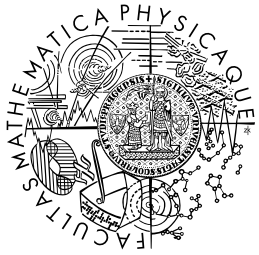

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Vojtěch Hudeček

Exploiting user's feedback to improve pronunciation of TTS systems

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague

Supervisor of the master thesis:	doc. Ing. Zdeněk Žabokrtský, Ph.D.
Study programme:	Informatics
Study branch:	Artificial Intelligence

Prague 2017

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Exploiting user's feedback to improve pronunciation of TTS systems

Author: Bc. Vojtěch Hudeček

Department: Name of the department

Supervisor: doc. Ing. Zdeněk Žabokrtský, Ph.D, Institute of Formal and Applied Linguistics

Abstract: Although spoken dialogue systems have greatly improved, they still cannot handle communications involving unknown topics and are very fragile. We will investigate methods that can improve spoken dialogue systems by correcting or even learn the pronunciation of unknown words. Thus we will provide better user experience, since for example mispronounced proper nouns are highly undesirable. Incorrect pronunciation is caused by imperfect phonetic representation, typically phonetic dictionary. We aim to detect incorrectly pronounced words by exploiting userâĂŹs feedback as well as using prior knowledge of the pronunciation and correct the transcriptions accordingly. Furthermore, the learned phonetic transcriptions can be used to improve speech recognition module by refining its models. Models used in speech recognition cannot handle words that are not in their vocabulary or have phonetic representation. Extracting those words from userâĂŹs utterances and adding them to the vocabulary should lead to a better overall performance.

Keywords: text-to-speech, automatic speech recognition, user's response, phonetic dictionary, machine learning, mel cepstral distortion

Dedication.

Contents

1. Introduction

Voice control or communication is a common feature of many systems nowadays. Its applications range from simple one-word control commands to complex communication in spoken dialogue systems. In this work, we consider mainly such complex systems. For the sake of clarity, we now briefly describe setting of such system.

It usually contains Automatic Speech Recognition (ASR) module, so the natural speech can be recognized and translated into words. The system then derives an appropriate response, typically in the form of sentence written in natural language. The process of this derivation depends on the concrete implementation. The response can be displayed in the textual form, however, it is more common to generate audio recording with human voice reading the response. Although it is possible to use a set of prerecorded utterances, this approach has obvious limitations since it is not able to read an arbitrary phrase. Particularly, it may be difficult to read named entities and numerical values such as time and date. Also, the usage of variable utterances provides better user experience.

To overcoma the mentioned issues, a Text-To-Speech (TTS) module is usually also part of dialogue systems. The purpose of this module is to transform a (generally arbitrary) written text utterance to natural speech. Modern TTS systems produce audio waveforms that sound quite naturaly and the pronunciation is sufficiently good. Nevertheless, it may experience some difficulties, mainly when it comes to unknown words. This may happen, because the system is usually trained using certain set of words, typically from one language. But real applications often require to pronounce named entities or other language- or domain- specific words, that cannot be present during the training phase. This cause situations, when the system has to employ some mechanism to derive the pronunciation and the words may be mispronounced. Succh words are called Out-Of-Vocabulary (OOV). The derivation of the pronunciation is inherently imperfect and may cause errors. Although this does not occur often, the negative effect can be quite strong, since it is inconvenient for the user, especially when his or her name is pronounced with mistakes.

In this work, we aim to improve the TTS system pronunciation of OOV words. First, we explore methods that can identify words that are potentially difficult to pronounce. The identification is first step towards the improvement. We propose several measures that can reflect badly pronounced words without any prior language knowledge.

Next, we try to improve the TTS system's pronuncication of the desired words. To achieve this, we employ the user and obtain correct pronunciations from him. So we get training examples and we are able to improve the TTS system by processing the obtained recording, deriving a phonetic transcription (i.e. pronunciation) and adding it to the TTS vocabulary. Moreover, the derived pronunciations can be used to improve the recognition ability of the ASR module, since it is also

dictionary-based.

As it has been suggested, there are several issues, that are related with the OOV words so methods of deriving correct pronunciations has got potentially very useful applications. It can be used to enlarge vocabularies of TTS or ASR systems both offline or on the fly using the user's feedback. This leads to better pronunciation in case of TTS and improved performance in case of ASR systems. There exist several ways how to obtain such feedback, however, this is not a subject of this work. Theoretically, the method can work with just one gold example, however, it is better to obtain more recordings in general. In Figure 1.1 we provide basic example of simple dialogue, illustrating how real application could look like. However, in this work we assume the user's recording(s) have been gathered already and we do not consider the dialogue policy.

```
System:  Hello, /AANDRZHEZH/.
User:  You said it wrong, my name is /ONDRZHEI/.
System:  /ANDREY/, correct?
User:  No, it is /ONDRZHEI/.
System:  Oh, /ONDRZHEI/?
User:  That's right.
```

Figure 1.1: Sample dialogue illustrating the pronunciation correction. The transcriptions of the user's name are given in ARPABET[1]

2. Overview of used techniques and algorithms

2.1 Audio signal processing

We sketch here the basic principles as described in [13], so we can understand our input data. Speech signal in the real world are mechanical waves, so it is obviously analogous quantity. When we process the speech signal with computers, it is assumed to be digitised, so it is converted into discrete form. When performing digital signal processing, we are usually concerned with three key issues.

1. To remove the influence of phase, because the ear is not sensitive to phase information in speech, and we can use frequency domain representation.

2. Performing source/filter separation, so that we can study the *spectral envelope* of sounds. Spectral envelope characterizes the frequency spectrum of the signal, which is essential for the speech recogniton and processing.

3. We often wish to transform these spectral envelopes and source signals into more efficient representations.

Overview of the main standard processing steps follows.

The input signal is divided into *windows*. Windowing considers only some part of the signal. Usually, the signal is transformed at window borders to prevent discontinuities. This is achieved for example by use of *Hamming* window. The complete waveform is therefore a series of windows, that are sometimes called *frames*. The frames overlap, this is influenced by the size of the *frame shift*. Inside one frame, we assume, that the speech signal is stationary and we transform it to a frequency domain by Discrete Fourier Transformation[1].

It is better for observing the natural speech characteristics, to use a logarithm scale instead of linear. In fact, the *mel scale*[2] is frequently used, which even better corresponds with the human perception. Further, the so called *cepstrum*[3] is computed from the log magnitude spectrum, by performing inverse Fourier transformation. Cepstrum can be represented by coefficients, number of which can be chosen. Those are called the *Mel Frequency Cepstral Coefficients(MFCCs)*. To sum up, we have described how to process the digital audio signal and convert it into discrete series of overlaping frames, each of which is represented by a fixed-length vector of MFCCs.

We gave a brief overview of the signal processing procedure that is essential to work with the audio signal. When we mention the input speech signal, we mean series of MFCC vectors, unless explicitly stated otherwise.

[1] https://en.wikipedia.org/wiki/Discrete_Fourier_transform
[2] https://en.wikipedia.org/wiki/Mel_scale
[3] https://en.wikipedia.org/wiki/Cepstrum

2.2 Mel Cepstral distortion

Mel Cepstral Distortion [**?**] is a well described measure, that should mirror differences between speech samples.There are some caveats worth stating at the outset. Firs, there are many other factors that contribute to the perception of voice quality. For example it takes no account of speech dynamics, either short-range differentials or long-range prosodic effects. Morevoer, distortions in the pitch contour are ignored. However, it should reflect the differences in speech quality and pronunciation.

As we have described in the previous section, the audio sample can be described by its *cepstrum*. This cepstrum may be then represented by Mel Frequency Cepstral Coefficients (MFCCs). Its order can be chosen arbitrarily, we used 35 coefficients length vectors. Thus we can transform each audio record into a sequence of float vectors with length 35. The MCD [**?**] of two sequences is basically normalized Mean Squared Error between those sequences after aligning. We define it as follows:

$$MCD(v^1, v^2) = \frac{\alpha}{T'} \sum_{ph(t) \neq SIL} \sqrt{\sum_{d=1}^{D} (v_d^1(t), v_d^2(t))^2} \tag{2.1}$$

The T' stands for the number of frames of the shorter of the recordings. The common use of MCD involves omitting the 0-th coefficient, since it corresponds to the intensity of signal and therefore it is not usually desired. The scaling factor α is present for historical reasons.

The MCD measure has one substantial disadvantage: it does not reflect, that the two compared recordings may not have the same length and, what is more, they can be misaligned. Thus, we use a modified algorithm that first aligns the sequences using dynamic time warping and then computes MCD with the result.

2.3 Finite state transducers

Finite state transducers (FSTs) [7], are basically finite state automata that are augmented by addition of output labels to each transition. We can further modify the FST and add a weight to each edge. We define a semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$ as a system that fullfills the following conditions:

- $(S, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$
- $(S, \otimes, \overline{1})$ is a monoid with identity element $\overline{0}$
- $\otimes$ distributes over $\oplus$
- $\forall a \in S : a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$

With this definition of the semiring, we can describe a weighted FST [8] T over a semiring S formally as a 8-tuple $(\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ provided that:

- Σ, Δ are finite input and output alphabets

- Q is a finite set of states
- $I \subset Q$ is a set of initial states
- $F \subset Q$ is a set of final states
- E is a multiset of transitions
- $\lambda : I \to S$ is an initial weight function
- $\rho : F \to S$ is a final weight function.

Each transition is element of $Q \times \Sigma \times \Delta \times S \times Q$. Note, that since E is a multiset, it allows two transitions between states p and q with the same input and output label, and even the same weight. However, this is not used in practice. An example of weighted FST is given in Figure 2.3.

Finite state transducers have been used widely in many areas, especially linguistics. They can represent local phenomena encountered in the study of language and they usage often leads to compact represantations. Moreover, it is also very good from the computational point of view, since it advantageous in terms of time and space efficiency. Whole area of algorithms has been described that consider FSTs. One of the most important is *determinization*, which determinize paths in the FST and thus allows the time complexity to be linear in the length of input. Weighted FSTs have been widely used in the area of automatic speech recogniton. The allows to compactly represent the hypothesis state of the acoustic model as well as combining it with the information contained in the language model.

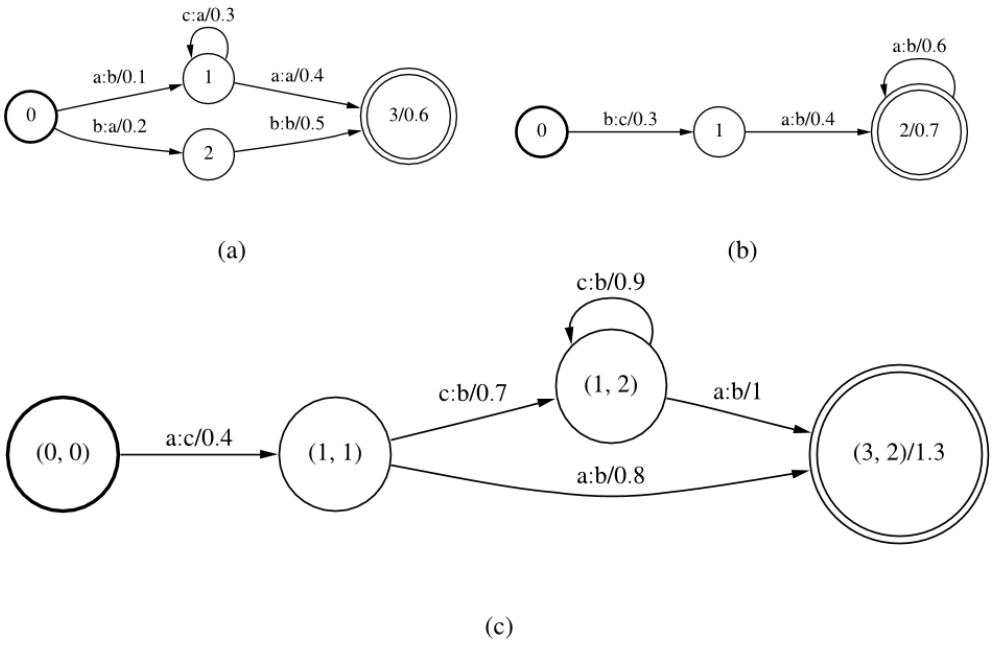

Figure 2.1: Example of Finite State Transducers and their composition

2.4 Automatic speech recognition (ASR)

Overview

The task in ASR is quite clear: An utterance spoken in natural language should be translated into its text representation. Formally, we are given a sequence of acoustic observations $\mathbf{X} = X_1 X_2 \dots X_n$ and we want to find out the corresponding word sequence $\mathbf{W} = W_1 W_2 \dots W_m$ that has a maximum posterior probability $P(W|X)$i.e., according to the Bayes rule:

$$\hat{\mathbf{W}} = argmax_w \frac{P(\mathbf{W})P(\mathbf{W}|\mathbf{X})}{P(\mathbf{X})} \tag{2.2}$$

The problem's solution consists of two subtasks. First, we have to build accurate acoustic model that describes the conditional distribution $P(W|X)$. The second problem is to create a language model that reflects the spoken language that should be recognized. Usually, a variation of the standard N-gram approach is sufficient.

Individual words are composed of phonetic units, which are in turn modeled using states in probabilistic machinery such as a finite state transducers. Because the speech signal is continuous, it is transformed to discrete sequence of samples. MFFC's are commonly used to represent the signal. The process of deriving this sequence is described in further detail in section 2.1. In general, it is difficult to use whole-word models for the acoustic part, because every new task may contain unseen words. Even if we had sufficient number of examples to cover all the words, the data would be too large. Thus, we have to select basic units to represent salient acoustic and phonetic information. These units should be *accurate*, *trainable* and *generalizable*. It turns out that the most appropriate units are phones. Phones are very good for training and generalization, however, they do not include contextual information. Because of this, so called triphones are commonly used.

To model the acoustics, Hidden Markov Models(HMM) are commonly used, because they can deal with unknown alignments. In recent years, neural networks have experienced big breakthrough and they found application even in the area of speech recognition. Namely, recurrent neural networks are able to work with an abstraction of memory and process sequences of variable length, so it overcomes the HMM's in terms of accuracy. The language and acoustic models are traditionally trained independently and they are joined during the decoding process. The decoding process of finding the best matched word sequence $\mathbf{W}$ to match the input speech signal $\mathbf{X}$ in speech recognition systems is more than a simple pattern recognition problem, since there is an infinite number of word patterns to search in continuous speech recognition. A lattice is built in order to make the decoding. Its nodes represent acoustic units and edges are assigned costs. This assignment corresponds to the likelihood determined by the acoustic model as well as the language model, so the FST is a composition of theses two. Decoding the output is realized as searching the best path through this lattice.

The decoding graph is costructed in such a way that the path can contain only

words present in the vocabulary the language model is built on. Thus, each possible path consists of valid words. Because of that, we can see the word as a basic unsplittable unit, although it is actually composed from phones, or triphones respectively. However, we can build an FST that allows to construct the path from phones and thus recognize the input on the phonetic level. Also the search algorithm preserves alternative paths so in the end it gives us not only the best path but also list of alternative hypotheses. We call it the n-best list. Each hypothesis in the n-best list is associated with its likelihood that expresses information from both the language and acoustic model. In TODO:chapter we explore the n-best list and propose method that exploits it.

2.5 Text-to-speech (TTS)

Overview

[13] The text-to-speech problem can be looked at as a task of transforming an arbitrary utterance in natural language from its written form to the spoken one. In general, these two forms have commonalities in the sense that if we can decode the form from the written signal, then we have virtually all the information we require to generate a spoken signal. Importantly, it is not necessary to (fully) uncover the meaning of the written signal, i.e. employ Spoken Language Understanding (SLU).

On the other hand, we may need to generate prosody which adds some information about emotional state of the speaker or emphasizes certain parts of the sentence and thus changing the meaning slightly. To obtain prosodic information, sophisticated techniques need to be involved, since its not a part of common written text, except some punctuation. Another difficulties stem from the fact, that we often need to read numbers, or characters with special meanings such as dates or mathematical equations. The problem of converting text into speech has been heavily explored in the past and the TTS systems (engines) are very sophisticated nowadays. The subsequent section, describes briefly the typical architecture of TTS systems.

TTS system architecture

Let us now describe a common use of TTS system. The architecture usually consists of several modules, although some end-to-end systems base on neural networks have appeared recently ([15], [16]). First, the input text is divided into sentences and each sentence is further tokenized, based on whitespace characters, punctuation etc. Then, the non-natural language tokens are decoded and transformed into text form. We then try to get rid of ambiguity and do prosodic analysis. Although much of the information is missing from the text, we use algorithms to determine the phrasing, prominence patterns and tuning the intonation of the utterance.

Next is the synthesis phase. The first stage in the synthesis phase is to take the words we have just found and encode them as phonemes. There are two

main approaches how to deal with the synthesis phase. More traditional approach is so called **unit concatenation**. This approach uses a database of short prerecorded segments of speech (roughly 3 segments per phoneme). These segments are then concatenated together with some use of signal processing so they fit together well. An alternative is to use machine learning techniques to infer the specification-to-parameter mapping from data. While this and the concatenative approach can both be described as data-driven, in the concatenative approach we are effectively memorizing the data, whereas in the statistical approach we are attempting to learn general properties of the data.

Trained models are able to transform the phonemes into audio signal representation. This representation is then synthesized into waveforms using a vocoder module. Although unit concatenation approaches generally achieve better results, the statistical ones are more flexible and have good possibilities to fine tuning or postprocessing.

Grapheme to Phoneme conversion and its drawbacks

In the stage of converting graphemes (i.e. text) to phonemes, a *g2p* module is commonly used. The g2p task is to derive pronunciations from ortographic transcription. Traditionally, it was solved using decision trees models. It can also be formulated as a problem of translating one sequence to another, so neural networks can be used to solve this task ([17]). However, in our work we use joint-sequence model ([2]), which estimates joint distribution of phonemes based on probabilities derived from n-grams. The *g2p* module is a crucial component of the system and it has great impact on the final pronunciation. Since we use machine learning techniques and train on the dictionary data, the model inherently adopts pronunciation rules from the respective language. Although it is in principle possible to train multilingual *g2p* ([11]), it is usually not the case in common TTS systems. The problem arises when words that originate in some foreign language should be pronounced. In this work we address this issue by deriving pronunciations from the ASR output.

Speech Synthesis Markup Language (SSML)

[14] SSML is an XML standard that allows to specify aspects of the speech. It is an input to the synthesis module and many TTS engines support its use. We can specify emotions, breaks etc. with the help of SSML. More importantly, it can be used to input particular phonemes and thus circumvent the *g2p* module. In our work we use this method to feed our phonetic transcriptions into the engine. The disadvantage of this method is, that many TTS engines process the SSML input imperfectly or not at all. However, in principle it is possible to add the transcriptions directly into the system's vocabulary.

```
 <?xml version="1.0"?> <speak version="1.0" xmlns="..."
xmlns:xsi="..." xsi:schemaLocation="..." xml:lang="en-US">
    <voice gender="female">Mary had a little lamb,</voice>
    <voice gender="female" variant="2">
        Its fleece was white as snow.
    </voice>
    <voice name="Mike">I want to be like Mike.</voice>
</speak>
```

Figure 2.2: Example of SSML code

Overview of the used TTS systems

1. Cereproc[4] This engine is a commercial software that is free for educational purposes. Although it is not open-sourced, we use it for its good quality and reliable outputs.

2. MaryTTS[5] MaryTTS is a German open-source project written in Java. It is highly customizable and modular. Thus we can explore output of an arbitrary module or replace it in the processing pipeline. MaryTTS is based on client-server architecture.

3. gTTS[6] is a TTS service provided online by Google. It achieves very good quality, however it allows ony use of one voice in the free version.

4. Pico [7] SVOX Pico TTS is a lightweight engine that lacks a good quality of the gTTS, however it offers a large selection of voices and it is commonly used on the phones with Google's Android operating system.

2.6 Algorithms description

Dynamic Time Warping (DTW)

[9] The measurement of similarity between two time series is an important component of many applications. Moreover, sometimes we need to align two sequences that describe same data but are of different lengths. Both this tasks can be solved with use of DTW. Suppose we have two time series, a sequence Q of length n, and a sequence C of length m, where

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n$$
$$C = c_1, c_2, \ldots, c_j \ldots, c_m$$

To align these two sequences using DTW, we first construct an n-by-m matrix where the (i^{th}, j^{th}) element of the matrix corresponds to the squared distance, $d(q_i, c_j) = (q_i - c_j)^2$, which is the alignment between points q_i and c_j. To find the best match between these two sequences, we retrieve a path through the matrix

[4]https://www.cereproc.com/
[5]http://mary.dfki.de/
[6]https://pypi.python.org/pypi/gTTS
[7]https://github.com/stevenmirabito/asterisk-picotts

that minimizes the total cumulative distance between them. In particular, the optimal path is the path that minimizes the warping cost:

$$DTW(Q,C) = \sqrt{\sum_{k=1}^{K} w_k} \tag{2.3}$$

where w_k is the matrix element $(i,j)_k$ that also belongs to k^{th} element of a warping path W, a contiguous set of matrix elements that represent a mapping between Q and C. Methods of dynamic programming are used to fill in the values and find alignment of sequences. Thus a result of this algorithm is a mapping σ that can be used to construct sequence of pairs A containing members of both sequences C, Q in each time step. σ is constructed in such a way, that it holds:

$$A_i = (C_{\sigma(C,i)}, Q_{\sigma(Q,i)}); i = 1 \dots K \tag{2.4}$$

$$max(|C|, |Q|) \leq K \tag{2.5}$$

$$\sum_{i=0}^{K} d(\sigma(C,i), \sigma(Q,i) is minimal \tag{2.6}$$

Where $d(x,y)$ is an arbitrary distance metric. In our application, we want to align two sequences of phonemes, hence we can treat it as strings and work with repsective distance. We choose Levensthein distance to be our metric.

2.7 Related Work

2.7.1 Grapheme-to-phoneme conversion

An automatic grapheme-to-phoneme conversion was first considered in the context of TTS applications. The input text needs to be converted to a sequence of phonemes which is then fed into a speech synthesizer. It is common in TTS systems that they first try to find the desired word in the dictionary and if it doesn't find it, it employs the grapheme-to-phoneme ($g2p$) module. A trivial approach is to employ a *dictionary look-up*. However, it cannot handle context and inherently covers only finite set of combinations. To overcome this limitations, the rule-based conversion was developed. Kaplan and Kay [4] formulate these rules in terms of finite-state automata. This system allows to greatly improve coverage. However the process of designing sufficient set of rules is difficult, mainly since it must capture irregularities. Because of this, a *data-driven* approach based on machine learning has to be employed. Many such techniques were explored, starting with Sejnowski and Rosenberg [12]. The approaches can be divided into three groups.

Techniques based on local similarities

The techniques presuppose an alignment in the training data between letters and phonemes or create such an alignment in a separate preprocessing step. The alignment is typically construed so that each alignment item comprises exactly

one letter. Each slot is then classified (using its context) and a correct phoneme is predicted. Neural networks and decision tree classifiers are commonly used for this task.

Pronunciation by analogy

This term is typically used for methods that could be described as nearest-neighbor-like. They search for local similarities in the training lexicon and the output pronunciation is chosen to be analogous to retrieved examples. In the work of Dedina and Nusbaum [3] the authors first identify words from the database that match the input string and then build a pronunciation lattice from them. Paths through the lattice then represent the derived pronunciations.

Probabilistic approaches

The problem can also be viewed from a probabilistic perspective. Pioneers in this area were Lucassen and Mercer [5]. They create $1 - to - n$ alignments of the training data using a context independent channel model. The prediction of the next phoneme is based on a symmetric window of letters and left-sided window of phonemes. Authors then construct regression tree, the leafs of which carry probability distribution over the phonemes. A popular approach based on probability modelling is to employ so called *joint sequence models* [2]. We use an open-source implementation of such a model in our work. This approach formalizes the task as follows:

$$\boldsymbol{\varphi(g)} =_{\varphi' \in \Phi^*} p(\boldsymbol{g}, \boldsymbol{\varphi'}) \tag{2.7}$$

where * denotes a Kleene star. In other words, for a given ortographic form $\boldsymbol{g} \in G^*$ we want to find the most likely pronunciation $\boldsymbol{\varphi} \in \Phi^*$, where G, Φ are ortographic and phonetic alphabets. The fundamental idea of joint-sequence models is that the relation of input and output sequences can be generated from a common sequence of joint units. These units carry both input and output symbols. Formally, the unit called *grapheme* is a pair $q = (\mathbf{g}, \varphi) \in Q \subset G * \times \Phi$ where $\mathbf{g}$ and φ are letter and phoneme sequences which can be of different lengths. In the simplest case, each unit carries zero or one input and zero or one output symbol. This corresponds to the conventional definition of finite state transducers (FST) that are described in 2.3. The letter and the phoneme sequences are grouped into an equal number of segments. Such a grouping is called a joint segmentation. The joint probability is defined as:

$$p(\mathbf{g}, \varphi) = \sum_{\mathbf{q} \in S(\mathbf{g}, \varphi)} p(\mathbf{q}) \tag{2.8}$$

since there are many possible groupings of input sequences in general. The joint probability distribution $p(\mathbf{g}, \varphi)$ has thus been reduced to a probability distribution $p(\mathbf{q})$ over graphone sequences $\mathbf{q} = q_1, \ldots, q_K$, which can be modeled using N-gram approximation:

$$p(q_1^K) \cong \prod_{j=1}^{K+1} p(q_j | q_{j-1}, \ldots, q_{j-M+1}) \tag{2.9}$$

The probability distribution is then typically estimated by an Expectation-Maximization algorithm.

Generally, the problem of the wrong pronunciation in TTS is caused by a bad phonetic transcription. Traditional TTS systems are modular, one module's output is inputted into the next one. Because of this fact, the errors cumulate and thus the mistakes made by $g2p$ cannot be repaired. So if we want to improve the pronunciation, we can try to improve the $g2p$ as it is done in [**?**]. Authors in this work propose a method of exploiting a $g2p$ trained on a language with a high number of available resources to create a $g2p$ for language for which we do not have sufficient number of examples. This method relies on the existence of a conversion mapping between these two languages. Also it requires to do the conversion for every new language. In theory, a model can be created, that is able to transcribe grapheme sequences into an appropriate phonetic representation and can handle multiple languages. However, it needs to somehow obtain information, which language the input sequence comes from, which is not straightforwardly doable. This method has several drawbacks, because the language is not always known and the set of known languages is limited, so it does not realy solve the OOV problem. Also, it potentially requires a lot of training data. Moreover, if we want to learn a new pronunciation of just one word, it's more convenient to do it in a different way.

2.7.2 Learning pronunciation from spoken examples

. This group of methodologies aim to derive phonetic transcriptions directly from audio input. They are built on the theory of Automatic Speech Recognition which we discuss in 2.4. Authors of [**?**] introduce method of deriving correct pronunciation for a word in order to enlarge the recognizer's dictionary. They propose a data-driven approach to automatically add new words and their variants, respectively. The authors argue that in the spontaneous speech, the most frequent pronunciation does not need to be the one that is marked as correct and is used in the training phase. Thus the overall performance of the recognizer may be degraded since the phonetic units are bound with inadequate acoustics. The method is proposed, that relies on the use of both phoneme and word-level speech recognizer. The phoneme recognizer is constructed using smoothed bigram Language model. We discuss the phoneme recognizers in more detail in 2.4. The algorithm collects all occurences of words in the database and creates phonetic transcriptions using the recognizer. It then sorts the variants, rejects some of them an creates an $n-best$ list which is added to the dictionary. The recognizer can then be retrained, allowing multiple pronunciations for each word. Thus the recognition performance can be improved by an automatic procedure without the need for using the phonological rules.

In the work of [6], the concept of Pronunciation Mixture Models is introduced. Authors use a special kind of speech recognizer, search space of which has four primary hierarchical components: the language model G, the phoneme lexicon L, the phonological rules P that expand the phoneme pronunciations to their phone variations, and the mapping from phone sequences to context-dependent model labels C. These can be with advantage represented as FSTs and thus the

full decoder network can be represented as a composition of these components: $R = C \circ P \circ L \circ G$. A probabilistic lexicon is considered in a sense, that several pronunciations are allowed for each word and there is no hard limitation that would force the recognzier to choose one. Instead, kind of soft voting is considered, meaning, that each transcription is used with certain probability. Also, joint-sequence modelling is considered, as we introduced in 2.7.1, so each transcription is considered together with its orotgraphic form. That means, that we can describe the log-likehood of M utterances $D = \{\mathbf{u}_i, \mathbf{W}_i\}$ where $\mathbf{u}_i$ are speech data and $\mathbf{W}_i$ their transctiptions as follows:

$$\mathcal{L}(\Theta|D) = \sum_{i=1}^{M} log \sum_{B \in \mathcal{B}} P(\mathbf{u}_i, \mathbf{B}, \mathbf{W}_i; \Theta) \tag{2.10}$$

where $\mathbf{B}$ are respective phone sequences (i.e. pronunciations) and Θ represents the model parameters. Then, we derive using a chain rule:

$$P(\mathbf{u}_i, \mathbf{B}, \mathbf{W}_i; \Theta) = P(\mathbf{u}_i|\mathbf{B})P(\mathbf{B}|\mathbf{W}_i; \Theta)P(\mathbf{W}_i) \tag{2.11}$$

If we further assume, that pronunciation sub-units $\mathbf{b}_i \in \mathbf{B}$ are context independent, we can transcribe the above expression:

$$P(\mathbf{u}_i|\mathbf{B})P(\mathbf{B}|\mathbf{W}_i; \Theta)P(\mathbf{W}_i) = P(\mathbf{u}_i|\mathbf{B})(\prod_{j=1}^{k_i} P(\mathbf{b}_j|\mathbf{w}_j^i; \Theta))P(\mathbf{W}_i) \tag{2.12}$$

The model parameters are then estimated using the EM-algorithm. Parameters related to the language model, can be initialized with use of graphone language model. Several technical issues has to be dealed with, however the Pronunciation Mixture Models can be trained on the same data as traditional ASR engines and can be used to obtain phonetic transcriptions from audio data.

Similar approach is considered in the work [10], except they do not have access to the acoustic models or phone lattices, only the word recognition mistakes. An OOV word is passed through an ASR decoder giving an n-best word recognition output. Since the words are OOVs, every hypothesis will be a recognition mistake. These mistakes are then exploited, assuming that the following generative story of the recognition output for a word $\mathbf{w}$ holds:

1. A pronunciation baseform $\mathbf{b}$ is drawn from the distribution Θ.

2. A phonetic confusion function from the word $\mathbf{w}$ and the selected baseform $\mathbf{b}$ is applied in order to generate a phoneme sequence $\mathbf{p}$ with probability $P(\mathbf{p}|\mathbf{b}, \mathbf{w})$

3. A word sequence $\mathbf{e}$ with probability $P(\mathbf{e}|\mathbf{p}, \mathbf{b}, \mathbf{w}) = 1$ is generated using the pronunciation lexicon.

Authors model the joint probability of hypothesis and reference word $P(\mathbf{e}, \mathbf{w}) = P(\mathbf{w} \sum_b f_{e,b,w})$, where $f_{e,b,w} = P(\mathbf{e}|\mathbf{b}, \mathbf{w})$ is the phonetic confusion function and it is used to estimate the distribution $P(\mathbf{b}|\mathbf{w}, \mathbf{e})$. Thus they are able to derive pronunciations without access to ASR lattices, i.e. it only consideres the recognizer as a black-box.

2.7.3 Conclusion

Many approaches were introduced, that are able to convert an utterance in ortographic or audio form to its phonetic representation. $G2P$ converts the grapheme transcriptions, using only the text input. It is a well explored field of study with many different variants of realization. Although it achieves very good results nowadays, it suffers from the fact, that same groups of letters may have different pronunciations in different languages. We typically don't have access to the information which language is considered and it may be difficult to get access to sufficient number of datasets. The latter problem can be partially solved by transfer learning as proposed in [**?**]. Alternatively, one can derive pronunciations directly from audio signal. This approach has been also explored by some authors, however it usually requires quite low level modifications of the speech recognizer. Also, the authors used the derived pronunciations to enlarge the phonetic dictionary of the recognizer, not to improve the Text-To-Speech sytems. We explore methods of merging the mentioned approaches to combine both textual and acoustic information and its usability when confronted with human judgments.

2.8 Thesis overview

3. Methodology

3.1 Used approaches

- use of ssml - phonetic alphabets

3.2 Overview and key insight

We aim to improve the pronunciation, that means, we need to derive phonetic transcriptions of good quality. Straightforward way of achieving this is to use a g2p module. As we discuss in 2.5 and 2.7, this approach has some limitations, mainly because the pronunciation of some tokens in different languages may vary. However, grapheme-to-phoneme conversion is a well explored area and its output may serve as sufficiently good baseline. In 2.7 we also discuss methods, that are able to derive phonetic transcriptions directly from the speech signal, using a speech recognition framework. These methods are somewhat complex and they are typically specific for a particular recognizer. Rather than improving any of these methods, we aim to explore a possibility to combine them. That is, we want to exploit both textual and acoustic information to derive phonetic transcriptions of the desired words.

In the desired settings, the acoustica data with user's recordings are obtained online. It can be difficult to make user say words we are interested in, so the ultimate goal is to develop algorithm, that is able to extract those words from a dialogue history or ask user to say them. Crucial point of this approach is, that the user should not feel bored by the process. For a purpose of this work, we assume that we have already obtained recordings with correct pronunciations and work with it. In real setting we would have to employ dialogue policy to be able to gather our own recordings or process the history and try to identify respective words.

3.3 Identifying difficult words

If we want to improve pronunciations, we should first identify the words that are mispronounced. Obviously, we could let the TTS system to pronounce each word in a best effort style and user would identify mistakes and correct them. This approach hase several drawbacks. First, the communication with the user is rather complicated, since the badly pronounced word has to be isolated prior to obtaining the correct pronunciation. Second, it may be unpleasant for the user if he or she has to undergo this process and hear the bad pronunciation. Thus, it would be better if we could recognize the possibly difficult words somehow. It would mean, that we can ask user directly for the correct pronunciation of the world. We explore this issue in the rest of this section.

3.3.1 Data

TODO: probably move to experiments section We explore the problem on two datasets. The first one, we call it D_1 is artificial and it contains Czech and English words. The Czech are chosen in a way, that the should be difficult to pronounce for a TTS engine trained on English. On the other hand, the English words were picked from the cmu dictionary TODO: cite so it should be pronounced correctly. We then synthesized each word by three synthesizers, namely it was *gtts*, *cereproc* and *svox*. The other dataset, D_2 was created as the random subsample of the Autonomata corpus. It contains one hundred Dutch names.

The data labeling. Labeling of D_1 was performed by three annotators. Each of them had to listen each of the words three times, each time synthesized by different engine. He then labeled each of the recording with a discrete number ranging from one to five. The recordings labels were averaged. Thus we obtain labels per each recording and we have three labels in total for each word. We average these three labels and obtain a score between one and five, that describes overall quality of the word's pronunciation.

3.3.2 Measuring the difficulty

In order to estimate the difficulty of each word's pronunciation, we propose three measures, which we introduce in this section. The key idea is, that we obtain values for each of this measures and then combine them in one feature vector that represents the recording. Then we can train a classifier, that learns to predict quality of the pronunciation.

M_1 measure - averaged Mel Cepstral Distortion

We discuss Mel Cepstral Distortion in detail in TODO: ref. We use it here to define the M_1 measure. We can describe it as follows:

$$M_1(k) = \frac{1}{N} \sum_{(i,j)} MCD(r_{ki}, r_{kj}) \tag{3.1}$$

Where $N = \binom{3}{2}$, (i, j) stands for every combination of synthesizers, r_{ki} is name k synthesized by engine i and MCD is the Mel Cepstral Distortion. The key idea motivating this measure is, that if there is a problem with a pronunciation of some part of the word, every synthesizer have to deal with it somehow. It is likely, that each of them will do it in sligthly differnt way. This implies, that the pairwise MCD will increase.

Nevertheless, the process of computing value of M_1 introduce some problems. The troubles stem from the fact, that there is high variability in the data obtained from the synthesizers. That is, one synthesizer's output may differ greatly from the others, so it biases the result. TODO: figure. If we had enough synthetizers, we could afford to ignore values of the outliers and thus smooth the results

We tested, how good is the correlation of M_1 measure and human judgement.

First, we synthesized the words by different synthesizers. Each recording created this way was then labeled by human. Set of recordings for each word was then used to compute M_1 value and this was compared with the mean of the respective human judgements. Results on our sample dataset with the use of the 0-th coefficient and without it are shown in Figure **??** and Figure **??** respectively. Based on these results, it may seem, that the M_1 measure is not very good, the best correlation was 0.471. However, it may add important peace of information to the feature vector.

The MCD has one feature, that can be looked at as a disadvantage: It does not weight its coefficients, respectively it weights all with the same weights. We propose to change it - for example linear model could be trained, that finds the combination of MFCC's that corresponds the best. The coefficients derived by this model would be difficult to interpret without further research. However, we can train it and try to use it instead of typical MCD. TODO: further variations

M_2 measure - averaged phonetic distance

Another measure we can possibly use is based on phonetic transcriptions. We first recognize the recordings with a phoneme recognizer, thus we obtain a sequence of characters per each recording. We can compute pairwise distances of these transcriptions, using for example Levensthein [**?**] or Hamming distance, which we normalize. The motivation is similar to the M_1 measure. Assuming the difficult words have positions that are problematic for the TTS engine, the recognized phonemes on these positions should differ and thus the distance between transcriptions should increase. Four our purposes, we have used Levenshtein distance as a metric.
The M2 measure is described by equation:

$$M_2(k) = \frac{1}{N} \sum_{(i,j)} \frac{LD(Hyp(r_{ki}), Hyp(r_{kj}))}{max(len(r_{ki}, len(r_{kj})))} \tag{3.2}$$

Where $N = \binom{3}{2}$, (i, j) stands for every combination of synthesizers and r_{ki} is name $\{k\}$ synthesized by engine $\{i\}$. LD means Levenshtein Distance and Hyp represents the best hypothesis from the phonetic recognizer. Note, that we normalize the distance by length of the longer transcriptions.

M_3 measure - occurence of bad bigrams

The M_3 measure is based on a different approach. We want to learn the typical mistakes of $grapheme - to - phoneme$ converter, concretely which groups of graphemes are difficult to transcribe for it. We suppose, that words with many occurences of such groups are difficult to pronounce. To compute the measure, we need two corpuses (C_1 and C_2) in different languages and a $grapheme - to - phoneme$ ($g2p$) training algorithm. The corpuses should consist of pairs (word, transcription). We can then prepare for computation of M_3 in three stages:

1. Train $g2p$ model G_1 on corpus C_1 in language L_1.
2. Use trained G_1 to transcribe words contained in corpus C_2.

3. Identify problematic parts.

Stage 3 needs further description. We obtained list of triples, i^{th} of which is structured:

(word w_i from C_2, original transcription t_i^o, transcription t_i^h derived by G_1)

We can then use the Dynamic Time Warping algorithm to obtain pairwise alignments of these sequences, illustrated on Figure 3.3.2 In fact, we first transform the phoneme sequences into sequences of bigrams. This is because graphemes and phonemes are not in one-to-one correspondence and bigrams capture the relations better. Once we have the alignments, we can identify positions, where original

h	ɛ	l	ə	ʊ
h	ə	l	o	ʊ

Figure 3.1: Sample alignment of two phonetic sequences

transcriptions from C_2 differ from the hypothesis derived in stage 2. Respective bigrams from the original word can then be identified. For each bigram, we count number of times it was marked as difficult. Each word can then be scored according to number of bad bigrams it contains. To evaluate the M_3 measure, the MaryTTS framework was used, because we can extract phonetic transcriptions from it and thus use its $g2p$ module. This means, we are able to derive the transcriptions with the exactly same module, that is used in the synthesizer. We have also used the Cereproc engine. Because we do not have access to the $g2p$ that Cereproc uses, we trained our own model on the *cmu* dictionary[1]. The data was then processed and the confusion matrix was created, containing number of times, respective bigrams are confused with each other. The matrix turns out to be quite sparse, which is not surprising, since the majority of bigram pairs are interchanged with probability nearly zero. TODO: describe settings, move to experiments? If we restrict the matrix only to values greater than certain threshold, its dimensions decrease dramatically and we can visualize it, the result can be seen in Figure 3.3.2 The scoring procedure counts for each word a number of occurrences of bigrams that were confused and the number of confusions is summed and divided by the length of the word. We plot the scores in Figure 3.3.2. The shape of the curve corresponds to the fact, that many bigrams does not occur a lot, or are not problematic. The correlation of this measure and annotations was 0.20 for the MaryTTS and 0.31 for the Cereproc, respectively. Intuitively, if we have access to the $g2p$ module, which is the MaryTTS case, it is expectable, that the results would be better. However, this turns out not to be true, which is surprising. TODO: why?

The disadvantage of this approach is, that it relies quite heavily on the alignment process, which is imperfect, so sometimes it can mark as confused pair of

[1] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

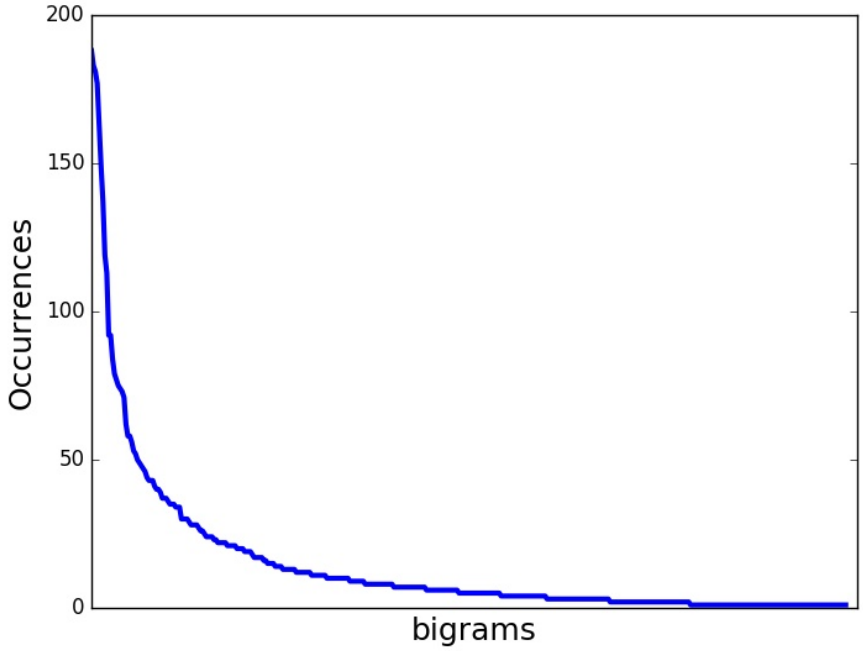

Figure 3.2: Frequencies of confusing respective bigrams, i.e. how many times it occurs in some confused pair. We can see, that the majority of bigrams has somewhat low scores.

bigrams, that is not correct. Also occurence of some bigrams is very sparse, so its scores may be estimated poorly. Another possible problem is, that it relies on the corpuses with phonetic transcriptions and then it is specific to certain pair of languages.

Measure combination

We haver seen, that neither of the measures correlates well with the annotators' labeling. Despite this fact, it may by interesting to explore, whether the combination of the measures can work better. As described in the introduction to this section, we combine the measured values to one feature vector that represents the word. We then train a *Ridge Regression* model using the averaged human labels as a target variable. We have used 5 fold cross validation and measured the R^2 and $MeanSquaredError(MSE)$, which we averaged over all the folds. The measured MSE was 0.43 and R^2 was 0.58. We remind, that the human labels are discrete values in the interval $[1, 5]$, so the MSE value is good, saying, that we differ from the human label by less than 0.5. We can choose a threshold and state, that if the model's predicted value is greater than the threshold, the respective word is hard to pronounce and we should try to improve the pronunciation.

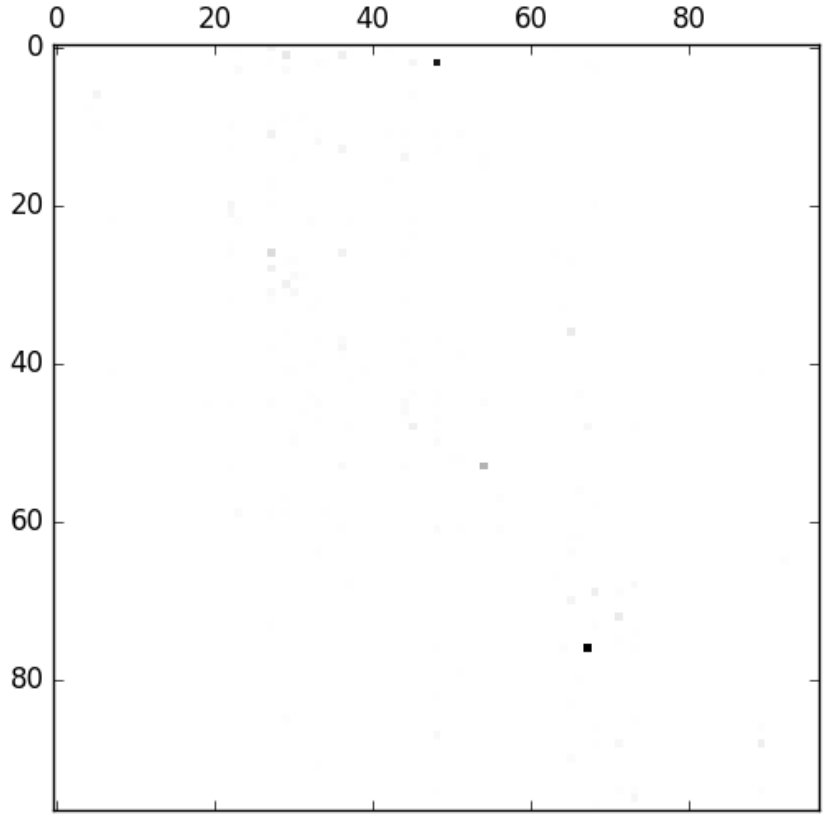

Figure 3.3: Visualization of the confusion matrix computed from the results.

Conclusion

Bibliography

[1] Arpabet overview. https://nlp.stanford.edu/courses/lsa352/arpabet.html. Accessed: 2017-04-16.

[2] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.

[3] Michael J Dedina and Howard C Nusbaum. Pronounce: a program for pronunciation by analogy. *Computer speech & language*, 5(1):55–64, 1991.

[4] Ronald M Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378, 1994.

[5] John Lucassen and Robert Mercer. An information theoretic approach to the automatic determination of phonemic baseforms. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, volume 9, pages 304–307. IEEE, 1984.

[6] Ian McGraw, Ibrahim Badr, and James R Glass. Learning lexicons from speech using a pronunciation mixture model. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(2):357–366, 2013.

[7] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

[8] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

[9] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

[10] Sravana Reddy and Evandro B Gouvêa. Learning from mistakes: Expanding pronunciation lexicons using word recognition errors. In *INTERSPEECH*, pages 533–536, 2011.

[11] Tim Schlippe, Sebastian Ochs, and Tanja Schultz. Grapheme-to-phoneme model generation for indo-european languages. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4801–4804. IEEE, 2012.

[12] Terrence J Sejnowski and Charles R Rosenberg. *NETtalk: A parallel network that learns to read aloud*. MIT Press, 1988.

[13] P. Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009. ISBN 9781139477260. URL https://books.google.cz/books?id=T0O-NHZx7kIC.

[14] Paul Taylor and Amy Isard. Ssml: A speech synthesis markup language. *Speech communication*, 21(1-2):123–133, 1997.

[15] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[16] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: A fully end-to-end text-to-speech synthesis model. *arXiv preprint arXiv:1703.10135*, 2017.

[17] Kaisheng Yao and Geoffrey Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196*, 2015.

List of Figures

List of Tables

List of Abbreviations

Attachments