BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

OUT-OF-VOCABULARY WORDS DETECTION AND RECOVERY

DETEKCE A OBNOVA SLOV MIMO SLOVNÍK

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **EKATERINA EGOROVA**
AUTOR PRÁCE

SUPERVISOR **JAN ČERNOCKÝ**
ŠKOLITEL

BRNO 2022

Abstract

The thesis explores the field of out-of-vocabulary word (OOV) processing within the task of automatic speech recognition (ASR). It defines the two separate OOV processing tasks – that of detection and recovery – and proposes success metrics for both the tasks. Different approaches to OOV detection and recovery are presented within the frameworks of hybrid and end-to-end (E2E) ASR. These approaches and compared on an open access LibriSpeech database to facilitate replicability.

Hybrid approach uses modified decoding graph with phoneme substrings and utilizes full lattice representations for detection and recovery of recurrent OOVs. Recovered OOVs are added to the dictionary and the language model (LM) to improve ASR system performance.

The second approach employs inner representations of a word-predicting Listen Attend and Spell architecture (LAS) E2E system to perform OOV detection task. Detection recall and precision rates improved drastically in comparison with the hybrid approach. Recurrent OOV recovery is performed on a separate character-predicting system with the use of detected time frames and probabilistic clustering.

Finally, we propose a new speller architecture with a capability of learning OOV representations together with the word predicting network (WPN) training. The speller forces word embeddings to be spelling-aware during the training and thus not only provides OOV recovery, but also improves the WPN performance.

Keywords

out-of-vocabulary words, automatic speech recognition, hybrid ASR, end-to-end ASR, neural architectures, Listen Attend and Spell.

Reference

EGOROVA, Ekaterina. *Out-of-Vocabulary Words Detection and Recovery.* Brno, 2022. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Jan Černocký

Abstrakt

Tato disertační práce zkoumá oblast zpracování slov mimo slovník (out-of-vocabulary word, OOV) v rámci úlohy automatického rozpoznávání řeči (automatic speech recognition, ASR). Definuje dvě samostatné úlohy zpracování OOV – detekci a obnovu – a pro obě úlohy navrhuje metriky úspěšnosti. Prezentuje několik přístupů k detekci a obnově OOV v rámci hybridních a end-to-end (E2E) ASR systémů. Experimentální práce a srovnání přístupů bylo provedeno na otevřené databázi LibriSpeech, aby byla zajištěna reprodukovatelnost experimentů.

Hybridní přístup využívá upravený dekódovací graf s fonémovými podřetězci a pro detekci a obnovu opakujících se OOV využívá reprezentaci založenou na plných rozpoznávacích grafech (lattices). Obnovená OOV jsou přidána do slovníku a jazykového modelu (LM), což vede ke zlepšení úspěšnosti ASR systému.

Druhý přístup využívá k řešení úlohy detekce OOV vnitřní reprezentace systému E2E architektury "Listen Attend and Spell" (LAS) s predikcí slov. Tato metoda oproti hybridnímu přístupu výrazně zlepšuje míru úplnosti a přesnosti (recall a precision). Obnova opakujících se OOV se provádí pomocí samostatného systému predikce znaků s využitím detekovaných časových rámců a pravděpodobnostního shlukování.

Nakonec navrhujeme novou "speller" architekturu se schopností učit se reprezentace OOV společně s trénováním sítě pro predikci slov (word predicting network, WPN). Komponent "speller" ovlivňuje během trénování slovní embeddingy tak, aby dobře reprezentovaly i fonetickou podobu slov, a tím zajišťuje nejen možnost kvalitní obnovy OOV, ale i zlepšení výkonu sítě pro predikci slov.

Klíčová slova

Slova mimo slovník, automatické rozpoznávání řeči, hybridní ASR, E2E ASR, neurální architektury, Listen Attend and Spell.

Citace

EGOROVA, Ekaterina. *Detekce a obnova slov mimo slovník.* Brno, 2022. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel Jan Černocký

Out-of-Vocabulary Words Detection and Recovery

Declaration

I hereby declare that this doctoral thesis was prepared as an original work by me under the supervision of Jan Černocký and the scientific advising of Lukáš Burget. Code for baseline E2E systems has been adapted from Harikrishna Vydana. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Ekaterina Egorova
August 31, 2022

Acknowledgements

I would like to thank my supervisor Jan Černocký not only for excellent scientific guidance and group management, but also for unwavering administrative support in all the life difficulties that plague the life of an immigrant.

Special thanks also go to my scientific advisor Lukáš Burget for ambitious ideas, inspiring conversations, and coffee.

Part of the experiments would not happen without fruitful cooperation with my then-colleague Harikrishna Vydana, and I am very thankful for being able to share his code and expertise.

In my work I drew on the knowledge of many of my colleagues in bordering areas. Although it is not an exhaustive list at all, I would like to thank Martin Karafiat, Karel Veselý, Mirko Hannemann and Igor Szöke for useful talks and help. Although not directly affecting my work, I would like to thank all my colleagues in the Speech group for friendly atmosphere and inspiring coffee talks.

I would not have been introduced to speech recognition and Honza's research group if not for the wonderful professors at the phonetics department of Saint-Petersburg State University, and I sincerely thank them for directing me towards this opportunity. Special thanks go to my Bachelor's thesis advisor there Daniil Kocharov.

From the non-academic world, I would like to first thank my parents for giving me every opportunity to excel and some great genes. The biggest support in my life and work comes from my two close friends, Olga and Irina, who are always there, if mostly virtually. I would like to dedicate this thesis to the loving memory of Dima, whose admiration of me was always bigger than my belief in myself.

Contents

Abbreviations

AM	acoustic model
ARI	adjusted Rand index
ASR	automatic speech recognition
BDiHMM	block diagonal infinite hidden Markov model
biLSTM	bi-directional long short-term memory
BPE	byte-pair encoding unit
CE	cross-entropy
CER	character error rate
CMVN	cepstral mean and variance normalization
CRP	Chinese restaurant process
CTC	connectionist temporal classification
DNN	deep neural network
E2E	end-to-end
fMLLR	feature space maximum likelihood linear regression
G2P	grapheme-to-phoneme
GMM	Gaussian mixture model
HMM	hidden Markov model
IV	in-vocabulary word
KWS	key word spotting
LAS	Listen Attend and Spell architecture
LDA	linear discriminant analysis
LID	language identification
LM	language model
LR	learning rate
LSTM	long short-term memory (recurrent neural network)
MFCC	Mel-frequency cepstral coefficients
MLLT	maximum likelihood linear transform
MT	machine translation
NLP	natural language processing
NN	neural network
OOV	out-of-vocabulary word
OOVC	OOV cost
P2G	phoneme-to-grapheme
PDF	probability density function
PIP	phoneme insertion penalty
PLMSF	phoneme LM scaling factor
PLP	perceptual linear predictive

RI	Rand index
RNN	recurrent neural network
SAT	speaker adaptive training
SID	speaker identification
SOTA	state of the art
STC	(global) semi-tied covariance
TFT	timed factor transducer
WER	word error rate
WFSA	weighted finite-state acceptor
WFST	weighted finite-state transducer
WPN	word predicting network

Chapter 1

Introduction

Speech data mining is an area of computer science that deals with extracting information from human speech by means of computation. Depending on the application, different information is obtained from the speech signal. Some of the most essential tasks are language identification (LID), speaker identification (SID), automatic speech recognition (ASR) and key word spotting (KWS).

Practically, speech processing tasks represent speech production as a model and estimate its parameters by means of machine learning. In the framework of supervised learning, the system must get a sufficient amount of training data: speech with relevant output labels (in case of ASR, transcriptions). In a perfect scenario, the training data would be from the same domain (language, channel, type of speech) as the test data, otherwise the system may encounter problems and not function as well as expected. Often, additional external data is used to improve the systems, for example, dictionaries, language models (LMs), pre-trained models, etc.

After decades of massive research in the field of speech processing, neither of the tasks that the field encompasses can be deemed fully solved. New challenges emerge due to immense diversity of languages, speaking conditions and human factors. One of the biggest yet unsolved issues in speech data mining springs from the fact that a language's vocabulary is virtually unlimited, with new words constantly emerging in response to the ever-changing world. These words pose big problem in speech data mining applications, as it is practically impossible to keep up with them, providing new dictionaries and language models in response to their emergence.

Words which are unseen in the training data are called out-of-vocabulary words (OOVs), and they constitute one of the biggest challenges in ASR and other speech processing tasks. The OOV problem gets even more interesting if one considers that OOVs are usually topic-specific words or proper names, meaning they are often key words important for proper understanding of the message.

In the classical component-based hybrid ASR system with neural network (NN) / hidden Markov model (HMM) - based architecture, OOVs are not represented in the dictionary or the LM and therefore cannot be correctly recognized. Instead, the system will try to find the (acoustically) closest in-vocabulary word (IV), often confusing the end user and interfering with the proper decoding of the words around it due to LM dependencies. In such a system the aim of OOV detection and recovery is to augment the dictionary and the LM with newly discovered words and thus enable their correct recognition.

The more modern end-to-end (E2E) approaches avoid the limitations of dictionary by predicting character or subword labels. In spite of the temptation to declare the OOV issue solved because of the advance of subword-predicting E2E systems, evidence shows that accuracy of subwords is still inadequate for rare words. Moreover, subword-based E2E systems limit direct application of external word-based LMs and usualy utilize rather ad-hoc units such as byte-pair encoding units (BPEs) instead of linguistically motivated words and characters, which hurts system explainability. Finally, new word analysis may be beneficial for downstream and user-experience-oriented applications, such as personalized auto-complete, slot filling, etc. However, a subword system does not "know" that an OOV is discovered and cannot use this information. Thus, in an E2E environment, our goal again would be to detect OOVs and to find their representations in the form of phonemes or numerical vector representations.

1.1 Tasks and Metrics

In this section, we introduce the speech data mining tasks that are relevant to this research and explain evaluation metrics for each of them.

1.1.1 Automatic Speech Recognition

ASR Task Definition

Automatic speech recognition (ASR) is one of the most complex speech processing tasks. The ultimate goal of ASR is to transcribe text from acoustic input. The standard approach is to find the most likely sequence of words $W^* = w_1^*, ..., w_n^*$ given an utterance represented as a matrix of acoustic features $\mathbf{X}$.

$$W^* = \arg\max_W P(W|\mathbf{X}) \approx \arg\max_W P(\mathbf{X}|W)P(W), \tag{1.1}$$

where $P(W)$ represents the LM, and $P(\mathbf{X}|W)$ is the likelihood of the acoustics given a particular transcript. While in a hybrid setup the LM is separate from the acoustic model (AM), and they are combined via graph composition during decoding, in an E2E sequence-to-sequence setup, they are less separable.

ASR Metrics: Word Error Rate

The metric for evaluating ASR system performance is usually the word error rate (WER), which is a version of Levenshtein distance on the word level. The system output is aligned to and compared with the reference annotation, and WER is calculated as a ratio of insertions (I), deletions (D) and substitutions (S) required to change one sequence into the other to the total number of words (N) in the reference sequence:

$$WER = \frac{I + D + S}{N}. \tag{1.2}$$

There are several approaches to factor OOVs into the WER calculation. For example, in a widely used NIST SCLITE scoring tool[1], transcriptions variants are allowed and treated as correct, and OOVs (labeled as <unk>) are excluded from the WER estimation. This exemplifies the attitude of accepting that nothing can be done with OOVs in a classic ASR system, but hardly reflects the experience of the user with the final transcript.

[1] http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm

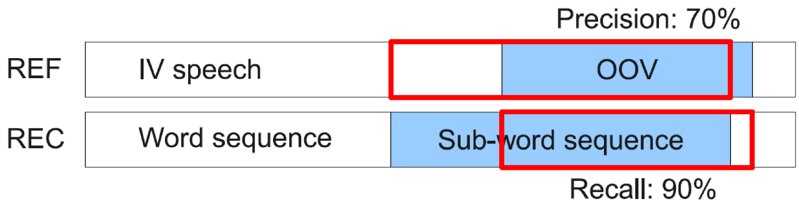


Figure 1.1: f-score precision and recall according to [Kombrink et al., 2010b].

In our experiments, we use WER to evaluate ASR performance the following way: we calculate the distance between the predicted transcriptions that include <unk> label and the unchanged reference transcriptions. In this setup, predicting the OOV label always causes an error. This can be seen as the baseline WER, the one we will aim to improve by applying OOV processing techniques. We call this metric WER1 (see the upper panel of Figure 1.2).

1.1.2 OOV Processing Tasks

We can separately define two tasks in OOV processing pipeline, one that we call detection and another recovery. There are no standardised success metrics for these tasks as there is for ASR, therefore we describe ours in detail.

OOV Detection Task Definition

Detection task deals with finding places in speech which are OOVs and should return start and end times of where an OOV can be found. The output of the detection step is tuples of start and end times of OOV regions. The times can be represented in frames or seconds.

Detection is treated as a separate task in Chapter 5, where we explicitly evaluate the detected times and apply them for OOV extractions, and in Chapter 4, where detection is done on full lattices, and OOV lattices start and end times are evaluated based on time alignment of the lattices. In the speller setup in Chapter 6, we estimate detection just based on the system output, as time information is not needed.

OOV Detection Metrics

In case the goal is to explicitly evaluate OOV detection, we need to calculate how well the detected OOV regions corresponds to reference OOV regions. Time overlap between reference and hypothesis OOV regions is rarely ideal, so a good success metric would show how good the partial detections are.

In the experiments in Chapter 4, we use the OOV detection f-score proposed in [Kombrink et al., 2010b]. This f-score is calculated for every detected OOV candidate and shows the quality of the detection. Precision is calculated as the overlap duration divided by the hypothesis OOV duration and recall is the overlap duration divided by the reference OOV duration. An illustration is provided in Figure 1.1.

F-score for each detection is then calculated as:

$$f = \frac{2 \times precision \times recall}{precision + recall}. \tag{1.3}$$

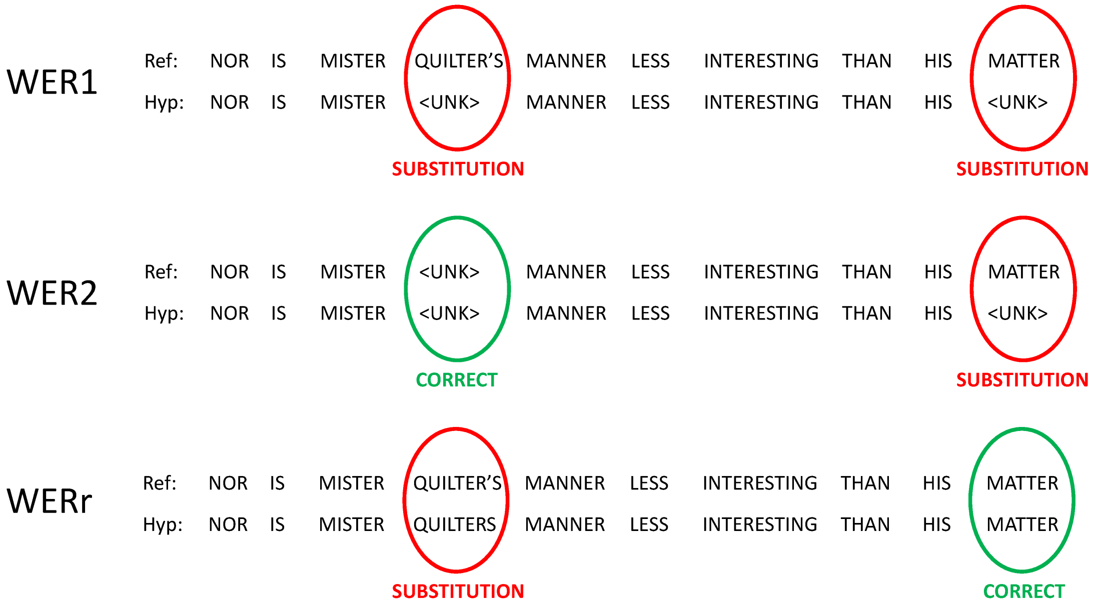

WER1
Ref: NOR IS MISTER QUILTER'S MANNER LESS INTERESTING THAN HIS MATTER
Hyp: NOR IS MISTER <UNK> MANNER LESS INTERESTING THAN HIS <UNK>
SUBSTITUTION
SUBSTITUTION
WER2
Ref: NOR IS MISTER <UNK> MANNER LESS INTERESTING THAN HIS MATTER
Hyp: NOR IS MISTER <UNK> MANNER LESS INTERESTING THAN HIS <UNK>
CORRECT
SUBSTITUTION
WERr
Ref: NOR IS MISTER QUILTER'S MANNER LESS INTERESTING THAN HIS MATTER
Hyp: NOR IS MISTER QUILTERS MANNER LESS INTERESTING THAN HIS MATTER
SUBSTITUTION
CORRECT

Figure 1.2: Comparison of WER1, WER2, and WERr scores as proposed and used throughout the thesis.

Detections with $f = 0$ are false alarms, detections with $f = 1$ are perfect matches and f-scores of partial overlaps of reference and detected OOVs will range between 0 and 1. For system scoring, an average f-score across all candidates is calculated.

Disappointingly, this logical and informative metric has not been widely accepted, and most of the papers reporting OOV detection choose to use hard decisions instead of a soft f-score overlap metric. So, when we needed to compare detection results with other papers, like in Chapter 5, we defined a successful detection as following: an OOV occurrence is treated as a true positive if the hypothesis overlaps with the reference for more than half of the reference duration. Thus, the detection recall is calculated as the percentage of true positives in the reference list of OOVs, and detection precision as the percentage of true positives in the list of detected OOV occurrences.

As a way of assessing detection success directly from output labels, without time consideration, we use a modified WER score that we call WER2. As we want to isolate the errors that happen due to the appearance of OOVs, WER2 treats the OOV label as a word. In terms of WER2, when a system predicts the OOV label for a reference word that is an OOV with the current vocabulary, there is no error. To obtain WER2, we calculate the distance between the predicted sequence and the reference sequence with all OOVs substituted with <unk> label (see Figure 1.2). WER2 will then show how good a system is in predicting words labels, including the OOV label, which is essentially OOV detection. The differences between WER1 and WER2 show how many of WER1 errors are due to vocabulary limitation. With bigger vocabularies, this difference is smaller, as the OOV rate is smaller as well.

OOV Recovery Task Definition

OOV recovery task aims to recover acoustic and/or graphemic representation of OOVs or link them to an existing word identity.

In Chapters 4 and 5, the task is to recover recurrent OOVs by clustering similarly pronounced detected OOV candidates. The output of this recovery process is a list of previously unseen OOVs in their graphemic representation. Successful recurrent OOV recovery depends on 1) enough occurrences of the same word present in the text and 2) successfully detected, 3) the clustering assigning enough of these occurrences to the same cluster, and 4) the phoneme-to-grapheme (P2G) conversion correctly discovering the graphemic representation of the word from its subword unit representation.

The speller architecture (Chapter 6) does not rely on recurrency of OOVs to recover their pronunciation. Instead, it trusts the trained speller model to predict the spelling of every encountered OOV directly from the state information of a word predicting network (WPN). In this case, the goal of recovery is to produce correct spelling for every occurrence of an OOV word.

OOV Recovery Metrics

When reporting recurrent OOV recovery success, this list of recovered OOVs is compared to the list of the reference OOVs (see Appendix A) with the help of Levenshtein distance. A word is marked as recovered (true positive) if its graphemic representation does not differ from some reference word by a distance more than 1. For example, a recovered word "morover" (closest dictionary word "moreover") would be considered correctly recovered, but "anctious" (closest dictionary word "anxious") not. Recovery recall shows the percentage of OOVs from the reference list that were recovered, and recovery precision shows the percentage of the recovered words that belong to the reference OOV list.

As the speller architecture (Chapter 6) does not rely on recurrency of OOVs to recover their pronunciation, we do not use the list of OOVs to report recovery in speller experiments. Instead, we report WERr – recovery WER. Speller predicts words from a word predicting network, and whenever an OOV is generated, it outputs its spelling given by the speller. WERr is calculated from the alignment with the original reference transcriptions, same as WER1, and the spelled word is correct only when it is the same as in the reference transcription (see Figure 1.2). WERr improvement in comparison to WER1 shows the system's potential for OOV recovery with completely correct spelling.

1.2 Thesis Claims

The main contributions of this thesis are the following:

- **OOV processing field in general**: We separately define the two tasks of OOV processing, namely OOV detection and OOV recovery. We propose and adapt representative metrics for both and provide results with the usage of these metrics on a well-known open-source database LibriSpeech. Thus the thesis promotes standardization and replicability in a somewhat disjointed OOV field.

- **FST-based OOV detection and recovery in a hybrid ASR system**: we show the benefit of working with full probabilistic lattices as opposed to one-best output at every step of OOV detection and recovery from a hybrid ASR with weighted finite-state transducer (WFST) – based decoding.

- **OOV detection in an E2E ASR system**: we investigate the usefulness of inner representations of an E2E LAS ASR system for locating word boundaries; attention and connectionist temporal classification (CTC) approaches are compared. Moreover, a principled probabilistic clustering method for lattices is introduced.
- **Speller architecture for OOV detection and recovery**: we introduce a new architecture for jointly training an E2E ASR with two granularities – words and characters. Training with speller both allows OOV recovery directly during E2E decoding and also improves the word-predicting WER. We also provide a lot of analysis into the information learned by different parts of a LAS decoder.

1.3 Thesis Structure

This thesis is structured as following: Chapter 2 provides an overview of the field of OOV detection and recovery. Chapter 3 describes the Librispeech dataset and its adaptation for our experiments. Chapter 4 proposes a procedure for recurrent OOV detection and recovery in a classic hybrid ASR system. OOV hypotheses are generated from the output lattices provided by a decoding with the usage of a hybrid word-subword FST. The candidates in a lattice form are then clustered in order to discover repeating phoneme patterns that suggest OOVs to be added to the dictionary and the LM. Chapter 5 introduces the usage of E2E LAS architecture for OOV detection: start and end times of OOV regions are inferred from either attention weights or CTC alignments. The detected times are used for recovery of phonetic representation from a phoneme system. Chapter 6 presents a new speller architecture that allows for joint training of a word-predicting system and a speller trained to predict graphemic representation of words from embeddings and other inputs. Finally, Chapter 7 provides conclusions, takeaways, and a look to the future.

Chapter 2

OOV Field Overview

OOV research is most relevant to the tasks of ASR, natural language processing (NLP), and key word spotting (KWS). In this chapter, we will first talk about sub-word units, and then we will present existing approaches to handling OOVs within two distinct flavors of ASR modeling: a classical component-based approach and a more recently popular E2E approach. Where relevant, work from NLP and KWS fields that provided us with inspiration will be cited also, but the tasks will not be described in depth.

2.1 Sub-Word Units in OOV Representations

One of the most common ways to represent OOVs is to use smaller than word units – sub-word units – as recognition targets. This term can be an umbrella for units derived in many different ways.

The first group of sub-word units are **linguistically motivated** sub-word units. These can include phonemes, morphemes or syllables. Although these units have the benefit of explainability (and being liked by the linguists), their usage requires language-specific expert data. For example, if a transcription dictionary is used in an ASR system to provide mapping of words into phonemes, this dictionary has to be generated by experts first. Splitting words into linguistically "correct" syllables or morphemes also usually requires some rule-based conversion system. We will use phonemes as sub-word units in the framework of a component-based ASR system in Chapter 4, where a dictionary is provided.

Another group of sub-word units are **data driven** units. Uncovering these units does not require any external knowledge, just the analysis of the data itself. The simplest sub-word units is a character. Although there is no guarantee that there is any one-to-one mapping between a character and a sound, strong E2E models do not care about that, and character outputs are fairly popular. We will use characters for out speller architecture experiments in Chapter 6.

With the release of the fast Sentencepiece algorithm [Kudo and Richardson, 2018] and a related free tool[1], segmentation based on the byte-pair encoding unit (BPE) compression algorithm [Sennrich et al., 2016], for better of for worse, has solidified as the most frequently used for discovering subword units. BPE [Gage, 1994] is a data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. For BPE unit discovery, instead of merging frequent pairs of bytes, characters or character sequences are merged. The symbol vocabulary is initialized as the character vocabulary

[1] https://github.com/google/sentencepiece

with a word boundary symbol. Iteratively, each occurrence of the most frequent pair ('A', 'B') is replaced with a new symbol 'AB'. Each merge operation produces a new symbol which represents a character n-gram. Frequent character n-grams (or whole words) are eventually merged into a single symbol. Discovered set of BPEs depends on the training data and on the number of units the algorithm was tasked to discover. We will use BPE-predicting systems for out E2E baselines in Chapters 5 and 6.

Even though BPEs dominates the sub-word scene these days, there are still efforts to bring back acoustic motivation into subword unit discovery. For example, [Zhou et al., 2021] propose an acoustic data-driven subword modeling approach, in which BPE-like merging is performed on grapheme-to-phoneme (G2P)-initialized phonetic units and later mapped to characters. These units produce a more "logical" segmentation of the text and achieve better performance as targets on Librispeech data than the classical BPEs. Some of older but interesting data driven approaches to sub-word unit discovery also include spectral clustering plus machine translation (MT) [Hartmann et al., 2013], block diagonal infinite hidden Markov model (BDiHMM) [Vanhainen and Salvi, 2014], and others.

2.2 Component-based ASR

Component-based speech recognition is a widely used approach to speech modeling for ASR with a rich history. First, the basics of applying statistical pattern recognition to a problem of ASR have been formulated as long ago as in the 1970s [Jelinek, 1976]. A continuous speech signal can be parameterized by splitting it into (assumed stationary) frames and extracting relevant features from these frames. The most popular features used in ASR are Mel-frequency cepstral coefficients (MFCC) [Davis and Mermelstein, 1980] and perceptual linear predictive (PLP) [Hermansky, 1990] features. In order to model the correspondence between the input features and the output word sequence (see section 1.1), a classic component-based speech recognition system combines three components: AM, LM, and a dictionary:

- Acoustic model (AM) connects input features and phonetic units (usually phonemes or context-dependent phonemes) that are responsible for generating them. AM corresponds to $P(X|W)$ in (1.1). Because of the sequential nature of speech and the assumption of conditional independence, AM is well suited to be modeled as an hidden Markov model (HMM), each state of which can generate a feature vector using the distribution associated with the state. The state output probability density functions are modeled by, classically, a Gaussian mixture model (GMM) [Juang et al., 1986] or, more recently, a NN [Hinton et al., 2012]. Component-based ASR that uses a combination of HMMs and NNs to model speech [Bourlard and Morgan, 1994] is called a hybrid ASR.

- Language model (LM) is a probability distribution over sequences of words, which means that it estimates the likelihood of certain strings of words appearing in the system output. It corresponds to $P(W)$ in (1.1). LM is usually represented with an n-gram model (trigrams are most common) assigning each word a conditional probability given the word history [Kneser and Ney, 1995]. Recently, recurrent neural network (RNN) – based LMs have been gaining popularity [Mikolov et al., 2010].

- Dictionary is a mapping of words to their phonetic representations, and as such, it connects the aforementioned two models. The dictionary may be a one-to-one mapping or it can allow several variants how a word can be pronounced. Usually, the dictionary is given (expert-generated), but in case of insufficient expert effort available, a wider dictionary can be generated automatically using G2P models trained on a short expert dictionary [Bisani and Ney, 2008]. There have also been efforts to discover the dictionary in a completely unsupervised way from just a phoneme recognizer output (see, for example, [Heymann et al., 2014]).

All the components of the component-based ASR have to be combined for the decoding, and this is done with the help of WFSTs [Mohri et al., 2008]. The process of building the decoding graph is described in detail in Chapter 4 with further references.

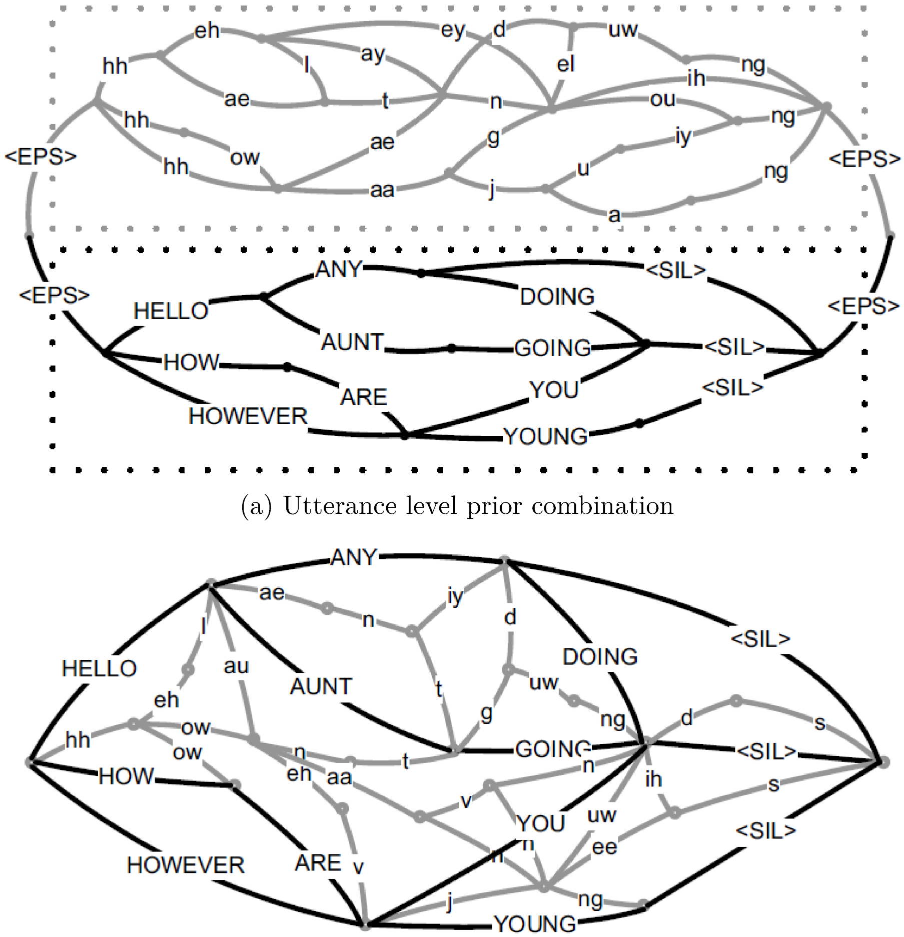
2.3 OOVs in Hybrid ASR

In the context of a component-based ASR, OOVs are defined literally as words which are not in the dictionary. As the dictionary provides information connecting AM with LM, a system is not able to output the correct word, as the decoding graph limits the search space to known words only. Often, instead of outputting something useful for the user, the system will simply incline towards a word which is allowed by the language model and is phonetically close to the OOV word in question.

With an addition of a bigger external LM, the OOV problem can be partially alleviated by expanding the dictionary with **automatically generated transcriptions** for all the words in the bigger LM. One of the popular ways of training a pronunciation model that we also make use of in our experiments is G2P conversion with joint-sequence models presented by [Bisani and Ney, 2008]. The approach makes use of joint units called graphones which carry both input and output symbols; a sequence of such graphones can model a combination of input and output strings. Graphone sequences are presented as WFSTs and the model is trained on the dictionary using maximum likelihood estimates. The trained model can be used to transcribe input grapheme strings into a phonemic transcription by choosing the most likely graphone sequence. [Bisani and Ney, 2008] shows that, with appropriate smoothing, the system achieves 9.47 % phoneme error rate on different English datasets (Librispeech not included). After expanding the dictionary with the newly-generated pairs of words and transcriptions, the decoding graph will have to be recompiled with the updated dictionary and the LM. However, first, this in no way solves the issue of domain-specific or new OOVs that are completely unseen and not included in the bigger LM, and second, these automatically generated transcriptions are inferior to expert-generated ones.

The idea of **hybrid decoding**, which is also a popular robustness technique in KWS, is that the system outputs the words that are in the dictionary if there is sufficient probability for them and goes one level below and outputs a sub-word string when no word has sufficient probability to be output. That way, the user gets some important information about the acoustics of the segment of speech instead of getting a wrong word hypothesis. There are different ways the two granularities – word and subword – can be combined to perform hybrid word-subword decoding. In general, there are two different approaches: prior combination and posterior combination [Yu and Seide, 2004].

In case of **posterior combination**, the decoding is done separately on two separate systems of different granularities – word and subword – and the hypotheses are evaluated with a combination of word and subword scores. Alternatively, the decoding works on a word level and switches to a sub-word level only in case the output does not fit pre-set conditions (e.g. minimum confidence score etc.) [Lee et al., 2015, Yazgan and Saraclar, 2004, Kombrink et al., 2010a]. In this approach, OOV detection and recovery stages are separated:



(a) Utterance level prior combination

(b) Word level prior combination

Figure 2.1: Example of a decoded lattice in a hybrid system with different prior combinations from [Yu and Seide, 2004].

low confidence of a word system means an OOV is detected, and then its representation is recovered from a subword system. The drawbacks of posterior combination are time and space consumed by essentially doing the decoding twice and keeping twice as many lattices than in a regular system.

The two granularities – word and subword – can also be combined in a **prior combination** to perform hybrid word-subword decoding. In this case, the decoding graph contains both word-level and sub-word-level paths. This is achieved by combining word and subword language models in the graph generation step. The combination can be performed on either the utterance or the word level (see Figure 2.1). If the combination happens on an utterance level, the decoder can choose either the word or the sub-word path. The word level combination has the benefit of the system being able to choose sub-word or word path at every word boundary. Word level prior combination allows combining the two tasks – OOV detection and recovery – in one: an OOV is detected when output is string sub-word units, and these units also provide acoustic representation of the OOV.

In the field of KWS, a more principled approach to hybrid decoding graph construction has been proposed. In [Szőke, 2010], the decoding graph is modified in such a way that

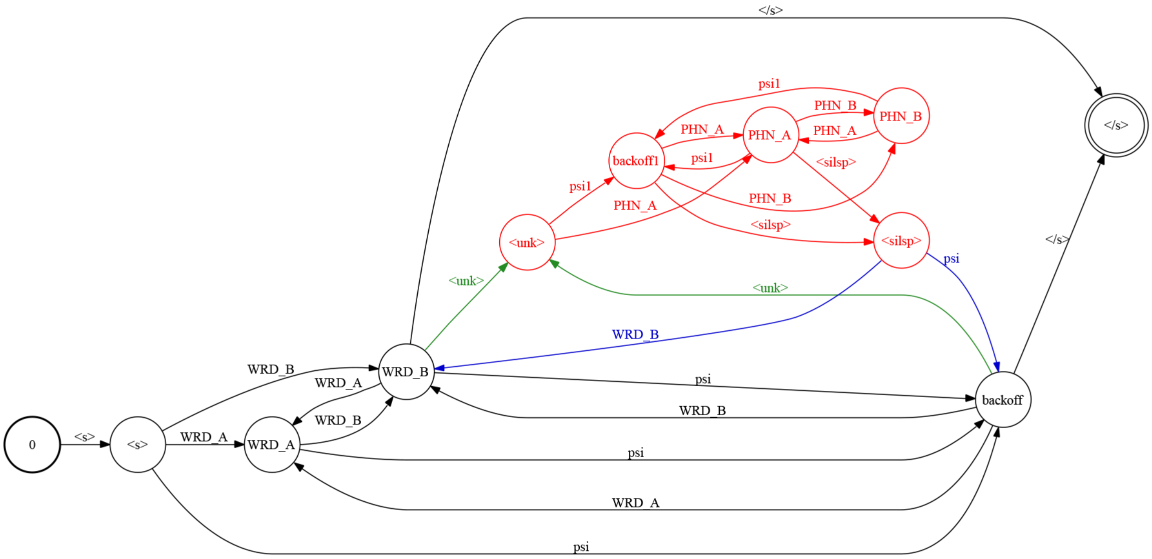


Figure 2.2: Example of inserting subword model in place of an OOV in the decoding graph taken from [Szőke, 2010].

in place of every link with an OOV label, a subword graph is inserted (see Figure 2.2). Decoding graph built like this can thus produce either outputs consisting of just words or combined outputs, consisting of word and subword units. The decision whether the subword region of the decoding graph comes into play or not depends on the comparative costs of the paths through the graph. This allows for preserving correct posteriors from both word and subword graphs.

This principled approach has been adopted for OOV detection and recovery in [Kombrink et al., 2010b]. In it, the OOV detection potential is analyzed with the f-score that takes into consideration partial overlaps (see 1.1.2), and detections with high f-score are used to recover OOVs as strings of phonemes from one-best output and map them to graphemes using a P2G system [Bisani and Ney, 2008]. With this approach, they were able to detect about one third of all OOV words, and were able to recover the correct spelling for 26.2 % of all detections on TED talks database[2].

Prior combination can also be achieved by incorporating sub-word units into the LM instead of modifying the decoding graph. For instance, [Bisani and Ney, 2005] substitutes rare words with their phoneme strings to obtain training data for the language model, while [Shaik et al., 2011] adds morphemic subwords to the lexicon. In [Qin and Rudnicky, 2013], the pronunciation of OOV words is estimated through the G2P conversion, and then used to train the sub-lexical units. After that, OOV words in the training text were replaced by corresponding sub-lexical units to get a new hybrid text corpus from which a LM is trained. Thus, they combine word and sub-word levels into a single flat hybrid LM. The decoding graph created from such a model is also capable of performing OOV detection and recovery with the same result as word-level graph combination.

[2] https://www.ted.com/

2.4 End-to-End ASR

E2E techniques have long taken over hybrid approaches to ASR and continue developing with remarkable speed. Unlike a component-based ASR system, E2E directly maps a sequence of input acoustic features into a sequence of output labels (characters, sub-word units, words, etc.). The system is somewhat of a black box, trained to optimize the single objective instead of individual components. Not only does it help with finding the global optimum (given there is enough data) and out-performing the classical ASR systems, it also greatly simplifies the training process, which in combination explains the popularity of the approach. The most popular milestone architectures are connectionist temporal classification (CTC), attention and transformer architectures.

CTC was the first E2E system to be widely used in ASR, and it was first described in [Graves and Jaitly, 2014]. In it, the neural network (five levels of bidirectional LSTM) outputs probabilities of acoustic units and the probability of a blank symbol for each frame. The probability of a sequence of acoustic units is found by summing the probabilities over all possible alignments (remove repeated symbols, remove blank symbols). The training is possible because of the long short-term memory (recurrent neural network)s (LSTMs) [Hochreiter and Schmidhuber, 1997] capability of remembering (and forgetting) information selectively over longer time series. To further simplify the training, no feature extraction was performed for the input, just the spectrogram was used. When compared with the traditional hybrid deep neural network (DNN)/HMM system, CTC model has achieved state of the art (SOTA) results on WSJ dataset.

Another milestone was Listen Attend and Spell architecture (LAS) [Chan et al., 2016] neural network for ASR. LAS system has two components: a listener and a speller (also known as encoder and decoder), which are trained jointly. The listener/encoder is a pyramidal recurrent network encoder that accepts filter bank spectra as inputs. The speller/decoder is an attention based recurrent network decoder that emits characters as outputs. The network produces character sequences without making any independence assumptions between the characters, which is the key improvement of LAS over previous end-to-end CTC models. Even though LAS did not surpass the current state-of-the-art hybrid DNN-HMM model (on a subset of the Google voice search task), the idea took off. Later, [Zeyer et al., 2018] showed that with BPE units combined with an external LM, a LAS-like E2E system can achieve SOTA results on Switchboard and Librispeech.

CTC and attention architectures have been successfully combined in [Watanabe et al., 2017] for character-predicting E2E ASR. In the proposed hybrid CTC/attention architecture, a CTC-based decoder and an attention-based decoder share the encoder (see Figure 2.3), and the system is optimized with a combination of CTC and attention losses. The attention objective is an approximated letter-wise objective, whereas the CTC objective is a sequence-level objective. Therefore, this multi-objective learning is able to mitigate this approximation with the sequence-level CTC objective, in addition to helping the process of estimating the desired alignment. The paper demonstrated better performance in terms of character error rate of this hybrid approach in comparison with CTC-only and attention-only approaches on several datasets, including WSJ and CHiME-4.

With tens of thousands of citations for the seminal “Attention is all you need” [Vaswani et al., 2017], the introduction of Transformers was the next big milestone. Transformers use multi-headed self-attention in both encoder and decorer, without the use of recurrence or convolution. With the ease of parallelization enabled by the transformers, new big architectures have been introduced, such as, for example, convolutional-augmented transformer

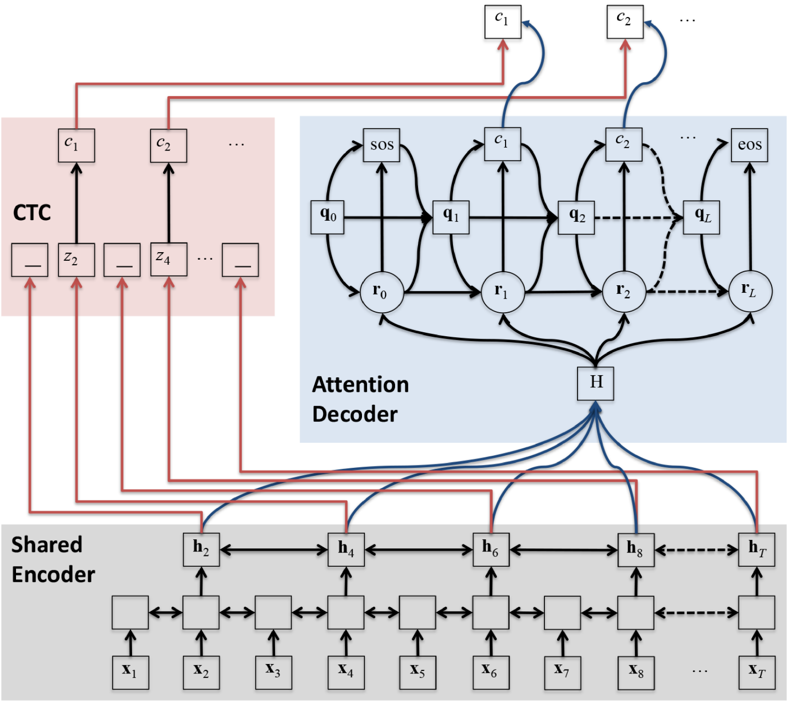


Figure 2.3: Hybrid CTC/attention architecture from [Watanabe et al., 2017].

architecture (Conformer) in [Gulati et al., 2020] and Perceiver [Jaegle et al., 2021]. Transformers have also enabled effective pre-trained models such as BERT [Devlin et al., 2018], W2V-BERT [Chung et al., 2021] and WavLM [Chen et al., 2022].

2.5 OOVs in an E2E Framework

The majority of the E2E systems predict subword units, most often phonemes [Chorowski et al., 2015a] or BPEs [Sennrich et al., 2016], and thus no OOVs are present. However, some systems still report on predicting words [Soltau et al., 2017] in the interest of preserving wider context information, and these system do face an OOV issue.

Notable for us, a word-generating E2E system has been used for OOV detection. In [Thomas et al., 2019], an E2E system predicting directly word sequences with attention mechanism was used for discovering positions of OOV words. For each output label, attention points to relevant input frames. Therefore, frames with the maximum attention weight for the OOV label can be used to find OOV center frame. These positions of OOVs were then used to improve subtitles generation with the use of subword units on broadcast data. This paper was used as an inspiration for our E2E OOV detection experiments presented in Chapter 5. However, we did not observe the performance they report of attention maximums pointing necessarily in the centers of words.

Experiments have also been conducted using two levels of granularities – words and phonemes – in an E2E setup. For example, [Li et al., 2017] uses CTC training to implement backing-off to phoneme level when an OOV label is produced. Sometimes, instead of two granularities, a mix of word and sub-word units is used at the same level, for example, as the neural network (NN) output for CTC [Li et al., 2018].

2.6 Post-processing of OOVs

After a successful discovery of OOV words as strings of sub-word units, further post-processing can be performed. We have already introduced P2G conversion using [Bisani and Ney, 2008] in Section 2.3, and this is the most common step if an OOV was recovered in a phonetic form. Another common post-processing includes associating the discovered OOVs with IVs. For example, [Kombrink et al., 2010a] uses a per-phoneme similarity scores to check if the discovered OOVs are derivatives of words existing in the vocabulary.

If OOVs are repeating, which is a reasonable assumption for most new, trending words (which we are interested in recovering), the strings of sub-word units can be clustered to discover these repeating OOVs. For example, in [Qin and Rudnicky, 2013], after OOV candidates are detected using a flat-hybrid LM decoding, a bottom-up clustering is performed to iteratively find multiple instances of the same OOV word. Each OOV candidate starts as a single cluster and in each iteration, two clusters with the smallest distance are merged. We adopt the same approach in Chapter 4 before switching to a more complicated probabilistic clustering. As a similarity metric, [Qin and Rudnicky, 2013] use a combination of phonetic edit distance, acoustic distance between feature vectors, and context distance to achieve recovery rates of 40 % to 80 % on different databases for recurrent OOVs.

[Hannemann et al., 2010] made use of an error WFST to align phoneme paths of discovered OOVs to each other and to find closest candidates for clustering. The usage of error model allowed to recover somewhat from the limitations of working with one-best output path and also to adapt to repeating error pattern. We were inspired by this work to continue bringing fuzziness into OOV recovery and going for a full-lattice approach to clustering (see Chapter 4).

2.7 OOVs in NLP

Serving as an inspiration for our speller architecture for ASR is the speller architecture for open-vocabulary NLP introduced in [Mielke and Eisner, 2018]. In it, the LM for generating the corpus of tokens consists of two RNNs: the first one captures the sentence structure, and the second one, called the speller, captures the word structure. The speller can generate new word types following the spelling style of IVs. The novel words generated by this model fit the grammatical sentence structure well, but otherwise the model can produce a range of possible spellings that fit the language in question.

Transformers have been widely used for pre-training in NLP tasks too, however, predefined subword-based general-domain vocabularies do not perform well when used in the context of specialized domains. Therefore, work has been done on modeling two granularities in NLP as well, for example, CharacterBERT [El Boukkouri et al., 2020] drops the wordpiece system and uses a Character-CNN module instead to represent entire words by consulting their characters, producing robust, word-level, and open-vocabulary representations. They show that this model improves the performance of BERT on a variety of medical domain tasks. [Aguilar et al., 2020] proposes a character-based subword module (char2subword) that learns the subword embedding table in pre-trained models like BERT, building representations from characters out of the subword vocabulary. The module is robust to character-level alterations such as misspellings, word inflection, casing, and punctuation, and improves the performance on the social media linguistic code-switching evaluation (LinCE) benchmark.

Chapter 3

Data and OOV Simulation

The real case scenario of OOV detection and recovery would be continuously collected new data with newly emerging words or highly specialized data with topic-specific words. Due to the nature of the task, most of the relevant data is kept private, and a lot of research mentioned in Chapter 2 uses proprietary data to showcase their results, which makes them not easily replicable. To avoid that pitfall, we choose free access data to conduct our experiments.

3.1 Librispeech Dataset

The database for experiments was LibriSpeech ASR corpus of read audiobooks [Panayotov et al., 2015]. The choice of the dataset was dictated by the fact that it is free access, big in size, and also it is clean speech that would exclude the complications of having a bad WER baseline interfere with the OOV tasks. No external data was used in any of the experiments either for acoustic or language model (pre-)training.

LibriSpeech consists of several datasets of varying quality: for training, there are sets of 100 and 360 hours of "clean" speech (lower-WER speakers) and 500 hours of "other" data (higher-WER speakers). In addition, there are two developement and two test sets, also "clean" and "other". Each of the four evaluation sets is about 5 hours 20 minutes long. There is no speaker overlap between test and train data and the amount of speech by each speaker is balanced. In total, there is 1000 hours of 16 kHz data. There are several n-gram LM provided with LibriSpeech dataset[1]. For our experiments, we used the 3-gram ARPA-style LM trained on 14500 public domain books with around 803 million tokens in total and 900 000 unique words. The official dictionary contains 200 000 words.

3.2 ASR Training Data

Table 3.1 summarizes how different sets of Librispeech data are used in the experiments of the thesis.

For our hybrid setup in Chapter 4, the acoustic model was trained on 100 hours of clean data, and the Librispeech LM was used. Recurrent OOV detection and recovery is performed on the 360 hours of clean data – keeping a lot of unseen data for OOV tasks was necessary to provide enough OOV occurrences for successful clustering. WER is reported on a combined dataset of all dev and test subsets.

[1] http://www.openslr.org/11/

dataset		experiments set		
		FST-based Chapter 4	E2E Detection Chapter 5	Speller E2E Chapter 6
train-clean-100	100.6 hrs	train	train	train
train-clean-360	363.6 hrs	OOV recovery	OOV recovery	train
train-other-500	496.7 hrs	-	train	train
dev-clean	5.4 hrs	dev, test	dev, test	dev, test
dev-other	5.4 hrs	dev, test	dev, test	dev, test
test-clean	5.3 hrs	test	test	test
test-other	5.1 hrs	test	test	test

Table 3.1: LibriSpeech data usage for thesis experiments.

For the E2E detection experiments in Chapter 5, the system is trained on 100 hours of clean plus 500 hours of other, as more data is needed for a successful E2E training. OOV detection and recovery is again done on the remaining chunk of 360 hours of clean data to provide enough recurrent OOV occurrences. WER is reported separately on each dataset for comparison with other results in literature.

In the speller E2E setup, OOV recurrency is not a requirement for recovery, so all the training data can be used for actual training. Thus, in Chapter 6, the training was done on all the training sets, 960 hours in total, and results in terms of different WERs as described in Section 1.1 were reported on all the test sets.

3.3 OOV Simulation Setup

In a real-world scenario, OOVs would be newly-coined words and names, but in audiobooks, this is not a viable setup – the corpus majorly consists of free domain books, which are predominantly from the 19th century. Moreover, in a real OOV scenario, there is no information in the data about the OOVs, how they are written or pronounced. So OOVs need to be artificially simulated in order to be able to evaluate detection and recovery.

For a word-predicting E2E system, OOV introduction is fairly staightforward: the number of output word labels has to be reduced from the initial 200000 to a manageable amount of outputs, and the less common labels will then be deemed OOVs and a special output label will be assigned to represent them. For our E2E training, the number of word labels has been reduced from 200 000 to 5000 most common words (min. 137 occurrences per word, 10.4 % OOV rate), 10000 most common words (min. 53 occurrences per word, 5.9 % OOV rate) and 27 000 most common words (min. 9 occurrences per word, 1.7 % OOV rate) for different experiments.

For the recovery techniques presented in Chapters 4 and 5, OOV recurrency is crucial: the unseen OOVs must be repeated enough times for the system to discover it through clustering. Thus, relying only on infrequent words as chosen OOVs is not a viable setup. Therefore, we simulate the real-world scenario of new appearing words. We "reverse" the task and designate archaic and out-of-usage words as OOVs. These words are not likely to be in a modern LM trained on Internet data and thus fit the OOV function.

In order to choose archaic words as OOVs, we used Google ngram dataset of word usage statistics in books [Lin et al., 2012]. For each word, the database provides its number of occurrences in sources published each year over the last 5 centuries. We calculate the relative frequency of a particular word in a particular year by dividing the number of occurrences of this word this year by the total number of words in this year's publications. For our purposes, words with twice as much frequency before year 1900 than after 1900 are chosen as OOVs. Moreover, all names are also added to the OOV list. Whether the word is a name can be checked by the relative number of its occurrences in the ngram with a capital letter and without.

The resulting list of OOVs picked as described above is given in Appendix A complete with OOV frequencies. The list consists of 1000 designated OOVs, which present an example of 19-century bookish English. For example, it includes such words as *interposed*, *hastened*, *mademoiselle*, *indignantly*, *countenance*, etc. With the OOV list obtained as a result of this method, the OOV rate (percentage of OOVs in all the words) on the 360 hours clean set reaches 1.5 % with the default Librispeech 200 000 word dictionary.

3.4 Data Availability

All the experiments in the thesis are conducted on open access data in the interest of replicability.

LibriSpeech corpus of read books is available[2] under Creative Commons license (all types of usage allowed with proper attribution). LibriSpeech is a widely used and reported on dataset, which has recently been expanded for unsupervised training with the release[3] of Libri-Light 60k hours of unlabeled data. Libri-Light can be used under MIT license.

Most of the popular ASR toolkits have pre-made LibriSpeech recipes available, notably Kaldi[4] and espnet[5].

Ngram Viewer graphs and data are available[6] to be freely used for any purpose.

Our list of 1000 simulated OOVs is also available[7] for download.

[2] http://www.openslr.org/12/

[3] https://github.com/facebookresearch/libri-light

[4] https://github.com/kaldi-asr/kaldi/tree/master/egs/librispeech

[5] https://github.com/espnet/espnet/tree/master/egs/librispeech/asr1

[6] https://books.google.com/ngrams/

[7] www.fit.vutbr.cz/~iegorova/public/LibriSpeech_1000_OOV_list.txt

Chapter 4

WFST-Based OOV Detection and Recovery in a Hybrid ASR System

This chapter covers possibilities for OOV detection and recurrent OOV recovery in the framework of a WFST-based decoder of a hybrid ASR. OOVs are discovered in the form of lattices of sub-word units, and then a free clustering of these discovered candidates is performed in order to find recurring sequences and to add them to the dictionary as new words.

Unlike previous hybrid decoding approaches mentioned in Chapter 2, the work covered here attempts to automatically discover new words in a decoding lattice rather than on the one-best hypothesis. A sub-word decoding lattice may contain paths that correspond to slightly different pronunciations. Thus, clustering performed on lattices instead of one-best output strings allows us to discover OOV patterns even if the same OOV is pronounced somewhat differently on different occasions. Moreover, this approach should be more robust in the case of an ASR output of low quality. These proposed OOV lattices can be seen as pronunciation models of OOV words. After discovering new words in this manner, we can suggest how they would look in graphemic form, for example using a pre-trained P2G system.The pairs of graphemic and phonetic representations of the newly discovered words can then be added to the dictionary.

In this chapter, first, the WFST-based decoding is presented, then we describe the recurrent OOV recovery procedure and comment on the results. The core of the OOV detection and recovery procedure described in this chapter was published in [Egorova and Burget, 2018]. In the thesis, we describe the WFST operations, lattice indexing, and candidate extraction in more detail. Moreover, there is more discussion of how the parameters of the hybrid decoding graph affect the detection and recovery performed on the lattices resulting from the decoding with this graph.

4.1 WFST-based ASR Decoding

In a component-based ASR system, the decoding is usually performed with the use of a decoding graph to limit the search space. This is facilitated by using WFSTs to combine different levels of recognition system models: AM, LM, and the dictionary [Mohri et al., 2008]. Here, we introduce some useful definitions for the WFST framework, most of them adapted from the iconic [Mohri et al., 2002].

- A **semiring** defines how the weights are combined during various operations on a WFST. A semiring is an abstraction that allows to define all WFST operations disregarding what exact mathematical operations are used for weight combination. A semiring is given by a tuple of numbers, two operations, $\oplus$ and $\otimes$, and identity elements $\overline{1}$ and $\overline{0}$:

$$\mathcal{K} = \{\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1}\}. \tag{4.1}$$

  In the framework of ASR, mostly two semirings are used: tropical and log. We will also use two semiring combinations: a product semiring and a lexicographic semiring.

- **Log semiring** ($\mathcal{L}$) is given by

$$\mathcal{L} = \{\mathbb{R} \cup \{\infty\}, \oplus_{log}, +, \infty, 0\}, \tag{4.2}$$

  where $x \oplus_{log} y = -\log(e^{-x} + e^{-y})$. $\mathcal{L}$ semiring is used for example for representing LM weights in the graphs.

- A (minimal) **Tropical semiring** ($\mathcal{T}$) is given by:

$$\mathcal{T} = \{\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0\}, \tag{4.3}$$

  which means that computing the sum of paths in a graph in this semiring locates the minimum path; this is useful for Viterbi approximation.

- A **product semiring** is a combination of two semirings $\mathcal{A} = \{\mathbb{A}, \oplus_{\mathbb{A}}, \otimes_{\mathbb{A}}, \overline{0}_{\mathbb{A}}, \overline{1}_{\mathbb{A}}\}$ and $\mathcal{B} = \{\mathbb{B}, \oplus_{\mathbb{B}}, \otimes_{\mathbb{B}}, \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{B}}\}$ defined by

$$\mathcal{A} \times \mathcal{B} = \{\mathbb{A} \times \mathbb{B}, \oplus_{\times}, \otimes_{\times}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}}\}, \tag{4.4}$$

  where $\oplus_{\times}$ and $\otimes_{\times}$ are component-wise operators, i.e. $(a_1, b_1) \oplus_{\times} (a_2, b_2) = (a_1 \oplus_{\mathbb{A}} a_2, b_1 \oplus_{\mathbb{B}} b_2)$ and $(a_1, b_1) \otimes_{\times} (a_2, b_2) = (a_1 \otimes_{\mathbb{A}} a_2, b_1 \otimes_{\mathbb{B}} b_2)$.

  Operator $\times$ for creating a product semiring is associative, and so the product of more than two semirings can be defined recursively.

- A **lexicographic semiring** is given by:

$$\mathcal{A} * \mathcal{B} = \{\mathbb{A} \times \mathbb{B}, \oplus_{*}, \otimes_{*}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}}\}, \tag{4.5}$$

  where $\otimes_{*}$ is again a component-wise multiplication operator and $\oplus_{*}$ is a lexicographic priority operator which gives order priority to the elements of the semiring $\mathcal{A}$, and only operates on the elements of semiring $\mathcal{B}$ in case the elements of $\mathcal{A}$ are equal:

$$(a_1, b_1) \oplus_{*} (a_2, b_2) = \begin{cases} (a_1, b_1 \oplus_{\mathbb{B}} b_2) & \text{if } a_1 = a_2 \\ (a_1, b_1) & \text{if } a_1 = (a_1 \oplus_{\mathbb{A}} a_2) \neq a_2 \\ (a_2, b_2) & \text{if } a_1 \neq (a_1 \oplus_{\mathbb{A}} a_2) = a_2 \end{cases} \tag{4.6}$$

  Operator $*$ for creating the lexicographic semiring is also associative, and so the lexicographic semiring of more than two semirings can be defined recursively.

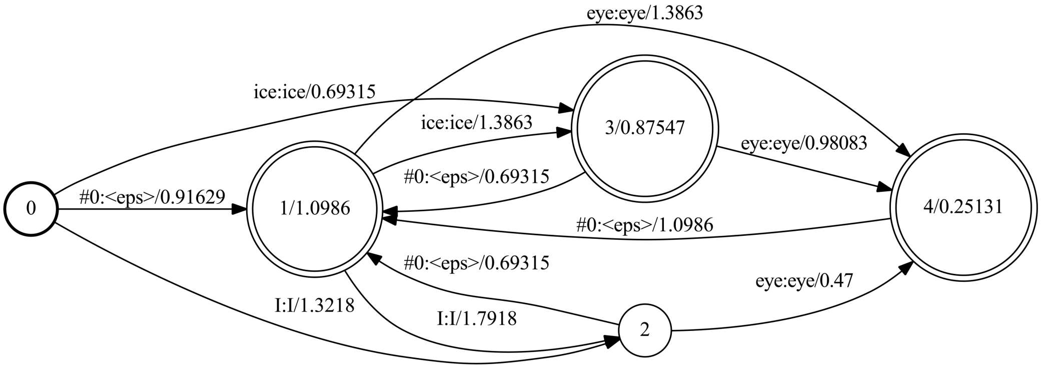


Figure 4.1: Example of a WFST representing LM (grammar transducer G). The weights on the arcs are negative log-probabilities.

- A **weighted finite-state transducer (WFST)** (see Figure 4.1 for an example) is a finite automaton that defines a set of possible paths that go from the initial state (a circle labeled 0 in Figure 4.1) to any of the final states (double circled in Figure 4.1) through any of the allowed transitions (arrows labeled with *input*:*output*/*weight* in Figure 4.1).

  Formally, given a semiring $\mathcal{K}$, a WFST T can be defined as an 8-tuple:

  $$T = (\Sigma, \Omega, Q, E, i, F, \lambda, \rho), \tag{4.7}$$

  where Σ and Ω are input and output alphabets respectively, Q is a finite set of states, E is a finite set of transitions $E \subseteq Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times \mathcal{K} \times Q$, an initial state $i \in Q$, a set of final states $F \subseteq Q$, λ is initial weight, and ρ is a final weight function. ϵ is the empty label, representing a lack of input or output.

  A successful path $\pi = t_1 \ldots t_n$ is a path from the initial state i to a final state $f \in F$, in which input labels of each consecutive transition t_n correspond to the consecutive symbols in the input string x. The output string will then be comprised of the output symbols on this path. Thus, a transducer can map a string of input symbols to a string of output symbols if it contains a valid path that follows the symbols in the input string.

  If a transducer is weighted, the weight of a path π is calculated as follows:

  $$w[\pi] = \lambda \otimes w[t_1] \otimes \cdots \otimes w[t_n] \otimes \rho(n[t_n]). \tag{4.8}$$

  The weight associated to the input sequence x is then the $\oplus - sum$ of the weights of all the successful paths π that take x as an input.

- A weighted transducer with input symbols only is called a **weighted finite-state acceptor (WFSA)**. In OpenFST[1] framework, an acceptor is represented by a WFST with equal input and output labels. A string of symbols is said to be accepted (thus the name "acceptor") by a WFSA if the acceptor contains a valid path on which input labels of the transitions correspond to the characters in the input string x.

[1] https://www.openfst.org/twiki/bin/view/FST/WebHome

Numerous operations are defined on WFSTs, the ones we will mostly use further are the operations of composition and union of two WFSTs, and the operations of minimization, determinization, and closure on a single WFST.

- The **union** operation ($\cup$) can be seen as a sum of two WFSTs. If WFST A maps string x to y with weight a and WFST B transduces string w to v with weight b, then their union transduces x to y with weight a and w to v with weight b. Thus, the union of two transducer is a transducer that contains paths from both.

- The **composition** operation ($\circ$) is used to combine different levels of a recognition system model. If WFST A maps string x to y with weight a and WFST B maps y to z with weight b, then their composition maps string x to z with weight $a \otimes b$.

- The **determinization** operation ($det()$) creates an equivalent (associates the same output sequence and weights to each input sequence) deterministic (each state has at most one transition with any given input label and there are no input ϵ labels) WFST from a nondeterministic one. The benefit of a deterministic WFST over an equivalent non-deterministic one is its irredundancy: it contains at most one path matching any given input sequence, thereby reducing the time and space needed to process an input sequence.

  Not every WFST can be determinized, so in the framework of WFST-based ASR, decoding WFSTs are made non-cyclic with the help of disambiguation symbols before determinizatoin operation.

- The **minimization** operation ($min()$) creates an equivalent WFST that has the least number of states and the least number of transitions among all equivalent deterministic WFSTs. Minimization operation can always be performed on a deterministic WFST, so determinization is a required step before applying minimization.

- The **closure** operation (*) creates a loop from the final states of an WFST to its initial state.

Kaldi[2] framework [Povey et al., 2011] uses the aforementioned OpenFST library for handling FSTs. In Kaldi setup, the ASR decoding graph is given by a multi-step composition which combines different parts of the recognition system into a large HMM. This large HMM, on which decoding is performed, represents search space constraints on each of the levels in the system. These individual levels include:

- G is a WFSA that encodes the LM; its input and output symbols are word labels and weights that represent LM probabilities. G graph accepts only word sequences that are valid in a current LM.

- L is a WFST representing the lexicon; its output symbols are words and its input symbols are phones, so it transduces valid pronunciation sequences into word labels that they represent.

- C represents the context-dependency; its output symbols are phones and its input symbols represent context-dependent phones, the more detailed acoustic units, which are represented by AM; C graph can be omitted if a system uses phones or even larger entities as AM units.

[2] https://kaldi-asr.org/

- H contains the HMM definitions of acoustic units; typically, each context-dependent phoneme is modeled using a 3-state HMM, and H is the closure of the union of the individual HMMs; in H FST, the output symbols represent context-dependent phones, and its input symbols encode information about states and transitions probabilities of the HMMs.

To help visualize all the aforementioned WFSTs, a simple example is illustrated in Appendix B.

The composition of all the levels of the decoding graph is then constructed the following way:

$$HCLG = min(det(H \circ det(C \circ det(L \circ G)))) \tag{4.9}$$

Note that minimization and determinization operations are performed with auxiliary symbols (ϵ and disambiguation symbols[3]) to avoid loops and non-determinism. After the composition auxiliary symbols are removed.

The resulting $HCLG$ graph is called the decoding graph, and during decoding, it limits the search space to valid sequences of units on every level. Thus, a path through the decoding graph encodes a mapping from a string of input symbols (transition-ids, which encode pdf-ids and other information[4]) to a string of output symbols (words). The weights on the links constituting a path in the $HCLG$ graph are combined probabilities from the LM and HMM transition probabilities. These weights are often referred to as "costs", where a cost is a floating point number that typically represents a negated log-probability. Acoustic model likelihoods (HMM emissions) are evaluated during decoding using the probability density functions (PDFs) corresponding to $HCLG$'s input symbols.

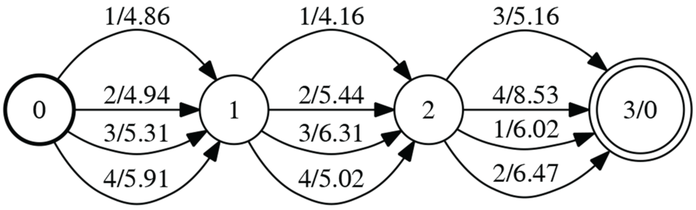


Figure 4.2: Acceptor U for 3 frames and 4 HMM states from [Povey et al., 2012].

A WFST interpretation of the decoding problem is then defined as following, according to [Povey et al., 2012]. Imagine we want to "decode" an utterance of X frames, i.e. we want to find the most likely word sequence, and the corresponding alignment. We construct a WFSA called U, which has $X + 1$ consecutive states representing boundaries between frames. Every state is connected with the next state with arcs for every context-dependent HMM state (see Figure 4.2). The costs on these arcs are the negated and scaled acoustic log-likelihoods. Then the search graph for the utterance can be constructed as following:

$$S = U \circ HCLG \tag{4.10}$$

The decoding problem is equivalent to finding the best path or a number of best paths through S. Practically, the search is constrained by beam-pruning.

[3] https://kaldi-asr.org/doc/graph.html#graph_disambig

[4] https://kaldi-asr.org/doc/hmm.html#transition_model_identifiers

The feature sequence is the input to the decoding with the decoding graph and a string (or several best-fitting strings) of words is the output. Alternatively, we can consider as an output a lattice (labeled, weighted, directed acyclic graph) of all possible decoding paths. For our OOV retrieval, we will use the full lattice output of the decoding, defined as the following: Let B be a pruned subset of states and arcs of S obtained with a pruning beam α. Then a lattice L must have the following characteristics:

- Lattice L contains a path for every word sequence within α of the best-scoring one and does not contain duplicate paths with the same word sequence.
- For every path in L, the score and alignment corresponds to the best-scoring path in B for the corresponding word sequence.

4.2 OOV Detection and Recovery Procedure

For successful OOV detection and recovery in a component-based ASR system that uses WFST-based decoding, several steps have to be made. First, the $HCLG$ graph has to be adapted for hybrid word and subword decoding. This hybrid decoding graph will generate decoded lattices that contain paths with subword units that will inform OOV post-processing. This framework also requires efficient sublattice retrieval from lattices, and clustering of lattices. All these methods are presented in this section.

For the ease of navigation, here we give the general algorithm for recurrent OOV detection and recovery that is followed in this chapter. Each step of this algorithm will be described in more detail in the following sections.

Algorithm 1 WFST-Based OOV Detection and Recovery

```
1: train an ASR system (section 4.2.1)
2: create a modified hybrid decoding graph (section 4.2.2)
3: decode all the utterances with the hybrid decoding graph to create hybrid decoded
   lattices (section 4.2.3)
4: create a TFT search index for all the decoded utterances (section 4.2.4)
5: extract OOV candidates in form of phoneme sub-lattices from the index (section 4.2.5)
6: cluster OOV candidates (section 4.2.6)
7: incorporate discovered recurrent OOVs into the LM and the dictionary (section 4.2.7)
```

My implementation of the full OOV recovery procedure can be found on Github[5].

4.2.1 Baseline ASR

The system is trained using Kaldi toolkit [Povey et al., 2011], following the official Librispeech recipe[6] provided within the toolkit. The OOVs from the OOV list (see Appendix A) are excluded from the training. train-clean-100 dataset is used for the ASR training, and train-clean-360 for OOV extraction (see Table 3.1). The LM provided with Librispeech is used as word-level LM, and the phonotactic language model is a 3-gram ARPA LM trained on the whole Librispeech dictionary (200 000 words).

[5] https://github.com/BUTSpeechFIT/OOV-recovery-in-hybrid-ASR-system

[6] https://github.com/kaldi-asr/kaldi/tree/master/egs/librispeech

The features used in the training are Mel-frequency cepstral coefficients (MFCC) [Davis and Mermelstein, 1980] with 13 cepstra, 23 Mel-bins, extracted from the frames of length 25 ms with 10 ms overlap. Energy is computed before preemphasis and windowing, and dc-offset is removed. The raw MFCC features have 13 coefficients for every frame. After feature extraction, cepstral mean and variance normalization (CMVN) is applied for every speaker.

Pure MFCCs are only used for initial monophone training, later the features are expanded to include Δ and $\Delta\Delta$ features. Unlike the instantaneous MFCC features, Δ and $\Delta\Delta$ approximate first and second derivative of the features. With the addition of deltas, features grow to size $3 \times 13 = 39$; Kaldi adds deltas in an online fashion via convolving the features with a sliding window.

Another way the features are modified in the later stages of training is through the application of linear discriminant analysis (LDA)+maximum likelihood linear transform (MLLT) transforms; LDA+MLLT transformation is computed on MFCCs the following way: we splice across 9 frames (4 left context and 4 right context), reduce the dimension to 40 using LDA, and then later estimate, over multiple iterations, a diagonalizing transform known as MLLT or (global) semi-tied covariance (STC) [Gales, 1999]. LDA+MLLT features have dimension of 40 for every frame.

For DNN training, features transformed with feature space maximum likelihood linear regression (fMLLR) are used (dimensionality 40); fMLLR transforms LDA+MLLT features to speaker adapted features using an affine transform.

The sequence of systems in Kaldi baseline (nnet2 recipe) is:

1. A simple initial hidden Markov model (HMM)/Gaussian mixture model (GMM) monophone system is trained on the subset of 2000 short utterances from train-clean-100; each phoneme is represented by a three-state HMM; 1000 total Gaussians are used.

2. A triphone system with MFCC + Δ + $\Delta\Delta$ features is trained on a subset of 5000 utterances from train-clean-100 and based on alignments from the monophone system; maximum number of Gaussians is 10000.

3. A triphone system with LDA+MLLT; realignment is done every 10 iterations, and MLLT is recalculated 4 times during the training.

4. A triphone system with LDA+MLLT+speaker adaptive training (SAT); SAT performs speaker and noise normalization by adapting to each specific speaker with a particular data transform.

5. A DNNs on top of the fMLLR features, using the decision tree and state alignments from the LDA+MLLT+SAT system as supervision for training. The DNN is a p-norm network [Zhang et al., 2014] using 4 hidden layers. Input dimensions are 360 (40×9), as the network sees a window of fMLLR features, with 4 frames on each side of the central frame. As it is hard for this particular architecture to learn from correlated input, these 360-dimensional features are multiplied by a fixed transform that decorrelates the features. Output dimension is 3440 (context-dependent HMM states). The initial learning rate is 0.01, going down to 0.001 at the end of the training.

The WER1 that is possible to obtain with this setting is 11.7 % on the test-clean set. Note that it is more than the 9.32 % reported by [Panayotov et al., 2015] for training on train-clean-100, but our baseline is for a weaker LM (only 3-gram vs 4-gram - we were not able to use 4-gram in hybrid decoding graph due to size restrictions) and also already for the system with artificially chosen and excluded OOVs at 1.5 % OOV rate.

4.2.2 Hybrid Decoding Graph

The issue with OOVs in the $HCLG$ decoding framework is that although we can have OOVs represented by a special word in the LM, there is no dictionary entry that specifies what string of acoustic units represents each particular OOV. Phonetically, OOVs are often modeled with a special garbage acoustic model. While practical, such a solution does not provide good acoustic representation of an OOV, so in our hybrid system we model the likely phonetic sequences representing an OOV by building a hybrid decoding graph in the same way as in [Szőke, 2010].

While in a hybrid word-subword graph subwords can be chosen as any smaller-than-words units, such as syllables, BPEs, etc., in the following experiments we will use phonemes as subword units, as they are intuitive and linguistically defined, and also correspond to the representation of words in a dictionary.

To obtain a hybrid word-phoneme decoding graph, we first build two G WFSAs: G_w is built from a word level LM, and G_p from a phonotactic LM trained on the dictionary. As the word-level LM contains probabilities for OOVs, G_w also contains arcs labeled <unk> which is the OOV label in the LM. The weight on the <unk> labeled arcs is the correct LM probability of an OOV in current context. We then modify G_w by replacing every node to which an <unk> labeled arc leads with a copy of the G_p graph. There are such nodes in the G_w graph for every LM history of the OOV word, and thus the resulting modified graph contains as many copies of G_p. Figure 2.2 illustrates such modification, but note that it is a toy bigram example, it only has one occurrence of an OOV, and thus the G_p is shown to be inserted only once.

Technically, this G graph modification is done by the composition of G_w with the closure of G_p modified to add a word loop to the initial state:

$$G = G_w \circ (mG_p)^*, \tag{4.11}$$

where mG_p is the modified G_p with a word loop over the initial state.

When this is done, every <unk> link in the decoding graph leads to a phoneme sub-FST, and there is a special label for the end of subword sequence on all the links going from phoneme sub-FST back to word transducer.

In the hybrid graph, we can manually control the preference that the system shows towards paths containing phonemes. This is achieved by boosting the probability of following the OOV link and by increasing or decreasing probabilities inside of the phoneme sub-graph to encourage choosing paths through OOV. In particular, three parameters control decoding in the hybrid graph and can be adjusted for better performance:

- OOV cost (OOVC) penalizes the hypothesis for entering phoneme sub-graph. It only affects the weight of the phoneme sub-graph entry link in the following way:

$$w'[t_{OOV}] = w[t_{OOV}] + OOVC. \tag{4.12}$$

  As the weights in G are represented as negative log probabilities, the bigger the OOVC, the less OOVs are generated.

- Phoneme LM scaling factor (PLMSF) balances phonotactic and word model scores. It affects every link cost inside a phoneme sub-graph in the following way:

$$w'[t_{phoneme}] = w[t_{phoneme}] \times PLMSF. \tag{4.13}$$

- Phoneme insertion penalty (PIP) regulates the length of the generated phoneme strings. It is a number added to weights on every link inside the phoneme sub-graph:

$$w''[t_{phoneme}] = w'[t_{phoneme}] + PIP. \tag{4.14}$$

My implementation of hybrid decoding graph creation in a Kaldi framework can be found at Github[7].

4.2.3 Decoding with a Hybrid Decoding Graph

This hybrid G graph is then composed with L, C, and H graphs normally as described in Section 4.1 and the decoding procedure is not changed from (4.10). Decoding with a hybrid decoding graph $HCLG$ built in this way gives us hybrid decoded lattices containing both paths with words and paths with phonemes representing OOVs. The decision whether the subword region of the decoding graph comes into play or not depends on the comparative costs of the paths through the graph. Ideally, the cost of the paths going through subword region would only be lower than any word-only decoding path if and only if there is an OOV at this position in an utterance.

Extracting OOV candidates from a lattice resulting from decoding with a hybrid graph allows an OOV to be correctly modeled with the combination of the probability of the OOV being in a certain place in the utterance and the conditional probability (given that the word is OOV) that the OOV is represented by a particular pronunciation.

4.2.4 Lattice Indexing with TFT

Working with full lattices instead of one best decoding output is beneficial for the OOV recovery task. First, this approach may avoid many errors stemming from a possible low quality ASR output. Second, it allows potential OOVs to be represented by several possible paths, and also to find OOVs on suboptimal word paths. However, while being more robust, this approach poses some challenges, as it can be time and space consuming to traverse the output word-phoneme lattices forward and backward to ensure that all phoneme paths are found. Thus, an effective way of presenting the information from the lattices is needed. We can speed up the process of OOV extraction if we first apply indexing to decoded lattices.

The inverted index is a search optimization technique that is mostly used in the KWS task where it provides correspondences between words and times where they can be found in an utterance. A reverse index can be in many formats (e.g. lookup table), but here we will consider only one of them: a WFST. This WFST index is a tree-like structure which can take as an input any string that is found in the lattice, and outputs utterance labels, time frames and scores for the input query. During the search, the query is also represented as a WFST and the composition operation of the query with the index WFST is performed [Lee et al., 2015].

[7] https://github.com/BUTSpeechFIT/ASR-hybrid-decoding

For our needs of providing fast search for the timing of OOVs, we used lattice indexing introduced in [Can and Saraclar, 2011]. The indexing procedure transforms output lattices into a single factor transducer which stores the timing information and scores on the output labels. The output symbols quadruple consists of utterance ID, start time, end time and posterior probability. This newly introduced type of factor transducer is called a timed factor transducer (TFT). The whole procedure is described in the source paper in great detail, so here we only give the general understanding of it that is relevant for the efficient OOV extraction task.

The input of the procedure of TFT construction is a word/phone lattice output by an ASR system. In these lattices, path weights correspond to a weighted combination of acoustic and language model probabilities. Factor generation requires some preprocessing of these lattices. First, we should perform a weight-pushing algorithm in the log semiring $\mathcal{L}$ to make the output lattices stochastic WFSTs, meaning that the weights leaving one state correspond to probabilities that sum up to 1. Then, the arcs with the same input label and overlapping time spans are clustered together: the new cluster labels are introduced and inserted as an output label of each arc.

The left side of Figure 4.3 shows lattices for two utterances after pre-processing. Arcs are labeled with word or phoneme labels (a and b) as input symbols, cluster identifiers (1 and 2) as output symbols, and stochastic probabilities obtained from weight-pushing as weights.

After preprocessing is finished, TFT can be generated. On each arc, the weight (in $\mathcal{L}$) is replaced with a product semiring, consisting of weight (in $\mathcal{L}$), start time (in min tropical semiring $\mathcal{T}$) and end time (in max tropical semiring $\mathcal{T}'$). Moreover, initial and final states are generated as well as arcs that connect them to each state of the FST. After that, the paths with the same factor-pair (i.e. input (word or phoneme label) and output symbols (cluster label)) are merged and the posterior weight becomes the sum of all probabilities of all successful paths that contain that factor pair. The transducer is then minimized and determinized in $\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$, and we map the weight to the full tropical lexicographic semiring $\mathcal{T} * \mathcal{T} * \mathcal{T}$. After this, cluster labels are removed on non-final arcs; on final arcs we insert disambiguation symbols, because we want to preserve each non-overlapping word occurrence separately. The resulting disambiguated TFT can now be determinized and minimized in $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring.

In the end, union operation of the transducers corresponding to indices for individual utterance is performed to obtain one big index for the whole dataset. Each utterances' input-output labels are encoded as a single label and utterance identifiers are put as outputs.

The right side of Figure 4.3 shows the result of building an index on the example lattices. The output labels on the final arcs of each valid path point to the utterance label where the word string on this path can be found. The weight triple shows word posteriors, start time and end time.

Search over TFT can be performed by simply composing the query (word FSTs with word labels on input and output) with the index, and the weights are sorted in a natural $\mathcal{T} * \mathcal{T} * \mathcal{T}$ order.

Practically, this indexing procedure is implemented[8] in the Kaldi toolkit and can be used as it is on the lattices generated by the hybrid decoding graph. However, OpenFst does not have the $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring implemented, so an extension of OpenFST had to be

[8] https://kaldi-asr.org/doc/kws.html

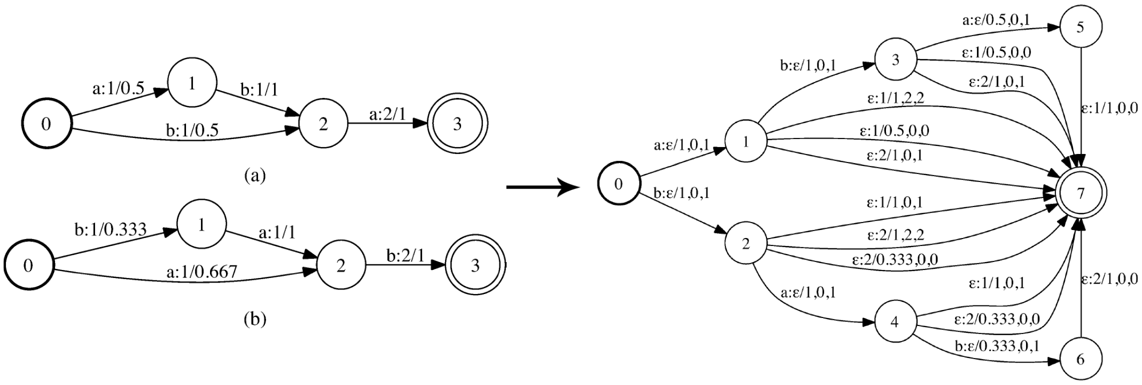


Figure 4.3: An example of creating a TFT index from [Can and Saraclar, 2011]. Note that the weights here are stochastic probabilities instead of negative logs.

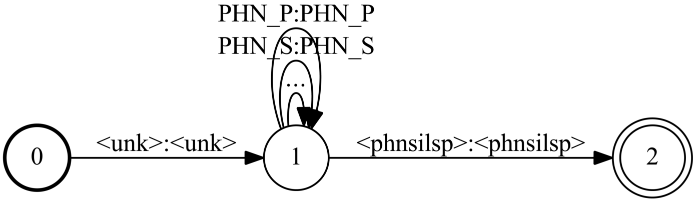


Figure 4.4: Phoneme acceptor P.

written. My implementation of the $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring is included in the Github repository[9] with my OOV recovery procedure.

4.2.5 OOV Extraction from Lattice Index

The output of this indexing procedure on decoded lattices is a tree-like WFST which contains all partial paths through the decoded lattice, of which we care only about phoneme ones: we have to extract OOV candidates in form of phoneme sub-lattices from our decoded lattices, i.e. separate sub-lattices of phonemes from the rest of the lattices containing also words (that will not be used in further information extraction). We will search for sub-lattices starting with the <unk> symbol that marks the entry point of phoneme lattice and ending with <phnsilsp> output symbol marking the exit point.

Practically, to find only paths consisting of phonemes in the index tree I, we need to compose an 3-state unweighted acceptor P (see Figure 4.4) to the index tree. This acceptor P has 3 states, the arc between states 0 and 1 is accepting input symbol <unk>, the arc between states 1 and 2 is accepting input symbol <phnsilsp>, state 2 is the final state, and state 1 has a phone loop with all the possible phonemes. Thus, P only accepts strings of phonemes from the index tree that start with the <unk> symbol (phoneme subgraph entry symbol) and end with <phnsilsp> (phoneme subgraph exit symbol):

$$L_{OOV} = P \circ I. \tag{4.15}$$

A tree-like structure is fast to traverse, and after only the relevant phoneme paths are left, separate OOV candidates are extracted based on non-overlapping start and end times

[9] https://github.com/BUTSpeechFIT/OOV-recovery-in-hybrid-ASR-system

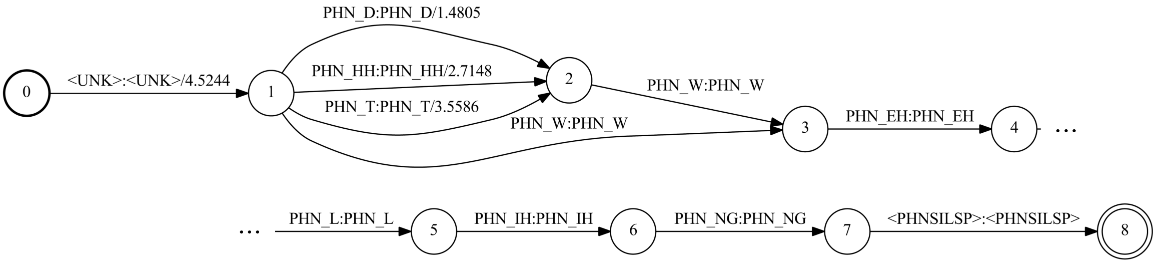


Figure 4.5: OOV candidate in lattice form: $min(L_{OOV})$.

that are stored in the TFT weights. The minimization operation returns the extracted candidates to the structure they had in the decoded lattices (see Fig. 4.5).

After they are extracted from the index, OOV candidates are represented as probabilistic phoneme lattices that now contain both the decoding graph probabilities and acoustic scores. The weight of every path of an OOV candidate lattice can be transformed to the posterior probability via weight-pushing in log semiring [Mohri et al., 2008]. We discover lattices that represent the same OOV with the help of clustering.

4.2.6 OOV Candidates Clustering

As OOV candidates are represented probabilistically in the form of lattices, we need a technique for discovering the best pronunciation of recurrent OOVs among all the pronunciation variants. Clustering together OOV representations based on similar pronunciations in some of the paths will reinforce the repeating paths and provide better reference pronunciation.

The similarity of two FSTs can be estimated by performing their composition and calculating the shortest distance from the initial state to a final state in this composition in log semiring. We call this distance Cscore and it can be interpreted as the probability of both OOV candidates being present in the recording and both being pronounced as the same sequence of phonemes. If two FSTs do not have a single common path, the composition output is empty and Cscore equals to 0.

The first type of clustering we tried was hierarchical clustering. In the beginning, pairwise Cscores of all the OOV candidates are put into a matrix. At each step of the clustering, the system looks for the biggest Cscore and, if it is bigger than a pre-selected threshold, performs the union of the two corresponding OOV candidates. The newly united lattice preserves paths from all of the initial lattices that were merged into it at different steps. At the end of a clustering step, pairwise Cscores involving the two merged candidates are recalculated by performing the composition of the newly merged candidates with all other candidates. The pre-selected threshold defines a stopping criterion for the clustering – if the Cscore is smaller than the threshold, the clustering is stopped, as we might not want to merge such unlike candidates. After the clustering is stopped, each OOV cluster is represented by the union of all the phoneme paths in all the occurrences that were clustered in it.

To evaluate the hierarchical clustering of OOV hypotheses and to estimate the stopping point, we have looked at the clustering quality with adjusted Rand index (ARI) [Santos and Embrechts, 2009], a standard way of analyzing relations between two clusterings based on [Rand, 1971]. Given two partitions into subsets X and Y over a set of elements n, Rand index (RI) is calculated as a number of agreements over the total number of pairs:

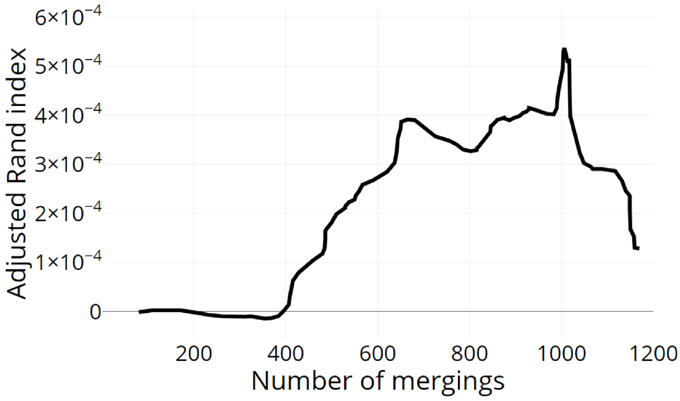


Figure 4.6: An illustration of the progress of ARI score during hierarchical clustering of OOV hypotheses.

$$RI = \frac{a+b}{\binom{n}{2}}, \tag{4.16}$$

where a is the number of pairs of elements that are in the same subset in X and in the same subset in Y, and b is the number of pairs of elements that are in different subsets in X and in different subsets in Y. ARI is the corrected-for-chance version of the RI and can take negative values if the index is less than the expected index.

At each step of the clustering procedure, we compared the current hypothesis clustering with the true clustering given by the word labels obtained from alignment with a full dictionary. While ARI is small due to the huge number of initial clusters, it keeps growing while the Cscore is big enough to create mergings that make sense and falls once the system starts over-merging clusters (see Figure 4.6). The steeper increases in ARI occur when bigger clusters are merged in a meaningful way. The maximum of ARI can be used to choose stopping criterion.

Hierarchical clustering, although adequate, has several problems. First, the threshold must be chosen experimentally, and may vary for different number of OOV candidates. And second, once merged, a candidate can never leave the cluster. This sometimes led to appearance of big "garbage" clusters if no pruning is introduced. In the next chapter experiments, a more principled clustering approach was adopted that avoids these problems.

4.2.7 Final Recovery Steps

After the clustering, the system ignores the clusters with less then two candidates assigned to them - if there is no recurrency of the patterns and no reinforcing of the most common pronunciation paths, it is difficult to assess the quality of the cluster. From the clusters of size ≥ 2, one best path is extracted - this path has appeared in the minimum of two OOV candidates with good costs and is thus the best bet. Then this path is turned into a grapheme form to obtain the new word with the help of a P2G model.

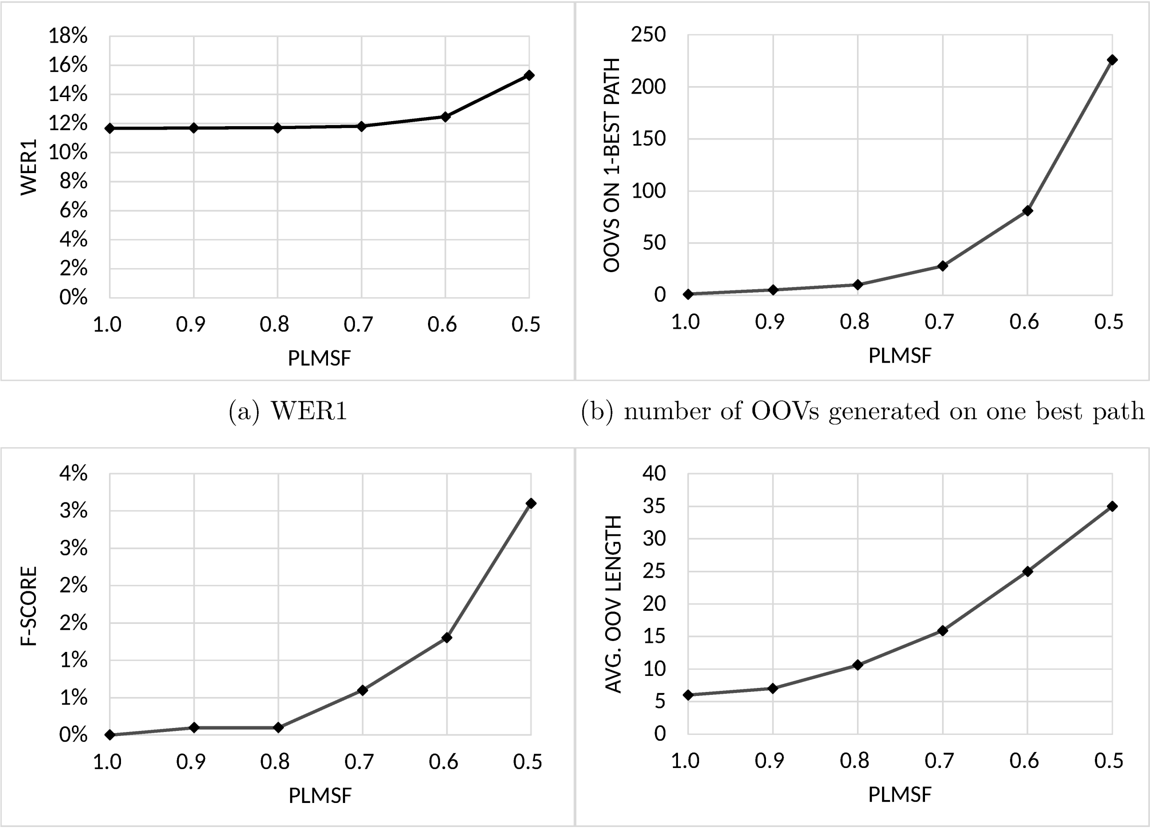
The P2G model is trained on the reverse dictionary following [Bisani and Ney, 2008]. The dictionary for the training is the default Librispeech dictionary of 200 000 words with excluded 1000 OOVs from the list. This model can be applied to the best path or to the whole lattice of a candidate to propose new entries to the dictionary.

4.3 Results

4.3.1 OOV Detection

The baseline system described above achieves 11.7 % WER and zero f-score (as proposed in Section 1.1.2) if estimated on one-best output on test-clean set. This means it is not capable of detecting OOV candidates at all.

In a hybrid system, hybrid parameters described in section 4.2.2 can be changed to trade-off between detecting more OOV candidates and retaining a good WER. As an example, Figure 4.7 shows the sensitivity of OOV detection depending on PLMSF. It can be seen that the more we lower the cost of paths inside the phoneme subgraph and thus encourage the system to choose paths going through the OOV region, the more OOV candidates we get and the better our f-score, mainly due to recall. However, WER is suffering from over-producing OOVs on the best path, so one should be careful not to over-encourage paths with OOVs at the cost of compromising the best path.



(a) WER1

(b) number of OOVs generated on one best path

(c) f-score

(d) average length of OOVs in phonemes

Figure 4.7: Different ASR system characteristics depending on PLMSF.

Tuning of the three hybrid parameters was done on the test-clean set (reference number of OOVs is 1070) to speed up the search process, but the results should be indicative of the performance on train-clean-360 that we will use for OOV extraction. All three of the hybrid parameters are tuned separately. It can be seen that without encouraging the system to enter subword graph and produce phonemes, detection is very poor; however, if we lower the weight on phoneme paths too dramatically, WER grows, and the average length of

hybrid parameters: PIP, PLMSF, OOVC				WER %	f-score %	# unks	phn duration	missed OOVs
baseline				11.59	0	0	0	1070
default	0	1	0	11.66	0	1	6	1070
	0	1	-2	11.66	0	1	6	1070
	0	1	-4	11.67	0.2	8	6.5	1067
	0	1	-6	11.68	0.4	22	6	1064
	-0.5	1	0	11.69	0.1	8	11.4	1068
	-1	1	0	11.96	1	50	19.18	1048
	-1.5	1	0	15.12	3.2	239	32.9	968
	0	0.9	0	11.68	0.1	5	7	1068
	0	0.8	0	11.70	0.1	10	10.6	1067
	0	0.7	0	11.80	0.6	28	15.9	1058

Table 4.1: Finding good hybrid graph parameters on test-clean dataset. Column "# unks" shows the number of OOVs predicted on the best decoding path; column "phn duration" shows the average duration of predicted OOVs in phonemes; column "missed OOVs" shows how many OOVs from the reference 1070 in the dataset were not detected.

generated OOVs becomes too long compared to normal word length. These experiments confirm the results in [Szőke, 2010] and allow us to choose the best setting for OOV recovery task.

The maximum f-score we were able to reach on the train-clean-360 dataset was 7.2 % but this setup was not viable due to the average length of phoneme stings of 25 – much longer than a word should be. The following recovery experiments are performed on a hybrid system with OOVC $= -10$, PLMSF $= 0.8$ and PIP $= 0$. This system provides f-score of 1 % at 11.77 % WER on the test-clean set - a reasonable balance between the two. When evaluating the OOV detection performance on full lattices in comparison to a one-best output, benefits of the full lattice approach can be observed. On 360 hours of data, extracting OOV candidates as phoneme strings from one-best decoding output results in just 1247 OOV candidates, while from the full lattices, we get 15991 (see Table 4.2), which is more than 12 times more. However, considering the reference number of OOVs in the 360 hours dataset is 60661, recall is still very poor: even if we assume 100 % precision, maximum recall is only 26 %.

4.3.2 Recurrent OOV Recovery

Below are the OOVs that are obtained from the clustering with Cscore threshold 0.01 (as defined in 4.2.6) of phoneme strings from one-best decoding output results. The corresponding graphemic representation is obtained from a P2G system trained on the same dictionary as the phonotactic LM [Bisani and Ney, 2008] but reversed, so that the graphone units are phonemes on the input and graphemes on the output. Only the OOVs that are a result of clustering of more than 2 candidates are considered, which produces the following 7 recovered words:

COURAGE	K ER IH JH
VOYAGE	V OY IH JH
FLANE	F L EY N
KLEY'S	K L IY Z
SALOOSKI	S AH L UW S K IY
IMETHEUS	IH M IY TH IY AH S
ANCTIOUSLY	AE NG K SH AH S L IY

As can be seen, less than half of the words make sense. Only two of the OOVs out of 1000 are recovered correctly and one ("ANCTIOUSLY") is close enough to be recognizable. This gives us a recovery recall of 0.3 % and recovery precision of 43 % if it even makes sense to calculate precision on just 7 outputs.

To compare, below are the 20 OOVs obtained with clustering candidates from full decoding lattices with 0.01 threshold. In the brackets is the number of the word's reference occurrences where applicable. Again, only OOVs that are the result of merging more than 2 candidates are considered:

COURAGE (288)	K ER IH JH
VOYAGE (120)	V OY IH JH
THRACE	TH R EY S
THRONG (48)	TH R AO NG
SNESS	S N AH S
UNESE	AH N IY Z
ATHO'S	AE TH OW Z
HITHER (95)	HH IH DH ER
IGARLY (176)	IH G ER L IY
SAVAGE (182)	S AE V IH JH
WELLING (82)	W EH L IH NG
ANXIOUS (296)	AE NG K SH AH S
BOLDLY (68)	B OW L D L IY
TRICHERY (43)	T R IH CH ER IY
DIGNITY (190)	D IH G N AH T IY
ERLOGINGS	ER L AA JH IH NG Z
ERNESSNESS	ER N AH S N AH S
ANCTIOUSLY (99)	AE NG K SH AH S L IY
CORMALIS	K AO R M AE L AH S
HITHERINTHITHER	HH IH DH ER IH N TH IH DH ER

Of these 20, 8 are ideally recovered words from the 1000 on the OOV list, which is four times as many as with one-best approach. Furthermore, there are some close-to-ideal recoveries, like a name from *The Three Musketeers* (Athos, recovered as "ATHO'S"), and 6 words that are still recognizable ("THRACE", "UNESE", "IGARLY", "TRICHERY", "ERNESSNESS", "ANCTIOUSLY"), although the graphemic representation is not completely right. So the OOV recovery recall in full-lattice clustering equals 1.4 %, which is more than 4 times better than one-best clustering, even though still underwhelming. Recovery precision rate can be estimated at about 50 % depending on how the words recovered similar enough to the reference are judged. All the results are summarized in Table 4.2.

Of special interest are entries "SNESS" and "HITHERINTHITHER". The first is a suffix, which can help with the recognition of nouns that are derived from adjectives using

Detection method	Detection		Recovery		
	# of candidates	recall %	clusters	recall %	precision %
one best	1247	2	7	0.3	43
full lattice	15991	26	20	1.4	50

Table 4.2: Comparison of detection and recovery results using one best and full lattice approaches on train-clean-360 dataset.

this morpheme. The second is a phrase “hither and thither”, which repeats more often in the data than any of its parts separately. As this phrase has a distinct meaning and usage, it may be profitable to treat it as an individual lexical entity in a language model. We may also note that “anxious” and “anxiously” differ only in two phonemes in the end. This way, we can discover that it is a suffix and use this information for further analysis of discovered words.

Even though the recovery procedure provides limited amount of new words, they seem to be helpful additions to the system. Adding these newly-discovered OOVs to the dictionary with the learned pronunciation and to the LM as unigrams with the same probability as an OOV reduces WER from 11.77 % to 11.62 %.

4.4 Conclusion

It has been shown that the newly-proposed lattice-based approach outperforms one-best approaches both in terms of OOV detection and in terms of the recovery of phonetic and graphemic representations of OOVs. The proposed system shows promise of enhancing ASR user experience by bringing to their attention newly discovered words that may be added to the dictionary almost without adjustments.

The benefits of the system is that it performs both detection and recovery and for each OOV phonetic representation, it preserves the correct posterior scores that reflect both the probability of the OOV being in the utterance in the given context and the probability of the OOV being realized as this specific phoneme string. Moreover, words represented as lattices reflect pronunciation variants well and this representation also helps to cluster them into good OOV representations in spite of minor errors or variations. As there is no jumping between two granularities, OOV boundaries are well defined and there is no risk of “stealing” a phoneme from a neighbouring word.

Since the first publication of this work, the approach described here has been taken on and furthered by [Zhang et al., 2020]. They used the same hybrid graph creation with the same hybrid parameters and decoding as we did, and, having obtained OOV candidates by hybrid decoding, performed a second pass decoding by applying word-level RNNLM rescoring. They showed that by calibrating OOV candidates’ language model (LM) scores, both OOV recovery and overall decoding performance can be significantly improved.

There are several drawbacks discovered in the system however:

1. Hybrid lattices can get unmanageably big in case of more-than-trigram models, and not fit in memory. Moreover, decoding takes a long time as the beam has to be wider to keep all the optimal variants including the phoneme paths.

2. Hybrid parameters need tuning to achieve both good f-score rates and reasonable WER.

3. Even if the hybrid parameters are set perfectly, the detection recall is comparatively poor (26 %), which is reflected in low recovery recall (1.4 %), as clustering does not get enough examples. These suboptimal results are explainable by the fact that in case of several words with similar pronunciations, the decoding is prone to choose them rather than to explore phoneme paths.

Chapter 5

OOV Detection in an E2E System

The work presented in this chapter attempts to improve the detection recall issues encountered in the hybrid setup by employing E2E techniques. For OOV detection, we will now rely on an E2E ASR system trained to predict directly word sequences. The input to the E2E system are acoustic features, and as output, we get predicted word labels, and also access to internal hidden representations of words. These word labels and other information from the E2E systems serve to obtain timestamps for detected OOVs. Then, a procedure similar to the one in the previous chapter is used: OOVs extracted from the timestamps in the forms of lattices are clustered according to their phonetic similarities, and recovered OOVs are added to the dictionary.

The core of the OOV detection research described in this chapter was published in [Egorova et al., 2021]. In the thesis, we describe the Chinese restaurant process (CRP) clustering procedure in more detail and provide more results on different output label counts.

5.1 Attention-based E2E ASR System

The baseline E2E ASR system used in this work is a Listen Attend and Spell architecture (LAS) model [Chan et al., 2016]. This is a variation of Attention-based [Chorowski et al., 2015a] encoder-decoder architecture that has enough capacity to learn complex patterns and enough explainability to use some of the inner representations for OOV tasks.

5.1.1 LAS System Architecture

Feature extraction follows the standard Kaldi filterbank extraction with 80 Mel-bins for higher resolution. For each frame of 25 ms with 10 ms overlap, feature vector is of size 83: 80 filterbanks, plus 3-dimensional pitch features. The full input sequence matrix $\mathbf{X}$ has dimensions of the number of 25 ms frames in the utterance times 83.

The input feature sequence $\mathbf{X}$ is transformed into the encoded hidden representation $\mathbf{H}$ by the **encoder**. Let X be the length of utterance in frames, F the dimension of input features, and H the size of the hidden representation, then the pipeline of encoder transformations with all the input-output dimensions can be seen in Figure 5.1. The encoder consists of six layers, each containing a bi-directional long short-term memory (biLSTM) layer followed by a linear projection layer. The input sequence is sub-sampled in the time dimension by a factor of 2 in the first two encoder layers; the sub-sampling is done by max-pooling with the kernel size 3 and stride 2. Non-subsampling layers have residual

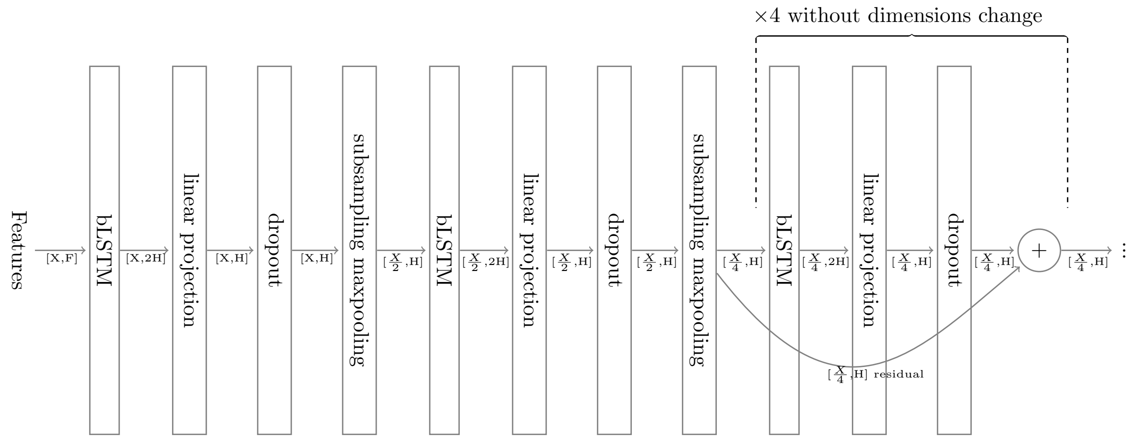


Figure 5.1: LAS encoder architecture; X is the length of utterance in frames, F is the dimension of input features, and H is the size of the hidden representation.

connections, and dropout is applied to the outputs of the biLSTM networks. Dropout probability is 0.1 in the subsampling layers and 0.3 in other layers. The output of the encoder step is hidden representation $\mathbf{H}$ of the size of H times $^X/_4$.

The task of the **decoder** is to predict word labels from the hidden representation $\mathbf{H}$. Figure 5.2 shows LAS decoder architecture graphically. For each timestep i a word label is predicted as following: first, we predict the LSTM state $\mathbf{s}_i$ (vector of size H), represented as the output of the decoder LSTM:

$$\mathbf{s}_i = LSTM(\mathbf{c}_{i-1}, \mathbf{y}_{i-1}), \tag{5.1}$$

where $\mathbf{c}_i$ is the context vector (of size H) of the attention module from the previous timestep and $\mathbf{y}_{i-1}$ is the embedding vector (of size H) of the word label predicted for the previous timestep w_{i-1}[1].

Then, given the LSTM state $\mathbf{s}_i$ and encoded hidden representation $\mathbf{H}$, the context vector $\mathbf{c}_i$ (of size H) is calculated the following way:

$$\mathbf{c}_i = Attention(\mathbf{s}_i, \mathbf{H}), \tag{5.2}$$

where $Attention(\mathbf{s}_i, \mathbf{H})$ block has a complex inner structure introduced in [Chorowski et al., 2015b]. Attention evaluation steps are the following: first,

$$\mathbf{f}_i = \mathbf{F} * \boldsymbol{\alpha}_{i-1}, \tag{5.3}$$

where $\mathbf{F}$ is a trainable set of convolution filters, and $\mathbf{f}_i$ is a frame-by-frame representation informing the network about previous attention. This location-sensitive attention $\mathbf{f}$ is then used as an additional feature in the attention mechanism:

$$e_{i,j} = \mathbf{z}^T \tanh(\mathbf{U}\mathbf{s}_{i-1} + \mathbf{V}\mathbf{h}_{j-1} + \mathbf{W}\mathbf{f}_{i,j} + \mathbf{b}) \tag{5.4}$$

[1] Note that our notation in (5.1) is different from [Chan et al., 2016], as we do not explicitly pass the previous state $\mathbf{s}_{i-1}$ as a parameter to the LSTM block. We consider our notation to be more correct as the LSTM block implicitly depends on the previos state $\mathbf{s}_{i-1}$ (as well as on the cell state). Our notation is consistent with [Hsiao et al., 2020], which also tries to re-write LAS definitions in a more coherent way.

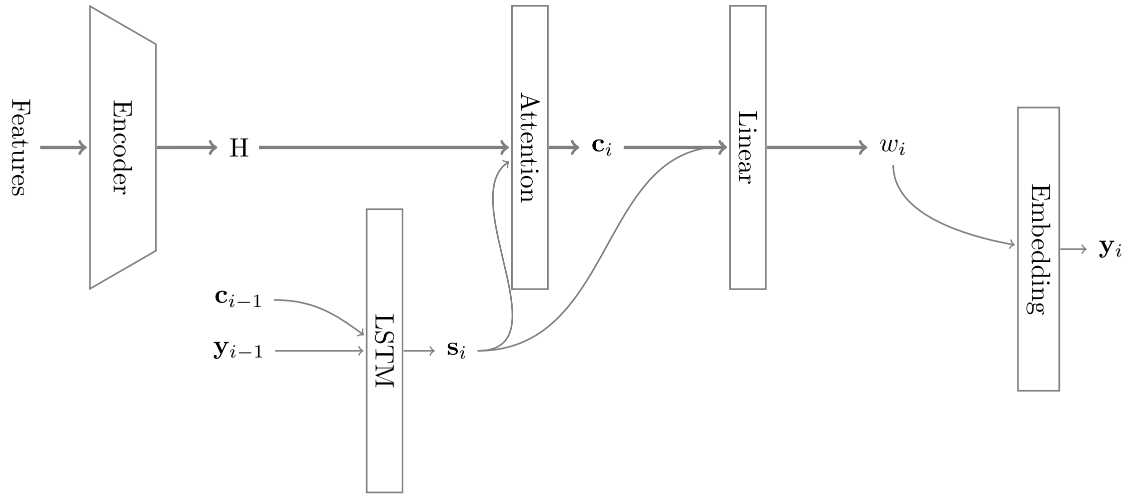


Figure 5.2: LAS system decoder architecture.

$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{T} \exp(e_{i,k})} \tag{5.5}$$

$$\mathbf{c}_i = \sum_{j}^{T} (\alpha_{i,j} \mathbf{h}_j). \tag{5.6}$$

Here, $T = {}^{X}\!/_{4}$ is the number of acoustic frames subsampled by the encoder by the factor of 4; the weights $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ are trainable weight matrices and $\mathbf{z}$, $\mathbf{b}$ are trainable vectors; $\mathbf{c}_i$ is a weighted sum of $\mathbf{H}$ using the frame-level attention weights $\alpha_{i,j}$. Therefore, $\mathbf{c}_i$ can be seen as a summary vector representing a subsequence in $\mathbf{H}$ that is responsible for producing the current prediction for timestep i.

Given the LSTM output $\mathbf{s}_i$ and the attention output $\mathbf{c}_i$, the current output label w_i is predicted by the LAS decoder as follows:

$$P(w_i|\mathbf{X}; w_1, .., w_{i-1}) = softmax(Linear([\mathbf{s}_i, \mathbf{c}_i]))). \tag{5.7}$$

The one-best output from the softmax layer gives the word label w_i which is associated with the embedding vector $\mathbf{y}_i$. This one-best embedding from the current timestep will be one of the inputs to the LSTM in the next timestep. During training, teacher forcing or scheduled sampling can be used to pass the correct embedding to the next step.

Attention loss (L_{att}) is cross-entropy (CE) with label smoothing between the predicted and the ground-truth label sequences.

Described above is the pure LAS system, but we also use a hybrid CTC/attention training within the same encoder-decoder architecture from [Watanabe et al., 2017]. While attention works in a word-synchronous way, CTC is frame-synchronous, and generates output label predictions for every subsampled frame in hidden representation $\mathbf{H}$. During hybrid training, attention loss (L_{att}) and CTC loss (L_{ctc}) for every utterance are evaluated separately following [Watanabe et al., 2017] and are combined for multiobjective learning using a tunable hyper-parameter $\alpha \in (0, 1)$:

$$L_{hybrid} = \alpha L_{ctc} + (1 - \alpha) L_{att}. \tag{5.8}$$

Although CTC and attention decoders can be used jointly during the decoding to inform each other's predictions, we only use hybrid approach for training. During the decoding, we use CTC alignments and attention separately for extracting information necessary for OOV detection.

5.1.2 System Training

In the following experiments we use hidden size $H = 800$ and the model is optimized with ADAM [Kingma and Ba, 2014] optimizer. In the beginning of the training, we use 30 000 steps of gradual warmup [Goyal et al., 2017] to get from the initial learning rate (LR) of 0.000001 to 0.0001. After that, LR stays constant while WER1 on the dev-clean set keeps decreasing. If there is an increase in WER1, LR is halved. The training stops when the LR reaches a pre-defined minimum.

We perform scheduled sampling [Bengio et al., 2015] during training: for every timestep, the correct label (teacher forcing) is selected with the probability 0.6 and the rest of the time the most likely predicted label is selected. This selected label w_i is then used for retrieving its embedding $\mathbf{y}_i$ to serve as the decoder LSTM input for the next time step. The predicted labels sequences are decoded with a beam-size of 10.

5.1.3 E2E Baselines

We have chosen to compare our baselines to the results in [Zeyer et al., 2018], as it uses a similar architecture (LAS) and model size (90M parameters) and does not use any external data. First, we trained a BPE-predicting LAS system to verify that out results are compatible with the reference, and then we switched to systems with word label outputs – word-predicting networks (WPNs) – as they are the ones suitable for OOV detection.

Table 5.1 shows, that, when tested on the full LibriSpeech database (960 hours), our BPE-predicting system functions on par with the systems with no LM reported in [Zeyer et al., 2018].

Lines 3 – 7 of the table show WERs for WPNs with different target vocabularies: 5000 words or 10000 words not including the words from the OOV list described in Section 3.3. Rows 3, 5, and 7 show WER1 as calculated on reference transcription (see Section 1.1.1); it is much higher than for the BPE experiment due to the introduction of OOVs. Rows 4 and 6 show the potential of the E2E system as an OOV detector, as it treats predicting the OOV label for an OOV word as correct, and not as substitution (see WER2 in Section 1.1.2).

Detection experiments in this chapter will be mostly reported on the system with 10000 target word labels. The last row shows WER1 for a hybrid CTC/attention system that gives most weight to the CTC training objective during training ($\alpha = 0.9$ in (5.8)). For the ASR task, it performs worse than the pure attention system, but as we will see later, this system is helpful in the OOV detection task.

5.2 OOV Detection

We experiment with two approaches to OOV detection. The first involves estimating OOV positions from attention weights, and the second uses per-frame CTC predictions.

Detection success is evaluated as following: The reference timing is obtained by force aligning the reference transcriptions containing target OOVs to the acoustic features. These force alignments have been done using a system trained as described in Chapter 4 following

	system	dev_clean	dev_other	test_clean	test_other
1	[Zeyer et al., 2018] no LM WER1	4.87	14.37	4.87	15.39
2	5000 BPEs WER1	4.99	15.18	5.02	15.65
3	5000 words attention WER1	15.38	26.75	16.05	27.24
4	5000 words attention WER2	6.47	19.12	6.92	19.43
5	10000 words attention WER1	14.21	26.61	14.58	27.19
6	10000 words attention WER2	8.66	21.78	8.79	22.36
7	10000 w. ctc+att ($\alpha = 0.9$) WER1	15.38	27.29	16.00	27.85

Table 5.1: Comparison of BPE baseline results with [Zeyer et al., 2018] and results for word-predicting E2E systems on 5k and 10k vocabulary.

a standard Kaldi [Povey et al., 2011] HMM/GMM recipe. When reporting the OOV detection, we use a hard decision metric described in Subsubsection 1.1.2: an OOV occurrence is treated as a true positive if the hypothesis overlaps with the reference for more than half of the reference duration.

As the ultimate goal of our experiments is improving recovery of repeating OOVs, the detection recall is more important – while incorrectly detected occurrences will most likely form singleton clusters and therefore be ignored after the clustering stage, the occurrences that are not detected at all have no chance to be recovered.

5.2.1 OOV Detection with Attention

A pure LAS-based E2E system in (5.8)) is used for the OOV detection experiments described in this section. For each output label, attention vector $\boldsymbol{\alpha}_i$ (5.5) is pointing to certain frames that are relevant to the current decision. However, in contradiction with [Thomas et al., 2019], where the centers of attention were assumed to point to the centers of OOVs, there is no guarantee that attention will be aligned to the real position of the word in the output.

Table 5.2 illustrates results of OOV detection with attention weights $\boldsymbol{\alpha}_i$ for systems with different vocabulary sizes. Two methods are used to estimate word position. In the first scenario, each OOV occurrence is detected at the position of the maximum attention weight corresponding to an OOV output label. We consider this detection to be true positive if its position is anywhere between the start and end of the reference OOV occurrence. An OOV is considered unfound if no maximum of OOV label attention for the utterance falls in between its reference start and end times. With this calculation method, the recall on the reference list of words reaches only 30 %.

However, if we look if there is overlap between the reference timings and the frames responsible for 90 % of attention mass, recall goes dramatically up. This experiment shows that the maximum of attention is definitely not the center of the corresponding word, nor is attention symmetric around the maximum. As shown in the last three columns of Table 5.2, the length of attention span for 90 % of attention weight is about 22.8 (subsampled) frames, and has a longer tail in the direction of the future.

If we plot LAS attention vectors $\boldsymbol{\alpha}_i$ for each predicted word label as shown in Figure 5.4 (a), we get more insight into what happens. It can be noted that some of the attention maxima lie before the centers of words and the span of attention is indeed located earlier in time than the reference word alignments. Figure 5.3 shows the amount of OOVs unfound by the maximum of attention with different attention time shifts. It can be seen

Number of labels	Detection with max. of attention		Detection using 90 % att. mass range		Attention span (frames)		
	unfound	recall %	unfound	recall %	left	right	total
5000	47524	21.6	26568	56.2	3.8	11.3	19.1
10000	44304	26.9	19798	67.3	6.2	11.3	21.5
27000	41650	31.3	20208	66.6	8.4	10.5	22.8

Table 5.2: OOV recall from attention depending on different label setups. Unfound words and recall are calculated on the reference list of OOVs (60661 occurrences).

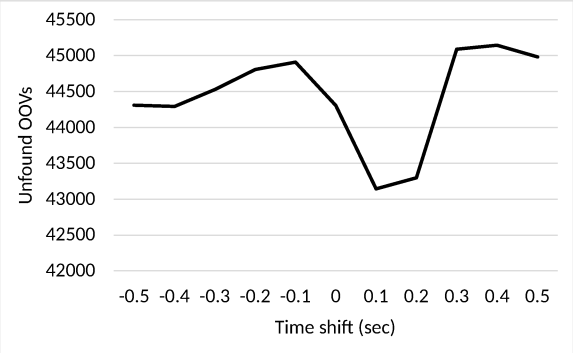


Figure 5.3: OOVs unfound by attention maximum with different time shifts of attention.

that the best attention maximum recovery can be reached with attention delay of 0.1 – 0.2 seconds.

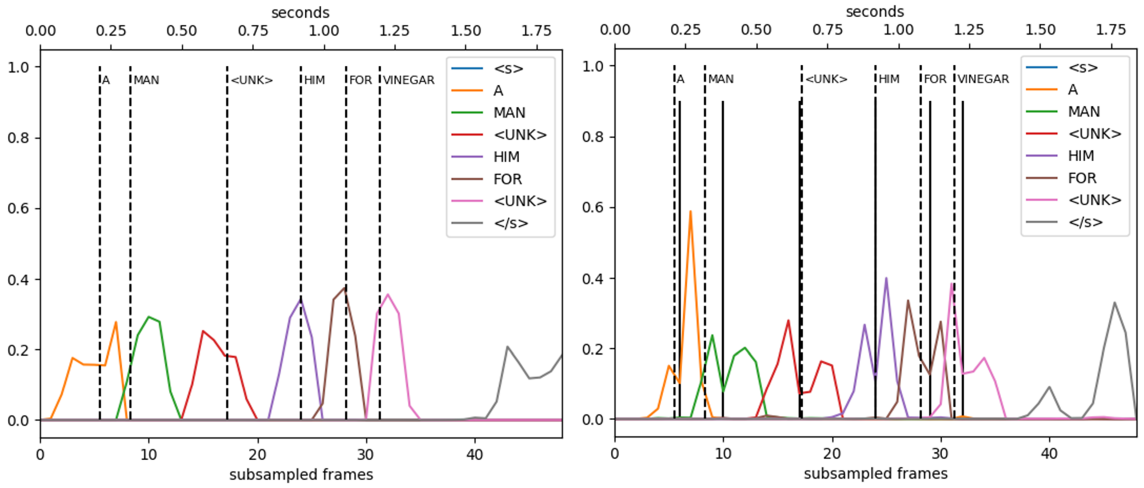
The discrepancy between attention and reference word and phoneme alignments has negative influence on recovery: if we convert frames receiving 90 % of attention mass for OOVs into times, and then extract their pronunciations according to these times and run clustering on them, the result will be very poor (see Table 5.3, the first row). There are almost no repeating phoneme strings to cluster, hence, there are not many clusters of at least size 2 and a low recovery recall. To solve this problem, the next experiment was made with timings obtained from attention with different delays. The best results have been observed with a delay of 0.2 seconds (see Table 5.3, the second row).

5.2.2 OOV Detection with CTC Alignments

It is observed that in E2E systems trained with CTC objective, CTC alignment behaves in such a way that it is possible to obtain timing information from the output labels. Word boundaries are positioned on the (subsampled) frames for which a new label is predicted. When the output is a blank symbol or the same label as before, there is no word boundary.

Table 5.3 shows the OOV detection results for the hybrid attention/CTC system with different weights given to attention and CTC costs (α from (5.8)) in rows 3 and 4, respectively. Both result in more extracted OOV occurrences than with the attention-based OOV detection due to the fact that attention tends to be very spiky. These occurrences also have much better overlap with reference timings, as confirmed by the detection scores. They also cluster much better and improve the recovery recall and precision metrics.

Interestingly, while attention provides better WER in general, CTC alignments are more trustworthy for the task of OOV timings extraction. This can be seen from the results



(a) OOV detection with a pure LAS

(b) OOV detection with a hybrid attention/CTC system ($\alpha = 0.9$)

Figure 5.4: Comparison of attention (a) and CTC (b) alignments to real OOV times. Lower x axis is in subsampled frames and the upper one is in seconds. Dashed lines with labels show reference transcriptions with time alignments. Black lines show borders of words according to CTC. Colored plots show attention weights for corresponding output labels.

obtained by the system with 0.9 cost given to CTC (4th row in Table 5.3). In Figure 5.4 (b), alignments from CTC are drawn as black lines. It can be seen that, although attention does not reflect OOV position well, the word boundaries from CTC alignment mostly correspond to the reference word boundaries.

5.3 Recurrent OOV Recovery

The goal of this next step is to find only recurrent OOVs and to recover their pronunciations and spelling. To this goal, we cluster the OOV occurrences based on their pronunciation similarity with the aim to obtain clusters, each corresponding to one unique (possibly recurrent) OOV word.

The OOV occurrences detected in the previous step serve as the first input to the recovery operation. Detected OOVs with the lengths of less than 0.5 seconds are discarded, to avoid getting too many occurrences to cluster. Moreover, shorter occurrences are usually not full words but rather suffixes and hesitations.

Another input is possible phonetic pronunciations and probabilities of these pronunciations for each OOV occurrence extracted from a phoneme recognizer. A Kaldi [Povey et al., 2011] phoneme recognizer system is trained, following the same recipe as for words in Chapter 4, on the 100 hours of clean data to generate decoded phoneme lattices. From decoded phoneme lattices, 50-best phoneme strings hypotheses are extracted with their probabilities and time alignments. From these, all phoneme substrings that are within the detected start and end times are extracted. This way, we extract a set of alternative pronunciations for each detected OOV occurrence. We efficiently store the alternative pronunciations and their probabilities in the form of a WFST by performing union of the linear WFSTs corresponding to the individual 50 alternative pronunciations followed by minimization of the resulting WFST.

More formally, for each OOV occurrence n, we have distribution $P(\Theta)_n$ over its possible phonetic pronunciations Θ, represented by WFSTs.

5.3.1 Probabilistic OOV Candidate Clustering

The clustering is based on a non-parametric probabilistic model – Dirichlet Process, where the base distribution is a uniform distribution over all possible pronunciations. The usual Gibbs sampling using Chinese restaurant process (CRP) is used for the iterative inference in the model. At each iteration of the Gibbs sampling process, a particular assignment of OOV occurrences to clusters is sampled. Upon convergence, these can be seen as samples from the posterior distribution over the possible clusterings (i.e. likely clusterings are suggested). Before describing the iterative inference in the model, we need to introduce a few terms:

Let us assume a particular assignment of OOV occurrences to clusters. Our model assumes that each cluster corresponds to an OOV word with only one correct pronunciation and all the OOV occurrences that end up in the same cluster have that pronunciation. However, since we are uncertain about the correct pronunciation of individual OOV occurrences (as expressed by the distributions of the possible pronunciations $P(\Theta)_n$), we will be also uncertain about the correct pronunciation of a cluster. Given an assignment of OOV occurrences to cluster C, we calculate the probabilities of the possible pronunciations of corresponding OOV word as

$$P(\Theta|C) \propto \prod_{n \in C} P(\Theta)_n. \tag{5.9}$$

The product of the pronunciation probabilities in this equation expresses that the likely pronunciations for the cluster are the ones that all the OOV occurrences in the cluster agree on. Since we represent the distributions $P(\Theta)_n$ using WFSTs, we can easily represent $P(\Theta|C)$ by a WFST constructed as a composition of the individual WFSTs from the cluster and re-normalized using weight pushing.

Given a particular assignment of OOV occurrences to clusters, the CRP defines the prior probability that a "new" (yet unclustered) OOV occurrence comes from an existing cluster C or starts a new cluster as

$$P(C) = \begin{cases} \frac{\alpha}{\alpha+N} & \text{for a new cluster} \\ \frac{N_C}{\alpha+N} & \text{for existing cluster } C, \end{cases} \tag{5.10}$$

where N is the total number of observations already assigned to clusters , N_C is the size of cluster C, and α is the concentration parameter controlling the probability of creating a new class (i.e. controls the number of discovered clusters).

If we were certain that the pronunciation of a "new" OOV occurrence n is Θ_n, we could calculate the posterior probability that this occurrence belongs to cluster C simply using Bayes rule

$$P(C|n) \propto P(\Theta_n|C)P(C). \tag{5.11}$$

However, since we are uncertain about the pronunciation Θ, we replace the likelihood $P(\Theta_n|C)$ by the expected likelihood where the expectation is taken with respect to the distribution $P(\Theta)_n$. The posterior probability is therefore given as

$$P(C|n) \propto \Big(\sum_{\Theta} P(\Theta)_n P(\Theta|C)\Big) P(C). \tag{5.12}$$

The complete iterative Gibbs sampling inference can be carried out as follows:

Algorithm 2 Gibbs Sampling Inference for Probabilistic OOV Candidate Clustering

```
initialize the clustering with each OOV occurrence forming a separate cluster
while not converged do
    for every OOV occurrence $n$ do
        “remove” $n$ from its cluster: update $P(\Theta|C)$ and $P(C)$ of the affected cluster
        re-assign $n$ to a (potentially new) cluster according to distribution $P(C|n)$
        update $P(\Theta|C)$ and $P(C)$ of the cluster
    end for
end while
```

We start with each OOV occurrence forming a separate cluster. We cycle one-by-one through the individual OOV occurrences. At each step, we remove one occurrence from its cluster, which affects probabilities $P(\Theta|C)$ and $P(C)$; they need to be updated at this point. Then, we re-sample its assignment to a (potentially new) cluster according to distribution $P(C|n)$. Upon the convergence, (e.g. we do not see significant changes in the clustering), we can pick the current assignment of OOVs to clusters as a likely clustering and derive pronunciations of OOV words corresponding to each cluster as its most likely pronunciation according to $P(\Theta|C)$. Since we are interested only in recovering the recurrent OOVs, we drop any singleton clusters and we do not consider them in the OOV recovery evaluation.

In practice, this correct Gibbs sampling procedure takes a lot of time, and we use an approximation: In each iteration, we fix $P(\Theta|C)$ and $P(C)$ given the current clustering and re-sample all the occurrences in parallel.

5.3.2 OOV Recovery Results

After clustering, clusters with two or more OOV candidates are post-processed to obtain graphemic representations of the recovered OOVs. For this, one best path is taken from the composition of all OOV occurrences in each cluster, and these paths are given to a phoneme-to-grapheme system. Phoneme to grapheme conversion that we use is again a joint-sequence P2G model [Bisani and Ney, 2008], trained on a reverse, phoneme to grapheme dictionary of size 200 000 with the 1000 OOVs from the list excluded.

Table 5.3 shows that successes in detection recall mostly translate into recovery success - the more candidates are there at the start of the clustering process, the more chance the clustering process has for discovering a repeating pattern. Improving the detection boundaries, for example by the time shift in attention-only detection, or giving more weight to CTC objective, also improves both recovery recall and precision.

5.4 Comparison with the Hybrid ASR

Experiments have shown that the E2E approach to OOV detection and recovery achieves better results than the hybrid FST approach (see Table 5.4), especially in terms of detection recall (25 % → 81.5 %), which is crucial for also improving recovery recall (1.4 % → 14 %). To be precise, detection recall of the E2E system should more honestly be compared

Detection method	WER %	# cands	Detection		Recovery		
			recall %	precision %	# clusters	recall %	precision %
attention	9.8	57701	33.4	14.5	147	0.7	6
att. 0.2 sec right shift	9.8	57669	34.4	14.9	489	5.5	29
ctc+att $\beta = 0.5$	10.7	134843	73.7	32.0	1996	8.3	15
ctc+att $\beta = 0.9$	13.0	148428	81.5	35.3	3485	14.0	15

Table 5.3: OOV Detection and recovery results using different combination of attention and CTC decoding. Column "Cands" shows the amount of OOV candidates on the clustering input, and column "clusters" shows the number of clusters of size 2 or more after clustering saturates.

Approach	Detection method	Detection		Recovery		
		# cands	recall %	# clusters	recall %	precision %
WFST	one best	1247	2.0	7	0.3	43
	full lattice	15991	26.0	20	1.4	50
E2E	attention	57669	34.4	489	5.5	29
	ctc+att	148428	81.5	3485	14.0	15

Table 5.4: Comparison of the best detection and recovery results in the frameworks of WFST and E2E approaches on train-clean-360 dataset.

with detection results for one-best path in an WFST-based system, because in the experiments presented in this chapter we only worked with one-best output of a WPN. Detection precision is not directly comparable to WFST-based decoding, as there we consider also candidates extracted from sub-optimal paths.

Despite the improvement of E2E detection and recovery results in comparison with FST results, it is problematic to utilize recovered OOVs to improve the word-predicting E2E system for future uses with this approach. The newly discovered words can be added as output labels but, unlike in a hybrid ASR system, which uses lower-level phoneme representations, this output will stay untrained.

5.5 Conclusion

We have shown that E2E approach has definite potential to be applied for the task of OOV detection. CTC alignments provide better temporal information about word position than the pure attention-based E2E system, and so are more suitable for the task of extracting OOV occurrences. Improved detection results also correlate with better recovery of recurrent OOVs. For the pure attention-based E2E model, it can be seen that even though the system performs better in terms of word error rate, there is no guarantee that attention actually provides the position of frames that are directly responsible for producing the output label in question.

Drawbacks of the described approach are the following:

- It utilizes two unconnected granularities (words and phonemes), which leads to problems with time alignment and boundaries of the discovered OOV; also, two separate

E2E systems need to be trained: a word-predicting main one and a separate phoneme recognizer.

- It is problematic to utilize recovered OOVs to improve the word-predicting E2E system: the newly discovered words can be added as output labels but these output labels will not be predicted in the decoding as they were not present in the training data, and the weights and embeddings representing them will be untrained.

Chapter 6

Speller Architecture for OOV Detection and Recovery

This part of research was partly inspired by the field of NLP (section 2.7), in particular by [Mielke and Eisner, 2018] that presents a two-level "word plus speller" generative LM for tasks involving OOVs. In [Mielke and Eisner, 2018], the open-vocabulary LM consists of two RNNs: the first one captures the sentence structure, and the second one, called the speller, captures the word structure. The speller can generate new word types following the spelling style of IVs. The novel words generated by this model fit the grammatical sentence structure well, and the model can produce a range of possible spellings that fit the language in question.

We propose to extend this approach to ASR and implement word-predicting E2E ASR training with a speller-like network. While the speller trained for LM tasks in [Mielke and Eisner, 2018] had only a text input to train on, ASR has the benefit of providing the speller with acoustic information too. The novelty lies in jointly training the word predicting network (WPN) and the speller instead of working with two separate ASR systems with outputs of different granularity as in Chapter 5. This approach can potentially recover OOVs that are not only plausible from the LM point of view, but also acoustically correct. This training should also benefit in-vocabulary word (IV) representations (and thus improve WER1 metric) by forcing the word embeddings within the ASR system to learn character representations as the second objective.

The architecture presented in this chapter was first published in [Egorova et al., 2022]. The thesis shows results on more different speller inputs and different vocabularies, reports the results of experiments with several OOV embeddings that allow for natural OOV representation clustering, and also provides in-depth analysis into what information is stored in the embeddings.

6.1 WPN Baselines

The WPN architecture and training schedule used in this chapter are the same as described in Chapter 5 and are shown in Figure 6.1 in black and blue colors. The only change in comparison with the LAS E2E system of the previous chapter is that the embedding layer weights (*Embedding* box in Figure 6.1 denoting a linear layer that projects word label into its embedding of size 1600) are tied [Inan et al., 2017] to the weights of the linear layer

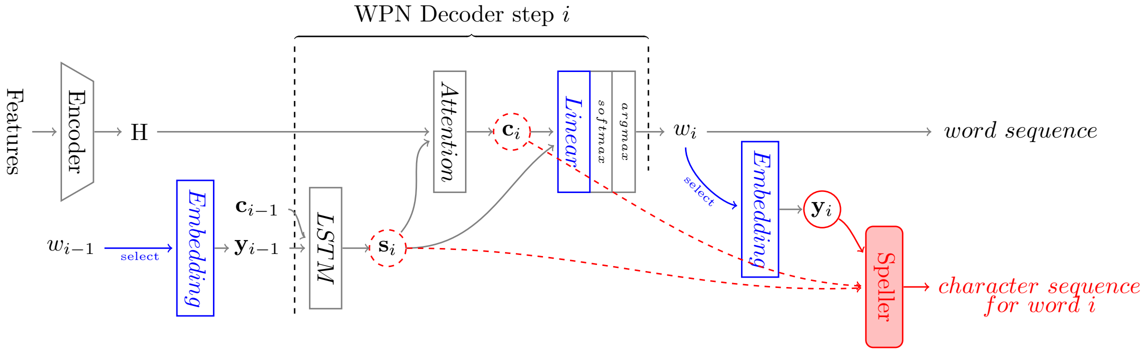


Figure 6.1: Baseline LAS-like WPN (in black) and speller network trained on different inputs (in red). Weights of layers in blue are tied.

from (5.7): $Embedding = Linear^T$. Thus, for each word label w_i, the embedding $\mathbf{y}_i$ is the corresponding row from the linear predicting layer weight matrix.

The baselines we will be comparing with are the same as reported in Table 5.1. We reiterate BPE and 10000 words baselines (denoted 10000w) in the first part of Table 6.1. As we will be comparing our OOV recovery with the BPE-predicting system, we first provide an analysis of BPE recovery potential in the form of a word accuracy metric. Lines named “BPE rOOVs” and “BPE rIVs” show percentages of correctly recovered rare (OOVs) and common (IVs) words. There are no real OOVs in a BPE system, so we count the most frequent 10000 words (same as IVs in the 10000w system) as N_{IV}, and T_{IV} is the number of IVs that is correctly recognized. Then,

$$rIVs = \frac{T_{IV}}{N_{IV}}. \tag{6.1}$$

rIVs recovery rate is reported in line “BPE rIVs”. Rare words N_{OOV} are words which are OOVs in the 10000w system and T_{OOV} is the number of such words that was recognized correctly. Then,

$$rOOVs = \frac{T_{OOV}}{N_{OOV}}. \tag{6.2}$$

rOOVs recovery rate is reported in line “BPE rOOVs”.

Table 6.1 shows that although rIV recovery rate is very high: more than 95% on clean data, for rare words, rOOV is just over 60%, which means that while BPE system is very successful at recognizing common words, there is potential for improvement in the task of recovering rare words in a BPE system.

6.2 Speller Architecture

The speller consists of a single layer LSTM and a linear output layer, and it is optimized for predicting a string of characters that constitute a word. In Figure 6.1, it is shown in red. Several red lines leading to the speller represent different inputs that were given to the speller in the following experiments. The line going from the word embedding $\mathbf{y}_i$ is solid as this is the main representation of the word, which we then concatenate to other inner representations from the WPN.

The speller solves a simpler problem, and so is much smaller than the WPN model: our baseline WPN model with 5000 outputs contains 90M trainable parameters; the same model with the addition of the speller has 98M trainable parameters. As the speller does not increase the WPN model dramatically, no change in training was needed; the learning rate, batch sizes, warmup etc. stayed the same as for the baselines.

My speller implementation in Python can be found on GitHub[1].

6.3 Experiments with Speller Inputs

In the experiments, the general architecture will stay fixed, and we will experiment with different inputs to the speller to discover which parts of the WPN benefit most from being trained jointly with the speller and, vice versa, in which representations of the WPN information relevant to spelling can be discovered. The spellers that have been tested are the following:

1. speller with only embeddings $\mathbf{y}_i$ of the word to be spelled as input (section 6.3.1)
2. speller with input of concatenation of word embedding $\mathbf{y}_i$ with either or both of context vector $\mathbf{s}_i$ and attention output $\mathbf{c}_i$, and speller with the concatenation of just $\mathbf{s}_i$, and $\mathbf{c}_i$ as the input (section 6.3.2)
3. the same speller but with OOVs allowed to be represented by several embeddings (section 6.3.3)
4. two spellers trained concurrently to investigate OOV embeddings clustering, one predicting from just word embeddings $\mathbf{y}_i$ and another predicting from the concatenation of $\mathbf{y}_i$, $\mathbf{s}_i$, and $\mathbf{c}_i$ (section 6.3.4)

6.3.1 Embedding Speller

Inspired by the LM-speller in [Mielke and Eisner, 2018], the first speller architecture we trained takes word embeddings $\mathbf{y}_i$ as an input and generates letters as an output. The limitation of this approach is that a number of diverse OOVs are assigned a single OOV label and thus a single embedding. This single embedding cannot learn all the spellings of OOVs, and therefore the OOV embedding is not updated through the speller.

To spell an IV word, its embedding is repeated as a constant input for every output letter (one-to-many RNN). There is interaction between the WPN and the speller as the word embeddings are shared by both networks and are updated with both the WPN loss (L_{att}) and the speller loss (L_{sp}) during the training. L_{sp} is calculated for every word in the dictionary as CE between the reference spelling and the speller output for this word's embedding. To accommodate updates from both networks, the training iterates between updating the WPN and the speller network as shown in Algorithm 3. This training ensures that the word embeddings are trained to represent not only the information useful for word prediction but also the information useful for the speller task, i.e. it makes word embeddings spelling-aware, which is beneficial for WPN training.

We have found it sufficient to update the speller once for each word in the vocabulary after every 500 mini-batches of 20 utterances of WPN training. Thus, in every update,

[1] https://github.com/BUTSpeechFIT/speller

system		metric (all %)	dev_clean	dev_other	test_clean	test_other
no speller	5000 BPE	WER1	4.99	15.18	5.02	15.65
		rOOVs	63.8	36.5	62.0	33.2
		rIVs	95.7	85.0	95.6	84.7
	10000w WPN	WER1	14.21	26.61	14.58	27.19
		WER2	8.66	21.78	8.79	22.36
10000w WPN with speller	$\mathbf{y}_i$	WER1	11.56	23.53	11.80	24.05
		WER2	8.05	20.82	8.61	21.10
		WER1 (spIV)	12.84	24.75	13.33	25.13
	$[\mathbf{y}_i, \mathbf{s}_i]$	WER1	10.79	21.56	10.98	22.07
		WER2	5.69	17.50	5.98	17.74
		WERr	11.39	22.39	11.36	22.76
		rOOVs	12.0	5.2	12.2	5.1
	$[\mathbf{y}_i, \mathbf{c}_i]$	WER1	10.94	21.53	10.85	22.40
		WER2	5.82	17.39	5.80	18.07
		WERr	8.84	20.59	8.79	21.42
		rOOVs	32.0	14.8	32.3	13.3
	$[\mathbf{y}_i, \mathbf{s}_i, \mathbf{c}_i]$	WER1	**9.99**	20.43	10.17	20.70
		WER2	**4.74**	16.23	5.05	16.27
		WERr	**7.16**	19.05	**7.29**	19.44
		rOOVs	**44.4**	**23.3**	**45.9**	**19.2**
	$[\mathbf{s}_i, \mathbf{c}_i]$	WER1	10.13	**19.89**	**10.12**	**20.28**
		WER2	4.86	**15.59**	**4.97**	**15.81**
		WERr	7.42	**18.58**	7.36	**18.91**
		rOOVs	41.8	20.7	43.2	19.1

Table 6.1: Comparison of different speller architectures with BPE and 10000w WPN baselines. WER1 and WER2 show the performance of the WPN without speller participation. WERr shows results with OOVs recovered through the speller, and rOOVs shows the percentage of OOVs that were recovered. In the speller section, the best results for every metric are shown in bold.

each word in the dictionary updates its embedding once and also contributes with an equal weight to updating the speller weights (WPN weights are not affected by the updates from the speller training step).

Two LRs are used in this speller architecture: WPN LR and speller LR. Both the WPN LR and the speller LR start with the initial value of 0.000001. In the beginning of the training, we use 30 000 steps of gradual warmup for WPN LR and $30000/500 = 60$ warmup steps for speller LR to ensure that it is maximized at the same time as the WPN LR. The maximum LR for both WPN and speller equals to 0.0001. After that, the two LRs are updated separately: WPN LR is halved when WER1 on dev-clean increases, and speller LR is halved when speller character error rate (CER) calculated on the dictionary increases. The training stops when the WPN LR reaches a pre-defined minimum.

Algorithm 3 Embedding Speller Training

```
while LR > predefined minimum do
    for number of mini-batches in training epoch do
        evaluate $L_{att}$
        update WPN weights to improve $L_{att}$
    end for
    for every IV word do
        evaluate $L_{sp}$
        update speller weights and word embedding to imrove $L_{sp}$
    end for
    calculate WER for WPN on the validation set
    if WER > WER from the previous epoch then
        do WPN LR halving
    end if
    calculate CER for the speller on the dictionary
    if CER > CER from the previous epoch then
        do speller LR halving
    end if
end while
```

The fist section of speller experiments in Table 6.1 shows the results for the join training of the WPN with embeddings-only speller. Note that this system uses only a single embedding representing all the diverse OOVs and although we can sample different OOV spellings from it, we cannot recover spelling for particular OOVs based on acoustic evidence or context. This is the reason why OOV recovery results (WERr and rOOVs, defined in Section 1.1.2) are not reported for this speller input.

It can be seen that spelling-aware embeddings improve both WER1 and WER2 (defined in Section 1.1.2). As the speller is not used in the decoding, the improvement in WER1 and WER2 does not come from the slight increase of the number of parameters, but solely from the fact that word embeddings are forced to be aware of the spelling.

The third metric ("WER1 (spIV)") shows the performance of the speller. First, the word label is predicted, and if it is an IV label, it is passed through the speller to obtain character representation. Only if the spelling is correct, the word is not considered an error. The "WER1 (spIV)" WER increases only slightly in comparison to WER1, which shows that the speller part learns to spell in-vocabulary embeddings almost perfectly.

6.3.2 Context- and Acoustics-Aware Speller

To give the speller more information about the OOVs that we want it to spell, several other speller inputs have been tested (see dashed red lines in Figure 6.1):

1. concatenation of word embedding $\mathbf{y}_i$ and context vector $\mathbf{s}_i$ from decoder LSTM,
2. concatenation of word embedding $\mathbf{y}_i$ and attention output $\mathbf{c}_i$,
3. concatenation of $\mathbf{y}_i$, $\mathbf{c}_i$, and $\mathbf{s}_i$,
4. concatenation of only $\mathbf{c}_i$, and $\mathbf{s}_i$.

Algorithm 4 Context- and Acoustics-Aware Speller Training

```
while LR > predefined minimum do
    for number of mini-batches in training epoch do
        for number of words in a batch do
            evaluate $L_w$
            get $\mathbf{y}_i$ embedding of one-best prediction
            generate speller output for $\mathbf{y}_i$ and/or other inputs
            evaluate $L_{sp}$
            calculate combined loss for the word = $(1 - \alpha)L_w + \alpha L_{sp}$
        end for
        evaluate $L_{utt}$ (6.3)
        update all weights to improve $L_{utt}$
    end for
    calculate WER for WPN on the validation set
    if WER > WER from the previous epoch then
        do LR halving
    end if
end while
```

The speller is trained to predict the correct sequence of characters (spelling) given the speller input. Concatenating the word embedding with $\mathbf{s}_i$ gives the speller context information, as decoder LSTM preserves label history, while concatenating the word embedding with $\mathbf{c}_i$ gives the speller knowledge about acoustic information relevant for the currently decoded word. $\mathbf{s}_i$ has no acoustic information about the first word in an utterance and is not useful for spelling it.

As the speller needs current inner representations from the WPN for every timestamp, the training cannot proceed iteratively as for the embedding-only architecture. This is why the speller is updated simultaneously with the WPN during the training. After every word hypothesis is generated by the WPN, the speller is given the embedding $\mathbf{y}_i$ of the current one-best word label together with the current $\mathbf{c}_i$ and/or $\mathbf{s}_i$. There is no teacher-forcing for the speller, meaning that it always gets the embedding of the predicted word label, not the reference word label. This is due to the fact that the weights of the embedding $\mathbf{y}_i$ are tied to the weights of the last WPN linear layer, and we want the speller to update the relevant weights.

The training for the context- and acoustics-aware speller is summarized in Algorithm 4. The overall loss for joint training of the WPN and the speller is a weighted combination of the losses:

$$L_{utt} = \sum_{w \in utt} ((1 - \alpha)L_w + \alpha L_{sp}), \tag{6.3}$$

where L_w is the WPN loss calculated from the CE between the WPN output and the reference word label. L_{sp} is the speller loss for this word and is calculated from the CE between the speller output and the spelling of the reference word. The reference character sequence is obtained by tokenizing the correct word label into characters. L_{sp} is normalized by the number of characters in the reference word. The two losses are combined for every word using a tunable hyper-parameter α.

For the experiments presented in Table 6.1, equal cost weights are given to the speller and the WPN updates ($\alpha = 0.5$). Unlike during the embedding speller training described in Subsection 6.3.1, here, each embedding is not updated equal number of times during training, but the amount of times it appears in the training data, so the speller is better attuned to more frequent words.

In the test time, the WPN calculates the output for every timestamp (word) first, and if the best predicted word label happens to be the OOV label, the speller network predicts the spelling of the current OOV from the current input. This cascade is thus able to both recognize IVs as well as recover OOVs. The output is a string of words: part of them are IVs and part are speller-generated.

The last four sections of Table 6.1 show performances of the four different speller input variants. As before, WER1 and WER2 are scored on the output of the WPN and do not use the speller network during decoding. Any improvements happening with WER1 and WER2 in comparison with the no speller baseline come due to the regularizing effect that the speller has on the word embeddings. The best results are achieved by the systems with speller input either concatenating $\mathbf{y}_i$, $\mathbf{c}_i$, and $\mathbf{s}_i$ or just $\mathbf{c}_i$, and $\mathbf{s}_i$. These systems outperform both the concatenations $[\mathbf{y}_i, \mathbf{c}_i]$ and $[\mathbf{y}_i, \mathbf{s}_i]$.

The two metrics that we use to evaluate speller systems are WERr and rOOVs, which show the capacity of these systems to recover OOVs through spelling. First, the word label is predicted, and if it is the OOV label, the inputs from the current decoding step are passed through the speller to obtain the character representation. The resulting output is then a sequence of words, some of them predicted IV words, and some recovered OOVs. This output is then scored against the reference transcription to obtain "WERr" metric (see section 1.1.2). Meanwhile, "rOOVs" shows the percentage of OOVs that were ideally recovered through this process.

Concatenating word embedding $\mathbf{y}_i$ with $\mathbf{s}_i$ as the speller input makes the speller context-aware. This architecture has the drawback of not being able to spell an OOV if it happens to be the first word in a sentence: the input to the decoder LSTM is a vector of zeros. Table 6.1 clearly shows that the addition of the context information is not as helpful for WERr as the other architectures. However, this training still improves WER1 and WER2, and it is sometimes better at improving WER1 and WER2 than the $[\mathbf{y}_i,\mathbf{c}_i]$ speller architecture.

As $\mathbf{c}_i$ contains representation of the acoustics relevant to the current word, concatenating the word embedding $\mathbf{y}_i$ with $\mathbf{c}_i$ makes the speller acoustics-aware. This information proves to be vital for OOV recovery, as is illustrated in Table 6.1 by the dramatically improved WERr as compared to WER1.

The speller system that takes the concatenation of $\mathbf{y}_i$, $\mathbf{s}_i$ and $\mathbf{c}_i$ as an input seems to take the best from both worlds and shows improvements across all metrics.

Indeed, the presence of word embedding $\mathbf{y}_i$ at the speller input is not even vital for the good performance, as can be seen from the experiments with only $\mathbf{s}_i$ and $\mathbf{c}_i$ at the input that show comparable, and in some cases better, results. The result with $\mathbf{s}_i$ and $\mathbf{c}_i$ and without the embedding $\mathbf{y}_i$ is actually one of the best performing out of all the systems. This can be explained by the fact that the word label prediction is made from the concatenation of $\mathbf{s}_i$ and $\mathbf{c}_i$ in the WPN, so all the necessary information should be in them, meanwhile, the absence of the definite embedding does not force a hard decision on the input. We investigate further what information is stored in the embeddings in the next sections.

For every speller system, rOOVs shows the percentage of reference OOVs that were ideally recovered after speller decoding. We are able to reach $44 - 46\,\%$ rOOVs for clean data and $19 - 23\,\%$ on other. This can be directly compared to the system with BPE

system	dev_clean	dev_other	test_clean	test_other
5000w 1oov	9.79	21.74	10.03	23.06
5000w 10oov	9.91	21.34	9.93	22.35
5000w 100oov	10.08	21.62	10.16	22.42
5000w 1000oov	10.38	22.98	11.02	22.38
10000w 1oov	7.16	19.05	7.29	19.44
10000w 10oovs	7.64	19.83	7.85	20.37
10000w 100oovs	7.73	19.31	7.83	20.4
10000w 1000oovs	7.55	19.3	7.67	20.22
20000w 1oov	6.76	18.44	7.17	19.14
20000w 10oov	6.54	18.36	6.9	18.66
20000w 100oov	6.47	18.19	6.68	19
20000w 1000oov	7.05	19.16	7.16	19.73
10000w 100oovs prob oov embs	8.51	20.6	8.55	21
10000w 100oovs $\mathbf{y}_i$ input only	11.11	21.5	10.88	22.25
10000w 100oovs two spellers	7.82	19.47	7.96	20.26

Table 6.2: Results (WERr) of experiments with multiple OOV embeddings.

targets. Although the BPE system does not have OOVs per se, we score rOOVs on the words that are OOVs in the 10000 word vocabulary system. Table 6.1 shows 62 – 64 % rOOVs in a BPE system for clean data and 33 – 36 % on other.

While our numbers do not reach the recovery rates of the BPE system, our double-granularity system provides additional information about the word being an OOV and also its useful internal representation in the form of the speller input. Speller output does also better than BPEs for words that are not easily reproducible from common morphemes. For example, the speller correctly recovered the words “amputation” and “adventuring” whereas the BPE suggested “amutaion” and “adventureing”. However, in general, BPE recovery performance is still better if one only cares about improving WER.

6.3.3 Multiple OOV embeddings

In an attempt to investigate if a single OOV embedding is hurting the speller performance by forcing part of the speller input to be the same for every output, a speller system was trained with multiple OOV embeddings representing OOV words.

We assume that there are N individual OOV “classes” that are represented with independent OOV labels. The system is allowed to figure freely during training which specific OOVs fall into which class, or, in other words, how to cluster the OOV representations. To this end, the output layer is extended by N outputs. As the reference transcription only contains one OOV label, the probabilities of individual OOV outputs are summed for CE calculation. The speller is always given the OOV embedding with the biggest predicted score in the WPN softmax layer as an input.

Table 6.2 shows results of systems with the $[\mathbf{y}_i, \mathbf{s}_i, \mathbf{c}_i]$ input speller and different numbers of output labels and OOV labels. Contrary to our intuition, not forcing all of the OOVs to be represented by a single embedding is not helping in recovery. It seems that spreading the

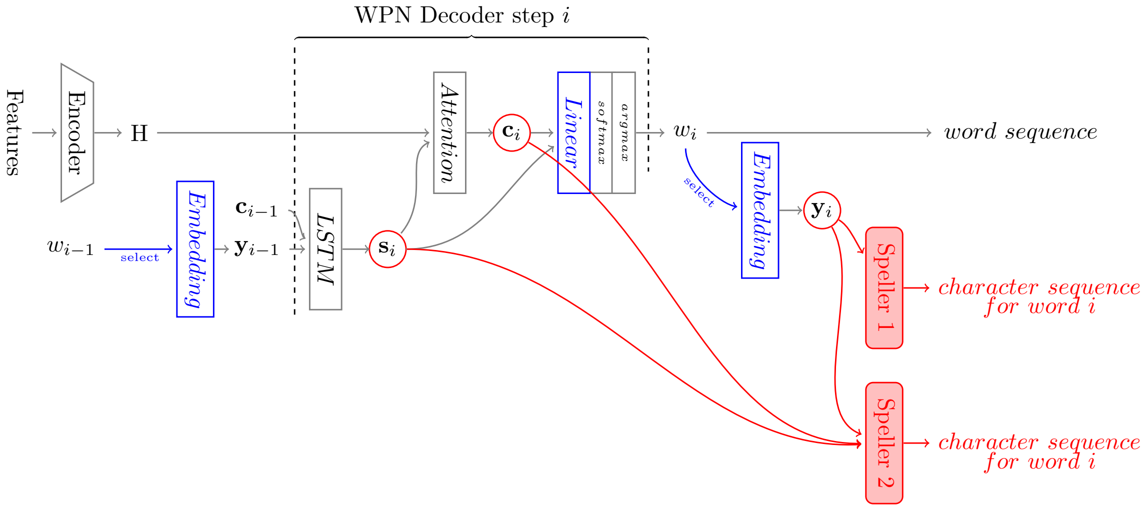


Figure 6.2: Baseline LAS-like word-predicting network (WPN) (in black) with two speller networks trained on different inputs (in red). Weights of layers in blue are tied.

training information from one to several OOV embeddings is actually hurting the training and WERr as a result.

Another approach to representing OOVs with several embeddings is to not treat these embeddings as separate entities but rather as one big probabilistic embedding matrix. In this approach the embedding $\mathbf{y}_i$ that goes as the input to the speller is calculated as the sum of all N OOV embeddings weighted by their output probabilities π_n:

$$\mathbf{y}_i = \sum_{n \in N} \pi_{i,n} \mathbf{y}_{i,n}. \tag{6.4}$$

However, this soft representation also did not prove helpful to OOV representation as can be seen in line "10000w 100oovs prob oov embs" in Table 6.2.

The failings of multiple-embedding OOV representations can be explained by the fact that spelling information is not extracted from the embedding, but rather from $\mathbf{s}_i$ and $\mathbf{c}_i$. If only 100 separate embeddings are used as the speller input, with no addition of $\mathbf{s}_i$ or $\mathbf{c}_i$ (line "10000w 100oovs $\mathbf{y}_i$ input only"), WERr shows almost no recovery has happened. Meanwhile, Table 6.1 shows that the system that provides the speller with only $\mathbf{s}_i$ and $\mathbf{c}_i$ that ignores the embedding completely performs similarly to the $[\mathbf{y}_i, \mathbf{s}_i, \mathbf{c}_i]$ system.

6.3.4 Two Spellers

To investigate what information is relevant for spelling prediction, a system with two spellers has been trained. Speller1 predicts spelling from just the embedding $\mathbf{y}_i$ (multiple separate OOV embeddings are used) and Speller2 predicts spelling from a full concatenation of $\mathbf{y}_i$, $\mathbf{s}_i$ and $\mathbf{c}_i$. Apart from discovering what information is relevant to the speller, this will give insight into how OOVs are grouped together into sets that the system learns to be represented by the same embedding. The schematic of the two speller training can be seen in Figure 6.2.

Speller1 learns an average representation of all the OOVs that are grouped under it. It can be said that it represents a specific type of OOV and OOVs of this type can be sampled from it. However, without the usage of context or acoustic information it is very improbable

True label	NOR IS MISTER QUILTER'S MANNER LESS INTERESTING THAN HIS MATTER
recovery output 1	NOR IS MISTER <UNK10085>[SAREE'S][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 2	NOR IS MISTER <UNK10086>[BANTE'SS][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 3	NOR IS MISTER <UNK10033>[SASSSS][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 4	NOR IS MISTER <UNK10021>[PORTEESS][QUOLTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 5	NOR IS MISTER <UNK10085>[SAREE'S][QUILTER'S] MANNER LESS INTERESTING THAN HIS <UNK10023>[MANAA][METTOR]
recovery output 6	NOR IS MISTER <UNK10085>[SAREE'S][QUILTER'S] MANNER LESS INTERESTING THAN HIS <UNK10102>[MANNE][METTOR]
recovery output 7	NOR IS MISTER <UNK10046>[TRRERSSS][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 8	NOR IS MISTER <UNK10075>[CAADEE][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 9	OR IS MISTER <UNK10088>[PANTIN][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER
recovery output 10	NOR IS MISTER <UNK10063>[TORTERD][QUILTER'S] MANNER LESS INTERESTING THAN HIS MATTER

Table 6.3: Recovery output example for two spellers. The identity of the OOV embedding is shown in angle brackets, Speller1 output is shown in the first square brackets and Speller2 output is in the second square brackets.

that it will actually suggest a most likely output that will spell a particular OOVs correctly. Thus Speller1 does not contribute to WERr, but this architecture allows us to analyze what information is stored in the OOV embeddings and how OOV representations cluster into OOV embeddings. For this kind of recovery analysis, whenever an OOV is predicted, the one-best spelling is first predicted with the use of Speller1. The spelling will be the same for every OOV that has been represented by this particular OOV label. Then, the one-best spelling will be predicted by Speller2 from the concatenation of $\mathbf{y}_i$, $\mathbf{s}_i$ and $\mathbf{c}_i$. We will look at the output that preserves 10 best system outputs of the joint WPN plus two spellers architecture.

Table 6.3 shows an example of one decoding output preserving 10 best recovery strings. The name "Quilter's" in the possessive form is the OOV that we will analyze. <UNK> with a number shows which of the 100 OOV embeddings was the most likely according to the WPN; the word in the first square brackets is the one-best output of Speller1, representing the averaged spelling of all the OOVs it represents; the word in the second square brackets is the output of Speller2 ([$\mathbf{y}_i$, $\mathbf{s}_i$, $\mathbf{c}_i$] input) which is able to recover the spelling. In 9 out of 10 best recovery outputs (except for output 4), Speller2 has successfully recovered the OOV in question "Quilter's", even though the most likely embedding has been different. Moreover, 6 out of the 10 best outputs (namely, outputs 1, 2, 3, 7, 8, and 10) are completely correct and differ only in the most likely OOV embedding predicted by the WPN. However, embedding information is not completely irrelevant: from one-best spellings generated from Speller1, it can be seen that the predicted embeddings mostly represent possessives and names.

Another useful analysis that we can perform is to look at all the OOV embeddings and compare the spellings generated from them to the reference OOVs instead of which they were predicted. Table 6.4 highlights the OOV embedding 10085 that generated the correct spelling "QUILTER'S" on the best path (recovery output 1 in Table 6.3) and several other paths (recovery outputs 5 and 6 in Table 6.3) for the same utterance. Its one-best Speller1 output, "SAREE'S" suggests that the cluster groups together mainly possessive words. Grammatical characteristics are observed commonly as a grouping criterium, however, some clusters also group words by first letters or spelling similarities rather than grammatical. The first column shows reference words that have been substituted with the generated OOVs and the second column shows words generated by the Speller2 with the input of $\mathbf{y}_i$, $\mathbf{s}_i$, and $\mathbf{c}_i$. Indeed, a lot of reference words are possessive names and pronouns, but there are also some words that just end with the letter "S".

6.4 Comparison with Previous Methods

We cannot directly compare detection and recovery of a speller system with previously presented approaches, as the metrics are calculated differently due to the nature of the task. But what is directly comparable is the ease of training. Speller architecture demands just adding an LSTM layer to the LAS architecture, which is very straightforward: it does not require any graph recompilation, time-aligning or training a separate phoneme recognizer.

Similar to a WFST decoding approach, speller architecture is a principled way of jointly modeling two speech granularities, and the outputs preserve correct probabilities of predicting an OOV and its spelling. However, speller is much faster than hybrid decoding, and does not necessitate optimizing as many parameters as a hybrid graph.

Speller uses similar base architecture as the E2E OOV detection system and makes good use of meaningful inner representation that it provides. However, speller does not just utilize the E2E output to recover OOV spellings from another character-predicting system, it instead trains the two granularities jointly which gives benefits to both and avoids problems with word boundaries.

More importantly than all of the above, the speller has the power to perfectly recover up to 46 % of OOVs on clean data and 23 % on other data. Speller can also naturally group OOVs into meaningful clusters if the training is performed with several OOV embeddings. All in all, speller brings together the best of the hybrid and E2E approaches described in Chapters 4 and 5 respectively and avoids some of their pitfalls.

6.5 Conclusion

We have proposed a new neural architecture for the ASR task that jointly trains two networks predicting both words and characters using shared inner representations. We have shown that forcing embeddings of a word-predicting system to also be spelling-aware improves WER1 of the word predicting task.

Different inputs to the speller network have been tested, and we have shown their capability of recovering OOVs through spelling. The best improvements across all WERs

Reference	Speller2 output
ABALONE'S	ABALONIS'S
ALEXANDER'S	ALEXANDER'S
ARABESQUE	ARABUS'
CHARLIE'S	CHARLEE'S
COOK'S	COOK'T
CREETERS	CREETOS'S
CUCUMBERS	CUCUMBER'S
CULPRIT'S	CULPEIT'S
FREDERICK'S	FREDERICK'S
JEM'S	JEM'S
GRIFFIN'S	GRIFFIN'S
HILDA'S	HILDER'S
JIM'S	GEM'S
KITTY'S	KITTY'S
LASSEN'S	LASSON'S
LUCY'S	LUCY'S
NORMAN'S	NORMAN'S
ORTHODOX	ORTHODOX
QUILTER'S	QUILTER'S
RABBIT	RABEETONSS
REGENT'S	REGENT'S
REGENT'S	REGINT'S
RUGGEDO'S	RIGOEDO'
RUDE	RUTH'S
SAILOR'S	SAILOR'S
SYNESIUS'S	SANECIEUS
SATURDAY'S	SATTERYY'S
SHEEP'S	SHEEP'S
SYDNEY'S	SIDDEY'
SYDNEY'S	SIDNEY'S
SOMEBODY'S	SOMEBODY'S
SERENA	SSSRRENA
STEVIE'S	STEVYE'S
TAD'S	TAD'S
TRAITS	TRATE'S
VERLOC'S	VERLIC'S
WAVERLEY'S	WAVERLEY'S

Table 6.4: Example of outputs of Speller2 for OOV embedding 10085 (One-best Speller1 output - “SAREE'S”) in dev-clean dataset compared to reference words that were recovered.

ware achieved with speller inputs that include use the concatenation of attention and LSTM outputs as an input, together with or without the OOV embedding.

When the speller is trained with several embeddings representing OOVs, is has also shown a capability to naturally assign OOVs to meaningful clusters that reflect grammatical and spelling similarities of the words grouped there.

Chapter 7

Conclusions

7.1 Summary

In the thesis, we have explored the field of OOV processing within the task of ASR. We have re-established clear definitions of two separate OOV processing tasks – that of detection and recovery. We have proposed and defined success metrics for both tasks and modified WER to evaluate OOV effect on ASR. We have described hybrid and E2E setups on an open access database viable for testing OOV processing approaches and provided system descriptions and code to facilitate replicability.

Three different approaches have been presented and compared, all bringing their own merits and challenges to the table. Hybrid approach used modified decoding graph with phoneme substrings for detection and recovery of potential OOVs. The novelty of the work was working in full lattice mode instead of with one-best outputs and utilizing properties of WFSTs to preserve correct probabilities of all paths. Hybrid approach successfully recovered some OOVs, which proved to be valid additions to the dictionary. However, this approach suffered from low recall rates due to the nature of the decoding. Moreover, memory and time needed for training were a problem.

The second approach utilized inner representations of a word-predicting E2E system to perform OOV detection task. Detection recall and precision rates improved drastically in comparison with the hybrid approach. However, OOV recovery had to be performed on a separate character-predicting system which led to boundary synchronization issues. Moreover, E2E architecture limits the possibilities of re-introducing recovered OOVs into the ASR system. This second approach also made use of a Chinese restaurant process-based clustering, defined in a principled way for the task of clustering OOV candidates with probabilistic pronunciations.

In the final approach, we propose a new speller architecture with a capability of learning OOV representations together with the word predicting network training. Speller architecture brings together the principled combination of two granularities observed in a hybrid decoding system with the power of E2E training. Speller architecture also does not rely on an external clustering procedure for OOV recovery, and is able to find meaningful groups of OOVs during training with multiple OOV embeddings. Using the speller architecture also shows improvements both in terms of ASR and OOV recovery.

7.2 Future Directions

Even though automatic speech recognition (ASR) works “well enough” now in applications like Google, Siri etc., OOV post-processing can make the applications more user-oriented. For example, it can help a customer select their own target words from a list suggested by an OOV recovery system. These suggestions can change over time with user needs and their personal vocabulary development.

Speaking of speller architecture in particular, one of the immediate uses would be its integration with NLP tasks. Spelling-aware embeddings can be useful in other speech processing applications, such as machine translation or slot-filling. There is a theoretical possibility of combining spelling-aware training with pre-trained models to improve their performance on out-of domain tasks. Multiple OOV embeddings trained to predict different classes of words can be potentially used for part-of-speech tagging, as they clearly contain information about morphemes, their spelling and the grammatical classes they represent.

In the context of multi-linguality and code switching, OOVs from one language may be IVs in another language, and cross-utilizing the speller-aware embeddings might help recover foreign words in a multilingual ASR environment.

A less immediate but promising extension of the speller architecture would be joint training of a speech recognizer on two or more different speech granularities, with information from all of them feeding into and informing others. This would be a principled speech representation approach, explainable and versatile for various tasks.

Bibliography

[Aguilar et al., 2020] Aguilar, G., McCann, B., Niu, T., Rajani, N., Keskar, N., and Solorio, T. (2020). Char2subword: Extending the subword embedding space using robust character compositionality. *arXiv*.

[Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. M. (2015). Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *NIPS*.

[Bisani and Ney, 2005] Bisani, M. and Ney, H. (2005). Open vocabulary speech recognition with flat hybrid models. *Proceedings of INTERSPEECH*.

[Bisani and Ney, 2008] Bisani, M. and Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication 50*, pages 434–451.

[Bourlard and Morgan, 1994] Bourlard, H. and Morgan, N. (1994). *Connectionist Speech Recognition: A Hybrid Approach*. Springer New York.

[Can and Saraclar, 2011] Can, D. and Saraclar, M. (2011). Lattice indexing for spoken term detection. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8):2338–2347.

[Chan et al., 2016] Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964.

[Chen et al., 2022] Chen, S., Wang, C., Chen, Z., Wu, Y., Liu, S., Chen, Z., Li, J., Kanda, N., Yoshioka, T., Xiao, X., Wu, J., Zhou, L., Ren, S., Qian, Y., Qian, Y., Zeng, M., and Wei, F. (2022). Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *ArXiv*, abs/2110.13900.

[Chorowski et al., 2015a] Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015a). Attention-based models for speech recognition. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

[Chorowski et al., 2015b] Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015b). Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585.

[Chung et al., 2021] Chung, Y.-A., Zhang, Y., Han, W., Chiu, C.-C., Qin, J., Pang, R., and Wu, Y. (2021). w2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training. *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 244–250.

[Davis and Mermelstein, 1980] Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken se. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv.*

[Egorova and Burget, 2018] Egorova, E. and Burget, L. (2018). Out-of-vocabulary word recovery using fst-based subword unit clustering in a hybrid asr system. In *Proceedings of ICASSP 2018*, pages 5919–5923. IEEE Signal Processing Society.

[Egorova et al., 2022] Egorova, E., Vydana, H. K., Burget, L., and Černocký, J. H. (2022). Spelling-aware word-based end-to-end asr. *IEEE Signal Processing Letters*, 29:1729–1733.

[Egorova et al., 2021] Egorova, E., Vydana, K. H., Burget, L., and Černocký, J. (2021). Out-of-vocabulary words detection with attention and ctc alignments in an end-to-end asr system. In *Proceedings Interspeech 2021*, volume 8, pages 2901–2905. International Speech Communication Association.

[El Boukkouri et al., 2020] El Boukkouri, H., Ferret, O., Lavergne, T., Noji, H., Zweigenbaum, P., and Tsujii, J. (2020). CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

[Gage, 1994] Gage, P. (1994). A new algorithm for data compression. *The C Users Journal archive*, 12:23–38.

[Gales, 1999] Gales, M. J. F. (1999). Semi-tied covariance matrices for hidden markov models. *IEEE Trans. Speech Audio Process.*, 7:272–281.

[Goyal et al., 2017] Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR*, abs/1706.02677.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32/2 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China. PMLR.

[Gulati et al., 2020] Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented Transformer for Speech Recognition. *Proc. Interspeech.*

[Hannemann et al., 2010] Hannemann, M., Kombrink, S., Karafiát, M., and Burget, L. (2010). Similarity Scoring for Recognizing Repeated Out-of-Vocabulary Words. *Proceedings of INTERSPEECH*, page 897–900.

[Hartmann et al., 2013] Hartmann, W., Roy, A., Lamel, L., and Gauvain, J.-L. (2013). Acoustic unit discovery and pronunciation generation from a grapheme-based lexicon. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*, pages 380–385.

[Hermansky, 1990] Hermansky, H. (1990). Perceptual linear predictive (plp) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752.

[Heymann et al., 2014] Heymann, J., Walter, O., Haeb-Umbach, R., and Raj, B. (2014). Iterative Bayesian Word Segmentation for Unsupervised Vocabulary Discovery from Phoneme Lattices. *Proceedings of ICASSP 2014*.

[Hinton et al., 2012] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29:82–97.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Hsiao et al., 2020] Hsiao, R., Can, D., Ng, T., Travadi, R., and Ghoshal, A. (2020). Online automatic speech recognition with listen, attend and spell model. *IEEE Signal Processing Letters*, 27:1889–1893.

[Inan et al., 2017] Inan, H., Khosravi, K., and Socher, R. (2017). Tying word vectors and word classifiers: A loss framework for language modeling. In *ICLR (Poster)*.

[Jaegle et al., 2021] Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. (2021). Perceiver: General perception with iterative attention. *arXiv*.

[Jelinek, 1976] Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556.

[Juang et al., 1986] Juang, B.-H., Levinson, S. E., and Sondhi, M. M. (1986). Maximum likelihood estimation for multivariate mixture observations of markov chains. *IEEE Trans. Inf. Theory*, 32:307–309.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kneser and Ney, 1995] Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.

[Kombrink et al., 2010a] Kombrink, S., Hannemann, M., and Burget, L. (2010a). Out-of-Vocabulary Word Detection and Beyond. In *ECML PKDD 2010 Proceedings*, pages 1–8.

[Kombrink et al., 2010b] Kombrink, S., Hannemann, M., Burget, L., and Heřmanský, H. (2010b). Recovery of rare words in lecture speech. In *Proc. Text, Speech and Dialogue 2010*, volume 9, pages 330–337. Springer Verlag.

[Kudo and Richardson, 2018] Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

[Lee et al., 2015] Lee, L.-s., Glass, J., Lee, H.-y., and Chan, C.-a. (2015). Spoken content retrieval—beyond cascading speech recognition with text retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(9):1389–1420.

[Li et al., 2018] Li, J., Ye, G., Das, A., Zhao, R., and Gong, Y. (2018). Advancing Acoustic-to-Word CTC Model. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5794–5798.

[Li et al., 2017] Li, J., Ye, G., Zhao, R., Droppo, J., and Gong, Y. (2017). Acoustic-to-Word Model without OOV. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 111–117.

[Lin et al., 2012] Lin, Y., Michel, J.-B., Aiden, E., Orwant, J., Brockman, W., and Petrov, S. (2012). Syntactic Annotations for the Google Books Ngram Corpus. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, page 169–174.

[Mielke and Eisner, 2018] Mielke, S. and Eisner, J. (2018). Spell Once, Summon Anywhere: A Two-Level Open-Vocabulary Language Model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 9, pages 1045–1048. International Speech Communication Association.

[Mohri et al., 2002] Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88.

[Mohri et al., 2008] Mohri, M., Pereira, F., and Riley, M. (2008). Speech Recognition with Weighted Finite-State Transducers. *Handbook on Speech Processing and Speech Communication.*

[Panayotov et al., 2015] Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

[Povey et al., 2011] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemanna, M., Motlícek, P., Qian, Y., Schwarz, P., Silovský, J., Stemmer, G., and Vesel´y, K. (2011). The Kaldi Speech Recognition Toolkit. *Proceedings of ASRU.*

[Povey et al., 2012] Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motlíček, P., Qian, Y., Riedhammer, K., Vesel, K., and Vu, T. (2012). Generating exact lattices in the wfst framework. *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on.*

[Qin and Rudnicky, 2013] Qin, L. and Rudnicky, A. (2013). Learning Better Lexical Properties for Recurrent OOV Words. *Proceedings of ASRU 2013.*

[Rand, 1971] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850.

[Santos and Embrechts, 2009] Santos, J. M. and Embrechts, M. (2009). On the use of the adjusted rand index as a metric for evaluating supervised classification. In *Artificial Neural Networks – ICANN 2009*, pages 175–184.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

[Shaik et al., 2011] Shaik, M. A. B., Mousa, A. E.-D., Schlüter, R., and Ney, H. (2011). Hybrid Language Models Using Mixed Types of Sub-Lexical Units for Open Vocabulary German LVCSR. In *Proc. Interspeech 2011*, pages 1441–1444.

[Soltau et al., 2017] Soltau, H., Liao, H., and Sak, H. (2017). Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary Speech Recognition. *Proceedings of INTERSPEECH*.

[Szőke, 2010] Szőke, I. (2010). *Hybrid word-subword spoken term detection*. PhD thesis, Brno University of Technology.

[Thomas et al., 2019] Thomas, S., Audhkhasi, K., Tüuske, Z., Huang, Y., and Picheny, M. (2019). Detection and Recovery of OOVs for Improved English Broadcast News Captioning. *Proceedings of INTERSPEECH*.

[Vanhainen and Salvi, 2014] Vanhainen, N. and Salvi, G. (2014). Pattern discovery in continuous speech using block diagonal infinite hmm. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 3719–3723.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Watanabe et al., 2017] Watanabe, S., Hori, T., Kim, S., Hershey, J. R., and Hayashi, T. (2017). Hybrid CTC/Attention Architecture for End-to-End Speech Recognition. *Journal of Selected Topics in Signal Processing*.

[Yazgan and Saraclar, 2004] Yazgan, A. and Saraclar, M. (2004). Hybrid Language Models for Out of Vocabulary Word Detection in Large Vocabulary Conversational Speech Recognition. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–745.

[Yu and Seide, 2004] Yu, P. and Seide, F. (2004). A Hybrid Word/Phoneme-Based Approach for Improved Vocabulary-Independent Search in Spontaneous Speech. *INTERSPEECH 2004 - ICSLP, 8th International Conference on Spoken Language Processing*.

[Zeyer et al., 2018] Zeyer, A., Irie, K., Schlüter, R., and Ney, H. (2018). Improved training of end-to-end attention models for speech recognition. *Interspeech*.

[Zhang et al., 2020] Zhang, X., Povey, D., and Khudanpur, S. (2020). OOV recovery with efficient 2nd pass decoding and open-vocabulary word-level RNNLM rescoring for hybrid ASR. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6334–6338.

[Zhang et al., 2014] Zhang, X., Trmal, J., Povey, D., and Khudanpur, S. (2014). Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 215–219.

[Zhou et al., 2021] Zhou, W., Zeineldeen, M., Zheng, Z., Schlüter, R., and Ney, H. (2021). Acoustic Data-Driven Subword Modeling for End-to-End Speech Recognition. In *Proc. Interspeech 2021*, pages 2886–2890.

Appendix A

Simulated OOV list for LibriSpeech dataset

Numbers show the number of occurrences of the OOV in the train-clean-360 dataset; the list is sorted according to these occurrences.

replied 1302
captain 879
gentleman 535
exclaimed 531
scarcely 437
wished 425
colonel 365
madame 331
carriage 297
anxious 296
courage 288
evidently 282
servant 281
gentlemen 278
seized 272
nearer 266
servants 256
possessed 252
succeeded 246
remarked 241
sorrow 240
delight 240
nevertheless 238
handsome 237
divine 233
peculiar 228
splendid 221
mistress 220
countenance 218
virtue 217
moreover 215
companions 213
monsieur 212
ceased 212
proceeded 210
wholly 207
delighted 201
whence 197
affection 197
fellows 196
begged 196
escaped 195
seldom 191
whilst 190
dignity 190
kindly 186
behold 186
plainly 185
temper 182
savage 182
inhabitants 181
uttered 178
eagerly 176
maiden 175
lighted 175
earnest 175
kindness 174
trembling 173
utterly 172
immense 169
melancholy 165
admiration 163
squire 162
departed 161
magnificent 160
hastily 159
conscience 159
commanded 159
rendered 154
majesty 154
solemn 153
resumed 149
astonishment 145
throne 144
gratitude 144
triumph 143
hitherto 143
beheld 143
descended 142
compelled 141
likewise 140
fierce 139
duchess 139
deserted 139
astonished 136
chiefly 135
pursued 133
doubtless 133
concealed 131
beasts 131

marched 126
fancied 126
hastened 125
furnished 125
beloved 124
ambition 123
voyage 120
fastened 120
multitude 119
pretended 118
sailed 115
parted 115
englishman 115
dearest 115
ventured 114
thither 114
exceedingly 114
notwithstanding 112
passions 111
disposed 111
scarce 109
virtues 108
misfortune 108
countess 107
contented 105
wretched 104
mingled 104
gloomy 103
heartily 102
condemned 102
preserved 101
tenderness 100
quarrel 100
belonging 100
farewell 99
anxiously 99
vengeance 97
thence 97
procession 97
distinctly 96
betrayed 96
trembled 95
princes 95
pierre 95
hither 95
cunning 95
jewels 94
earnestly 93
awakened 93
wherein 92
obeyed 92
fortunes 92
steamer 91
feeble 91
commenced 91
wherefore 90
solitude 90
sentiments 90
savages 90
regiment 90
florence 90
philip 89
gladly 88
alarmed 88
frightful 87
disagreeable 87
deceived 87
constance 87
yonder 85
indignation 85
whereupon 84
repose 84
pleasures 84
rejoined 83
dwelling 82
whither 81
supposing 81
murmur 81
devotion 81
amiable 81
roused 79
exquisite 79
cheerfully 78
yielded 77
disgrace 77
crimson 77
consented 77
tenderly 76
solemnly 75
politeness 75
graceful 75
bestowed 75
eustace 74
sufferings 73
speedily 73
confessed 73
wrought 72
careless 72
advancing 71
serpent 70
seeming 70
prevailed 70
conspicuous 70
summoned 69
picturesque 69
afforded 69
descend 68
boldly 68
tremble 66
thyself 65
horsemen 65
gallant 65
furnish 65
workmen 64
mademoiselle 64
justly 64
jasper 64
impatience 64
clergyman 64
nobles 63
exhibited 63
erected 63
destined 63
contrived 63
conquered 63
thereupon 62
rejoiced 62
herbert 62
conveyed 62
unworthy 61
robbers 61
guarded 61
grieved 61
delicacy 61
crowned 61
cherished 61
wretch 60
glittering 60
frenchman 60
errand 60
decidedly 59
cultivated 59
propriety 58
nobility 58

clothed 58
onward 57
dreadfully 57
commanding 57
westward 56
venerable 56
falsehood 56
eagerness 56
sublime 55
reproach 55
reigned 55
prudence 55
marguerite 55
hounds 55
descending 55
coachman 55
stately 54
robber 54
bridle 54
sweetness 53
reverence 53
perceiving 53
monarch 53
misfortunes 53
interposed 53
bravely 53
apprehension 53
tumult 52
therein 52
ascended 52
waistcoat 51
illustrious 51
hereafter 51
hasten 51
endeavoured 51
despised 51
curate 51
respectfully 50
practised 50
carriages 50
watchful 49
northward 49
mutton 49
maidens 49
henceforth 49
headlong 49
bestow 49
warmly 48
throng 48
tempest 48
occasioned 48
napoleon 48
innumerable 48
desirous 48
charms 48
chanced 48
bertram 48
ardent 48
reckoned 47
fancies 47
excellency 47
countrymen 47
adorned 47
uneasiness 46
torrent 46
southward 46
retreated 46
reginald 46
indignant 46
hospitality 46
eloquence 46
withered 45
stooping 45
sorrows 45
proprietor 45
elinor 45
dainty 45
carlyle 45
springing 44
procured 44
negroes 44
malice 44
joyous 44
indulged 44
forthwith 44
barbarous 44
assented 44
amidst 44
affections 44
triumphant 43
treachery 43
sullen 43
softened 43
saluted 43
rejoicing 43
mournful 43
henrietta 43
haughty 43
gallop 43
courtiers 43
augustus 43
attentively 43
trifling 42
tidings 42
sprung 42
plunder 42
eastward 42
devoured 42
defiance 42
whereof 41
vainly 41
superstition 41
singularly 41
richly 41
pierced 41
blushing 41
admiring 41
restrained 40
relics 40
pleasantly 40
indignantly 40
exalted 40
endowed 40
ecclesiastical 40
earnestness 40
carelessly 40
wickedness 39
weariness 39
thrice 39
slumber 39
prostrate 39
personage 39
nobleman 39
labours 39
insensible 39
horace 39
doings 39
delights 39
declaring 39
composure 39
alfred 39
uttering 38
suspicions 38
resolute 38

rapidity 38
overtook 38
obstinate 38
lizzie 38
heaped 38
draught 38
dashing 38
convent 38
respecting 37
renders 37
principally 37
possesses 37
manhood 37
humbly 37
eloquent 37
confounded 37
alighted 37
vexation 36
silken 36
serpents 36
merrily 36
lodgings 36
inasmuch 36
galloped 36
forlorn 36
englishmen 36
cornelius 36
consternation 36
bessie 36
beauties 36
banished 36
attentions 36
assuredly 36
assent 36
ascertained 36
serene 35
resembling 35
professed 35
penetrated 35
obscurity 35
loveliness 35
kneeling 35
glided 35
cruelly 35
coolness 35
bustle 35
allusion 35
verily 34
temperance 34
seizing 34
revived 34
modesty 34
magnificence 34
lodged 34
homeward 34
homage 34
feebly 34
fearfully 34
excepting 34
discharged 34
boughs 34
adventurer 34
undertook 33
thickly 33
strove 33
rector 33
recollected 33
rascal 33
possessing 33
peeping 33
peculiarly 33
eminence 33
displeasure 33
despatched 33
confided 33
conceit 33
ascending 33
ascend 33
afresh 33
unbroken 32
sorrowful 32
renown 32
perished 32
mortals 32
industrious 32
hesitating 32
exerted 32
desiring 32
conferred 32
clearness 32
affords 32
traversed 31
murmuring 31
maxims 31
manifested 31
heathen 31
enveloped 31
dismounted 31
chivalry 31
bravery 31
trifles 30
surrendered 30
statesmen 30
splendour 30
rudely 30
quarrels 30
mortimer 30
haunts 30
comprehended 30
betwixt 30
antiquity 30
summons 29
sombre 29
seclusion 29
schooner 29
regiments 29
pitied 29
noiselessly 29
horseman 29
coolly 29
boasted 29
apprehensions 29
admirably 29
tremulous 28
swiftness 28
swelled 28
sweetly 28
subsided 28
soothed 28
quarrelled 28
perpetually 28
pecuniary 28
parson 28
natures 28
insolent 28
impelled 28
gratified 28
fortified 28
flattered 28
extinguished 28
exertions 28
conversed 28
conqueror 28
churchyard 28

cheerfulness 28
amusements 28
alternately 28
toiled 27
solicitude 27
redoubled 27
perils 27
obstinacy 27
marquis 27
insolence 27
girdle 27
footman 27
divinity 27
cordial 27
uncommonly 26
traitor 26
sundry 26
rapture 26
quickened 26
muslin 26
marian 26
interfered 26
hastening 26
habitation 26
graces 26
forgetfulness 26
firmness 26
exquisitely 26
envied 26
encamped 26
charley 26
carelessness 26
benevolence 26
wearied 25
therewith 25
resided 25
repast 25
presided 25
overpowered 25
joyfully 25
incessantly 25
impracticable 25
groves 25
genial 25
ejaculated 25
drooping 25
contemptuously 25
bridegroom 25
bertha 25
asunder 25
apparition 25
tolerably 24
terrors 24
seamen 24
scanty 24
plaintive 24
personages 24
pallid 24
musing 24
multitudes 24
lustre 24
impudent 24
implored 24
hollows 24
hoisted 24
grandeur 24
gentleness 24
entreaties 24
effectually 24
discontent 24
declares 24
dearer 24
dazzled 24
concealing 24
chevalier 24
avarice 24
audacity 24
arrayed 24
approbation 24
amounted 24
adventurers 24
thronged 23
solemnity 23
ornamented 23
languid 23
impetuous 23
fugitive 23
frenchmen 23
flocks 23
exclaiming 23
disconcerted 23
boldness 23
alacrity 23
vehemence 22
unobserved 22
unmoved 22
thickets 22
sympathies 22
shrieks 22
schoolmaster 22
rosalie 22
poetical 22
mourned 22
merriment 22
mended 22
fringed 22
fondness 22
foaming 22
ferocity 22
endeavouring 22
dick’s 22
despairing 22
courier 22
cordially 22
busily 22
baronet 22
wanderings 21
trodden 21
spaniard 21
sorely 21
resounded 21
recollections 21
proffered 21
mournfully 21
meditated 21
marjorie 21
lamented 21
kindled 21
irresistibly 21
implacable 21
imperfectly 21
habitually 21
frankness 21
flourishing 21
flattery 21
ferdinand 21
despatch 21
desolation 21
assailed 21
affectation 21
venetian 20
tottering 20
steamers 20
speculations 20

revolted 20
pretensions 20
pauline 20
mightily 20
matilda 20
indifferently 20
imperious 20
hospitable 20
heiress 20
greedily 20
extravagance 20
enmity 20
counsels 20
contrivance 20
confound 20
wreaths 19
thereon 19
signified 19
salutation 19
myriads 19
impertinent 19
heretofore 19
harriet 19
godfrey 19
discontented 19
commended 19
busied 19
brutes 19
visage 18
ursula 18
undulating 18
treading 18
sovereigns 18
scruples 18
rebuke 18
needful 18
madeleine 18
loosed 18
livery 18
heedless 18
freshness 18
despotism 18
desertion 18
cowardice 18
coldness 18
betrothed 18
beholding 18
attired 18
antagonist 18
agreeably 18
adoration 18
accosted 18
wherewith 17
thundering 17
therefrom 17
surmounted 17
successively 17
smitten 17
reproof 17
perchance 17
murmurs 17
meekly 17
jacques 17
irishman 17
gleams 17
gertrude 17
forsaken 17
familiarly 17
courteously 17
consoled 17
conscientiously 17
billows 17
admirers 17
wilful 16
unheeded 16
studded 16
straggling 16
steadiness 16
saluting 16
repelled 16
prettily 16
mortification 16
messrs 16
indolent 16
idleness 16
graciously 16
gallantry 16
gaiety 16
flitting 16
feigned 16
exasperated 16
evermore 16
devouring 16
claret 16
blanche 16
availed 16
ascribed 16
affectionately 16
wedded 15
vivacity 15
unmolested 15
unhappily 15
toiling 15
rascals 15
quarrelling 15
mingling 15
methinks 15
katharine 15
ingratitude 15
imparted 15
imitated 15
handsomely 15
furnishes 15
frederic 15
feasted 15
exclaim 15
exaltation 15
epistle 15
enjoined 15
dwells 15
disgraced 15
descends 15
decked 15
cromwell 15
contended 15
consecrated 15
caresses 15
caprice 15
ardently 15
uncouth 14
sufficed 14
reproached 14
remarking 14
pitying 14
pervaded 14
peaceably 14
overhanging 14
numberless 14
muriel 14
miseries 14
leaden 14
lamentations 14
incredulity 14
imploring 14

hester 14
glories 14
follies 14
evinced 14
edifice 14
cometh 14
aforesaid 14
unwillingly 13
unawares 13
timidity 13
sixpence 13
reverently 13
resuming 13
musket 13
menaced 13
maurice 13
kinsman 13
grecian 13
flocked 13
finery 13
fetters 13
exultation 13
exclamations 13
entertainments 13
diffused 13
devoutly 13
detested 13
despondency 13
crowning 13
clergymen 13
blotted 13
vagabond 12
ulysses 12
slumbering 12
shewed 12
seizes 12
sculptured 12
relapsed 12
protestations 12
orator 12
levity 12
languidly 12
lances 12
glimmering 12
damsels 12
cunningly 12
confiding 12
cheerily 12
antony 12
acquiescence 12
abashed 12
triumphs 11
swarms 11
sauntering 11
rogues 11
pervading 11
peopled 11
penitent 11
mildred 11
gaston 11
frolic 11
forgetful 11
fearlessly 11
deportment 11
commencing 11
comely 11
bonaparte 11
avowed 11
aright 11
ardour 11
abounded 11
swarthy 10
ripened 10
muskets 10
mollie 10
moistened 10
magnanimity 10
intimated 10
gallantly 10
chieftain 10
blundering 10
archibald 10
annals 10
affording 10
undaunted 9
triumphed 9
tradesmen 9
slackened 9
recurred 9
rebuked 9
journeying 9
jennie 9
hector 9
forbearance 9
fierceness 9
disdained 9
cornelia 9
consigned 9
chafed 9
boudoir 9
bestowing 9
skilfully 8
protruded 8
oratory 8
moorish 8
margery 8
josiah 8
grandly 8
exacted 8
brightening 8
awaking 8
amounting 8
philippa 7
oration 7
mistook 7
minutely 7
exulted 7
esther 7
enlivened 7
drapery 7
divinely 7
delighting 7
commences 7
bequeathed 7
begotten 7
beatrice 7
acquiesced 7
wordsworth 6
voltaire 6
thirsting 6
tennyson 6
swells 6
prussian 6
helen's 6
growths 6
geoffrey 6
foreseeing 6
filial 6
deeming 6
credulity 6
bespoke 6
theodore 5
pricking 5
lottie 5

dragoons 5
divested 5
convulsively 5
cleopatra 5
accursed 5
thunders 4
reclined 4
prescott 4
percival 4
messer 4
josephine 4
francois 4
francesco 4
coleridge 4
cicero 4
agatha 4
abounding 4
trembles 3
roderick 3
odysseus 3
impartially 3
eugenia 3
camilla 3
thackeray 2
rowland 2
johannes 2
jimmie 2
gustave 2
exclaims 2
clarissa 2
winifred 1
rosalind 1
prussia 1
nellie 1
midshipman 1
lucien 1
lilian 1
jeanne 1
herrick 1
gladys 1
cecilia 1
bertie 1
thereto 0
signora 0
johnnie 0
godwin 0
desmond 0

Appendix B

Examples of H, C, L, and G graphs in a hybrid ASR system

The following simple example helps visualize how the FSTs repesenting different components of a hybrid ASR system look like.

Assume that we have a set of just 2 phonemes and 3 words, 2 of them being homophones (words that have the same pronunciation but different graphemic representation). Here is a sample dictionary:

I	ay	
ice	ay	s
eye	ay	

A small example of a training text might look like the following:

ice eye
I eye
ice

A bigram language model, estimated from this small training corpus by Kneser-Ney smoothing [Kneser and Ney, 1995], will then look like shown in Table B.1 in ARPA format[1] (<s> is sentence start and </s> is sentence end symbols).

The LM and the dictionary provide all the needed information for the creation of C, L, and G graphs which will then be composed with H that represents acoustic states.

A G graph for the LM in Table B.1 is shown in Figure B.1, #0 being the vocabulary disambiguation symbol that substitutes empty (<eps>) input symbol in order to keep the graph determinizable. Note that G graph contains valid paths for all 1-grams and 2-grams in the LM. Also, there are back-off paths to node 1 with input disambiguation symbols that allow for stacking 1-grams and 2-grams together.

Figure B.2 shows lexicon transducer (L) for this small example. Note that words with the same pronunciations have acquired disambiguation symbols at the end of each repeating transcription path, so now there paths of input strings *ay* #1 for output word *I* and *ay* #2 for word *eye*.

[1] http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html

```
\data\
ngram 1=5
ngram 2=6

\1-grams:
-0.4771213   </s>
-99          <s>    -0.39794
-0.7781513   I      -0.30103
-0.60206     eye    -0.4771212
-0.60206     ice    -0.3010301

\2-grams:
-0.5740313   <s>    I
-0.30103     <s>    ice
-0.20412     I      eye
-0.1091445   eye    </s>
-0.3802112   ice    </s>
-0.4259687   ice    eye

\end\
```

Table B.1: Example LM in ARPA format

Context dependency graph C can be viewed in Figure B.4. Even for the small example we have, note how big it is as it expands the existing phonemes into the context-dependent variants.

And finally, a part of H transducer is depicted in Figure B.3. Output labels are context-dependent phonemes and input labels are transition-ids encoding phone, HMM state index withing the triphone, PDF ID and the index of outgoing arc from the node in the second field.

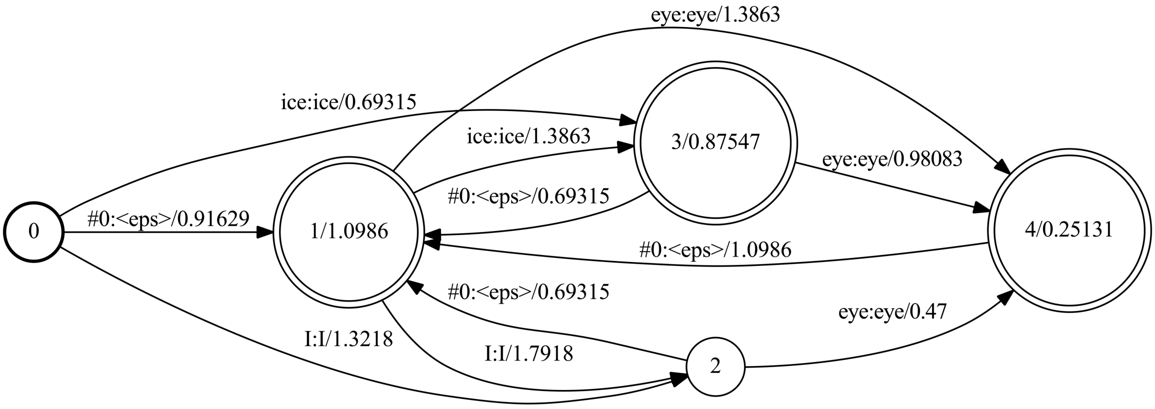


Figure B.1: Example of a grammar transducer G.

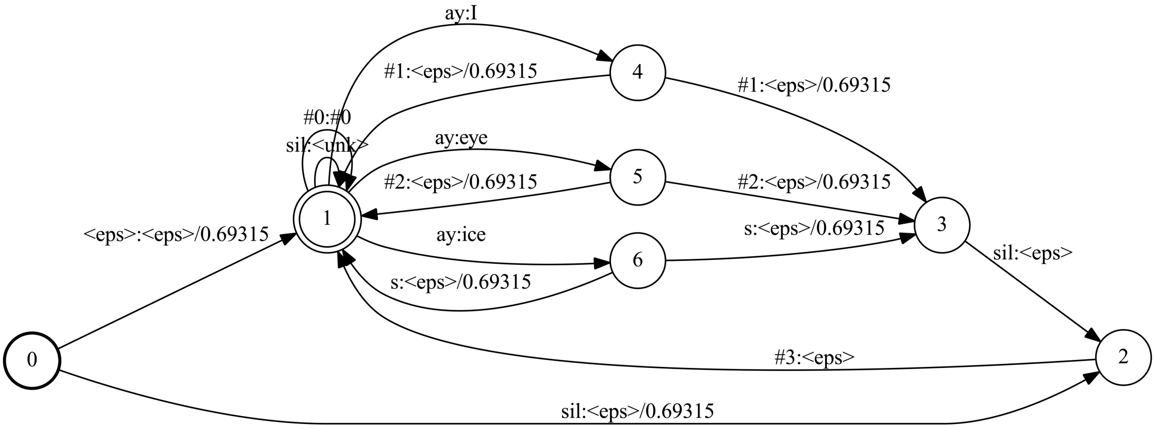


Figure B.2: Example of a lexicon transducer L.

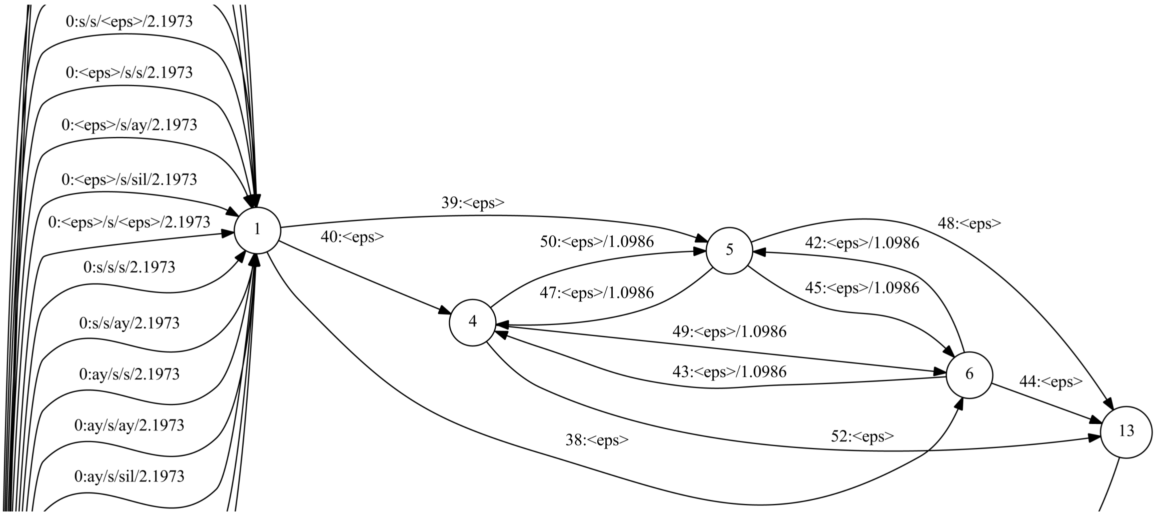


Figure B.3: Example of an HMM definition transducer H.

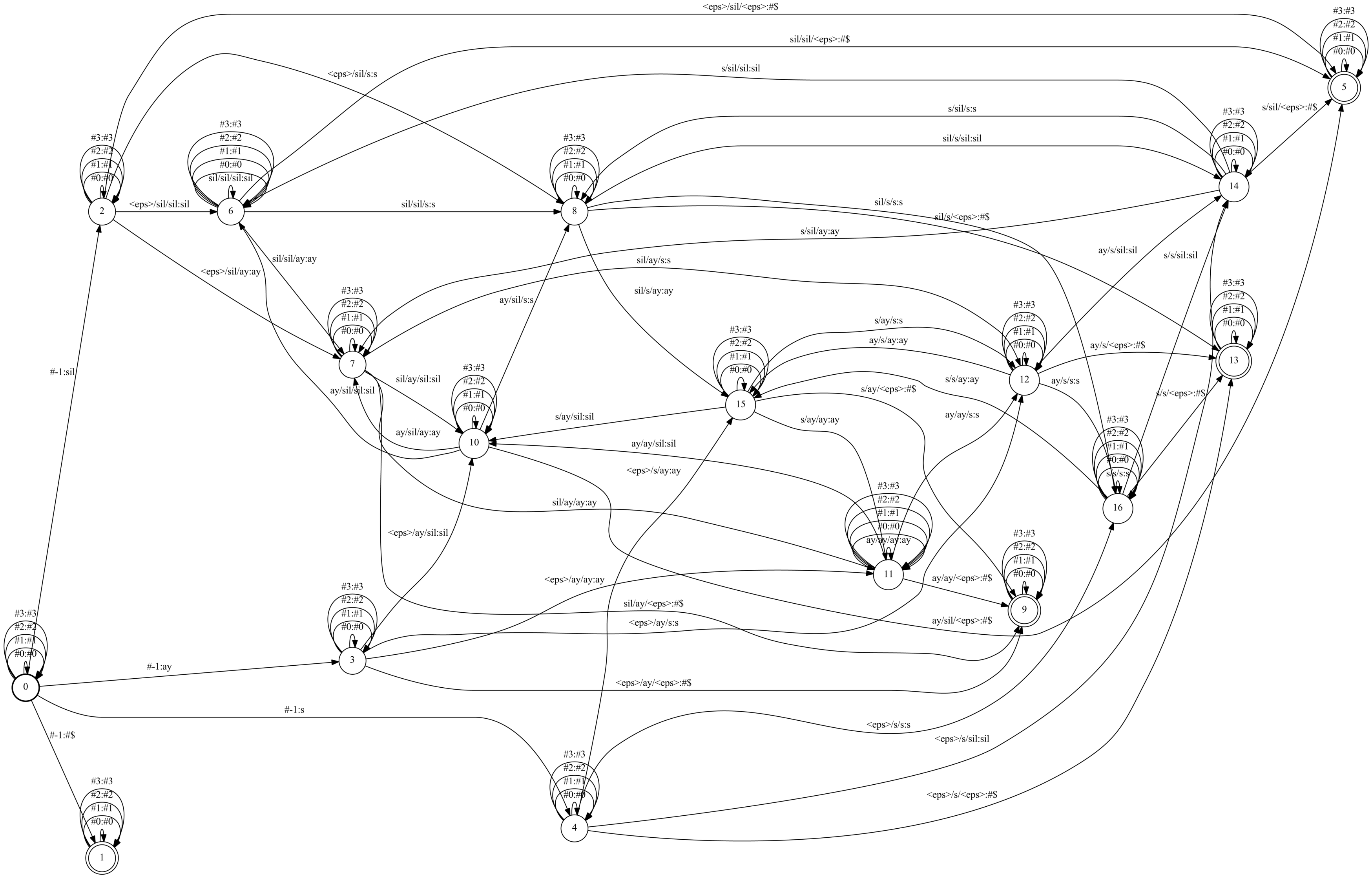


Figure B.4: Example of a context dependency transducer C.