THESIS

SINGLE-SPEAKER END-TO-END NEURAL TEXT-TO-SPEECH SYNTHESIS

Master Thesis

by

Yves-Noel Weweler

Thesis submitted to the Department of Electrical Engineering and Computer Science, University of Applied Sciences Münster, in partial fulfillment of the requirements for the degree

Master of Science (M.Sc.)

Advisor: Prof. Dr.-Ing. Jürgen te Vrugt
Co-Advisor: Prof. Dr. rer. nat. Nikolaus Wulff

Steinfurt, October 10, 2018

Abstract

Producing speech is a natural process for humans, but challenging for computers. Text-to-Speech (TTS) synthesis is generally considered a large scale inverse problem, transforming short written sentences into waveforms. This transformation is highly ambiguous and is influenced by many factors that are difficult to model. Traditional systems are based on complex multi-stage hand-engineered pipelines, requiring extensive domain knowledge; posing a significant challenge to model. This thesis introduces and analyzes a neural end-to-end generative speech synthesis architecture derived from *Tacotron* [1]. Instead of having humans identify and extract complicated rules and patterns, the computer learns to produce speech from unaligned text-audio-pairs. All components are described and training behavior as well as component effects are analyzed. Using a paired comparison 5-scale Mean Opinion Score (MOS) test comparing the proposed system against two current TTS reference systems it achieves a score of 3.4. Analysis suggests, that the feeding of ground-truth frames during training and folding of the decoder target sequence using a reduction factor are essential for predicting long sequences.

Contents

List of Abbreviations

ASR	Automatic Speech Recognition
BN	Batch Normalization
CBHG	1-D convolution bank + highway network + bidirectional GRU
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DSP	Digital Signal Processing
DT	Decision Tree
EOS	End of sequence
FFT	Fast Fourier Transform
GRU	gated recurrent unit
HMM	Hidden Markov Model
IDTFT	Inverse Discrete-Time Fourier Transform
LSMSPEC	linear scale magnitude spectrogram
LSTM	long short-term memory
MCD	Mel-cepstral distortion
MCEP	Mel-cepstral coefficient
MFCC	Mel-frequency cepstral coefficient
MOS	Mean Opinion Score
MSE	Mean Squared Error
MS-LSD	Modulation Spectrum Log-Spectral Distortion
MSMSPEC	mel scale magnitude spectrogram
NLP	Natural Language Processing
NMT	Neural Machine Translation
NN	Neural Network
ReLU	Rectified Linear Units
RNN	Recurrent Neural Network
seq2seq	sequence-to-sequence
SOS	Start of sequence
STFT	short-time Fourier transform
TTS	Text-to-Speech

Glossary

aperiodic parameter	Aperiodic parameters are the non-periodic parts of voice. One example of this are unvoiced fricatives, i.e. sounds that are emitted without any vibration of the vocal cord. They result from the constriction of air flow between the tongue and the roof of the mouth.
end-to-end	Neural networks generally receive inputs and produce outputs through processing the inputs across layers. Training a network in an end-to-end manner refers to training the entire system at once from the input layer to the output layer. Therefore, optimizing all components as one unit contrary to training units separate and combining them later. In order to train a network end-to-end, all modules of a network need to be trainable themselves.
exploding gradient problem	The exploding gradient problem refers to a large increase in the norm of the gradient during training. [2] Usually resulting in numerical problems handling the gradient using a finite representation.
fine-tuning	In the domain of neural networks, fine-tuning refers to the process of further refining an already trained model. Often with a larger or completely different dataset than used to train the model initially.
gram	An N-gram is a continuous sequence of N symbols, commonly characters, words or phonemes for example. An N-gram of size 1 is referred to as an “unigram”. Subsequently, named “bigram” for size 2 and “trigram” for size 3.
grapheme	A grapheme is a basic building block used to create words in written language. Graphemes are highly language dependent. In alphabetic languages like English and French, graphemes can be thought of as letters or characters. While in languages like Chinese each grapheme represents a full word or part of a word. [3, p. 28]

overfitting	Overfitting generally describes the process of a neural network not capturing the underlying structure and relationships of data processed, but memorizing and predicting results for the training data. Therefore, performing well on the training data but failing when predicting data other than that seen during training.
phoneme	A phoneme is the principal unit of form for verbal content, i.e. spoken sound. Phonemes have no direct meaning, they are the basic units or building blocks used to form spoken words. [3, p. 14]
pre-train	In the domain of neural networks, pre-training often refers to the process of bootstrapping a model. The typically randomly initialized network weights are conditioned on a small or more generic dataset for example. Afterwards, the model is fine-tuned specifically.
prosody	In linguistics, prosody is commonly referred to as the rhythm, stress and intonation of speech. In terms of acoustics, the prosody of oral languages involve variation in syllable length, loudness, pitch and the characteristic frequencies of speech sounds. [4]
spectral envelope	The spectral envelope of a signal is the envelope of the signal's amplitude. It serves as a measure of how amplitude varies over time.
vanishing gradient problem	The vanishing gradient problem refers to the gradient going exponentially fast to norm 0 during training. This behavior makes it impossible for a recurrent network to learn correlation between temporally distant events. [2]
vocoder	A vocoder or voice coder, generates voice from a sequence of previously extracted parameters; hence reversing parameters back into voice. Typically, modeling the human vocal tract and its spectral characteristics.

1 Introduction

The generation of synthetic voices using computers is a topic researchers investigate since decades [5]. The process of Text-to-Speech (TTS) synthesis is a subdomain of speech synthesis that aims to generate speech from written text. As stated by Taylor in [3], TTS systems complement several other technologies from the domain of Natural Language Processing (NLP). Their direct counterparts are Automatic Speech Recognition (ASR) systems, whose purpose is to transcribe speech.

This thesis analyzes how neural generative speech synthesis can be used to synthesize speech from unaligned text-audio-pairs in an end-to-end fashion. The proposed model derives from the *Tacotron* architecture [1]. It is an attention-based sequence-to-sequence (seq2seq) model achieving state-of-the-art results; that is 3.4 in a 5-scale Mean Opinion Score (MOS) test[1]. A simplified schematic of the architecture is illustrated in figure 1. It needs to be emphasized that this thesis focuses primarily on approaches from recent years; particularly the Neural Network (NN) based ones.

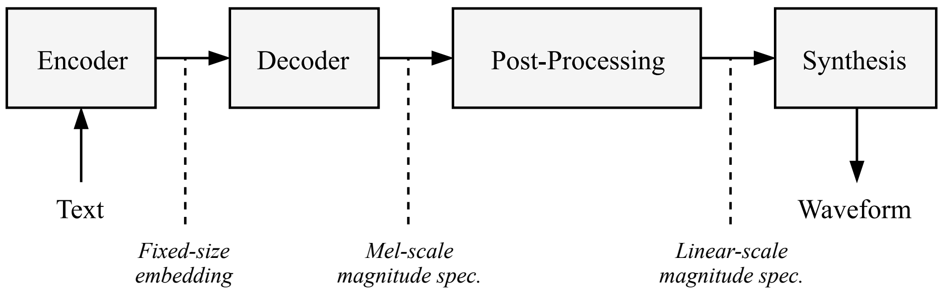


Figure 1: Outline of the models processing stages. It is composed of four stages that transform written text into an audio signal. The encoder and decoder produce a parameter sequence from the text while the post-processing and synthesis components generate a waveform from it.

Neural network based approaches perform well in various domains. Especially setting records in accuracy on many tasks when handling input of varying length using seq2seq models. [6]

Such tasks include image caption generation [7], text style transfer [8] and attention-based seq2seq Neural Machine Translation (NMT) [9]–[12]. However, using these technologies in building a neural speech synthesis system shows that there are unresolved questions to be explored. For example, little empirical data on the amount of mel filter banks required to construct reasonable features for neural models is available [1]. Also worth exploring are the effects of substituting *Bahdanau-style* attention [11] with the computationally more efficient *Luong-style* [10].

Speech synthesis might still be associated with the subject of science fiction. Namely, through the voice of HAL 9000 from the movie *2001: A Space Odyssey* [13], or Majel Barrett-Roddenberry's voice of the onboard computer in the *Star Trek* television series [14]. Albeit these particular systems being purely fictional, sophisticated synthesis systems exist these days. They are a regular part of our daily life with probably most people

[1]Higher is better, with 5 being the best possible score.

having heard a synthetic voice at least once in their lifetime. A prominent example of speech synthesis that comes to mind, might be the voice of professor Stephen Hawking [15]. Today TTS systems are used to support vision impaired people, automate announcement tasks at railroad stations or navigation systems, and include automated handling of customers on the phone; just to name a few. Especially in the past speech synthesis has seen an even more widespread use through the rise of smart-phones, smart-home devices and personal assistants [16, 17, 18].

The synthesis of speech is generally considered a large scale inverse problem. The input being short written sentences that contain a lot of compressed information. This input has to be transformed into a sparse audio waveform of much greater length. The result of this transformation is highly ambiguous and influenced by many factors that are difficult to model. The same text can correspond to a variety of different pronunciations, speaking styles and intonations. Even at a low sample rate like 16 kHz, on average 6,000 samples per word have to be generated [19].

1.1 Contributions

This thesis contributes to the area of experimental single-speaker neural speech synthesis. The primary contribution is the implementation and evaluation of a neural voice synthesis solution that is inspired by current state-of-the-art architectures. The approach makes little assumptions on the training data such that it is generally capable of modeling different languages and alphabets. This work evaluates the influence of post-processing as described in [1] and replaces their Bahdanau style attention with a Luong attention mechanism. Furthermore, it contributes to a better understanding of the principle of function by comparing performance across datasets and exploring different amounts of features.

The overall goal is not to create a production ready system, but to explore and investigate neural voice synthesis. As described in section 1.3.2 there is a common notion on what qualities are expected from a speech synthesis system. For once it has to clearly get across the message, and secondly do this using a human-like, natural voice.

The primary contributions are as follows:

Implementation of a neural speech synthesis system that:

- can synthesize speech for a single speaker.
- can be trained using freely available corpora to maximize verifiability.
- can be trained from unaligned text-audio-pairs only in an end-to-end fashion, were the text is composed of separated sentences.
- can be trained without the need for already existing helper models.

Evaluation of:

- the naturalness of the produced speech compared to two other TTS systems using a MOS survey and paired difference hypothesis tests.
- the influence of different model components.

- the performance on corpora with varying quality of the data.
- the effects of different amounts of target features.

The following configurations are explicitly not investigated:

- Active control of prosody using a markup language.
- Synthesis of speech for multiple speakers using a single model.

1.2 Thesis Structure

This thesis is organized as follows:

- Section 2 provides a basic introduction to NN fundamentals like Recurrent Neural Networks (RNNs), seq2seq models, dropout and batch normalization.
- Section 3 gives an overview over previous work in the area of speech synthesis; especially neural TTS approaches.
- Section 4 presents the abstract building blocks the model is composed of, and establishes a fundamental grasp of *Luong*-style attention and the *Griffin-Lim* algorithm.
- Section 5 discusses the model architecture and its abstract processing stages. It reviews the datasets and gives details on the implementation and the training process.
- Section 6 presents the resulting models, the MOS test survey and visualizes the effects of individual components. Each of the findings is discussed separately.
- Section 7 briefly summarizes the main contributions, discusses the limitations and offers future research suggestions.
- Appendix A contains the design and realization of the evaluation survey.
- Appendix B gives detailed statistics on the corpora used.

1.3 Text-to-Speech Synthesis

The generation of synthetic voices using computers is a topic researchers investigate since decades [5]. TTS synthesis is a subdomain of speech synthesis that aims to generate speech from written text.

There are numerous different models found in TTS systems. According to Taylor they can be differentiated into various forms including *Common-form*, *Signal-to-Signal*, *Pipelined* or *Grapheme and phoneme form* models [3, pp. 38–40], just to name a few. However, as pointed out by Taylor, these models are not mutually exclusive and many production systems are a combination of two or more models [3, pp. 40–41]. Since they all share the same fundamentals, the following explanation in section 1.3.4 uses a common pipelined model for exemplification.

Spoken language has a dual structure of sentences made up of words, and words consisting of phonemes. Phonemes are the principal unit of form for verbal content, i.e. spoken sound. They have no direct meaning, they are the basic units or building blocks used to form spoken words. These words in turn are further combined into spoken sentences [3,

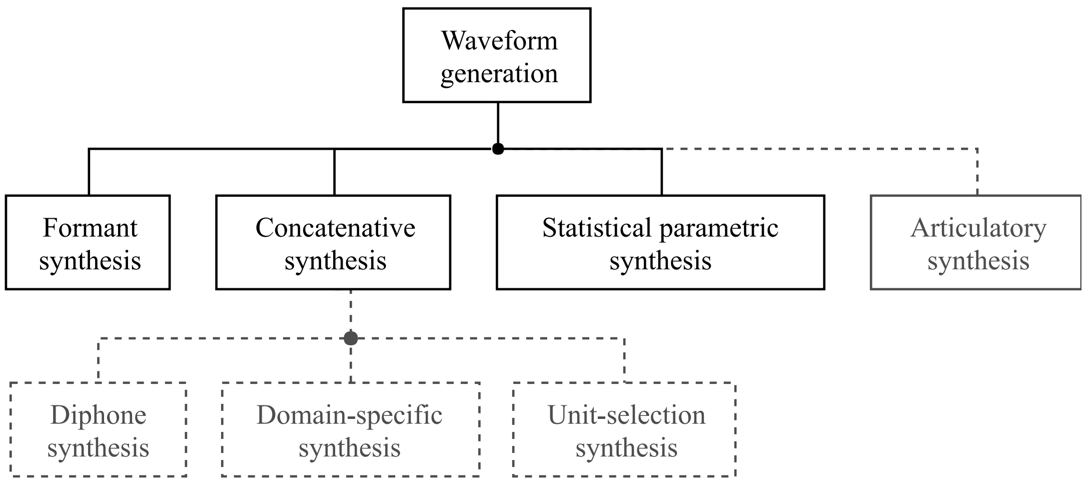


Figure 2: Overview of the primary waveform synthesis technologies. [4] The three primary methods most commonly found [4] are shown with solid lines. For completeness the rare articulatory synthesis method is shown grayed out (solid lines) as well. Indicated with dashed lines are the sub-categories for the concatenative synthesis branch.

pp. 14 sq.]. There exists a similar dual structure for written text, were sentences likewise are made from words. These words are, however, constructed using graphemes not phonemes. Graphemes are similar to phonemes but are much more language-dependent. [3, pp. 28 sq.] Taylor describes them as follows:

> In alphabetic languages like English and French, graphemes can be thought of as letters or characters [...]; in syllabic writing like Japanese hira-gana, each grapheme represents a syllable or mora, and in languages like Chinese each grapheme represents a full word or part of a word. [3, p. 28]

This form of decomposition is not specific to TTS solutions, but is used in many applications within the field of NLP.

1.3.1 Methodology

There are three primary synthesis technologies commonly encountered [4]. These are namely *formant synthesis*, *concatenative synthesis* and *statistical parametric synthesis*. Figure 2 gives an overview over the different technologies. For completeness a fourth method called articulatory synthesis is also listed. This distinction is somewhat detached from the actual implementation; may it be Decision Trees (DTs) [20], Hidden Markov Models (HMMs) or NNs for example.

Formant synthesis tries to excite a set of resonators using a signal source like a noise generator, forming the desired speech spectrum using filters [4]. Statistical parametric models on the other hand usually extract parameters from speech and model these using statistics. The actual speech is later constructed given the parametric representation of a target speech sequence. Moreover, there exist concatenative systems. These systems generate speech by selecting small sections of recorded voice and stitch them together to form new speech. The synthesis quality is, therefore, directly related to the quality of the recordings available. Articulatory synthesis models all the components of the human vocal tract and their interaction with a simulated airflow. Note that this distinction is not as rigorous in reality and that numerous mixed and hybrid solutions exist.

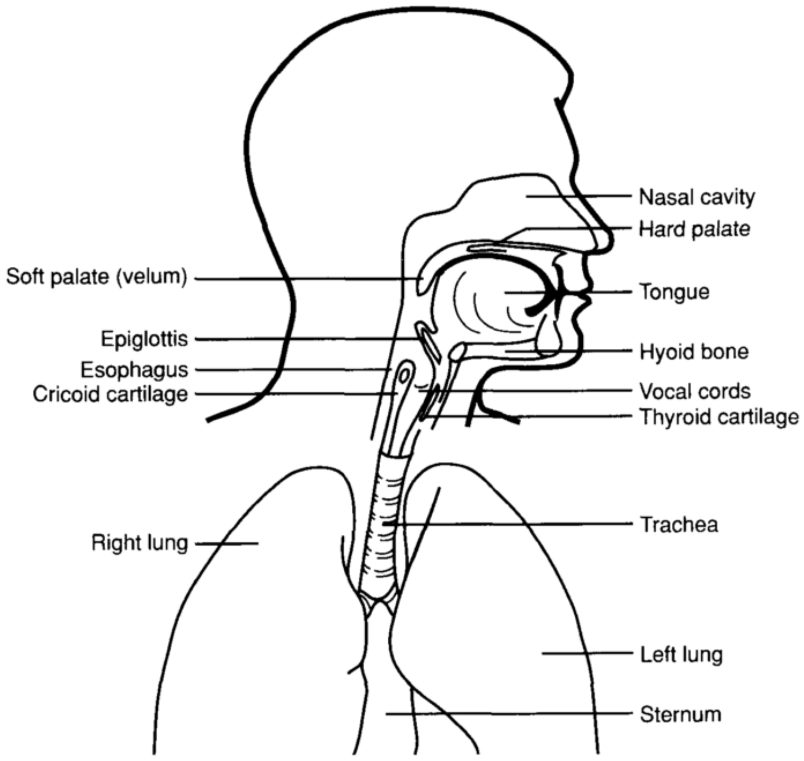


Figure 3: Visualization of the human vocal mechanism [5]. Air from the lungs flows through the glottis located between the vocal folds. As a result of air flow and muscle tension the vocal folds vibrate and produce sound.

1.3.2 Objectives

There is a common notion on what qualities are expected from a speech synthesis system. A synthesis system is expected to clearly get across the message, and secondly achieve this using a human-like voice. [3, p. 3]

This two main priorities are referred to as intelligibility and naturalness [4, 3, pp. 3, 47–49, 21]. Here, natural means that the system should sound just like a human. Intelligibility on the other hand will be defined as the listeners' ability to properly decode and hence understand the message from the produced speech.

1.3.3 Physical Nature

Speech is a continuous, acoustic waveform humans produce by using their vocal organs. A schematic diagram of the human vocal mechanism is shown in figure 3. Creating speech like that requires a significant amount of intricate timing and motoric control. O'Malley summarizes this process as follows:

> Air pressure developed in the lungs flows through the trachea and the vocal folds in the larynx. This opening between the vocal folds is called the glottis, and the air flow as a function of time is called the glottal waveform. For voiced sounds, the correct combination of air pressure and muscle tension causes the vocal folds to vibrate, generating a series of pulses of air. [. . .] The noise generated at the glottis or elsewhere in the vocal tract is modified as it passes through the oral and nasal cavities and radiates from the head as a speech waveform. [5]

1.3.4 System Organization

As mentioned above, there exists a multitude of different methods used to realize TTS systems. The actual setup is highly dependent on the system build. However, in many systems, one often finds a list of modules arranged in a pipeline. Such pipelines might include modules for pre-processing, tokenization, text-normalization, part-of-speech tagging, parsing, morphological analysis, lexicon, post-lexical rules, intonation, phrasing, duration, fundamental frequency generation, unit selection and signal processing [3, p. 41]. The process is seen as one of passing representations from one module to the next as needed.

The following explanation confines the set of stages mentioned above to the ones that are most essential and most likely to be encountered. Namely, the pre-processing, duration estimation and the prosody generation. Figure 4 depicts a functional diagram of a general TTS system. The dotted stages are omitted and only the grayed out stages are discussed hereafter.

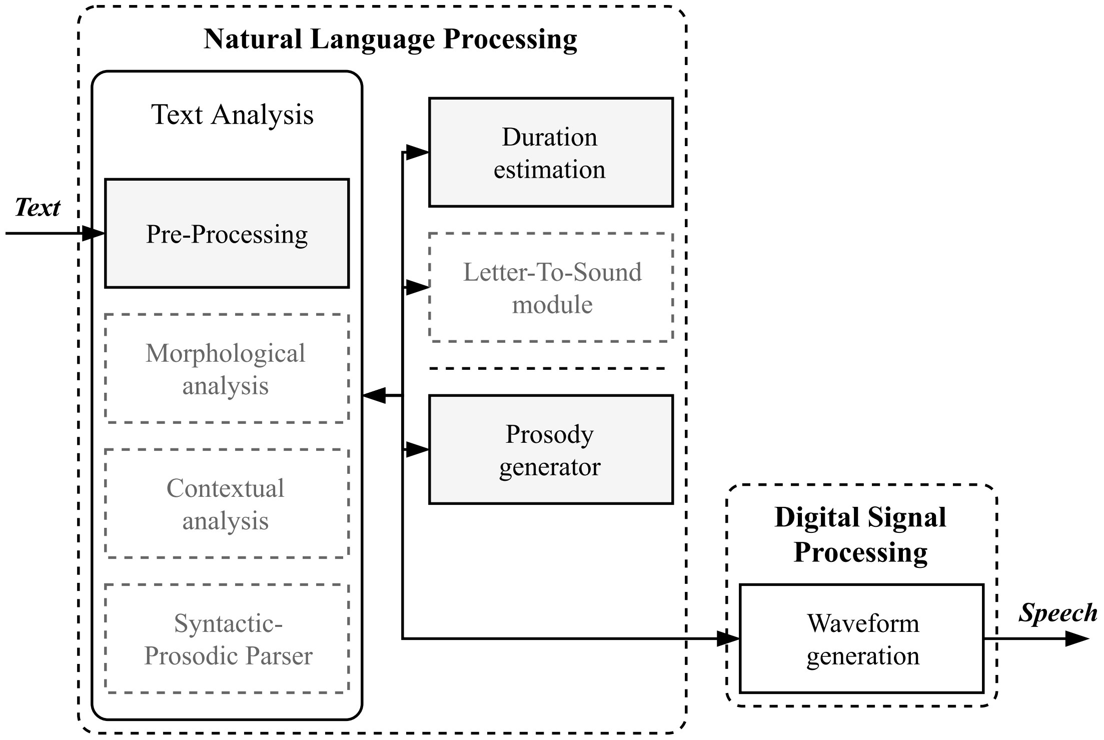


Figure 4: Schematic layout of a conventional Text-to-Speech system, derived from [22]. On the highest level it is usually decomposed into a NLP component generating parameters and a Digital Signal Processing (DSP) part synthesizing speech from it. The dotted parts are omitted in this explanation and only the grayed out parts are discussed.

On the highest level the system is split into a NLP and a DSP component. The NLP part is responsible for reading a text and transforming it into a sequence of parameters suitable for synthesis in the DSP component. Frequently in the form of a phonetic transcription augmented with prosodic information. An integral part of the NLP component is the text analysis stage. It makes a distinction between written text (i.e. in the form of characters) and its linguistic form (e.g. in the form of graphemes). The linguistic form can be seen as clean, abstract and unambiguous, and the written form as a noisy signal that has been encoded from this form [3, p. 26]. The purpose of the text analysis phase is to extract this linguistic form.

The parameters produced in the NLP component are used as input for the DSP phase. This signal processing phase transforms the parameters into an actual waveform containing speech.

Listed below are the procedures to be discussed:

Pre-Processing:
Writing as it occurs in the real word is ambiguous and noisy and requires cleanup. Text contains several non-standard words such as numbers, abbreviations, homographs and symbols built using punctuation characters such as exclamation point '!' or smileys ':-)'. [4]

During pre-processing the text is, therefore, generally tokenized (i.e. dissected into tokens such as words) and normalized. Normalization spells out numbers that were written using digits or replaces abbreviations with their written out form.

Duration:
Aspects of intonation and phrasing are manifested in differences in the timing of speech. The task of duration estimation is to assign a temporal structure to utterances. [3, pp. 257 sq.] This timing is most commonly formulated in terms of unit durations. From experimental phonetics it is known that the temporal properties of speech are complex. When talking more quickly every part an utterance contracts at a different rate, not by a constant factor. For example, emphasis generally lengthens sections of speech in particular ways. [3, pp. 257 sq.] Possible solutions for the generation of duration one might come across are, deterministic rule based approaches like developed by Klatt [23, 3, pp. 259 sq.], or statistical models like used in [24].

Prosody:
In linguistics, prosody is commonly referred to as the rhythm, stress and intonation of speech. The prosody of oral languages involves variation in syllable length, loudness, pitch and the characteristic frequencies of speech sounds. [4]

It is vital to create natural sounding speech and a key component to distinguish between linguistic constructs like statements and questions.

However, prosodic information cannot be decoded from the text itself, therefore, everything is spoken with neutral prosody. There is certainly a limited prosodic component in written language. Punctuation can be used to show structure; and underlined, italic or bold text is used to show emphasis. Affective prosody on the other hand is nearly entirely absent; there is no conventional way to show anger, pain or sarcasm. [3, p. 31]

2 Background

Subsequently, the following notation is used. Bold lower-case symbols (e.g. $\mathbf{x}_i$, $\mathbf{b}$) denote vectors and bold upper-case symbols (e.g. $\mathbf{W}_i$, $\mathbf{U}$) denote matrices. Italic characters (e.g. m, n, M) represent scalars and cursive upper-case characters (e.g. $\mathcal{C}$, $\mathcal{X}$) specify sets and sequences. For simplicity $\mathbf{0}$, $\mathbf{1}$ are defined to be vectors were each element equals 0 and 1, respectively. Furthermore, σ denotes the logistic sigmoid function[2]. It is assumed that applying a function that operates on a scalar (e.g σ or tanh) to a matrix or vector, it is applied to each element implicitly.

For graphical illustrations the following labeling and colors are used. Inputs are marked orange, outputs are green and intermediate states are colored violet. Special inputs or outputs are tagged in red. Simple operations (e.g activation functions) are illustrated as gray blocks with sharp edges. More complex building blocks composed of simple operations are depicted as blue blocks with rounded corners. There are two exceptions to that. The conventional matrix addition is denoted as $\oplus$. Further, the concatenation operation $\oslash$ is defined as shown in equation (1).

$$\begin{gathered}\oslash : \mathbb{R}^M \times \mathbb{R}^N \to \mathbb{R}^{M+N} \\ \left(\begin{bmatrix} a_1 & \dots & a_m \end{bmatrix}, \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix}\right) \mapsto \begin{bmatrix} a_1 & \dots & a_m & b_1 & \dots & b_n \end{bmatrix}\end{gathered} \tag{1}$$

2.1 Neural Networks

Due to this section covering common fundamentals of neural networks there exists a considerable overlap with other work.[3]

2.1.1 Feed Forward Neural Network

A feedforward neural network consists of L layers where the l-th layer ($l \in \{1, 2, \dots, L\}$) applies a non-linear transformation H on its inputs. H is an affine transformation parameterized by a weight matrix $\mathbf{W}_H^{(l)}$ and a bias vector $\mathbf{b}_H^{(l)}$ followed by an activation function f. Note that the activation function is expected to be differentiable in order for backpropagation to work. Each layer receives an input vector $\mathbf{x}^{(l)}$ and produces and output vector $\mathbf{y}^{(l)}$ as seen in equation (2). This is also known as a *dense* or *fully connected* layer.

$$\begin{aligned}\mathbf{y}^{(l)} &= f\left(H(\mathbf{x}^{(l)}, \mathbf{W}_H^{(l)}, \mathbf{b}_H^{(l)})\right) \\ &= f\left(\mathbf{W}_H^{(l)}\mathbf{x}^{(l)} + \mathbf{b}_H^{(l)}\right)\end{aligned} \tag{2}$$

2.1.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) [26, pp. 326 sqq.] apply a set of discrete convolution operations whereby they are able to learn the window functions the input is convolved with. Subsequently, these window functions are referred to as *kernels*.

At first let's take a closer look at the discrete convolution operation. See figure 5 for an illustration of a discrete convolution. Hereafter the two-dimensional case is considered.

[2] *sigmoid* function; $\sigma(x) = (1 + \exp(-x))^{-1} \quad x \in \mathbb{R}$

[3] The contents of this section are in part created in consultation with Dangschat [25] who is creating a ASR system at the time.

However, convolutions can be applied in any dimensionality. Let $\mathbf{I} \in \mathbb{R}^{A \times B}$ be the input and $\mathbf{K} \in \mathbb{R}^{M \times N}$ denote the kernel to convolve with. $\mathbf{K}$ is assumed to be a valid probability density function[4]. Then the convolution of $\mathbf{I}$ and $\mathbf{K}$ is written as shown in equation (3), where $*$ denotes the convolution operation. [26, pp. 329 sqq.]

$$\mathbf{C}(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{I}(i+m, j+n)\mathbf{K}(m,n) \quad 1 \leq i \leq A,\ 1 \leq j \leq B \tag{3}$$

The amount at which i and j are incremented is referred to as the *hop* or *stride*. Assuming a stride of 1 for any dimension, the operation results in a matrix $\mathbf{C} \in \mathbb{R}^{(A-M+1)\times(B-N+1)}$.

Using convolutions in NNs has several benefits. The kernel can be applied regardless of the input size, allowing the network to process variable length inputs. Furthermore, CNNs require substantially fewer parameters, in turn reducing the memory requirements and computational cost. Traditional feed forward NNs have a dense set of parameters describing the interaction of each input unit with each output unit (see section 2.1.1). Contrary, the weights of each kernel are effectively shared since multiple input units are processed using the same kernel. Another benefit of convolutions is, that they are invariant to translational changes in the input. [26, pp. 329 sqq.]

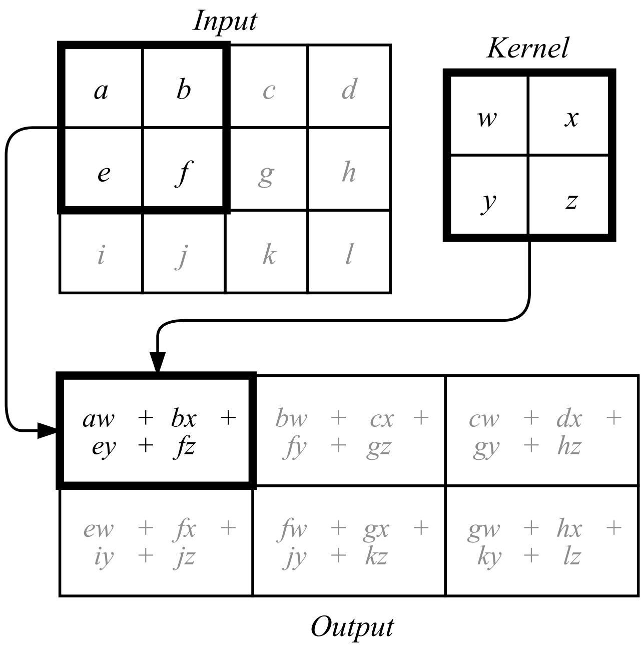


Figure 5: Illustration of a discrete 2-D convolution between an input of size 3×4 and a 2×2 kernel using stride 1 in any direction. The figure is derived from [26, pp. 330 sqq.].

2.1.3 Gated Recurrent Unit

A gated recurrent unit (GRU) is a type of recurrent network cell introduced by Cho et al. It is motivated by the long short-term memory (LSTM) cell [28] but has fewer computational demands. LSTM cells usually have a memory cell and four gating units that adaptively control the information flow inside the unit. GRU cells on the other hand have only two gating units. [27]

[4]Fulfilling: $\sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{K}(m,n) = 1 \ \wedge\ \mathbf{K}(m,n) \geq 0\ \forall m,n$

Note that lots of variations of both LSTM and GRU cells exist.

At each step t a GRU cell produces a hidden state vector $\mathbf{h}_t$. It is based on the cell's input vector denoted by $\mathbf{x}_t$ and the last hidden state $\mathbf{h}_{t-1}$. The current input and the last hidden state are combined into the new state $\mathbf{h}_t$ using the *reset gate* $\mathbf{r}_t$ and the *update gate* $\mathbf{z}_t$. Equation (4) describes the computation of the hidden state vector $\mathbf{h}_t$, were $\circ$ denotes the *Hadamard Product*[5]. The weight matrices $\mathbf{W}, \mathbf{U}$ as well as the bias vectors $\mathbf{b}$ are learned during training. The update gate controls how much information from the previous state $\mathbf{h}_{t-1}$ is carried over to the new hidden state. The reset gate drops information from the last state $\mathbf{h}_{t-1}$ before calculating the new state. Note that GRU cells only produce a hidden state $\mathbf{h}_t$ at each step t.

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z\right)\\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r\right)\\
\tilde{\mathbf{h}}_t &= \tanh\left(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h\left(\mathbf{r}_t \circ \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right)\\
\mathbf{h}_t &= \mathbf{z}_t \circ \mathbf{h}_{t-1} + \left(\vec{\mathbf{1}} - \mathbf{z}_t\right) \circ \tilde{\mathbf{h}}_t
\end{aligned} \tag{4}$$

Using the gating mechanism the cell can seamlessly vary its behavior. Let's assume the reset gate $\mathbf{r}_t$ is close to $\mathbf{0}$, then the new hidden state $\mathbf{h}_t$ ignores the last state $\mathbf{h}_{t-1}$ and is primarily influenced by the current input $\mathbf{x}_t$. Thus dropping previously captured information. Following, the update gate controls how much the previous state influences the new hidden state.

2.1.4 Bidirectional Gated Recurrent Unit

Bidirectional GRUs are designed to be able to simultaneously access information from past and future states. To achieve this two separate GRU cells are combined such that one moves forward through time t and one moves backward through time. Note that this is not limited to GRUs but rather arbitrary types of RNNs can be combined that way.

A GRU produces a sequence of forward hidden state vectors $\overrightarrow{\mathbf{h}}_t$ from the input $\mathbf{x}_t \in (\mathbf{x}_1, \ldots, \mathbf{x}_S)$ of length S. Another GRU produces a sequence of backward hidden state vectors $\overleftarrow{\mathbf{h}}_t$. For the backward direction the input is fed in reverse $(\mathbf{x}_S, \ldots, \mathbf{x}_1)$. This implies that the entire input sequence has to be available before it can be fed. The bidirectional RNN hidden state $\mathbf{h}_t$ at step t is then computed by concatenating the forward and backward hidden states as shown in equation (5).

$$\mathbf{h}_t = \overrightarrow{\mathbf{h}}_t \oslash \overleftarrow{\mathbf{h}}_t = \left[\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t\right] \tag{5}$$

2.1.5 Sequence to Sequence Architectures

Sequence to sequence (seq2seq) models are a type of recurrent neural architecture with the purpose of transforming sequences. Presented a source sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_S)$ of length S a seq2seq model can construct a target sequence $(\mathbf{y}_1, \ldots, \mathbf{y}_T)$ of length T. See figure 6 for an illustration of a seq2seq architecture.

[5]Hadamard Product: $\mathbf{A} \circ \mathbf{B} = (a_{ij}b_{ij}) \quad \text{with} \quad \mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n} \quad \text{and} \quad 1 \le i \le m,\; 1 \le j \le n$

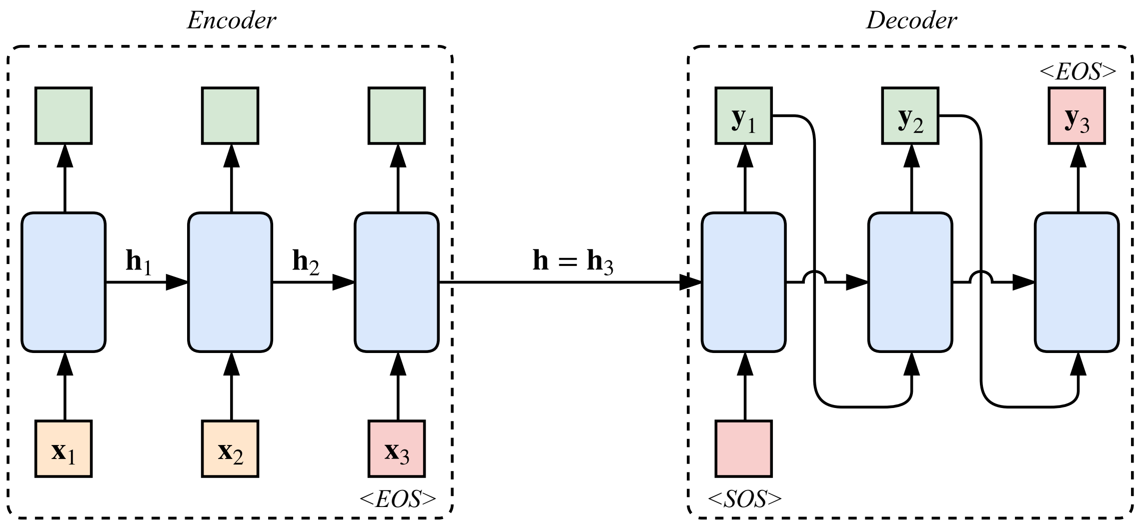


Figure 6: Exemplary layout of a generic sequence-to-sequence architecture. Here a recurrent encoder compresses a source sequence $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ into a fixed-dimensional representation $\mathbf{h}$. A recurrent decoder decompresses this representation into a target sequence $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$.

A generic seq2seq architecture consists of an *encoder* and a *decoder* stage [9]. The idea being that the encoder computes a fixed-dimensional representation $\mathbf{h}$ of the input sequence. The decoder decompresses this fixed-dimensional representation into the target sequence. Typically, the encoder processes inputs until it encounters an End of sequence (EOS) symbol, producing a sequence of encoder hidden states $(\mathbf{h}_1, \ldots, \mathbf{h}_S)$. The EOS symbol marks the end of the encoding sequence and causes the encoder to provide its last hidden state $\mathbf{h}_S$ as the fixed-dimensional representation $\mathbf{h}$ of the input. During decoding the fixed-size representation is used to initialize the first hidden state of the decoder RNN. Decoding then starts with a Start of sequence (SOS) symbol, generating one element of the target sequence with each decoding step. At step t, to produce the output $\mathbf{y}_t$ the last output $\mathbf{y}_{t-1}$ is usually fed as input.

In practice, this seq2seq approach has some restrictions especially visible with increasing sequence lengths. Modeling long sequences with a fixed-dimensional representation may require having hidden states with excessive dimensionality, slowing down computation. Furthermore, information contained early in the input sequence needs to be carried through all encoder steps into the fixed-dimensional state [9].

These restrictions can be eliminated using attention mechanisms. While in general seq2seq architectures do not depend on attention they are yet frequently combined. For specific details on the integration and implementation of attention refer to section 4.3.

2.2 Batch Normalization

During training of NNs the distributions of each layer's input changes, as the parameters of the previous layers change. This complicates the training process and leads to problems like the exploding gradient problem or the vanishing gradient problem. While especially the former one is often counteracted using variations of the Rectified Linear Units (ReLU) activation function [29] using careful initialization, Batch Normalization (BN) [30] can also be used. BN tries to ensure that the distribution of nonlinearity inputs remains more stable as the network trains. Allowing the usage of much higher learning rates to improve the training speed. [30]

Consider a mini-batch $\mathcal{B} = \{x_1, \ldots, x_m\}$ of size m with the input being scalars $x_i \in \mathbb{R}$ over the mini-batch. Let $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$ denote the normalized mini-batch with $\gamma, \beta \in \mathbb{R}$ being the parameters of BN to be learned. For intelligibility this example only uses scalar values. In reality mini-batches and hence the inputs can be of arbitrary dimensionality. The normalization process is split in two steps. During training first each scalar feature is independently normalized to have a mean of zero and a variance of 1 for each training mini-batch (equation (6)).

$$\widehat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{6}$$

Were $\mu_{\mathcal{B}}$ is the batch mean (equation (7)) and $\sigma_{\mathcal{B}}^2$ is the batch variance (equation (8)). Note that ϵ is a small value added for numerical stability.

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{7}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{8}$$

Finally, in a second step two learned scalar parameters γ and β are used for scaling and shifting (equation (9)); giving the batch normalized outputs y_i.

$$y_i = \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \tag{9}$$

However, during inference it is not desirable to use the above equations to perform BN. Using them would result in nondeterministic outputs depending on the other elements contained in a specific batch. Instead of normalizing over mini-batches like done during training time, during inference the whole population is used. For that purpose moving averages of $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are cumulated during training and are used instead during inference.

2.3 Dropout

Neural networks with many layers are capable machine learning systems. However, with increasing network depth and a growing amount of parameters networks become prone to overfitting. Dropout [31] is an approach to counteract overfitting of NNs.

Dropout is designed to resemble the combination of different models into *ensembles* otherwise used to reduce overfitting and improve results. Ensembles combine many separately trained networks by averaging their outputs. While this may improve the results it is also impractical and not always feasible. Instead, dropout alters the network during training, to simulate the training of an ensemble. The main principle is to randomly drop network units during training; preventing them from co-adapting too much. Here, dropping a unit refers to temporarily removing the unit and all its incoming and outgoing connections; thus forming a new NN. In its most simple form each unit is retained with a fixed probability $p \in \mathbb{R}$ with $0 < p < 1$. [31]

Removing units from the network is not desirable during inference. Therefore, when not training, all units are included in the model whereby each units weight is scaled by its probability of retainment p. Doing so effectively approximates the expected value of the units output. [31, pp. 1931, 1933 sq.]

2.4 Character Embeddings

Embeddings are numeric representations (usually a high-dimensional vector) for capturing the content of a sequence (e.g sentence) or a single element (e.g word, character). Constructing character embeddings refers to the process of transferring each character of an alphabet into a vector representation. Representations are intended to be an abstract representation of each character. The advantage is that they can be deducted from data. The following description is in part derived from [12].

Let $\mathcal{C}$ be the vocabulary of characters. Then let d be the character embeddings dimensionality and $\mathbf{Q} \in \mathbb{R}^{d \times |\mathcal{C}|}$ be the embedding lookup matrix. Where each column vector of $\mathbf{Q}$ represents an embedding for a character of the vocabulary. Note that the vocabulary often also contains "virtual" characters. These may be introduced to mark the start or end of a sequence. Furthermore, let $\mathcal{S}$ denote a character sequence $(c_1, c_2, \ldots, c_l)$, of length l constructed from the vocabulary $\mathcal{C}$. For each character of $\mathcal{S}$ the corresponding character embedding (i.e. column vector) is selected from $\mathbf{Q}$, resulting in the embedded matrix $\mathbf{C}^{\mathcal{S}} \in \mathbb{R}^{d \times l}$ for the sequence $\mathcal{S}$. See figure 7 for an illustration of the lookup process.

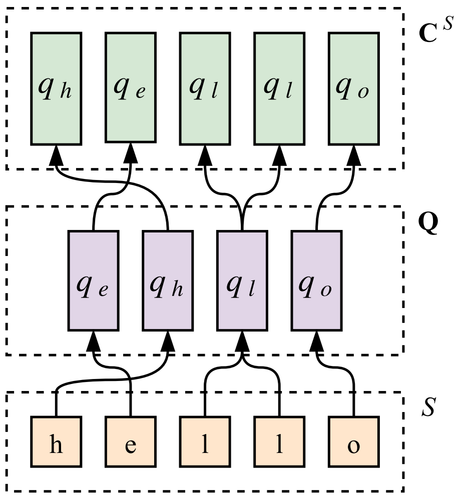


Figure 7: Visualization of the character embedding lookup process. A character sequence $\mathcal{S}$ is transformed into an embedded character sequence $\mathbf{C}^{\mathcal{S}}$. This is done by looking up embedding vectors for each character in the character embedding lookup matrix $\mathbf{Q}$.

2.5 Gradient Clipping

Gradient clipping is a technique to deal with the exploding gradient problem by modifying the gradients of a NN. When a network experiences a large increase in the norm of the gradients during training this is referred to as the exploding gradient problem. As described by Pascanu et al. dealing with such an increase in gradient norm is commonly done by rescaling the gradients as their norm exceeds a threshold. [2]

This procedure can be carried out on different sets of gradients and using arbitrary norms. Two approaches that might be encountered are clipping based on the gradient norm of individual layers or clipping by the global norm of all gradients in the network.

Hereafter the global approach is explained using the Frobenius norm.

Let $\mathcal{G} = \{\mathbf{G}_1, ..., \mathbf{G}_x\}$ denote gradient matrices calculated using any form of classical backpropagation [26, pp. 200 sqq.] for a NN. Furthermore, let $\tau \in \mathbb{R}$ denote the clipping threshold and $\lambda \in \mathbb{R}$ (equation (10)) be the global Frobenius norm of all gradients in $\mathcal{G}$. The clipped gradients $\widehat{\mathbf{G}}_i$ are then calculated as shown in equation (11).

$$\lambda = \sqrt{\sum_{i=1}^{x} ||\mathbf{G}_i||_F^2} \tag{10}$$

$$\widehat{\mathbf{G}}_i = \frac{\mathbf{G}_i * \tau}{\max(\lambda, \tau)}, \quad 1 \leq i \leq x \tag{11}$$

Note that if $\tau > \lambda$, the gradients remain the same. However, if the global norm λ exceeds the clipping threshold τ the gradients are scaled down in ratio to the global norm. While this approach helps to prevent the exploding gradient problem, there are two points to consider. First, it introduces the clipping threshold τ as an additional hyper-parameter. Second, the normalization process requires all dependent gradients of the network to be computed before clipping can be executed.

3 Related Work

There exist various approaches on how TTS can be implemented. Today one might roughly differentiate them into classes like statistical parametric [32, 24], unit-selection or the rarely seen formant synthesis [33]. Whereas numerous mixed and hybrid solutions are also in existence. See section 1.3 for more information. The following listing mainly focuses on the work of recent years and refers only to older systems for completeness.

The statistical parametric models are referred to as statistical since they usually extract parameters from speech and model them using statistics. The actual speech is later constructed given the parametric sequence. Generative HMM based models have seen a lot of use in this area [24].

Beside the statistical approaches, in the past unit-selection systems [34] were the dominating approach for over a decade [24]. According to [3, p. 522] in 2007, unit-selection was still rated the highest quality synthesis technique. These systems generate speech by selecting small sections of recorded voice and stitch them together to form new speech. The synthesis quality is, therefore, directly related to the quality of the recordings available. With enough high quality recordings the problem can be reduced to automatically finding good patches of audio in a database. However, recording large datasets is a costly and elaborate task [35].

Recently NN based approaches dominate the best performing models. Among them auto-regressive models like *WaveNet* [36], that started delivering human level audio quality. During 5-scale MOS[6] tests, measuring speech naturalness, *WaveNet* scored ∼4.21 compared to ∼4.55 measured for natural human speech [36]. *WaveNet* directly models the audio samples of a waveform by probabilistically regressing new samples based on all previous ones in a time series. While auto-regressive models tend to generate high quality results their generation speed is slow because they process individual samples.

With the *Deep Voice* architecture [37] researchers replaced all components of the pipeline (see section 1.3) with NNs. In addition, *Char2Wav* [21] demonstrates that the number of pipeline stages can be condensed down significantly. *Char2Wav* uses RNNs paired with attention [11]. They employ a seq2seq attention encoder-decoder architecture [9] to predict parameters for the *WORLD* vocoder [38]. They combine this with a neural vocoder model based on *SampleRNN* [19] for waveform generation. This results in only two stages they need to pre-train separately. The model employs traditional vocoder parameters like the spectral envelope, and aperiodic parameters. It requires separate pre-training prior to end-to-end model fine-tuning.

With the publication of *Tacotron* [1] significant improvements in combining nearly all stages of the pipeline into a single model were made. As a consequence, reducing the overall complexity of the system. Like *Char2Wav*, *Tacotron* utilizes a seq2seq attention encoder-decoder architecture. *Tacotron* produces magnitude spectrograms from text on a frame-level. It can be trained in an end-to-end fashion without having to model intermediary processing stages. This removes the need for pre-existing aligner models like required by *Char2Wav* or *Deep Voice*.

[6]Higher is better, with 5 being the best possible score.

The later models *Tacotron 2* [39] and *Deep Voice 2* [40] are heavily influenced by *Tacotron*. *Deep Voice 2* introduces low-dimensional speaker embeddings to the architecture in order to model multiple speakers. *Tacotron 2* streamlines the *Tacotron* architecture and replaces the *Griffin-Lim* vocoder with a neural vocoder based on *WaveNet*. *Deep Voice 1* and *Deep Voice 2* still retain some traditional pipeline stages like grapheme-to-phoneme conversion, duration and frequency prediction. However, their successor *Deep Voice 3* [41] implicitly learns such conversions on the fly. Similar to *Tacotron* and *Char2Wav* their architecture employs an attention-based seq2seq approach. But, *Deep Voice 3* avoids RNNs and employs a fully-convolutional version in order to speed up training and create a production ready system.

4 Model Components

This section introduces the fundamental components needed for construction of the model proposed in section 5. Note that the mathematical notation and the graphical elements used hereafter are identical to that defined in section 2.

4.1 Highway Network

Accomplishments achieved by neural networks are often reached by stacking of successive network layer to great depths[7]. However, with an increasing number of layers networks become more and more difficult to train. Highway networks are a modification to conventional feed forward networks that allow training with great depths that were nearly impossible before. [42]

Highway networks introduce gating units to each layer of the network. These gates are able to regulate the information flow, allowing unimpeded information flow across several layers on so called *information highways.* The gating behavior is learned and enables the highway networks to skip different processing steps hence rerouting the data flow around layers as needed. This strategy is related to the regulation of information flow in GRU RNNs [27]. GRUs use reset and update gates to control which information to throw away and which values to update.

Let's recall from section 2.1.1 how plain feedforward fully connected neural networks work. Such a network consists of L layers where the l-th layer ($l \in \{1, 2, ..., L\}$) applies a non-linear transformation H on its inputs. H is an affine transformation parameterized by weights $\mathbf{W}_H^{(l)}$ and a bias $\mathbf{b}_H^{(l)}$ followed by a non-linear activation function. Each layer receives an input vector $\mathbf{x}^{(l)}$ and produces an output vector $\mathbf{y}^{(l)}$. For reasons of readability and comprehensibility the layer index l is dropped hereafter. Furthermore, the bias terms of the non-linear activation functions are omitted for brevity.

As shown in equation (12) each highway network layer adds two additional non-linear transformations $T(\mathbf{x}, \mathbf{W}_T)$ and $C(\mathbf{x}, \mathbf{W}_C)$. Were the weight matrices $\mathbf{W}_T$, $\mathbf{W}_C$ as well as the bias vectors are learned during training.

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H)T(\mathbf{x}) + \mathbf{x}C(\mathbf{x}) \tag{12}$$

T is referred to as the *transform* gate and C as the *carry* gate since they express how much of the layers output $\mathbf{y}$ is produced by transforming and carrying its input $\mathbf{x}$. The implementation done in this work follows [42] and defines $T(\mathbf{x}) = \sigma(\mathbf{W}_T\mathbf{x} + \mathbf{b}_T)$ were σ denotes the sigmoid function. For simplicity the carry gate is defined as $C(\mathbf{x}) = \mathbf{1} - T(\mathbf{x})$ and each layer is activated using the ReLU[8] activation function. The layers output is implemented as shown in equation (13).

$$\mathbf{y} = \mathrm{relu}\left[H(\mathbf{x}, \mathbf{W}_H)T(\mathbf{x}) + \mathbf{x}\,(\mathbf{1} - T(\mathbf{x}))\right] \tag{13}$$

Each highway layer can seamlessly vary its behavior for every element of the input vector as can be seen in equation (14). It can vary between passing inputs to the next

[7]One usually speaks of depth although referring to an increasing amount of layers that are stacked.
[8]*rectified linear unit* function; $\mathrm{relu}(x) = \max(0, x) \quad x \in \mathbb{R}$

layer, therefore representing an identity transformation, or applying a transformation to recreate the behavior of a plain feedforward layer.

$$\mathbf{y} = \begin{cases} \mathbf{x} & \text{if } T(\mathbf{x}) = \mathbf{0} \\ H(\mathbf{x}, \mathbf{W}_H) & \text{if } T(\mathbf{x}) = \mathbf{1} \\ \text{relu}\left[H(\mathbf{x}, \mathbf{W}_H)T(\mathbf{x}) + \mathbf{x}\,(\mathbf{1} - T(\mathbf{x}))\right] & \text{otherwise} \end{cases} \tag{14}$$

To build a highway network an arbitrary number of highway layers is connected in series. However, note that when building such networks the dimensionality of $\mathbf{x}$, $\mathbf{y}$, $H(\mathbf{x})$ as well as $T(\mathbf{x})$ must be the same for all layers.

4.2 CBHG Module

A 1-D convolution bank + highway network + bidirectional GRU (CBHG) module is a combination of multiple NNs, designed with the purpose of extracting representations from sequences [1]. The subsequent explanation is inspired by work in NMT [12] that first introduced this kind of module. Figure 8 shows a schematic overview of the components of a CBHG module and their connections.

Let the input to the module be a sequence of vectors $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_l) \in \mathbb{R}^{d \times l}$ of dimensionality d, were l is the length of the sequence. This input is first processed by a 1-D convolutional filter bank that employs filters of different width as described in section 4.2.1. Its behavior is comparable to extracting N-grams from the sequence. The result is of higher dimensionality than the input and is subsequently compressed down to the same dimensionality as the input using convolutional projections (section 4.2.2).

Finally, the representations are fed into a multi-layer highway network followed by a bidirectional GRU RNN. This RNN is used to extract sequential features from both the forward and the backward direction.

4.2.1 Filter Bank

For a schematic illustration of the filter bank module see figure 9. Let $\mathcal{F} = \{f_1, \dots, f_m\}$ be the filter bank consisting of m sets of 1-D convolutional filters $f_i \in \mathbb{R}^{d \times i \times n}$. The i-th set contains a number of n different filters[9] (i.e. kernels) of width $i \in \{1, 2, \dots, m\}$. Each of the filters can span multiple vectors and is convolved with stride 1 along the last dimension of $\mathcal{X}$ (i.e. its columns). Hence, modeling unigrams, bigrams, up to m-grams. Additionally, the columns of $\mathcal{X}$ are separately padded using $\mathbf{0}$ vectors for each set of filters. They are inserted both at the beginning and end of the sequence, such that the number of columns in the output of all convolutions is the same. This padding strategy is known as *half convolutions* or *same padding.*

For clarification let's consider a single arbitrary filter set $f_w \in \mathbb{R}^{d \times w \times 1}$ with only a single kernel of width w. The same padding strategy applies $\mathbf{0}$ vectors to the beginning and end of $\mathcal{X}$, such that the padded sequence becomes $\overline{\mathcal{X}}_w \in \mathbb{R}^{d \times (l+w-1)}$. Therefore, $w-1$ additional zero vectors are inserted. When applying the convolution the resulting output matrix $\left(\overline{\mathcal{X}}_w * f_w\right) \in \mathbb{R}^{1 \times l}$ has the same number of columns as $\mathcal{X}$; independently of the filter width w.

[9]As demonstrated in [12] one may also use a varying number of filters n_i for each set f_i.

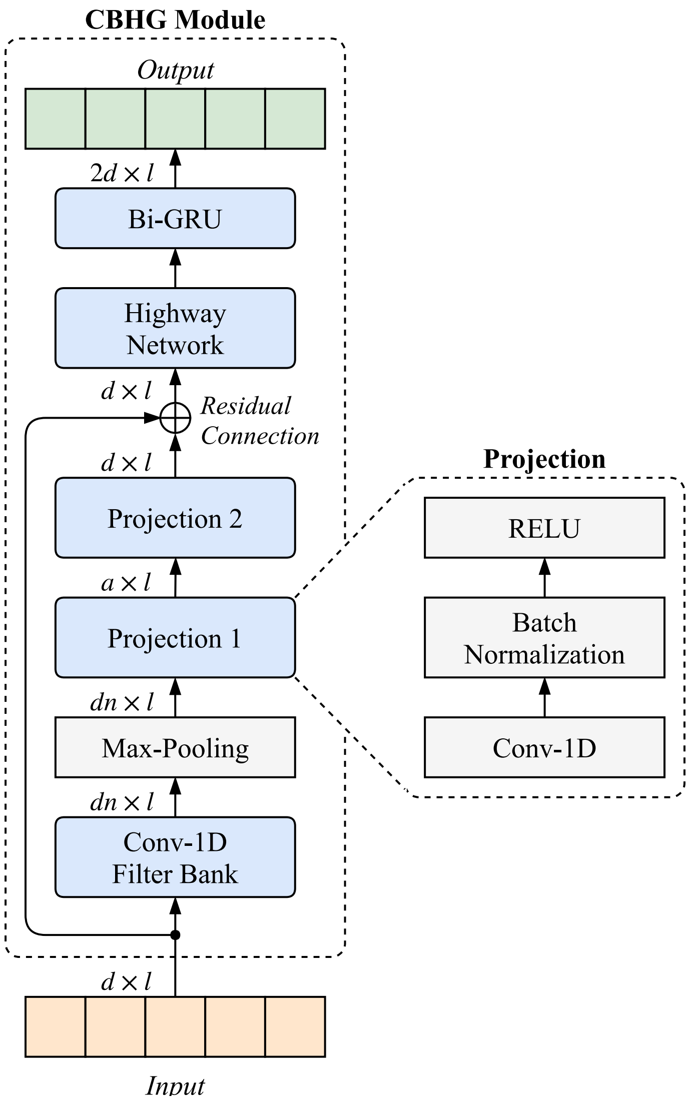


Figure 8: Schematic diagram of a CBHG module. The input sequence is first processed by a 1-D convolutional filter bank. Extracting representations comparable to N-grams. These are further compressed down to the dimensionality of the input sequence using two projections. The output of these projections is added to the modules input using a residual connection. Finally, a multi-layer highway network and a bidirectional GRU produce the output sequence. The dimensionality of the data passed between components is indicated on the left of the arrows.

All filter sets in $\mathcal{F}$ are applied such that $\mathbf{Y}^i = \left(\overline{\mathcal{X}}_i * f_i\right) \in \mathbb{R}^{n\times l}$ with $1 \leq i \leq m$, applies all filters contained in each set. Finally, the output matrices are stacked row-wise as shown in equation (15).

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}^1 \\ \mathbf{Y}^2 \\ \vdots \\ \mathbf{Y}^m \end{bmatrix} \in \mathbb{R}^{mn\times l} \tag{15}$$

Note that Wang et al. state that batch normalization is used for all convolutional layers [1]. However, it is ambiguous if this statement also includes the convolutions within the filter bank. In this work separate batch normalization is applied for each filter set f_i.

4.2.2 Projections

The purpose of this stage is to project the filter bank outputs down to a lower dimensionality. First the filter bank output $\mathbf{Y}$ is max pooled along the columns using a stride of 1 and the same padding scheme that is also used for the filter bank convolutions. Pooling is supposed to increase local invariances while also preserving the original time resolution (i.e. the number of columns) [1].

Subsequently, the max pooled outputs are passed through two 1-D convolutions $p_1 \in \mathbb{R}^{a\times w}$, $p_2 \in \mathbb{R}^{d\times w}$ of fixed width w to reduce the dimensionality. Note that the second set contains d filters, were d is the number of rows of the original CBHG input $\mathcal{X}$. The convolutions use a stride of 1 and employ same padding. Each convolution output is batch normalized, whereby the first layer activates the normalized output using the ReLU activation function.

Applying both projections results in a matrix $\mathbf{P} \in \mathbb{R}^{d\times l}$ that is added to the CBHG modules original input $\mathcal{X} \in \mathbb{R}^{d\times l}$ using a residual connection [43]. Therefore, the matrices are added ($\mathcal{X} \oplus \mathbf{P}$), hence the requirement for having the same dimensionality.

4.3 Luong Attention Mechanism

The proposed model utilizes a seq2seq architecture. As detailed in section 2.1.5 such approaches commonly compress all information into a fixed-dimensional vector. However, the fixed-size compression leads to restrictions as the sequence lengths increase. Here the attention mechanism comes into play to eliminate the restrictions. Rather than presenting a fixed representation of a variable length sequence to the decoder (section 2.1.5), it is presented all information; selecting only the relevant data as needed.

Instead of *Bahdanau* attention [11] as proposed by Tacotron [1] this work implements *Luong* attention [10]. *Luong* attention uses a simplified calculation and hence, can speed up training. Luong et al. introduce two classes of attention mechanism [10]:

(a) A *global* approach which always attends all encoder hidden states.

(b) A *local* one that only considers a subset of encoder states at a time.

Following, this thesis describes only the global approach[10] using GRU RNN cells. Figure 10 illustrates how the seq2seq model is fitted with global Luong attention.

[10]Deployment of the local approach is discontinued for this thesis.

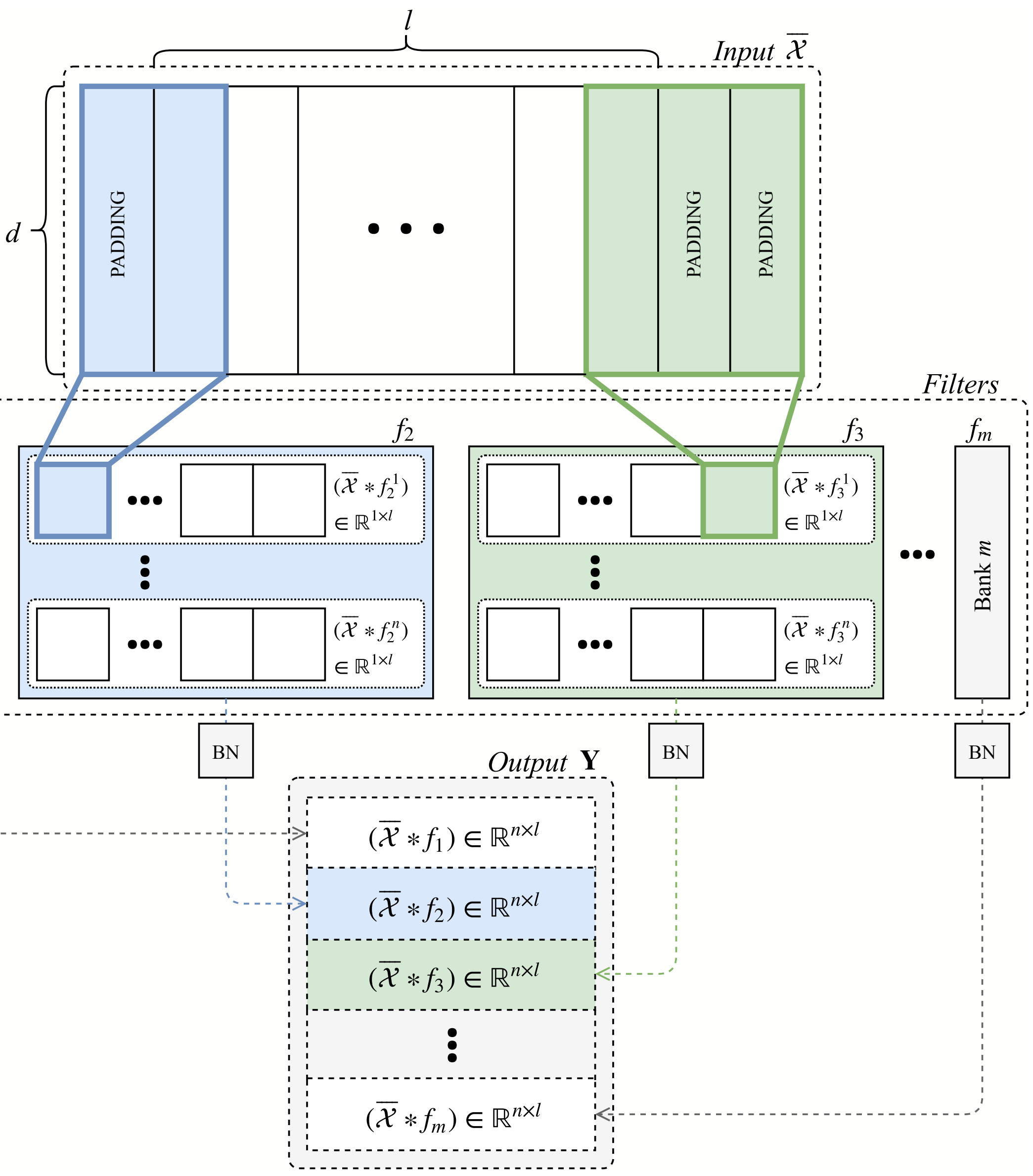

l
Input $\overline{\mathcal{X}}$
d
PADDING
PADDING
PADDING
Filters
f_1
f_2
f_3
f_m
Bank 1
Bank m
$(\overline{\mathcal{X}} * f_2^1) \in \mathbb{R}^{1\times l}$
$(\overline{\mathcal{X}} * f_2^n) \in \mathbb{R}^{1\times l}$
$(\overline{\mathcal{X}} * f_3^1) \in \mathbb{R}^{1\times l}$
$(\overline{\mathcal{X}} * f_3^n) \in \mathbb{R}^{1\times l}$
BN
BN
BN
BN
Output $\mathbf{Y}$
$(\overline{\mathcal{X}} * f_1) \in \mathbb{R}^{n\times l}$
$(\overline{\mathcal{X}} * f_2) \in \mathbb{R}^{n\times l}$
$(\overline{\mathcal{X}} * f_3) \in \mathbb{R}^{n\times l}$
$(\overline{\mathcal{X}} * f_m) \in \mathbb{R}^{n\times l}$

Figure 9: Schematic diagram of the CBHG filter bank module. The input sequence is processed by m different filter sets containing n filters each. The input is padded separately for each of these banks such that the convolution outputs are all of same dimensionality. For convenience the padded input $\overline{\mathcal{X}}$ is assumed to always have the correct padding. For each bank the results of the n filters are stacked into matrices of dimensionality $\mathbb{R}^{n\times l}$; each one is batch normalized separately. Finally, all matrices are stacked row wise into a single matrix.

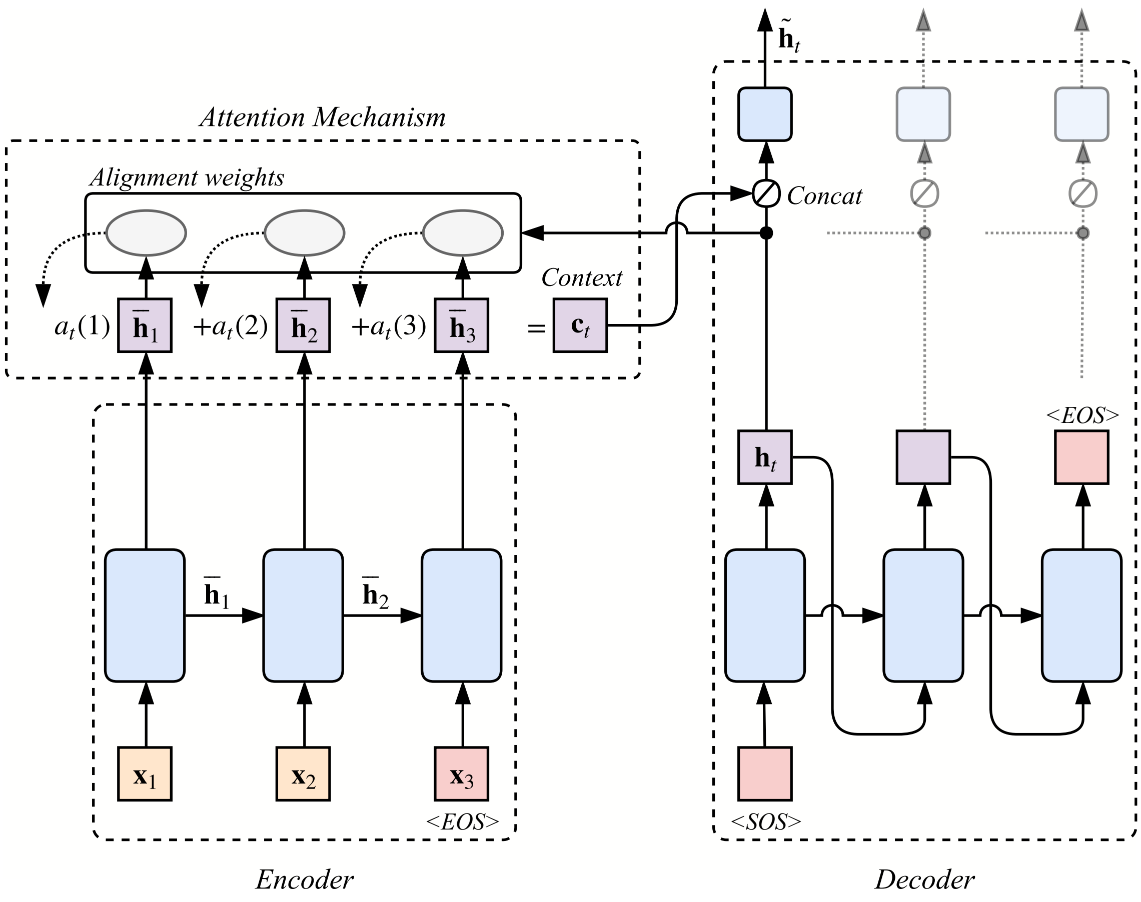


Figure 10: Global Luong attention mechanism using GRU RNN cells. At each decoder time step t the model infers a variable-length alignment weight vector $\mathbf{a}_t$. This vector is based on the current target state $\mathbf{h}_t$ and all source states $\overline{\mathbf{h}}_s$. Afterwards a global context vector $\mathbf{c}_t$ is computed as the weighted average of encoder source states using the alignment weights a_t. [10] Note that contrary to the generalized seq2seq architecture shown in section 2.1.5 no fixed-dimensional representation is passed between the encoder and the decoder.

The recurrent encoder processes inputs $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_S)$ and generates a sequence of encoder hidden states $\overline{\mathbf{h}}_s \in \left(\overline{\mathbf{h}}_1, \dots, \overline{\mathbf{h}}_S\right)$ also called source states. These source states are also commonly referred to as the *memory*. Afterwards a recurrent decoder generates a hidden state $\mathbf{h}_t \in (\mathbf{h}_1, \dots, \mathbf{h}_T)$ called the target state for each decoder time step t. However, contrary to the generalized seq2seq architecture described in section 2.1.5 no fixed size representation is passed between the encoder and the decoder. Instead, the decoder hidden states are initialized using zero vectors.

At each step t the model infers a variable-length alignment weight vector $\mathbf{a}_t$ based on the current target state $\mathbf{h}_t$ and all source states $\overline{\mathbf{h}}_s$. For simplicity the s-th element of the vector $\mathbf{a}_t$ is accessed by $\mathbf{a}_t(s)$. As shown in equation (16) each alignment weight is computed by comparing the current hidden state with each source state using the *softmax* function; consequently $\mathbf{a}_t(s) \in [0, 1]$ and $\sum_{s=1}^{S} \mathbf{a}_t(s) = 1$.

$$\mathbf{a}_t(s) = \operatorname{align}(\mathbf{h}_t, \overline{\mathbf{h}}_s) = \frac{\exp(\operatorname{score}(\mathbf{h}_t, \overline{\mathbf{h}}_s))}{\sum_{s'}^{S} \exp(\operatorname{score}(\mathbf{h}_t, \overline{\mathbf{h}}_{s'}))} \tag{16}$$

Using the alignments from $\mathbf{a}_t$ as weights a global *context* vector $\mathbf{c}_t$ (equation (17)) is derived as the weighted average of the encoder source states $\overline{\mathbf{h}}_s$.

$$\mathbf{c}_t = \sum_{s=1}^{S} \mathbf{a}_t(s)\overline{\mathbf{h}}_s \tag{17}$$

Luong et al. consider three different alternatives for the score function namely *dot*, *general* and *concat* [10]. This work deploys the *dot* variant (equation (18)).

$$\text{score}(\mathbf{h}_t, \overline{\mathbf{h}}_s) = \mathbf{h}_t^T \cdot \overline{\mathbf{h}}_s \tag{18}$$

In order to compute the new attended hidden state $\widetilde{\mathbf{h}}_t$ the current decoder hidden state $\mathbf{h}_t$ and the context vector $\mathbf{c}_t$ are concatenated. Subsequently, an affine transformation using a matrix $\mathbf{W}_c$ without a bias term is applied and the result is activated using the tanh function (equation (19)).

$$\widetilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c(\mathbf{c}_t \oslash \mathbf{h}_t)) \tag{19}$$

While the global Luong attention approach closely resembles that of the Bahdanau attention mechanism [11], there is an architectural difference that needs to be considered. Bahdanau et al. use a bidirectional encoder and hence the concatenated forward and backward state $\overline{\mathbf{h}}_s$. However, the original Luong publication [10] only uses the hidden states at the top of a stacked recurrent decoder. Different from [10] the proposed architecture in this work uses the concatenated hidden state for the attention mechanism.

4.4 Griffin-Lim Algorithm

This section is inspired by Sun et al. [44]. The *Griffin-Lim* algorithm [45] allows the estimation of a signal only from the magnitudes of its short-time Fourier transform (STFT). To reconstruct a signal from this representation it is necessary to invert the STFT, yielding a reconstructed signal. However, often only the magnitudes and not their phases are known making reconstruction problematic.

Let $|Y_\phi(mR, \omega)|$ denote the existing discrete-time magnitude STFT, were $R \in \mathbb{N}$ is the hop size, $\phi(n)$ denotes the analysis window used in the STFT and $n \in \mathbb{N}$ is the time. The variables $m \in \mathbb{N}$ and $\omega \in \mathbb{R}$ index the time frames and frequency, respectively. The goal is to find a signal $x(n)$, whose magnitude STFT $|X_\phi(mR, \omega)|$ minimizes the sum of the squared errors with the existing $|Y_\phi(mR, \omega)|$ (equation (20)). [44]

$$\sum_{m=-\infty}^{\infty} \int_{-\pi}^{\pi} \left(|X_\phi(mR, \omega)| - |Y_\phi(mR, \omega)|\right)^2 d\omega \tag{20}$$

Griffin et al. achieve this by starting with a random guess $x^{(0)}(n)$ of the signal. Then iteratively applying two processing stages until the convergence of equation (20). For each iteration i the two stages can be summarized as follows:

1. Compute the STFT $X_\phi^{(i)}(mR, \omega)$ of the current signal estimate $x^{(i)}(n)$. Retain the phases of $X_\phi^{(i)}$ and replace the magnitudes with the magnitudes $|Y_\phi(mR, \omega)|$ of the existing spectrogram.

2. Compute the Inverse Discrete-Time Fourier Transform (IDTFT) of $X_\phi^{(i+1)}(mR,\omega)$, denoted $\hat{x}_{\phi,mR}^{(i+1)}(n)$, for each m.[11] Using equation (21) a new approximation $x^{(i+1)}$ of the signal $x(n)$ is calculated. This is achieved by finding a windowed version $x_{\phi,mR}(n)$ of $x(n)$, that is close to $\hat{x}_{\phi,mR}^{(i+1)}(n)$.

$$x^{(i+1)}(n) := \frac{\sum\limits_{m=-\infty}^{\infty} \phi(mR-n)\hat{x}_{\phi,mR}^{(i+1)}(n)}{\sum\limits_{m=-\infty}^{\infty} \phi^2(mR-n)} \tag{21}$$

Griffin et al. show that this algorithm decreases the objective (equation (20)) with each iteration, converging to a stationary point. However, note that different initializations of $x^{(0)}$ can lead to different solutions. There is no guarantee that the algorithm converges to a global optimum. [44]

[11]The optimal solution is considered: $\hat{x}_{\phi,mR}^{(i+1)}(n) = \phi(mR-n)x(n)$

5 Method

This section discusses the proposed architecture, gives an overview over the corpora, hyper-parameter used and the experiments. Section 5.1 introduces the architecture. It decomposes the model into separate stages and illustrates each of them in-depth. Section 5.2 highlights the corpora as well as the preparation of the training data. Finally, section 5.3 details the training process and lists the used hyper-parameters. Following, figures use the same colors and labeling as defined in section 2.

5.1 Model Architecture

The proposed system directly derives from the Tacotron architecture [1]. Like Tacotron it employs a seq2seq encoder-decoder architecture with attention integrated into the decoder. A schematic of the architecture is illustrated in figure 11.

The model receives a written sentence in the form of text as input and models speech in the form of a waveform. The architecture is divided into four consecutive stages. The **encoder** processes the written text and produces a fixed-size embedding for each character. By iteratively producing spectrogram frames the **decoder** stage generates a mel scale magnitude spectrogram (MSMSPEC) from this fixed-size embedding. The subsequent **post-processing** stage is used to clean and refine the spectrogram. It also transforms the mel spectrogram into a linear scale magnitude spectrogram (LSMSPEC) for the final waveform **synthesis**. Note that this section does not refer to hyper-parameters and dimensions directly; all parameters for each stage can be found in section 5.3.2.

Training a system as a whole unit to directly model a target based on the given input (without explicitly modeling intermediate stages), is referred to as end-to-end training. While the encoder, decoder and post-processing stage employ neural networks, the synthesis stage converting spectrograms into waveforms does not. Waveform synthesis is done using a set algorithm; hence it is not optimized through the training process. When considering the conversion of text to speech (i.e. waveform synthesis) this particular architecture can not be considered direct end-to-end trainable but rather trainable in an end-to-end fashion.

As hinted in section 1.3, TTS systems can be differentiated into numerous approaches. The proposed model may best be classified as the *Text as language* approach. Taylor characterizes this approach as follows:

> [...] the process is seen as basically one of synthesis alone. The text itself is taken as the linguistic message, and synthesis is performed from this. [3, p. 39]

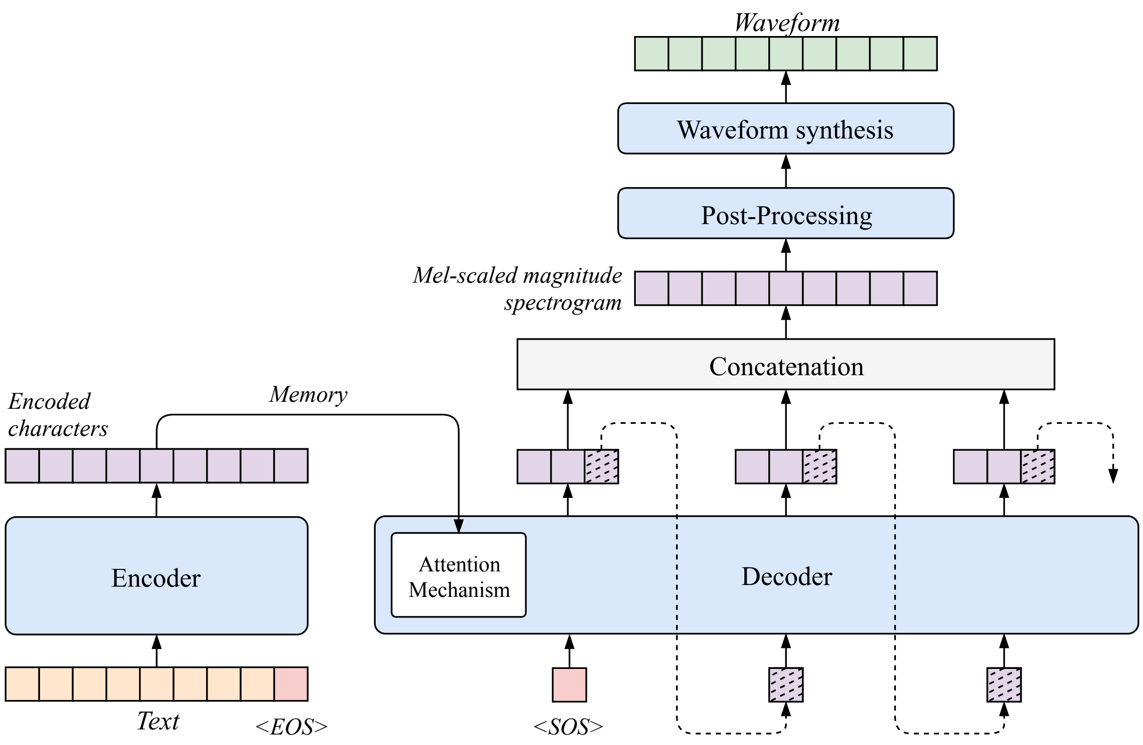
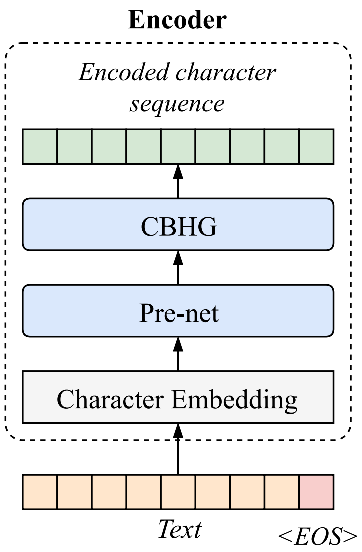
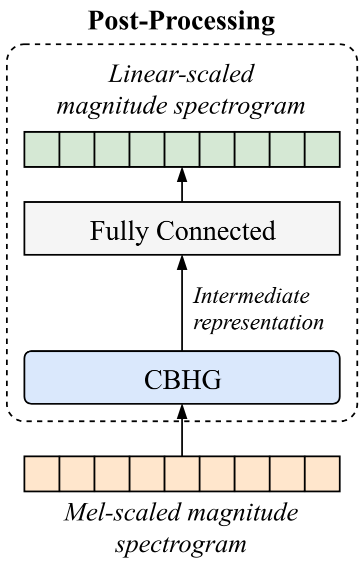


Figure 11: Schematic diagram of the proposed model architecture. It receives text as input and models speech as a waveform. The encoder produces a fixed size embedding for each character in the text. The attention based decoder iteratively produces spectrogram frames forming a MSMSPEC from the fixed-size embeddings. A post-processing stage transforms the MSMSPEC into a LSMSPEC for waveform synthesis in the synthesis stage.



(a) Schematic diagram of the encoder stage. The entered text is transformed into character embeddings of higher dimensionality. These embeddings are passed through a pre-net and a CBHG module to extract a robust high-dimensional representation for each character of the text.


Post-Processing
Linear-scaled magnitude spectrogram
Fully Connected
Intermediate representation
CBHG
Mel-scaled magnitude spectrogram

(b) Schematic of the post-processing stage. A CBHG module reduces artifacts and errors in the MSMSPEC producing an intermediate representation. This representation is transformed into a LSMSPEC using a fully connected layer. The intermediate representation is not expected to be a spectrogram.

Figure 12: Schematic diagrams of the encoder and the decoder stages.

5.1.1 Encoder

The encoder is the first processing stage in the architecture. Its main purpose is to generate a robust sequential representation of the text it receives as input. Figure 12a gives a schematic overview of the encoder.

First for each character of the entered sequence a fixed-size character embedding is generated (refer to section 2.4). Additionally, the virtual *EOS* and the padding (*PAD*) symbols are assigned an embedding as well. The *EOS* symbol is appended to each written sentence such that the encoder's RNN has a clear indication to when a sentence ends. Contrary, the padding symbol (*PAD*) does not yield any information. It is solely used to pad multiple sequences in a mini-batch to equal length such that they can be processed in parallel. Note that the embeddings are optimized like normal parameters (i.e. learned) during model training.

Subsequently, a non-linear transformation referred to as "pre-net" is applied to each embedding. The pre-net consists of a set of stacked fully-connected network layers of gradually decreasing size; of which each is paired with a dropout component. This pre-net is used as a bottleneck such that the embedding vectors are compressed into a lower-dimensional representation. The dropout is supposed to improve generalization and reduce overfitting.

A CBHG module is used to transform the output of the pre-net into the final encoder representation. See section 4.2 for a detailed description on the mechanics of the CBHG module. The convolutional filter banks used in the module explicitly model local and contextual information comparable to modeling unigrams, bigrams, up to N-grams for text [1]. Since the CBHG module incorporates a bidirectional RNN it additionally provides sequential features both from the forward and the backward context. Note that the output still contains the same number of elements (although higher dimensional) as the encoder input has characters. Wang et al. found that this CBHG-based encoder not only reduces overfitting, but also leads to fewer mispronunciations compared to a standard multi-layer RNN used as the encoder [1]. Especially the contained highway networks are reported to significantly improve the quality of character-level models when used with convolutional layers [12, 46].

5.1.2 Decoder

The decoder is the central component of the model. Based on the encoded characters it receives from the encoder it generates a MSMSPEC that is suitable for later synthesis. Please refer to section 5.2.2 for a detailed description on why this feature representations is used. It is implemented as a seq2seq attention decoder and employs a stateful recurrent attention layer. Figure 13 shows a schematic representation of the decoder.

The decoder does not generate the MSMSPEC target in a single step, but rather models it as a sequence of spectrogram frames. With each decoding iteration a set of r individual non-overlapping frames is produced. Since the decoder is an attention based seq2seq decoder there is no direct connection between the encoder's output and the decoder RNN cell input. The actual input for to each iteration is a single spectrogram frame. The encoded text representation from the previous stage is fed as the "memory" to an attention

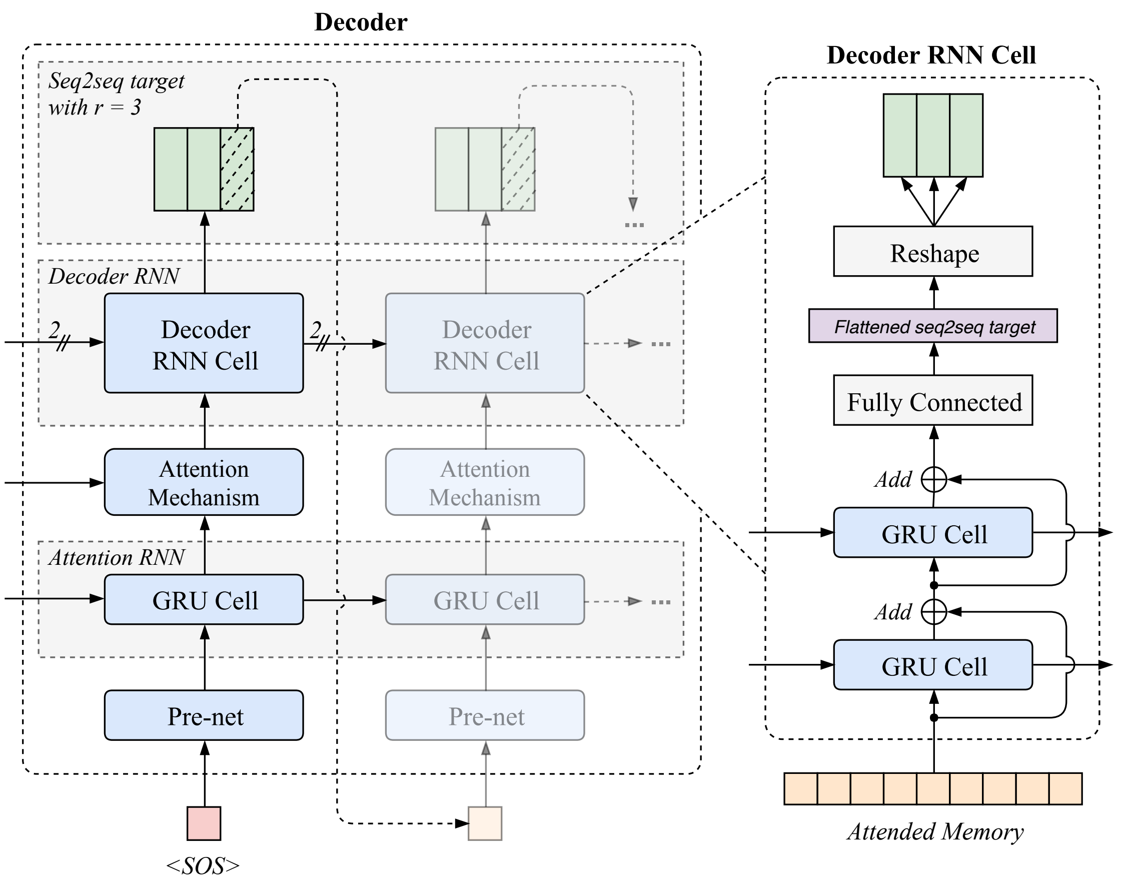


Figure 13: Schematic diagram of the decoder stage. Single MSMSPEC frames are fed as input to produce a set of r subsequent non-overlapping spectrogram frames at each decoding step. Starting with a SOS frame each input first passes a pre-net and an attention RNN producing a robust sequential input for the attention mechanism. The attention mechanism uses it to derive new alignment weights and select encoder character representations for the current decoding step. It produces the "attended memory" which is then decoded into the sequence of r spectrogram frames using a stacked residual GRU RNN. The last of the r frames is then fed as input to the next decoder step.

mechanism. This data flow can be seen in figure 11. Until a break condition is met the last of the r frames from the previous decoder iteration is used as the input to the next one. Note that this is only done during inference and evaluation. While during training r frames are likewise predicted, instead of feeding the last frame as input the corresponding frame from the ground-truth spectrogram (i.e. the training data) is fed.

As the decoder has no direct indication when to stop producing frames decoding is stopped after a set number of iterations. At the first decoding step there is no input frame and likewise there are no previous cell states for the internal RNN cells. A synthetic SOS frame (zero vector) is fed as the first input and all GRU cell hidden states are likewise initialized using zero vectors.

During decoding the entered spectrogram frame is first processed using a two layer pre-net. Like the pre-net used in the encoder it uses dropout which is essential for the model to generalize. After the pre-net, the processed frame enters the attention RNN composed of a single GRU layer. This layer produces a robust sequential attention query at each decoder time step. The query generated by the attention layer is used by the attention mechanism. At each iteration the attention mechanism calculates a discrete probability distribution (called alignment) over the encoded characters based on the query. The alignments capture which encoder steps (hence characters) are expected to be relevant for the current r frames to be generated. Memory, alignments and attention query are combined by the mechanism producing the attended memory (refer to figure 10).

The subsequent decoder RNN consists of stacked GRU cells with vertical residual connections [47] to speed up convergence. As can be seen in figure 13, a subsequent fully-connected layer and a reshape operation transform the cell output into the final seq2seq target.

The process of generating r subsequent non-overlapping frames at each decoder step is an important design choice. When predicting r frames in one decoder step, the total number of decoder steps required divides by r. This in turn reduces the training and inference times while also reducing the model size. As reported by Wang et al. this trick substantially increases convergence speed, as measured by a much faster and more stable alignment learned from attention. They attribute this to the high temporal correlation between successive spectrogram frames. Emitting multiple frames at once forces the model to compress the redundant information and capture modalities of multiple frames in one step. Attention is not forced to attend to the same input token for multiple time steps such as when only emitting one frame at a time. [1]

5.1.3 Post-Processing

The post-processing stage is the last neural network based stage of the model. Figure 12b shows a schematic of this stage. Post-processing has two main tasks. First it enhances the MSMSPEC it receives as input by correcting decoder prediction errors. Second it converts the input into a LSMSPEC suitable for waveform synthesis.

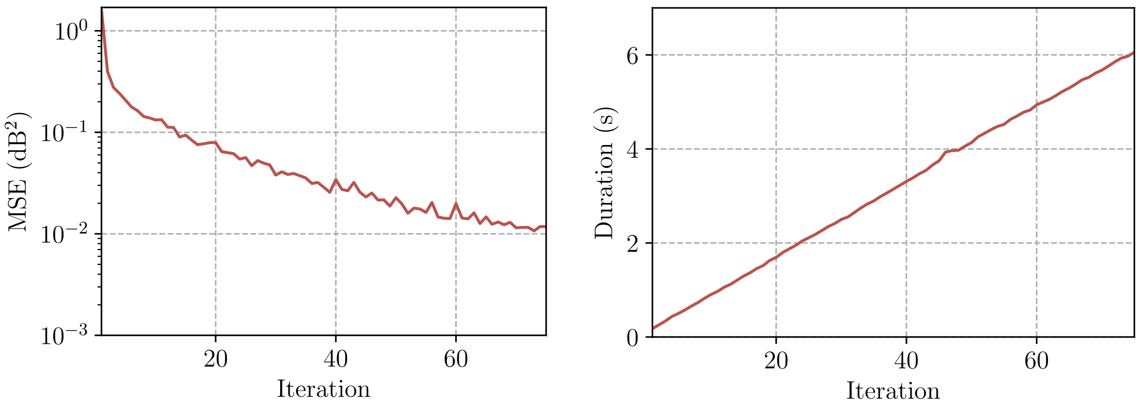
The spectrogram received from the decoder is enhanced using a CBHG module. Like in the encoder stage the contained filter banks can capture local and contextual information; identifying inconsistencies in the magnitudes. Additionally, it uses forward and backward information to correct the prediction error for each individual frame; in contrast to the decoder which always runs from left to right [1]. Note that the output of the CBHG module is of higher dimensionality than the MSMSPEC and is not expected to be a spectrogram but rather an intermediate representation. However, the subsequent synthesis stage excepts a LSMSPEC as input. In order to generate this a fully-connected layer transforms the intermediate representation into a magnitude spectrogram of linear scale.

5.1.4 Waveform Synthesis

Waveform synthesis is the final stage of the architecture. Like in [1] it uses the *Griffin-Lim* algorithm to synthesize a waveform signal from the LSMSPEC the post-processing stage produces. The algorithm iteratively minimizes the Mean Squared Error (MSE) between the STFT of the reconstructed signal and the target signals STFT. For a detailed description of the algorithm refer to section 4.4.

As hinted by Wang et al., while *Griffin-Lim* is differentiable, it does not have trainable weights [1]. Hence, it is not a neural component and therefore it is not optimized through training. In this work it solely implemented on the Central Processing Unit (CPU) as a subsequent processing step to the neural network.

Prior to feeding the predicted magnitudes to the algorithm, they are raised by a power of 1.3 as it is reported to reduce artifacts in the reconstructed signal [1]. Then 50 iterations of the *Griffin-Lim* algorithm are performed. Wang et al. also use 50 cycles whilst reporting that 30 iterations seem to be enough [1]. Figure 14a illustrates the reconstruction MSE in relation to the number of iterations used. Note that the computation time for reconstruction grows linearly with the number of iterations as shown in figure 14b.



(a) Plot of the Mean Squared Error of the Griffin-Lim algorithm in relation to the number of iterations used. The plotted MSE is averaged over 250 randomly selected waveforms from the Blizzard corpus. Note that the error is plotted on a logarithmic scale.

(b) Computation time (CPU) of the Griffin-Lim algorithm in relation to the number of reconstruction iterations used. The algorithm scales linearly with the number of iterations. While performance depends on the hardware, the linear behavior remains consistent.

Figure 14: Reconstruction error and computation time of the Griffin-Lim algorithm.

5.2 Corpora

In accordance with the goals from section 1.1 the training data is freely available such that results can be copied and verified. The datasets contain speech from a single speaker in the form of continuous spoken sentences together with corresponding transcripts. That way the model can be trained in end-to-end fashion imposing little restrictions on the data. However, little information exists on how much data is actually required to achieve a working single speaker model. Nevertheless, like with unit selection systems, one can expect a clear relation between having more data and achieving better results [3, p. 522]. Better results manifesting in having less miss-pronunciations, less noise in the generated signal or an overall increase in perceived naturalness.

As the objective is to explore the architecture the amount of data used for training is derived from other exploratory work in this area. Table 1 lists dataset durations used by other single speaker TTS models. As can be seen most systems are evaluated using non-public English corpora and are trained with 20 to 35 hours of speech. In order to achieve comparable results for evaluation ideally the same dataset must be used to train each model. As nearly all datasets listed in table 1 are internal datasets that are not freely accessible, this work focuses on different publicly available corpora.

Table 1: Overview of datasets and their durations used by other single speaker TTS models. Almost all models are trained using non-public internal English datasets. The duration used for training ranges from 20 to 35 hours of speech. The amount of data used by Char2Wav is unknown.

Model	Duration (h)	Language	Corpus
Tacotron [1] Tacotron 2 [39]	24.6	eng.	internal
WaveNet [36]	24.6 34.8	eng. chi.	internal
Deep Voice [37] Deep Voice 2 [40] Deep Voice 3 [41]	20.0	eng.	internal
Char2Wav [21]	—	eng.	CSTR VCTK [48]

Two distinct datasets are used for training:

Blizzard Challenge 2011 Dataset

The *Blizzard Challenge 2011* dataset[12] [49] is a single speaker English voice corpus. The recorded speaker is a professional female voice coach with American English accent. It contains a total of 16 hours and 45 minutes of studio quality recordings of short sentences in neutral speaking style with corresponding transcriptions (see appendix section B for statistics). This corpus was used as the official dataset for the Blizzard Challenge 2011 [49].[13] It's licensing allows it to be used for research free of charge.

This corpus contains around 3 hours and 15 minutes fewer data than the smallest dataset listed in table 1. However, it is selected because it is actually created for the purpose of building a TTS system and offers high quality recordings with a neutral speaking style.

LJ Speech Dataset

The *LJ Speech* dataset[14] [50] is a public domain speech dataset recorded by the *LibriVox* project[15]. It consists of short audio clips read in English by a single female speaker[16]. All recordings are passages from 7 non-fiction books together with corresponding transcriptions. The texts were published between 1884 and 1964, they are public domain.

The recorded clips have a total length of approximately 24 hours. All clips were segmented automatically based on silences in the recording. Ito matched the text to the audio manually and a QA pass to ensure that the text accurately matches the words in the audio. [50]

The duration matches that used by other models as listed in table 1. However, the recordings are of lower quality compared to the Blizzard Challenge 2011 dataset. To asses the ability to learn from different amounts of data with varying quality, this dataset is used to train separate models.

[12] Also known as the *Nancy* corpus; referring to the speaker Nancy Krebs.

[13] The Blizzard challenge is held annually with the goal to compare researched techniques in building corpus-based speech synthesizers on the same data.

[14] Version 1.1 of the LJ Speech dataset is used in this work.

[15] LibriVox project: `https://librivox.org/`

[16] Recordings by Linda Johnson (`https://librivox.org/reader/11049`)

5.2.1 Pre-Processing

Although training is done in an end-to-end fashion, data from the corpora is pre-processed to ensure the data is consistent. The textual transcriptions and the speech recordings from the datasets are processed as follows:

Text Processing:

- All characters are converted to lowercase.
- Sentences containing digits (e.g. ages, dates, years, weights) are replaced with a full textual representation matching to the recordings.[17]
- Abbreviations are written-out[18].
- All characters other than `a-z`, `␣`, and `" ' - , ; : ( ) ! ?` are stripped.
- Each sentence is closed with the virtual EOS character.

Note that this text processing procedure closely resembles that used by many other TTS approaches (see section 1.3.4).

Waveform Processing:

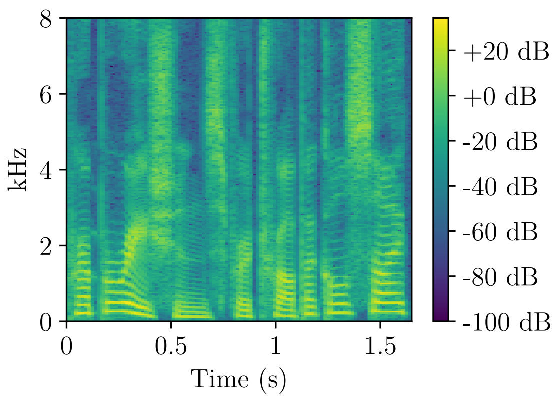
- Files containing only silence or breathing are dropped.[19]
- Non-deterministic silence at the beginning of recordings (applies only to LJ Speech) is removed. This is done by thresholding the signal power with a manually chosen limit.
- Spectrograms are extracted and normalized as described in section 5.2.2.

5.2.2 Feature Extraction

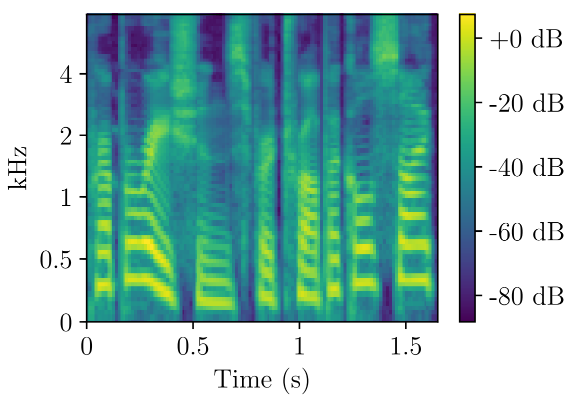
The model receives written text as input and produces speech in the form of a waveform signal. However, as described in section 5.1 the final synthesis stage is of non-neural nature and requires a LSMSPEC as input. While a LSMSPEC is suitable for waveform synthesis it is a highly redundant representation of the information actually needed during decoding. This redundancy makes the model more complex and restricts the process of learning an alignment between the written text and the speech signal. However, learning this alignment is ultimately the goal of the seq2seq attention architecture. Therefore, the decoder is designed to predict more compact MSMSPEC frames instead.

The basic idea behind using MSMSPEC is, that not all frequency bands contained in the linear scale spectrogram are of equal importance in human speech. The mel scale is a perceptual scale of pitches that effectively allows the reduction of frequency resolution as the frequency increases. Let's consider the typical range of human voice. The major bulk of signal energy is contained below 5 kHz, although voice can contain frequencies up to 20 kHz [51]. This can greatly vary depending on age, sex and origin of the speaker.

[17]For example: "He died in 1855." becomes "He died in eighteen fifty five."
[18]For example: "mr." becomes "mister"
[19]Files containing only breathing or silence are identified by their missing transcriptions.



(a) Linear scale magnitude spectrogram in decibel representation.



(b) Mel scale magnitude spectrogram in decibel representation.

Figure 15: Visualization of linear and mel scale magnitude spectrograms. Using the beginning of the randomly chosen utterance “Preparations for tropical cyclone [. . .]” (RURAL-02198) from the Blizzard dataset as an example.

Using a randomly chosen utterance from the Blizzard dataset figure 15a illustrates that the main structural part is contained below 4 kHz. Since frequency bands above 4 kHz carry less information the spectrogram can be highly compressed, as long as it provides enough intelligibility and prosody information for an inversion process. A comparison between a linear scale and a corresponding mel scale magnitude spectrogram can be seen in figure 15.

The parameters used for spectral analysis are listed in table 2. For the calculation of the STFT spectrograms the *Hanning* window function is used and the waveform signal is sliced into windows of 50 ms width with a hop of 12.5 ms. The linear scale frequencies are build using a Fast Fourier Transform (FFT) window size of 2,048 and the mel scale is build using 80 mel banks. As opposed to [1], no pre-emphasis filtering is applied to the waveform signal before spectrogram calculation.

Prior to training the minimal and the maximal linear and mel scale spectrogram magnitudes are calculated for each dataset. They are determined over the training subset of the Blizzard and the LJ Speech dataset. Before feeding the spectrograms for training or evaluation they are converted into an absolute decibel representation with the reference being a magnitude of 1. Towards zero the magnitudes are capped to 10^{-5}, hence -100 dB. Afterwards they are linearly scaled down to the range $[-100, 0]\,dB$ using the minimum and maximum decibel magnitudes calculated over the datasets. Subsequently, this range is scaled linearly such that the numeric values fit the range $[0, 1]$. Note that the spectrograms are not processed to have zero mean or a constant standard deviation.

5.3 Training Description

The architecture is trained separately both on the Blizzard and the LJ Speech dataset for the experiments described in section 5.3.3. Both datasets are split into non-overlapping train and evaluation sets as shown in table 7. Training progression is measured and tracked using the evaluation metric described in section 5.3.1. The models are evaluated every 5,000 batches to check if overfitting occurs. This strategy helps to determine if training still progresses or if performance has converged and the training has to be halted. The gradient clipping threshold used to prevent the gradients from exploding is set to 5.0.

Prior to training *Glorot* normal initialization also known as *Xavier* initialization is used [52]. It is applied to nearly all weights of the model except for some bias vectors. The GRU cells internal biases and the ones used in the convolutional layers are initialized to $\vec{\mathbf{1}}$ and $\vec{\mathbf{0}}$, respectively. All highway network layers use a negative transform gate bias of $(-\vec{\mathbf{1}})$, as a small negative bias is reported to be beneficial for highway layers [42].

For a detailed specification of the hyper-parameters refer to section 5.3.2.

5.3.1 Evaluation Metric

Synthesis performance is measured using ℓ_1 loss for both the decoder MSMSPEC target and the post-processing LSMSPEC target. Let the matrices $\mathbf{M}, \widehat{\mathbf{M}} \in \mathbb{R}^{N_m \times T}$ denote the magnitudes of the mel scale spectrograms from the training data and the decoder, respectively. Were N_m is the number of mel bins used and T the number of spectrogram frames. Likewise, let $\mathbf{L}, \widehat{\mathbf{L}} \in \mathbb{R}^{N_l \times T}$ denote the equivalent magnitude matrices of linear scale from the training data and the post-processing stage, respectively. With N_l being the number of discretized frequency bins for the linear scale. Then the decoder loss ℓ_d is calculated as shown in equation (22) and the post-processing loss ℓ_p as shown in equation (23).

$$\ell_d = \parallel M - \hat{M} \parallel_1 = \sum_t^T \sum_f^{N_m} \mid M_{ft} - \hat{M}_{ft} \mid \tag{22}$$

$$\ell_p = \parallel L - \hat{L} \parallel_1 = \sum_t^T \sum_f^{N_l} \mid L_{ft} - \hat{L}_{ft} \mid \tag{23}$$

The rows of the spectrograms are indexed by frequency f and the columns are indexed by time t. Note that the zero padding frames inserted when mini-batches are created are not masked during the calculation of the loss. Hence, the network is conditioned to produce zero frames after it stops producing speech. Both losses are combined with equal weight as shown in equation (24) to form the total model loss ℓ_t used for training and evaluation.

$$\ell_t = \ell_d + \ell_p \tag{24}$$

5.3.2 Hyper-Parameters

The parameters in use are listed in table 2. The models are trained using stochastic gradient descent with the *Adam* optimizer [53], setting $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Instead of using a piecewise constant learning rate like [1] the learning rate is set to 0.001 and decays exponentially with an anneal rate of 0.9 every 20,000 batches. For training, mini-batches of size 40 are build and the entire dataset is shuffled after each epoch.

Prior to feeding the LSMSPEC to the Griffin-Lim algorithm the magnitudes are raised to the power of 1.3.

Table 2: Hyper-parameters of the proposed model architecture. The term "conv-k-c-ReLU" denotes a 1-D convolution with kernel width k and c output channels (i.e. separate kernels). ReLU denotes the Rectified Linear Units (ReLU) activation function. FC denotes a fully-connected layer.

Component	Parameter
Training	Batch size: 40 Optimizer: *Adam*, $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ Gradient clipping norm: 5.0
Spectral analysis	Frame length: 50 ms, frame shift: 12.5 ms FFT window size: 2048 Window type: Hann Mel bands: 80
Character embedding	256-D
Encoder pre-net	FC-256-ReLU $\rightarrow$ Dropout(0.5) $\rightarrow$ FC-256-ReLU $\rightarrow$ Dropout(0.5)
Encoder CBHG	Conv1D bank: K=16, conv-k-128-ReLU Max pooling: stride=1, width=2 Conv1D projections: conv-3-128-ReLU $\rightarrow$ conv-3-128-Linear Highway network: 4 layers of FC-128-ReLU Bidirectional GRU: 128 cells
Decoder pre-net	FC-256-ReLU $\rightarrow$ Dropout(0.5) $\rightarrow$ FC-256-ReLU $\rightarrow$ Dropout(0.5)
Attention RNN	1-layer GRU (256 cells)
Attention mechanism	Type: Luong-style Mode: Global attention (dot)
Decoder RNN	2-layer residual GRU (256 cells)
Post-processing CBHG	Conv1D bank: K=8, conv-k-128-ReLU Max pooling: stride=1, width=2 Conv1D projections: conv-3-256-ReLU $\rightarrow$ conv-3-80-Linear Highway network: 4 layers of FC-128-ReLU Bidirectional GRU: 128 cells
Reduction factor (r)	5
Griffin-Lim	50 iterations

5.3.3 Experiments

To evaluate and assess the capabilities of the proposed architecture the following experiments are carried out.

Training on two different datasets:
To obtain information on how the architecture handles training data of varying quality it is trained separately on the Blizzard and the LJ Speech dataset. Both models use the same parameters as described in section 5.3.2.

- Section 6.1 examines the resulting Blizzard model.
- Section 6.2 examines the resulting LJ Speech model.

Analysis of the effects of different model components:
The effects of the post-processing stage, the spectrogram inversion process and the models capability to predict attention alignments are examined.

- To assess the impact of the post-processing stage (see section 6.5) the LJ Speech model is trained in a second configuration. Supplementary to the model with the post-processing (as described in section 6.2) it is also trained without post-processing. In the later case the decoder still produces a mel scale spectrogram which is transformed to linear scale using a fully-connected layer.
- The attention alignments and the spectrogram inversion process are examined using the Blizzard model (section 6.1) and the LJ Speech model (section 6.2) described initially. Particularly the capability to predict continuous alignments (section 6.1.1) and the effects of raising the spectrogram magnitudes to a power (section 6.6) are examined.

Analysis of the effects of different amounts of target features:
The influence of the decoder target frame size on the models training capability is analyzed (see section 6.4). Supplementary to the Blizzard model that uses 80 mel bands (section 6.1) four additional models are trained using 20, 40, 120 and 160 mel bands. Otherwise, all models share the same parameters as described in section 5.3.2.

Evaluation of the naturalness of the produced speech:
In order to rank the proposed architecture in contrast to other systems a MOS test is conducted (see section 6.7). In the process the Blizzard model trained in section 6.1 is compared against two other TTS systems.

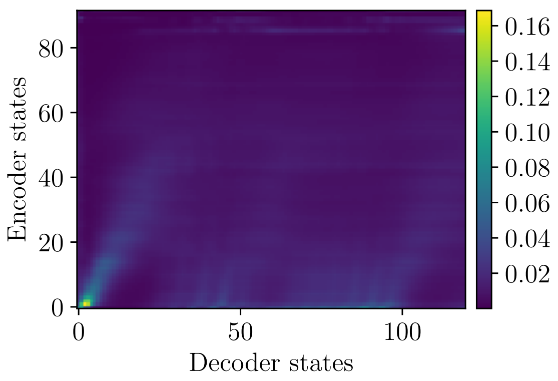
6 Results and Discussion

This section lists the results for the experiments described in section 5.3.3. All results are followed by separate discussions, interpreting the findings. Each result is derived from a single training run, thus there is no measure of error available for the collected data.

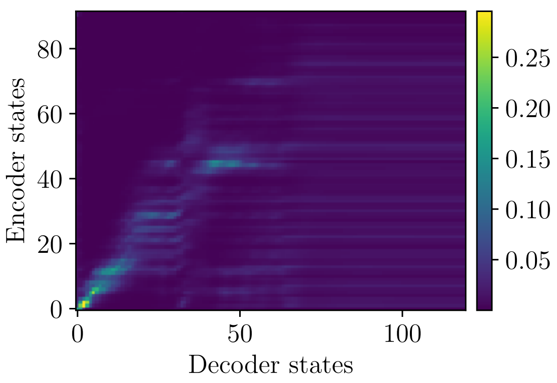
6.1 Blizzard Model Training

6.1.1 Results

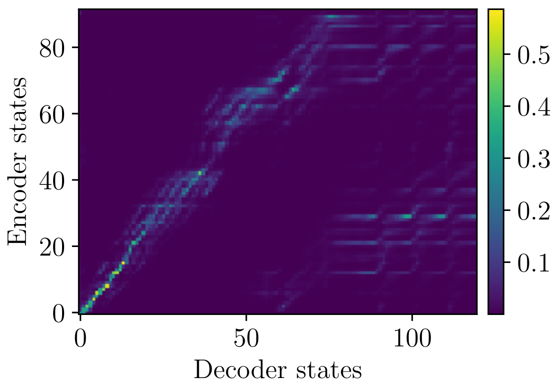
Figure 17a depicts the progression of the total loss ℓ_t during training. For completeness figure 17c and figure 17d show the decoder loss ℓ_d and the post-processing loss ℓ_p. The Blizzard model is trained for 520,000 steps over a total computation duration of 28 days and 13 hours. Every 5,000 batches the model is evaluated on the validation portion of the dataset (see figure 17b). Progression of the predicted attention alignments is illustrated in figure 16. The alignments are predicted during inference for different amounts of training. Also, the model does not experience exploding-gradients during training. Experimental development models trained without gradient-clipping suffered from gradient-explosions.



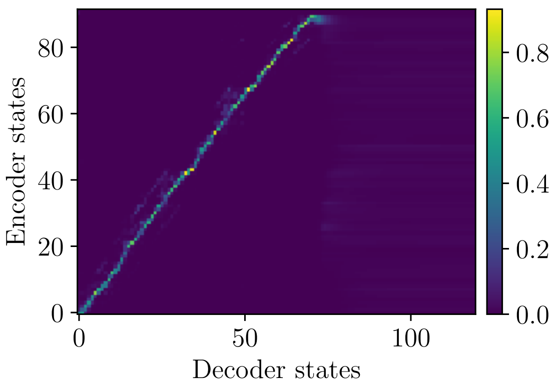
(a) Predicted alignments for all decoder states after 11,500 training steps.



(b) Predicted alignments for all decoder states after 13,000 training steps.

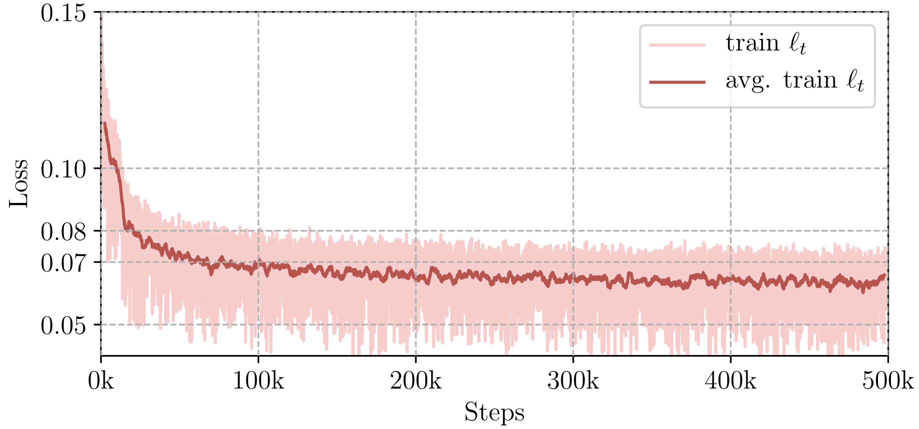


(c) Predicted alignments for all decoder states after 14,000 training steps.

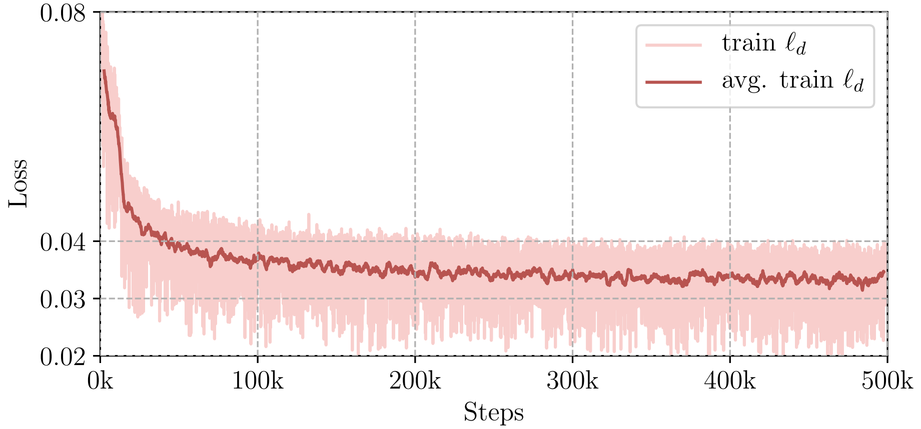


(d) Predicted alignments for all decoder states after 15,000 training steps.

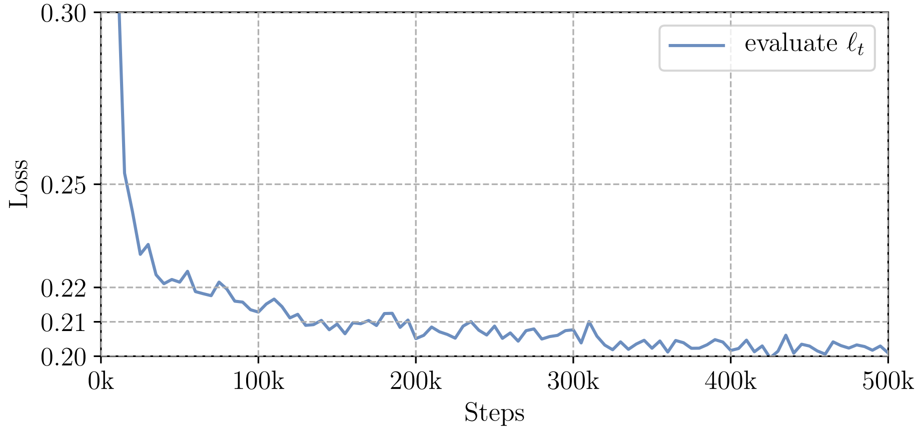
Figure 16: Progression of the attention alignment during inference. Each encoder state corresponds to a character of the input sentence. All alignments are generated for the same randomly chosen utterance using the Blizzard model. Note that the alignments in each column sum up to 1 as they are a probability distribution over the encoder states. This does also hold true for the alignments produced after 15,000 steps.



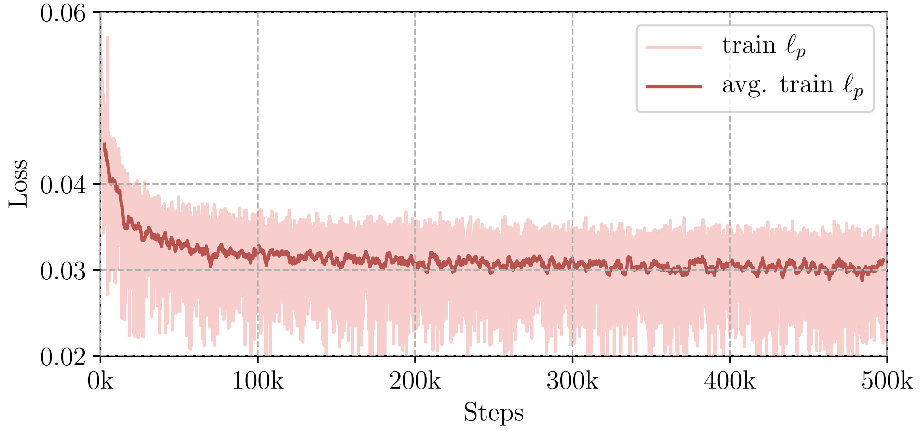
(a) Progression of the total loss (ℓ_t) during training on the Blizzard dataset.



(c) Progression of the decoder loss (ℓ_d) during training on the Blizzard dataset.



(b) Progression of the total loss (ℓ_t) during evaluation on the Blizzard dataset. The first 15,000 steps are truncated for better scaling.



(d) Progression of the post-processing loss (ℓ_p) during training on the Blizzard dataset.

Figure 17: Progression of the models losses during training and evaluation on the Blizzard dataset. The training losses are plotted every 50 steps and the evaluation losses every 5,000 steps. A moving average is calculated for all training losses using a window width of 50 and a stride of 1. The average is only calculated when the window completely overlaps with the signal. Note that the total training loss is smaller than the evaluation loss.

6.1.2 Discussion

The training of the Blizzard model seems successful. It can be observed from figure 17 that more training is generally positive for the synthesized result. After initialization, the total loss during training seems to decrease until around 300,000 to 350,000 steps are reached. Beyond that the model barely progresses further in terms of training loss. Note that the training loss is substantially smaller than the evaluation loss. This is probably due to the feeding of ground-truth frames during training. Subjective listening on a sample basis suggest that extended training periods still achieve marginal improvements. The evaluation loss, which decreases until 420,000 steps seems to correspond with this observation. A potential explanation might be that the used loss metric is not capable of capturing all nuances of the human voice. For example the pace at which to speak can not directly be derived from the entered text, there are small inhomogeneous translations and varying pauses that are not captured correctly. This is also supported by the analysis of the post-processing stage (section 6.5), showing inhomogeneous pauses in the spectrogram for the same input text.

Manual inspection of the alignments during evaluation suggests, that there are no cases of endless repetition at the end of generated speech. This might be attributed to not masking padding frames during training. As mentioned above there is no information on the pace a sentence is spoken or for how long the model has to produce frames. The model is required to derive this from the training data and decide on its own when to stop generating speech. That implies that the more homogeneous the speech and the less variation there is the better the model might be at fitting the data.

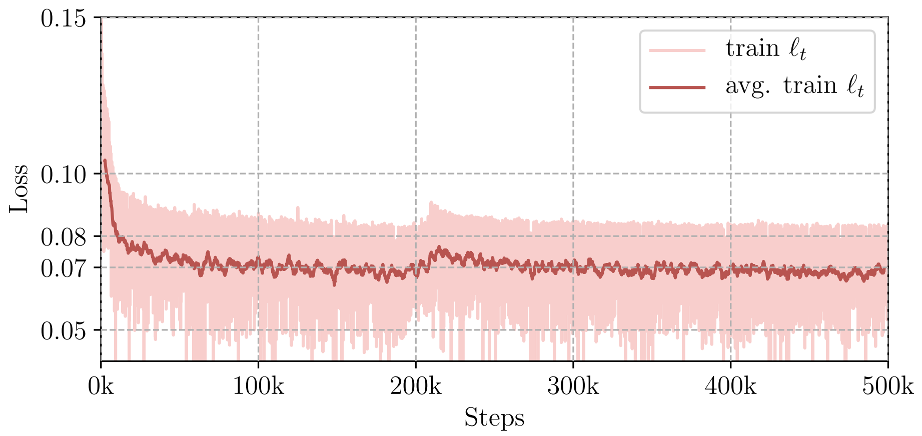
The alignments are learned from the beginning of the encoder sequences and “grow” coherently without temporal jumps. While they can be predicted early (after around 15,000 steps), at this point during inference they are not particularly robust. However, as training progresses the alignment prediction gets more stable. Note that the losses decrease rapidly until the model is capable of predicting the first coherent attention alignments for the whole sequence.

The evaluation procedure shows no indication of overfitting. This can be observed in figure 17, with the evaluation loss converging throughout the entire process without starting to diverge from the training loss. This is likely due to the high dropout rates used in all the pre-net layers. After the implementation of gradient-clipping, the exploding gradient problem did not occur again. Hence, indicating that the gradient-clipping strategy successfully prevents it.

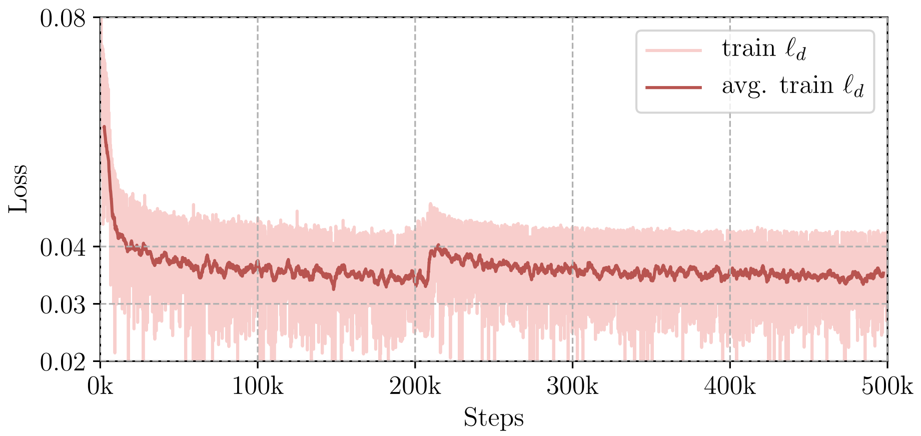
6.2 LJ Speech Model Training

6.2.1 Results

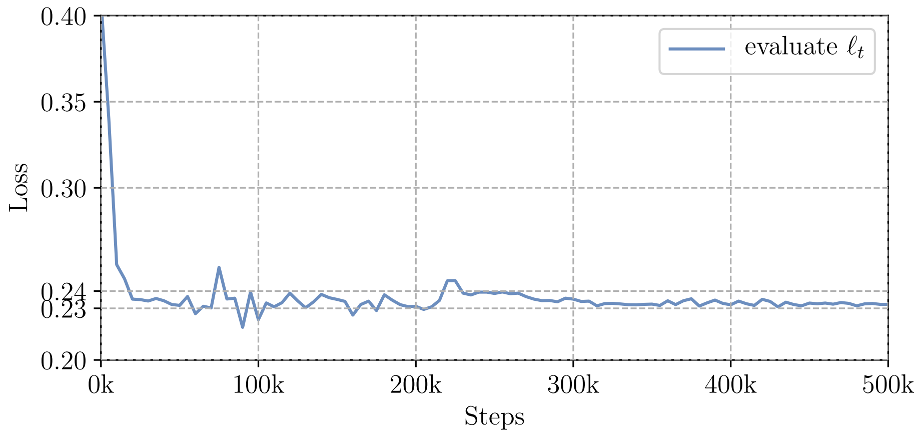
Figure 18a depicts the progression of the total loss ℓ_t during training. For completeness figure 18c and figure 18d show the decoder loss ℓ_d and the post-processing loss ℓ_p. The LJ Speech model is trained for 505,000 steps over a total computation duration of 5 days and 2 hours. Every 5,000 batches the model is evaluated on the validation portion of the dataset as shown in figure 18b. For brevity visualization of the alignments is omitted.



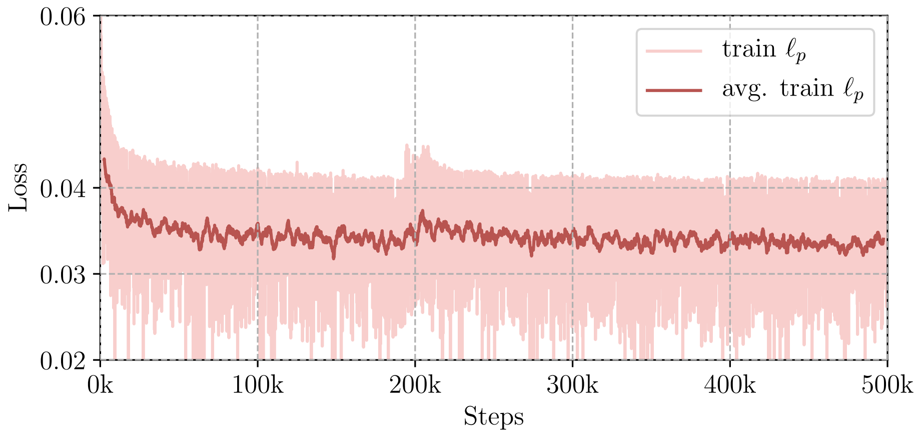
(a) Progression of the total loss (ℓ_t) during training on LJ Speech.



(c) Progression of the decoder loss (ℓ_d) during training on LJ Speech.



(b) Progression of the total loss (ℓ_t) during evaluation on the LJ Speech dataset.



(d) Progression of the post-processing loss (ℓ_p) during training on LJ Speech.

Figure 18: Progression of the model's losses during training and evaluation on the LJ Speech dataset. The training losses are plotted every 50 steps and the evaluation losses every 5,000 steps. A moving average is calculated for all training plots using a window width of 50 and a stride of 1. The average is only calculated when the window completely overlaps with the signal. Note that the total training loss is smaller than the evaluation loss.

6.2.2 Discussion

It can be observed from figure 18 that the model seems to converge fairly early in training. After around 60,000 steps, barely any progression both in terms of total training loss can be observed. The same seems to hold true for the evaluation loss, albeit it still exhibits noticeable variance. Note that the evaluation loss experiences a surge at around 220,000 steps. While this surge is temporary and vanishes 100,000 steps later, the model does not improve any further. Due to this increase vanishing again it seems the model does not experience any long term overfitting. Note that when compared to the Blizzard model, the main difference in absolute runtime is because the models are trained on different hardware. This can also partially be attributed to the difference in file durations' between the used datasets as shown in appendix section B.

Listening to results on a per sample basis reveals that although intelligible voice is generated the LJ Speech model appears to generate more perceivable noise compared to the Blizzard model. Examining random samples from the dataset one can notice that the ground-truth audio waveforms from the LJ Speech dataset contain low background noise themselves. Also, as described in section 5.2.1 removal of the of non constant duration silence in the recordings might not have worked in all cases. While the model is capable of producing comprehensible speech, the reasons mentioned above seem to preventing it from producing a low-noise signal. Nevertheless, the model is still able to learn to predict attention alignments after around 15,000 steps. It is also able to predict when it has finished a sentence and hence has to stop producing speech. Therefore, it is able to derive a general temporal structure. Note however, that the evaluation loss barely progresses any further once the model learned to produce continuous attention alignments. That might indicate that the model is restrained by the decoder, not the encoder or the attention mechanism in general. The fact that the model is still capable of training and producing speech may be attributed to the feeding of ground-truth frames during training. This would also explain why attention alignments can be learned, while the decoder possibly has trouble fitting the data.

Ultimately, based on this individual run, it can not be excluded that the behavior might partially be caused by poor weight initialization or unsuitable hyper-parameters.

6.3 General Training Discussion

Although trained on different data, the models described above are comparable as in both cases the loss is calculated using normalized spectrograms. Comparing the evaluation loss between both models makes the early progression of the LJ Speech model particularly apparent. It can be observed, that while the Blizzard model is trained on less material (∼7 hours less), it is able to achieve a better evaluation error (figure 17). Based on the training loss, the progress can barely be observed in either case. This is certainly due to the feeding of ground-truth frames during training. Subsequent spectrogram frames are expected to have a high correlation. Therefore, the fed frames have a strong influence on the loss when predicting r frames with each iteration. Both models learn to predict attention alignments at the same pace. This is most likely also due to the feeding of ground-truth frames during training. As it allows the architecture to learn right from the start, limiting the effect of defective outputs from the encoder. In turn the decoder likewise can produce outputs that the post-processing can learn from; even in the early stages of training.

6.4 Feature Size

To assess the impact of the number of mel filter banks multiple models with 20, 40, 80, 120, 160 banks are compared. The model using 80 banks is equal to the model described in section 6.1.

6.4.1 Results

Additional models using 20, 40, 120, 160 banks are trained for 200,000 iterations each. The total computation time for these four models combined is 6 days. Figure 19 and figure 20 shows the influence of the number of mel banks on the decoder loss during training. Figure 20 illustrates the influence on the post-processing loss. The effect of the number of banks on the total evaluation loss can be seen in figure 21.

6.4.2 Discussion

One might generally expect a better result using more banks as the spectrogram gets more detailed. However, increasing the number of banks lead to an increase in total loss during training, while reducing the number of banks reduces the total loss. As can be seen in figure 19 while the decoder loss reduces, the post-processing loss shown in figure 20 remains overall consistent. The uniform evaluation losses from figure 21 confirm that the post-processing loss remains consistent. This implies, that predicting the post-processing target does not suffer from the reduced decoder target size. But it also indicates, that the architecture has difficulties predicting a decoder target with increasing size. Hence, either the number of parameters in the decoder RNN cell has to increase to better capture the increased decoder target. Or another way of transforming mel into linear scale spectrograms has to be devised to reduce the post-processing loss.

6.5 Post-Processing

6.5.1 Results

The LJ Speech model without post-processing is trained for exactly the same number of steps as the version with it (see section 6.2). The difference between predicting the linear scale spectrogram with and without the post-processing network can be seen in figure 22a and figure 22b.

6.5.2 Discussion

The results from training the proposed model with and without the post-processing stage reveals that it is an important component of the model. It apparently contributes finer details to higher frequencies most notably in the range of 1,000 Hz to 4,000 Hz. Nevertheless, post-processing further contributes to improving the result by repairing artifacts and non-continuities in the linear spectrogram. Condensing multiple frames during decoding using the reduction factor, efficiently reduces the number of decoder steps required. However, a side effect are artifacts introduced between the transitions of the condensed frame groups can be seen in figure 22a. Figure 22b shows, that such artifacts are fixed or at least moderated by the post-processing stage.

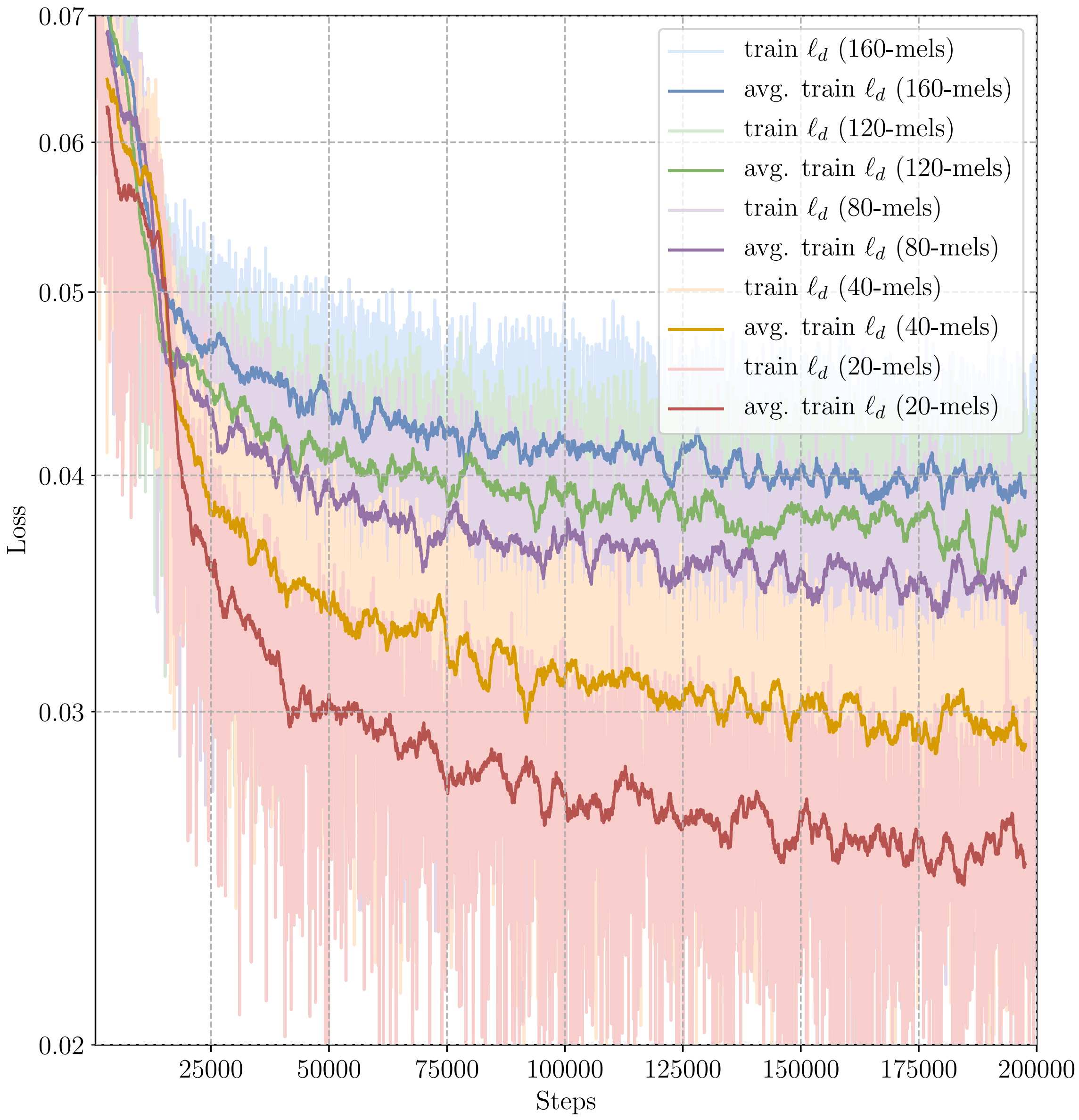


Figure 19: Progression of the decoder loss during training for models using different amounts of mel banks as the decoder target. All models are trained on the Blizzard dataset and the losses are plotted every 50 steps. A moving average is calculated for all training plots using a window width of 50 and a stride of 1. The average is only calculated when the window completely overlaps with the signal. For better scaling the first 1,000 steps are omitted.

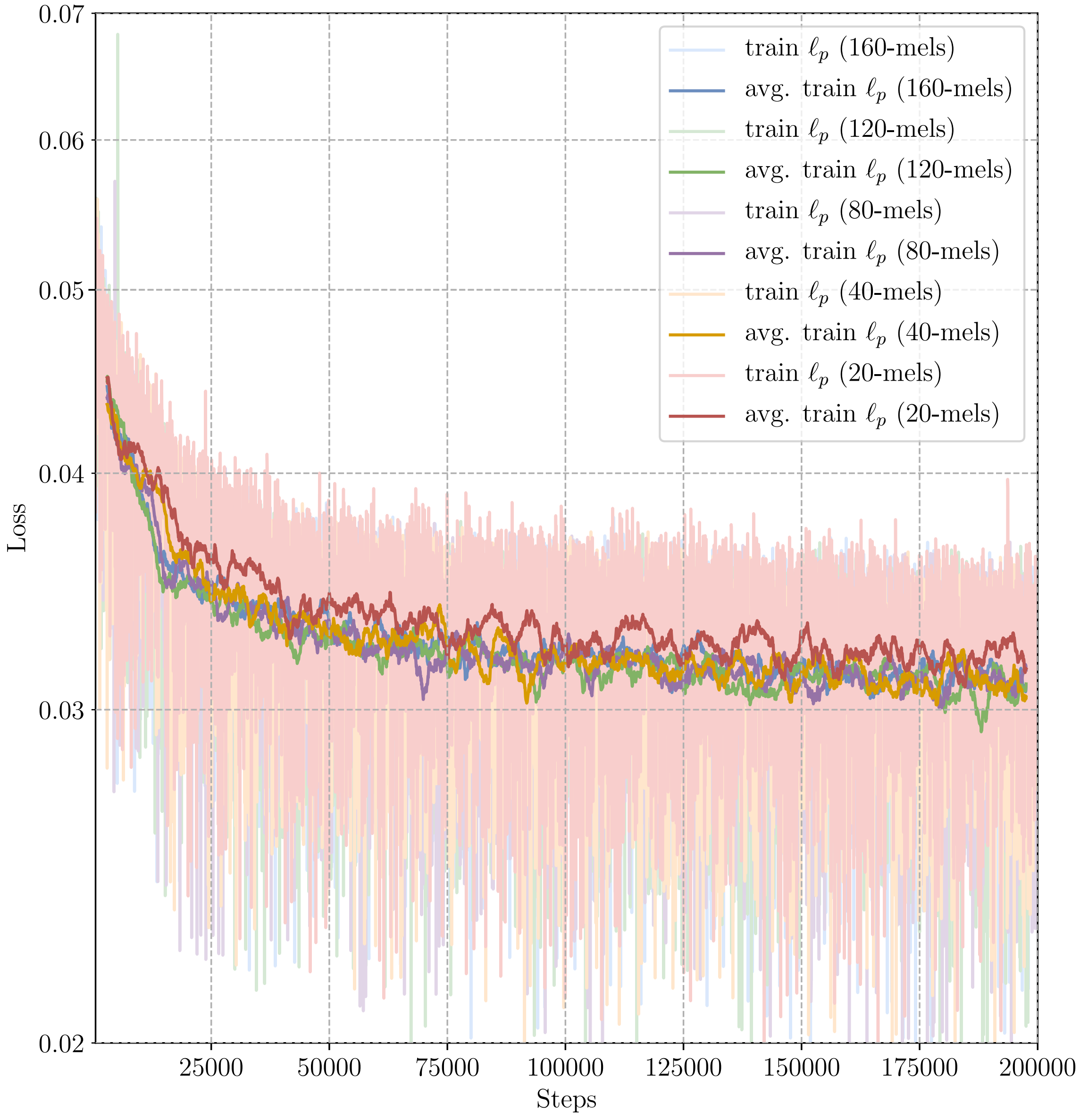


Figure 20: Progression of the post-processing loss during training for models using different amounts of mel banks as the decoder target. All models are trained on the Blizzard dataset and the losses are plotted every 50 steps. A moving average is calculated for all training plots using a window width of 50 and a stride of 1. The average is only calculated when the window completely overlaps with the signal. For better scaling the first 1,000 steps are omitted.

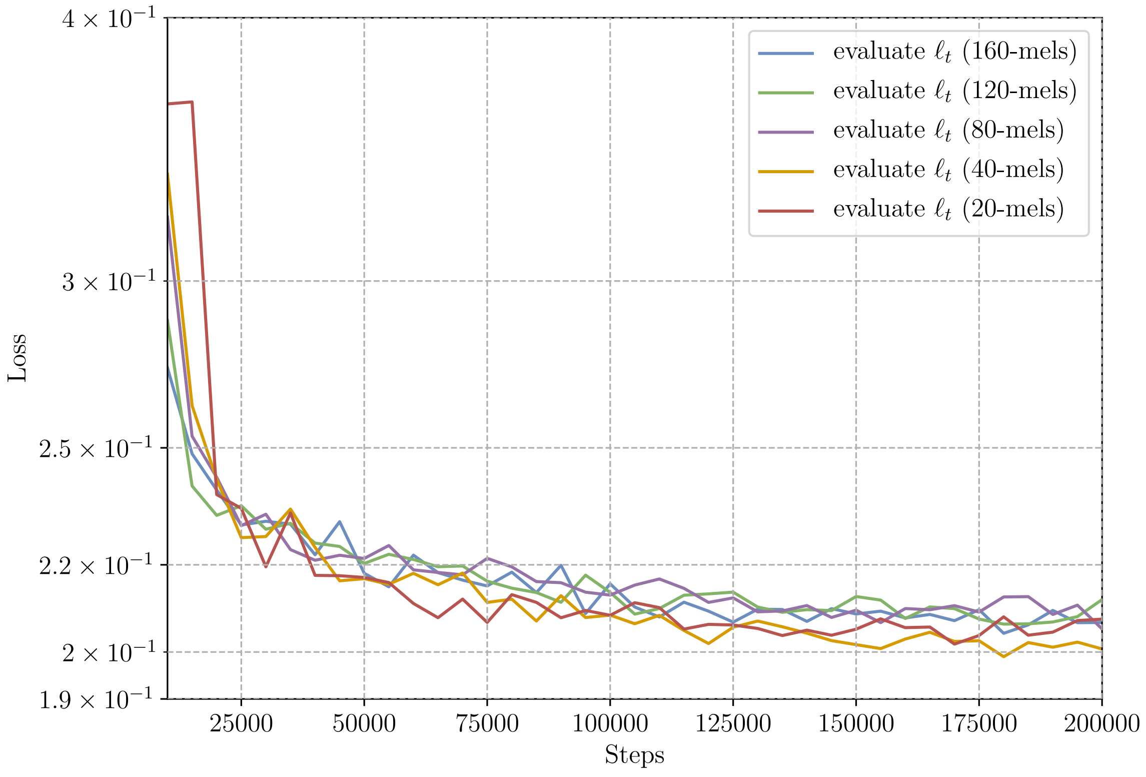
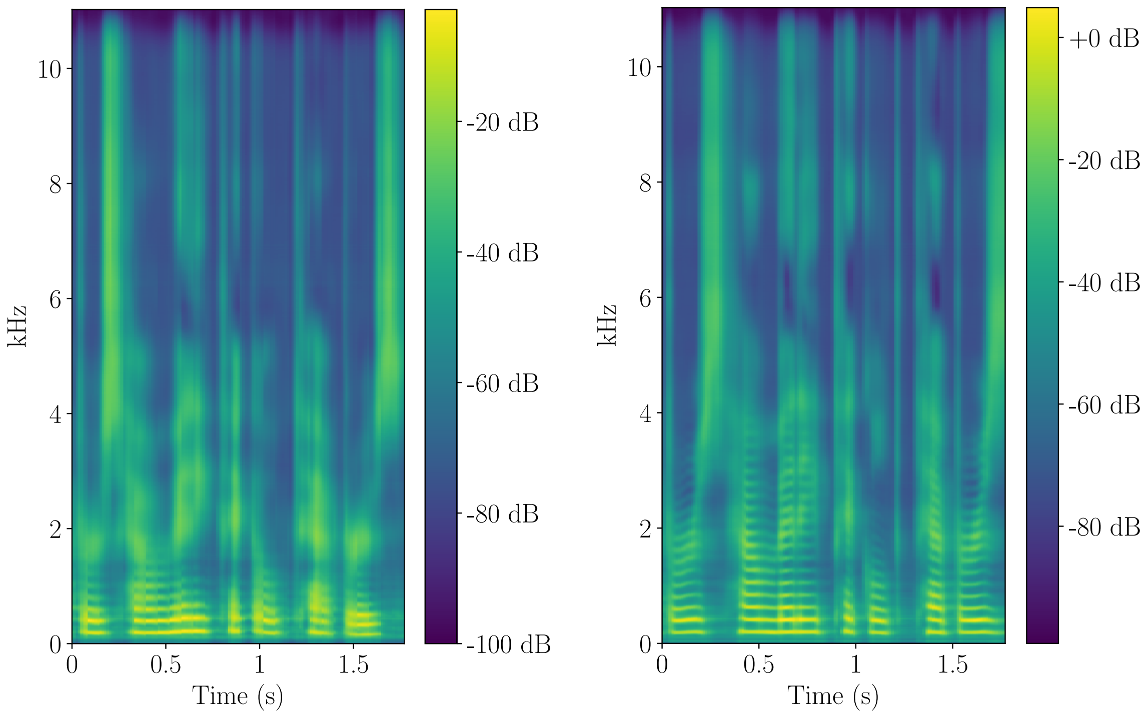


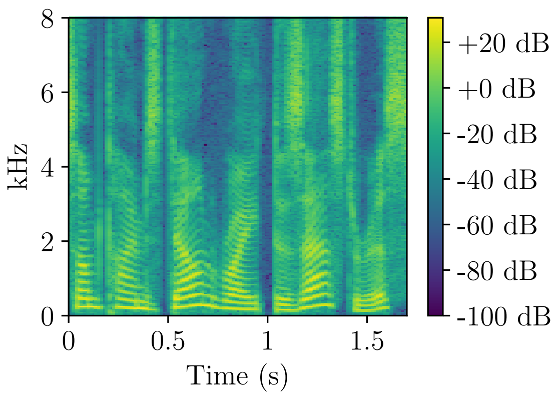
Figure 21: Progression of the total evaluation loss for models using different amounts of mel banks as the decoder target. The losses are plotted every 5,000 steps. The first 10,000 steps are skipped for better scaling.



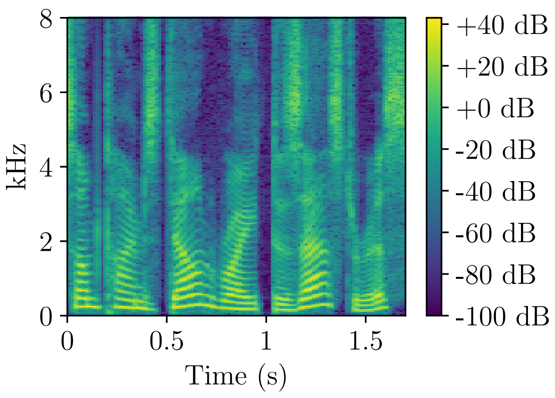
(a) Linear scale magnitude spectrogram without the post-processing applied.

(b) Linear scale magnitude spectrogram with the post-processing applied.

Figure 22: Comparison of the predicted linear scale spectrogram with and without the post-processing network applied. Both spectrograms are generated by separate models trained on the LJ Speech dataset for an utterance selected randomly from the test portion.



(a) Linear scale magnitude spectrogram without modification of the magnitudes.



(b) Linear scale magnitude spectrogram with the magnitudes raised to the power of 1.3.

Figure 23: Effect of raising the linear scale magnitude spectrogram to a power of 1.3. Using a part of the utterance "sometimes downright disastrous, decisions." (APDC2-008-03) from the Blizzard dataset as an example. Raising the magnitudes does effectively sharpen the spectrogram, dampening homogeneous noisy areas.

Observe that there is a noticeable translation in timing and variation in the pauses produced. To a certain degree this behavior is expected as the timing and overall duration are derived from plain text.

6.6 Spectrogram Inversion

Prior to synthesis using Griffin-Lim the magnitudes of the linear scale spectrogram are raised to the power of 1.3.

6.6.1 Results

Figure 23 illustrates the effect of raising the linear scale magnitude spectrogram to a power. With an increasing power one can observe that the higher frequency components vanish. Also, homogeneous low power regions in the spectrogram are scaled down.

6.6.2 Discussion

Subjective listening tests on individual files during training indicates, that this sharpening of the spectrogram effectively dampens noise. While, the spectrogram gets "sharpened" with an increasing power, the higher frequency components also tend to vanish. This appears to lead to a muffled and overall dampened voice that subjectively sound unclear. Note that the exponent 1.3 is chosen based on testing and might be selected different for each dataset for the best results.

As indicated by figure 14b this processing stage is particularly demanding. While improvements to the algorithm are possible, computation time spend in this stage is substantially higher than that spend in the prior stages combined.

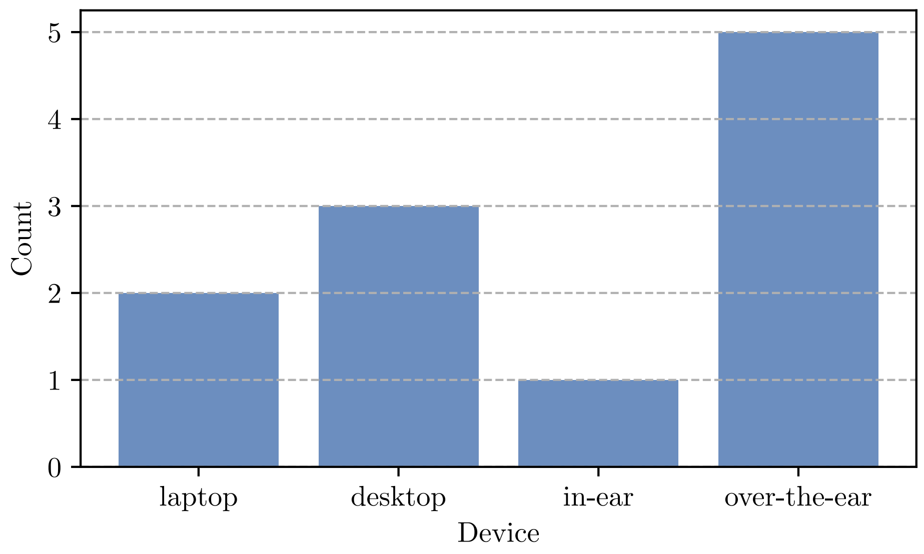


Figure 24: Distribution of the playback devices used in the survey. Before the participants hear samples, their playback configuration is queried from a list of four categories. While participants are encouraged to use headphones, it is not required for attending. A total of 11 participants attended the survey.

6.7 Evaluation Survey

In order to rank the proposed architecture in contrast to other systems an evaluation survey is conducted. It aims to compare the quality of different TTS systems to get an idea of how the proposed system relates. It is not supposed to be a statistically representative evaluation.

The model's performance is evaluated using a paired comparison MOS test in combination with a hypothesis test at a significance level of $\alpha = 0.05$. For more details on the survey design and implementation and the chosen null hypothesis refer to appendix section A.

6.7.1 Results

In total 11 participants of age 21 to 38 attend the survey. Out of these, 1 participant is female and 10 are male. All listeners are native German speakers from the domain of computer science. The subjects are presented pairs of audio files containing the same utterance generated by the proposed architecture and two other TTS systems. The subjects are asked to rate the naturalness of each file using a 5-point MOS. The order in which the models are presented is altered randomly with each sentence.
Namely, *MaryTTS* [54] (unit-selection) and the *Merlin* system [55] (statistical-parametric) are used as reference systems. All models are trained on the same English sentences from the Blizzard dataset (see section 5.2) to maximize comparability. Six attendants used headphones and five attendants used either laptop or desktop speakers. See figure 24 for a detailed visualization of the distribution of playback devices. Table 3 lists the calculated MOSs and the sample standard deviations. Refer to figure 25 for a visualization of the scores. Table 4 lists the test statistics and the p-values for both reference models in comparison to the proposed model.

Table 3: Results of the 5-point mean opinion score evaluation. Listed are the arithmetic mean MOSs ($\hat{\mu}_a$) and the uncorrected and corrected sample standard deviations ($\hat{\sigma}_{a,\Delta}$). All values are rounded to one decimal place.

Model (a)	MOS ($\hat{\mu}_a$)	STD ($\hat{\sigma}_{a,0}$)	STD ($\hat{\sigma}_{a,1}$)
MaryTTS	2.0	0.9	0.9
Merlin	3.3	0.8	0.8
Proposed	3.4	1.0	1.0

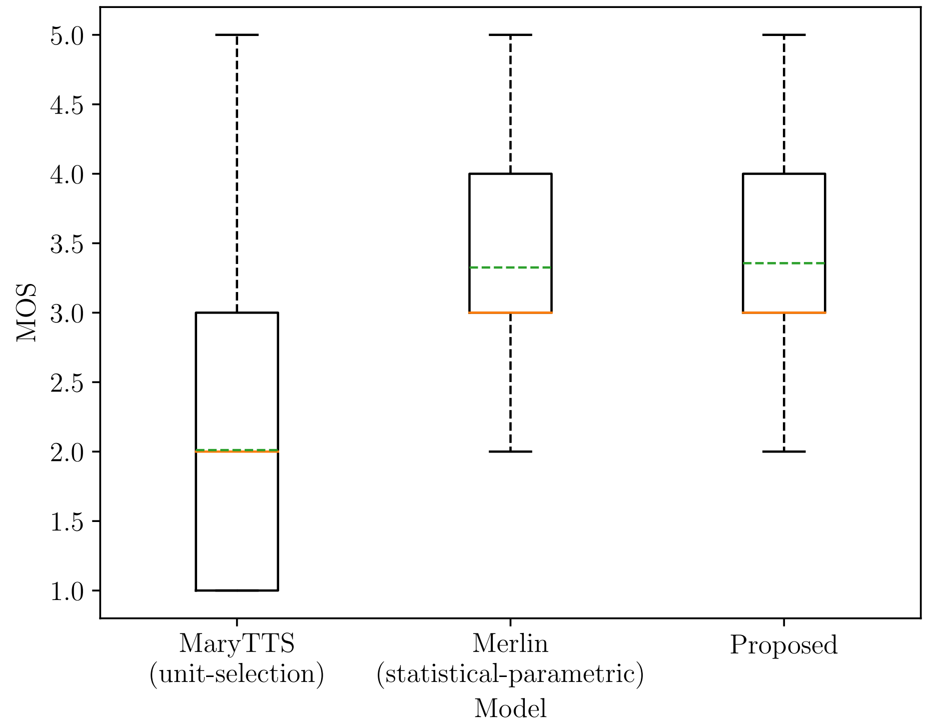


Figure 25: Results of the 5-point mean opinion score evaluation. The orange line depicts the median of the corresponding model ratings, while the green dashed line depicts the arithmetic mean. The lower box boundary illustrates the first quartile and the upper bound the third quartile. The black dashed whisker lines indicated the minimum and maximum ratings.

Table 4: Results for test statistics $t_{r,\Delta}$ and the p-values for the reference systems using different delta degrees of freedom. All values are rounded to two decimal places.

Model (r)	$t_{r,0}$	$t_{r,1}$	$p_{r,0}$	$p_{r,1}$
MaryTTS	-23.91	-23.89	0.00	0.00
Merlin	-0.58	-0.57	0.56	0.57

Table 5: Calculated confidence intervals $\text{conf}_{a,\Delta}$ for all models. All values are rounded to one decimal place.

Model (a)	$\text{conf}_{a,0}$	$\text{conf}_{a,1}$
MaryTTS	$(1.9, 2.1)$	$(1.9, 2.1)$
Merlin	$(3.3, 3.4)$	$(3.3, 3.4)$
Proposed	$(3.3, 3.4)$	$(3.3, 3.4)$

6.7.2 Discussion

Let's contemplate the underlying null hypothesis devised in section A.1.3:

> *There is no significant difference between the MOS from the reference models and that of the proposed model.*

Table 4 lists the test statistics and the p-values for both reference models in comparison to the proposed model. Note that the influence of delta degrees of freedom Δ is negligible for the decision on the null hypothesis. Considering the p-value for $\Delta = 0$ of the MaryTTS model, the null hypothesis H_0 has to be rejected $(0.00 \leq 0.05)$. Consequently, there exists a significant difference in terms of MOS between the MaryTTS and the proposed model at the $\alpha = 0.05$ level. However, when considering the p-value for $\Delta = 0$ of the Merlin model, rejecting the null hypothesis H_0 fails $(0.56 > 0.05)$. Consequently, there exists no significant difference in terms of MOS between the Merlin and the proposed model at the $\alpha = 0.05$ level.

Using participants that are not native speakers may result in slightly better scores. The attendants might not be capable of distinguishing and discriminating small errors in the speech as well as a native speaker could.

For evaluation 50 sentences not contained in the training dataset are used. Therefore, the scores are not normalized against ground-truth recordings from the dataset. For the calculation of the final scores all ratings are used, independent of the playback device used. As described in [56], participants using loudspeakers generally have smaller discrimination capacity than users wearing headphones. However, one cannot require participants to always wear headphones, especially in an environment like an online survey.

With ten males to one female, the male participants are over-represented in the collected data. Hence, the random sampling assumptions made in section A.2.2 for the hypothesis test are not optimal. Compensating for this discrepancy by weighting the data based on gender is renounced. It is presumed that the data is sufficient to relate the models against each other.

Considering the achieved scores one can notice the unit-selection (*MaryTTS*) system performs poorly compared to the other systems. This may be due to the training data not containing enough material to always find matching snippets for concatenation.

7 Conclusions

Traditional speech generation approaches are based on complex multi-stage pipelines. Often modeling a multitude of properties of written and spoken language as well as their interplay (section 1.3). Neural generative speech synthesis models and the originating possibility of end-to-end fashion model training might simplify the creation of TTS systems.

7.1 Summary

The objective of this work is to implement and evaluate a neural voice synthesis solution that is inspired by current state-of-the-art architectures. Capable of training from unaligned text-audio-pairs in an end-to-end fashion, making little assumptions on the data.

In order to be able to classify the proposed architecture, a general understanding of common TTS systems and their structure is established. To understand the system itself a basic knowledge of the underlying neural technologies is established. Before introducing the architecture a brief overview of related work is presented, focusing primarily on recent neural TTS systems. The proposed model is directly derived from the Tacotron [1] architecture. It can be trained in an end-to-end fashion from unaligned text-audio-pairs without the need for supplementary alignment or duration models. It is composed of a separate encoder, decoder, post-processing and a synthesis stage. These abstract stages are decomposed into smaller components that are introduced separately. Using them as building blocks for the more complex stages. The encoder transforms written text into variable length embeddings. The subsequent decoder evaluates these embeddings using the Luong attention mechanism. With each decoding iteration it produces multiple non-overlapping MSMSPEC frames. The decoded MSMSPEC is improved by the neural post-processing stage, converted to linear scale and turned into a waveform using the Griffin-Lim algorithm.

Different models of the same architecture are trained on two different freely available English datasets. Each dataset consists of unaligned text-audio-pairs of different quantity and quality. The capabilities and components of the resulting models are analyzed and evaluated. A paired comparison test using human listeners is conducted to evaluate the naturalness of the produced speech on a 5-point MOS scale. The proposed system is compared against two publicly available TTS systems namely *MaryTTS* (unit-selection) and *Merlin* (statistical-parametric). The *MaryTTS* model scores 2.0, the *Merlin* model scores 3.3 and the proposed system achieves 3.4, on average. A paired difference hypothesis test shows that The *Merlin* show no significant difference in terms of MOS at a significance level of $\alpha = 0.05$. Demonstrating that the proposed architecture is capable to train in an end-to-end fashion and is able to produce comprehensible speech. In the process predicting robust and precise attention alignments using the implemented Luong attention mechanism.

Investigations on the models components are carried out. Endless repetition at the end of sentences can efficiently be prevented by not masking padding when calculating the loss. During decoding particularly the feeding of the ground-truth frames seems to help the models to train. Therefore, enabling the decoder to train right from the start when even the encoder or the post-processing are still unable to fit the data. Combined with the reduction factor reducing the number of decoder steps required, the models are able to reliably predict long sequences. Visualizing the effect of the post-processing stage clarified that it is essential for improving the resolution of higher frequencies in the produced spectrogram. Raising the predicted linear scale spectrogram to a power helps to remove noise and improves the perceived quality. With increasing power, higher frequencies vanish and the result can be perceived as muffled. The results suggest, that it is not primarily the amount of data that is vital. Rather, the overall quality of the data is crucial. Especially a neutral voice, homogeneous speaking rate as well as no random silence at the beginning of the waveforms are important.

Furthermore, the model behavior for different amounts of mel filter banks generated with each decoder iteration is analyzed. Using a more detailed mel spectrogram for training, one might anticipate a better result. However, increasing the number of banks leads to an increase in loss, while reducing the number of banks reduces the loss. While the decoder loss reduces, the post-processing loss remains consistent. Implying that predicting the post-processing target does not suffer from the reduced decoder target size. Hence, either the number of parameters in the decoder RNN cells has to increase, or another way of transforming mel spectrograms into linear scale spectrograms has to be devised.

7.2 Limitations And Practical Implications

Despite the models accomplishments this architecture has several limitations. The model is not capable of producing speech at a pace that could be considered real time. While the neural stages are capable of doing so it is primarily the computational cost of the Griffin-Lim synthesis approach preventing this. The used loss function is suitable for training, however, in the advanced stages of training it seems unable to capture the models progress. Hence, it is probably unable to capture all the nuances of the generated voice. For example there are small displacements and pauses that the loss does not account for. Further more, while the end-to-end fashion training produces working models capable of producing speech, it lacks direct control mechanisms to influence the result. Properties like pronunciation, emotion or the talking speed for example are solely approximated from the training data used.

Nevertheless, the results clearly show that a system like this enables the generation of high quality speech even without extensive domain knowledge or huge datasets. Making the creation of such systems less complex could lead to general improvements in the quality and availability of TTS solutions. That might have implications in all kinds of currently available TTS systems. Particularly synthesis solutions that do not require powerful hardware or an active internet connection might become feasible. Allowing the creation of more complex, interactively reacting content in the entertainment sector. For example video games might increase immersion by individually reacting on the users actions. Having systems that can be trained on little data with all kinds of speech might also make rebuilding of voices possible. Benefiting the creation of high quality personalized voices for patients who have lost their voice due to medical conditions requiring tracheostomy[20] for example.

[20]Surgical procedure that allows a person to breathe without the use of the nose or mouth: https://en.wikipedia.org/wiki/Tracheotomy

7.3 Future Work

The proposed architecture is directly derived from *Tacotron* [1], which in turn directly influenced the development of current state-of-the-art architectures [39, 40]. As such, the current implementation can be extended to recreate these more sophisticated neural architectures. This in turn facilitates the exploration of applications like multi-speaker generation, active prosody modeling, improved neural vocoders, voice conversion and modeling of emotions.

Considering the lack of control over the generated result with the current architecture it is desirable to have the possibility to actively model prosody. Most synthetic voices produce voice with a neural style of speaking, having more control over the prosody might improve the naturalness and expressiveness. This might also help produce more consistent pauses and timing such that the speech is better captured by the current loss function.

Likewise, of interest might be the model's dependency on data from a single speaker as there are only limited amounts of data available. It would be desirable to develop an architecture that is capable of training from data of multiple speakers. Ideally complementing each other, while being able to produce voice for each of the individual speaker.

Note that an approach capable of capturing the characteristics of multiple speakers might also be utilized to transform speech between the characteristics of different speakers. This could allow one to take language from one speaker and transform it, so that it sounds like another. In turn being able to capture such characteristics might also allow the modeling of emotional speech.

Apart from the application of the concept to other problems, there are other refinements that can be addressed in the current architecture. As mentioned above, its primary limitation is the Griffin-Lim based synthesis stage. Replacing it with a neural vocoder like WaveNet [36] as demonstrated in Tacotron 2 [39] could improve the results. Besides the vocoder, exploring the potential of other features and in turn loss functions might be beneficial. Promising alternative features to explore are Mel-cepstral coefficients (MCEPs) [57] and Mel-frequency cepstral coefficients (MFCCs) [58]. As shown by the results in section section 6, the loss function seems to be unable to capture all nuances of the produced voice. Alternative loss functions to explore might be per frame Mel-cepstral distortion (MCD) or Modulation Spectrum Log-Spectral Distortion (MS-LSD) [59]. Further, exploring the effects of using different reduction factors might be beneficial. As there is a trade-off between shorter sequences with an increasing amount of artifacts and longer sequences with fewer artifacts. Fine-tuning of a model trained on one speaker to another is also worth investigating as it might allow deriving models in shorter time with less material.

Acknowledgments

The author conveys his tanks to Jürgen te Vrugt and Nikolaus Wulff for their constructive feedback and words of advice. Furthermore, he would like to express his acknowledgment to Markus Gilbert and Kathrin Ungru for the cooperation and assistance with the resources of the Steinfurt Campus Cluster (SCC) and the continuous support with the hardware. The author would also like to thank Marc Dangschat for his insightful comments and the fruitful discussions.

List of Figures

List of Tables

References

[1] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: Towards End-to-End Speech Synthesis", Mar. 2017. arXiv: 1703.10135. [Online]. Available: https://arxiv.org/abs/1703.10135.

[2] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks", Feb. 2013. arXiv: 1211.5063v2. [Online]. Available: https://arxiv.org/abs/1211.5063v2.

[3] P. Taylor, *Text-to-Speech Synthesis*, 1st. Cambridge University Press, 2009, p. 626, ISBN: 9780511816338. DOI: 10.1017/CBO9780511816338. [Online]. Available: http://ebooks.cambridge.org/ref/id/CBO9780511816338.

[4] R. Kaur, R. K. Sharma, and P. Kumar, "Building a Text-to-Speech System For Punjabi Language", *IT-CSCP 2017*, vol. 7, pp. 71–87, 2017. DOI: 10.5121/csit.2017.70806. [Online]. Available: https://airccj.org/CSCP/vol7/csit77006.pdf.

[5] M. H. O'Malley, "Text-to-speech conversion technology", *Computer*, vol. 23, no. 8, pp. 17–23, Aug. 1990, ISSN: 0018-9162. DOI: 10.1109/2.56867. [Online]. Available: https://ieeexplore.ieee.org/document/56867/.

[6] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning", Oct. 2015. arXiv: 1506.00019v4. [Online]. Available: https://arxiv.org/abs/1506.00019v4.

[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", Apr. 2016. arXiv: 1502.03044v3. [Online]. Available: https://arxiv.org/abs/1502.03044v3.

[8] M. Han, O. Wu, and Z. Niu, "Unsupervised Automatic Text Style Transfer using LSTM", in *6th CCF International Conference, NLPCC 2017*, X. Huang, J. Jiang, D. Zhao, Y. Feng, and Y. Hong, Eds., Beijing Institute of Technology, Dalian, China: Springer International Publishing AG, 2017, pp. 281–292. DOI: 10.1007/978-3-319-73618-1. [Online]. Available: http://tcci.ccf.org.cn/conference/2017/papers/1135.pdf.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks", in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 3104–3112. arXiv: 1409.3215v3. [Online]. Available: https://arxiv.org/abs/1409.3215v3.

[10] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation", Sep. 2015. arXiv: 1508.04025v5. [Online]. Available: https://arxiv.org/abs/1508.04025v5.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", *ICLR 2015*, 2016. arXiv: 1409.0473v7. [Online]. Available: https://arxiv.org/abs/1409.0473v7.

[12] J. Lee, K. Cho, and T. Hofmann, “Fully Character-Level Neural Machine Translation without Explicit Segmentation”, in *Transactions of the Association for Computational Linguistics*, A. Lopez, Ed., vol. 5, Jun. 2017, pp. 365–378. arXiv: 1610.03017v3. [Online]. Available: https://arxiv.org/abs/1610.03017v3.

[13] S. Kubrick, *2001: A Space Odyssey*, 1968. [Online]. Available: https://www.imdb.com/title/tt0062622/.

[14] G. Roddenberry, *Star Trek*, 1966. [Online]. Available: https://www.imdb.com/title/tt0060028/?ref_=nv_sr_6.

[15] S. Hawking, *The Computer - Stephen Hawking*, 2017. [Online]. Available: http://www.hawking.org.uk/the-computer.html (visited on 06/25/2018).

[16] Apple Inc., *iOS - Siri - Apple*, 2011. [Online]. Available: https://www.apple.com/ios/siri/ (visited on 06/25/2018).

[17] Google LLC., *Google Assistant*, 2016. [Online]. Available: https://assistant.google.com/#?modal_active=none (visited on 06/25/2018).

[18] Amazon.com Inc., *Amazon Echo*, 2016. [Online]. Available: https://developer.amazon.com/echo (visited on 06/25/2018).

[19] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, “SampleRNN: An Unconditional End-to-End Neural Audio Generation Model”, *ICLR 2017*, Feb. 2017. arXiv: 1612.07837v2. [Online]. Available: https://arxiv.org/abs/1612.07837v2.

[20] N. S. Krishna, P. P. Talukdar, K. Bali, and A. G. Ramakrishnan, “Duration Modeling for Hindi Text-to-Speech Synthesis System”, in *INTERSPEECH 2004*, Jeju Island, Korea, 2004, pp. 789–792. [Online]. Available: http://talukdar.net/papers/icslp04_hpl.pdf.

[21] J. Sotelo, S. Mehri, K. Kumar, J. Santos, K. Kastner, A. Courville, and Y. Bengio, “Char2Wav: End-to-End Speech Synthesis”, *ICLR 2017 Workshop*, 2017. [Online]. Available: https://openreview.net/pdf?id=B1VWyySKx.

[22] T. Dutoit, *An Introduction to text-to-speech synthesis*, 1999. [Online]. Available: http://tcts.fpms.ac.be/synthesis/introtts_old.html (visited on 07/06/2018).

[23] D. H. Klatt, “Interaction between two factors that influence vowel duration”, *The Journal of the Acoustical Society of America*, vol. 54, no. 4, pp. 1102–1104, 1973. DOI: 10.1121/1.1914322. [Online]. Available: http://asa.scitation.org/doi/10.1121/1.1914322.

[24] A. W. Black, H. Zen, and K. Tokuda, “Statistical Parametric Speech Synthesis”, *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP ’07*, vol. 4, pp. 1229–1232, 2007. DOI: 10.1109/ICASSP.2007.367298.

[25] M. Dangschat, *End-To-End Speech Recognition Using Connectionist Temporal Classification*, 2018.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[27] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. arXiv: 1406.1078v3. [Online]. Available: https://arxiv.org/abs/1406.1078v3.

[28] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://ieeexplore.ieee.org/document/6795963/.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[30] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., vol. 37, Lille, France: PMLR, 2015, pp. 448–456. arXiv: 1502.03167v1. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.pdf.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[32] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Simultaneous Modeling Of Spectrum, Pitch And Duration In HMM-Based Speech Synthesis", in *EUROSPEECH'99*, Budapest, Hungary, 1999, pp. 2347–2350. [Online]. Available: https://www.isca-speech.org/archive/archive_papers/eurospeech_1999/e99_2347.pdf.

[33] D. H. Klatt, "Review of text-to-speech conversion for English", *The Journal of the Acoustical Society of America*, vol. 82, no. 3, pp. 737–793, Sep. 1987, ISSN: 0001-4966. DOI: 10.1121/1.395275. [Online]. Available: https://doi.org/10.1121/1.395275.

[34] A. J. Hunt and A. W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database", in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, Atlanta, GA, USA: IEEE, 1996, pp. 373–376. DOI: 10.1109/ICASSP.1996.541110.

[35] A. W. Black, "Unit Selection and Emotional Speech", in *EUROSPEECH 2003*, Geneva, Switzerland: ISCA, 2003, pp. 1649–1652.

[36] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio", Sep. 2016. arXiv: 1609.03499v2. [Online]. Available: https://arxiv.org/abs/1609.03499v2.

[37] S. Ö. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, S. Sengupta, and M. Shoeybi, "Deep Voice: Real-time Neural Text-to-Speech", in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., vol. 70, PMLR, Aug. 2017,

pp. 195–204. arXiv: 1702.07825v2. [Online]. Available: http://proceedings.mlr.press/v70/arik17a/arik17a.pdf.

[38] M. Morise, F. Yokomori, and K. Ozawa, "WORLD: A Vocoder-Based High-Quality Speech Synthesis System for Real-Time Applications", *IEICE Transactions on Information and Systems*, vol. E99.D, no. 7, pp. 1877–1884, Jul. 2016, ISSN: 0916-8532. DOI: 10.1587/transinf.2015EDP7457.

[39] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. J. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", Feb. 2018. arXiv: 1712.05884v2. [Online]. Available: https://arxiv.org/abs/1712.05884v2.

[40] S. Ö. Arik, G. Diamos, A. Gibiansky, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou, "Deep Voice 2: Multi-Speaker Neural Text-to-Speech", in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Long Beach, CA, USA: Curran Associates, Inc., Sep. 2017, pp. 2962–2970. arXiv: 1705.08947v2. [Online]. Available: https://arxiv.org/abs/1705.08947v2.

[41] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning", Feb. 2018. arXiv: 1710.07654v3. [Online]. Available: https://arxiv.org/abs/1710.07654v3.

[42] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks", 2015. arXiv: 1505.00387v2. [Online]. Available: https://arxiv.org/abs/1505.00387v2.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Dec. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[44] D. L. Sun and J. O. Smith, "Estimating a Signal from a Magnitude Spectrogram via Convex Optimization", in *133rd AES Convention*, San Francisco, CA, USA: Audio Engineering Society, Inc., Oct. 2012. arXiv: 1209.2076v1. [Online]. Available: https://arxiv.org/abs/1209.2076v1.

[45] D. W. Griffin and J. S. Lim, "Signal Estimation from Modified Short-Time Fourier Transform", in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, IEEE, 1984, pp. 236–243. DOI: 10.1109/TASSP.1984.1164317.

[46] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-Aware Neural Language Models", in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, Arizona: AAAI Press, Feb. 2016, pp. 2741–2749. arXiv: 1508.06615v4. [Online]. Available: https://arxiv.org/abs/1508.06615v4.

[47] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", Oct. 2016. arXiv: 1609.08144v2. [Online]. Available: https://arxiv.org/abs/1609.08144v2.

[48] C. Veaux, J. Yamagishi, and K. MacDonald, *CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit*, 2012. [Online]. Available: https://datashare.is.ed.ac.uk/handle/10283/2651 (visited on 09/15/2018).

[49] SynSIG, *Blizzard Challenge 2011*, 2011. [Online]. Available: https://www.synsig.org/index.php/Blizzard_Challenge_2011 (visited on 07/12/2018).

[50] K. Ito, *The LJ Speech Dataset*, 2017. [Online]. Available: https://keithito.com/LJ-Speech-Dataset/.

[51] S. O. Ternström, "Hi-Fi voice: observations on the distribution of energy in the singing voice spectrum above 5 kHz", *The Journal of the Acoustical Society of America*, vol. 123, no. 5, pp. 3379–3379, May 2008. DOI: 10.1121/1.2934016.

[52] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, Y. Whye and M. Titterington, Eds., vol. 9, Sardinia, Italy: PMLR, 2010, pp. 249–256. DOI: 10.1.1.207.2059.

[53] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", in *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, May 2015. arXiv: 1412.6980v9. [Online]. Available: https://arxiv.org/abs/1412.6980v9.

[54] S. Le Maguer and I. Steiner, "The MaryTTS entry for the Blizzard Challenge 2016", in *Blizzard Challenge*, Cupertino, CA, USA, Sep. 2016. [Online]. Available: http://festvox.org/blizzard/bc2016/MARYTTS_Blizzard2016.pdf.

[55] Z. Wu, O. Watts, and S. King, "Merlin: An Open Source Neural Network Speech Synthesis System", in *9th ISCA Speech Synthesis Workshop (SSW9)*, Sunnyvale, CA, USA, Sep. 2016, pp. 202–207. DOI: 10.21437/SSW.2016-33. [Online]. Available: http://www.isca-speech.org/archive/SSW_2016/abstracts/ssw9_PS2-13_Wu.html.

[56] F. Ribeiro, D. Florêncio, C. Zhang, and M. Seltzer, "CROWDMOS: An approach for crowdsourcing mean opinion score studies", in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, Czech Republic: IEEE, 2011, pp. 2416–2419. DOI: 10.1109/ICASSP.2011.5946971.

[57] S. Imai, "Cepstral analysis synthesis on the mel frequency scale", in *ICASSP '83. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 8, Institute of Electrical and Electronics Engineers, pp. 93–96. DOI: 10.1109/ICASSP.1983.1172250.

[58] A. F. Abka and H. F. Pardede, "Speech recognition features: Comparison studies on robustness against environmental distortions", in *2015 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, IEEE, Oct. 2015, pp. 114–119, ISBN: 978-1-4799-8773-3. DOI: 10.1109/IC3INA.2015.7377757.

[59] S. Takamichi, T. Toda, A. W. Black, G. Neubig, S. Sakti, and S. Nakamura, "Postfilters to Modify the Modulation Spectrum for Statistical Parametric Speech Synthesis", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 4, pp. 755–767, Apr. 2016, ISSN: 2329-9290. DOI: 10.1109/TASLP.2016.2522655.

[60] E. H. Rothauser, W. D. Chapman, N. Guttman, H. R. Silbinger, M. H. L. Hecker, G. E. Urbanek, K. S. Nordby, and M. Winstock, “IEEE Recommended Practice for Speech Quality Measurements”, *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 3, pp. 225–246, 1969. DOI: 10.1109/TAU.1969.1162058.

Appendices

A Evaluation

A quality survey is conducted to assess the performance of the proposed architecture. The objective is to asses how the Blizzard model (section 6.1) ranks in a direct comparison against two other models. This is not a statistically representative study, but a small scale informal estimation. The concrete question is: Is there a significant difference between the Mean Opinion Scores (MOSs) of the reference models compared to the proposed model?

The remainder of this section is organized as follows: Section A.1 introduces the MOS scale as well as the reference systems. Subsequently, it formulates the null hypothesis that is tested using the survey in a paired difference hypothesis test. Section A.2 characterizes the approach and describes how the hypothesis test and the collected ratings are analyzed. The results are presented and discussed in the main thesis in section 6.7.

A.1 Survey Design

Evaluating a Text-to-Speech (TTS) system involves human perception, this is commonly captured using subjective listening tests [1, 39, 37, 40]. In this evaluation naturalness is measured by conducting paired comparison tests assigning MOSs (see section A.1.1). Comparing the performance of the Blizzard model discussed in section 6.1 against two other reference TTS systems. Namely, the *Merlin* toolkit [55] and the *MaryTTS* system [54] (see section A.1.2). The reference systems are trained on the same subset of the Blizzard dataset to ensure comparability of the results.

The study is conducted on a small scale with unpaid volunteers answering questions on a website. Attendants are allowed to participate independently of the used hardware or their location. While this is not a controlled environment with specific audio characteristics, it is sufficient for assessing how the models relate to each other.

Contrary to some publications mentioned above, sentences used in the evaluation are not selected from the test portion of the dataset. Tacotron and Tacotron 2 for example used 100 randomly selected unseen sentences from their internal non-public datasets [1, 39]. Due to the limited data available, this survey does not use sentences from the dataset. Note that as a consequence listeners are not presented with an original sample as reference. Hence, the ratings are not calibrated against a ground-truth.

For evaluation, 50 sentences not contained in the dataset are selected. Refer to appendix section C for a listing. They consist of sentences with heteronyms[21], tongue twisters and a subset of the *Harvard Sentences* [60]. The idea behind selecting utterances with heteronyms is that while models likely do not capture the meaning of the sentence, they have to produce different pronunciations. Ideally producing varying prosody for the same words based on the surrounding context. Additionally, the tongue twisters are included to test the models ability to handle unusual sentences. This is supposed to stress the attention mechanism, to make sure it is robust. Finally, a small subset of the *Harvard Sentences* is also included. They are selected because they consist of phonetically balanced standardized phrases with

[21] Words that have a different pronunciation and meaning from another but the same spelling.

the phoneme frequencies resembling that of conventional English speech [60].

A.1.1 Mean Opinion Scores

Naturalness is often assessed using MOS were each listener indicates their opinion on a 5-point scale from 1 (bad) to 5 (excellent). [3, pp. 536 sq.] Table 6 lists the individual levels of the used scale.

Table 6: Overview of the 5-point MOS scale used to measure the naturalness of speech.

Score	Quality of the Speech	Naturalness
5	Excellent	Completely natural
4	Good	Mostly natural
3	Fair	Equally natural and unnatural
2	Poor	Mostly unnatural
1	Bad	Completely unnatural

Note that it is difficult to specifically determine what the listeners are actually assessing. As hinted by Taylor, listeners generally tend to rate how much they like a system. He reckons that this kind of scale is probably best suited for ranking tests, like the one carried out in this thesis. [3, pp. 536 sq.]

A.1.2 Reference Models

The reference systems are trained on the same training subset of Blizzard dataset to ensure comparability of the results.

Merlin Model (statistical parametric):

Merlin is an open-source Neural Network (NN) based speech synthesis system for statistical parametric speech synthesis. *Merlin* is also the neural benchmark system used in the 2016 *Blizzard Challenge*. [55]

For this survey a model using *Festival*[22] as the front-end text processor and the open-source *WORLD*[23] vocoder is trained.

MarryTTS Model (unit selection):

MaryTTS is an open-source TTS synthesis platform that supports building voices using unit selection and Hidden Markov Models (HMMs). [54]

For this survey the default configuration of *MaryTTS* based on the unit selection paradigm is used. No manual tuning of the selection cost function or weights for prosodic and phonological target features is carried out.

A.1.3 Null Hypothesis

Let $\bar{\mu}_r$, $\bar{\mu}_b$ denote the actual MOS of the population for a reference system $r \in \{\text{MaryTTS}, \text{Merlin}\}$ and the proposed model $b \in \{\text{Proposed}\}$. The significance level is chosen as $\alpha = 0.05$. Then the null hypothesis $H_0 : \bar{\mu}_r - \bar{\mu}_b = 0$ is formulated as follows:

[22] Festival Speech Synthesis System: http://www.cstr.ed.ac.uk/projects/festival/
[23] WORLD vocoder: https://github.com/mmorise/World

There is no significant difference between the MOS from the reference models and that of the proposed model.

Therefore, if the null hypothesis is rejected (i.e. $\bar{\mu}_r - \bar{\mu}_b \neq 0$) there exists a significant difference in terms of the MOS between model a and model b. If rejecting the null hypothesis fails, (i.e. $\bar{\mu}_r - \bar{\mu}_b = 0$), there is no significant difference between them.

A.2 Methodology

The survey is conducted in an uncontrolled environment, hence information on the setup is collected to ensure the results can be put into perspective. Prior to starting the evaluation each user is asked if he is using in-ear headphones, over-the-ear headphones, desktop speakers or laptop speakers. This procedure is adopted from the *crowdMOS* [56] approach. As described in [56], workers using loudspeakers generally have smaller discrimination capacity than users wearing headphones. While participants are asked to use headphones during the test, this configuration is not enforced. With this metadata and the knowledge of the influence of the playback device it is possible to better assess the results.

During the survey each participant rates samples from a pool of 50 unique sentences that are the same for all listeners. For each sentence the listener is presented with three audio files, each spoken by one of the evaluated systems. The order in which the samples are presented is randomized with each utterance in order to prevent the users from predicting their position and sticking to a pattern. In total each participant rates 150 audio files with a duration of 2 to 10 seconds each.

A.2.1 Score Estimation

In total $A = 3$ algorithms are evaluated using $S = 50$ sentences. As all sentences are spoken by each algorithm resulting in $A * S$ ratings for each of the P participants. It is ensured that each listener rates all samples. For each algorithm a, with $a \in \{\text{MaryTTS}, \text{Merlin}, \text{Proposed}\}$, its mean rating score $\bar{\mu}_a$ needs to be estimated. Let $x^a_{s,p}$ be the score given to sentence s by participant p for algorithm a, with $1 \leq s \leq S$ and $1 \leq p \leq P$. Then the estimated mean score $\hat{\mu}_a \in \mathbb{R}$ is calculated as shown in equation (25).

$$\hat{\mu}_a = \frac{1}{SP} \sum_{p=1}^{P} \sum_{s=1}^{S} x^a_{s,p} \tag{25}$$

The standard deviation $\hat{\sigma}_{a,\Delta} \in \mathbb{R}$ for algorithm a is estimated as shown in equation (26), where $\Delta \in \{0, 1\}$ denotes the delta degrees of freedom.

$$\hat{\sigma}_{a,\Delta} = \sqrt{\frac{\sum_{p=1}^{P} \sum_{s=1}^{S} \left(x^a_{s,p} - \hat{\mu}_a\right)^2}{SP - \Delta}} \tag{26}$$

Supplementary all investigations are given using both the uncorrected ($\Delta = 0$) and the corrected ($\Delta = 1$) sample standard deviation.

A.2.2 Hypothesis Test

The tests rely on the standard normal distribution and make the following assumptions:

- The samples are sampled randomly.
- The population is normally distributed.

To determine if there is a difference in the population MOS between two models a paired difference test is conducted. As the difference between scores can be negative or positive a two-tailed test is implemented. The test statistic $t_{r,\Delta}$ between model $r \in \{\text{MaryTTS}, \text{Merlin}\}$ and the proposed model $b \in \{\text{Proposed}\}$ is calculated as shown in equation (27). With $N = SP$ and $\hat{\mu}_r$, $\hat{\mu}_b$ being the estimated algorithm MOSs (see section A.2.1). Additionally, $\bar{\mu}_r$, $\bar{\mu}_b$ denote the population MOSs defined with the null hypothesis H_0 (see section A.1.3).

$$t_{r,\Delta} = \frac{(\hat{\mu}_r - \hat{\mu}_b) - (\bar{\mu}_r - \bar{\mu}_b)}{\sqrt{\frac{\hat{\sigma}^2_{r,\Delta}}{N} + \frac{\hat{\sigma}^2_{b,\Delta}}{N}}} \tag{27}$$

Further, let $z^* = \Phi\left(1 - \frac{\alpha}{2}\right)$ be the critical value delimiting the rejection region, where Φ denotes the cumulative distribution function of the standard normal distribution. H_0 is rejected if: $t_{r,\Delta} \leq (-z^*)$ or $t_{r,\Delta} \geq z^*$. Contrary the test fails to reject H_0 if: $t_{r,\Delta} \in \left]-z^*, z^*\right[$.

The p-value for a given test statistic $t_{r,\Delta}$ is calculated as supplementary information as shown in equation (28) to provide a representation that is more approachable.

$$p_{r,\Delta} = 2\Phi\left(t_{r,\Delta}\right) \tag{28}$$

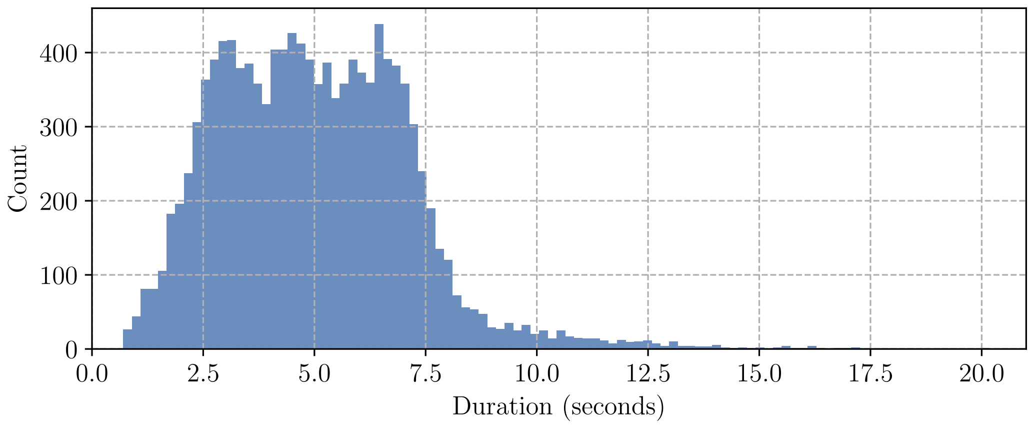
Consequentially using this representation H_0 is rejected if: $p_{r,\Delta} \leq \alpha$ smaller or equal to the significance level α.

A.2.3 Confidence Intervals

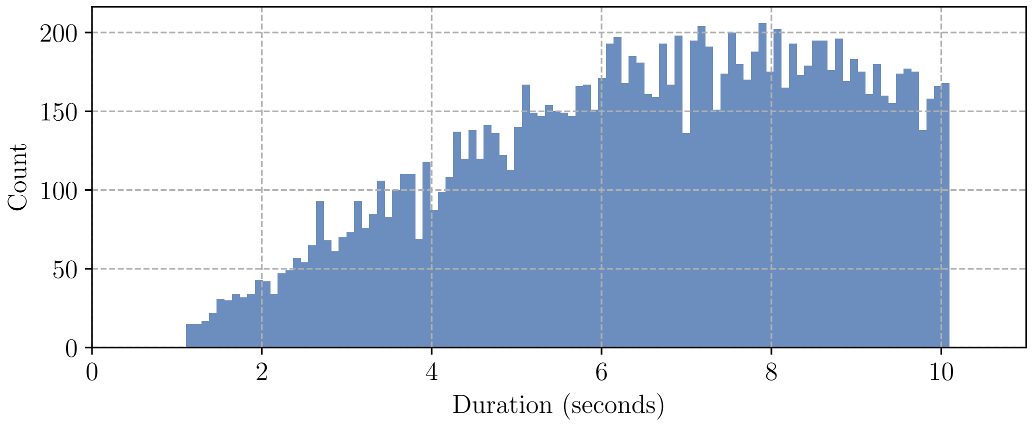
For an algorithm $a \in \{\text{MaryTTS}, \text{Merlin}, \text{Proposed}\}$ the confidence interval $\text{conf}_{a,\Delta}$ is estimated as shown in equation (29). Were z^* denotes the critical value and Δ denotes the delta degrees of freedom.

$$\text{conf}_{a,\Delta}(z^*) = \left(\hat{\mu}_a - z^* \frac{\hat{\sigma}_{a,\Delta}}{\sqrt{N}},\ \hat{\mu}_a + z^* \frac{\hat{\sigma}_{a,\Delta}}{\sqrt{N}}\right) \tag{29}$$

B Corpora Statistics



(a) Distribution of the file durations in the Blizzard Nancy dataset. The bulk of the files has a duration between 0.7 and 10 seconds. Only a small portion of files has a duration above 12.5 seconds with single files even being as long as 20 seconds. Plotted using a total of 100 bins.



(b) Distribution of the file durations in the LJ Speech v1.1 dataset. The file durations range from roughly 1 up to 10 seconds and the number of files available increases with the file duration. It can be seen that the dataset does not contain samples that are longer than roughly 10.1 seconds. Plotted using a total of 100 bins.

Figure 26: Distribution of the file durations in the used corpora

Table 7: Statistics for the Blizzard Nancy and the LJ Speech dataset.

Property	Blizzard Nancy	LJ Speech v1.1
Total Size	1.9 GB	3.6 GB
License	Research License Agreement[b]	Public Domain[c]
Format	16-bit PCM WAV	16-bit PCM WAV
Sampling	16,000 Hz	22,050 Hz
Speakers	1	1
Language	English	English
Sex	Female	Female
Accent	US	US
Total Words	170,018	225,715
Distinct Words	26,786	13,821
Mean Words per Clip	14.06	17.23
Total Characters	993,612	1,308,674
Distinct Characters	35	40 (37 used)
Total Duration	16:45:36	23:55:17
Mean Clip Duration	4.99 s	6.57 s
Min Clip Duration	0.70 s	1.11 s
Max Clip Duration	20.18 s	10.10 s
Total Clips	12,187 (12,095 used)	13,100 (13,100 used)
Train Clips	11,000	12,000
Validate Clips	1,095	1,100

[b] Blizzard Nancy license: http://www.cstr.ed.ac.uk/projects/blizzard/2011/lessac_blizzard2011/license.html

[c] LJ Speech license: https://librivox.org/pages/public-domain/

C Evaluation Sentences

Listing of the 50 sentences used in the evaluation survey. The sentences are pre-processed before synthesis as described in section 5.2.1.

```
1  Clowns grow glowing crowns.
2  I did not object to the object.
3  I met an august man last August.
4  Don't desert me here in the desert!
5  Four hours of steady work faced us.
6  A rod is used to catch pink salmon.
7  Black background, brown background.
8  Seventy-seven benevolent elephants.
9  Rice is often served in round bowls.
10 A pessemistic pest exists amidst us.
11 We must polish the Polish furniture.
12 The boy was there when the sun rose.
13 Do you know what a buck does to does?
14 The juice of lemons makes fine punch.
15 It's easy to tell the depth of a well.
16 Santa is ready to present the present.
17 The garden was used to produce produce.
18 A pot of tea helps to pass the evening.
19 These days a chicken leg is a rare dish.
20 Large size in stockings is hard to sell.
21 The salt breeze came across from the sea.
22 The wind was too strong to wind the sail.
23 Kick the ball straight and follow through.
24 The insurance was invalid for the invalid.
25 The birch canoe slid on the smooth planks.
26 He could lead if he would get the lead out.
27 Glue the sheet to the dark blue background.
28 I was reading a book in Reading, Berkshire.
29 Which witch switched the Swiss wristwatches?
30 When shot at, the dove dove into the bushes.
31 Can you can a can as a canner can can a can?
32 They were too close to the door to close it.
33 A loyal warrior will rarely worry why we rule.
34 He thought it was time to present the present.
35 After a number of injections my jaw got number.
36 The weather was beginning to affect his affect.
37 I had to subject the subject to a series of tests.
38 How can I intimate this to my most intimate friend?
39 Upon seeing the tear in the painting I shed a tear.
40 The soldier decided to desert his post in the desert.
41 A seamstress and a sewer fell down into a sewer line.
42 Give papa a cup of proper coffe in a copper coffe cup.
43 The dump was so full that it had to refuse more refuse.
44 Frivolously fanciful Fannie fried fresh fish furiously.
45 To help with planting, the farmer taught his sow to sow.
46 Before I mow the lawn let me place this grain in the mow.
47 How much wood would a woodchuck chuck if a woodchuck could chuck wood?
48 I seconded the motion that the official be seconded to another department.
49 One morning I shot an elephant in my pajamas. How he got in my pajamas, I
       don't know.
50 Peter Piper picked a peck of pickled peppers. How many pickled peppers did
       Peter Piper pick?
```

Selbständigkeitserklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.

Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und ist noch nicht veröffentlicht worden. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ort, Datum	Unterschrift