eman ta zabal zazu

Universidad del País Vasco | Euskal Herriko Unibertsitatea

Data-Driven Lexicon Generation for ASR

Author: Maria Obedkova
Advisors: Dr. Eva Navas, PhD

Dr. Ibon Saratxaga, PhD

hap/lap

Hizkuntzaren Azterketa eta Prozesamendua
Language Analysis and Processing

Final Thesis

September 2019

Departments: Computer Systems and Languages, Computational Architectures and Technologies, Computational Science and Artificial Intelligence, Basque Language and Communication, Communications Engineering.

Abstract

In ASR systems, dictionaries are usually used to describe pronunciations of words in a language. These dictionaries are typically hand-crafted by linguists. One of the most significant drawbacks of dictionaries created this way is that linguistically motivated pronunciations are not necessarily the optimal ones for ASR. The goal of this research was to explore approaches of data-driven pronunciation generation for ASR. We investigated several approaches of lexicon generation and implemented the completely new data-driven solution based on the pronunciation clustering. We proposed an approach for feature extraction and researched different unsupervised methods for pronunciation clustering. We evaluated the proposed approach and compared it with the current hand-crafted dictionary. The proposed data-driven approach could beat the established baselines but underperformed in comparison to the hand-crafted dictionary which could be due to unsatisfactory features extracted from data or insufficient fine tuning.

Contents

List of Figures

List of Tables

1 Introduction

Automatic Speech Recognition (ASR) is a very important field of speech processing which allows understanding of human speech by computers. The last two decades showed a rise in interest in speech recognition and research in this field has been intensively carried out worldwide. This interest resulted in big improvement of speech recognition systems due to various advances in speech signal processing, different speech recognition algorithms and architectures and diverse data modelling techniques. All this progress in the field of ASR enabled to successfully use speech recognition in many different applications that vary in their difficulty from simple isolated word recognition to complex large vocabulary speech recognition (Furui, 1986; Dahl et al., 2011). However, even such noticeable improvements in speech recognition field cannot eliminate all problems that human speech poses to a speech recognizer. There still exist several tasks for ASR that do not gain the desired performance such as real-time speech recognition, speech recognition of noisy input and recognition of accented speech (Radha and Vimala, 2012).

For building an ASR model, a pronunciation dictionary should be available where every word has a corresponding pronunciation sequence. The pronunciation dictionary (also called lexicon) allows a speech recognizer to build the link between the audio signal and its phonetic representation. The most common way to obtain the ASR lexicon is to hand-craft it using expert knowledge.

1.1 Motivation

Having the human factor involved, lexicons cannot be considered as a fully-reliable source of phonetic knowledge due to some level of perception subjectivity and discrepancy along the lexicon. Even having several linguists working on lexicon creation does not solve the problem of subjectivity since there exists some amount of disagreement among lexicon entries created by different linguists which is hard to resolve. These inconsistencies of a hand-crafted dictionary may have an impact on ASR system performance because speech recognition systems require quite a consistent lexicon to perform well. Moreover, linguistically motivated pronunciations do not necessarily suit the best for ASR modelling. The linguistic knowledge that is used for pronunciation creation does not fit the real-world picture of unarticulated, not very clear and full of disfluencies speech.

One of the possible ways to escape from linguistically motivated pronunciations in a lexicon is to use automatically generated lexicons. ASR lexicons that are generated automatically from data are more motivated by statistics of the data and do not involve linguistic knowledge. Potentially, this kind of lexicons is very powerful and can provide information that is very valuable in acoustic modelling for a speech recognizer. The work conducted in the field of automatic pronunciation generation is extensive and covers many different applications starting from generating pronunciation entries for out-of-vocabulary words and finishing with the creation of a complete lexicon from scratch. Most of the proposed methods use grapheme-to-phoneme conversion for lexicon generation (Taylor, 2005; Bisani and Ney, 2008; Rao et al., 2015) which makes pronunciation generation system less language independent since it relies on a strong relation between phonemes and graphemes. Some works propose unsupervised techniques of pronunciation generation that do not need any initial supervision for lexicon creation and can generate word pronunciations based on data provided (Takahashi et al., 2016; Hartmann et al., 2013). These systems usually concentrate not only on the phonetic sequence generation for a word but also on the derivation of minimal acoustic units that are used in these sequences. However, according to our knowledge, most of the proposed methods in the literature consider the generation of the best pronunciation for a word in a lexicon without possibility to have multiple pronunciations. This is not the case for most words due to various existing pronunciations for a word in a language or even different parts of speech.

Additionally, this thesis was done as a part of an internship at Sony Europe. The thesis motivation was framed by the industrial need for investigations in the pronunciation generation field, especially in the area of data-driven improvement of ASR dictionaries.

1.2 Goal

Considering the already carried out research in the pronunciation generation domain, we propose to use another approach for data-driven lexicon generation that was not investigated so far in the pronunciation generation field. This approach is also fully data-driven and derives phonetic sequences for lexicon entries directly from audio data. Moreover, it can generate multiple meaningful pronunciations for a word in a chosen vocabulary. As the main application for this approach, we see the regeneration of a lexicon so that the ASR system could benefit from adapting the lexicon generation process to a particular system, improving overall ASR performance. An additional application can be out-of-vocabulary word handling.

This work is structured as follows. In chapter 2, we provide the theoretical background on ASR field and how lexicons are used in ASR systems. Chapter 3 gives an overview on the conducted research on pronunciation generation. In chapter 4, we outline the proposed lexicon generation approach and describe the whole experiment design. Chapter 5 introduces the corpus that we chose for our experiments. In chapters 6 and 7, we discuss in detail the proposed system for data-driven pronunciation generation. In chapter 6, we explain the necessary feature extraction procedure for pronunciation clustering which is discussed further in chapter 7. Chapter 8 considers different systems that can be used for comparison with the proposed approach such as various baseline and benchmark systems. In chapter 9, we present our experiments on data-driven pronunciation generation and discuss the obtained results. In conclusion, we review the work that has been done and consider possible future work.

2 Theoretical Background

Pronunciation dictionaries play a big role in speech recognition systems giving the opportunity to model the acoustics of phonemes. In this chapter, we introduce the main theory on how automatic speech recognition works and how lexicons are involved in speech recognition systems.

In section 2.1, we introduce the notion of automatic speech recognition system, its architecture and performance evaluation. Section 2.2 provides general information on neural networks and describes recurrent neural networks that are widely used for speech recognition. In section 2.3, we give a brief overview of approaches for ASR. Finally, section 2.4 is devoted to lexicons, their role in ASR systems and the ways they can be created.

2.1 ASR

Automatic Speech Recognition (ASR) is one of the research areas in speech processing and its task is to convert a speech signal to a sequence of words (Karpagavalli and Chandra, 2016). The goal of the ASR system is to predict the most likely sequence of words that exists in some language for the given observation. Thus, the task of ASR can be defined with the following formula:

$$\hat{W} = \arg\max_{W} P(W|O) \tag{1}$$

where W is a sequence of words that exists in a language, O is an observed acoustic sequence and $\hat{W}$ is a hypothesized phonetic sequence.

According to Bayes theorem (Bayes, 1763), the ASR goal can be modified as follows:

$$\hat{W} = \arg\max_{W} \frac{P(O|W)P(W)}{P(O)} \tag{2}$$

We rewrite the formula in terms of $P(W|O)$ and $P(W)$ because we are able to estimate these statistical models using available data. $P(W)$ is the prior probability of the phoneme sequence, namely word, and is calculated based on the frequency of n-grams that contain word W in a language corpus. The probability $P(W)$ is also known as the probability of a language model in ASR. The probability $P(O|W)$ is the likelihood of the acoustic observation O given some sequence W. This probability is referred to as probability of the acoustic model in ASR. The probability $P(O)$ is the probability of an observation and it is always the same disregarding the instance of W. Thus, this probability term can be omitted resulting into the following formula for ASR:

$$\hat{W} = \arg\max_{W} P(O|W)P(W) \tag{3}$$

2.1.1 ASR system architecture

A typical speech recognition system includes an acoustic feature extractor, an acoustic model, a language model, a lexicon and a decoder and is presented in figure 1. At the feature extractor stage, the audio signal is converted into a feature vector containing important acoustic information for speech recognition. Using the extracted feature vectors for training data, the acoustic model estimates phone distribution. The language model models word distribution from a text corpus. Finally, the decoder searches through all the possible word sequences to find the most likely one to correspond to the given acoustic data using the provided language and acoustic probability distributions.

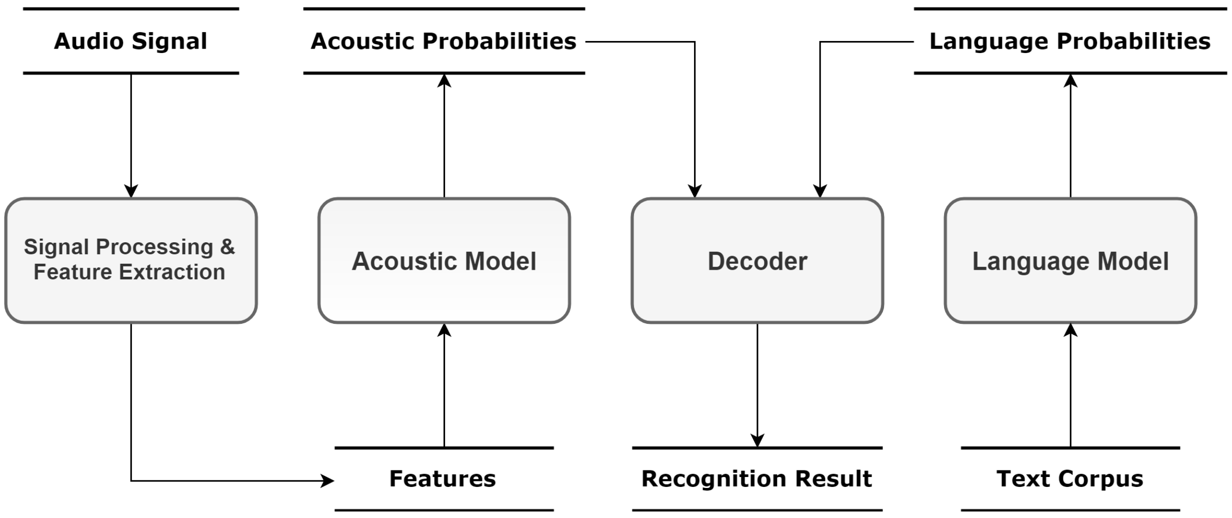

Figure 1: Pipeline of a typical automatic speech recognition system.

The feature extractor in ASR should provide features extracted from speech signal so that an ASR system can discriminate between similar speech sounds and these features can generalize well across different speakers and speaking conditions. There are many feature representations that are used in ASR but the most commonly used are mel-frequency cepstral coefficient (MFCC) features. ASR feature generation involves many steps of processing which we do not cite here (for more information see Dave (2013)).

The language model is a way to signal to the speech recognizer that some particular word exists in a language and that some sequence of words can occur in a language more probably than others. We can denote the language model as a way of providing some context to a speech recognition system. The language model is usually represented as an n-gram model estimated from a text corpus. The most common language models are bigram and trigram models where probabilities are calculated for sequences of two or three words.

The acoustic model is the core component of a speech recognition system. Acoustic modelling tries finding the statistical representation for the extracted feature vector of a speech fragment. After the modelling procedure, the acoustic model is defined as a set of statistical representations for each of the acoustic units that are derived for a language. These representations are derived based on a big amount of training speech data with the use of algorithms for statistical modelling. The most common way of statistical modelling for speech signals are the Hidden Markov Models (HMM).

The HMM helps to represent speech as a sequence of observations and usually models a basic unit of speech such as phone or word. HMMs are an extension of Markov Models with the difference that we are unaware in which state we are currently in. The hidden states in HMM are usually represented by phones and observations are usually acoustic features of the speech signal. In addition, the important components of HMM are the transition probabilities to move from one state to another and the observation likelihoods for a particular observation to be generated by a state. The scheme of an HMM is presented in figure 2. The phone **P** has three subphones for modelling the beginning ($\mathbf{P_b}$), middle ($\mathbf{P_m}$) and the end ($\mathbf{P_e}$) of a phone with transition probabilities $\mathbf{a_{ij}}$ and observation likelihoods $\mathbf{b_j}$. More extensive information on HMMs can be found in Poritz (1988).

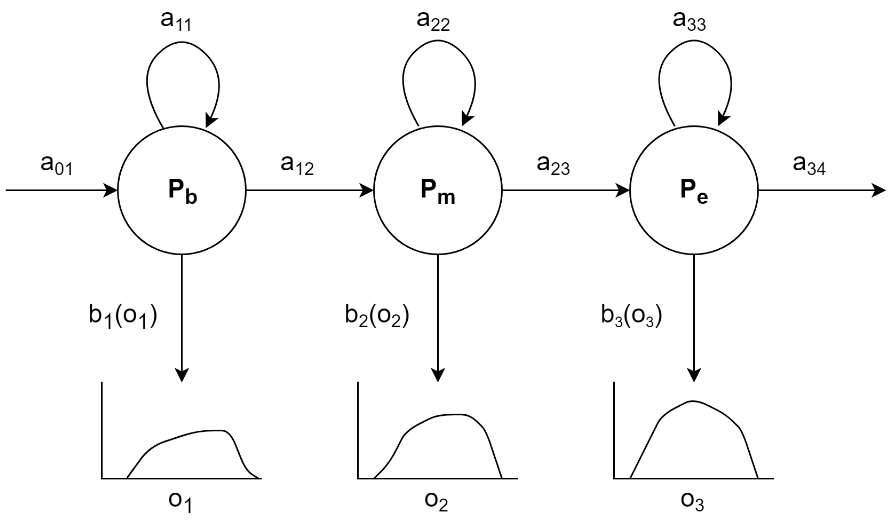

Figure 2: Illustration of an HMM for a phone.

In the decoding procedure, the decoder searches for the most likely word sequence given the speech input considering the different types of models available (language model, acoustic model, pronunciation model). The decoding task is usually performed by dynamic programming algorithms. A simple algorithm for the decoding problem is the Viterbi algorithm that finds the most likely sequence of hidden states in the context of hidden Markov models which could be generated by the observed sequence. Here we present the basic explanation of the Viterbi algorithm, for more information refer to Forney (1973).

For the illustration of the Viterbi algorithm, the notion of trellis diagram, a two-dimensional grid for a left-to-right HMM, is usually used where the horizontal axis represents the time frame and the vertical axis represents the HMM state (see figure 3).

For the functioning of the Viterbi decoding, two different types of information must be provided: HMM state transition probabilities and emission probability distribution conditioned on each state. The maximum joint likelihood $\delta_i(t)$ of the partial observation sequence o_1^t at the time t and the corresponding state sequence q_1 ending in the state i at time t is described by the following formula:

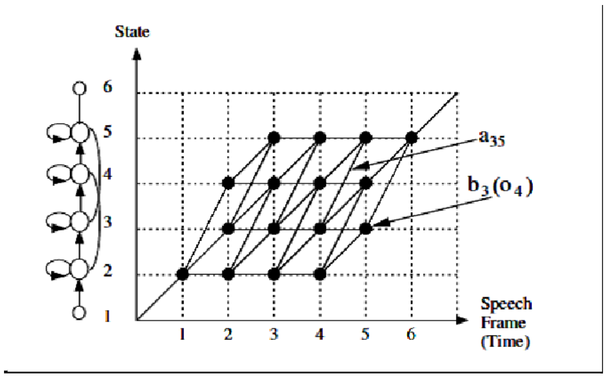

Figure 3: Trellis for Viterbi decoding. The picture is taken from Alhanjouri (2012).

$$\delta_i(t) = \max_{q_1, q_2, \ldots, q_{t-1}} P(o_1^t, q_1^{t-1}, q^t = i) \tag{4}$$

The optimal partial likelihood at the time step $t+1$ for each state j can be recursively computed as:

$$\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_j(o_{t+1}) \tag{5}$$

where a_{ij} is the state transitional probability from the state i to the state j and $b_j(o_{t+1})$ is the emission probability distribution for the state j given observation sequence o_{t+1}.

To find the best Viterbi path through states after the termination of the computation, simple backtracking can be used. For the faster computation of the Viterbi algorithm, the pruning procedure can be used which removes less likely Viterbi paths at each step so they are not considered in further steps.

2.1.2 ASR evaluation

The performance of an ASR system is usually defined in terms of its word error rate (WER), i.e. in how many words in the test set a speech recognizer makes mistakes. The WER score is defined in terms of the Levenshtein distance that measures the difference between two sequences of words and categorizes errors into three groups: insertions, deletions and substitutions. For ASR performance evaluation, a hypothesized sequence of words and a reference sequence of words from a transcription are considered. Insertion error means that a word was missed in the decoded sequence, deletion error signifies that a word appeared in the decoded sequence while not appearing in a reference sequence and substitution error means that a word is different in a decoded sequence in comparison to the reference sequence. WER score can be calculated by the following formula:

$$\text{WER} = \frac{I + D + S}{N} * 100\% \tag{6}$$

where I is the number of insertions, D is the number of deletions, S is the number of substitutions and N is the total number of evaluated words.

2.2 Neural Networks

Neural networks became very popular in recent years due to breakthrough results in many different fields including speech recognition. Neural network (NN) is a computing system inspired by the human brain structure that consists of interconnected neurons organized into layers which form the network. These layers usually perform various computations transferring information from one layer to another.

2.2.1 Neuron

The neuron is a basic computation unit of an NN which computes an output based on an incoming input. The scheme of a neuron is presented in figure 4. The neuron computation is a weighted sum of products coming from the previous neuron, or input, plus bias if introduced. Weights of neuron connections are parameters of the NN that can be modified and which the NN learns during training. Then, neurons of the NN apply the activation function that usually serves to introduce non-linearity for a final transformation of the output. Non-linear functions are important since we would like a neuron to learn a non-linear representation of the data because most data is not linear. There are many different activation functions applied for NN neurons such as sigmoid, relu, tanh, etc.

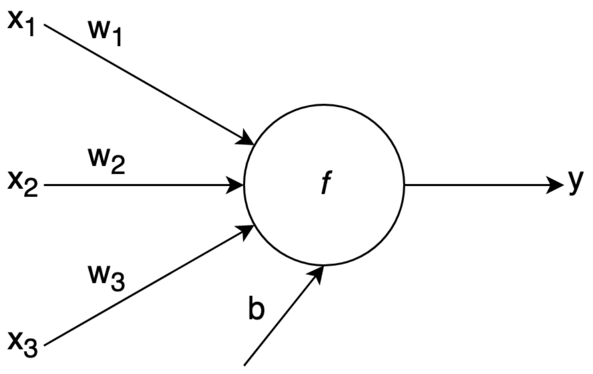

Figure 4: A basic scheme of an artificial neuron of NN.

2.2.2 Neural network architecture

The standard architecture of a feed-forward neural network is illustrated in figure 5. A neural network consists of layers arranged with multiple neurons. The basic architecture of a feed-forward NN is comprised of an input layer, an output layer and a hidden layer placed between them. The number of hidden layers can vary based on the desired complexity of NN. For the so called deep neural network (DNN), the number of hidden layers usually starts from two. The connection between layers is done through layer neurons that are connected to all neurons in a subsequent layer. The input layer is responsible for the processing of the input data that is fed into the network and transferring it to the following hidden layers. Each following layer of the NN is highly dependent on the information got from the previous layer. The output layer does the final transformation of the information obtained from hidden layers. Performing subsequent neuron computations at each layer and then calculating the loss of the final layer output compared to the ground truth define forward propagation step of NN.

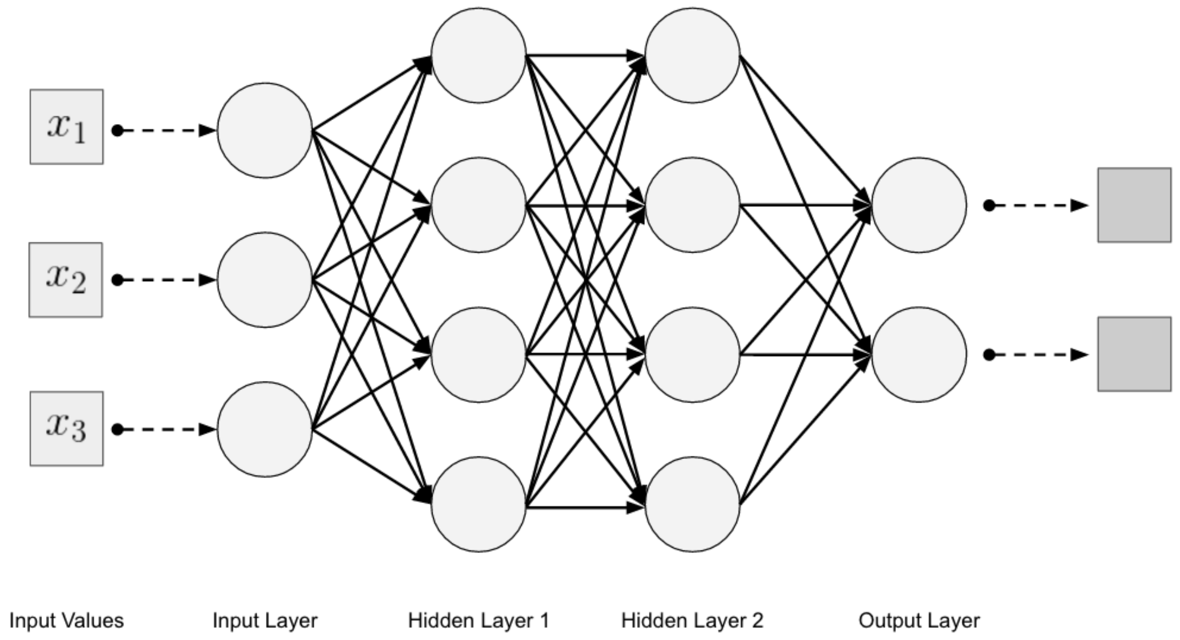

Figure 5: Architecture of a feed-forward NN. The picture is taken from Patterson and Gibson (2017)

2.2.3 Training

The most common way for a neural network to be trained (correct its weights and biases) is by means of the backpropagation algorithm. Backpropagation is a supervised training technique that relies on labelled data to backpropagate the error and assign the correct weights for neuron connections. By having properly defined weights, NN can determine the reasonable output for a given input.

NN's weights and biases are at first initialized at random. After forward propagation step, NN knows the obtained error having compared the output label with the ground truth label. Having this error, NN can backpropagate it by calculating the gradients which point out the direction of the error increase. The adjustment of NN weights is done by using Gradient Descent optimisation method which goal is to minimize the resulting error (for more information on Gradient Descent see Ruder (2016)).

2.2.4 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a type of neural networks that are good for modelling sequential data. For this reason, RNNs are very successful in using in natural language processing or speech recognition tasks. In comparison to feed-forward neural networks that do not have any notion of order and consider only a current input instance, RNNs can remember what has been seen previously in time. RNNs have two sources of input, present and recent past, which are combined in some extent in order to better describe data.

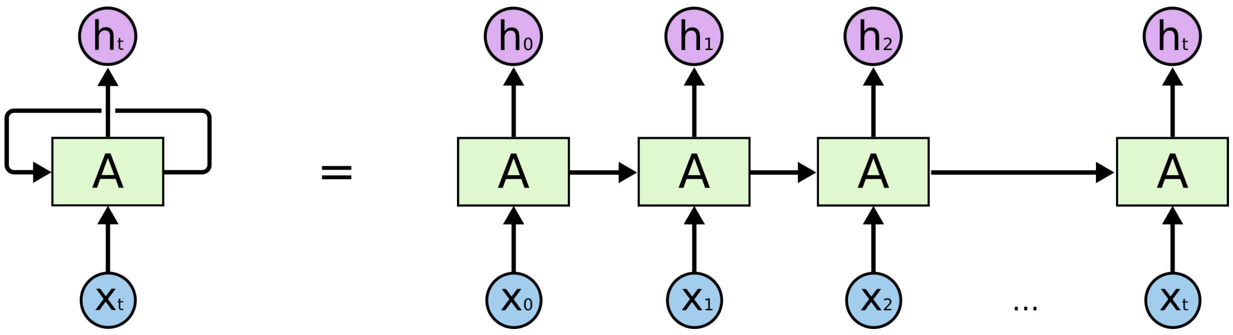

Figure 6: RNN architecture. The picture is taken from Christopher Olah blog.

The illustration of an RNN is presented in figure 6. As can be seen from the figure, RNN has a looping mechanism that allows the network to retain information across sequential input instances. There exist several ways for RNN to memorize information introduced previously. The most common ways are to use Long Short-Term Memory (LSTM) cells or Gated Recurrent Units (GRU). These both units have multiple gates that are used for memorizing the important parts of the input and forgetting unimportant past information. In figure 7, we present the LSTM cell architecture because it is the RNN cell that we use in our further experiments.

The LSTM cell can learn long-term dependencies of the data due to its architecture with three types of gates. Using input, forget and output gates, LSTM cell can remove or add information to the cell state. More on LSTM and fundamentals of RNNs can be found in Sherstinsky (2018).

2.3 ASR approaches

2.3.1 Generative approach

The most common approach for ASR is a generative learning approach that uses Gaussian Mixture Model (GMM) HMM for modelling a representation of the sequential structure of speech. As we already discussed, the acoustic unit representation is usually modelled by

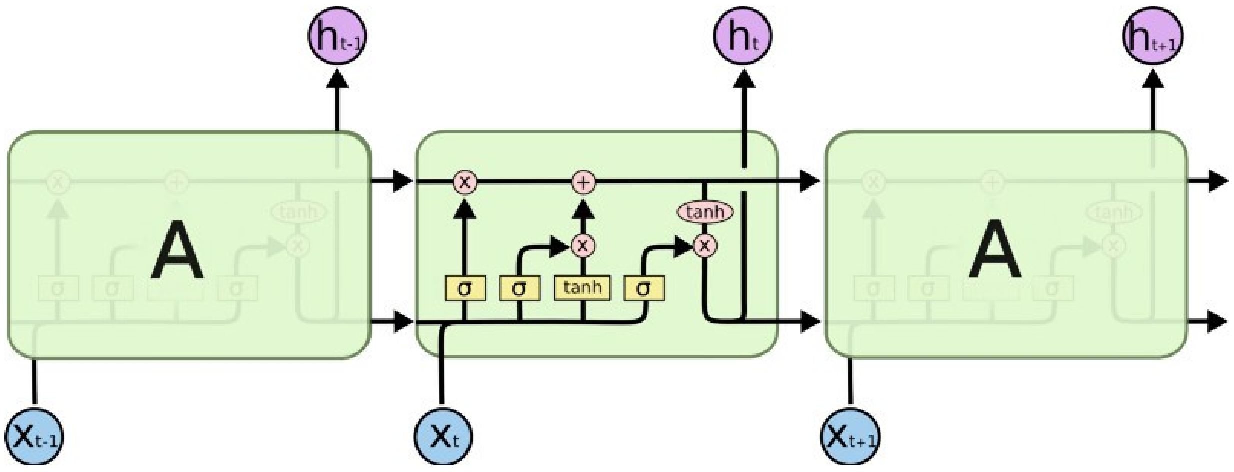

Figure 7: LSTM cell architecture. The picture is taken from Christopher Olah blog.

the HMM state. In the case of GMM-HMM, a mixture of Gaussians is used in the HMM state to model the acoustic unit representation (see figure 8).

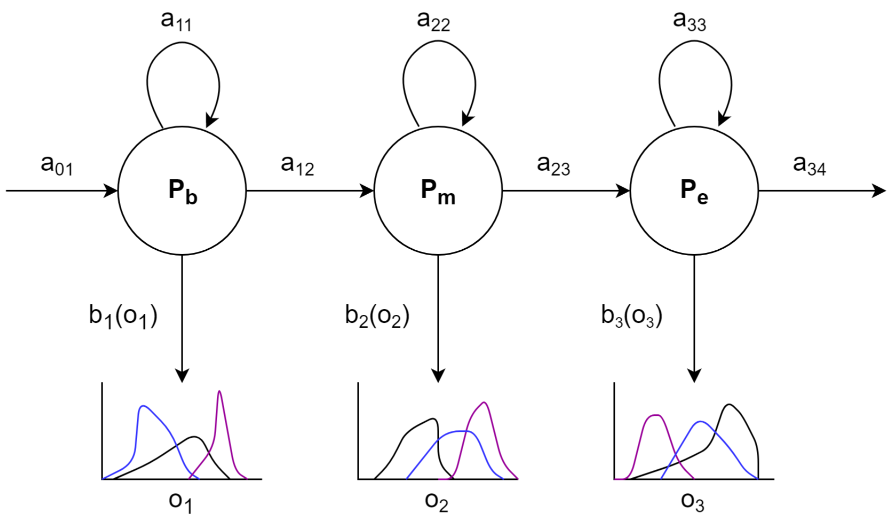

Figure 8: Illustration of GMM-HMM for a phone.

GMMs can easily model quite complex distributions and GMM classifiers are highly effective in speech-related tasks. The GMM-HMM systems are very popular in ASR domain because they can handle variable-length data which is the characteristic feature of speech. These systems can achieve good results for speech recognition and became the default option in speech recognition tasks (Su et al., 2010; Yan et al., 2013).

2.3.2 Hybrid approach

One of the approaches that is proved to work well in the ASR domain is the combination of DNNs with HMMs which is called hybrid DNN-HMM. In this system, the variable-length speech signal is modelled with HMM and emission probabilities of phonemes are modelled by DNN (see figure 10). For this task, the DNN is trained to estimate the posterior probability of an HMM state given an observation.

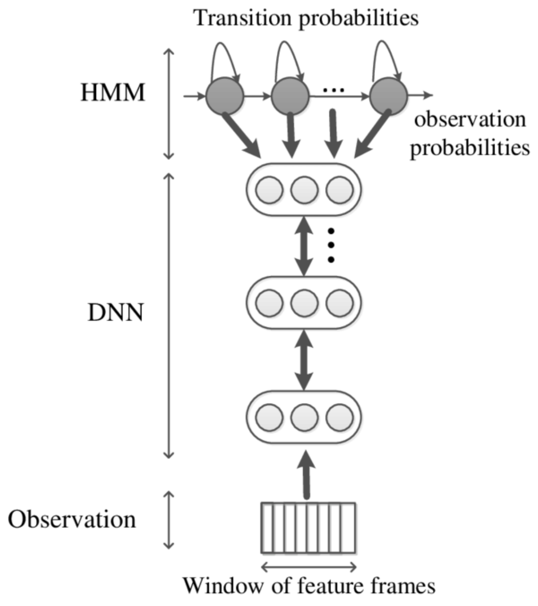

Figure 9: Illustration of DNN-HMM for a phone. The picture is taken from Najafian (2016).

This framework is widely used and considered to be successful due to HMM's ability to model the sequential property of a speech signal and the strong learning power of DNNs (Swietojanski et al., 2013; Li et al., 2013; Xue et al., 2014).

2.4 Lexicons in ASR

The lexicon for an ASR system is usually represented as a list of words where for each entry the orthographic transcription of a word is given with its corresponding transcribed pronunciation. The words of a lexicon are usually taken from the vocabulary of a text corpus. Pronunciations are commonly composed of elementary units derived for a particular language.

The lexicon, or pronunciation dictionary, plays an important role in the ASR system being the link between the acoustic representation for a word and the word output by the ASR decoder (Adda-Decker and Lamel, 2000). The first task of the lexicon is to specify the list of words that will be known by the ASR system and used to model the representations. Having lexical knowledge helps the ASR system to produce meaningful word sequences. The second task is to provide some information about word acoustics. This information serves for building the acoustic model by the ASR system.

The lexicon can be considered as another model involved in the ASR process. This model is called pronunciation model and defines the probability $P(Q|W)$ of an acoustic unit sequence Q given a word W. The ASR formula can be rewritten as follows to consider possible sequences of acoustic units (pronunciations) Q for a word W given observation O:

$$\hat{W} = \arg\max_{W} \sum_{Q} P(O|Q)P(Q|W)P(W) \tag{7}$$

Words in a lexicon may have multiple pronunciations due to various accents, speech impediments, phonological phenomena or simply a part of speech. This might imply many pronunciations for words in a lexicon in order to model context-dependent pronunciation variation or general speaker characteristics. This can increase the modelling power of a pronunciation model. However, this must be taken with caution since it can increase ambiguity for observed sequences. Pronunciations in a lexicon should be as consistent as possible so that to increase the descriptive power of an acoustic model.

2.4.1 Lexicon generation

Lexicon generation is a separate task in speech processing. The process of lexicon generation involves vocabulary selection, basic units modelling and pronunciation representation for each lexicon entry.

The vocabulary selection tasks consider which words existing in a language and what number of them will be included in the dictionary. The main goal of ASR is to obtain the maximum possible coverage of the language vocabulary in the developed lexicon. The lexicon word entries can be extracted from the vocabulary of various already existent text corpora for a language.

The modelling of the representations suggests choosing elementary acoustic units which in case of ASR system can be phonemes or sub-word (phone-like) units. The choice of elementary units is usually motivated by the particular application or language of ASR.

The task of pronunciation creation poses the most problems for lexicon generation because it requires linguistic knowledge to model such word pronunciations. There exist several ways of pronunciation modelling. Already existing pronunciations dictionaries for a language can be a source of these pronunciations but they are quite rare, can be unavailable for a particular language or built with inconvenient basic acoustic units for ASR. The pronunciations for words in a lexicon can be created manually exploiting linguistic knowledge. This procedure of lexicon creation is quite time-consuming and prone to have errors and inconsistencies since the human factor is involved. However, hand-crafted pronunciations are highly linguistically justified. Alternatively, word pronunciations can be generated automatically by a pronunciation generation system. The most common way of automatic pronunciation generation for a lexicon is grapheme-to-phoneme conversion (G2P) (Bisani and Ney, 2008; Taylor, 2005; Chen, 2003; Toshniwal and Livescu, 2016). We discuss this way of lexicon creation in chapter 3 where we provide the related work. Finally, the two approaches for pronunciation generation can be combined. The pronunciations in a lexicon can be generated by the automatic pronunciation generation system and then they can be edited by linguists.

3 Related Work

The goal of automatic pronunciation generation is not new in the ASR field. Several methods to get lexicons were proposed for ASR (Svendsen, 2004). There are two main approaches to automatically obtain pronunciation dictionaries. The first one involves grapheme-to-phoneme (G2P) conversion and it can usually be considered as a semi-supervised approach since it relies on some small amount of the initial lexicon provided to G2P model in order to catch the correct conversions. Another approach is fully unsupervised and aims to generate a lexicon without any prior information available. Usually, such approaches focus on the derivation of basic acoustic units and creating pronunciation sequences from them for words in the vocabulary.

In sections 3.1 and 3.2, we report the most recent and relevant research that was done in the area of pronunciation generation with supervised and unsupervised methods.

3.1 Semi-supervised lexicon generation

Semi-supervised lexicon generation mostly suggests using G2P models with a small initial lexicon available so that to generate the full lexicon from it. There exist many approaches for G2P, however, in this section, we provide the recent research on G2P lexicon generation that aims to fully generate a lexicon and not only to handle out-of-vocabulary (OOV) words which is considered as a primary task for G2P systems.

In Goel et al. (2010), the authors exploit the G2P system and a small initial lexicon in order to derive the full lexicon for ASR training. They use the initial lexicon as a bootstrap for training a G2P model that uses joint-multigram approach to learn pronunciation rules. Since no acoustic model is available at this step, they train G2P models to generate the pronunciations for all remaining words and train the acoustic model on them. The acoustic model, in turn, gives the set of pronunciations for creating a new dictionary which is used for further retraining of G2P. The process of training both acoustic and G2P model continues iteratively until the best performing acoustic model is obtained.

In Kantor and Hasegawa-Johnson (2011), the authors model each context-dependent phoneme as an HMM and treat letters as observations generated by the HMM. Then, the model is iteratively trained with the EM algorithm. The decoding is done by the Viterbi algorithm. The advantage of the proposed method is that no mapping between letters and phonemes is needed which means no need in linguistic knowledge. This work shows that there still remains a lot of lexical ambiguity with lexicons generated with this approach for conversational speech.

In Razavi and Doss (2015), the authors propose the novel approach of pronunciation generation based on the automatic derivation of subword units. The subword units are derived from the clustered-context dependent units in a grapheme-based system using a maximum-likelihood criterion. Then, Kullback-Leibler divergence HMM learns the pronunciations. This approach shows a significant reduction in word error rate compared to other G2P systems and can help to reduce the need in expert knowledge for lexicon creation.

The main problem of methods that use G2P for discovering correct pronunciations of words is that they seem to work only for languages with a strong correlation between pronunciations and graphic writing of words such as German or Czech. For languages where graphemes do not necessarily correspond to some particular pronunciations and there exist a lot of deviations in pronunciations in general such as English or Arabic, it leads to a problem for G2P to get correct relations between graphemes and phonemes which results in poor lexicon generation systems. Thus, methods using G2P proves to be very efficient for languages with strong grapheme-phoneme relation but other languages will be provided with an unreliable lexicon in this case. Based on these considerations, we do not want to explore G2P-related approaches for data-driven pronunciation generation since we would like to be as less language dependent as possible in the task of pronunciation dictionary creation.

3.2 Unsupervised lexicon generation

Another approach of automatic pronunciation generation is unsupervised and fully relies on provided audio data. It can exploit different strategies for lexicon generation but the main idea is to use audio data and transcriptions as the only source for pronunciation dictionary generation.

There are not so many unsupervised methods in the literature. One of them is still about G2P conversions (Hartmann et al., 2013). The authors in their work used grapheme-based recognition system in order to determine the minimal acoustic units and to generate the pronunciation dictionary. Via spectral clustering approach, the context-dependent graphemes are clustered into the acoustic units. The first draft of the lexicon is derived from this clustering. Based on this lexicon, a set of context-independent models (one GMM per grapheme) are trained. The output of grapheme recognition is considered as pronunciation hypothesis and becomes a training example for statistical machine translation grapheme-to-grapheme system (lexicon pronunciation is used as a source language and a hypothesis is used as a target language). As a scoring method they proposed to generate a new dictionary using the grapheme-to-grapheme model, do forced alignment of the training data using the context-independent models and measure the average effect on the likelihood of each sentence. As results, they report the relative reduction in WER in comparison to baseline grapheme-based systems.

Another method completely relies on audio data and not on word transcriptions, namely graphemes (Takahashi et al., 2016). The approach the authors invented is based on detecting subword units, or automatic acoustic elements (AAEs) that can further comprise the pronunciations for words. The initial step is to cluster the acoustic space in order to determine the initial AAEs. Then they perform iterative training of GMM that jointly learns to model the dictionary and AAEs. Given AAEs, the model tries to update the dictionary and on the next step, given a new dictionary, it tries to update AAEs. After some number of iterations, the authors switch from GMM to DNN training with the same iterative procedure. At each stage, the pronunciations are got with the help of k-dimensional Viterbi approximation (Naghibi et al., 2013) which is based on the k-dimensional Viterbi algorithm

(Gerber et al., 2011). This approach shows to outperform known phoneme-based systems with handcrafted dictionaries.

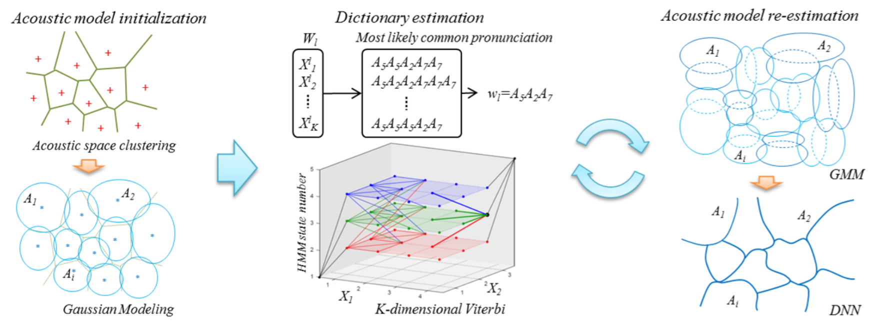
Acoustic model initialization
Acoustic space clustering
Gaussian Modeling
Dictionary estimation
Most likely common pronunciation
$w_l=A_5A_2A_7$
HMM state number
K-dimensional Viterbi
Acoustic model re-estimation
GMM
DNN

Figure 10: Illustration of the framework proposed by Takahashi et al. (2016). The picture is taken from Takahashi et al. (2016).

These approaches for unsupervised lexicon generation aimed to discover minimal acoustic units and generate lexicon entries utilizing determined units. The presented methods can generate a lexicon completely from scratch from provided audio data and transcriptions. For this reason, these works need some starting point that does initial modelling of the data which, in fact, very weak model since it does not have any supervision.

In this research, we would like to explore if data-driven pronunciations can improve the quality of ASR in comparison to linguistically motivated pronunciations. To explore this, the completely new data-driven approach for pronunciation generation is proposed that aims to improve the existent lexicon and, to our knowledge, has not been explored in the literature so far.

The similarities of our new approach for pronunciation generation in comparison to introduced unsupervised research are that our new approach:

- is also data-driven because it relies on acoustic information of the data for pronunciation generation;
- uses no initial lexicon for bootstrapping;
- aims to generate a full lexicon and not just a part of it for ASR purposes.

In contrast to related work on unsupervised pronunciation generation, our new approach:

- has an objective of regeneration of an existent ASR dictionary aiming at improving the ASR performance by introducing a better lexicon;
- does not discover a new phoneme set and works with the existing one;
- considers more than one pronunciation to be generated for ASR lexicon while all related works tried to find one best pronunciation for a word in a lexicon.

4 Research Outline

The goal of this research is to explore the area of data-driven pronunciation generation. We aim to introduce an experimental procedure to discover the method to determine which pronunciations and how many of them should be written into the pronunciation dictionary for a word.

In general, our approach is based on pronunciation clustering as a source of pronunciation variants which can serve as good lexicon entries for a word. For lexicon generation, we propose to completely rely on data and to use a clustering algorithm to decide whether a word has several pronunciation variants, which they are and how many of them appear in the data.

The motivation for this approach is that the differences in pronunciations for a word should be reflected in the features extracted from the audio segment for this word. The more similar features of the same word are to each other, the more similar pronunciations of these word utterances are. If we have several utterances for the same word, we can use acoustic features for the word speech segments to cluster them into different pronunciations.

In this chapter, we discuss the whole pipeline of the thesis research. In section 4.1, we discuss the proposed approach of data-driven pronunciation generation. Section 4.2 provides information on experiment design considering the systems for comparison and evaluation criteria.

4.1 Proposed approach

The pipeline of the proposed approach is illustrated in figure 11. The main idea behind the approach that we would like to explore is to use a clustering procedure for the pronunciation variant identification directly from audio features. After the variants are clustered, the simple Viterbi decoding is used in order to obtain phoneme sequences that denote different pronunciation variants. Finally, decoded pronunciations can be written into an ASR lexicon. Further in this section, we discuss each component of the proposed system in more details.

4.1.1 Pronunciation clustering

Clustering, or cluster analysis, is a task of grouping data points in groups, or clusters, based on their feature similarity. The clustering is done in a way that data points that are more similar to each other in some sense are grouped together. Clustering is a technique of statistical data analysis and belongs to the area of unsupervised machine learning where no labelled data is introduced to an algorithm.

In our task, we would like to cluster data points corresponding to features of different utterances of a particular word. As a result, we would like to see clusters of different pronunciation variants for a word. The main goal at this stage is to derive the number of pronunciation clusters automatically for each word without having it predefined. We would like to detect the number of clusters per word based on statistics and not to use

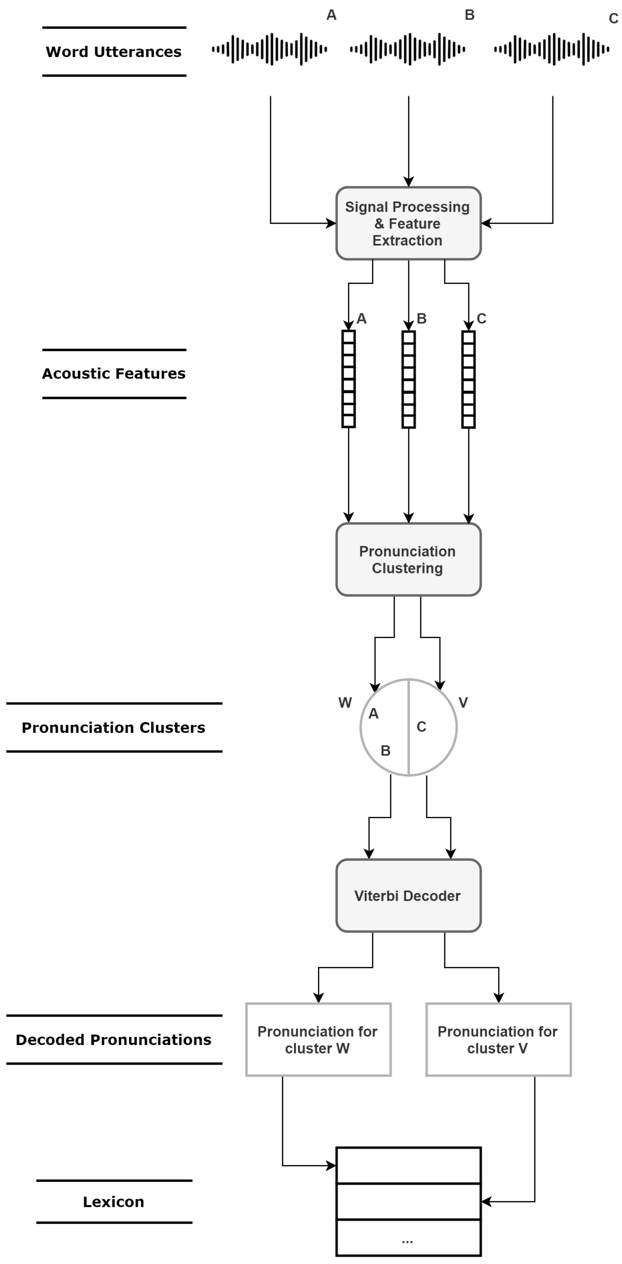

Figure 11: The pipeline for the proposed approach that consists of feature extraction, clustering procedure and decoding.

some constant number of clusters since the number of pronunciations can differ from word to word.

4.1.2 Feature extraction

For the pronunciation clustering with a machine learning algorithm, we need to have features of some particular dimensionality for all word utterances in the corpus. In ASR, there are several ways of feature extraction. The most frequently used are MFCC, PLP and LPC. MFCC is a prevalent way of feature extraction for speech recognition purposes. MFCC is a representation of the real cepstrum of the Fast Fourier Transform of a windowed short-time signal. The main problem of MFCC features for a clustering algorithm is that an audio signal gets windowed by a fixed length (see figure 12). Taking into account the length variability in speech (even the same word can be pronounced by the same speaker in a different time), it results in a variable-length feature vector after applying all transforms.

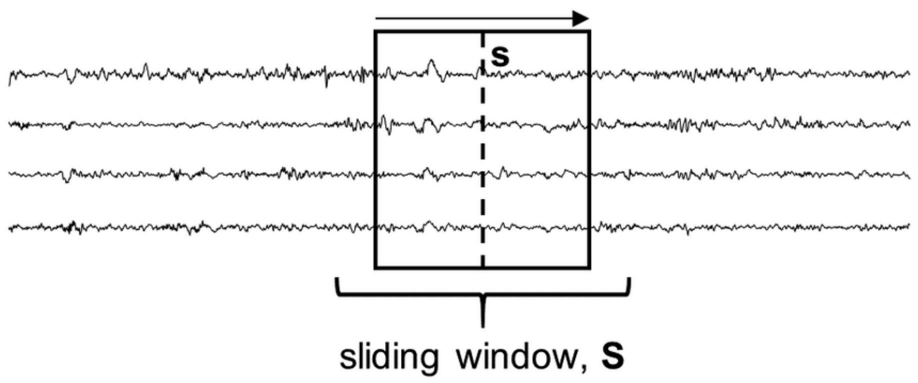

Figure 12: Illustration of sliding window that is used for feature extraction in ASR. The picture is taken from (Dereymaeker et al., 2017).

Thus, we propose a trick for feature extraction step so that feature vectors could be used for clustering. The main idea that we would like to preserve as much of acoustic information as possible but at the same time to get feature vectors of the same dimensionality. One of the existing ways of bringing variable-length feature vectors to fixed-dimensional ones in the speech domain is to exploit acoustic word embeddings (AWE).

AWE is a fixed-dimensional vector representation of a spoken word segment. Mainly, AWEs are generated based on acoustic information of a word with some additional information involved. There are many approaches for AWEs training but the most common is the one that uses MFCC features as the source of acoustic information about a speech word segment and tries to learn the place of embeddings in the AWE space by discriminating different word types[1]. Thus, the task of AWE training is to place embeddings for words of the same word type closer to each other while embeddings for words of different word types are further from each other in the AWE space.

Hence, we can extract AWEs for all words in a corpus and use them as features for a clustering algorithm since they are no longer of variable-length. Then, for each word in a corpus we can perform pronunciation clustering using an unsupervised machine learning algorithm.

[1]By word types, we mean different words (*cat* vs. *dog*).

4.1.3 Decoding

After the clustering is performed, we get the clusters comprised of acoustically similar word utterances, namely pronunciation variants, for each word in a corpus. Since our main goal is to generate the ASR lexicon, at this point, we would like to descend from a vague category of clustered pronunciation variant to a phonetic sequence that corresponds to this detected pronunciation variant. For converting a speech segment into a pronunciation sequence, the decoding procedure of ASR can be used.

Decoding in ASR is the process of finding the most likely sequence of phonemes for a particular speech fragment. The most known decoding algorithm is the Viterbi algorithm which is a dynamic programming algorithm that finds the most likely sequence of hidden states for an observed sequence in the context of HMM. The Viterbi algorithm requires several types of probabilities such as the initial probability for a phoneme to be in word-initial position, the transition probability for a phoneme to transit to another phoneme and the emission probability for a phoneme to occur given this audio fragment (see section 2.1.1 for more information on Viterbi algorithm).

Applying Viterbi decoding for all words in a corpus gives us decoded sequences with which we can operate at post-clustering step. Having a pronunciation sequence for each word utterance in a cluster provides us with the information we can put into a potential ASR lexicon. Since we believe that we can successfully cluster word pronunciations, the decoded sequences for words in a cluster should be mostly similar. Thus, we can take the most frequent decoded sequence per cluster in order to decide which pronunciation corresponds to each cluster. After all these steps, we have some number of decoded sequences corresponding to detected clusters, or different pronunciation variants, that we would like to write into a pronunciation dictionary.

This approach generates a fully data-driven lexicon because at each step it considers information derived from data in contrast to the widely used hand-crafted pronunciation dictionaries that are created by linguists.

4.2 Experiment design

The experiment design for this research needs to consider small experiments for each component of the proposed approach. The objectives of these experiments are:

- to compare the quality of the generated data-driven lexicon to other lexicons of interest;

- to compare different clustering algorithms for the task of pronunciation clustering;

- to find the most appropriate way to obtain acoustic features for clustering.

In order to evaluate the quality of the obtained data-driven lexicon, we propose to train and compare the ASR performance for the data-driven lexicon and the hand-crafted one in terms of their WER. As an ultimate goal, we see outperforming the hand-crafted dictionary.

However, we understand that reaching the performance of the hand-crafted dictionary is a very difficult task. Nevertheless, we are interested in seeing if we can at least approach the performance of hand-crafted dictionary with the lexicon generated completely from data.

In this regard, we define the hand-crafted dictionary as a benchmark lexicon. For estimating if we can perform better than some simple approaches we also need to define some baseline lexicons. As baseline models, we choose ones that exploit sequences decoded by the Viterbi algorithm. For each word in a corpus, we have several pronunciation sequences corresponding to decoded word utterances. We consider the lexicon in which for each word we write the three most frequent decoded sequences to be a quite strong baseline for our experiment.

In view of some additional experiment, we would like to introduce some other baseline lexicons. They are comparable to the already introduced one but include either only one the most frequent pronunciation or all pronunciations decoded by the Viterbi algorithm. These baselines are interesting to consider because it is unclear what number of pronunciations is the most beneficial to include into the ASR pronunciation dictionary. This kind of experiment can reveal if writing just one pronunciation per word is enough or the more pronunciations we include in the lexicon, the better the ASR performance is. At this time we lack evidence that having multiple pronunciations for a word in a dictionary benefits ASR.

In order to obtain the best possible clustering of different pronunciations, the examination of different clustering algorithms can be done. Moreover, we would like to investigate different strategies of feature extraction for a clustering algorithm and see if we can improve a simple approach of AWEs extraction for pronunciation differentiation task since the main approach for AWE modelling works with word differentiation. The main idea is that the better AWEs represent acoustics of a word, the better clustering will perform. The goal of these experiments is to get the best possible result of these two steps of data-driven pronunciation generation.

4.2.1 Experiment pipeline

Thus, the whole pipeline of the research can be described as follows:

1. We experiment with fixed-dimensional feature extraction for clustering algorithm trying different approaches for improving the discriminative power for pronunciations of AWEs.

2. We cluster obtained AWEs for each word in a corpus with different unsupervised clustering algorithms in order to discover the most suitable one for the task of pronunciation clustering.

3. We decode the whole corpus with the Viterbi algorithm so to get a pronunciation sequence for each word. These decodings are used in the final lexicon generation step.

4. We generate several lexicons (baseline lexicons and data-driven lexicons) and train the ASR system using these lexicons in order to compare their WER scores between each other and with the hand-crafted lexicon performance.

In order to compare different lexicon generation models, we train the ASR model on the corpus of multi accented speech. We choose this type of corpus because we would like to test pronunciation generation on some data that has many pronunciations of a particular word due to some variation in the pronunciation of different speakers.

The training of ASR models is done using Kaldi (Povey et al., 2011), widely known speech recognition toolkit. In our experiment, we train the most recently introduced neural network model in this toolkit that uses TDNN-f layers (Povey et al., 2018) in order to have up-to-date and reliable results for our experiments.

The whole experiment helps to understand if we can improve over hand-crafted dictionary or the linguistic knowledge still outperforms the statistical approaches. Moreover, based on experiment results, we can conclude whether the number of pronunciations that are written into a dictionary should be statistically modelled.

5 Data

For our experiment on data-driven pronunciation generation for improving ASR performance, we have to choose the appropriate corpus so that we could witness the improvement easily. Since the most benefited from data-driven pronunciation generation will be multi pronouncing words, we would like to choose the corpus with as many pronunciations per word in the data as possible. That is why, as a good choice for our pronunciation generation task, we consider a corpus with a lot of accented speech and which includes many dialects. As a language for the whole experiment, we propose English.

There exist several corpora for English that include many different dialects. First of them is CSTR VCTK Corpus[2], English Multi-speaker Corpus for CSTR Voice Cloning Toolkit, that includes speech data uttered by English native speakers with various accents (Veaux et al., 2017). Each of 109 speakers reads about 400 sentences selected from newspapers. We consider the size of the corpus to be insufficient for the task of pronunciation generation.

Second possible corpus is Common Voice[3] by Mozilla. It is an open-source, multi-language dataset of recorded speech. The recording can be done by anyone following the instructions on the website. Common Voice developed quite quickly and already reached 1 000 hours of English speech having almost 40 000 different speakers. However, it is a new corpus for which no baselines for the comparison exist, especially in Kaldi toolkit[4].

The last corpus with accented speech is VoxForge[5]. This corpus has a sufficient amount of data, namely ∼110 hours of speech, available open-source and Kaldi has a recipe for this corpus providing a baseline for the comparison.

This chapter is devoted to the data overview. In section 5.1, we discuss the VoxForge data and provide some statistics on it. Section 5.2 provides information on the data processing that we used for the experiments.

5.1 Data overview

For all our experiments we use VoxForge corpus of audio recordings. VoxForge is an open-source project that collects the transcribed speech data that later can be used for free and open-source ASR toolkits such as HTK[6], Kaldi, CMU Sphinx[7], etc. The VoxForge data is comprised of audios that were recorded by volunteers in many different languages. Volunteers record these audios directly on their computers following instructions that can be found on the main project web site.

The motivation behind choosing the VoxForge corpus is that the improvement of a pronunciation generation system will be more noticeable for the corpus where we definitely

[2]`https://datashare.is.ed.ac.uk/handle/10283/2651`
[3]`https://voice.mozilla.org/en`
[4]`https://kaldi-asr.org/`
[5]`http://www.voxforge.org`
[6]`http://htk.eng.cam.ac.uk/`
[7]`https://cmusphinx.github.io/`

can find multiple pronunciations for a big number of words in the corpus. Multiple pronunciations are not frequent in ASR lexicons, only few words can be found in the lexicon that have multiple pronunciations and these lexicons are usually hand-crafted with only linguistically motivated pronunciations included. The corpus that includes a lot of accented speech is the best choice for testing if we can actually improve the ASR system by introducing acoustically motivated pronunciations in the lexicon entries. In our research, we use English VoxForge data which includes audio recordings of people of different English dialects.

In table 1, we cite the main statistics of the VoxForge corpus. The English data contains 106.7 hours of speech recordings in total. For the ASR experiments, we split data into training and testing sets following Kaldi recipe for VoxForge, having 106.1 hours in the train set and 0.6 hours in the test set. We use the training set for all of our following experiments, even unrelated to ASR training, and for ASR training itself. The testing data set was used only for the evaluation of the ASR system and is never used in any other experiment that we conducted. After the data processing of the training set that is described in section 5.2, we estimated the size of the corpus in words to be equal to 538 622, having 6 725 unique words.

	Total	Train	Test
# hours	106.7	106.1	0.6
# utterances	76 551	76 179	372
# speakers	2 890	2 870	20
# words	541 878	538 622	3 526
# unique words	6 758	6 725	1 232

Table 1: The number of hours, utterances, speakers, words and unique words for total VoxForge corpus and its train and test sets.

In figure 13, we showed statistics across the speakers who did recordings for the VoxForge corpus. From the figure, we can see that the VoxForge corpus is biased towards adult male speech and it lacks recordings by other age ranges and female speech. In the VoxForge corpus, approximately ten different accents are represented with prevailing dialect being American English. We consider the problem of under-representation of different ages, genders and dialects to be acceptable since there are not many other corpora that have equal representation of different groups and the amount of other dialects and ages in the VoxForge corpus is quite substantial.

5.2 Data processing

The ASR lexicon is represented by a list of words with their pronunciation equivalents. The task of pronunciation generation implies pronunciation derivation for each word of the vocabulary. As was discussed in section 4.1, for our approach we would like to cluster words based on their acoustics in order to find clusters of pronunciations and, then, to

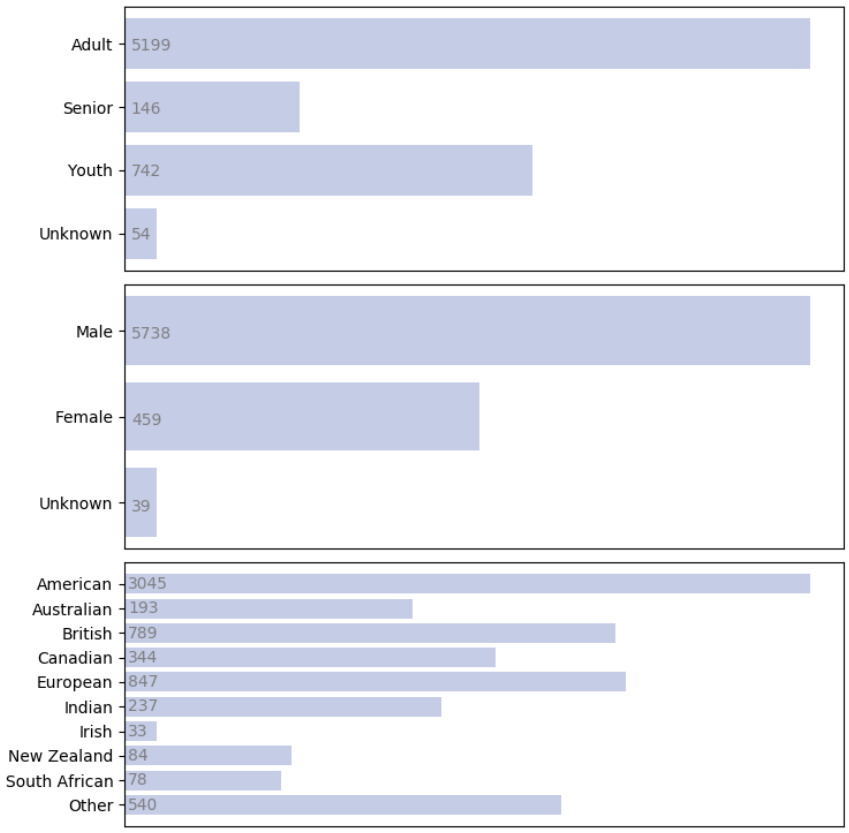

Figure 13: The statistics for the VoxForge corpus in terms of age, gender and dialect of speakers. X axis is log scaled.

detect the most appropriate phonetic sequence for each pronunciation cluster with the Viterbi algorithm. To perform such clustering, we have to descend to the word level and operate with words instead of utterances.

There exist several spoken isolated word corpora such as TIDIGITS[8], TI 46-Word[9], The Nationwide speech Project[10], etc. However, none of these corpora serves our purpose of having various accents and dialects in its data and they are mostly too small or have limited vocabulary which is not suitable for our goals.

Thus, sticking to VoxForge, which is an utterance-based corpus, requires to find the way of extracting word-level features from utterance-level ones. This can be achieved with the forced alignment of utterances and, then, cutting utterance-based features into words following the obtained alignments.

[8]https://catalog.ldc.upenn.edu/LDC93S10
[9]https://catalog.ldc.upenn.edu/LDC93S9
[10]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3060775

5.2.1 Phone alignment

Forced alignment refers to the process where audio transcriptions are aligned to audio recordings in order to get phone level segmentation, namely at which particular time some phone occurs in the audio segment. This process is different from ASR since ASR derives the particular phone for some speech segment whereas forced alignment already possesses the particular phone sequence and needs only to match its phones with the proper speech segments.

In order to align the data, we need to obtain a reliable, in terms of phone alignments, model. For this purpose, we train the triphone ASR model using Kaldi toolkit. For more robust results, we train it in a consecutive manner starting with the monophone GMM model[11] on a subset of 1 000 utterances of the data, proceeding with the triphone GMM model on delta + delta-delta features[12] followed by the triphone GMM model with LDA[13] + MLLT[14] feature transforms[15] and finishing with the triphone GMM model on LDA + MLLT features with applying SAT[16] [17]. For the training, the CMU pronunciation dictionary and phoneme set were used.

This final model we use to align VoxForge utterances to phones. The quality of triphone GMM model is considered to be reasonable for obtaining word alignments and we do not need to train a more advanced ASR model. The phone sequence for an utterance transcription is also obtained by this final ASR model. The quality of phone recognition does not matter for us at this stage since we are interested more in detecting word boundaries.

For obtaining phone alignments from the GMM model, we use the internal Kaldi `ali-to-phone` script[18] that takes the trained model and archives of aligned data that the model produced at the alignment step and writes down the phone alignments in human-readable format. The result is the audio file name with the corresponding phones and their starting and ending times.

5.2.2 Word alignment

Our goal is to align utterances to word transcripts and not to phones. Thus, following the format of the obtained alignments, we have to get words that are present in each audio file. These words can be extracted from transcriptions files provided in the corpus with the simple custom Python script.

The language preparation scripts in Kaldi convert each phone from a phoneme set to the position-dependent version of this phone by attaching one of _B, _I, _E and _S suffixes. These suffixes mark the position of the phone in a word. For the phone with the suffix_B,

[11]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_mono.sh`
[12]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_deltas.sh`
[13]Linear Discriminative Analysis
[14]Maximum Likelihood Linear Transform
[15]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_lda_mllt.sh`
[16]Speaker Adaptive Training
[17]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_sat.sh`
[18]`https://github.com/kaldi-asr/kaldi/blob/master/src/bin/ali-to-phones.cc`

the phone is in word-initial position. Suffix _E means that the phone is in word-final position. If the phone is in word-internal position, it is marked with suffix _I. Suffix _S denotes singleton words. This is the Kaldi language processing that happens before any model training and, then, models operate with this extended position-dependent phoneme set. Thus, all phone alignments that we got at the previous step are done with these position-dependent phones.

These position-dependent phones allow us to easily detect word boundaries. By a simple custom Python script, we detect _E and _S phones that denote the end of a word and extract word-level alignments based on these word boundaries. Thus, we obtain time intervals that correspond to a separate word and, with the help of files with words per each audio file, we can get actual word alignments. The final format of these word-level alignments is as follows: for each audio file, we have corresponding time interval for each word in the transcription of this file.

5.2.3 Word-based features

The goal of our data-processing is to extract word-level features, namely MFCC features. As far as we have time intervals for words, we can cut utterance-based MFCC features into words based on these time intervals. Thus, we need to obtain utterance-based MFCC features which can be done with internal Kaldi script[19]. After extraction, all features are stored in the archive, that is why we need to use the Kaldi `subset-feats` script[20] to convert them in the human-readable format.

At last, we can easily cut these utterance-based features by obtained word beginning and end times in order to get word-based MFCC features. This is also done with a simple custom script.

Word-level MFCC feature extraction is the only data processing we did for our experiments. These word-level features are used for the pronunciation clustering experiment which will be discussed in chapter 7 and for the decoding part of lexicon generation which we will consider in chapter 8. For the final ASR performance comparison discussed in chapter 9, we use original utterance-based MFCC features without any additional processing.

[19]https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/make_mfcc.sh
[20]https://github.com/kaldi-asr/kaldi/blob/master/src/featbin/subset-feats.cc

6 Feature Extraction

The data-driven pronunciation generation task involves the step of pronunciation detection in data and the step of writing detected pronunciations into a dictionary. For pronunciation derivation from data, we proposed to exploit the clustering analysis in order to detect different word pronunciations. The task for pronunciation clustering is to group word utterances into clusters corresponding to different pronunciations of a word. For clustering different word pronunciations, we explore different unsupervised machine learning algorithms that can identify internal data structure using features extracted from data.

In this chapter, we discuss the feature extraction step for pronunciation clustering and the following chapter is devoted to clustering of pronunciation variants. In section 6.1 of this chapter, we discuss the problem of MFCC features for machine learning algorithms. In section 6.2, we review related work for feature extraction that can be applied for our purpose of pronunciation clustering, compare them with each other and define the approach that we chose for our experiments. Section 6.3 describes in detail the feature extraction procedure that we use for a clustering algorithm, the conducted AWE experiments and their results. In section 6.4, we outline some problems of a chosen feature extraction method for pronunciation clustering and propose the extension of the method for better feature extraction.

6.1 Feature extraction problem

In order to cluster pronunciations so that we could distinguish different pronunciations for a word, we need to get proper features that we can feed to a clustering algorithm. The main problem that we face is that speech fragments are usually different in time lengths. For example, for different speakers exactly the same word can take different time to pronounce (measured in milliseconds). Even pronouncing the same word several times for one particular speaker can be different in length. The common features that are used for speech-related computational tasks, such as MFCC or PLP (more information on these feature extraction techniques can be found in Dave (2013)), usually involve a sliding window over the whole speech input which results in variable-length feature vectors.

For the task of pronunciation clustering, having fixed-length features is very important since it involves standard machine learning techniques that cannot work on variable-length input. In order to get fixed-length feature vectors for variable-length audio segments, we researched the field of acoustic word embeddings (AWEs), which task is to map the variable-length input onto a fixed-dimensional space and to preserve the acoustic information of these inputs.

6.2 AWE review

Acoustic Word Embeddings (AWEs) are a fixed-dimensional representation of a speech signal. This idea was inspired by the textual word embeddings. In comparison to original word embeddings that for semantically similar words try to create similar vector representations, AWEs aim not at semantical similarity but acoustic similarity (see figure 14). The task of AWE space is to represent acoustically similar word segments of speech in such a way that similarly sounding utterances are clustered together. Example of the AWE space is illustrated in figure 15.

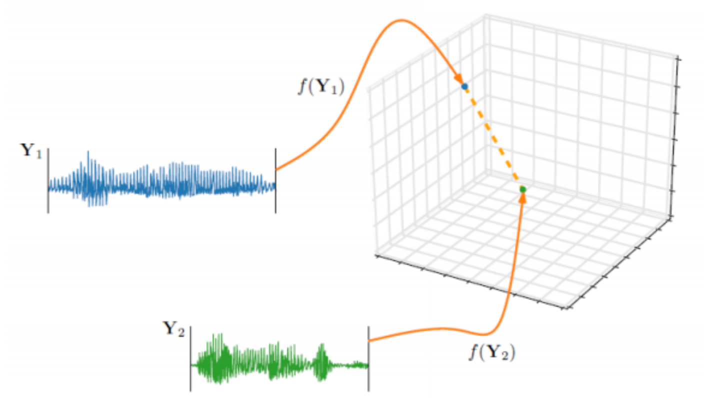

Figure 14: The process of AWE space creation. The picture is taken from Livescu

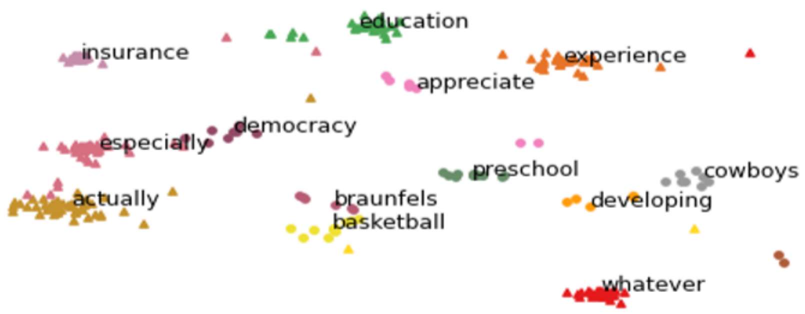

Figure 15: Illustration of AWE space. The picture is taken from Settle and Livescu (2016)

6.2.1 Research overview

Some investigations have already been done in the field of AWEs and quite promising results have been obtained. In this section, we review some prominent works in the field of AWEs and, where possible, compare their performance and define the method that we want to explore for our task of pronunciation clustering.

DTW approach

In Kamper et al. (2014), the authors approach quite a similar task to that we want to investigate. They worked in the field of grouping unlabelled acoustic word tokens according to their type (the word *cat* vs. the word *dog*). For such grouping, they needed fixed-dimensional acoustic features. They proposed a new approach for embedding a variable-length segment in a fixed-dimensional space. Their task was to find a mapping function which maps a variable-length segment to a high-dimensional space so that shorter distance in this resulting space corresponds to the greater similarity between two speech segments.

For measuring the similarity between two speech segments of different lengths, dynamic time warping (DTW) is usually exploited. However, having fixed-dimensional embeddings are more useful if we need any further data modelling. Given two frame-level acoustic feature vectors (a test vector and a vector from a reference set), the DTW alignment cost is calculated. Thus, for each test vector they constructed a reference vector that consists of the DTW cost values for each vector from a reference set. After applying dimensionality reduction to this reference vector, they got the desired fixed-dimensional embedding. It was one of the first attempts to extract AWEs and they got quite sufficient results.

CNN approach

In Bengio and Heigold (2014), the authors set the goal of revisiting the basic ASR architecture and replaced it with a fully data-driven approach based on deep neural network. This approach does not require any linguistic knowledge for lexicon and phonetic set description. The authors followed the intuition that for humans it is easier to segment acoustic sequences into words instead of phonemes and perceive them this way. Thus, they propose to rethink the ASR in terms of word probability given the acoustic evidence. Then, this model can be easily used in combination with classical decoding after adding a particular lattice re-scorer.

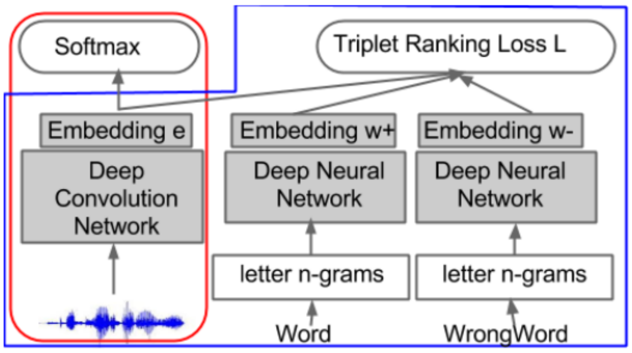

Figure 16: Deep architecture that was used to train acoustic word embeddings by Bengio and Heigold (2014). The picture is taken from Bengio and Heigold (2014).

As word input features, the authors used a bag of letter-n-grams where all possible combinations for the start and the end of a word were specified. To train the mapping between letter-n-gram word representation and the actual acoustic word embedding, the authors proposed the following architecture (illustrated in figure 16). The deep convolution network learns the dependence between acoustic sequence and posterior probability over words. Two deep neural networks return word embeddings of the same size that is returned by the convolution network given the letter-n-gram word representation. One of these two networks is fed with the word that is represented by the acoustic sequence that comes into the convolution network. Another neural network gets the word of the different word type as the first network. Both these networks share parameters with each other. The whole model is trained by minimizing the triplet ranking loss where the margin parameter denotes the desired distance between different word types in the embedding space and should be selected. Training of this model moves letter-n-gram representation near the acoustic word representation. The authors even report small improvement in WER for speech recognition task if this approach is used in the combination with the lattice re-scorer.

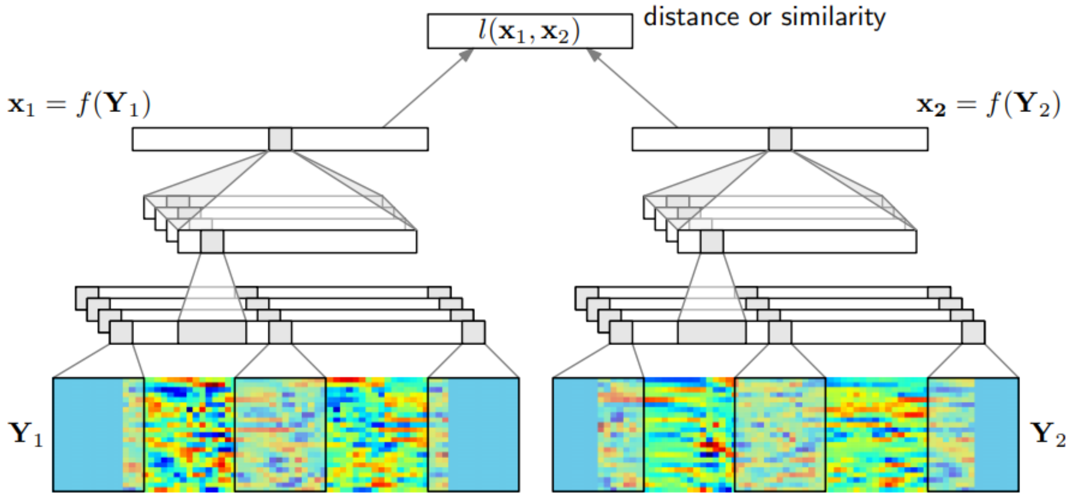
$l(\mathbf{x}_1, \mathbf{x}_2)$
distance or similarity
$\mathbf{x}_1 = f(\mathbf{Y}_1)$
$\mathbf{x}_2 = f(\mathbf{Y}_2)$
$\mathbf{Y}_1$
$\mathbf{Y}_2$

Figure 17: Siamese CNN for obtaining acoustic word embeddings from padded speech input used by Kamper et al. (2016). The picture is taken from Kamper et al. (2016).

In Kamper et al. (2016), the authors tried to discriminate between words directly in the derived embedding space. They also relied on a function that maps the variable-length segment to the fixed-dimensional space. Their approach involves Siamese convolutional neural network which was trained with the acoustic word pairs. This Siamese network (Bromley et al., 1994) is represented as a pair of tied networks, each of which gets the frame-level feature vector and produces the word embedding (see figure 17). The training proceeds due to a hinge loss (Gentile and Warmuth, 1999) that serves to separate different words by some margin while the network tries to maximize this distance between different word types. Thus, the minimum loss corresponds to closer distance for word pairs of the same type and bigger distance for word pairs of different types in the embedded space. Since CNN requires fixed-dimensional input, the authors zero-padded word segments to the same length that is equal to the longest word segment in the training data. To alleviate the effect of padding, convolutional and pooling layers were used.

In comparison to the other known CNN approaches, the Kamper et al. (2016) approach shows much better results with the hinge loss function. Moreover, the authors point that DTW technique shows to have worse results in comparison to their approach and is slower to compute. They also claim that paired supervision can generalize to unseen words and is suitable for low-resource languages.

RNN approach

In Settle and Livescu (2016), the authors explored the field of AWEs and proposed a new discriminative embedding model based on a recurrent neural network. An acoustic word embedding is represented as a function that takes a frame-level feature vector as an input and outputs the fixed-dimensional vector representation of a segment. The embedding model in their setup consists of a deep LSTM RNN with some number of stacked layers and final fully-connected layer (see figure 18). The fully-connected layer serves as a useful transformation that improves word representation.

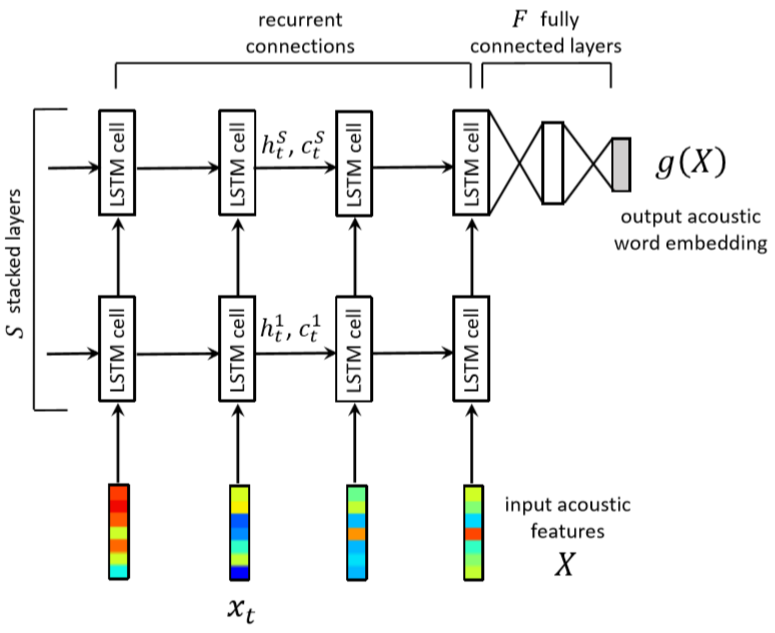

Figure 18: RNN architecture for obtaining acoustic word embeddings used by Settle and Livescu (2016). The picture is taken from Settle and Livescu (2016).

Siamese networks were trained with weak supervision in the form of segment pairs. The network gets three input segments: the anchor segment, the segment with the same to anchor label and the segment with the different to anchor label. The network is trained using "cos-hinge" loss. The authors also experimented with the way of sampling the examples of the different label. They proposed the two-step non-uniform sampling which targets pairs that violate margin constraint of the loss. This approach not only helps to speed up the training but also to obtain better results in comparison to uniform sampling.

The authors claim that their embedding approach significantly outperforms DTW as well as other known CNN approaches on word discrimination related tasks. The main advantage of the Siamese RNN training setting is that it is quite successful in distinguishing of different pronunciations since it has better relative distances between word clusters with similar and dissimilar pronunciations.

In Chung et al. (2016), the authors were also interested in the representation of variable-length audio segments with fixed-dimensional vectors. As the main domain of their research, they saw query-by-example Spoken Term Detection. They proposed an unsupervised setup which employs a Sequence-to-sequence Autoencoder since it does not take large amount of training data. Autoencoders (Liou et al., 2008) proved to be efficient for extracting representations but it needs a fixed-dimensional input which is not applicable to the audio input. This limitation of autoencoder was solved by introducing sequence-to-sequence autoencoders.

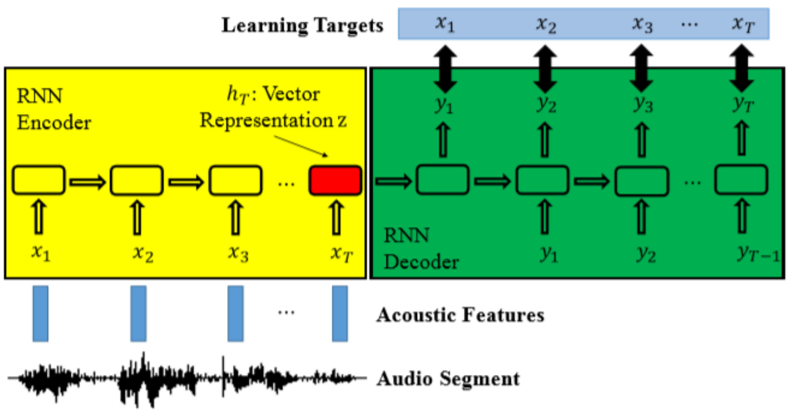

Figure 19: Sequence-to-sequence Autoencoder (RNN Encoder and RNN Decoder) for obtaining acoustic word embeddings used by Chung et al. (2016). The picture is taken from Chung et al. (2016).

In the research, they used the RNN encoder-decoder system to learn audio embeddings. Figure 19 shows the proposed architecture. The encoder RNN gets the input segment and updates the hidden vector which turns out to be a learned representation. The decoder RNN gets this learned representation and generates the output which should be as similar to the initial input sequence as possible. Thus, encoder and decoder are jointly trained by minimizing the reconstruction loss which is measured by the mean squared error. To learn more robust embeddings, the authors used the denoising sequence-to-sequence autoencoder: some noise was added to the input to better learn the internal structure of features.

The experiments they conducted are not comparable to the previous works in this field and, in our opinion, too local and not large-scale. However, they claim that their approach outputs good vector representations that catch the phonemic similarities and differences and can be used in real-world applications.

In Chen et al. (2015), the authors also tried to explore the field of fixed-dimensional AWE. Their primary motivation and application lie in the domain of detecting user-specified keywords for which query-by-example technique was the most appropriate. In their approach, the LSTM neural network is trained with word label targets. In their 2-layer LSTM, given the variable-length audio, the representation of the audio segment is got by using the hidden state vector from the second LSTM layer. As each hidden vector at the particular time encodes the information up to this particular time, they decided not to store all state vectors. They created a fixed-length representation by choosing some number of last state vectors. If the length of a segment occurs to be smaller than the chosen value for a fixed-dimensional representation, they pad the state vectors with zeros in front.

From the experiments that they have conducted specifically for their task, it is hard to evaluate the quality of the obtained representations, e.g. if the distance between the similarly pronounced words is smaller than of those with dissimilar pronunciations.

Some other approaches to AWE extraction were proposed for various tasks such as Zweig and Nguyen (2009), Maas et al. (2012), Voinea et al. (2014), Levin et al. (2015). However, they have results that are hard to compare with the rest of investigations and to interpret in terms of our needs.

Thus, the most successful results were obtained by using Siamese recurrent neural networks. In general, the RNN approach seems to be more suitable for catching the sequential information of audio data in comparison to the CNN approach. Moreover, the ability of RNNs to model sequential data can be beneficial for distinguishing word pronunciations. In Settle and Livescu (2016), they claim that their approach outperforms the well-known DTW technique and also other CNN approaches proposed in the literature. Other RNN approaches Chung et al. (2016), Chen et al. (2015) also look promising but they lack sufficient evaluation and testing in terms of word embedding extraction quality. That is why, we choose to explore Settle and Livescu (2016) approach for our experiments of feature extraction. In the following section, we investigate Settle and Livescu (2016) approach and explore how it performs in our word embeddings extraction and clustering tasks.

6.3 AWE extraction

In the pronunciation clustering task, we need to train the clustering algorithm to distinguish between different pronunciations of the same word if these different pronunciations exist. The training of such an algorithm needs a fixed-dimensional input. As has already been discussed at the beginning of this chapter, in case of speech data we lack this kind of fixed-dimensional features. Instead, we have audio segments which are of variable lengths. In order to extract fixed-dimensional feature vectors, we refer to Settle and Livescu (2016) and use their approach of AWE extraction. The approach that the authors used in Settle and Livescu (2016) was discussed in the previous section on AWE extraction methods overview, and in this section we cite technical details applicable to our task and data.

6.3.1 Data handling

We operate with the data that we got at the data preparation step discussed in section 5.2, namely MFCC features for word segments obtained from aligned utterance-based acoustic data. The corpus that we use in our investigation is different from what the authors of

Settle and Livescu (2016) used (VoxForge vs. Switchboard), thus, we need the particular data set construction for our data so that we can use the Siamese RNN training proposed in Settle and Livescu (2016).

First of all, we need to split the data into train and development sets. In Settle and Livescu (2016), the open-source data that they used was already split into train and development sets, 10k and 11k respectively. We have significantly more data in our corpus. After cleaning and processing the data at the data preparation step, we got 538 620 word utterances. We use the same strategy of almost equal splitting of the data into train and test sets since we find it justified for the model evaluation we do, which we discuss in section 6.3.3.

The main peculiarity of data processing for our training is data sampling. For the Siamese training, we need to have samples of three words: anchor, same and different words. Further, we explain these sampled words:

anchor a randomly chosen word utterance from a training set in a batch (for example, a word utterance corresponding to the word *cat*)

same a word utterance of the same word type as the chosen anchor word (for example, another word utterance corresponding to the word *cat*)

diff a word utterance of a different word type as the chosen anchor word (for example, a word utterance corresponding to the word *dog*)

In the Siamese network training, these sampled words are arranged as anchor-same and anchor-diff pairs, or just anchor-same-diff triplet.

For this sampling process, the training data should always have two utterances for the same word type in order to sample the anchor-same pair. Thus, we have this condition as a constraint on our training data: all word utterances that have frequency one in the training set are excluded from training. The original paper Settle and Livescu (2016) has this constraint even stronger, having the minimum frequency set to two. However, we do not see any justification to reduce the amount of training data more than it is necessary for the training procedure. Hence, we randomly split data in the proportion of 0.6 data for the train set and 0.4 data for the development set and, after that, apply the constraint for the training data which results in the more or less equal size of train and test sets.

In the sampling procedure, for every batch data point, we construct anchor-same-diff triplet. In order to have more fast and robust training, we exploit the strategy that is used by Settle and Livescu (2016): sampling more diffs while having fixed same. This helps to adjust the anchor word in the fixed-dimensional space faster. Thus, for each batch data point, we sample five diffs for the chosen anchor-sample pair. Due to this sampling procedure, the data that has to be stored in RAM expands five times. This data, in combination with memory costly evaluation part, does not fit into RAM. Thus, in contrast to Settle and Livescu (2016), we have to use the procedure of loading data from the file system which is slower but allows us to train on more data. We used the strategy when we load data per batch and after using the features of one batch we replace them with the features of a new batch. That allows us to fit the data into the memory for the training and to compute the memory costly evaluation part.

For our implementation, we use TensorFlow[21], a popular machine learning framework. Since we have the TensorFlow implementation of our training algorithm, we need to have the proper tensor for the batch and the variable-length input also becomes a problem. However, RNNs can handle variable-length inputs and, in the TensorFlow implementation of RNN, we just need to provide the lengths of each data point that is wrapped into a tensor. Thus, we can simply pad variable-length features with zeroes up to the longest length present in the training data in order to have the tensor with the proper shape. In our case, we have the artificial feature-length set to 398.

6.3.2 Neural network architecture

The algorithm for acoustic word embeddings training was successfully implemented by Settle and Livescu (2016) and stored open source[22]. We adapted this algorithm to train our own acoustic word embeddings on the VoxForge data. The implementation follows the methodology described in section 6.2. We exploit the bidirectional RNN in the Siamese training setup. The training proceeds with the anchor-same-diff samples using the triplet loss. Further, follow some technical details on the neural network architecture and its training.

The RNN we use is composed of three bidirectional dynamic recurrent layers with 256 LSTM cells in each layer. In between these bidirectional layers, a dropout layer with keep probability equal to 0.7 is present. The bidirectional layers are provided with the lengths of each data point for the particular batch. The loss that we use for the network training is the triplet loss that pays attention to the cosine distance between different words. The formula of the cos hinge loss is as follows:

$$l_{cos\ hinge} = max\left\{0, m + d_{cos}\left(x_a, x_s\right) - d_{cos}\left(x_a, x_d\right)\right\} \tag{8}$$

where $d_{cos}\left(x_1, x_2\right) = 1 - cos\left(x_1, x2\right)$ is a cosine distance between two vectors x_1 and x_2 and m is a margin for how far we want to place two vectors of different words from each other. In our experiment, we set the margin m equal to 0.5

For the minimization of the loss function, the Adam optimizer is used with the learning rate equal to 0.001. The network is trained for 100 epochs.

6.3.3 Evaluation

After every epoch, we use the trained model to evaluate our obtained acoustic word embeddings. For the evaluation of the whole model, we use the average precision metric (AP).

In order to introduce AP, we first outline the precision metric. Precision is one of commonly used metrics to assess the performance of a model. Precision of some class in

[21]`https://www.tensorflow.org/`

[22]`https://github.com/shane-settle/neural-acoustic-word-embeddings`

a classification task is the ratio of true positive (TP) and the total number of predicted positives, namely true positives (TP) and false positives (FP). The formula for precision is given as such:

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

Average precision is a metric that is used to work with rankings. The most common application of AP is for relevance document ranking in information retrieval.

The formula for AP is as follows:

$$AP = \frac{1}{GTP} \sum_{i=1}^{k} \frac{STP}{i} \tag{10}$$

where GTP means the total number of true positives in the class and STP refers to number of seen true positives till k^{th} element in the ranked list of objects.

The simple illustration of how AP is calculated can be seen in figure 20.

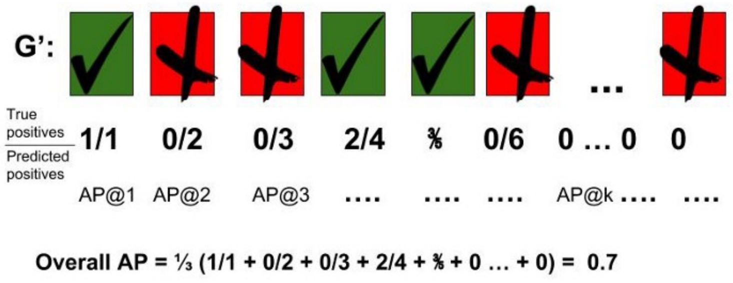

Figure 20: AP calculation for a list of ranked objects. The picture is taken from Towards-DataScience.com

For AWE quality evaluation, we consider all possible pairs of the development set data, extract AWEs for them and measure the cosine distance between them. The precision equals to 1 if word utterances of the same word type are the closest to each other. If among closest words there are some utterances of the different word type, precision for the anchor word type drops in the proportion of the number of not matching word types among the closest words (similarly as shown in figure 20). We take average precision by dividing the sum of all precisions for each same word type words by the number of same word type words.

6.3.4 Results

We conducted some experiments on AWE training, the results for which are present in table 2. In the table, the sizes of the training and development set are present in total words and unique words. The tilde is introduced due to the fact that, after the random split of the data into the training and development set and the filter procedure for the training set, we could have different number of utterances for the training set and the different number of unique words in each set. We also cite the minimum word frequency that was considered for including a word into the train set (denoted as word freq column in table 2). In the last column of the table, we provide the best AP scores that each trained model gained.

#	Train set		Dev set		Word freq	AP
	Size	Unique words	Size	Unique words		
1	∼ 15 000	∼ 1 100	12 000	∼ 2 644	3	0.495
2	∼ 180 000	∼ 5 250	120 000	∼ 5 758	2	0.590
3	∼ 420 000	∼ 6 550	120 000	∼ 5 750	2	0.600

Table 2: Three trained AWE models with different configurations and their AP results.

The first experiment is done as a comparison of our model trained on some subset of VoxForge data and the model described in (Settle and Livescu, 2016) trained on Switchboard data. After training the model, they got AP equal to 0.67 which is about 0.17 better than for our trained model. As the main reason for this result, we can name the difference in the chosen corpus (VoxForge for our experiment and Switchboard for (Settle and Livescu, 2016) experiment). The difference in the number of unique words for the same subset of the corpus can be the main reason for the difference in AP scores.

The second experiment shows the training on a bigger amount of VoxForge data and, which is more important, even more unique words. The bigger number of unique words helps to better distinguish between different word types due to the sampling procedure and triplet loss function while the number of utterances in the corpus does not play a big role. Using the second model from table 2, we could obtain a better AP than for the first model that can be considered comparable to Settle and Livescu (2016). The difference of 0.08 seems to be negligible in the light of different corpora that were used for these experiments and, more importantly, the bigger amount of development data that was used in our experiment which potentially increases the number of unique words for the comparison in the evaluation. In Settle and Livescu (2016), the authors do not mention the number of unique words, thus, we cannot scale our results in order to be able to compare them with their results.

The third experiment considers even more data for the training of the model. However, we could not improve further in comparison to the second model from table 2. We slightly increased average precision but not significantly. That shows that increasing of word types does not bring any improvement anymore and we probably reached the AP threshold on VoxForge data.

In comparison to Settle and Livescu (2016), our results are 0.07 worse in AP. This can be explained by different data sets that we use. The peculiarity of our VoxForge data set is that we chose the data with rich number of English dialects since it is important for the purpose of our research. Switchboard data set does not have a big variety in accents or dialects in the data. The AP score is highly influenced by the purity of the closest feature vectors. If we have words of different types intervened in the closest vectors for a word, the AP score drops. The accented speech makes the variance of the data higher and the likelihood to have similarly sounding words as closest ones is higher. Thus, we can accept our results being worse than those of Settle and Livescu (2016).

In order to better understand the AWE space, we picked several words that have similarity in the way they sound. The similarity can concern the beginning or the end of the word or can be assonant in overall. Among these words are *word, world, sort, sold, port, name, game*. We also chose the word *minute* in order to see the probable clusters that can be detected since we know that this word has several pronunciations in the corpus. The visualisation of AWE space is shown in figure 21.

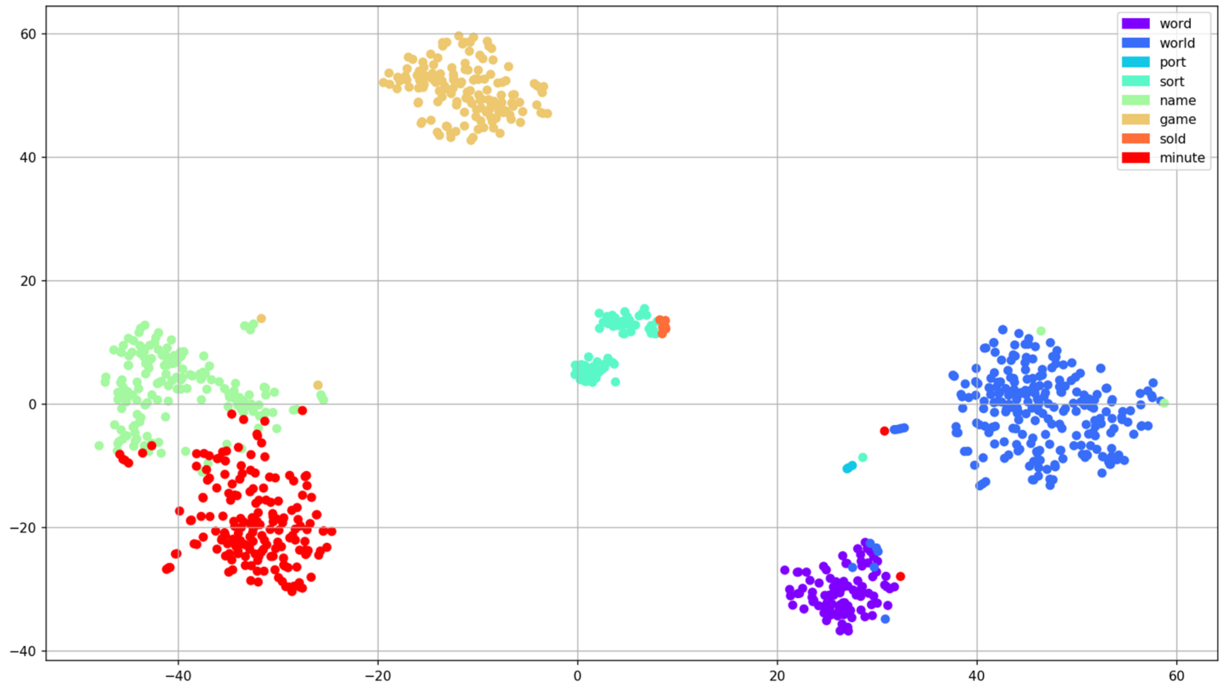

Figure 21: Embedding space for words *word, world, port, sort, name, game, sold, minute.*

In the figure, we can see that AWEs are grouped in clusters by different word types. There exist some outliers but many of them can be easily explained by the similarity of word pronunciations such as for *game* examples being in the *name* cluster, for *world* examples being in the *word* group, for *sort* example being near the *port* cluster, the vicinity of *sort* and *sold* clusters. Some visualized data is hard to explain in some meaningful way and they can be considered only as very noisy utterance examples such as two *minute* outliers and one *name* outlier. However, in general, we can admit that the quality of the AWEs assessed by the visualisation of clusters seems to be satisfactory.

Further, we visualize the subspace of AWEs for *minute* utterances. From the corpus data, we know that the word *minute* has multiple pronunciations present in the data set. The visualization of this particular subspace will help us to make sure that with our training of AWEs we can get quite nice clusters for different pronunciations within the same word cluster. The visualized AWEs for the word *minute* and their decodings obtained by the Viterbi algorithm as labels are presented in figure 22.

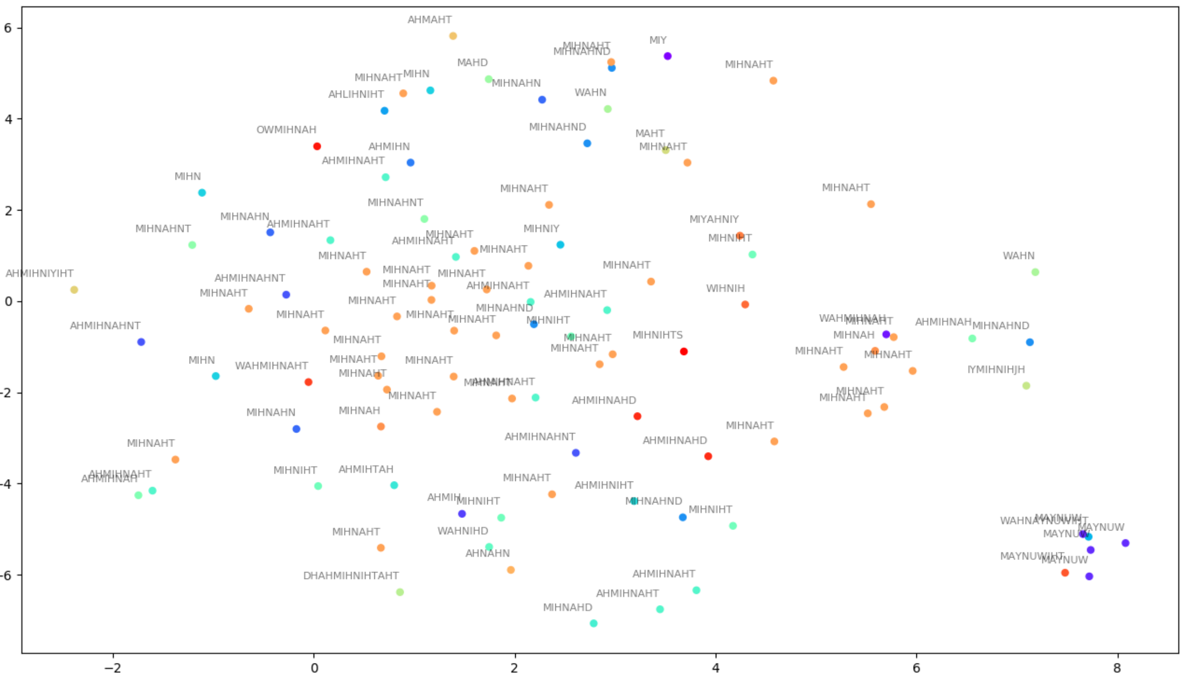

Figure 22: Pronunciation distribution for the word *minute*.

The word *minute* has three different pronunciations in CMU hand-crafted dictionary: *M IH N AH T*, *M AY N UW T* and *M AY N Y UW T*. In the figure, all different pronunciations decoded by the Viterbi algorithm are colour-coded (precise description of Viterbi algorithm and decoding procedure is described in section 2.1.1). Since we have many different decoded pronunciations and our system was trained purely on words, there is no surprise that we have quite mixed subspace for the word *minute*. However, we can detect a small pronunciation cluster that corresponds to the lexicon pronunciations *M AY N UW T* and *M AY N Y UW T* in the right lower corner of the figure. In this regard, the whole subspace of the word *minute* can be divided into two pronunciation clusters, the small right-corner cluster (*M AY N UW T* and *M AY N Y UW T* lexicon variants) and the big cluster consisting of the rest utterances (*M IH N AH T* lexicon variant).

However, among utterances for the word *minute*, there exist several other frequent pronunciations such as, for example, *M IH N IH T* or *M IH N AH N D* that we might want to include into the pronunciation dictionary. Thus, we are highly interested to see the AWEs for these decoded pronunciations as separate clusters in the AWE space. Hence, we consider the training of AWEs that we performed so far insufficient to distinguish between different pronunciations and we propose another training procedure to achieve better performance for the pronunciation differentiation task in section 6.4.

Nevertheless, we would like to see what causes the position of AWEs for a word. The particular place of a word embedding for an utterance in AWE space can be caused by the internal information of an audio segment that is different from phoneme acoustics such as noise, speed of the speech, gender of a speaker, particular speaker, etc. We did the listening test where we listened to utterances positioned in different places in the AWE space.

We could not derive any pattern for the data based on noise or speech speed. We also extracted some information from the corpus metadata for wav files such as speaker gender and pronunciation dialect. We plotted the results in figure 23 for visualizing the relation of gender to vector position in AWE space and the figure 24 for visualizing the relation of dialect to AWE position. We can witness that there is no any tractable dependency between gender or dialect and embedding position in AWE space.

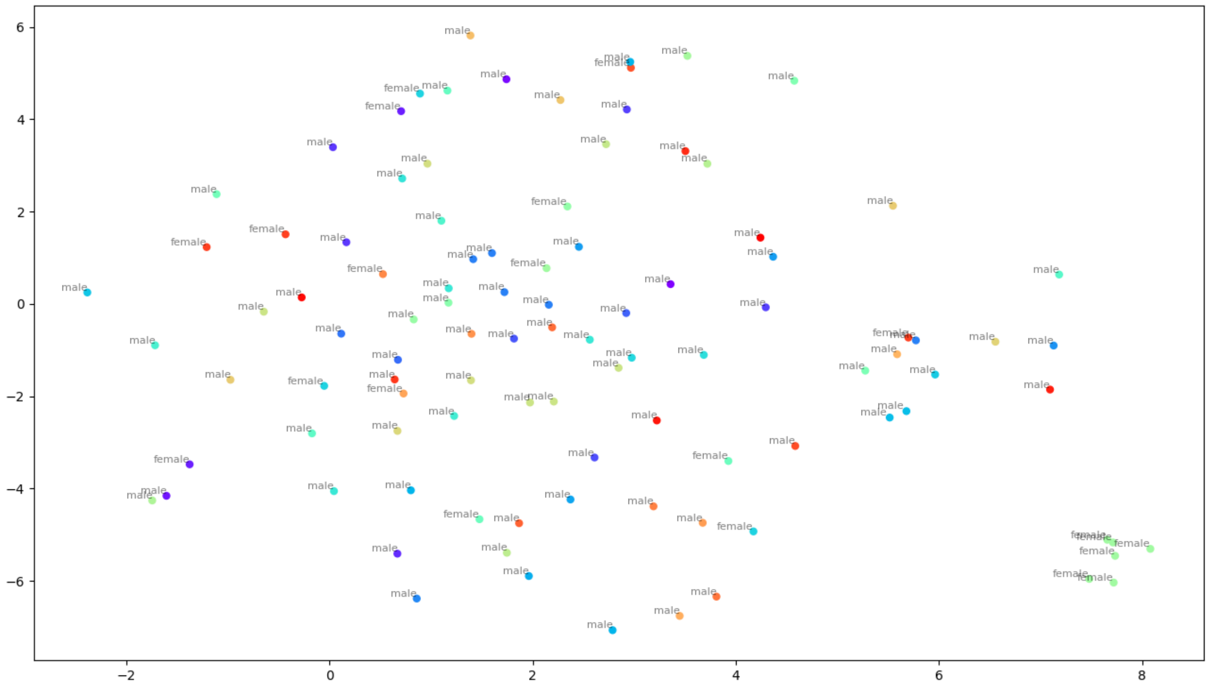

Figure 23: Pronunciation distribution for the word *minute* in relation to the speaker gender. Different pronunciation decodings are colour-coded.

This observation shows that RNN mostly pays attention to the acoustics and is not distracted by other factors such as the voice of a speaker, speech speed or noise which means that there is no objective obstacle for RNN not to distinguish between different pronunciations in AWE space.

6.4 AWE extraction extension

The way acoustic word embeddings are trained described in the previous section provides us with the opportunity to distinguish different words. Due to RNN nature, the obtained word embeddings bear a lot of information about the acoustics of a particular audio seg-

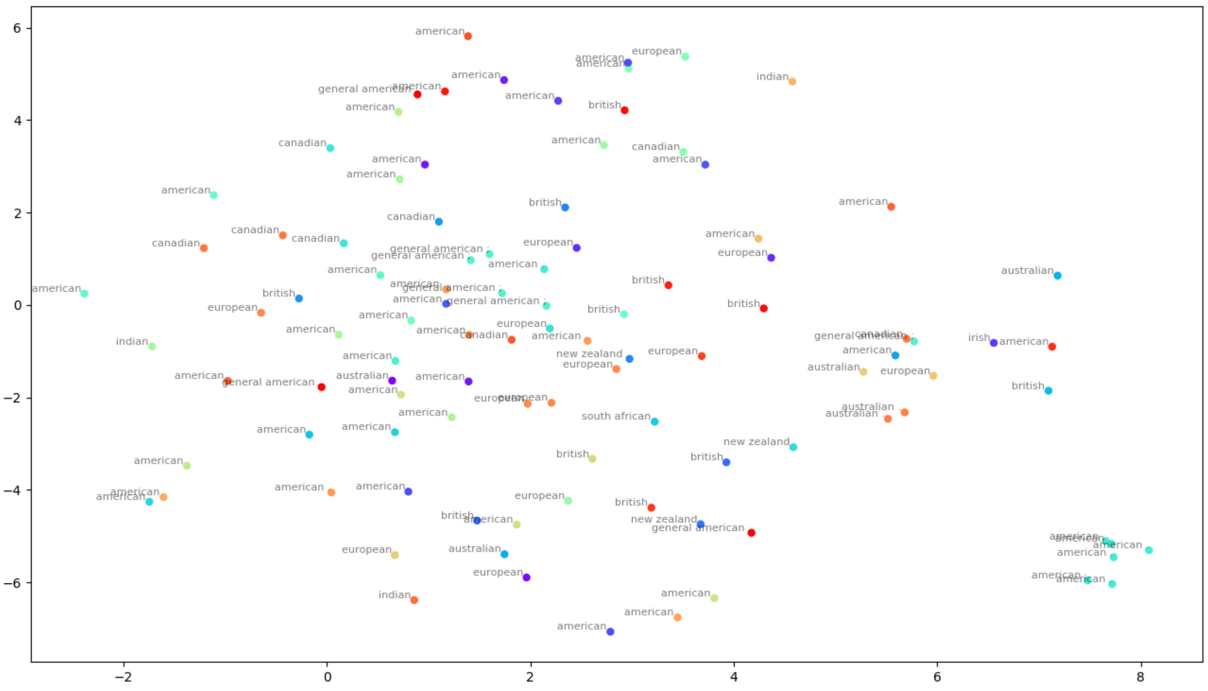

Figure 24: Pronunciation distribution for the word *minute* in relation to the speaker dialect. Different pronunciation decodings are color-coded.

ment and about the acoustic dependencies that occur in our data. However, this acoustic information that is caught by the recurrent layers seems to be insufficient for distinguishing between different pronunciations of the same word if we train our model that maps word-segment MFCC features onto embedding space only with word labels. Thus, we propose an extension for the training of acoustic word embeddings that might better distinguish between different pronunciation variants (illustrated in figure 25).

Since we focus on unsupervised pronunciation generation, we do not have any available source of pronunciations to directly train our model. However, we consider the Viterbi decoding of word utterances to be a good source of artificial pronunciations because they are obtained with a full reliance on provided acoustic information. Pronunciation sequences decoded by the Viterbi algorithm could be used for the training of our model to differentiate between different pronunciations. In this case, if we use decoded sequences as labels for model training, we completely rely on the decoding quality and fully trust the result of the Viterbi algorithm. Nevertheless, as we will see in the section 8.1 in table 3, not every decoded pronunciation produced by the Viterbi algorithm is meaningful and we hardly can tell which pronunciations belong to the same pronunciation variant. Thus, we propose a simple trick on how to train the model to approximate the position of the word utterance in the pronunciation space using Viterbi decoding.

For the training, we consider the following scenario: first, we train our model to differentiate word types in the embedding space as was described in 6.3 and, then, we switch to the pronunciation training. The pronunciation training is done in the same manner, using the Siamese RNN setup. The main difference is that we do not longer train the model at the word level. For this training, the most important part is the proper data sampling at the word pronunciation level for the weak supervision of the algorithm.

As we already pointed out, we use Viterbi decoding in order to train the model to better distinguish between different pronunciation variants. At each iteration of the training, one of word utterances from a batch is chosen to be an anchor. As a further step, the anchor word utterance and all word utterances from the training set that correspond to the same word as the anchor are decoded by the Viterbi algorithm. After decoding, we have multiple possible decoded sequences for the word. The situation can be that we will obtain all decoded pronunciations as different from each other or some of them might actually coincide. In the case when the anchor decoded sequence and the randomly taken decoded sequence are equivalent to each other, we definitely can sample this pair as anchor-same following the procedure described in Settle and Livescu (2016). However, when these utterances are dissimilar, we are unsure whether these utterances are of the same pronunciation or not.

We propose to exploit the edit-distance criterion in order to identify how different the decoded sequences are from each other. We introduce the threshold for the calculated edit-distance that draws the line between surely different pronunciations and somewhat similar ones. For our task, we use Damerau–Levenshtein edit distance (Damerau, 1964) which denotes the minimum number of operations needed to change one string into the other. The Damerau–Levenshtein distance includes transpositions in the list of possible operations in contrast to classical Levenshtein distance (Levenshtein, 1966) that allows only insertions, deletions and substitutions.

There exist two strategies to calculate edit-distance: unnormalized and normalized. Unnormalized edit distance corresponds to the total number of steps for changing one string into the other. Normalized edit distance normalizes this value by the number of characters in the reference string. We would like to try both these strategies of edit distance calculation for AWE training and compare them with each other in order to understand which suits better for this particular task.

The main task is to come up with the sampling procedure that can draw proper anchor-same-diff triplets. Calculating the edit distance can help to understand how much is the difference between two word utterances. In order to arrange triplet samples for training, we need to derive thresholds for putting word utterances in *same* or *diff* group based on obtained edit distance for two utterances.

For each chosen anchor, we calculate edit distances between the decoded anchor and every decoded word utterance of the same word type as the anchor. After we obtained all possible edit distance values for the particular anchor, the edit-distance thresholds for being in *same* or *diff* group are derived for this anchor. In order to calculate these thresholds, we consider the sorted array of unique values of edit distances for a particular anchor.

With unnormalized edit-distance, the threshold for word utterance being in the *same* group is floored first 20% of a sorted array of edit-distances. All values of edit distances that fall into the first 20% of the sorted array are considered to be small enough to neglect the difference in decoded phonetic sequences and to refer word utterances with these edit distances to the *same* group for training. The threshold for being in the *diff* group is floored last 20% of a sorted array of edit-distances. In the same manner, all the word utterances that have edit distance value larger than the defined threshold are considered to have quite different pronunciation from the anchor. We will denote the training that uses this threshold method with **unnorm_ed** suffix. For the normalized edit-distance, the threshold for word utterance being in the *same* group is 0.4 and for being in the *diff* group is 0.6. This threshold method for training will be referred further with **norm_ed** suffix.

Those utterance pairs that have edit distance above the *diff* threshold can be arranged into anchor-diff pairs. As anchor-same pairs, the utterances with the edit distance beyond the *same* threshold are chosen. This sampling procedure for AWE training extension based on the threshold for anchor-same and anchor-diff pairs helps to distinguish between completely different pronunciations according to Viterbi decoding and to keep those that are likely similar closer to each other in the embedding space. The figure 25 illustrates the sampling pipeline of the AWE extension algorithm.

The training is faster and more robust in the case if we sample more anchor-same-diff triplets for each anchor in the data. However, we would like to exploit as much data as we have. In this regard, we cannot choose only words with a high frequency of occurrences in the corpus to be able to sample more anchor-same-diff triplets because it reduces the amount of available data. Thus, we do not restrict the number of anchor-same or anchor-diff pairs for the anchor in contrast to what was done by Settle and Livescu (2016).

In turn, we just set the number of how many anchor-same and anchor-diff pairs we would like to have in total per batch independently from the number of such pairs per anchor. We set this number to be two. We cannot set this number higher because we cannot be sure that there are enough anchor-same-diff triplets in the data so that to arrange a complete batch. In case the anchor word type has more than two appropriate *diff* and two appropriate *same* in the training data, we save the rest for later compensation of lacking pairs for the batch. In the case when the anchor has less than two *diff* and two *same* utterances, we still use whatever is possible, namely one anchor-diff and one anchor-same pairs. However, in this situation, we have less number of anchor pairs per batch than is needed. That is why we compensate this lack by randomly sampling anchor-diff-same triplets from the anchors with the exceeding number of occurrences that we saved. This procedure is beneficial because the low-frequent words are not excluded and still present in the data and high-frequency words can have the greater impact on the training due to the random sampling in compensation procedure since they have higher chance to be chosen.

The whole data sampling for this AWE training extension was implemented in Python using the open-source `pyxDamerauLevenshtein` library for Damerau-Levenshtein edit-distance calculation[23].

This is the whole data sampling procedure on which we rely for the discriminative training of pronunciations. The results for models trained with edit-distance and their comparison with other trained models in our experiment are presented in chapter 9.

[23]`https://github.com/gfairchild/pyxDamerauLevenshtein`

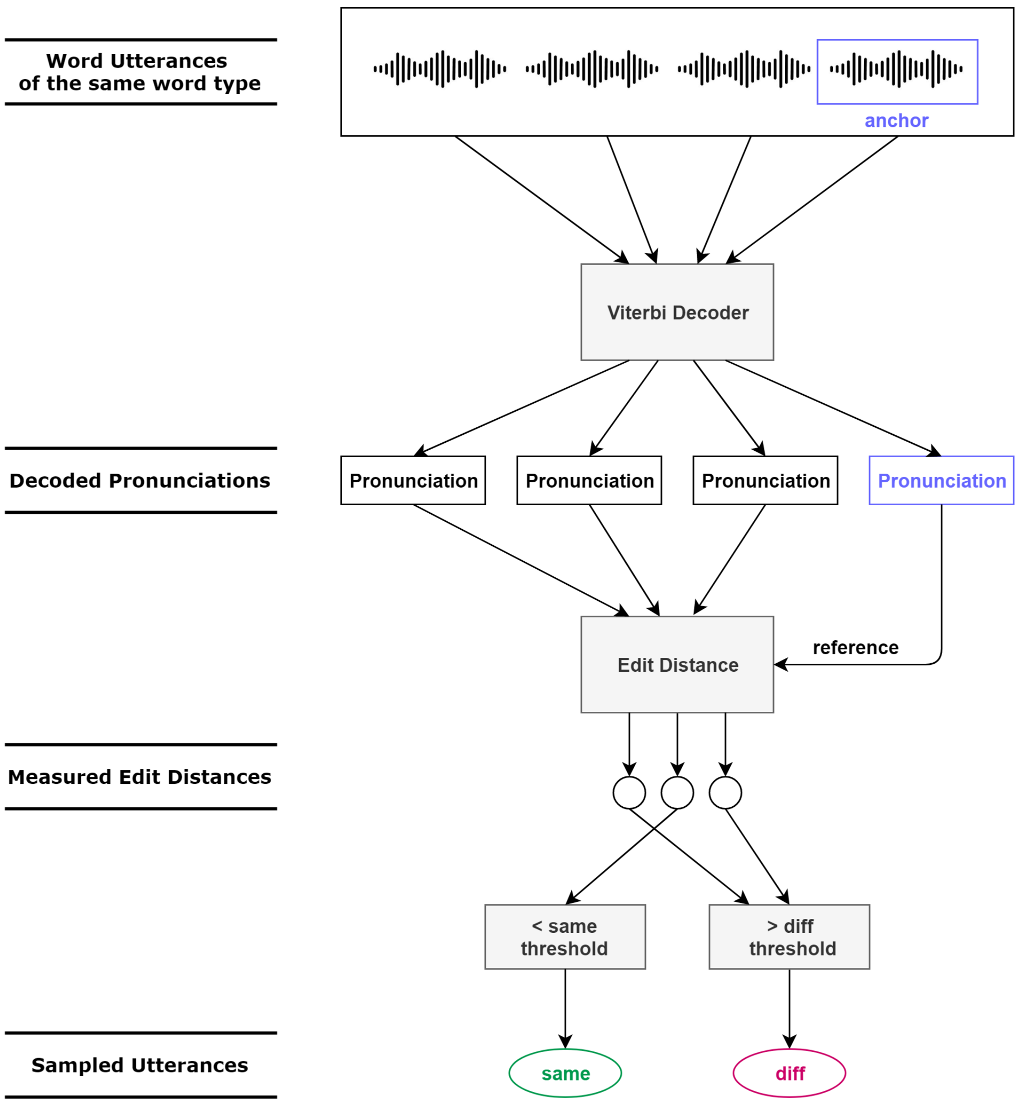

Figure 25: Illustration of the sampling procedure for AWE training extension.

7 Clustering

In order to generate a pronunciation dictionary completely based on data, we propose the method that exploits cluster analysis for pronunciation derivation. Clustering can be beneficial because there exist many words pronounced in a similar way that could correspond to one pronunciation variant and we would like to detect these variants. The main idea of the pronunciation clustering is to cluster word utterances that correspond to different pronunciations for a word. For this purpose, we extracted AWEs for word utterances so to get their fixed-dimensional features. We needed fixed-dimensional representations of speech fragments that correspond to words in order to be able to apply machine learning clustering algorithms. After the clustering of utterances for different words is performed, we hope to obtain meaningful clusters that denote different pronunciations for a word that can be put into the pronunciation dictionary.

This chapter describes the clustering step of the proposed pronunciation generation system. Section 7.1 gives information on the clustering algorithms that we chose for our experiments on pronunciation clustering. Section 7.2 is devoted to the discussion of the results obtained by the clustering algorithms presented in section 7.1.

7.1 Clustering algorithms review

In this section, we outline all the clustering methods that we would like to try in our experiment on pronunciation clustering. We discuss the theory of various clustering algorithms and their applicability for our task.

There exist various types of cluster analysis because the notion of a cluster can be defined from many different perspectives. We would like to concentrate on the clustering types that can automatically detect the number of clusters. However, we also consider some simple clustering types for evaluation.

The list of clustering types and corresponding algorithms that we would like to experiment with is the following:

1. Centroid-based clustering
    - K-Means
2. Probabilistic clustering
    - GMM
3. Density-based clustering
    - DBSCAN
    - OPTICS
4. Deep clustering

- SOM
- GNG

Further in this section, we discuss each clustering method from the list presented above.

7.1.1 K-Means

We consider using the K-Means algorithm as a baseline system for our experiments on pronunciation clustering. This method of clustering needs to know into how many groups to divide input, thus, K denotes the number of desired clusters. The K-means clustering algorithm uses the notion of centroids for data clustering. The algorithm starts with the initial estimate for the position of centroids and continues with the iterative procedure of centroid adjustments. This approach of parameter estimation is called Expectation-Maximization (Dempster et al., 1977). In the E-step, each data point is assigned to the nearest centroid that defines one of the clusters. The assignment relies on calculation of the squared Euclidean distance between a centroid and a data point. The M-step recomputes centroids by averaging all data points assigned to a centroid cluster. The K-Means algorithm iterates between E- and M-steps until some stopping criteria are satisfied such as a maximum number of iterations is reached, the position of centroids do not change significantly, etc.

The K-Means algorithm is great for capturing spherical-shaped clusters but, in the case of pronunciation clustering, we cannot be sure that the AWE model arranges data into spheres. For word-level AWEs, it can be easier for the model to arrange embeddings into spherical-shaped word clusters. However, in the case of pronunciation differentiation, it is more likely for one word pronunciation to flow slowly into another which will not give a spherical pronunciation cluster. Nevertheless, since we cannot assess the data shape for each word in the corpus, we still can expect K-means clustering to perform relatively well for the task of pronunciation clustering and use it as a baseline system for our clustering experiments.

7.1.2 GMM

Another clustering method we would like to explore is the Gaussian Mixture Model (GMM). The GMM finds a mixture of multivariate Gaussian probability distributions that best models the input data. The task for the GMM is to fit k Gaussians to the training data by estimating their parameters such as mean μ and variance Σ for each cluster. The number of components k must be chosen manually. The GMM also learns the weight ϕ for each modelled Gaussian. Thus, the GMM is a weighted average of k Gaussian distributions:

$$p(x) = \sum_{j=1}^{k} \phi_j \mathcal{N}(x; \mu_j, \Sigma_j) \tag{11}$$

For the parameter estimation, the Expectation-Maximization algorithm is used which we already discussed with the application to K-Means. In the E-step, for each data point, the probability to be generated by each of k Gaussians is calculated. In the M-step, the weights, means and covariance matrices are updated according to new data probabilities. The iterative training including these two steps proceeds until the stopping criteria are met. After all parameters are learnt, we can compute probabilities for each test data point to belong to each of Gaussian probability distributions, or simply clusters, based on its distance from a particular distribution.

The GMM fits our task of pronunciation clustering well since we expect that the data for each pronunciation will follow the Gaussian distribution. However, it can occur that we do not have enough data for a pronunciation variant to make a Gaussian distribution. Another problem is that we need to define a number of components for the GMM and keep it fixed while the number of pronunciations can be different for every distinct word. To deal with this issue, we propose the selection method for the number of components for each word that utilizes the Bayesian Information Criterion (BIC) (Schwarz et al., 1978). This criterion gives an estimation on how well the GMM predicts the data. Among some final set of models, the model with the lowest BIC score is preferred. The BIC is based on likelihood function and, in order to avoid overfitting, it penalizes models with a big number of clusters.

7.1.3 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm that separates clusters with high density from clusters with low density (Ester et al., 1996). DBSCAN searches for areas with high density in data points comparing them to the areas with low data density. For the clustering, DBSCAN needs two parameters: ϵ that specifies the maximum distance for the points to form a cluster and $minPts$ that defines the minimum number of points to form a cluster. The algorithm starts with a random data point and searches for all neighbours of it within ϵ distance. If the number of neighbours is greater or equal to $minPts$, the cluster can be initialized with this data point and all its neighbours. Otherwise, the data point is labelled as noise unless it is found among another data point's neighbours later. Then, the process of finding neighbours continues for the rest of the unconsidered data points in a cluster until the full cluster is discovered. If there are left unprocessed data points, they will initiate a new density-based cluster or will be labelled as noise.

The advantages of the DBSCAN algorithm are that it can model data of different shapes and can handle outliers well since it considers them as low-density regions. We consider this a benefit for the pronunciation clustering task because the speech domain is prone to have outliers in word pronunciations due to many phonological processes. Another advantage of this clustering algorithm for our application is that DBSCAN can derive the number of clusters automatically based on the data provided. The main disadvantage of DBSCAN is that it cannot handle high-density clusters of varying density.

7.1.4 OPTICS

The Ordering Points to Identify Cluster Structure (OPTICS) algorithm is also a density-based clustering method and can be seen as an extension of DBSCAN (Ankerst et al., 1999). It was introduced to tackle the problem that DBSCAN could not handle, namely the varying densities in clusters. OPTICS works similarly to DBSCAN but it does not provide cluster assignment directly but generates the ordering of data points based on their distance from each other. The data points of denser clusters are listed closer to each other in this ordering.

To present the OPTICS algorithm, we need to introduce two concepts. The first concept is core distance which is the minimum ϵ distance for a data point in order to initiate a cluster given $minPts$ parameter. The second one is reachability distance which implies the minimum distance between two data points p and o if o can initiate a cluster. The algorithm starts with computing core distances for all data points and, then, looping over all data to update reachability distances. The ordering of data point processing is defined by the reachability distance (first goes the data point with the smallest reachability distance). This way OPTICS keeps data points of dense clusters close to each other. All calculated reachability distances comprise a reachability plot (may be seen as a dendrogram). This reachability plot is used to do the cluster assignment to data points. Data points of the same cluster have a low reachability distance and can be seen as valleys in the reachability plot. By detecting these valleys by different means, clusters can be assigned.

In addition to advantages borrowed from DBSCAN, as an advantage of OPTICS algorithm for our task of pronunciation clustering, we see that OPTICS can detect clusters of varying densities which is a possible situation for the pronunciation distribution. For example, many pronunciations can vary in some particular part in different ways creating a big cluster for the variation in terms of place but having small clusters for different types of variation. We would like to detect smaller clusters within this big cluster since this variation can be meaningful.

7.1.5 SOM

Self-organizing map (SOM) is a clustering algorithm that exploits the notion of neural networks and works as a dimension reduction technique (Kohonen, 1982). Neurons in SOM are arranged in a 2-dimensional grid and have a rectangular or hexagonal shape. The main task of SOM is to adapt the grid to the internal shape of the data so neurons of the grid grouped around dense areas in data. This way, many neurons in one area of the grid could represent underlying clusters in the data.

The algorithm starts with random positioning of neurons of the grid in the data space. Then, SOM starts with data point selection and moves the closest neuron to this data point closer to it. The distance of the neuron shift is determined by the learning rate and it decreases with a number of iterations. SOM also moves all the neuron's neighbours in the grid closer to the selected data point while further neighbours are moved by a smaller distance. Neuron's neighbours are identified by the radius around the neuron which is also decreased with each iteration of the algorithm. The SOM algorithm is highly dependent on a learning rate and neuron radius parameters, thus, they must be tuned as well as the number of neurons in the grid. The resulting grid of SOM (can be seen in figure 26) helps to visualize the data and identify existing clusters in the lower dimension.

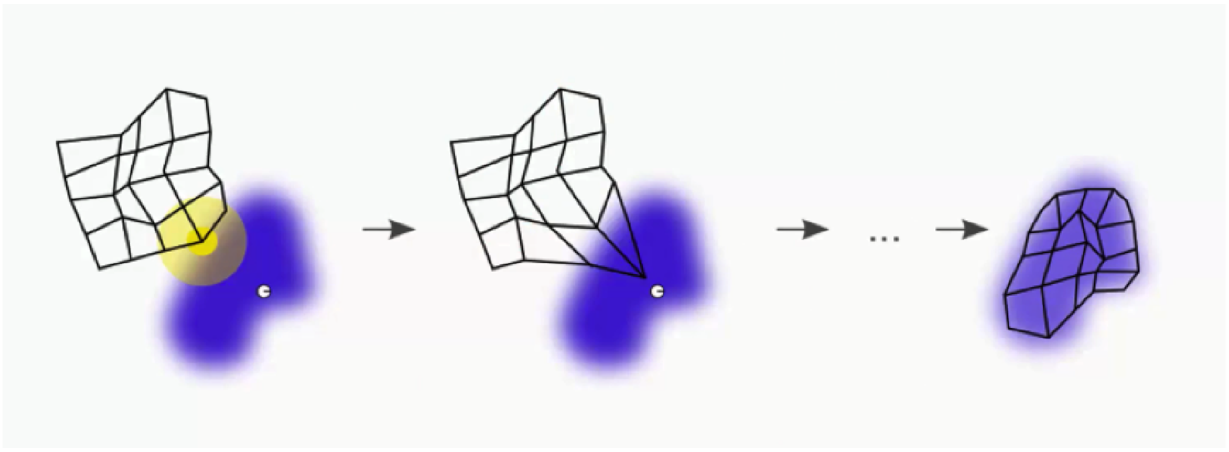

Figure 26: Visualization of SOM algorithm. The picture is taken from SuperDataScience.com

The advantage of the SOM algorithm for the task of pronunciation clustering is that it can learn the shape of the data with high precision and can identify intrinsic clusters without need to know the resulting number of clusters. For the AWE space, where the space of data points of word utterances can be complex, this algorithm behaviour can be beneficial.

7.1.6 GNG

Growing Neural Gas (GNG) (Martinetz et al., 1991) is a neural clustering algorithm inspired by the SOM algorithm which also learns the topology of the data. In comparison to SOM, GNG does not require the number of neurons in the grid to be specified. In the case of GNG, the number of neurons increases while the algorithm proceeds based on the specified conditions of the algorithm.

GNG starts with two randomly initialized neurons in the data space. At each iteration, the algorithm selects a data point and, similarly to SOM, moves the nearest neuron to this data point closer to it. The neighbours of the closest neuron are also moved closer to this data point. Then, the algorithm searches for the second closest neuron to the data point. If the two closest to the data point neurons are not connected, GNG connects them. If they are connected, the algorithm sets the age value to zero. The age value is a measurement for the edges between neurons defining the distance between data points and their corresponding neurons. If an edge between neurons has a quite large value, the edge is deleted. If a neuron got detached from others after this, it is deleted. After every constant number of iterations, the cumulative error of each neuron is calculated. The cumulative error is a sum of distances from a neuron to each data point. Between the worst-performing neuron and its worst-performing neighbour, a new neuron is inserted. The GNG proceeds until the stopping condition is met.

We can consider this algorithm for pronunciation clustering application since the algorithm works similarly to SOM with all its advantages for our task. GNG even overcame the limitation of SOM, that requires the number of neurons to be defined, and can determine the place where the data is represented the worst and can refine it accordingly.

7.2 Clustering experiments

The main idea of data-driven pronunciation generation is to include into the lexicon, whenever possible, all the meaningful pronunciation variants for a word that were found in the data. Here we face the question of what a pronunciation variant actually is. It is a pretty difficult task to define what should be considered as a pronunciation variant for an ASR system. Probably, we want to think not only of word utterances pronounced by English speakers of different dialects or two drastically different pronunciations that vary in several phonemes as pronunciation variants. We could also consider a word pronunciation influenced by the context at word boundaries as different pronunciations.

The main criterion that we define for considering word utterances to be pronunciation variants is that it would be beneficial for the ASR system to include them into the pronunciation lexicon. However, this vague definition of a pronunciation variant influences the possibility to evaluate pronunciation variants, for example, in clustering tasks, since clustering is an unsupervised technique of data analysis. Thus, in order to evaluate our clustering experiments, we relied on data visualisation as a manual verification for the clustering quality at the intermediate step. Nevertheless, at the final stages of our experiment, we evaluated the pronunciation clustering quality in terms of our primary metric of interest - the ASR performance. These results are presented in chapter 9.

We conducted several experiments on using different clustering methods that we described in the previous section. The clustering procedure for our data is as follows: for all utterances of a particular word, we take their MFCC feature vectors and extract fixed-dimensional vector representations with the use of the AWEs extractor (described in section 6.3); then, the obtained AWEs of all word utterances are used as features for a clustering algorithm. For the clustering experiments, the AWE model is used without the extension proposed in section 6.4.

7.2.1 Results

Firstly, we explored the simple K-Means clustering. The main disadvantage of this clustering method is that it needs to have a predefined number of clusters. We decided to try this approach as a simple baseline. We tried several numbers of clusters and in figure 28 in attachment A.1 we present the best result corresponding to four clusters. From figure 28, we cannot see any pattern for the clustering output except for the low right cluster in the corner which corresponds to the joint pronunciations *M AY N UW T* and *M AY N*

Y UW T for the word *minute* in the CMU dictionary. We could consider this clustering good, however, without defining a number of clusters for each word separately we cannot achieve good performance for the whole system. Thus, with this limitation of K-Means, this approach is not scalable for the whole data set.

Then, we tried two density-based clustering algorithms, OPTICS and DBSCAN. The visualisations of results for both clustering methods are shown in figure 29 and figure 30 in attachment A.1. These methods can derive the number of clusters based on the density of data points and it is not needed to assign them manually. For the word *minute*, OPTICS (figure 30) and DBSCAN (figure 29) algorithm defined two clusters. The inspection of the results suggests that there is no pattern for the clustering for both algorithms. At least, the results do not show any clusters of the identically or similarly decoded Viterbi sequences.

This result makes us conclude that density-based clustering is not appropriate for our task of clustering pronunciations in the AWE space. In the case of AWEs, we do not use actual acoustic features of a speech segment but we artificially create a new feature vector for this segment which not necessarily preserves the sufficient amount of acoustic information. This specific of AWEs training makes the chances of acoustically very similar segments be mapped very close to each other quite small. All the dense areas in our AWE space are more due to randomness. Thus, density-based clustering cannot catch the original acoustic similarities of the data but only the artificially created ones. Hence, we cannot rely much on this type of clustering in our task.

Then, we investigated two neural network approaches, Self-Organizing Maps (SOM) and Growing Neural Gas (GNG). The SOM algorithm detected five clusters in data and GNG detected only one cluster. The GNG result in figure 32 in attachment A.1 can be reasonable to some extent but in the case of the word *minute*, we would expect the algorithm to detect at least two clusters that are present in the hand-crafted dictionary. It is possible that GNG can detect only drastic differences in data. If so, this clustering method seems to be not suitable for the pronunciation detection task. The SOM results show more random allocation of cluster labels (figure 31 in attachment A.1). We cannot track any particular pattern for this. Thus, we consider this clustering method inappropriate for our clustering task.

Finally, we tried the Gaussian Mixture Model as a clustering method. This method is quite similar to K-Means and also expects the number of Gaussian components to be predefined. Based on the visualization in figure 33 in attachment A.1, we can conclude that the clustering results of GMM are quite similar to K-Means and this algorithm can detect the lower right corner cluster which we would like to have detected by the clustering algorithm.

7.2.2 Discussion

Having these results[24], we can conclude that none of the algorithms that have internal mechanisms for deriving the number of clusters can cope with the task of pronunciation clustering in the AWE space. This outcome can be influenced by several reasons. First of all, the results of a clustering algorithm are highly dependent on the quality of features fed to a clustering algorithm. Since we train the AWE space and do not use initial acoustic features, that can be among the reasons for the clustering to fail. The trained feature vectors might not contain the information that is required for a clustering algorithm to detect reasonable clusters for our data. That is the sign that our trained AWE space is not informative enough or even misleading and has to be improved further.

Another reason for clustering to seem to perform poorly can be the quality of the Viterbi decoding. Since for better understanding of the data visualization we provide labels represented by Viterbi decoded sequences for a particular word utterance, our judgements regarding the quality of clustering highly dependent on the labels that we provide. However, it might be the case that our AWE space represents data in a qualitative and reasonable way and clustering performs nicely on these AWEs. The main problem, in this case, is that we do not see that our clustering has good results because the Viterbi decoding does not show us that in the visualizations by providing misleading labels (decoded pronunciation sequences). If the output of Viterbi decoding is very noisy and often makes mistakes, we can miss the actual quality of clustering on AWEs since we highly rely on the labels in our judgements. If the problem is in the quality of the Viterbi decoding, we cannot improve this much.

The question is which part of our pipeline we trust more, Viterbi decoding or trained AWEs. If we trust the Viterbi algorithm more, then we are satisfied with the quality of clustering algorithms that derive the number of clusters in data on their own and the solution would be to improve our fixed-dimensional AWEs. As the improving strategy, the suggestion proposed in section 6.4 can be used. If we trust our trained AWEs more, we might want to use our best performing clustering methods such as K-Means or GMM and try to eliminate the need for the predefined number of clusters which is a limitation for our task of pronunciation clustering. For this purpose, we can incorporate the BIC as a stopping criterion into these clustering methods (Chen and Gopalakrishnan, 1998).

7.2.3 Improving clustering results

Based on the considerations presented in the previous section, we tried to improve the clustering results with the defined options. We tried using the most successful clustering algorithm, namely the GMM, on features extracted by the AWE model trained with the

[24] We also conducted small scale ASR experiment that evaluates WER score for all clustering algorithms. GNG achieved the best performance by clustering almost all pronunciations to one cluster (cluster rate is 1.011). GMM performed better than the rest of the clustering algorithms in this evaluation. Considering our task of introducing multiple meaningful pronunciations in the lexicon, the results of GNG seem to be unsatisfactory.

extension proposed in section 6.4 and applying the BIC as the stopping criterion for deciding how many pronunciation variants of a word are present in the corpus. The visualisation of the clustering results for the word *minute* is shown in figure 27.

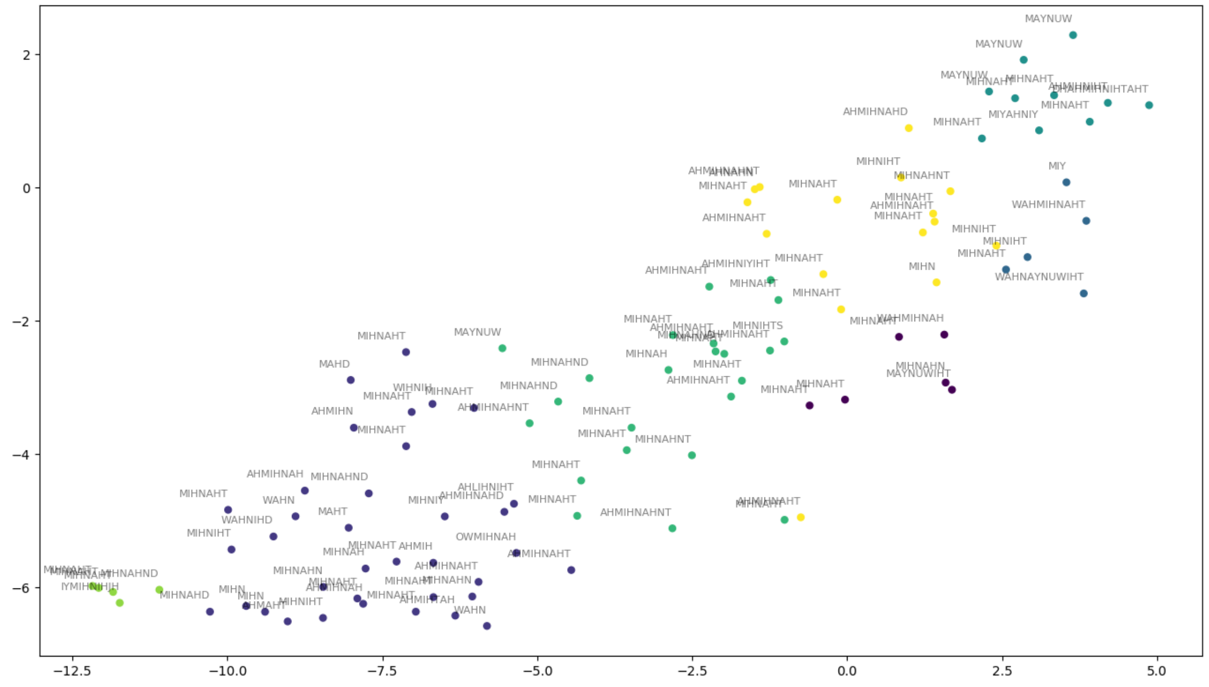

Figure 27: Clustering results obtained by the GMM algorithm with the BIC and performed on features extracted from the AWE model trained with unnormalized edit-distance.

We can see that the results presented in figure 27 are not satisfactory. In the figure, we cannot see the basis for the algorithm to choose this number of clusters and any pattern derived from the Viterbi decoded labels. This result can be due to insufficient time for the extension algorithm training so that it cannot perform good on the chosen word *minute* yet. Another reason can be that we chose quite a big margin for anchor-same-diff sampling for the AWE training. Since for this visualisation we plotted the result of **unnorm_ed** AWE model, we can suggest that the 20% of first edit-distances for the *same* group and 20% of last edit-distances for the *diff* group can be too broad to draw the clear line between different pronunciations in the AWE space.

Nevertheless, we try to use this clustering model for the pronunciation generation task in order to assess the quality of lexicon generation model based on cluster analysis.

8 Lexicon Generation

The primary aim of this research is to investigate different approaches of pronunciation generation and compare them with each other in terms of their influence on ASR performance. In this chapter, we describe different possible ways of lexicon generation for ASR.

Section 8.1 introduces the decoding procedure that is used in the majority of our experiments on lexicon generation and presents a small experiment on the understanding of the decoding output. In section 8.2 and section 8.3, we define the benchmark and baseline lexicons, respectively. In section 8.4, we discuss lexicons that can be generated by the proposed pronunciation generation system based on clustering analysis. Section 8.5 proposes additional lexicons for comparison and investigates methods for overall lexicon improvement. Section 8.6 provides some general information and comparative statistics for the considered lexicons.

8.1 Decoding

For the purpose of pronunciation generation for the dictionary entries, we would like to exploit as much acoustic information provided by the audio signal as possible. The main approach of phonetic sequence generation for an audio input is ASR decoding. The most known and widely used method for ASR decoding is the one that uses the Viterbi algorithm for finding the best phonetic sequence for the input. We would like to use this algorithm in our experiments on pronunciation generation since it mostly relies on the acoustic information which makes the decoding procedure to be predominantly data-driven. The Viterbi decoding algorithm was introduced in section 2.1.1. Further, in this section, we discuss how Viterbi decoding is used in our pronunciation generation system.

8.1.1 Decoding procedure

We implemented our own version of the Viterbi algorithm in Python in order to use it for our experiments on lexicon generation. We use a simple Viterbi algorithm without pruning since the decoding of words is not a really costly procedure. All the computations in our implementation of the Viterbi algorithm are performed in the log space in order to compute faster and prevent "underflow" when very small probabilities are rounded to zero.

For the computation of the Viterbi algorithm, we provide three types of likelihoods: state initial, state transition and emission likelihoods. All these likelihoods can be extracted with the help of the trained ASR model.

State transition probabilities are extracted from the phone alignments obtained by the ASR model trained on LDA + MLLT features applying SAT (for more information on phone alignments refer to section 5.2). As phone alignment material for transition probability extraction, the whole training set of VoxForge data is used. First, we align the audio signal with the sequence that the ASR model has decoded in an audio-segment-to-phone manner. With the custom Python script, we extract the probability to transit from one phone to another from the obtained phone alignments.

For the sake of initial probabilities extraction, we cannot rely on phone alignments. This is due to the fact that we deal with isolated words in our experiments and the ASR model aligns the training data consisting of utterances and not separate words. To overcome this, we use discrete uniform distribution for the phone initial probabilities. That is motivated by the assumption that a word can start with any phone with almost equal probability and the actual difference in phone initial probability distribution is insignificant. We conducted several small fetch-factoring experiments for initial probabilities and, based on these results, we claim that we can neglect this difference in initial probabilities extraction since initial probabilities do not influence much the decoding output[25]. The state initial probabilities were extracted by a custom Python script.

After transition and initial probabilities extraction, these probabilities are moved to the log space.

For the emission probabilities extraction, we use already trained neural network ASR model. The emission likelihoods were obtained by performing the forward pass of the Kaldi TDNN model[26] that was trained following the LibriSpeech Kaldi recipe for DNN training of ASR system. The output of the TDNN model is represented as a matrix of pdf-level likelihoods for each frame. For our task of Viterbi decoding, we aim to perform the phone-level decoding instead of pdf-level one. To overcome this representation problem of the forward pass, we first need to normalize the pdf-level likelihoods per frame since they come unnormalized from the neural network model. After the normalization of likelihoods in the log space, we need to find the correspondence between pdfs and phones. The mapping can be obtained from the `copy-transition-model` Kaldi script[27]. It takes the trained model as an input and outputs the pdf-to-phoneme mapping. We extract the phone-level likelihoods by normalizing the sum of all pdfs belonging to the particular phone per frame. This is also done in the log space in order not to lose the numerical precision. Thus, we successfully obtain the phone-level likelihoods from the likelihoods of our neural network model by a custom Python script.

As already mentioned, the extraction of initial and transition likelihoods is done based on the whole training set of the VoxForge corpus. These likelihoods can be used for decoding any word in the VoxForge corpus since they are shared. However, to decode some particular word, we need to have the emission likelihoods corresponding to this exact word. Thus, for the whole set of words that we would like to decode we need to obtain their emission likelihoods with the forward pass of the ASR model. All our experiments consider the lexicon generation based on the training set of VoxForge corpus. Thus, for the lexicon generation experiments, we extract all necessary emission likelihoods for words in the training set of the VoxForge corpus and then decode all words in it. The pronunciation sequences decoded by our Viterbi algorithm are further used for the lexicon generation experiments.

[25]We introduced different fetch factors to see if the difference in probabilities changes the decoding output.

[26]`https://github.com/kaldi-asr/kaldi/blob/master/src/nnet3bin/nnet3-compute.cc`

[27]`https://github.com/kaldi-asr/kaldi/blob/master/src/bin/copy-transition-model.cc`

8.1.2 Preliminary decoding experiment

We conducted a small decoding experiment in order to understand how clean the output of the Viterbi decoder is and if we can improve the lexicon using the decodings of the Viterbi algorithm. As a test word, we chose the word *minute* because it has the high number of pronunciations in the dictionary - three and has quite many utterances in the corpus in general (it constitutes 0.031 of the corpus size in word utterances). We picked these criteria since multi-pronouncing words pose a problem for ASR and we aim at improving exactly for this kind of words in the lexicon.

The results of the decoding for the word *minute* are presented in table 3 where the first column is the id of a particular decoded sequence, the middle column represents the unique decoded sequence obtained for the word *minute* and the final column shows the number of occurrences for a particular decoded sequence e.g. how many times this decoding occurred for the word *minute* based on the decodings of many word utterances.

ID	Decoded sequence	# word utts
1	M IH N AH T	35
2	AH M IH N AH T	9
3	M IH N AH N D	5
4	M IH N IH T	5
5	AH M IH N AH N T	3
6	M IH N	3
7	M IH N AH N	3
8	M AY N UW	3
9	AH M IH N AH D	2
10	W AH N	2
11	AH M IH N AH	2
12	M IH N AH	2
13	M IH N AH N T	2
14	AH M IH N IH T	1
15	W AH M IH N AH	1
16	AH M IH T AH	1
17	M IH N IY	1
18	AH N AH N	1
19	IY M IH N IH JH	1
20	M IH N AH D	1
21	AH L IH N IH T	1
22	M IY	1
23	W AH N AY N UW IH T	1
24	AH M IH	1
25	M AH D	1
26	AH M IH N	1

27	W IH N IH	1
28	M AY N UW IH T	1
29	AH M IH N IY IH T	1
30	W AH N IH D	1
31	M IY AH N IY	1
32	OW M IH N AH	1
33	AH M AH T	1
34	M IH N IH T S	1
35	DH AH M IH N IH T AH T	1
36	W AH M IH N AH T	1
37	M AH T	1

Table 3: Different decoded pronunciation sequences obtained by the Viterbi decoding algorithm for the word *minute*.

In the hand-crafted pronunciation dictionary, linguists put three different pronunciations for the word *minute*: *M IH N AH T*, *M AY N UW T* and *M AY N Y UW T*. We actually witness one of these pronunciations after decoding with the Viterbi algorithm in table 3 (highlighted with the dark grey colour) and it is the most frequent decoded sequence of phonemes. The other two pronunciations are not present in our decodings, however there is a variations of the second lexicon entry (the sequence under ID 8 in table 3).

As we can see from table 3, there are some variants in the decoded sequences which we do not want to put into the dictionary. However, we can see some promising pronunciations that we might want to include in the pronunciation dictionary (highlighted by the light grey colour), for example, the decoded pronunciation under ID 4. There are also some interesting examples that could be valuable and beneficial to include in the lexicon since they actually exist in the language. For instance, *D* ending in several decoded sequences instead of *T* or *AH* appearing in the beginning. These could be the reflection of some phonological processes such as voicing/de-voicing, epenthesis or final consonant deletion. If they happen in the real world, we could benefit by including them in the pronunciation dictionary.

Nonetheless, it is obvious that a special strategy is necessary here in order to choose the right pronunciations to be included in the new dictionary. We see some potential in using the Viterbi decoder to extract possible pronunciations but we also need a mechanism to filter out some noisy pronunciations and not to expand the dictionary too much. Thus, we believe that the system proposed in section 4.1 can help to generate clean and meaningful pronunciation entries for the dictionary.

8.2 Benchmark lexicon

As a benchmark lexicon, we chose the pronunciation dictionary introduced by CMU[28]. The CMU dictionary is an open-source pronunciation dictionary for North American English containing over 134 000 words and pronunciations for them. The dictionary represents words and their pronunciations expressed with the ARPAbet phoneme set[29] developed for ASR tasks. The current phoneme set has 39 phonemes.

The CMU dictionary is still actively maintained and expanded. This pronunciation dictionary is one of the most actively used for speech recognition and synthesis purposes for the English language. Particularly this lexicon is used in the standard Kaldi recipe for the VoxForge corpus. In our version of the CMU dictionary, there are 123 699 different words equalling to 132 982 pronunciations in total.

The CMU dictionary is a hand-crafted dictionary maintained by linguists which is the common practice in lexicon creation for ASR tasks. Hand-crafted dictionaries are usually considered to be quite good for training of ASR models. That is why we consider using the CMU dictionary as a benchmark for our experiment.

8.3 Baseline lexicons

For the generation of baseline lexicons, we use the Viterbi algorithm for decoding of the whole training set of the VoxForge corpus.

The first baseline approach for lexicon generation considers the most frequently decoded pronunciation sequence to be the most reliable for ASR training. Thus, in this approach, we generate the lexicon in a one-word-one-pronunciation manner. The word pronunciation to be written in the lexicon is chosen based on its absolute frequency among all sequences for the word decoded by the Viterbi algorithm. We will refer to the lexicon generated in this manner as **1_1**.

The second baseline approach uses a similar strategy for choosing pronunciations but, instead of writing only one pronunciation for a word into the dictionary, we write top three pronunciations based on their absolute frequency after the decoding. If there are less than three distinct decodings for a word, all of them are taken as word pronunciations to put into the lexicon. The choice of writing top three pronunciations is motivated by the fact that we want to write more than one pronunciation into the dictionary since we know that many words have more than one pronunciation in a language. However, we do not want to overload the dictionary with too many pronunciations. This approach for dictionary generation will further be referred as **1_3**.

These two approaches are motivated by the fact that if we can decode one word similarly several times with Viterbi decoding, we expect them to sound alike and there are more chances for the ASR system to decode them this way as well. Thus, it would be helpful to include this decoded entry into the lexicon.

[28]http://www.speech.cs.cmu.edu/cgi-bin/cmudict
[29]https://en.wikipedia.org/wiki/ARPABET

The third baseline approach for pronunciation generation includes writing all possible decoded pronunciation sequences into the pronunciation dictionary and, thus, having many same word entries in the lexicon. This approach is motivated by our interest in determining how this strategy of pronunciation generation influences the overall performance of the ASR system (if we improve or worsen the WER score) and the time required for the training of ASR system. This lexicon is denoted as **1_all** on-wards.

The baseline lexicons differ in a strategy of handling the decoded words, however, all baseline lexicons share a simple voting strategy for choosing word pronunciations. In table 5, for recapitulating purposes, all baseline lexicons are presented with short descriptions.

8.4 Data-driven lexicons

The approach for data-driven lexicon generation is based on our main research on pronunciation clustering considering the acoustic information. This approach uses MFCC acoustic features for word utterances and, with the help of the AWE extraction system, obtains fixed-dimensional AWEs for each word utterance. These AWEs are used to cluster pronunciations for each word in the training set into some pronunciation groups. The whole pipeline of the proposed system is discussed in section 4.1.

For all data-driven approaches, we explore the same strategy of taking only one the most frequent pronunciation decoded by the Viterbi algorithm as in **1_1**. However, the most frequent decoded pronunciation is taken not for a word but for each detected pronunciation cluster for a particular word. Thus, in the lexicon for each word, the number of written pronunciations corresponds to at least the number of detected clusters or more if we have several pronunciation variants with a maximum frequency in a cluster. The motivation for this strategy is the same as for **1_1** and **1_3** lexicon generation approaches. However, in this case, if clustering can detect different pronunciations of some word, we will be able to include all of them whereas in the case of the **1_1** and **1_3** lexicons, there is a high chance to miss some existent pronunciation because it is not frequent in the data.

The difference between various data-driven lexicons is due to the variations in the clustering procedure and the AWE space training. Clusters for the lexicon denoted as **clust** were generated by simple GMM clustering with five fixed components on features extracted by AWE word differentiation model. The lexicon referred as **clust_bic** is generated with clustering done by the GMM with using of the BIC as a stopping criterion for choosing the number of clusters for each word in the training data. The AWE training is the same as for **clust**. The motivation and description of this strategy were discussed in chapter 7. The last two data-driven lexicons denoted as **clust_bic_unnorm_ed** and **clust_bic_norm_ed** use the same strategy for clustering as the **clust_bic** lexicon but have different sampling procedure in the AWE training. According to its sampling procedure, the AWE space has extended pronunciation training with edit-distance utilization (extension training is discussed in section 6.4). The difference between **clust_bic_unnorm_ed** and **clust_bic_norm_ed** lexicons is in the way edit-distance is calculated for AWE training (unnormalized vs. normalized, see section 6.4 for more details).

All data-driven lexicons are present in table 5 in the form of a small overview.

8.5 Additional lexicons for comparison

The three lexicons introduced in the previous section are used for the comparison of the data-driven approach for lexicon generation with the baseline lexicons and one benchmark lexicon represented by the hand-crafted CMU dictionary. However, we conducted several more experiments on pronunciation generation techniques in order to compare their performance with the data-driven dictionaries and explored other strategies that could improve lexicon generation.

We decided to generate the fully G2P (more on G2P can be found in chapter 3) lexicon in order to compare its results with the results of introduced data-driven lexicons. This comparison is interesting because one of the possible applications for the proposed system is to generate OOV words for the ASR task. G2P systems are mainly used to handle the problem of OOV words in ASR. We would like to prove if the proposed data-driven lexicons can outperform the standard technique for OOV handling.

For lexicon generation, the pre-trained Sequitur model was used (Bisani and Ney, 2008). Then, the internal Kaldi script[30] was used to obtain G2P conversions and get the lexicon. We generated the lexicon only for in-vocabulary words in order to have comparable results with other dictionaries. In this lexicon, the number of total word pronunciations is equal to the number of words in the lexicon. From now on, this lexicon is denoted as **g2p**.

For research purposes in the area of pronunciation generation, we also include several more lexicons for the comparison among baseline systems. We would like to empirically find out what number of top pronunciations has to be included in the lexicon. These new lexicons are also based on the voting strategy and differ from already introduced baseline lexicons in a number of pronunciations written for a word. The **1_2** and **1_4** lexicons represent lexicons with top two and top four decoded pronunciations for a word, respectively. These lexicons are introduced as a source of comparison for the **1_3** lexicon in order to figure out what number of top written pronunciations is the most appropriate to be used in a lexicon.

All introduced additional lexicons can be found in the overview table 5.

The last method for lexicon generation that we would like to investigate involves the probability assignment to the pronunciations in the lexicon. Exploring different probability assignments in Kaldi can show us how pronunciation probabilities influence ASR results and which strategy for probability assignment is the most suitable for data-driven lexicons.

For lexicon probability investigation, we use the best performing data-driven lexicon. The performance of lexicons is presented in figure 7 in chapter 9. According to the obtained results, the best performing data-driven lexicon turns out to be **clust_bic_unnorm_ed**.

We aim to explore the influence of introducing non-equal pronunciation probabilities. The Kaldi toolkit by default makes all pronunciation probabilities in the lexicon equal unless the probabilities are explicitly specified. In this case, all the pronunciation variants have the probability equal to 1.0, meaning that it has zero cost in the Kaldi graph. We would like to use the knowledge that all pronunciation variants are not equally distributed.

[30]`https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/local/g2p.sh`

Since we have the output of the Viterbi decoding, we can use the frequency information of these decodings to calculate the probability of each pronunciation variant for a word.

We explored several ways of probability assignment for the Kaldi lexicon. The first approach is just to assign probabilities accordingly to pronunciation frequencies out of Viterbi decoding. In this approach, we use the main notion of the probability. For the second approach, we set the most frequent pronunciation variant to be equal to 1.0 and the rest variants‘ probabilities are proportionally recalculated.

8.6 General lexicon information

The main goal of this research is to compare all these data-driven pronunciation generation approaches to each other and to the actual hand-crafted dictionary that is used to train the ASR system for the VoxForge corpus in the Kaldi toolkit. As a comparison measure, we chose the WER score of the ASR system trained with each lexicon.

For these dictionaries to be comparable, we use the initial hand-crafted dictionary as a default option and we save the lexicon size for each generated lexicon with regard to this default dictionary. Thus, for each generated pronunciation dictionary, we operate only with words that are present in the training set of the VoxForge corpus. The words that are not present in the training set and, as a consequence, for which no word entries were generated in these dictionaries were borrowed from the default dictionary. Hence, the comparison of these dictionaries considers only the difference for some set of words but this helps to compare only the results caused by the change in lexicon generation procedure and not caused by other factors such as the number of OOV words for a particular dictionary.

It is important to mention that we do not handle OOV words anyhow. That means that we avoid using the G2P system for generating of OOV words for the lexicon as it is usually used in ASR systems. The avoidance of using the G2P system does not play any particular role for our research and using or not using the G2P system for OOV words does not influence the comparison of the dictionaries.

	# words	# prons	avg # prons
1_1		123 699	1.0
1_2		130 008	1.051
1_3		135 646	1.097
1_4		140 303	1.134
1_all	123 699	205 188	1.659
clust		133 945	1.083
clust_bic		132 597	1.072
cmu_bic_{unnorm,norm}_ed		130 229	1.053
g2p		123 699	1.0
cmu		132 982	1.075

Table 4: The number of words, the number of pronunciations and the average number of pronunciations for each of the lexicons.

In table 4, we provide the main statistics of the considered lexicons. For each lexicon (the first column of the table), in the second column we cite the number of words, in the third column the number of different pronunciations and in the fourth column the average number of pronunciations per word in the lexicon.

Lexicon	Way of creation	Basis for pron including	# of prons per word
1_1	majority voting for decodings	the most frequent among all prons	1
1_2	majority voting or decodings	the most frequent among all prons	1-2
1_3	majority voting for decodings	the most frequent among all prons	1-3
1_4	majority voting for decodings	the most frequent among all prons	1-4
1_all	majority voting for decodings	the most frequent among all prons	all available
clust	pronunciation clustering (GMM + AWE1)	the most frequent in a cluster	1-5
clust_bic	pronunciation clustering (GMM + BIC + AWE1)	the most frequent in a cluster	automatically derived
clust_bic_ unnorm_ed	pronunciation clustering (GMM + BIC + AWE2_un)	the most frequent in a cluster	automatically derived
clust_bic_ norm_ed	pronunciation clustering (GMM + BIC + AWE2_n)	the most frequent in a cluster	automatically derived
g2p	G2P conversion	motivated by grapheme correspondence	1
cmu	hand crafting	linguistically motivated	1-3

Table 5: The overview of all introduced lexicons for the experiment. AWE1 denotes feature extraction for pronunciation clustering that aims to differentiate words. AWE2 refers to feature extraction for pronunciation clustering that aims to differentiate pronunciations. AWE2 with _n suffix means the AWE training with normalized edit distance. AWE2 with _n suffix signifies the AWE training with unnormalized edit distance.

9 Results

To avoid linguistically motivated pronunciations, we defined a lexicon generated directly from data with the help of pronunciation clustering. We proposed several methods of feature training for clustering analysis with the GMM algorithm that we would like to evaluate in terms of ASR performance. For the evaluation of the generated data-driven lexicons, we came up with several baseline and benchmark lexicons for comparison. The evaluation of different lexicons is considered to be done through the training of ASR systems with the use of each of the determined lexicons and comparing their performance in terms of WER score.

In this chapter, we present the comparative results obtained for the whole pipeline of data-driven pronunciation generation based on pronunciation clustering. In section 9.1, we introduce the architecture and features of the ASR system that is used for training with different lexicons. Section 9.2 is devoted to the ASR results obtained for the different considered lexicons and to the exploration of the performance of these lexicons. Finally, in section 9.3, we discuss the results and make some conclusive remarks.

9.1 ASR training

To compare the quality of lexicons (listed in table 5), we train ASR systems using the generated lexicons and evaluate the performance of ASR systems in terms of WER score.

The ASR model was trained using the Kaldi speech recognition toolkit for the VoxForge English corpus (more information about the corpus can be found in chapter 5). We use our own extension of the existent recipe for the VoxForge corpus in Kaldi[31].

We use data of all available dialects for English in the VoxForge corpus (approximately 10 different dialects). The data was normalized, the language model was prepared with the help of SRILM[32]. The lexicon and phone lists varied dependently on which of the considered lexicons we test. OOV words were not handled in our experimental setup. The MFCC features were extracted with the internal Kaldi tool.

The data has been split into train and test sets. In the training set, utterances of 2 870 speakers were used. In the test set, utterances of 20 speakers were used. The number of utterances in the training set is 76 551.

Then, we train our model in a consecutive manner. Firstly, we train monophone model[33] on the 1 000 subset of the data. Then, we align the data using this model and train the triphone model on delta + delta-delta features[34]. Then, we again align all the data with the previously trained model and train the acoustic model with LDA + MLLT feature transforms[35]. After that, we align our data with the trained LDA + MLLT model and

[31]https://github.com/kaldi-asr/kaldi/tree/master/egs/voxforge
[32]http://www.speech.sri.com/projects/srilm/
[33]https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_mono.sh
[34]https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_deltas.sh
[35]https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_lda_mllt.sh

train the acoustic model on LDA + MLLT features with applying SAT[36]. Finally, we align the data with LDA + MLLT + SAT model and train deep neural network ASR model with the `nnet3` Kaldi module[37].

Further, we decode the test set with the internal Kaldi decoding[38] and measure WER and SER scores[39]. We discuss the obtained lexicon results in the next section.

9.2 ASR results

In table 6, we provide results for the ASR systems trained on different lexicons discussed in chapter 8. The lexicons are referred to by the labels assigned in chapter 8. We provide both WER and SER scores and mark the best WER and SER scores with bold script.

	1_1	1_3	1_all	clust	g2p	cmu
WER	15.97	14.66	35.73	19.99	9.78	**9.47**
SER	61.29	57.8	83.87	72.58	44.09	**42.74**

Table 6: The WER and SER scores for ASR systems trained with baseline **1_1**, **1_3**, **1_all**, data-driven **clust** and benchmark **cmu** lexicons.

As we can see from the table, the results for hand-crafted CMU dictionary (**cmu**) are the best. Writing down in the dictionary only the most frequent pronunciation (**1_1**) showed to be less successful in terms of WER than considering the top three frequent pronunciations for a word (**1_3**). We can see more than one WER score point difference between these two dictionaries. The result for the dictionary with all possible pronunciations (**1_all**) for a word is the worst among all. We expected this dictionary to perform quite badly since the more pronunciation alternatives we have, the more the ASR system is confused by them and the higher chance to get the wrong decoded sequence.

	clust	clust_bic	clust_bic_unnorm_ed	clust_bic_norm_ed
WER	19.99	16.17	**14.63**	14.69
SER	72.58	62.37	**58.6**	58.6

Table 7: The WER and SER scores for ASR systems trained with data-driven **clust**, **clust_bic**, **clust_bic_unnorm_ed** and **clust_bic_norm_ed** lexicons.

The results for the clustering experiment (**clust**) showed surprisingly bad WER results being almost twice worse than the hand-crafted dictionary. This lexicon could not even compete with fully G2P lexicon (**g2p**) which performs slightly worse than hand-crafted dictionary. In table 7, we present more experiments on pronunciation clustering. We can

[36]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_sat.sh`
[37]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/nnet3/chain/train.py`
[38]`https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/decode.sh`
[39]`https://github.com/kaldi-asr/kaldi/blob/master/egs/voxforge/s5/local/score.sh`

improve the results by introducing not five predefined clusters but the stopping criterion for choosing the number of clusters (**clust_bic**). Moreover, the proposed AWE training extension (**clust_bic_{unnorm, norm}_ed**) improves results even more. In the table, we present the results for two types of edit distance calculation (unnormalized and normalized). We can see that the difference between unnormalized edit distance **clust_bic_unnorm_ed** lexicon and normalized edit distance **clust_bic_norm_ed** lexicon is negligible. However, the WER score for the best-performing clustering method **clust_bic_unnorm_ed** is still worse than for the hand-crafted dictionary for more than five points.

	1_1	1_2	1_3	1_4	1_all
WER	15.97	16.62	**14.66**	16.45	35.73
SER	61.29	62.37	**57.8**	60.75	83.87

Table 8: The WER and SER scores for ASR systems trained with baseline **1_1**, **1_2**, **1_3**, **1_4**, **1_all** lexicons.

Table 8 shows the results of the experiment on finding the best possible number of lexicon entries per word. As we can see from the table, the lexicon with three pronunciations per word outperforms the rest of the considered lexicons with only one, two, four and all pronunciations. The number three seems to be very reasonable for the number of pronunciations to be included in the dictionary. For some words, we definitely have more than one pronunciation but including too many of them confuses the ASR system. From these results, we can suggest that an average number of four pronunciation entries in the dictionary already surpasses the threshold of a proper number of pronunciations per word in the lexicon.

In the following table, the comparative results of experiments on applying different pronunciation probability strategies in a lexicon are presented (pronunciation probabilities were discussed in chapter 8).

Pronunciation probability	Lexicon	WER
no prob	clust_bic_unnorm_ed	14.63
	clust_bic_norm_ed	14.69
prob	clust_bic_unnorm_ed	**11.17**
	clust_bic_norm_ed	**11.2**
rescaled prob	clust_bic_unnorm_ed	13.56
	clust_bic_norm_ed	13.35

Table 9: The WER comparison for ASR systems trained with two data-driven **clust-_bic_unnorm_ed** and **clust_bic_norm_ed** lexicons with using different pronunciation probability assignment strategies.

From the table, we can see that results for both **clust_bic_unnorm_ed** and **clust_-bic_norm_ed** lexicons are quite consistent in terms of behaviour when probabilities are applied. This experiment shows that writing no probabilities (”no prob” in the table) for word pronunciations (having all pronunciation probabilities equal to 1.0) results in worse performance for the ASR system in comparison to introducing such probabilities. Applying the standard notion of probabilities to pronunciation variants when probabilities of all pronunciation variants sum up to one (”prob” in the table) gives us the improvement of more than three percent for both lexicons in comparison to not using any probabilities. Rescaled probabilities (”rescaled prob” in the table) with the most probable word pronunciation having probability 1.0 also improves the result but not as much as normal probabilities. This result can mean that by lowering down pronunciation probabilities we give more power and freedom for the language model to influence the decoding by its probabilities that can predict candidates based on language model probabilities extracted from data when an acoustic model is unsure.

In figure 10, we gather the most successful lexicons of our experiments. Even though we surpassed the set baseline (**1_3** in figure 10), we could not reach the benchmark level (**cmu** lexicon).

	1_3	clust_bic_unnorm_ed	g2p	cmu
WER	14.66	11.17	9.78	**9.47**
SER	57.8	45.97	44.09	**42.74**

Table 10: The WER and SER scores for the ASR systems trained with best performing lexicons from baseline, data-driven and benchmark lexicons.

	1_3	clust_bic_unnorm_ed	g2p	cmu
Total words	517	516	345	334
Unique words	277	281	223	216
Percentage	53.6	54.5	64.6	64.7

Table 11: The total number of incorrectly decoded words, the number of incorrectly decoded unique words and the percentage of unique words from the total number of words for **1_3**, **clust_bic_unnorm_ed**, **g2p** and **cmu** lexicons.

In table 11, we calculated what percent of total words that were got wrongly are unique words for the lexicons presented in table 10. From this table, we can conclude that the difference in the percentage of unique words that comprise the incorrectly got words by the ASR system for our clustered lexicon and for the hand-crafted CMU dictionary is big. That means that our **clust_bic_unnorm_ed** dictionary mostly makes mistakes for the same words whereas the CMU dictionary has bigger variety of words that it misdecodes. This could mean that the clustered dictionary has low-quality pronunciations for quite frequent words in the test data so that in most cases if not all it cannot properly decode them. The reasoning for the percentage difference for the hand-crafted dictionary and **clust_bic_unnorm_ed** can be applied also to difference in **1_3** and **cmu** dictionaries.

In table 12, we provide two numbers that represent how many words the ASR system trained on the original hand-crafted dictionary misdecoded while the ASR system trained on the generated lexicons decoded correctly and vice versa. In this table, we also cite a similar comparison for the G2P system.

	1_3	clust_bic_unnorm_ed
CMU wrong, gen_dict right	14	10
CMU right, gen_dict wrong	98	100
Percentage	14.3	10
	1_3	clust_bic_unnorm_ed
g2p wrong, gen_dict right	18	14
g2p right, gen_dict wrong	101	103
Percentage	17.8	13.6

Table 12: Comparison of **1_3** and **clust_bic_unnorm_ed** (gen_dict) with the CMU and G2P dictionaries in terms of correctly decoded words.

From the table 12, we clearly see that the CMU dictionary outperforms significantly other dictionaries in the number of words these dictionaries could not decode correctly while the CMU dictionary could. However, the table 12 also shows that the number of words that the CMU dictionary misses whereas a custom dictionary got it right for all generated dictionaries is very small. This means that we could not even benefit from the merged system of two dictionaries, the hand-crafted and a generated one since the CMU dictionary performs quite successfully for the VoxForge corpus. Similar reasoning can be applied for the comparison with the G2P lexicon.

9.3 Discussion

The results presented in the previous section show us that a hand-crafted dictionary still stays the best option for the ASR training and we could not improve further on the CMU result. There exists a big gap in performance between the proposed system based on clustering analysis and the hand-crafted dictionary of more than 1.5 points. From this result, we can conclude that our automatically generated dictionary lacks the descriptive power for different pronunciations and the hand-crafted pronunciation dictionary cannot be successfully replaced by the generated lexicon. If there is a need for the data-driven lexicon generation, choosing three top pronunciations of a word and including them into a pronunciation dictionary can be a comparatively good alternative, with reasonable low WER score and computationally cheap.

The comparison with the lexicon generated by the G2P system also suggests that the data-driven lexicon based on pronunciation clustering cannot compete with the widely used system for OOV words generation. Thus, the proposed data-driven lexicon generation system cannot be safely used for OOV words handling and the G2P model is still preferred for such cases.

Our main experiment on data-driven pronunciation generation that involves clustering of pronunciations and writing pronunciations into the dictionary based on the detected clusters showed unexpected bad results which could mean that either this approach is not appropriate for the task of data-driven pronunciation generation or pronunciations generated directly from data are less suitable for ASR than linguistically motivated ones or it needs significant improvements. We can name several reasons why the proposed system failed to achieve good results in the ASR task.

The main reason for our approach performing worse than expected can be in the quality of the features used for pronunciation clustering, namely AWEs. For using machine learning methods, we extracted fixed-dimensional AWEs that are normally used for placing particular word utterances in the embedding space. The main aim of AWEs is to differentiate between different words. We proposed an extension of AWEs training in order to enable the AWE model to differentiate between different pronunciations for a word. However, the ability of the proposed training extension to distinguish between different pronunciations is questionable. The discriminative power of AWEs can be not satisfactory for a cluster algorithm to capture different word pronunciations which results in overall bad performance of the approach. The unsatisfactory AWEs quality can be due to high reliance on the correctness of the Viterbi decoding during AWE model training. The Viterbi decoding can produce very noisy outputs which are hard to assess and this can result in the worse resulting quality of acoustic embeddings since the noise confuses the model. Insufficient amount of time for training also can be a justification for the inappropriate quality of fixed-dimensional acoustic features.

Another reason for the proposed pronunciation generation approach to be still outperformed by the hand-crafted dictionary is clustering itself. In the case of pronunciation clustering of automatically generated AWEs, the algorithm must be very robust to noise and be able to model quite complex data topology. It can be the case that none of the explored unsupervised machine learning algorithms can cope with the task of pronunciation clustering. This can be due to artificially created features instead of more reliable features like MFCC extracted directly from an audio signal.

Finally, we did not devote enough time for fine-tuning of each step of our pronunciation generation system. The problem of insufficient fine-tuning could also be one of the most influential reasons for the system to underperform. Neural networks are really sensitive to hyperparameter tuning and the proper construction of its architecture for the task of AWE training. Clustering algorithms also require tuning of hyperparameters.

Based on these considerations, we tend to think that improvement at every stage of the proposed data-driven pronunciation generation could finally lead to the better performance of the generated dictionary for ASR and potentially outperform the hand-crafted dictionary.

In the end, for this research, we wanted to focus on the English language and the CMU hand-crafted dictionary that exists for this language to have a strong benchmark lexicon for comparison. Unfortunately, we could not outperform such a strong benchmark of this research since the hand-crafted lexicon developed for the widely and frequently used language must be crafted in a qualitative way. Probably, the evaluation for other languages trained with commonly used lexicons for these languages could lead to the better performance of the proposed pronunciation generation system.

10 Conclusion

This research was devoted to the data-driven pronunciation generation for ASR purposes. We developed a pronunciation generation system that completely relies on acoustic information derived from data for pronunciations candidates. This approach for lexicon generation is absolutely novel and was not researched previously. For the task of data-driven pronunciation generation, we introduced a method that uses clustering of word utterances in order to identify pronunciation variants for a word. We conducted several experiments on feature extraction (AWEs) for the clustering task and pronunciation clustering experiments with various clustering algorithms. In the end, we evaluated the lexicons generated by the proposed data-driven pronunciation generation system and compared their quality with several generated baseline lexicons and a very strong CMU hand-crafted lexicon.

In our experiments, we discovered that edit distance criterion can serve for improvement of the model that aims to differentiate between different pronunciations for a word. If we use edit distance for utterances decoded with the Viterbi algorithm to sample the data for the Siamese RNN setup in order to train AWEs, we can improve WER by more than 1.5 points. The extensive clustering experiments showed that the most appropriate clustering algorithm for pronunciation clustering with AWEs as feature vectors was the GMM clustering algorithm with using the BIC as stopping criterion for detecting of the number of clusters for a word. Moreover, we determined that applying probabilities in a lexicon with the generated multiple pronunciation variants for a word improves for ASR performance. In this regard, the most successful strategy for distributing probabilities of pronunciations for a word is to rely on the absolute frequency of different decodings for a word and calculate the corresponding probabilities that sum up to one.

Our final evaluation showed that the lexicon generated by our pronunciation generation system (namely **clust_bic_unnorm_ed** version of the proposed method) can outperform all our baseline lexicons. However, it cannot reach the performance level of a benchmark system represented by the CMU dictionary. Among introduced baselines, our pronunciation generation system could slightly outperform a strong **1_3** baseline which considers the three top pronunciation sequences decoded by the Viterbi algorithm as pronunciation variants. Even though the CMU dictionary is a very well constructed and maintained hand-crafted lexicon and it is a very hard task to beat its performance, the ultimate goal of surpassing the performance of the hand-crafted dictionary was not met. We hypothesize that the proposed system still lacks the necessary level of pronunciation modelling in order to replace hand-crafted dictionaries for ASR. The proposed pronunciation generation system could neither succeed to beat the lexicon generated by the G2P approach, the main supervised approach for automatic lexicon generation. This suggests that the task of OOV words also cannot be solved successfully by the proposed system. We observed that our method was able to perform better on some individual cases and, in theory, can be used in combination with a G2P system. This remains as a promising future work.

We consider the unsatisfactory quality of generated AWEs and the insufficient fine-tuning of all components of pronunciation generation system to be the main reasons for the proposed pronunciation generation system to still perform worse than the hand-crafted dictionary. Even though we attempted to increase the discriminative power of the AWE model to distinguish between different word pronunciations, the extended AWE training seems to lack the power of discrimination between different pronunciations for a word which was shown by the visualization of the AWE space. Partially, we attribute the performance of AWEs to specific implementation details and we plan to experiment with other implementation in future work. In our work, we did not set the goal to find the best parameters and architectures for the exploited machine learning algorithms and wanted to see if the proposed system can at least cope with the task of pronunciation generation.

The main advantage of the proposed system for the pronunciation generation is that generated pronunciations are completely derived from data using statistical methods. We hypothesized that pronunciations modelled this way and not linguistically motivated as in hand-crafted pronunciation dictionaries would have a positive impact on ASR modelling. Since we still believe that there is a potential for this pronunciation generation system to improve over the standard hand-crafted dictionaries and G2P systems, we consider that some future work can be done in order to improve the proposed system.

First of all, an improvement of AWE quality should be gained. For achieving the satisfactory quality of pronunciation discrimination of AWEs, the multitask training for word and pronunciation discrimination can be considered. Now, AWEs are trained in a sequential manner, first for word discrimination and then for pronunciation discrimination. We suspect that pronunciation training may confuse the achieved word discrimination level. For dealing with this problem, the simultaneous training for word and pronunciation discrimination can be beneficial. Moreover, the proper fine-tuning of the whole system should be done since it can bring some major improvement to the existent data-driven pronunciation generation system.

References

Martine Adda-Decker and Lori Lamel. The use of lexica in automatic speech recognition. In *Lexicon Development for Speech and Language Processing*, pages 235–266. Springer, 2000.

Mohammed Alhanjouri. Curvelet and waveatom transforms based feature extraction for face detection. *Alaqsa University Journal*, 15, 01 2012.

Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

Thomas Bayes. An essay towards solving a problem in the doctrine of chances. *Phil. Trans. of the Royal Soc. of London*, 53:370–418, 1763.

Samy Bengio and Georg Heigold. Word embeddings for speech recognition. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008. URL `http://dblp.uni-trier.de/db/journals/speech/speech50.html#BisaniN08`.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

Guoguo Chen, Carolina Parada, and Tara N Sainath. Query-by-example keyword spotting using long short-term memory networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5236–5240. IEEE, 2015.

Scott Shaobing Chen and Ponani S Gopalakrishnan. Clustering via the bayesian information criterion with applications in speech recognition. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 2, pages 645–648. IEEE, 1998.

Stanley F Chen. Conditional and joint models for grapheme-to-phoneme conversion. In *Eighth European Conference on Speech Communication and Technology*, 2003.

Yu-An Chung, Chao-Chung Wu, Chia-Hao Shen, Hung-Yi Lee, and Lin-Shan Lee. Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder. *arXiv preprint arXiv:1603.00982*, 2016.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2011.

Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

Namrata Dave. Feature extraction methods lpc, plp and mfcc in speech recognition. *International journal for advance research in engineering and technology*, 1(6):1–4, 2013.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Anneleen Dereymaeker, Kirubin Pillay, Jan Vervisch, Sabine Van Huffel, Gunnar Naulaers, Katrien Jansen, and Maarten De Vos. An automated quiet sleep detection approach in preterm infants as a gateway to assess brain maturation. *International journal of neural systems*, 27(06):1750023, 2017.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Sadaoki Furui. Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34 (1):52–59, 1986.

Claudio Gentile and Manfred K Warmuth. Linear hinge loss and average margin. In *Advances in neural information processing systems*, pages 225–231, 1999.

Michael Gerber, Tobias Kaufmann, and Beat Pfister. Extended viterbi algorithm for optimized word hmms. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4932–4935. IEEE, 2011.

Nagendra Goel, Samuel Thomas, Mohit Agarwal, Pinar Akyazi, Lukáš Burget, Kai Feng, Arnab Ghoshal, Ondřej Glembek, Martin Karafiát, Daniel Povey, et al. Approaches to automatic lexicon learning with limited training examples. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5094–5097. IEEE, 2010.

William Hartmann, Anindya Roy, Lori Lamel, and Jean-Luc Gauvain. Acoustic unit discovery and pronunciation generation from a grapheme-based lexicon. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 380–385. IEEE, 2013.

Herman Kamper, Aren Jansen, Simon King, and Sharon Goldwater. Unsupervised lexical clustering of speech segments using fixed-dimensional acoustic embeddings. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 100–105. IEEE, 2014.

Herman Kamper, Weiran Wang, and Karen Livescu. Deep convolutional acoustic word embeddings using word-pair side information. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4950–4954. IEEE, 2016.

Arthur Kantor and Mark Hasegawa-Johnson. Hmm-based pronunciation dictionary generation. *New Tools and Methods for Very Large Scale Phonetics Research, University of Pennsylvania*, 2011.

S Karpagavalli and E Chandra. A review on automatic speech recognition architecture and approaches. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 9(4):393–404, 2016.

Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

Keith Levin, Aren Jansen, and Benjamin Van Durme. Segmental acoustic indexing for zero resource keyword search. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5828–5832. IEEE, 2015.

Longfei Li, Yong Zhao, Dongmei Jiang, Yanning Zhang, Fengna Wang, Isabel Gonzalez, Enescu Valentin, and Hichem Sahli. Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 312–317. IEEE, 2013.

Cheng-Yuan Liou, Jau-Chi Huang, and Wen-Chie Yang. Modeling word perception using the elman network. *Neurocomputing*, 71(16-18):3150–3157, 2008.

Karen Livescu. Acoustic word embeddings. URL `http://media.aau.dk/smc/wp-content/uploads/2017/09/LivescuTalk_ML4Audio.pdf`.

Andrew L Maas, Stephen D Miller, Tyler M O'neil, Andrew Y Ng, and Patrick Nguyen. Word-level acoustic modeling with convolutional vector regression. In *ICML Workshop on Representation Learning, Edinburgh, Scotland*, 2012.

Thomas Martinetz, Klaus Schulten, et al. A "neural-gas" network learns topologies. 1991.

Tofigh Naghibi, Sarah Hoffmann, and Beat Pfister. An efficient method to estimate pronunciation from multiple utterances. In *INTERSPEECH*, pages 1951–1955, 2013.

Maryam Najafian. *Acoustic model selection for recognition of regional accented speech.* PhD thesis, University of Birmingham, 2016.

Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach.* O'Reilly Media, Inc., 2017.

Alan B Poritz. Hidden markov models: A guided tour. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7–13, 1988.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*, pages 3743–3747, 2018.

V Radha and C Vimala. A review on speech recognition challenges and approaches. *World of Computer Science and Information Technology Journal (WCSIT)*, 2(1):1–7, 2012.

Kanishka Rao, Fuchun Peng, Haşim Sak, and Françoise Beaufays. Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4225–4229. IEEE, 2015.

Marzieh Razavi and Mathew Magimai Doss. An hmm-based formalism for automatic subword unit derivation and pronunciation generation. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4639–4643. IEEE, 2015.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.

Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

Shane Settle and Karen Livescu. Discriminative acoustic word embeddings: Recurrent neural network-based approaches. In *2016 IEEE Spoken Language Technology Workshop (SLT)*, pages 503–510. IEEE, 2016.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL `http://arxiv.org/abs/1808.03314`.

Dan Su, Xihong Wu, and Lei Xu. Gmm-hmm acoustic model training by a two level procedure with gaussian components determined by automatic model selection. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4890–4893. IEEE, 2010.

Torbjørn Svendsen. Pronunciation modeling for speech technology. In *Signal Processing and Communications, 2004. SPCOM'04. 2004 International Conference on*, pages 11–16. IEEE, 2004.

Pawel Swietojanski, Arnab Ghoshal, and Steve Renals. Hybrid acoustic models for distant and multichannel large vocabulary speech recognition. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 285–290. IEEE, 2013.

Naoya Takahashi, Tofigh Naghibi, and Beat Pfister. Automatic pronunciation generation by utilizing a semi-supervised deep neural networks. *arXiv preprint arXiv:1606.05007*, 2016.

Paul Taylor. Hidden markov models for grapheme to phoneme conversion. In *Ninth European Conference on Speech Communication and Technology*, 2005.

Shubham Toshniwal and Karen Livescu. Read, attend and pronounce: An attention-based approach for grapheme-to-phoneme conversion. In *Workshop on Machine Learning in Speech and Language Processing (MLSLP), Interspeech*, 2016.

Christophe Veaux, Junichi Yamagishi, Kirsten MacDonald, et al. Cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit. *University of Edinburgh. The Centre for Speech Technology Research (CSTR)*, 2017.

Stephen Voinea, Chiyuan Zhang, Georgios Evangelopoulos, Lorenzo Rosasco, and Tomaso Poggio. Word-level invariant representations from acoustic waveforms. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Shaofei Xue, Ossama Abdel-Hamid, Hui Jiang, and Lirong Dai. Direct adaptation of hybrid dnn/hmm model for fast speaker adaptation in lvcsr based on speaker code. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 6339–6343. IEEE, 2014.

Zhi-Jie Yan, Qiang Huo, and Jian Xu. A scalable approach to using dnn-derived features in gmm-hmm based acoustic modeling for lvcsr. In *Interspeech*, pages 104–108, 2013.

Geoffrey Zweig and Patrick Nguyen. A segmental crf approach to large vocabulary continuous speech recognition. In *ASRU*, volume 1, page 35, 2009.

A Attachments

A.1 Results of clustering algorithms

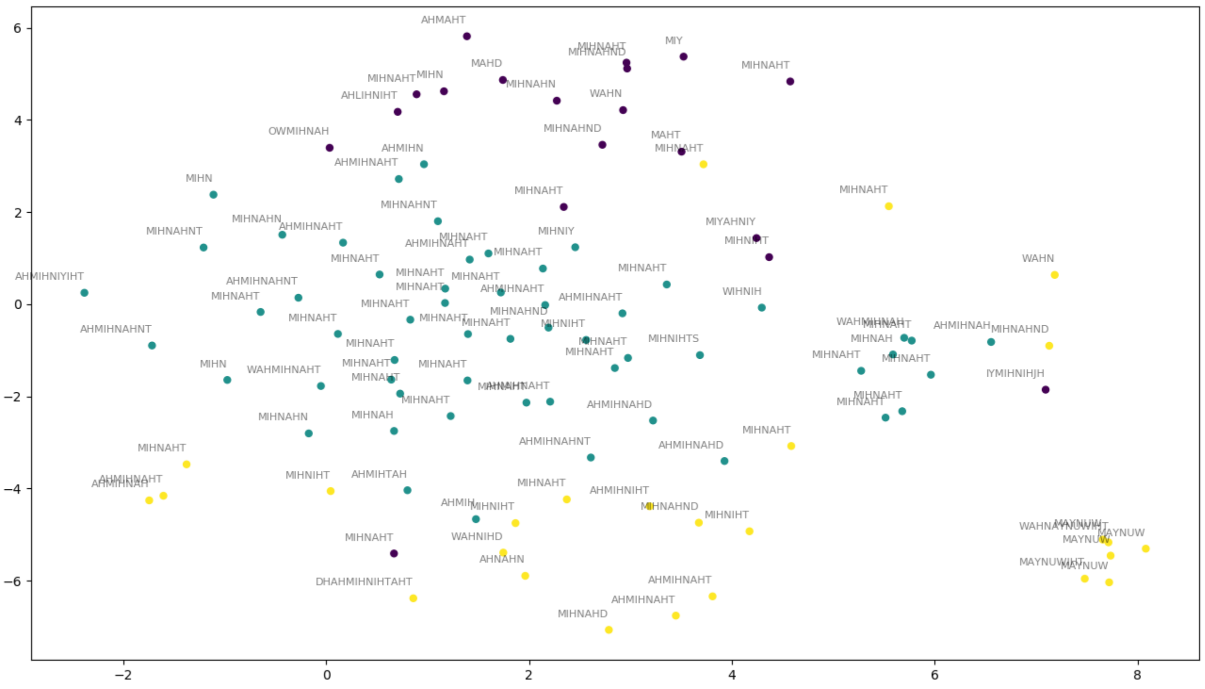

Figure 28: Clustering results for the word *minute* by the K-Means algorithm with predefined four clusters.

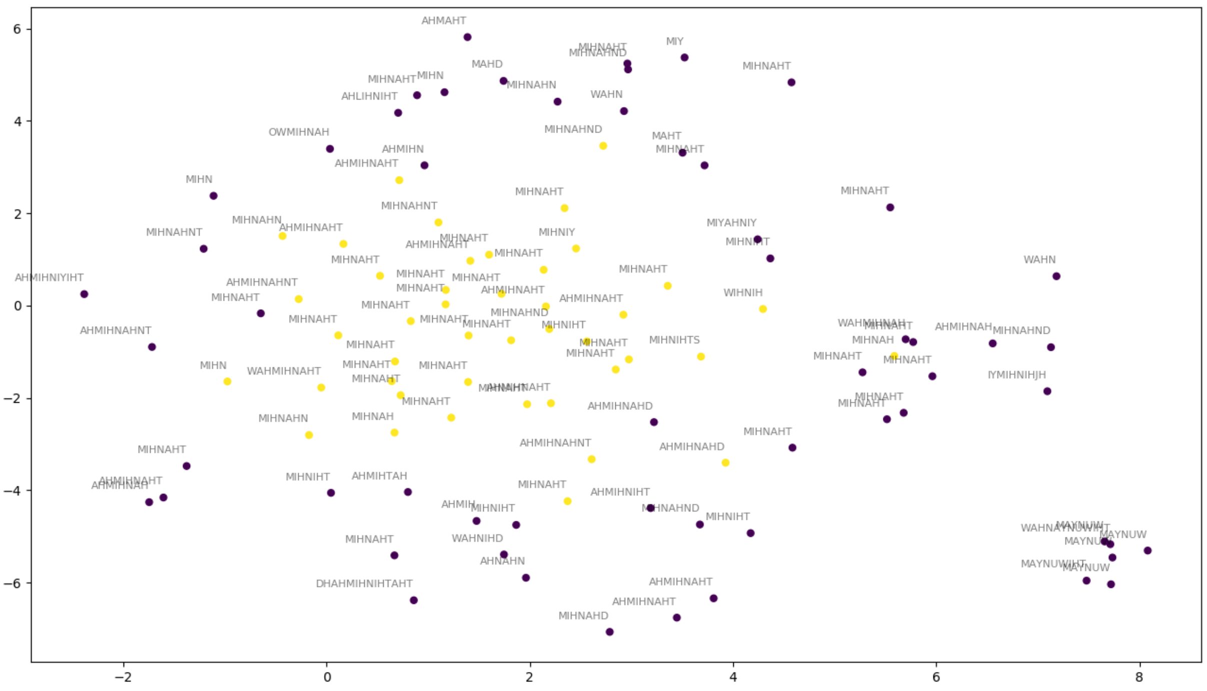

Figure 29: Clustering results for the word *minute* by the DBSCAN algorithm.

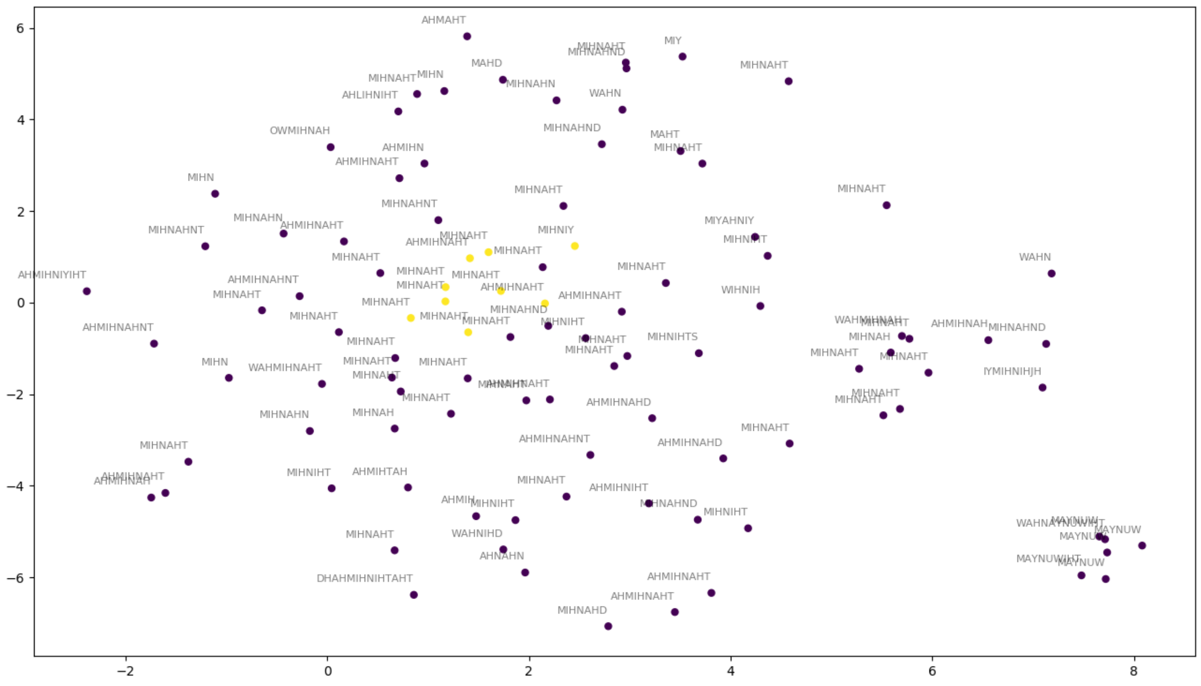

Figure 30: Clustering results for the word *minute* by the OPTICS algorithm.

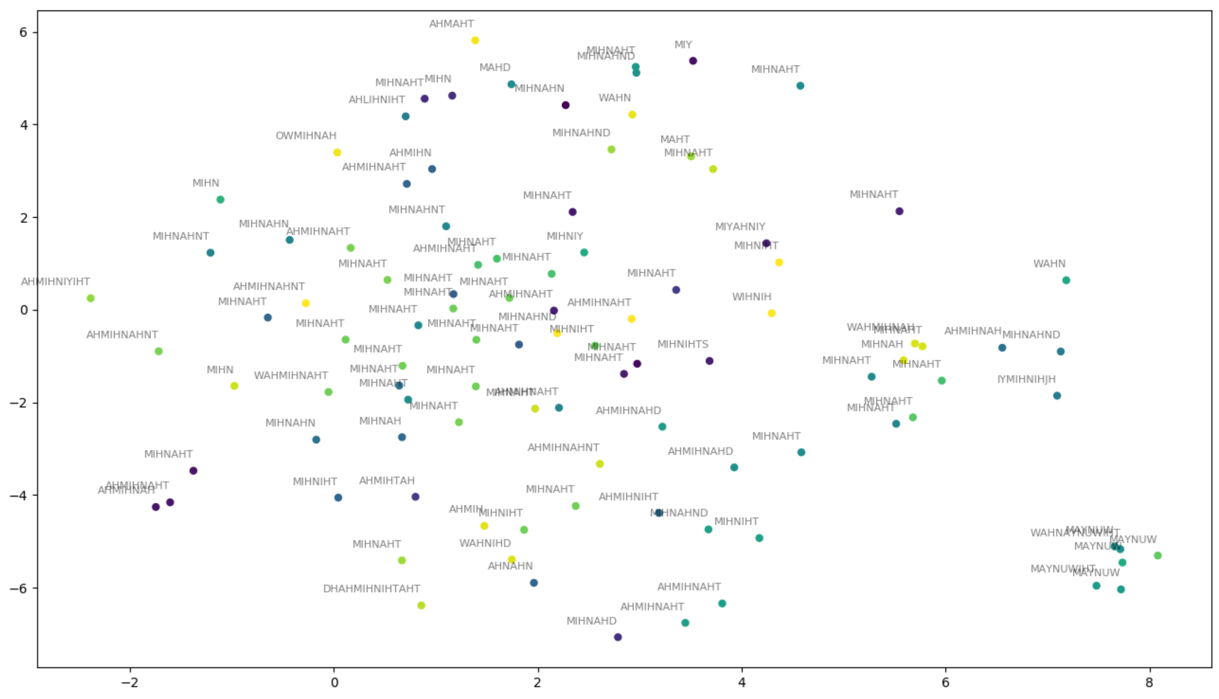
Figure 31: Clustering results for the word *minute* by the som algorithm.

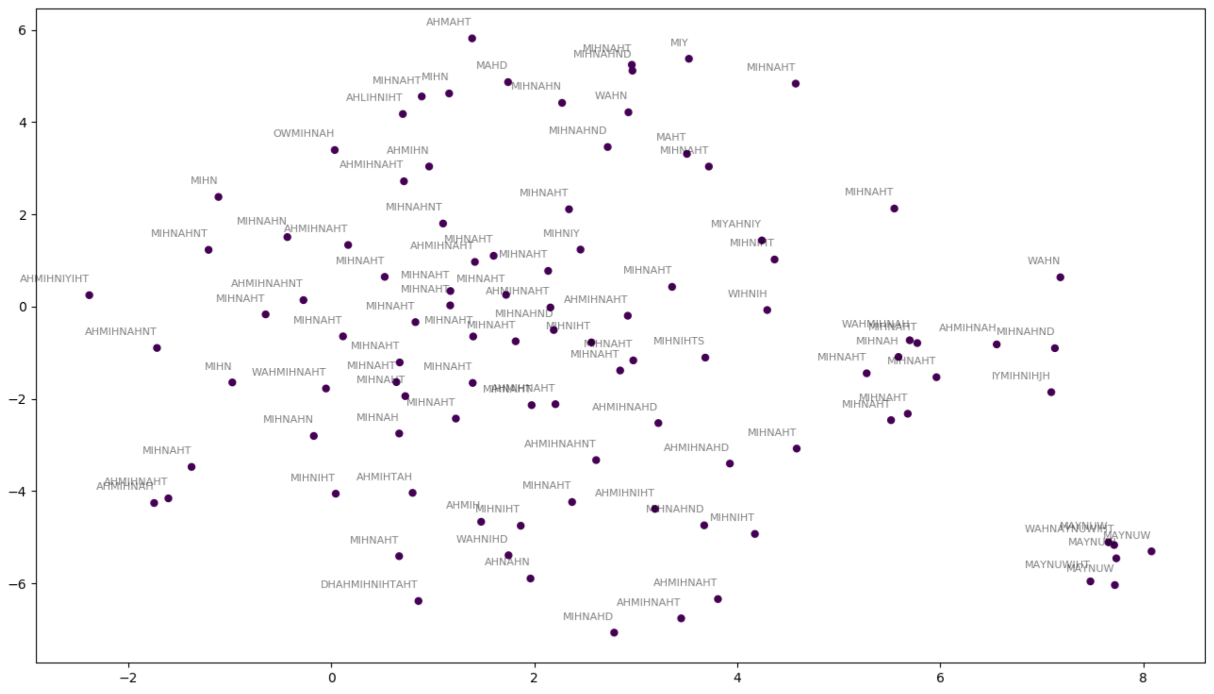
Figure 32: Clustering results for the word *minute* by the gng algorithm.

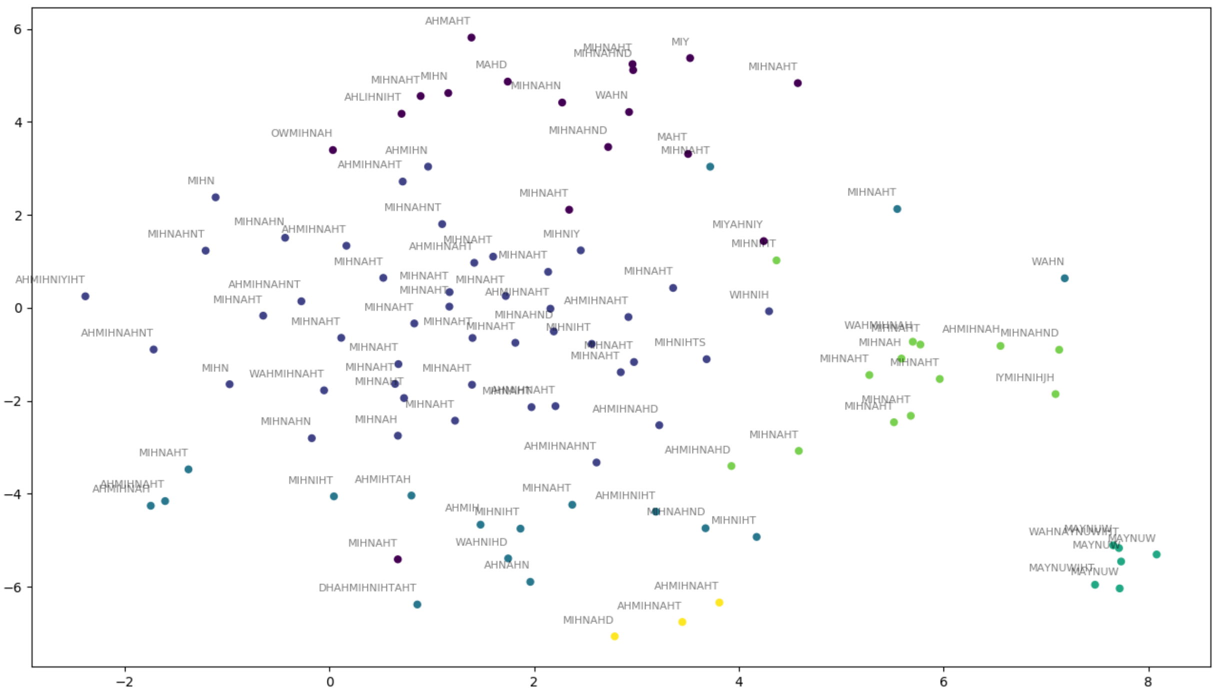

Figure 33: Clustering results for the word *minute* by the gmm algorithm.