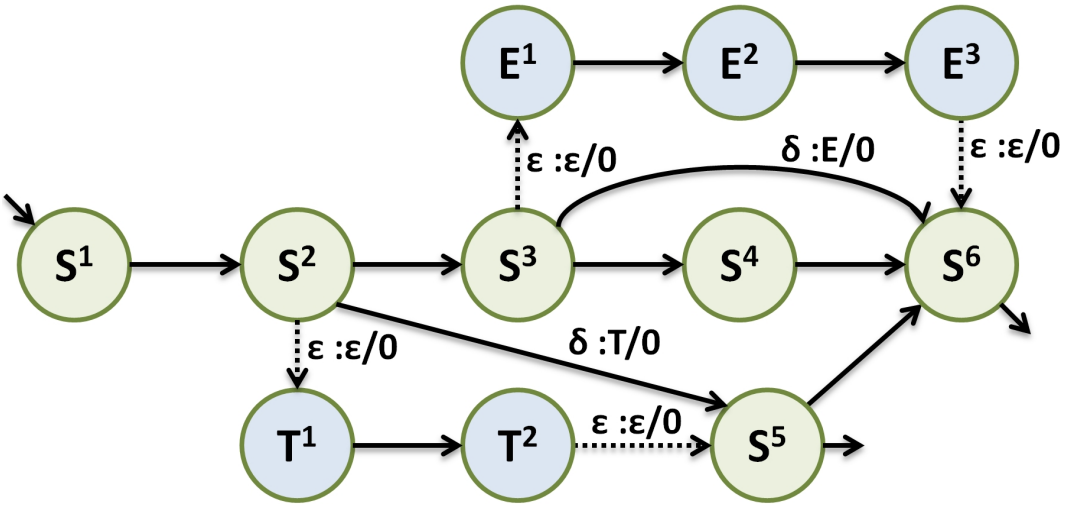


Figure 4.13: *Schematic Embedded transducer decoding.*

This construction is a hierarchical transducer nesting. Source and destination states of transitions with t_k output labels are marked. Those transitions have a δ input label, in order to prevent eliminations during determinization and minimization. The transducer G'_k which corresponds to t_k is embedded using an ϵ transition, if a marked state is reached. If a final state in G'_k is reached, an ϵ transition is taken back to the original transducer. This procedure requires just moderate modification in typical transducer frameworks. In addition, it is possible to omit not only the recognized word but also the origin transducer. Knowing the source may help identifying correlated semantic knowledge. For example, if the words origin comes from an address book transducer, the entire phrase might be more likely a command for voice activated dialing, e.g.

- i'd like to hear <Contacts>Marie</Contacts>

In contrast, if the words origin comes from a favorite music title transducer, it might be more likely a command for playing a song such as following example:

- i'd like to hear <Music_Artist>Jackson</Music_Artist>

Given the introducing example in Section 4.4.3, the abbreviation can be enclosed with some semantic tags as follows:

- about half these managers are in the <ABB>u. s.</ABB>

This nesting technique is applicable on devices with limited hardware without consuming much computational power and time. This could also introduce indetermin-

ism. Each transducer is deterministic and minimal, but the proposed transducer nesting method is a local operator without considering neighboring states and transitions. It might happen during search, that identical word hypotheses from various embedded grammars are active. Thus, a wider beam is required in order not to exclude other hypotheses. This effect is moderate for small $|S_K|$ and well predictable terms in S_k.

In this section, a technique is described which nests transducers on-the-fly. Its usage for speech recognition on embedded devices is described. This enables the use of user knowledge on the device.

4.7 Speech decoding on multiple devices

Using an on-the-fly transducer nesting technique enables some new applications. One of these is a speech decoding schema on multiple devices. Whereas some portion is recognized on one device, the rest might be recognized on a different one. This way, it is possible to use personal data without violating data privacy on the one hand. On the other hand, it reduces the effort for cloud bases speech services by avoiding memory expensive user-profiles. The technique was proposed by Georges et al. [10]. The following example illustrates speech decoding on multiple devices. A voice udialing application is considered with following sentence recognized on the cloud:

"Please call {NAME} on his|her cell phone"

where "{NAME}" was recognized by an acoustic filler model. Let "|" be an exclusive or. The result sentence is passed to the device. There the filler is replaced by user dependent knowledge. In this example, the users address book is introduced as follows:

"Please call Angelique|Isabel|Robert on his|her cell phone"

Where "Angelique", "Isabel" and "Robert" are names from the address book of the user. The entire sentence is used on the device to construct a finite-state transducer which is then used for speech recognition on the device, respectively.

Figure 4.14 shows a schematic overview. A first speech processing pass recognizes the natural language portion. In addition it identifies domain specific content such as entries from a user's address book. A second speech processing pass introduces domain specific content and finally recognizes the utterance. The technique uses dynamic language models as introduced in Section 4.3.3. A transducer nesting technique

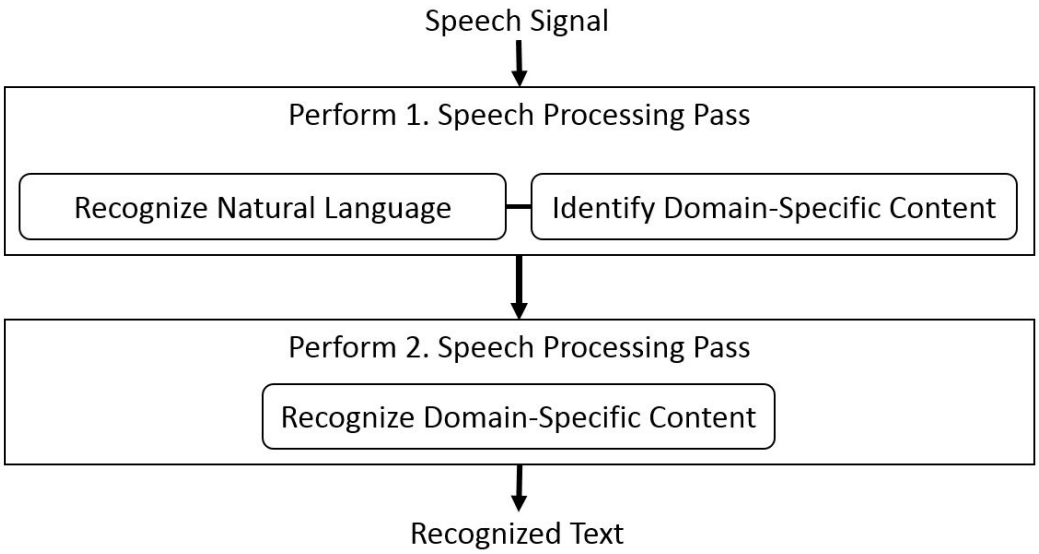


Figure 4.14: *Multiple pass speech recognition.*

is used as described previously in Section 4. In addition, a filler model is used which substitutes the nested transducer. It enables a postponed evaluation by replacing the filler model with the target transducer. The filler model is ideally derived from the target transducer. In practice, a generalized acoustical model is used.

Section 4.7.1 described the overview how multiple devices are connected to each other. The use of dynamic language models is described in Section 4.7.2. An introduction to the acoustical filler models is provided in Section 4.7.3.

4.7.1 Schematic overview for speech recognition on multiple devices

Recognizing speech on multiple devices is proposed using a dynamic language model. It is a hierarchical combination technique starting from less detailed to detailed. Each layer uses acoustical filler models from the layer below. Where each layer can be processed on a different device.

In this thesis, n-best lists are shared between decoders. This ensures backwards compatibility for existing cloud services. Alternatively, a word lattice can be shared. Hence, the natural language portion can be recognized on a speech server whereas the domain specific portion might be recognized in a car's head-unit or on a mobile phone using local available knowledge. The intention is to recognize speech wherever the data is localized. There is a huge amount of natural phrases available on the cloud.

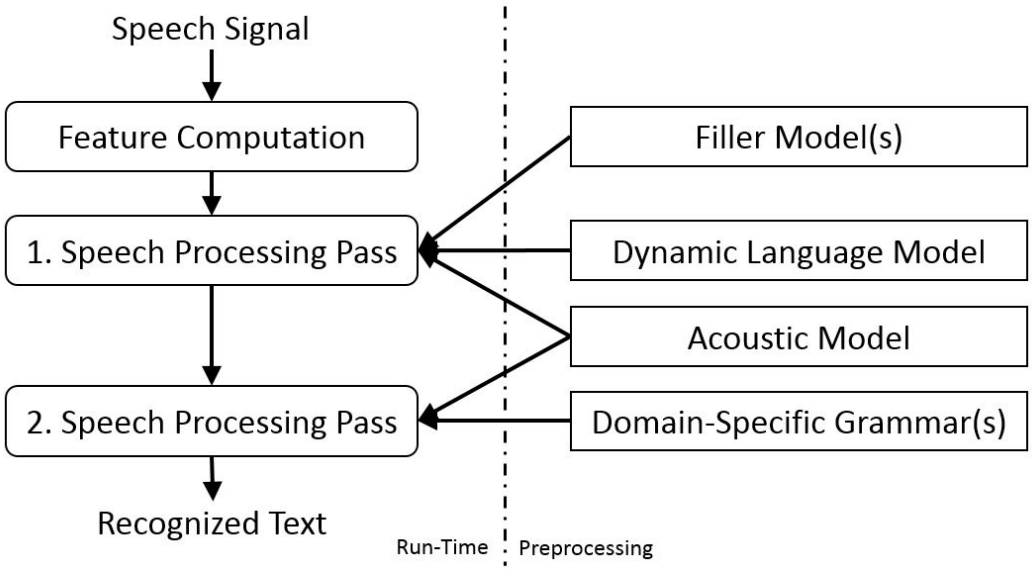


Figure 4.15: *Example data flow for speech recognition on two devices.*

Hence, the natural portion is best recognized on the cloud. In contrast, the users address book is available on a mobile phone. Hence, the domain specific content is best recognized on the device. In principle, this technique is not limited to two layers, e.g. two devices. It is also imaginable to use several recognizer, e.g. a smart phone for message dictation and a car-head unit to recognizes address entries.

In this thesis, a two layer implementation is described as proposed by Georges et al. [10]. The private data is kept on the device and also recognized on the device whereupon all other data, e.g. the carrier phrase is recognized on the cloud. The proposal is evaluated in section 5.3.

4.7.2 Use of dynamic language model

A dynamic language model is used together with the transducer nesting technique. Figure 4.15 gives an overview of the system. The first speech processing pass uses a dynamic language model as well as acoustic filler models. Each filler model replaces the "dynamic" content acoustically. The second speech processing pass processes the recognized word sequence from the first pass. In addition it introduces the domain specific content. Even though the acoustic model is shared in the figure, in principle, both passes can use different models. In fact, this is the case when the first processing pass is done on the cloud whereas the second one is done on an embedded device.

In this thesis, the transducer nesting technique is used together with acoustic filler models and domain specific content. As an example, a decoding on two devices is described in the following. Let G' be the transducer for recognizing the natural portion of an utterance. It is built by composing a language transducer G and a lexicon L. G is trained on a corpus with substituted domain-specific content. In addition, a phoneme context C and a hidden Markov model H is used. This was introduced in previous sections. G is constructed in the following way:

$$G' = \min(H \circ \det(C \circ \det(L \circ G)))$$

"NAME" would be substituted by an acoustic filler for the example sentences in Section 4.7. The decoding schema is analogous to the one described in Section 4 except that acoustic filler models are nested. An n-best sentence list is the result and passed to a second device. All sentences are used to create the grammar G_t where the filler denotes the domain specific content. It is possible to add some language model weights. Also the domain specific content which is introduced afterwards on the fly is not weighted here. This reduces the latency caused by the search transducer construction G''. Here, the same L, C and H transducer is used for the sake of clarity. In general, it is possible to use different transducers which enables the use of various acoustic models. The transducer G'' for the second device is constructed as follows:

$$G'' = \min(H \circ \det(C \circ \det(L \circ G_t)))$$

This computation happens on-the-fly. The domain specific content, e.g. G'_k, is also represented as transducer:

$$G'_k = \min(H \circ \det(C \circ \det(L \circ G_k)))$$

It is pre-processed on the client. The decoding on the second device uses the transducer nesting technique to nest G'_k in G'' on the fly. The result is a word sequence including both, the carrier phrase as well as the domain specific content.

In this section, the use of dynamic language modeling was described for recognizing speech on multiple devices. As example, the domain specific content of a natural phrase was recognized on a different device. The technique was described using finite-state transducers. It uses the transducer nesting technique.

4.7.3 Use of acoustic filler

Recognizing speech on multiple devices requires some abstraction on levels where not all knowledge is present. Here an acoustic filler model is proposed which substitutes the missing context acoustically.

Using filler models is widely used in speech recognition. It is also used to compute the confidence of recognition hypothesis. Adding new words automatically was proposed by Asadi et al. [195]. Modeling unknown words for speech recognition was described, e.g., by Kemp et al. [196] or Jiang et al. [197]. Klakow et al. proposes the use of word fragments as fillers for detecting out of vocabulary words [198]. A general model for out of vocabularies was proposed by Bazzi et al. [199]. The use of hierarchical Markov language model was proposed by Kokubo et al. [200]. An n-gram based filler model was proposed by Yu for grammar authoring [201]. Using flat hybrid models was proposed by Bisani et al. for open vocabulary speech recognition [202]. Qin et al. uses hybrid models with different fragments for unknown word detection and recovery [203]. In this thesis a phoneme 2-gram model is used. It is trained on phoneme sequences $\underline{\rho}$ from domain specific content as follows:

$$P(\underline{\rho}) = \left(\frac{F(\rho_{i-1}\rho_i)}{F(\rho_{i-1})} \right)^{\lambda_{AF}}$$

Where $F(\rho_{i-1}\rho_i)$ denotes the frequency of the 2-gram. A phoneme transition penalty λ_{AF} was used to further improve the recognition. This parameter was tuned on some tuning data. The model can be enhanced using smoothing techniques for unseen phoneme sequences. This is similar to the methods described in 4.3.2. A motivation for the usage of phonemes is given, e.g. by Savic et al. [204]. Here, a word model was investigated as "oracle" model as described by Georges et al. [10] and summarized in Section 5.3. The acoustic filler model was trained on its representing domain specific content, e.g. an address book.

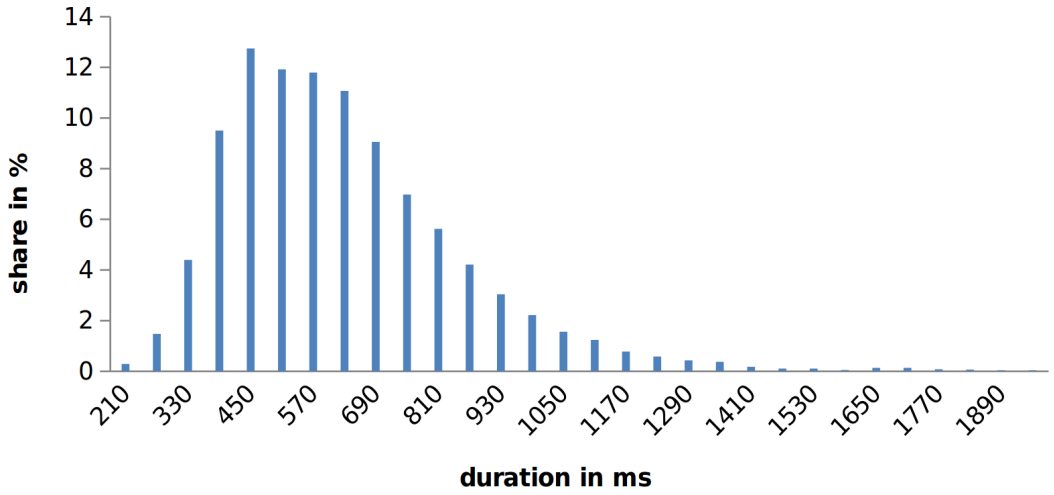


Figure 4.16: *The word duration measurement for the Wall Street Journal corpus indicates a Poisson distribution.*

4.7.4 Word duration modeling for slot fillers

The duration of filler models may have an impact on the recognition performance. In this thesis a word penalty is used, but in general more advanced technique are imaginable. This is motivated by Jennequin et al. [205] where a duration model is used during lattice re-scoring. A similar approach was also proposed by Dino et al. for large vocabulary speech recognition [206]. Kao et al. proposes a discriminative duration models for speech recognition with segmental conditional random fields [207].

The filler models in this thesis uses a transition penalty similar to a word start penalty in the language transducer. However, not only the filler model can use a duration model also the introduced domain specific content can use dedicated recognizers. A word duration model, e.g. for large address books, might improve speech recognition. In Figure 4.16 is a word duration statistic presented. It was derived from a recognition on the Wall Street Journal corpus [39]. The measure indicates a Poisson distribution of the word length. However, most words have a duration between 400ms and 700ms. This is often approximated with an uniform distribution.

This section introduces a word duration modeling for speech recognition. It was observed that the word duration differs significantly. This may have an impact on the recognition accuracy. A word transition penalty is therefor introduced and transferred to the acoustic filler models. In addition, the word transition penalty was also proposed to improve the recognition of domain specific content.

4.8 Summary

This chapter introduced a novel method to model dynamic content for language and speech processing. Dynamic content refers to word sequences which are sparsely represented in common training data. This could be user-dependent word sequences or words which are changing frequently. Examples are address book entries, medical records or stock prices. It is also possible to model word sequences with long range dependencies using a dynamic language model. In addition, a technique is described that enables the use of dynamic language models in low resource speech recognition. This could be speech recognition on mobile phones, car head-units but it could also be speech recognition on a highly scalable cloud infrastructure. The technique is based on weighted finite state transducers and uses a transducer nesting technique.

Principles of transducer based speech recognition are introduced in Section 4.1 The transducer for automatic speech recognition is composed of several automatons and transducers. A lexicon transducer defines the relation between grapheme and phoneme sequences given words. It is described in Section 4.2 as a tree-like representation for fast look-up of phonetic transcriptions. The language automaton represents the knowledge of word sequences, e.g. to determine whether a word sequence is probable, or not. The transducer is derived form statistical n-gram language models or grammars. Section 4.3 describes the developing process in detail.

The embedding of grammars in statistical models to model dynamic content is described in Section 4.4. Speech decoding with transducers is introduced in Section 4.5 together with an algorithm for efficient decoding. It is based on a token passing time synchronous Viterbi beam decoder which is implemented for low resource speech recognition, e.g. on embedded devices. A tree structured word history is proposed with a continuous garbage collector. Section 4.6 extends the decoding schema by a transducer nesting technique. This enables speech recognition using several transducers where the nesting happens during run-time as proposed by Georges et al. [9]. A transducer can be compiled on the device and nested into the deployed transducer which models carrier phrases. The recognition with nested transducers allows speech decoding on multiple devices as proposed by Georges et al. [7], [6], [8]. For this, acoustical filler models are used which enable to evaluate some portion of the utterance on a different device. This technique is described in Section 4.7.

5 Applications

There are many applications using speech recognizers. Today, short message dictation and voice search is available on nearly all smart phones. All high class navigation systems are offering voice destination entry. The methods and techniques described in this thesis are contributing to the developing of humanoid voice interfaces. In fact, these allow a voice command to be queried in a natural language. This chapter gives an overview of evaluation metrics and evaluates the transducer nesting technique as well as the technique for recognizing speech on multiple devices.

The chapter is organized as follows; Section 5.1 introduces evaluation methods for language processing. The perplexity measure is described for language model evaluation. Speech recognition accuracy is measured by the word and sentence error rate. In addition, some evaluation metrics for information retrieval tasks are defined.

Section 5.2 describes an application which embeds grammars in n-gram language models. For this the transducer nesting technique was used as described in Section 4.6. The evaluation was proposed by Georges et. al [9] for the use on embedded devices.

The decoding on multiple devices is evaluated in Section 5.3.5. This was proposed by Georges et al. [10]. It is an application to keep personal data on the client. These data is also recognized on the client whereas the rest of the utterance is recognized on the cloud. Finally, the chapter is summarized.

5.1 Evaluation of speech and language techniques

There are two aspects in evaluating speech and language techniques. First, it should measure how good a technique suites the assumption. Second, it should assess the human experience using the technique. Both is not always achieved by one measure. This section outlines evaluation methods used for developing speech applications.

Evaluating a language mode covers two questions; First, how well does a language model suits the assumption, e.g. evaluate the model on training data. Second, how good generalizes the model, e.g. test the model on test data. A language model is evaluated by computing its perplexity. This is described in Section 5.1.1. Whereas the language model performance is not visible to the user, the accuracy of the speech recognition is perceptible by the user.

The evaluation of a speech recognizer should measure an user's experience. The word or sentence error rate is a good measure for many speech applications, such as short message dictation. It is the ratio of wrongly recognized words or sentences overall words or sentences, respectively. The measure is described in Section 5.1.2. Evaluation methods known from the information retrieval research are used, too. It is used to evaluate applications such as voice destination entry or command and control tasks. In addition, note that a web search engine on a tablet computer may cause a different user experience than similar engine on a car head-unit. A set of standard metrics is described in Section 5.1.3.

5.1.1 Language model evaluation

The cross-entropy for a word sequence $\underline{w}$ with N_w words can be approximated as

$$H(\underline{w}) = -\frac{1}{N_w}\log_2(P(\underline{w})).$$

where $P(\underline{w})$ is the probability of a word sequence $\underline{w}$. The sequence $\underline{w}$ should be sufficient long, e.g. a test-corpora. The reciprocal average of the geometric probability for each word form a sentence $\underline{w}$ defines the perplexity measure:

$$PP(\underline{w}) = 2^{H(\underline{w})}$$

It is the geometric mean of the transducers branching factor. Moreover it can measure the generalization capabilities of a statistical language model.

The perplexity is computed on a suitable test-set. The test-set should match the user's expectation on the one hand. On the other hand, the test set should not be part of the training set. Both, test and train will give a good estimation on the overall performance. Variations of the perplexity measure were proposed, e.g. by Bimbota et al. [208] or Chen et al. [209]. In some sense, a lower perplexity correlates with the speech recognition accuracy on the same test set. In contrast, the perplexity does not consider the confusing between words acoustically. This fact makes a speech recognition evaluation unavoidable.

5.1.2 Evaluation of speech recognition

Speech recognizers are often evaluated by the Levehnstein distance between hypothesis and reference. This is a de facto standard evaluation. For which the reference was transcribed by human experts. There are three types of errors:

Substitution:	an incorrect word was substituted for the correct word
Deletion:	a correct word was omitted in the recognition sentence
Insertion:	an extra word was added in the recognition sentence

This is in line with the introduction provided by Huang et al. [66]: A matrix D is used to compute the minimum path for a recognized word sequence $\underline{w}$ and its reference $\underline{u}$. This matrix is computed by the following equations:

$$
\begin{aligned}
D_{0,0}(\underline{w},\underline{u}) &= 0 \\
D_{i,0}(\underline{w},\underline{u}) &= i \text{ if } 1 \le i \le |\underline{w}| \\
D_{0,j}(\underline{w},\underline{u}) &= j \text{ if } 1 \le j \le |\underline{u}| \\
D_{i,j}(\underline{w},\underline{u}) &= \begin{cases}
D_{i-1,j-1} & +0 \quad \text{if } \underline{w}_i = \underline{u}_i \\
D_{i-1,j-1} & +1 \quad \text{if substitution} \\
D_{i,j-1} & +1 \quad \text{if insertion} \\
D_{i-1,j} & +1 \quad \text{if deletion}
\end{cases}
\end{aligned}
$$

for $1 \le i \le |\underline{w}|$ and $1 \le j \le |\underline{u}|$. Where $|\cdot|$ denotes the number of words in the sequence. A weighted finite-state transducer could be used to compute the distances between $\underline{w}$

Algorithm 5.1 *Pseudo code for computing the levenshtein matrix*

```
initialize matrix d                                   // The result will be in d
for i index s_1 do                                    // iterate over all words in s_1
    d[i,0] ← 1                                        // Add a identity substitution
end for
for j index s_2 do                                    // iterate over all words in s_2
    d[0,j] ← 1                                        // Add a identity substitution
end for
for j index s_2 do                                    // iterate over all words in s_2
    for i index s_1 do                                // iterate over all words in s_1
        if s_1[i] == s_2[j] then                      // current words are identical
            d[i,j] ← d[i-1,j-1]                       // denote the identical words
        else                                          // current words are not identical
            d[i,j] ← min(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+1)
        end if
    end for
end for
```

and $\underline{u}$, e.g. proposed by Mohri [210]. In this thesis, Algorithm 5.1 was used to compute the matrix D together with a backtracing algorithm to retrieve the best path.

The Levehnstein distance is defined by:

$$\mathrm{Lev}(\underline{\mathrm{w}},\underline{\mathrm{u}}) = \mathrm{D}_{|\underline{\mathrm{w}}|,|\underline{\mathrm{u}}|}(\underline{\mathrm{w}},\underline{\mathrm{u}}).$$

It assigns the minimal number of errors, which are required to transform the hypotheses to the reference word sequence. The word error rate is the ratio between the Levehnstein distance and the number of correct words:

$$\begin{aligned}\mathrm{WER}(\underline{\mathrm{w}},\underline{\mathrm{u}}) &= \frac{S(\underline{w},\underline{u}) + D(\underline{w},\underline{u}) + I(\underline{w},\underline{u})}{C(\underline{w},\underline{u})} \cdot 100 \\ &= \frac{\mathrm{Lev}(\underline{\mathrm{w}},\underline{\mathrm{u}})}{C(\underline{w},\underline{u})} \cdot 100\end{aligned}$$

It is often helpful to see which word was recognized wrongly to investigate a recognizer output. The backtracing is shown in Algorithm 5.2. Note that the result is just one trace back. In general there might be several optimal paths with different back traces and identical overall distance. The word error rate will be consistent but the number of insertions versus deletions might differ from path to path. Let c denote a correct match. d denotes a deletion and a substitution is denoted by s. An example is the following hypothesis:

Algorithm 5.2 *Pseudo code for backtracing levenshtein matrix*

```
if i > 0 ∧ D[i−1, j]+1 = D[i, j] then                    // determine a deletion
   backtrace(D,i-1,j )+" D "                              // Mark a deletion
end if
if j > 0 ∧ D[i, j−1]+1 = D[i, j] then                    // determine a insertion
   backtrace(D,i ,j-1)+" I "                              // Mark an insertion
end if
if i > 0 ∧ j > 0 ∧ D[i−1, j−1]+1 = D[i, j] then          // determine a substitution
   backtrace(D,i-1,j-1)+" S "                             // Mark a substitution
end if
if i > 0 ∧ j > 0 ∧ D[i−1, j−1] = D[i, j] then            // determine a correct path
   backtrace(D,i-1,j-1)+" C "                             // Mark correct words
end if
```

- about half managers art in the u. s.

for the reference

- about half these managers are in the u. s.

Where on error path might be

- c c d c s c c c c

The word "these" is missing in the hypothesis and the word "are" was mis-recognized as "art". However, an other error path could be:

- c c s s d c c c c

The differences are small and usually unimportant since the same back-tracing methods is applied on all results. However, it might make a difference when computing distances between two hypothesis, e.g. for word confidence measuring.

The accuracy of a recognizer is defined as inverse to the word error rate as follows:

$$\mathrm{ACC}(\underline{w},\underline{u}) = 1 - \mathrm{WER}(\underline{w},\underline{u}) = \frac{C(\underline{w},\underline{u}) - S(\underline{w},\underline{u}) - D(\underline{w},\underline{u}) - I(\underline{w},\underline{u})}{C(\underline{w},\underline{u})} \cdot 100$$

Note that ACC may have values larger 100%. This is intuitively correct, since the recognizer might introduce much more words than spoken, actually. If the number of insertions is set to zero $I = 0$, the ACC becomes a recall value in sense of information retrieval. In addition, this measure is normalized. The correctness is defined by:

$$\mathrm{COR}(\underline{w},\underline{u}) = \frac{C(\underline{w},\underline{u}) - S(\underline{w},\underline{u}) - D(\underline{w},\underline{u})}{C(\underline{w},\underline{u})} \cdot 100$$

Alternatively, the sentence error rate can be used, which does not take the length of a sentence into account. Let S_F be the number of wrongly recognized sentences and S is the number of overall sentences. It is defined as follows:

$$\text{SER} = \frac{S_F}{S}$$

The measure of word and sentence error is well established, but not unrivaled for certain applications. Other alternatives were also proposed. Incorporating humane performance of recognizing speech was proposed, e.g. by Morris et al. [211]. Other metrics are proposed, e.g. by Mishra et al. [212]. Favre et al. proposed an alternative evaluation metric to Word Error Rate for the decision audit task of meeting recordings [213]. Note that other languages may require other metrics, e.g. mandarin is evaluated on character error rate and for Japanese an error scoring adapted to Katakana is used.

5.1.3 Evaluation of information retrieval

The precision is the ratio between correctly retrieved documents and the number of all retrieved documents. It can also be defined as ratio between the true positives and the sum of true positives and false positives as follows:

$$P = \frac{|\{\text{relevant documents} \cap \text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} := \frac{tp}{tp + fp} \tag{5.1}$$

Let tn and fp, fn and fp be the number of true positives and negatives and the number of false positives and negatives, respectively. In contrast, the recall is the ratio between correct retrieved documents and relevant documents. It is defined as follows:

$$R = \frac{|\{\text{relevant documents} \cap \text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} := \frac{tp}{tp + fn} \tag{5.2}$$

Figure 5.1 illustrates the precision and recall measure. The F-measure derives one value from precision P and recall R as follows:

$$F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R}$$

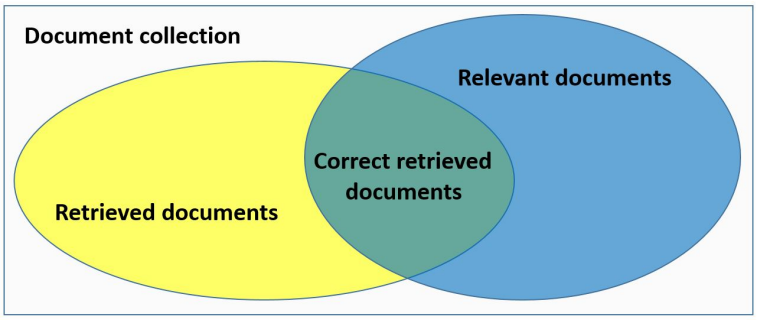


Figure 5.1: *Illustration of an information retrieval process.*

where $\beta \geq 0$. The balanced F-measure is a harmonic mean of precision and recall with $\beta = 1$. This F_1-measure is typically chosen:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

The Jaccard similarity coefficient is defined between set A, e.g. the set of retrieved documents and a set B, e.g. the set of relevant documents. It measures the ratio over correct retrieved documents over the union of both, relevant and retrieved documents. It is defined as:

$$J = \frac{|A \cap B|}{|A \cup B|} \tag{5.3}$$

with $J(A,B) = 1$ if $A = \emptyset$ and $B = \emptyset$ so that $0 \geq J(A,B) \geq 1$. Hence, the coefficient is 1 for identical sets and 0 when there is no compliance, e.g. the sets are different. Matthews proposed a correlation coefficient that uses true and false positives and negatives to determine the quality of binary classification [214]. The Matthews correlation coefficient is defined as follows:

$$MCC = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{5.4}$$

It is balanced and can be used even for strongly varying sample sizes for each class. The range is $-1 \geq MCC \geq +1$, where $+1$ means a perfect prediction, 0 is just random and -1 indicates total disagreement between prediction and observation. This can be generalized for multi dimensional purposes as described, e.g. by Jurman et al. [215].

5.2 Dynamic Language Model evaluation

This section proposes to embed universal grammars into N-gram Markov language models. This allows an accurate speech recognition even for N-gram models estimated on sparse grammatical word sequences. The technique was published by Georges et al. [9]. It allows explicit and user-dependent modeling of content, such as phone numbers, email addresses or US ZIP codes separately from the Markov model.

The method is described along with a feasible implementation overview. More precisely, a language model preprocessing step generalizes the enclosed grammatical word sequences during language model learning. These grammars are embedded during speech decoding by using a novel transducer nesting technique. The Wall Street Journal corpus was used to evaluate the proposed method. A word error rate reduction of 31.1% was achieved. A computational environment was used, which is typical for car head units or mobile devices

5.2.1 Introduction

The use of grammars is typical for various speech enabled applications. An example is voice destination entry. Also voice controlled car head units are using grammars to model dialogs for embedded speech recognition. In the future, grammars are embedded into natural sentences. This enables a more intuitive human-machine interaction by understanding utterances formulated in a natural language. Whereas grammars are well suited for structured informations, Markov models are suitable for natural languages. Word class based N-gram model were introduced in Section 3.4.3.

Georges et al.[9] proposed to embed grammars into N-gram Markov models. For this, a transducer nesting technique was used which allows to consider different transducers during speech decoding. It enables the use of grammars together with N-gram Markov models. In this way, a generalized Markov models for speech recognition is realized. This enables accurate speech recognition, even for N-gram models which are estimated on sparse data. This is relevant for developing applications without the need of expensive and time-consuming data collections. Furthermore, the transducer nesting technique allows a late customization with universal grammars as well as the integration of user-dependent knowledge. The proposed recognizer uses the nesting technique as described in Section 4.6. Section 5.2.2 presents a preprocessing which is

used to embed grammars into Markov models. Section 5.2.3 evaluates the proposed method on the Wall Street Journal task [167]. This has shown that a significant improvement can be achieved with the proposed technique. The used computational environment was comparable with typical car head-units or mobile phones.

5.2.2 Transducer nesting for dynamic language models

A preprocessing step replaces word sequences from a text corpus with corresponding tag labels. The word sequences are represented by grammars. The replacing method R is applied for each sentence in a corpus $\mathbb{C}$:

$$\mathbb{C}' = \{\underline{w} \in ((W \cup T)^* \backslash S_K)^* | \exists \underline{w}' \in \mathbb{C} : \underline{w} = R(\underline{w}')\}.$$

$\mathbb{C}'$ is used to estimate the N-gram language model. Section 5.2.3 evaluates the correlation between the number of substituted word sequences and the overall recognition accuracy. It has been shown that sparse grammatical word sequences are sufficient.

R can be computed in various ways. Here, R is computed by a finite-state transducer as motivated by Karttunen [182]. The technique was described in Section 4.4.3. Let S be a set of tuples (U, t) which associates each set $U \subseteq S_K$ with a corresponding grammar tag $t \in T$. This can be formalized as:

$$S = \{(U, t) | \exists k : U = S_k \wedge t = t_k\}$$

The transducer used to compute R represents the relation between word sequences in S_k and grammar tags t_k. Each sentence in S_K is assumed to be unique and the following equation is fulfilled:

$$|S_K| = \sum_k |S_k|$$

Nevertheless, there could still be ambiguity between partial sentences. In this case, the longest word sequence is selected, which can be computed by the smallest common substring. This is formalized by the following equation:

$$\underline{\hat{w}} = \min_{k \leq K} (\operatorname*{argmax}_{\underline{w} \in S_k} (\underline{w}))$$

Where max/min is the sequence with the maximal/minimal number of words. This can be computed recursively. The replacing method R is defined as follows:

$$\underline{w} \mapsto R'(\emptyset, 1, 1)$$

The computation starts with an empty result word sequence, the word w_1 at position 1 and a step size of 1. Four cases have to be considered. First, no initial part of any word sequence in S_K was observed. Second, a word sequence in S_K was partly observed. Third, a complete word sequence in S_K was observed and can be replaced with its corresponding tag symbol. Fourth, the remaining word sequence will not match any word sequence in S_K. This can be formulated for $i + j \leq |\underline{w}|$ as follows:

$$R'(o, i, j) = \begin{cases} R'(o\, w_{i:j}, i+j, 1) & \exists v \in W^+ : w_{i:j}\, v \notin S_K \\ R'(o, i, j+1) & \exists v \in W^+ : w_{i:i+j}\, v \in S_K \\ R'(o\, t, i+j, 1) & \begin{cases} \exists (U, t) \in S : w_{i:i+j-1} \in U \\ w_{i+j} \notin U \end{cases} \\ o\, w_{i,|\underline{w}|} & else. \end{cases}$$

This equation can be solved by using an extended finite-state transducer. Usual transducer implementations are using global declared functions to compute the input and output, e.g. described by Mohri et al.[13], [14]. If an input matches, the output word from the corresponding label is generated. This construction would require knowing all words in advance.

The transducer which is used here, declares local functions $IN : W \mapsto \{0, 1\}$ along with an output function $OUT : W \mapsto W \cup T$ for each transition. This robust finite-state transducer implementation was described in Section 2.7. Furthermore, the input word is shared over the input and output function. This enables an open vocabulary processing with transducers. Every time when an input function evaluates to 1, the corresponding output is computed and the new state and set of transitions can be evaluated as usual. This transducer is well suited for computing R, because just the vocabulary of S_K has to be known. All other words are treated equally and are passed through to the output.

5.2.3 Evaluation

The proposed technique is evaluated using the Wall Street Journal which is described by Paul et al. [167]. A computational environment with limited hardware resources was used. A similar one is typically used by car head units or mobile phones. In fact, the acoustic model was evaluated with integer values and no advanced acoustic adaptation technique was applied such as MLLR etc. Both the acoustic and language models have to be as small as possible to fulfill memory requirements of the embedded devices. The SRI Language Model toolkit proposed by Stolcke [216] was used to learn a 5000 word 2-gram language model.

The intention of the proposed technique is the use of specific grammatically given language knowledge. For this purpose, the corpus was preprocessed to simulate various grammar coverages. Five grammars were used to evaluate the proposed method on the Wall Street Journal corpus. Each grammar represents one of the following sets. The set of all monetary terms is denoted with $S_\$$ and the set of percentages is denoted with $S_\%$. S_κ denotes the set of abbreviations. Table 5.1 gives a detailed definition in Backus–Naur form. Also a statistical N-gram model for abbreviations was used.

The result was a slightly degraded speech recognition accuracy over all experiments. The set of Months is denoted with S_μ and S_η denotes the set of Weekdays. Further, let $S_i' \subseteq S_i$ with $i \in \{\$, \%, \mu, \eta, \kappa\}$ be the observable portion of S_i in the test corpus. Each set can be used alone or in combination with others. Here, there are $32 = \binom{5}{1} + \binom{5}{2} + \binom{5}{3} + \binom{5}{4} + 1$ combinations.

Table 5.1: *Overview of grammars in Backus-âĂŞNaur Form*

⟨*NUM*⟩ ::= ⟨*num*⟩ [‘`point`’ ⟨*num*⟩][‘`comma`’ ⟨*num*⟩]

⟨*num*⟩ ::= ‘`one`’ ⟨*num*⟩ | ‘`two`’ ⟨*num*⟩ | ... | ⟨*empty*⟩

⟨*MONETARY*⟩ ::= ⟨*NUM*⟩ ‘`dollar`’ | ⟨*NUM*⟩ ‘`cent`’ | ...

⟨*PERCENT*⟩ ::= ⟨*NUM*⟩ ‘`percent`’

⟨*ABBREVIATION*⟩ ::= ⟨*Letter*⟩ | ⟨*ABB*⟩

⟨*Letter*⟩ ::= ‘`A.`’ ⟨*Letter*⟩ | ... | ⟨*empty*⟩

⟨*ABB*⟩ ::= ‘`SVOX`’ | ...

Let π be one arbitrary combination. This defines the set S_π, S'_π as follows:

$$S_\pi = \bigcup_{\forall i \in \pi} S_i \text{ and } S'_\pi = \bigcup_{\forall i \in \pi} S'_i.$$

S'_π is used to adjust the grammar coverage in the text corpora which is then used for language model learning. This enables a realistic application with similar coverages. The corpus $\mathbb{C}_\pi$ includes each sentence of $\mathbb{C}$ which did not contain any term from S'_π:

$$\mathbb{C}_\pi = \{\underline{w} \in \mathbb{C} | \nexists m, n : w_{m:n} \in S'_\pi\}.$$

Each corpus $\mathbb{C}_\pi$ was used to learn a language model for a traditional speech recognizer. The proposed technique requires one further step. R is applied which replaces every term of S_π with a corresponding grammar tag:

$$\mathbb{C}'_\pi = \{\underline{w} \in ((W \cup T)^* \backslash S_\pi)^* | \exists \underline{w}' \in \mathbb{C}_\pi : \underline{w} = R(\underline{w}')\}.$$

Using Then, a language model for each $\mathbb{C}'_\pi$ was learned, too. Both the N-gram model and grammar of S_π were then used to evaluate the proposed technique by means of recognition accuracy.

In addition, define the corpus size is defined as the number of sentences in percent compared with the original corpus. Let the grammar coverage for $\mathbb{C}_\pi$ be defined as the percentage of sentences which contain a term from S_π:

$$\frac{100 \cdot |\{\underline{w} \in \mathbb{C}_\pi | \exists m, n : w_{m:n} \in S_\pi\}|}{|\mathbb{C}_\pi|}.$$

The test corpus has a grammar coverage between 3.64% and 40.61% depending on the considered S_π. Different S'_π for $\mathbb{C}_\pi$ causes a tag coverage between 6,72% and 36,44%.

Best recognition accuracy is achieved if the grammar coverage in $\mathbb{C}_\pi$ is high while the coverage in the test corpus is low. Conversely, the error rate increases if the grammar coverage in $\mathbb{C}_\pi$ is low while the coverage in the test corpus is high. In this way a word error rate of 19.3% was obtained. The effect can be reduced by using dynamic language models. This has been confirmed by the experiments. A word error rate of 13.3% was achieved with a low grammar coverage of 6.72% by using the proposed dynamic language model. Figure 5.2 shows the results for each experiment. The novel technique is beneficial for situation with a low grammar coverage.

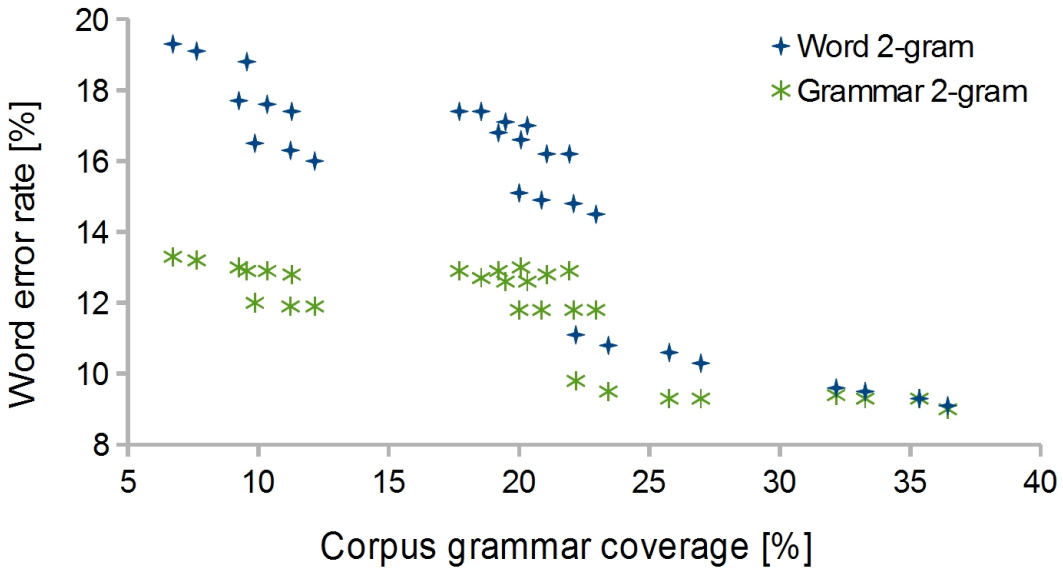


Figure 5.2: *Using* 2*-gram models reduces the word error rate by up to* 31% *for low grammar coverage. No significant word error rate improvements achieved for high grammar coverages of* 36%.

Another aspect next to grammar coverage is the corpus size, which was used to learn the language model. An increasing error rate for decreasing corpus size was expected. The proposed technique could as well prevent this deterioration. An analysis is provided on the next page in Figure 5.3.

Finally, the influence of the corpus size for constant grammar coverage was analyzed. Two use cases are presented here. In the first case, the grammar coverage is good, with 36.44%. Here, no further improvements could be achieved. The corpus randomly shrunk for each speech recognition experiment. No improvements were observed for various corpus sizes. This can be seen on the next page in Figure 5.4.

In the second case, the grammar coverage is low (6.86%±0.22% variance). Here, an improvement could be achieved using the proposed technique for large corpora. The result was a 6% reduction in word error rate from 19.3% to 13.3%. The achievement is still observable for small corpora, although the relative improvement decrease in this case. The baseline system has a word error rate of 27.0% using a word 2-gram language mode, while the proposed technique with the proposed grammar 2-gram model achieves 23.5% word error rate. The results are shown in Figure 5.5 on the next page.

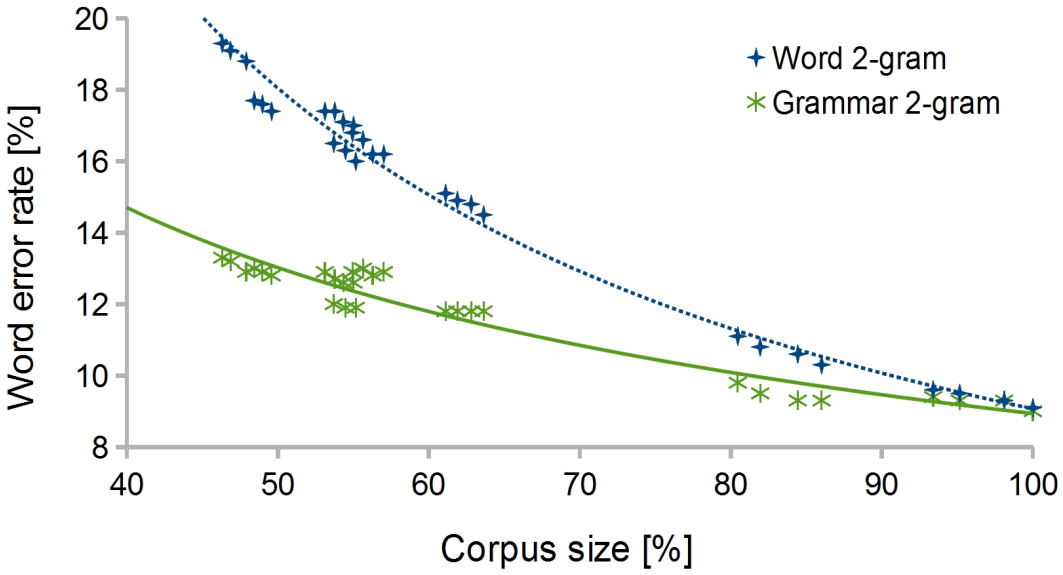


Figure 5.3: *The novel technique improves speech recognition for domains with a limited amount of train data. The word error rate could be reduced from* 19% *to* 13% *using a grammar* 2*-gram model for small corpora.

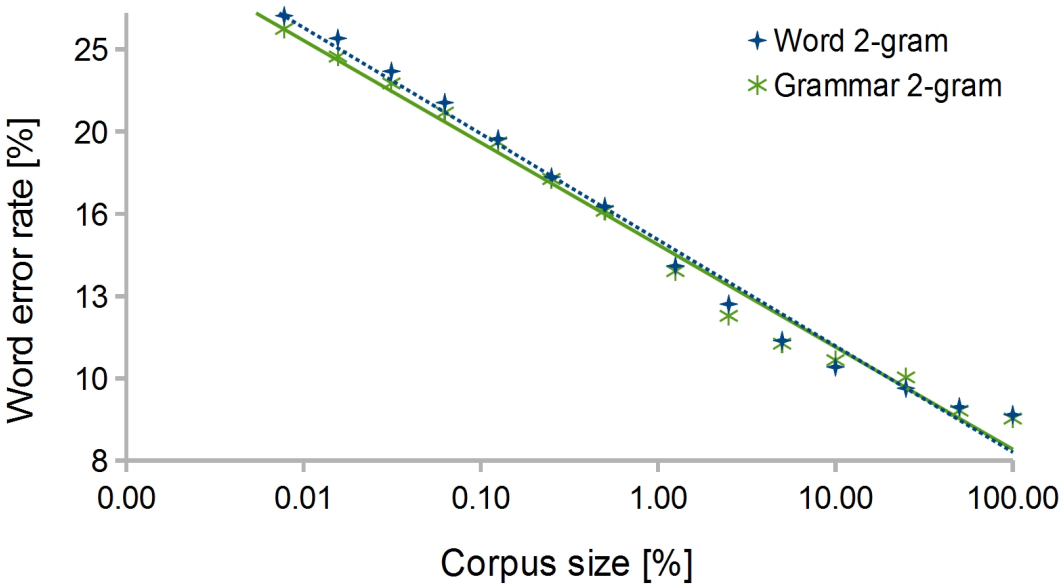


Figure 5.4: *No improvement achieved for a constant grammar coverage of* 36%*. The word error rate increases with a potential dependence for randomly shrunk corpus.*

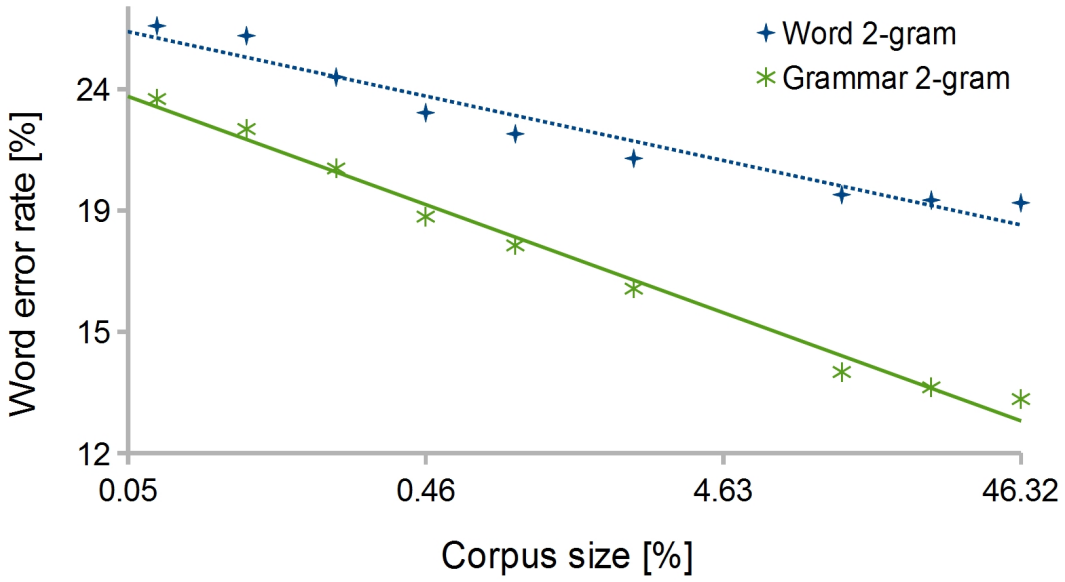


Figure 5.5: *A constant grammar coverage of* 7% *leads to a relative word error rate reduction from* 31% *to* 12% *for a randomly shrunk corpus.*

5.2.4 Conclusion

This section evaluated a technique which generalizes N-gram Markov language models with grammars. For this, grammatical word sequences were generalized during language model learning. The suitable grammars are then embedded dynamically during speech decoding using a transducer nesting technique. It is feasible to use universal, application-independent grammars or user-dependent grammars.

The evaluation shows that the novel technique can improve speech recognition on the Wall Street Journal corpus. Especially when grammatical word sequences are required to be estimated in N-gram models due to sparseness or long-range dependencies. A computational environment with limited hardware was used which allows a realistic evaluation for the use in car head-units or mobile devices. An improvement from 19.3% to 13.3% word error rate was achieved.

5.3 Evaluating recognition on multiple devices

This section describes, a novel technique that recognizes speech on a server but all private knowledge is processed on the client. There is no need anymore to transfer the users address book, calendar or medical data to the server to achieve a satisfying recognition. This work was previously published by Georges et al. [10].

The technique combines the advantage of a powerful server with almost unlimited memory and the advantage of using locally available user dependent knowledge. A dynamic language model is used to recognize speech with the help of content dependent acoustic fillers on a server. The result is then recognized including user dependent knowledge on a client, e.g., a smart phone. A word error rate reduction of 17% on the Wall Street Journal Corpus was achieved.

5.3.1 Introduction

It would be beneficial for various speech enabled applications to use local data such as address book entries, calendar entries or other private data. These private data is often not available on a server. This may be because of legal reasons, e.g., medical patient data. A framework for secure speech recognition was proposed, e.g. by Smaragdis et al. [217]. However, in this thesis a method is proposed which keeps personal data on the client. Using local data can also reduce the required server storage complexity for high demand speech applications. Recognition on an embedded device is often limited due to restricted computational power and memory.

A novel technique was proposed by Georges et al.[10] that combines client and server based speech recognition using dynamic language models and acoustic fillers. There is no need to synchronize user dependent private data to achieve accurate speech recognition. All private data is recognized on the client. It enhances the recognition hypotheses from the server with suitable locally available data. This allows the use of models that are highly optimized for the use on embedded devices on the one hand. On the other hand, the server recognizer can use precise acoustic models and language models which are estimated on crowd sourced data. The novel technique can take advantage of private data that is only locally available on the client.

The novel speech recognition technique uses several language models, simultaneously. Alternatively, Murveit et al.[218] described a technique that uses different levels

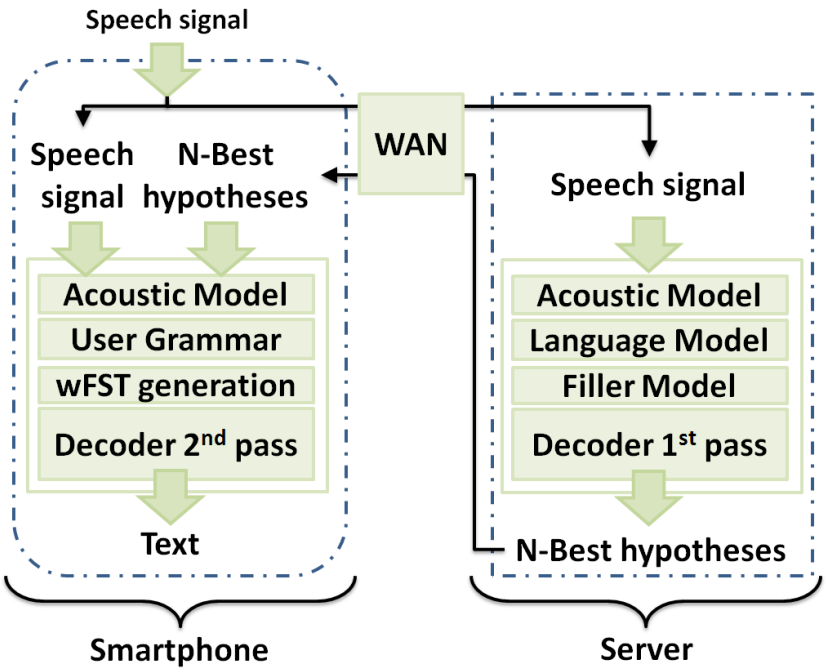


Figure 5.6: *Speech recognition hypotheses from a server are enhanced on the smart phone with user knowledge. The recognizers are connected through a Wide Area Network (WAN).*

of detail between recognition passes. Multiple pass search strategies were described in detail by Schwartz et al.[191]. Here statistical language models and grammars are combined. An overview of combining method was given in Section 3.4.7.

Figure 5.6 gives an overview of the novel technique. The speech signal is captured, without loss of generality, on a smart phone and passed through a wide area network to a server. A generalized language model is used for recognition along with an acoustic model and acoustic fillers. This is described in detail in Section 5.3.3. The recognition hypotheses are passed to the smart phone. The hypotheses were enhanced on the smart phone with user grammars and assembled to a weighted finite-state transducer and Finally recognized as described in Section 5.3.4. The technique was evaluated on the Wall Street Journal Corpus [167] in Section 5.3.5. A word error reduction of 17% has shown that a significant accuracy improvement can be achieved. A delay of 15% compared to real time speech recognition was observed. This delay can be used to provide preliminary recognition results. The client is typically equipped with embedded processors and advanced battery-saving modes. In fact, the novel technique can take advantage of these modes. The full computational power is only required for a short time period to compute the 2^{nd} pass.

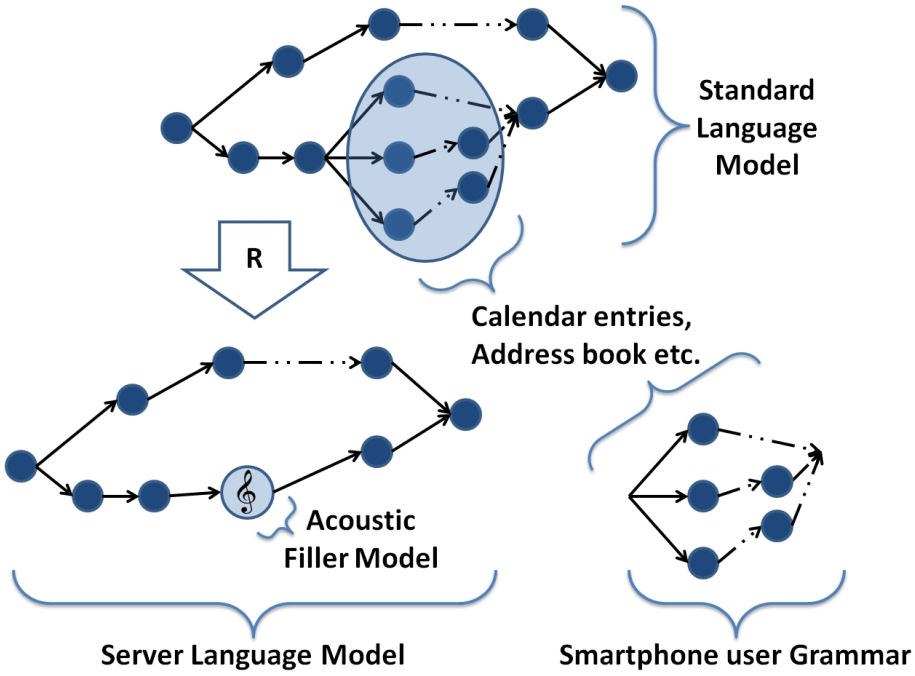


Figure 5.7: *The standard language model is divided into a user dependent grammar and a generalized language model. Both were represented as weighted finite-state transducers.*

5.3.2 Dynamic language models with acoustic fillers

Various language models are dynamically combined on multiple devices. An overview of the language models is given in Figure 5.7. The start point is a corpus $\mathbb{C} \subset W^*$ over vocabulary W for language model training. Further, there are K sets of user-dependent word sequences $S_k \subset W^*$ with $k \leq K$. User dependent word sequences could be terms from a calendar, proper names from an address book or credit card numbers, etc. Let S_K be the set of all user dependent word sequences:

$$S_K = \bigcup_{k}^{K} S_k \subseteq W^*.$$

Each occurrence of a term from S_k in $\mathbb{C}$ is substituted with a marker $t_k \in T$ where $|T| = K$ and $T \cap W = \emptyset$. The result is a generalized corpus $\mathbb{C}'$. The replacing operator R is used as described by Georges et al. [9]. Regular expressions can be used, too. The definition of R is given as:

$$R : W^* \rightarrow ((W \cup T)^* \backslash S_K)^*.$$

A generalized N-gram Markov language model is then estimated on $\mathbb{C}'$. The probability of a word sequence $\underline{w}$ is given by the sequence computed by $R(\underline{w})$. The generalized language model can be formulated as:

$$P(R(\underline{w})) = \prod_{i}^{|R(\underline{w})|} P(R(\underline{w})_i | R(\underline{w})_{i-N+1:i-1}) \cdot \begin{cases} P_k(\underline{w}_{m:n}) & \exists m,n : R(\underline{w}_{m:n}) = R(\underline{w})_i = t_k \\ 1 & else. \end{cases}$$

P_k is a conditional probability for a replaced word sequence by R where the probability is 1 if no word was replaced. The model is normalized if each word sequence in S_K is uniquely associated with one marker. Let $P(\underline{x}|\underline{w})$ be the acoustic model. The most probable word sequence $\underline{\hat{w}}$ is given by a sequence of speech features $\underline{x}$ [66], [65]. Here, the fundamental formula of speech recognition becomes:

$$\underline{\hat{w}} = \underset{\underline{w} \in W*}{\operatorname{argmax}} P(\underline{x}|\underline{w}) P(R(\underline{w})).$$

The server uses the generalized language model where each P_k is replaced on–the–fly with a corresponding acoustic filler as described in Section 5.3.3. The acoustic fillers are based on phoneme loop models estimated on the replaced word sequences. There are various filler alternatives described in the literature. Asadi et al.[195] proposed fillers that were used to obtain phonetic transcriptions for modeling out of vocabulary words. Jiang et al.[197] described fillers based on sub–word features for a vocabulary–independent word confidence measure. Fillers based on word fragments were proposed by Klakow et al [198] and various models were described by Bazzi et al.[199]. Section 5.3.5 investigates the use of oracle fillers for recognizing speech on multiple devices. The server recognition result is used along with user grammars on the client, e.g., a smart phone to assemble a user dependent transducer. This transducer is recognized on the client as described in Section 5.3.4.

5.3.3 Recognition on the server

The speech recognizer on the server uses a generalized language model where user dependent word sequences are recognized with acoustic fillers. Weighted finite-state transducers are used. This enables an on–the–fly nesting technique together with acoustic filler models.

The generalized N-gram Markov language model can be represented by a weighted automaton G_1. The relations between phoneme sequences $\mathbb{P}$ and words W is described by a lexicon transducer $L \subseteq (\mathbb{P} \times W)^*$. Further on, the context dependency between phonemes is given by the transducer C. A static search network can be assembled as follows:

$$M_1' = \min(\det(C \circ L \circ G_1)),$$

where "$\circ$" denotes the composition operator as described in Section 2.4. "min", "det" denote the transducer operator for minimization and determinization. The operators are introduced in Section 2.2 and 2.3, respectively. Finally, M_1' is composed on–the–fly with a hidden Markov model H along with the cross-word computation. This was initially described by Hori et al. [34], [35] and further improved by McDonough et al.[36], Allaucen et al. [37], [38]. The probability for a phoneme sequence, given a sequence of speech features, can be computed using a token passing time synchronous Viterbi beam search. An introduction is given by Mohri et al.[169] and Young et al. [23]. This was described in Section 4.5. Each acoustic filler is represented as a weighted transducer, sharing the same set of hidden states. There is no need to extend the acoustic model. The transducer replacing operator can be used to nest the filler model into the M_1' transducer. The filler is nested each time when a marker from the generalized language model is reached. The technique was introduced in Section 4.7.1.

The recognition result is an N-best list of sentence hypotheses, e.g. proposed by Schwartz et al.[191]. The acoustic filler location is tagged and will be later used to include user dependent knowledge. Using N-best sentence hypotheses ensures backward compatibility for other speech applications using the same server infrastructure. Alternatively, a lattice could be passed to the client.

5.3.4 Recognition on the client

The smart phone receives recognition hypotheses from the server. The user dependent language portion is marked. This could be proper names, dates or other private data. The user data is locally available as grammar, e.g., an address book, calendar or medical recordings. These data is used for speech recognition on the client, e.g., a smart phone.

Here, the received N-best sentence hypotheses from the server is summarized in one grammar G_2, where each sentence ends up in one grammar rule. This is comparable to an output voting error reduction system where different hypotheses from various recognizers are combined to improve the overall accuracy. The Recognizer Output Voting Error Reduction (ROVER) method was proposed, e.g. by Fiscus [219]. Schwenk et al.[220] proposed to include language model weights. This can also be introduced in the method described in this thesis. Alternatively, a word lattice can be delivered by the server. Each marker in G_2 points to a user grammar. Similar to the recognition on the server, a transducer M_2' can be assembled as:

$$M_2' = \min(\det(C \circ L \circ G_2)).$$

The phoneme dependency model is C and L is the lexicon transducer. M_2' is composed on–the–fly with a hidden Markov model H along with the cross-word computation. A token passing time synchronous Viterbi beam search is used similar to the server recognition system. In addition, histogram pruning is applied to fulfill the embedded memory requirement. An introduction in pruning techniques is provided, e.g. by Huang et al. [66].

5.3.5 Evaluation

The proposed system was evaluated using the Wall Street Journal Corpus [167]. The same decoder set-up was used on the server and on the client. This enables a fair comparison. An integer value based acoustic model evaluation was used. Acoustical adaptation techniques such as MLLR etc. were not used in this evaluation The SRI Language Model tool–kit described by Stolcke [216] was used to estimate the 5k word language models. Kneser-Ney discounting was used as described, e.g. by Kneser et al.[155]. The Wall Street Journal Corpus ℂ was prepared for language model training so that it becomes comparable to real world applications.

Table 5.2: *Used grammar for evaluation in Backus–Naur Form*

⟨*DAY*⟩	:=	'Monday' | 'Tuesday' | ...
⟨*MONTH*⟩	:=	'January' | 'February' | ...
⟨*NUM*⟩	::=	⟨*num*⟩ ['.' ⟨*num*⟩][',' ⟨*num*⟩]
⟨*num*⟩	::=	'one' ⟨*num*⟩ | 'two' ⟨*num*⟩ | ... | ⟨*empty*⟩
⟨*MONETARY*⟩	::=	⟨*NUM*⟩'dollar'|⟨*NUM*⟩'cent'|...
⟨*PERCENT*⟩	::=	⟨*NUM*⟩ 'percent'
⟨*ABBREVIATION*⟩	::=	⟨*Letter*⟩ | 'SVOX' | ...
⟨*Letter*⟩	::=	'A.' ⟨*Letter*⟩ | ... | ⟨*empty*⟩

Imagine a short message dictation application where the local available address book, the music title collection and the calendar should be included in the recognition. Here, the user dependent knowledge S is a subset of all weekdays, names of months, various number terms and abbreviations according to Table 5.2. All user dependent knowledge $S' \subset S$ from the corpus that occurs in the set of test sentences was excluded. This reduces the coverage of S' from 31% down to 7%:

$$\mathbb{C}' = \{\underline{w} \in \mathbb{C} | \nexists m,n : w_{m:n} \in S'\}.$$

This coverage seems realistic for real world applications when we analyze N-gram cut-off data. $\mathbb{C}'$ is used to estimate the language models for the server only system. Each term of S is replaced in the corpus $\mathbb{C}'$ with a corresponding marker symbol from T. Te grammar in Table 5.2 was used. The transducer replacement operator R to used to build $\mathbb{C}''$ as follows:

$$\mathbb{C}'' = \{\underline{w} \in ((W \cup T)^* \backslash S)^* | \exists \underline{w}' \in \mathbb{C}' : \underline{w} = R(\underline{w}')\}.$$

The generalized corpus $\mathbb{C}''$ is used to estimate the dynamic language model. This language model is used by the server along with the acoustic fillers. Those fillers are based on phoneme loop models. Every substituted word sequence from $\mathbb{C}''$ is used to estimate the 1-gram phoneme loop filler.

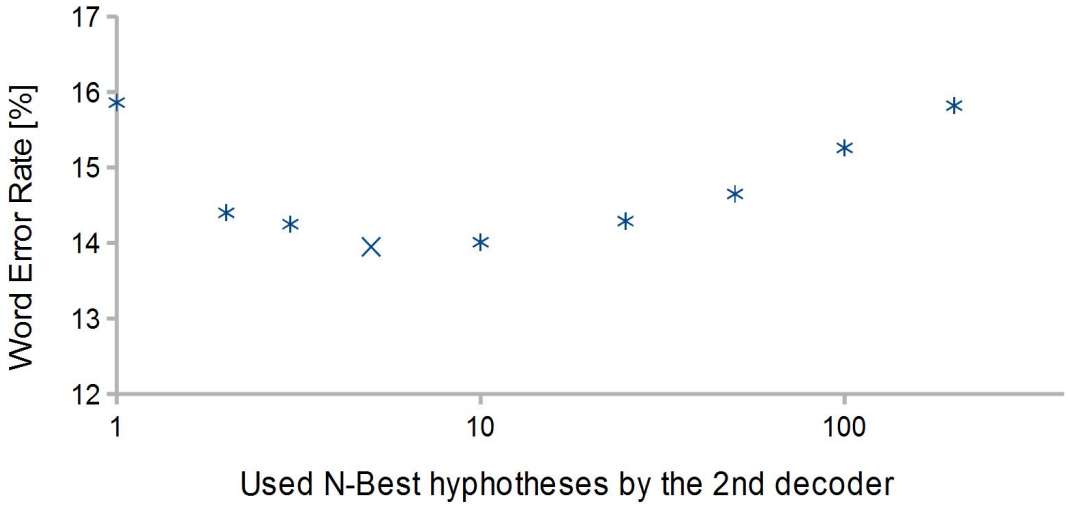


Figure 5.8: *The minimal word error rate was achieved when the 2^{nd} decoder was recognizing on 5-best hypotheses. This point is denoted by a cross.*

Initially, the impact of the number of N-best hypotheses was analyzed. The hypotheses are passed from the server (1^{st} decoder) to the client (2^{nd} decoder). It was expected that this influences the recognition accuracy significantly. This was confirmed. Figure 5.8 shows the influence using 3-gram dynamic language models. The same behavior was also observed for the 2-gram and 4-gram models. A minimal word error rate along with tenable recognition time for the 2^{nd} decoder was achieved using 5-best hypotheses. This experiment is denoted with a cross in Figure 5.8. The pruning behavior on the 2^{nd} decoder has a significant influence on the recognition time but nearly no influence on the accuracy for small N.

In this evaluation, phoneme loop fillers were used. Even when each model was estimated on representative data, the difference between each filler is minimal. The accuracy can be further improved using fillers which are strongly user adapted. The oracle full word filler model was estimated on the test data in the following experiment. Figure 5.9 on the next page illustrates the potential of improvement for dynamic 2-gram language models on representative hardware. The oracle filler outperforms the proposed phoneme loop model as expected. Further, the performance of the grammar 2-gram language model was compared, where user dependent grammars were nested during decoding. This is only possible when the user data is available on the server. The novel technique could achieve nearly the same recognition accuracy with user dependent fillers although it took a certain delay. Similar behavior was observed using the 3 and 4-gram language model set-up.

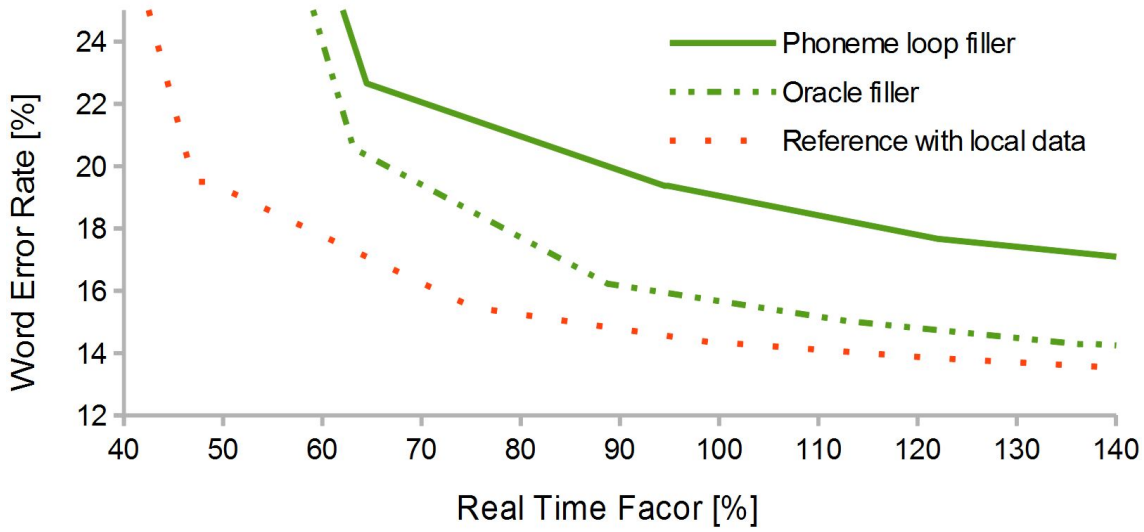


Figure 5.9: *The oracle filler gives an impression of the potential of improvement when user adapted fillers can be used compared to the proposed phoneme loop fillers.*

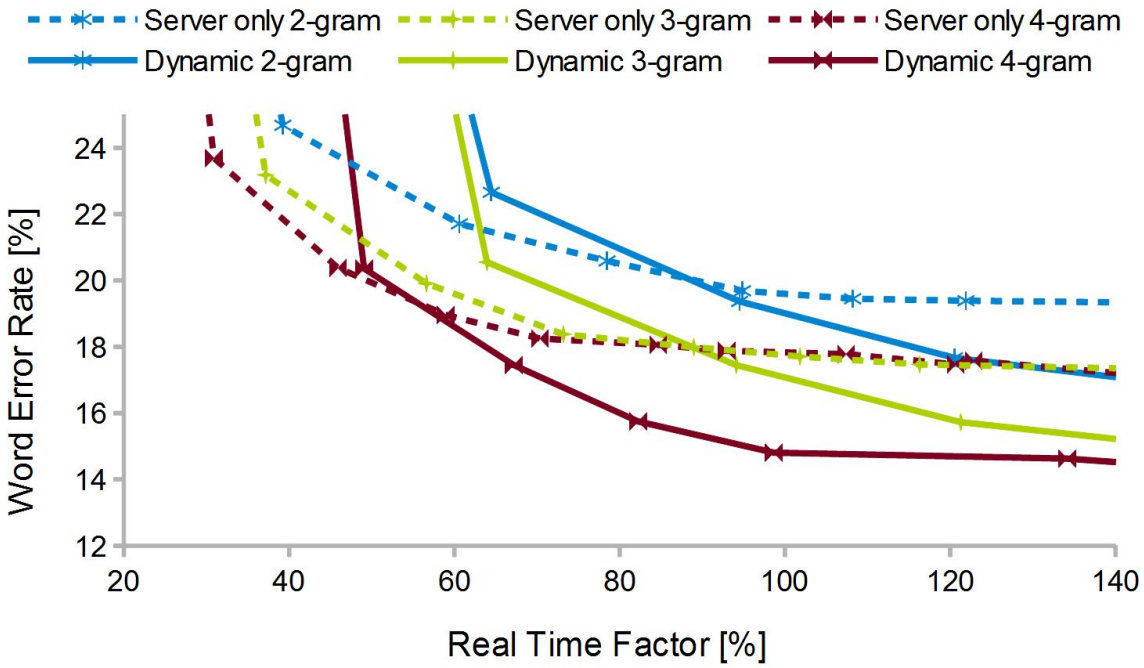


Figure 5.10: *The novel client-server speech recognition technique is beneficial if a short delay is acceptable.*

Finally, the novel system was compared with a server only speech recognition. A faster recognition was achieved with 4-gram language models whereas no further accuracy improvement was observable. In summary, the recognition accuracy of the novel technique outperforms the server only system as summarized in Figure 5.10. No private data has to be synchronized with the server. All private data such as the address book, calendar or medical data remains on the client. The latency of the proposed technique requires a user feedback mechanism for certain applications. Here, the latency was on average 15% of the utterance duration. The processor can stay in a battery-saving mode for 85% of the utterance duration. Note that the begin and end of speech detection is used to measure the utterance duration. A word error rate reduction of 17% was achieved for the 3-gram dynamic language model set-up.

5.3.6 Conclusion

A technique for recognizing speech on a client and server was described. The technique enables to keep private data on the client. Private data could be an address book, a private calendar or some medical patient data. A dynamic language model is used on the server along with acoustic fillers. The recognition result is then combined with user dependent knowledge on the client, e.g., on a smart phone.

It was experimentally shown that the proposed technique improves speech recognition on the Wall Street Journal corpus. An average latency of 15% was observed compared to real time recognition and, in the same time, a word error rate reduction of 17% was achieved.

5.4 Summary

This chapter introduced common evaluation metrics to evaluate language processing techniques. The evaluation of statistical language models is described followed by the measure of the speech recognition quality. This chapter puts all pieces together which were introduced previously. Weighted finite-states are used in the data preparation stage as well as in the recognizer run-time stage. In Chapter 2, transducers were introduced. Automatic speech recognition was described in Chapter 3. In Chapter 4 the use of weighted finite-state transducers for speech recognition are described. It also introduced decoding techniques which are used by the applications described in this chapter. Two applications were selected to be evaluated in the thesis.

First, an application is proposed which allows to embed grammars in statistical language models. This dynamic language model allows to use models from various sources, e.g. the user address book. The used transducer nesting technique enables the computation on embedded devices, such as mobile phones or car-head units as described in Section 5.2. An improved recognition performance was shown by an evaluation. Second, an application is proposed which recognizes speech in the cloud where the personal data is recognized on the client. A dynamic language model was used to distinguish between user data and data needed to estimate the language model. Section 5.3 described the application. It outlines which methods and techniques described in this thesis are used to realize the application.

Future research may increase the precision of acoustic fillers. The substituted word sequence duration can be estimated and used during decoding. The use of natural language understanding methods may also increase the accuracy by estimating more precise dynamic language models. However, the technique still needs to be evaluated on different language and domains. Research may further decrease the latency between passes, e.g. by optimizing the network communication. Also the use of alternative decoders for the 2^{nd} pass may become an important research question. It could use information from the 1^{st} pass to reduce the required computational power on succeeding passes. An phonetic sensitive database look-up could be used as well as pruning methods to speed up the recognition of dynamic content. Future research may also investigate the limits of this technology. This may help optimizing the system by determining the best number of nested transducers given an infrastructure. This could be a cloud infrastructure as well as a network of loosely connected wearables.

6

Thesis summary

This thesis describes methods and techniques for speech recognition for embedded and hybrid use-cases. It introduces novel techniques, such as a location aware speech recognition (Georges et al. [1]) or a query disambiguation method (Geroges et al. [2]). In particular, a recognizer was described which uses a dynamic language model to recognize speech on multiple devices (Georges et al. [6], [7], [8]). The presented speech decoder is based on weighted finite state transducer and is applicable for low resource speech recognition, e.g. on embedded devices.

Two real world applications are selected to evaluate the proposed methods and techniques. Both applications were evaluated on the Wall Street Journal Corpus and published at INTERSPEECH and ICASSP by Georges et al. [9], [10]. First, a transducer based speech recognition with dynamic language models was evaluated. It nests transducers on embedded devices to embed grammars in statistical language models. An improvement from 19% to 13% word error rate was achieved. Second, accurate client-server based speech recognition keeping personal data on the client was evaluated. It recognizes the utterance on a speech server whereas all personal data is recognized on the client. For this, acoustic filler models were used and a word error rate reduction of 17% was achieved. An average latency of 15% was observed compared to real time recognition and a word error rate reduction of 17% was achieved.

Thesis summary and future work

This thesis proposes the use of dynamic language models for automatic speech recognition. The proposed method allows to embed dynamic content into statistical language models. A technique is invented that allows to distribute the recognition process on multiple devices. Two question were considered:

(1) How to represent dynamic data in the speech search space?

(2) How to store and process the search space, e.g. to protect private data?

Dynamic content could be an address book, favorite music titles or medical records. More general, all content is considered that is sparsely represented in common training data. This includes stock prises, abbreviations and words with long range dependencies. Such data can often be represented by grammars that are written by human experts or derived from dedicated data bases. Combining dynamic language models with acoustic fillers lead to a novel technique. The possibility of recognizing speech on multiple devices enables new applications and business use-cases.

Weighted finite-state transducers are used in this thesis. A data structure and compression method was introduced that enables low resource speech recognition, e.g. on embedded devices. Operators on transducers are summarized and their use illustrated for many applications in the area of speech and language processing. The transducer composition operator was demonstrated by an application that compiles recognition grammars from relational databases. This is used to build a voice destination entry system. A robust transducer implementation was proposed that enables open vocabulary processing. This was required to retrieve and replace proper names, abbreviations or stock prices from arbitrary data. An introduction in grammars and statistical n-gram Markov model is provided together with techniques for text post- and preprocessing. Also smoothing, adaptation and interpolation methods on word and word-class based statistical models are introduced. A novel technique is proposed for location aware speech recognition using statistical language models.

Finally a dynamic language model is proposed together with a transducer nesting technique. The dynamic language model embeds grammars in statistical models where the grammar is used to represent dynamic content. An example for such data is the set of command and control phrases, addresses or named entities. These data is often sparsely represented in the training data or not accessible due to data privacy restrictions. The statistical model is used to compute the probability of word sequences

which are phrased in a natural language. The transducer representation of grammars, statistical and dynamic language models was described.

A nesting technique for transducers enables the use of dynamic language models on embedded devices. The nesting and decoding is described together with a novel technique to recognize speech on multiple devices. It uses acoustic fillers to substitute dynamic content to be processed in a postponed recognition pass. The first pass may happen by a cloud speech service where the second pass is processed on a personal device, e.g. a smart phone. Two applications are used in this theses to evaluate the proposed methods and techniques. The evaluation shows a significant improvement over the baseline:

(1) The first application described the use of the transducer nesting technique to decode dynamic language models on embedded devices. It enables the use of locally available data in a language model estimated on crowd sourced data. Local data can be a list of favorite music titles, proper names, abbreviations or number expressions. These phrases are often sparse represented in the training data.

(2) The second application described the recognition of speech on multiple devices. A hybrid set-up was chosen to recognize short massages by a speech-server whereas all personal data is recognized on the client. The personal data is not transfered to the server. In this way, the data on the client is protected on the one hand. On the other hand, it reduces the complexity of the server infrastructure.

Both applications are typical use-cases for speech recognizers of the next generation. The content which is intended to be used for speech recognition is going to be shared over a plurality of devices. The exchange of data will be limited due to network bandwidth, storage capacities or legal reasons. Not only the handling of distributed data, but also the distribution of computational power is getting more and more important. Load balancing within and across speech data centers is an important cost factor. Also new devices such as wearables requires such innovations in speech recognition. These devices often provide only a limited computational power and the network bandwidth is restricted, e.g. to protect the data. This thesis described dynamic language models and a novel technique for distributed speech recognition on multiple devices.

Future work needs to investigate the use of dynamic language models in other language domains such as natural command and control queries, voice search or question answering. Also its use in other languages needs to be investigated. It is also important to investigate the scaling capabilities for the use in an industrial environment. How suited is a dynamic language model for a one, two million word task? The method can also be improved by the use of natural language understanding to identify dynamic content. Also a result preparation between processing stages could further increase the accuracy. One example is the re-use of phonetic information that were recognized in the 1^{st} recognition pass. This information can be used in the 2^{nd} pass to prune the dynamic content in advance. An acoustic database look-up can be used for this as well as re-scoring methods. Further research may also investigate different 2^{nd} pass decoders. It is not required to use a time synchronous search. It could be beneficial to use alternative search techniques, e.g. a condition random field. Future research can also develop precise acoustic filler models. A starting point can be the use of duration models for words and word sequences, but also the use of dedicated pronunciations for special names could further increase the recognition accuracy.

Optimizing the latency will also become an important factor, at least when the recognizer will be distributed over a plurality of devices. Research can investigate new protocols to reduce the network bandwidth between recognition passes. This thesis proposed a vertical search space distribution over multiple devices. One part of the network is nested into a more general one which itself can be nested into an even more general network. All together is a huge search space that is distributed over multiple devices and connected over acoustic fillers. It is an hierarchical approach although applicable on different branches simultaneously. Future research could investigate a horizontal search space distribution where, e.g., the 1^{st} part of the sentence is recognized on one device and the rest of the sentence on a different one, subsequently. Horizontal and vertical search space separation can open a new way of novel load balancing approaches. It could enable a new set of distributed speech recognizers which can be connected in an arbitrary way. The network of speech recognizers can be self-organizing by analyzing available resources. This ambitious recognizer framework may achieve the always overall best recognition accuracy given the available models, network bandwidth and computational power.

References

[1] M. Georges, J. Anastasiadis, and O. Bender, "Motion adaptive speech recognition for enhance voice destination entry," U.S. application PCT/US2015/035 110, 2015.

[2] M. Georges, E. Vellasques, F. Niedtner, O. Bender, J. Anastasiadis, and D. Jung, "Method and apparatus for processing user input," U.S. application PCT/US2015/038 535, 2015.

[3] M. Georges, "A comparative study of features for audio-visual speech recognition," Master's thesis, Saarland University, Saarland, 2010.

[4] F. Faubel, M. Georges, B. Fu, and D. Klakow, "Robust gaussian mixture filter based mouth tracking in a real environment," Visual Computing Research Conference : 8. - 10. Dezember 2009, Saarbrücken, Saarland, 2009.

[5] F. Faubel, M. Georges, K. Kunatami, D. Klakow, and A. Bruhn, "Improving hands-free speech recognition in a car through audio-visual voice activity detection," *HSCMA : 2011 Joint Workshop on Hands-free Speech Communication and Microphone Arrays*, pp. 70–75, 30 May - 1 June 2011, dK ISSN: 978-1-4577-0997-5.

[6] M. Georges and S. Kanthak, "Multiple pass automatic speech recognition methods and apparatus," International application PCT/US2013/056 403, 2013.

[7] M. Georges and S. Kanthak, "Mehrstrangiges automatisches spracherkennungsverfahren und vorrichtung dafür," Europe Patent EP App. EP20,130,861,533, May 20, 2015.

[8] M. Georges and S. Kanthak, "Multiple pass automatic speech recognition methods and apparatus," U.S. Patent US App. 14/364,156, Feb. 26, 2015.

[9] M. Georges, S. Kanthak, and D. Klakow, "Transducer-based speech recognition with dynamic language models," *Proceedings of INTERSPEECH*, pp. 642–646, 2013.

[10] M. Georges, S. Kanthak, and D. Klakow, "Accurate client-server based speech recognition keeping personal data on the client," *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 3271–3275, May 2014.

[11] U. Schöning, "Theoretische informatik kurzgefasst," *Spektrum Akademischer Verlag*, vol. 36, no. 46, p. 118, 2008.

[12] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development*, vol. 3, no. 2, pp. 114–125, 1959.

[13] M. Mohri, F. Pereira, and M. Riley, "Weighted automata in text and speech processing," *IN ECAI-96 WORKSHOP*, pp. 46–50, 1996.

[14] M. Mohri, *Weighted Finite State Transducer Algorithms: An Overview*. Physica-Verlag, 2004.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[16] W. Kuich and A. Salomaa, Eds., *Semirings, Automata, Languages*. London, UK, UK: Springer-Verlag, 1986.

[17] M. Mohri, "Weighted automata algorithms," in *Handbook of Weighted Automata*, ser. Monographs in Theoretical Computer Science. An EATCS Series, M. Droste, W. Kuich, and H. Vogler, Eds. Springer Berlin Heidelberg, 2009, pp. 213–254.

[18] M. Mohri, "On some applications of finite-state automata theory to natural language processing," *Nat. Lang. Eng.*, vol. 2, no. 1, pp. 61–80, Mar. 1996.

[19] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *Signal Processing Magazine, IEEE*, vol. 16, no. 5, pp. 64–83, Sep 1999.

[20] G. V. Noord and D. Gerdemann, "Finite state transducers with predicates and identities," *Grammars*, vol. 4, p. 2001, 2001.

[21] C. Allauzen and M. Mohri, "An efficient pre-determinization algorithm," *CIAA 2003. LNCS*, pp. 83–95, 2003.

[22] S. Kanthak and O. Bender, "Efficient incremental modification of optimized finite-state transducers (fsts) for use in speech applications," Jul. 30 2014, eP Patent App. EP20,110,872,688.

[23] S. J. Young, N. H. Russell, and Thornton, "Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems," Cambridge University Engineering Department, Tech. Rep., 1989.

[24] M. Almeida, N. Moreira, and R. Reis, "On the performance of automata minimization algorithms," DCC - FC LIACC, UNIVERSIDADE DO PORTO, Tech. Rep., 2007.

[25] J. E. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," Stanford University, Stanford, CA, USA, Tech. Rep., 1971.

[26] M. Baclet and C. Pagetti, "Around hopcroft's algorithm." *CIAA*, vol. 4094, pp. 114–125, 2006.

[27] J. A. Brzozowski, "Canonical regular expressions and minimal state graphs for definite events," in *Mathematical theory of Automata*, ser. Volume 12 of MRI Symposia Series. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962, pp. 529–561.

[28] M. Mohri, "Minimization algorithms for sequential transducers," *Theor. Comput. Sci.*, vol. 234, no. 1-2, pp. 177–201, Mar. 2000.

[29] B. W. Watson and J. Daciuk, "An efficient incremental dfa minimization algorithm," *Nat. Lang. Eng.*, vol. 9, no. 1, pp. 49–64, Mar. 2003.

[30] S. Dobrisek, B. Vesnicer, and F. Mihelic, "A sequential minimization algorithm for finite-state pronunciation lexicon models." *Proceedings of INTERSPEECH*, pp. 720–723, 2009.

[31] G. Zweig, G. Saon, and F. Yvon, "Arc minimization in finite state decoding graphs with cross-word acoustic context." *INTERSPEECH*, 2002.

[32] D. Revuz, "Minimisation of acyclic deterministic automata in linear time," *Theor. Comput. Sci.*, vol. 92, no. 1, pp. 181–189, Jan. 1992.

[33] F. C. N. Pereira and M. D. Riley, "Speech recognition by composition of weighted finite automata," *Finite-State Language Processing*, pp. 431–453, 1996.

[34] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," *INTERSPEECH*, 2004.

[35] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for wfst-based speech recognition." *INTERSPEECH*, pp. 557–560, 2005.

[36] J. W. McDonough, E. Stoimenov, and D. Klakow, "An algorithm for fast composition of weighted finite-state transducers." *ASRU*, 2007.

[37] C. Allauzen and M. Mohri, "3-way composition of weighted finite-state transducers," *CIAA*, pp. 262–273, 2008.

[38] C. Allauzen and M. Mohri, "N-way composition of weighted finite-state transducers," *Int. J. Found. Comput. Sci.*, vol. 20, no. 4, pp. 613–627, 2009.

[39] M. Mohri, F. C. N. Pereira, and M. Riley, "The design principles of a weighted finite-state transducer library." *Theor. Comput. Sci.*, vol. 231, no. 1, pp. 17–32, 2000.

[40] M. Mohri and C. S. Yu, "Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers," 2000.

[41] A. Kempe, "Extraction of epsilon-cyclesfrom finite-state transducers," *Revised Papers from the 6th International Conference on Implementation and Application of Automata*, pp. 190–201, 2002.

[42] M. H. Alsuwaiyel, *Algorithms: Design Techniques and Analysis*, ser. Lectures Notes Series on Computing. Singapore: World Scientific, 1999.

[43] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, vol. 5, no. 11, pp. 558–562, Nov. 1962.

[44] R. E. Tarjan, "Edge-disjoint spanning trees and depth-first search," *Acta Informatica*, vol. 6, no. 2, pp. 171–185, 1976.

[45] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan, "Faster algorithms for incremental topological ordering." *ICALP (1)*, vol. 5125, pp. 421–433, 2008.

[46] A. E. Brouwer and W. H. Haemers, *Spectra of Graphs*. New York, NY: Springer, 2012.

[47] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis and Internet Examples*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2009.

[48] C. Allauzen, M. Mohri, and B. Roark, "A general weighted grammar library." *CIAA*, vol. 3317, pp. 23–34, 2004.

[49] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "Openfst: A general and efficient weighted finite-state transducer library." *CIAA*, vol. 4783, pp. 11–23, 2007.

[50] A. A. Aqrawi, S. Anne, and C. Elster, "Effects of compression on data intensive algorithms," 2010.

[51] M. Mohri, "Compact representations by finite-state transducers." *ACL*, pp. 204–209, 1994.

[52] D. K. Blandford, G. E. Blelloch, and I. A. Kash, "Compact representations of separable graphs." *SODA*, pp. 679–688, 2003.

[53] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[54] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *SIGPLAN Not.*, vol. 26, no. 4, pp. 63–74, Apr. 1991.

[55] Park, Penner, and Prasanna, "Optimizing graph algorithms for improved cache performance," *IEEETPDS: IEEE Transactions on Parallel and Distributed Systems*, vol. 15, 2004.

[56] D. Caseiro, "Wfst compression for automatic speech recognition." *INTERSPEECH*, pp. 1493–1496, 2010.

[57] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka, "Compression of weighted graphs." *KDD*, pp. 965–973, 2011.

[58] N. R. Brisaboa, S. Ladra, and G. Navarro, "Directly addressable variable-length codes." *SPIRE*, vol. 5721, pp. 122–130, 2009.

[59] H. E. Williams and J. Zobel, "Compressing integers for fast file access." *Comput. J.*, vol. 42, no. 3, pp. 193–201, 1999.

[60] P. Skibinski, S. Grabowski, and S. Deorowicz, "Revisiting dictionary-based compression." *Softw., Pract. Exper.*, vol. 35, no. 15, pp. 1455–1476, 2005.

[61] E. W. D. Whittaker and B. Raj, "Quantization based language model compression." *INTERSPEECH*, pp. 33–36, 2001.

[62] J. Olsen and D. Oria, "Profile based compression of n-gram language models," *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, p. I, may 2006.

[63] F. Pereira, M. Riley, and R. Sproat, "Weighted rational transductions and their application to human language processing." *HLT*, 1994.

[64] C. A. Furia, "A survey of multi-tape automata," *CoRR*, vol. abs/1205.0178, 2012.

[65] E. G. Schukat-Talamazzini, *Automatische Spracherkennung - Grundlagen, statistische Modelle und effiziente Algorithmen*, ser. Künstliche Intelligenz. Vieweg, 1995.

[66] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall PTR, May 2001.

[67] B. Lee, M. Hasegawa-johnson, C. Goudeseune, S. Kamdar, S. Borys, M. Liu, and T. Huang, "Avicar: Audio-visual speech corpus in a car environment," *in Proc. Conf. Spoken Language, Jeju, Korea*, pp. 2489–2492, 2004.

[68] J.-M. Valin. (2007) The Speex Codec Manual. Xiph.org Foundation. [Online]. Available: http://speex.org/docs/manual/speex-manual/

[69] T. Terriberry and K. Vos, "Definition of the Opus Audio Codec," Internet Requests for Comment, RFC Editor, Fremont, CA, USA, Tech. Rep. 6716, Sep. 2012.

[70] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*. Englewood Cliffs: Prentice Hall, 1978.

[71] D. Enqing, L. Guizhong, Z. Yatong, and C. Yu, "Voice activity detection based on short-time energy and noise spectrum adaptation," *Signal Processing, 2002 6th International Conference on*, vol. 1, pp. 464–467 vol.1, Aug 2002.

[72] P. N. Garner, T. Fukada, and Y. Komori, "A differential spectral voice activity detector," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2004.

[73] M. Moattar and M. Homayounpour, "A simple but efficient real-time voice activity detection algorithm," *EUSIPCO. EURASIP*, pp. 2549–2553, 2009.

[74] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, vol. 28, no. 4, pp. 357–366, 1980.

[75] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugen.*, vol. 7, pp. 179–188, 1936.

[76] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*, 2nd ed. Wiley-Interscience, November 2000.

[77] S. F. BOLL, "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Trans. Acoust, Speech & Signal Process*, vol. 33, no. 27, pp. 113–120, 1979.

[78] P. Lockwood and J. Boudy, "Experiments with a nonlinear spectral subtractor (nss), hidden markov models and the projection, for robust speech recognition in cars," *Speech Commun.*, vol. 11, no. 2-3, pp. 215–228, 1992.

[79] M. Wölfel and J. McDonough, *Distant Speech Recognition*, 1st ed. Chichester, UK: Wiley, 2009.

[80] B. S. Atal, "Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification," *The Journal of the Acoustical Society of America*, vol. 55, no. 6, pp. 1304–1312, 1974.

[81] E. Brill, "A simple rule-based part of speech tagger," in *Proceedings of the 3rd Conference on Applied Natural Language Processing*, 1992.

[82] N. Chomsky, *Syntactic Structures*. The Hague: Mouton, 1957.

[83] N. Chomsky, *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press, 1965.

[84] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translating and Compiling. Vol 1 : Parsing*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

[85] D. Jurafsky, C. Wooters, J. Segal, A. Stolcke, E. Fosler, G. Tajchman, , N. Morgan, and N. Morgan, "Using a stochastic context-free grammar as a language model for speech recognition," 1995.

[86] F. Duckhorn and R. Hoffmann, "Using context-free grammars for embedded speech recognition with weighted finite-state transducers." *INTERSPEECH*, 2012.

[87] J.-C. Junqua, *Robust Speech Recognition in Embedded System and PC Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[88] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987.

[89] L. Firoiu, T. Oates, and P. R. Cohen, "Learning regular languages from positive evidence," *In Twentieth Annual Meeting of the Cognitive Science Society*, pp. 350–355, 1998.

[90] F. Denis, A. Lemay, and A. Terlutte, "Learning regular languages using non deterministic finite automata," 2000.

[91] F. Denis, A. Lemay, A. Terlutte, and R. F. J. Curie, "Learning regular languages using rfsas," 2001.

[92] D. Klein and C. D. Manning, "Natural language grammar induction using a constituent-context model." *NIPS*, pp. 35–42, 2001.

[93] H. Feili and G. Ghassem-Sani, "Unsupervised grammar induction using history based approach." *Computer Speech & Language*, vol. 20, no. 4, pp. 644–658, 2006.

[94] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish, "Regular expression learning for information extraction," *In EMNLP*, pp. 21–30, 2008.

[95] A. Roberts and E. Atwell, "Unsupervised grammar inference systems for natural language," School of Computing, University of Leeds, Tech. Rep. 2002.20, 2002.

[96] A. Clark and S. Lappin, "Unsupervised learning and grammar induction," in *The Handbook of Computational Linguistics and Natural Language Processing*, A. Clark, C. Fox, and S. Lappin, Eds. Wiley-Blackwell, 2010, pp. 197–220.

[97] A. A. Markov, "An example of statistical investigation in the text of 'Eugene Onyegin' illustrating coupling of 'tests' in chains," *Proceedings of the Academy of Sciences*, vol. 7 of VI, pp. 153–162, 1913.

[98] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana and Chicago: University of Illinois Press, 1949.

[99] T. Segaran and J. Hammerbacher, *Beautiful Data: The Stories Behind Elegant Data Solutions*. O'Reilly Media, Jul. 2009.

[100] F. Jelinek, *Statistical Methods for Speech Recognition*. The MIT Press, Jan. 1998.

[101] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here," *Proceedings of the IEEE*, p. 2000, 2000.

[102] A. Franz and T. Brants. (2006, Aug.) All our n-gram are belong to you. Google Machine Translation Team. [Online]. Available: http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

[103] R. Kuhn and R. D. Mori, "A cache-based natural language model for speech recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 6, pp. 570–583, 1990.

[104] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks, "A closer look at skip-gram modelling," *Proceedings of the Fifth international Conference on Language Resources and Evaluation (LREC-2006)*, 2006.

[105] M. Mahajan, D. Beeferman, and X. Huang, "Improved topic-dependent language modeling using information retrieval techniques," *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, vol. 1, pp. 541–544 vol.1, Mar 1999.

[106] T. Kalt, "A new probabilistic model of text classification and retrieval title2:," University of Massachusetts, Amherst, MA, USA, Tech. Rep., 1998.

[107] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 275–281, 1998.

[108] X. Liu and W. B. Croft, "Statistical language modeling for information retrieval," *Annual Review of Information Science and Technology*, vol. 39, no. 1, pp. 1–31, 2005.

[109] G. Salton and M. J. McGill, *Introduction to modern information retrieval*. New York: McGraw-Hill, 1983.

[110] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition) (ACM Press Books)*, 2nd ed. Addison-Wesley Professional, Feb. 2011.

[111] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[112] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pp. 161–175, 1994.

[113] T. Vatanen, J. J. VÃČÂd'yrynen, and S. Virpioja, "Language identification of short text segments with n-gram models." *LREC*, 2010.

[114] G. R. Botha and E. Barnard, "Factors that affect the accuracy of text-based language identification." *Computer & Language*, vol. 26, no. 5, pp. 307–320, 2012.

[115] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning, "Named entity recognition with character-level models," *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, pp. 180–183, 2003.

[116] B. Kessler, G. Numberg, and H. Schütze, "Automatic detection of text genre," *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 32–38, 1997.

[117] Y. Su, "Bayesian class-based language models." *ICASSP*, pp. 5564–5567, 2011.

[118] E. Whittaker and R. Woodland, "Efficient class-based language modelling for very large vocabularies," *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, vol. 1, pp. 545 –548 vol.1, 2001.

[119] L. R. Bahl, P. Brown, P. De Souza, and R. Mercer, "A tree-based statistical language model for natural language speech recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 7, pp. 1001–1008, Jul 1989.

[120] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," Jun. 07 1992.

[121] L. Moisa and E. P. Giachin, "Automatic clustering of words for probabilistic language models," *EUROSPEECH*, 1995.

[122] M. Jardino, "Multilingual stochastic n-gram class language models," *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 161–163 vol. 1, May 1996.

[123] C. Beaujard, M. Jardino, and H. Bonneau-maynard, "Evaluation of a class-based language model in a speech recognizer," Jan. 18 1999.

[124] C. Samuelsson and W. Reichl, "A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics," *IEEE ICASSP-99*, pp. 537–540, 1999.

[125] A. Emami and S. F. Chen, "Multi-class model m." *ICASSP*, pp. 5516–5519, 2011.

[126] I. Zitouni, O. Siohan, and C.-H. Lee, "Hierarchical class n-gram language models: towards better estimation of unseen events in speech recognition," *INTERSPEECH'03*, pp. –1–1, 2003.

[127] H. Yamamoto and S. Isogai, "Multi-class composite n-gram language model for spoken language processing using multiple word clusters," *39 th Annual meetings of the Association for Computational Linguistics*, pp. 6–11, 2001.

[128] S. M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," in *IEEE Transactions on Acoustics, Speech and Singal processing*, vol. ASSP-35, no. 3, March 1987, pp. 400–401.

[129] S. C. Martin, C. Hamacher, J. Liermann, F. Wessel, and H. Ney, "Assessment of smoothing methods and complex stochastic language modeling," in *Sixth European Conference on Speech Communication and Technology, EUROSPEECH 1999, Budapest, Hungary, September 5-9, 1999*, 1999.

[130] J. T. Goodman, "Putting it all together: Language model combination," *In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1647–1650, 2000.

[131] B. Bigi, Y. Huang, and R. de Mori, "Vocabulary and language model adaptation using information retrieval." *INTERSPEECH*, 2004.

[132] F. Jelinek, "Readings in speech recognition," in *Readings in Speech Recognition*, A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Self-organized Language Modeling for Speech Recognition, pp. 450–506.

[133] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, pp. 310–318, 1996.

[134] P. Dupont and F. T. Cnet/laa/tss/rcp, "Interpolated word and class bigram models for spanish conversational speech recognition," Oct. 14 1997.

[135] D. Klakow, "Log-linear interpolation of language models." *ICSLP*, 1998.

[136] G. Maltese, P. Bravetti, H. CrÃČÂl'py, B. J. Grainger, M. Herzog, and F. Palou, "Combining word- and class-based language models: a comparative study in several languages using automatic and manual word-clustering techniques." *INTERSPEECH*, pp. 21–24, 2001.

[137] J. R. Bellegarda, "Statistical language model adaptation: review and perspectives," *Speech Communication*, vol. 42, pp. 93–108, 2004.

[138] J. Bilmes and H. Lin, "Online adaptive learning for speech recognition decoding." *INTERSPEECH*, pp. 1958–1961, 2010.

[139] C. Allauzen and M. Riley, "Bayesian language model interpolation for mobile speech input." *INTERSPEECH*, pp. 1429–1432, 2011.

[140] J. Wu, "Adaptation of language models and context free grammar in speech recognition," US Patent US7 925 505 B2, 2007.

[141] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[142] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: The MIT Press, 1999.

[143] J. Goodman, "A Bit of Progress in Language Modeling," Microsoft Research, 56 Fuchun Peng, Tech. Rep., 2000.

[144] H. Jeffreys, *Theory of probability*. Clarendon Press, Oxford, second edition, 1948.

[145] N. Chopin, C. P. Robert, and J. Rousseau, "Harold jeffreys' theory of probability revisited," Paris Dauphine University, Economics Papers from University Paris Dauphine, 2009.

[146] F. Jelinek and R. L. Mercer, "Interpolated estimation of markov source parameters from sparse data," *In Proceedings of the Workshop on Pattern Recognition in Practice*, pp. 381–397, May 1980.

[147] G. J. Lidstone, "Note on the general case of the bayes-laplace formula for inductive or a posteriori probabilities," *Transactions of the Faculty of Actuaries*, vol. 8, pp. 182–192, 1920.

[148] G. E. P. Box and G. C. Tiao, *Bayesian Inference in Statistical Analysis*, 1st ed. Wiley-Interscience, Apr. 1992.

[149] W. Johnson, "Probability: deductive and inductive problems," *Mind*, vol. 41, pp. 421–423, 1932.

[150] I. Witten and T. Bell, "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression," *IEEE Transactions on Information Theory*, vol. 37, no. 4, 1991.

[151] K. W. Church and W. A. Gale, "A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams," *Computer speech and language*, vol. 5, pp. 19–54, 1991.

[152] H. Ney, U. Essen, and R. Kneser, "On Structuring Probabilistic Dependencies in Stochastic Language Modelling," *Computer Speech and Language*, vol. 8, pp. 1–38, 1994.

[153] H. Ney, U. Essen, and R. Kneser, "On the estimation of 'small' probabilities by leaving-one-out," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 12, pp. 1202–1212, Dec. 1995.

[154] R. Rosenfeld and X. Huang, "Improvements in stochastic language modeling," *Proceedings of the Workshop on Speech and Natural Language*, pp. 107–111, 1992.

[155] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," *Processing of ICASSP*, 1995.

[156] T. Kaufmann, "A rule-based language model for speech recognition," 2009.

[157] J. Gillett and W. Ward, "A language model combining trigrams and stochastic context-free grammars." *ICSLP*, 1998.

[158] A. Nasr, Y. Esteve, F. Bechet, T. Spriet, and R. D. Mori, "A language model combining n-grams and stochastic finite automata," *Proceedings of Eurospeech*, pp. 2175–2178, 1999.

[159] Y.-Y. Wang, M. Mahajan, and X. Huang, "A unified context-free grammar and n-gram model for spoken language processing," *Proceedings ICASSP*, 2000.

[160] S. C. Martin, A. Kellner, and T. Portele, "Interpolation of stochastic grammar and word bigram models in natural language understanding." *INTERSPEECH*, pp. 234–237, 2000.

[161] Y. yi Wang, A. Acero, C. Chelba, B. Frey, and L. Wong, "Combination of statistical and rule-based approaches for spoken language understanding," *Proc. ICSLP 2002*, pp. 609–612, 2002.

[162] V. Goel, "Conditional maximum likelihood estimation for improving annotation performance of n-gram models incorporating stochastic finite state grammars." *INTERSPEECH*, 2004.

[163] M. Mohri, "Local grammar algorithms," in *Inquiries into Words, Constraints, and Contexts.*, A. Arppe, L. Carlson, K. Lindèn, J. Piitulainen, M. Suominen, M. Vainio, H. Westerlund, and A. Yli-Jyrä, Eds. CSLI Publications, 2005.

[164] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Comput. Linguist.*, vol. 16, no. 1, pp. 22–29, Mar. 1990.

[165] D. Klakow, "Language-model optimization by mapping of corpora," *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 2, pp. 701 –704 vol.2, may 1998.

[166] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," *From Form to Meaning: Processing Texts Automatically, Proceedings of the Biennial GSCL Conference 2009*, vol. Normalized, pp. 31–40, 2009.

[167] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus." *ICSLP*, 1992.

[168] M. Schuster, "Speech recognition for mobile devices at google," *Proceedings of the 11th Pacific Rim international conference on Trends in artificial intelligence*, pp. 8–10, 2010.

[169] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition." *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[170] C. Chelba, "A structured language model," *Computer Speech and Language*, vol. 14, pp. 283–332, 1997.

[171] A. Rastrow, M. Dredze, and S. Khudanpur, "Efficient structured language modeling for speech recognition," *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pp. 1660–1663, 2012.

[172] Y.-Y. Wang and A. Acero, "Combination of cfg and n-gram modeling in semantic grammar learning." *INTERSPEECH*, 2003.

[173] D. Yu, K. Wang, M. Mahajan, P. Mau, and A. Acero, "Improved name recognition with user modeling."

[174] N. Slonim and N. Tishby, "The power of word clusters for text classification," *In 23rd European Colloquium on Information Retrieval Research*, 2001.

[175] S. C. Martin, J. Liermann, and H. Ney, "Algorithms for bigram and trigram word clustering." *EUROSPEECH*, 1995.

[176] D. Lin, "Automatic retrieval and clustering of similar words," *Proceedings of the 17th international conference on Computational linguistics - Volume 2*, pp. 768–774, 1998.

[177] J. Wiebe, J. Maples, L. Duan, and R. Bruce, "Experience in wordnet sense tagging in the wall street journal," 1997.

[178] C. Kit, X. Liu, and J. J. Webster, "Abbreviation recognition with maxent model." *CICLing*, vol. 3878, pp. 117–120, 2006.

[179] E. Roche and Y. Schabes, "Deterministic part-of-speech tagging with finite-state transducers." *Computational Linguistics*, vol. 21, no. 2, pp. 227–253, 1995.

[180] G. Chrupala and D. Klakow, "A named entity labeler for german: Exploiting wikipedia and distributional clusters." *LREC*, 2010.

[181] E. Loper and S. Bird, "Nltk: The natural language toolkit," *CoRR*, vol. cs.CL/0205028, 2002.

[182] L. Karttunen, "The replace operator," 1994.

[183] P. F. Brown, "The acoustic-modeling problem in automatic speech recognition," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1987, aAI8727170.

[184] Z. Ou and J. Xiao, "A study of large vocabulary speech recognition decoding using finite-state graphs," *Chinese Spoken Language Processing (ISCSLP), 2010 7th International Symposium on*, pp. 123 –128, 29 2010-dec. 3 2010.

[185] D. G. Sturtevant, "A stack decoder for continous speech recognition," *Proceedings of the workshop on Speech and Natural Language*, pp. 193–198, 1989.

[186] J. J. Odell, V. Valtchev, P. C. Woodland, and S. J. Young, "A one pass decoder design for large vocabulary recognition." *HLT*, 1994.

[187] X. L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition." *Computer Speech & Language*, vol. 16, no. 1, pp. 89–114, 2002.

[188] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast lvcsr decoder." *INTERSPEECH*, pp. 549–552, 2005.

[189] D. Caseiro and I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition." *IEEE Transactions on Audio, Speech & Language Processing*, vol. 14, no. 4, pp. 1281–1291, 2006.

[190] S. Kanthak and H. Ney, "FSA: An efficient and flexible C++ toolkit for finite state automata using on-demand computation," *ACL*, pp. 510–517, 2004.

[191] R. Schwartz, L. Nguyen, and J. Makhoul, "Multiple-pass search strategies," in *Automatic Speech and Speaker Recognition*, ser. The Kluwer International Series in Engineering and Computer Science, C.-H. Lee, F. Soong, and K. Paliwal, Eds. Springer US, 1996, vol. 355, pp. 429–456.

[192] A. Alexandrescu, *Modern C++ Design Generic Programming and Design Patterns Applied*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[193] J. Schalkwyk, I. L. Hetherington, and E. Story, "Speech recognition with dynamic grammars using finite-state transducers," *Processing of INTERSPEECH*, 2003.

[194] P. R. Dixon, C. Hori, and H. Kashioka, "A specialized wfst approach for class models and dynamic vocabulary." *INTERSPEECH*, 2012.

[195] A. Asadi, R. Schwartz, and J. Makhoul, "Automatic modeling for adding new words to a large-vocabulary continuous speech recognition system," *Processing of ICASSP*, 1991.

[196] T. Kemp, A. Jusek, I. Systems, L. Ag, and A. Informatik, "Modelling unknown words in spontaneous speech," *In Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 530–533, 1996.

[197] L. Jiang and X. Huang, "Vocabulary-independent word confidence measure using subword features." *ICSLP*, 1998.

[198] D. Klakow, G. Rose, and X. L. Aubert, "Oov-detection in large vocabulary system using automatically defined word-fragments as fillers." *EUROSPEECH*, 1999.

[199] I. Bazzi, J. Glass, and A. C. Smith, "Modeling out–of–vocabulary words for robust speech recognition," 2000.

[200] H. Kokubo, H. Yamamoto, Y. Ogawa, Y. Sagisaka, and G. Kikui, "Out-of-vocabulary word recognition with a hierarchical doubly markov language model," *Automatic Speech Recognition and Understanding, 2003. ASRU '03. 2003 IEEE Workshop on*, pp. 543 – 547, nov.-3 dec. 2003.

[201] D. Yu, Y. C. Ju, Y.-Y. Wang, and A. Acero, "N-gram based filler model for robust grammar authoring," *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, p. I, may 2006.

[202] M. Bisani and H. Ney, "Open vocabulary speech recognition with flat hybrid models." *INTERSPEECH*, pp. 725–728, 2005.

[203] L. Qin, M. Sun, and A. I. Rudnicky, "Oov detection and recovery using hybrid models with different fragments." *INTERSPEECH*, pp. 1913–1916, 2011.

[204] M. Savic, M. Moore, and C. Scoville, "Statistical speech reconstruction at the phoneme level," *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, vol. 1, pp. 657 –660 vol.1, 2001.

[205] N. Jennequin and J.-L. Gauvain, "Modeling duration via lattice rescoring," *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–641 –IV–644, april 2007.

[206] D. Seppi, D. Falavigna, G. Stemmer, and R. Gretter, "Word duration modeling for word graph rescoring in lvcsr." *INTERSPEECH*, pp. 1805–1808, 2007.

[207] J. T. Kao, G. Zweig, and P. Nguyen, "Discriminative duration modeling for speech recognition with segmental conditional random fields." *ICASSP*, pp. 4476–4479, 2011.

[208] F. Bimbot, M. El-BÃČÂĺze, S. Igounet, M. Jardino, K. SmaÃČÂŕli, and I. Zitouni, "An alternative scheme for perplexity estimation and its assessment for the evaluation of language models." *Computer Speech & Language*, vol. 15, no. 1, pp. 1–13, 2001.

[209] S. Chen, D. Beeferman, and R. Rosenfeld, "Evaluation metrics for language models," 1998.

[210] M. Mohri, "Edit-distance of weighted automata: General definitions and algorithms," *Int. J. Found. Comput. Sci.*, vol. 14, no. 6, pp. 957–982, 2003.

[211] A. C. Morris, V. Maier, and P. Green, "From wer and ril to mer and wil: improved evaluation measures for connected speech recognition." *INTERSPEECH*, 2004.

[212] T. Mishra, A. Ljolje, and M. Gilbert, "Predicting human perceived accuracy of asr systems." *INTERSPEECH*, pp. 1945–1948, 2011.

[213] B. Favre, K. Cheung, S. Kazemian, A. Lee, Y. Liu, C. Munteanu, A. Nenkova, D. Ochei, G. Penn, S. Tratz, C. R. Voss, and F. Zeller, "Automatic human utility evaluation of asr systems: does wer really predict performance?" *INTERSPEECH*, pp. 3463–3467, 2013.

[214] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochim. Biophys. Acta*, vol. 405, pp. 442–451, 1975.

[215] G. Jurman, S. Riccadonna, and C. Furlanello, "A comparison of mcc and cen error measures in multi-class prediction," *PLoS ONE*, vol. 7, no. 8, p. e41882, 08 2012.

[216] A. Stolcke, "Srilm – an extensible language modeling toolkit," Jun. 06 2002.

[217] P. Smaragdis and M. V. S. Shashanka, "A framework for secure speech recognition." *IEEE Transactions on Audio, Speech & Language Processing*, vol. 15, no. 4, pp. 1404–1413, 2007.

[218] H. Murveit, J. Butzberger, V. Digalakis, and M. Weintraub, "Large-vocabulary dictation using SRI's DECIPHER speech recognition system: Progressive search techniques," *Proceedings of ICASSP*, 1993.

[219] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover)," *Proceedings ASRU*, 1997.

[220] H. Schwenk and J.-L. Gauvain, "Combining multiple speech recognizers using voting and language model information." *INTERSPEECH*, pp. 915–918, 2000.