Lexical pitch accent and duration modeling for neural end-to-end text-to-speech synthesis

by

Yusuke Yasuda

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

March 2021

Committee

Advisor Prof. Junichi Yamagishi
Professor of SOKENDAI/National Institute of Informatics

Subadvisor Prof. Akiko Aizawa
Professor of SOKENDAI/National Institute of Informatics

Subadvisor Prof. Atsuhiro Takasu
Professor of SOKENDAI/National Institute of Informatics

Examiner Prof. Keiichi Tokuda
Professor of Nagoya Institute of Technology

Examiner Dr. Takahiro Shinozaki
Associate professor of Tokyo Institute of Technology

Abstract

Text-to-speech synthesis (TTS) is a task to transform texts into speech. End-to-end TTS is one of the TTS frameworks, and its unique approach is characterized by using only a single model to generate speech directly from texts. This feature of end-to-end TTS makes contrast to conventional TTS framework called pipeline which consists of multiple models specialized to one function to realize TTS. End-to-end TTS reduces the burden of constructing TTS model without relying on complex labeling, and is therefore promising to expand its applicability to many languages. On the other hand, the end-to-end approach has limitation to learn every language or speech features implicitly.

This thesis focus on two lexical features of speech to improve end-to-end TTS: pitch accents and phoneme duration. These two features have been treated by dedicated models in traditional TTS frameworks. In contrast, existing end-to-end TTS methods do not model these features explicitly and try to capture these features implicitly without relying on any mechanism specialized for them.

In preliminary experiments, we show that some typical end-to-end TTS systems tend to produce flat pitch when they are applied to English, which is intonation languages in terms of prosodic description. In addition, end-to-end TTS systems fail to produce correct pitch accents without any supervision for pitch accents when they are applied to Japanese, which is pitch accent language in terms of prosodic description. We also show that soft-attention, which is used for alignments in most end-to-end TTS methods, is hard to avoid alignment errors which sometimes result in unacceptable quality of synthetic speech. These experimental results motivated us to focus on pitch accent and alignment modeling in end-to-end TTS.

This thesis proposes methods to incorporate a dedicated module to end-to-end TTS

to model these two features by using latent variable. For pitch accents, we introduce pitch accent modeling to end-to-end TTS. We represent pitch accent latent variable in the form of abstract tone contour patterns, which makes the latent space discrete. To learn and model pitch accents, tone recognizer and tone predictor are introduced, and all components are optimized jointly. Our method enables to predict pitch accents without relying on labels during test phase. To enforce latent variable to represent pitch accent information, we use supervision with tone contour labels during training. In experiments, our method can generate moderately correct pitch accents without relying on tone contour labels during test phase.

For alignment modeling, we develop end-to-end TTS method using discrete latent alignments. As a first step, we handle alignments in acoustic frame level. We design monotonic alignment by using transition variable to avoid fatal alignment errors. The transition variable has two values, stay or proceed, which makes alignment structure monotonic. The whole TTS model can be optimized by maximizing marginal likelihood by marginalizing with respect to all possible alignments. Our experiments show that this method successfully eliminates fatal alignment errors, but it still gives other kinds of alignment errors such as overestimation of phoneme duration.

We advance the end-to-end TTS method incorporating alignment modeling by constructing discrete latent alignments based on phoneme duration. Phoneme duration ensures monotonic alignment structure, and it enables to model alignments more efficiently compared with alignments represented in acoustic frame level because phoneme duration sequences are much shorter than acoustic feature sequences. We represent phoneme duration as discrete symbols in the number of acoustic frames. Linguistic feature inputs can be aligned to acoustic feature outputs by upsampling them based on phoneme duration. A duration recognizer, duration predictor, and forced aligner are introduced to model phoneme duration, and all components are optimized jointly. In experiments, our TTS method using phoneme duration modeling shows higher naturalness of synthetic speech compared to existing TTS method using duration predictor.

We use an identical framework to handle pitch accents and phoneme duration. Both pitch accents and phoneme duration are lexical feature depending on linguistic units, and can be represented in discrete symbols. We thus use conditional VQ-VAE (vector quantized variational autoencoder) to model the two features as latent variable.

We show both pitch accents and phoneme duration can be handled with the conditional VQ-VAE. We construct the pitch accent modeling by defining approximate posterior with tone contour pattern labels, using tone recognizer as encoder, and utilizing tone predictor as prior for VQ-VAE. We construct phoneme duration modeling by defining approximate posterior with forced aligner, using duration recognizer as encoder, and utilizing duration predictor as prior for VQ-VAE. There components have been used for conventional TTS methods consisting of multiple models dedicated to a specific functionality of TTS. Our method based on the conditional VQ-VAE enables for end-to-end TTS to incorporate the conventional modules designed for a specific feature of speech. This approach connects end-to-end TTS to conventional TTS methods to compensate its disadvantages by using established ways of the conventional TTS while keeping its advantage.

Acknowledgments

I would like to express my gratitude to:

- Prof. Junichi Yamagishi, who is my supervisor. I sincerely appreciate your decision to accept me as a Ph.D student even though I had no background in speech area. He offered me summer school of speech technology at Crete and an advanced research assistant job, which gave me basic knowledge of speech technology and supported my Ph.D life financially. I sincerely thank you for your supports and advice to my research.

- Prof. Akiko Aizawa and Prof. Atsuhiro Takasu, who are my subadvisors at SOKENDAI and members of the dissertation committee. I sincerely thank you for your technical comments and suggestions on topics related to this thesis and the presentation.

- Prof. Yusuke Miyao, who was my subadvisors at SOKENDA during my first and second year of Ph.D course. I sincerely thank you for your technical comments and suggestions on topics related to the presentation.

- Prof. Keiichi Tokuda and Dr. Takahiro Shinozaki, who are members of the dissertation committee. I sincerely thank you for your technical comments and suggestions on topics related to this thesis and the presentation.

- Dr. Xin Wang, who is my mentor. He has been a role model of a successful researcher for me. I sincerely thank you for your advice and contributions to the works related to this thesis.

- Dr. Shinji Takaki, who was my mentor during my first year of Ph.D course. I sincerely thank you for your advice to the works related to this thesis.
- Makiko Kuwahara, who is the secretary of Yamagishi-lab. I sincerely thank you for your support on document works, which enables me to focus on my research.
- SOKENDAI, which is my official affiliation as a Ph.D. student. SOKENDAI is ideal environment for students who continue a job during academic life like me. I sincerely thank for the wonderful program.
- National Institute of Informatics, which is my physical affiliation. I sincerely thank for the best research environment they provide.
- Chatwork, which is the company I work for, and the members I work with at Chatwork. I thank Shigetoshi Kasuga, my boss, for allowing me to start Ph.D course. I thank my team members for their patience to my limited contribution to the project during my Ph.D course.

Please forgive me for not listing all the names to whom I owe this thesis. Finally, I thank my parents for understanding my decision to start Ph.D course.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Text-to-speech synthesis (TTS) refers to technology used to convert text to a speech signal. TTS is usually composed of several components. Grapheme-to-phoneme conversion (G2P) converts text to phonemes, which are a linguistic feature. Text-to-duration predicts the acoustic length of each phoneme, called duration, and upsamples linguistic features to the length of acoustic feature. An acoustic model converts the upsampled linguistic features to acoustic features. A waveform model converts acoustic features to waveforms. TTS works as a system by pipelining these components.

Recently, a new TTS framework called end-to-end TTS was proposed. This framework enables speech to be generated directly from text by unifying all of the components above in one model. This approach is promising for applicability to many languages with less costs because it does not require language-specific annotation. Some end-to-end TTS methods are successful in terms of the quality of the generated synthetic speech.

Application of end-to-end TTS was mainly limited to English at the start of our research, although end-to-end TTS was promising as a language-versatile approach. For example, it was uncertain how to apply end-to-end TTS to tonal or pitch accent languages, and languages with orthographic writing systems. Examples of these languages are Mandarin and Japanese.

In addition, the conditions in which end-to-end TTS can successfully learn acoustic representations from text were not known well. Many end-to-end TTS systems were proposed independently, with various neural network structures, alignment methods, model parameter size configuration, and type of acoustic features. The impact these configurations had on the performance of end-to-end TTS has not been investigated.

Robustness of pronunciation was one of the most major issues in end-to-end TTS. It is a unique problem for end-to-end TTS which has to resolve pronunciation and relationships between linguistic features and acoustic features without relying on elaborated labels. Alignments especially suffer from the robustness issue of end-to-end TTS, which results in unacceptable levels of pronunciation errors.

1.2 Thesis overview

1.2.1 Motivation

End-to-end TTS unifies all modules in TTS. This approach has been advancing TTS, and has faced additional challenges at the same time. 1) The end-to-end approach, learning every aspect of speech from texts directly, is limited in its ability to consider language specific characteristics. 2) The unification of underlying TTS components makes it challenging to design each internal component specialized to TTS specific characteristics.

Our research was motivated by our first investigation on what kind of linguistic and acoustic features can be properly learned from text and configurations to improve the learning ability in existing end-to-end TTS methods for English. This investigation revealed that pitch accent and alignment were challenging features for the end-to-end TTS model to learn from texts. Accordingly, we investigated applicability of end-to-end TTS to pitch accent languages by using Japanese as an example. The investigation about pitch accents in end-to-end TTS led us to propose a new method to incorporate

pitch accent modeling in end-to-end TTS. Meanwhile, we developed an end-to-end TTS method using a hard-attention for alignment method instead of soft-attention, because soft-attention was the cause of the unstable alignments. We advanced the end-to-end TTS method further by introducing phoneme duration modeling for the alignment method. We showed that the same framework can be used for pitch accent modeling and phoneme duration modeling.

In summary, the aim of this thesis is 1) to expand applicability of end-to-end TTS to pitch accent and tonal languages, and 2) to stabilize alignments by rebuilding the end-to-end TTS framework by unifying components designed specifically for TTS based on conventional approaches.

1.2.2 Topic and scope

This thesis covers end-to-end TTS among many TTS frameworks. We refer to a TTS system whose input is text or phoneme and output is acoustic features as end-to-end. Our research goals for this thesis are two fold: 1) solve issues such that end-to-end TTS may be applied to many languages; 2) propose a common method to solve different issues. We focus on two features in TTS: 1) pitch accent and 2) phoneme duration, with consideration of cross-language applicability. We analyze end-to-end TTS in terms of the two features, and introduce components to model the two features in end-to-end TTS. We handle the two features separately in this thesis, but we propose methods which can be used for both of the features. Although we investigate our methods in English, Japanese, and Chinese only, our methods are applicable to many languages as well as the three languages.

This thesis does not cover the following topics. 1) Waveform modeling: ideally, outputs of an end-to-end TTS model should be a waveform. We do not investigate waveform modeling as a part of end-to-end TTS. It is basically signal-to-signal conversion so is a language agnostic task. We focus on language and text related issues in end-to-end TTS. 2) TTS involving logographic text: ideally, end-to-end TTS should handle texts from all languages, but it is quite challenging for languages using logographic writing system. We use phonemes instead to avoid the difficulty and focus on other topics for these languages.

1.2.3 Issues to be addressed

We report the following two features are very hard to model implicitly in current end-to-end TTS methods: 1) pitch accent and 2) phoneme duration. We clarify issues to learn these features implicitly with end-to-end TTS by analysis and comparison with conventional approaches at the beginning of this thesis. We propose a common framework to handle the two features better in end-to-end TTS in later chapters.

The first feature is pitch accent. Pitch accents are accents involving pitch change. Pitch accent is quite important in tonal or pitch accent languages such as Chinese and Japanese. Pitch accent modeling is thus one of the keys to expand the applicability of end-to-end TTS to various languages. This thesis handles the following issues regarding pitch accents:

- 1) Methods to treat pitch accents in tonal or pitch accent languages are not established in end-to-end TTS.
- 2) Pitch accent prediction from text is separately handled and is not incorporated as a part of end-to-end TTS.

The second feature is phoneme duration. Phoneme duration has a role to relate linguistic features with acoustic features. The relationship between source and target, which is linguistic features and acoustic features in TTS, is called alignment. In end-to-end TTS, alignment is modeled implicitly without depending on labels. This thesis tackles the following issues about alignment modeling in current end-to-end TTS:

- 1) Alignments modeled with the current schemes of end-to-end TTS are unstable and error-prone.
- 2) Alignment is not represented with phoneme duration in current end-to-end TTS methods, which is a more natural, robust, and efficient form for TTS.

The problems regarding the two features look independent, but can be tackled with common approach, namely latent variables. Internal constituents of speech such as pitch accent and phoneme duration can be captured by introducing latent variables to TTS model. We introduce latent variables to end-to-end TTS to model the two different factors.

1.2.4 Contribution

- Analysis (Chapter 3):
  - We show that increasing the parameter size of a model can help greatly to stabilize end-to-end TTS models to learn alignment and pronunciation from text directly, but it does not help to produce natural enough pitch accents.
  - We show that the design of the dominant alignment method used in end-to-end TTS, called soft-attention, makes fatal alignment errors difficult to avoid.
- Pitch accent (Chapter 4-5):
  - We show that end-to-end TTS can produce correct pitch accent if pitch accents labels are given in Japanese as an example.
  - We show that pitch accent can be modeled as a latent variable, and the model can predict reasonably correct pitch accent during inference without depending on labels.
- Phoneme duration (Chapter 6-7):
  - We show that hard latent alignment enables the design of a monotonic structure of alignment in end-to-end TTS. Monotonic structure of alignment is essential for TTS.
  - We show that phoneme duration can be modeled as a latent variable in end-to-end TTS.

1.3 Outline of thesis

Chapter 2 provides background about end-to-end TTS and conventional TTS methods.

Chapter 3 analyzes issues in end-to-end TTS. We compare end-to-end TTS with a conventional method, and clarify its issues. The experiments reveal that pitch accents and alignments are hard to learn in current end-to-end methods.

Chapters 4-5 focuses on pitch accents. Chapter 4 investigates learning ability of pitch accents in end-to-end TTS by targeting a pitch accent language, Japanese.

Chapter 5 proposes a framework to model pitch accent as a latent variable in end-to-end TTS.

Chapters 6-7 focuses on alignment. Chapter 6 proposes a new end-to-end TTS method using latent variables to design monotonic alignment, which avoids alignment errors in the TTS task. Chapter 7 proposes an end-to-end TTS framework that directly models phoneme duration instead of acoustic frame-level alignment. We design this framework based on the method to model pitch accent as a latent variable in Chapter 5. We show both pitch accent and phoneme duration can be modeled with the same framework.

1.4 Notation

Table 1.1: Notation used in this thesis.

Notation	Meaning	Example
t	A time index of acoustic features	$1, 2, 3, \ldots$
T	Maximum time index of acoustic features	$1, 2, 3, \ldots$
u	A time index of linguistic features	$1, 2, 3, \ldots$
U	Maximum time index of linguistic features	$1, 2, 3, \ldots$
$\mathbf{x}_t$	A frame of acoustic feature at time t	80 dimensional mel-spectrogram
$\mathbf{x}_{1:T}$	A sequence of acoustic feature with length T	$(\mathbf{x}_1, \cdots, \mathbf{x}_T)$
y_u	A symbol of linguistic feature at time u	A character, a phoneme
$y_{1:U}$	A sequence of linguistic feature with length U	$(y_1, \cdots, y_U)$
$\mathbf{y}_{1:U} = \text{Encoder}(y_{1:U})$	An encoded linguistic representation	$(\mathbf{y}_1, \cdots, \mathbf{y}_U)$
$\mathbf{y}_{1:U} = \text{Frontend}(\text{text})$	Linguistic features derived with front-end	$(\mathbf{y}_1, \cdots, \mathbf{y}_U)$
$\hat{y}_{1:T}$	Upsampled linguistic feature with length T	$(\hat{y}_1, \cdots, \hat{y}_T)$

2

End-to-end text-to-speech synthesis

2.1 TTS frameworks and Terminology

The task of TTS is to predict acoustic features $\mathbf{x}_{1:T}$ from linguistic features $y_{1:U}$. This thesis focuses on the end-to-end TTS framework, one of the frameworks for TTS. Here we summarize TTS frameworks related to the end-to-end TTS framework before we dive into end-to-end TTS. As a related TTS framework, we describe the pipeline TTS framework.

Since machine learning based speech synthesis was long studied prior to the deep learning period, this section clarifies the definition of the two TTS frameworks and a taxonomy of TTS methods belonging to the frameworks. Figure 2.1 shows TTS frameworks appearing in this thesis.

Statistical parametric speech synthesis (SPSS) is a TTS method that involves acoustic parametric representation of speech, i.e. vocoder parameter [4]. A typical SPSS-based TTS system contains an acoustic model to predict vocoder parameters and a vocoder to produce a waveform given the predicted vocoder parameters. For example,

hidden semi-Markov model (HSMM)-based SPSS uses the HSMM as the acoustic model to convert linguistic features into vocoder parameters [5]. Such an HSMM treats the phoneme duration as a hidden variable and does not require an independent aligner or a phoneme duration model. HSMM-based SPSS uses a conventional vocoder to produce the waveform [6, 7], which suffers from the artifact caused by minimum phase and other assumptions in the speech production process [8].

Deep neural network (DNN)-based SPSS [9] uses a DNN rather than an HSMM as the acoustic model. It outperforms the HSMM-based SPSS by switching from the state-level acoustic modeling to frame-level regression by modeling $p(\mathbf{x}_{1:T} \mid \hat{y}_{1:T})$ [10]. However, typical DNN-based SPSS relies on an external duration model to align linguistic features and target vocoder parameters, even though there has been an attempt to combine HSMM with DNN [11]. Traditional DNN-based SPSS has relied on the conventional vocoder used in HSMM-based SPSS. However, since the birth of neural vocoders [12, 13], some DNN-based SPSS systems have adopted a neural vocoder for waveform generation. These systems can improve the quality of synthetic speech by directly modeling the waveform with the neural vocoder.

There are also TTS approaches that do not depend on vocoder parameters. In these approaches, an acoustic model predicts a power spectrogram [14], an acoustic feature that is obtained with a simpler analysis and is closer in form to a waveform than vocoder parameters. In this method, a waveform is generated by applying inverse Short-Time Fourier Transform (STFT) with phase recovery. This approach is normally distinguished from SPSS and is referred to as vocoder-free TTS [15]. It inherits the pipeline structure of DNN-based SPSS, but it can alleviate buzzy sounds caused by the conventional vocoder. However, it causes another type of artifact when the prediction accuracy of the power spectrogram is poor or when the hop size of the windows is too large.

DNN-based SPSS and vocoder-free TTS are major approaches in pipeline TTS frameworks in recent works. Although the distinction between the two approaches is based on the type of acoustic feature, the difference becomes less important once a neural vocoder is used to generate the waveform. A neural vocoder can take arbitrary acoustic features as input. Vocoder parameters [16], fundamental frequency and frame-level linguistic features [12], and mel-scale amplitude spectrograms [17] have been used as input of the neural vocoder in pipeline TTS.

An end-to-end, also known as sequence-to-sequence, TTS framework is the main focus of this thesis. As figure 2.1 shows, an end-to-end TTS framework has a different structure to the pipeline one: it unifies the front-end, duration model, and acoustic model as one model by utilizing the DNN-based sequence-to-sequence framework [18] that can map a source-to-target sequence with a length mismatch. As well as pipeline TTS, sequence-to-sequence TTS has variant systems using different types of acoustic features and waveform generation methods: vocoder parameter and conventional vocoder [19, 20], vocoder parameter and neural vocoder [21], mel and linear spectrogram and invert STFT with phase recovery [1], and mel or linear spectrogram and neural vocoder [22].

In this thesis, we use "pipeline" to refer to all methods in a pipeline framework and "DNN-based pipeline TTS" to refer to both DNN-based SPSS and vocoder-free TTS. We clearly mention a specific TTS method if the distinction is important. We use "conventional" or "traditional" for the pipeline TTS framework even though some methods in the pipeline framework are relatively new in TTS history and the first sequence-to-sequence TTS was proposed only a few years ago [19].

2.2 End-to-end TTS

2.2.1 Overview

End-to-end TTS is a new TTS framework using a sequence-to-sequence learning method [18]. The end-to-end TTS unifies the linguistic model, acoustic model, duration model, and ultimately the waveform model [36, 37], from the conventional pipeline framework into a single model. It has provoked many studies because of its potential to unify all models into one and its handiness as it uses text and speech only for model training.

Many architectures have been proposed for sequence-to-sequence based TTS. Table 2.2 shows major methods of end-to-end TTS. Notable architecture examples are Tacotron [1], Char2Wav [21], VoiceLoop [20], DeepVoice3 [34], DCTTS [38], Tacotron2 [22], and Transformer [39]. All of them have an encoder-decoder [40] structure, which consists of an encoder to encode linguistic input and a decoder to decode acoustic features autoregressively. From the perspective of neural network (NN) types to

Table 2.1: List of languages which end-to-end TTS are applied to, except for English.

Language	Linguistic feature	Corpus	Reference
Chinese	Phoneme	Internal	[19, 23, 24]
	phone & tone	Internal	[25]
	pinyin	CSS10	[26, 27]
	character/phone/byte	Internal	[28]
	character	China Daily	[29]
	full-context	Internal	[30]
	pinyin	Data Baker	[31]
Korean	phoneme	KSS	[32]
	character	Internal	[27]
Japanese	phoneme & accentual type	ATR Ximera	[2]
	full-context	ATR Ximera	[33]
	roman	CSS10	[26]
Dutch	character	CSS10	[26, 27]
Finnish	character	CSS10	[26, 27]
French	character	CSS10	[26, 27]
German	character	CSS10	[26, 27]
Greek	character	CSS10	[26, 27]
Hungarian	character	CSS10	[26, 27]
Russian	character	CSS10	[26, 27]
Spanish	character	CSS10	[26, 27]
	character/phone/byte	Internal	[28]

implement the sequence-to-sequence framework, those TTS architectures can be roughly classified into recurrent neural network (RNN)-based [1, 21, 22], convolutional neural network (CNN)-based [34, 38], self-attention based ones [39] or memory buffer [20].

Note that, although end-to-end TTS is a new framework, each key element such as phone and linguistic feature embedding [41, 42], autoregressive decoder [17, 43], and postnet [44, 45] has already been investigated for DNN-based pipeline TTS methods separately. A notable difference between the sequence-to-sequence and pipeline TTS frameworks is not the elements themselves but joint training of the elements.

Table 2.2: Summary of major end-to-end TTS methods. All the methods use soft attention mechanism for implicit alignment learning.

System	Network	Alignment	Decoder output	Post-net output
Char2Wav [21]	RNN	GMM	Vocoder	-
Tacotron [1]	RNN	Additive	Mel	Linear
VoiceLoop [20]	Memory buffer	GMM	Vocoder	-
Deep Voice 3 [34]	CNN	Dot-product	Mel	Linear/Vocoder
Tacotron 2 [22]	RNN	Location-sensitive	Mel	Mel
Transformer [35]	Self-attention	Dot-product	Mel	Mel

2.2.2 Linguistic modeling

The pipeline TTS frameworks typically use grapheme-to-phoneme conversion. The sequence-to-sequence TTS instead uses an encoder $\mathbf{y}_{1:U} = \text{Encoder}(y_{1:U})$ to transform input characters $y_{1:U}$ into a linguistic representation $\mathbf{y}_{1:U}$ that is supposed to encode the pronunciation and possibly prosody information.

Table 2.1 shows languages which end-to-end TTS are applied to, except for English. An initial proof-of-concept of sequence-to-sequence TTS was investigated in Chinese with phoneme input [19]. Then character level input was investigated in English [1, 21] and was confirmed to be feasible in other alphabetical languages [26] and Chinese [29, 28]. The naturalness of synthetic speech using character input can be quite high for English [22].

Although implicit linguistic modeling is possible when the encoder has a number of sufficient non-linear transformation layers [46] and when there is a sufficient amount of training data, even large-scale speech copora such as LibriTTS [47] never match a lexicon dictionary in terms of word coverage [48]. This suggests a potential limitation of end-to-end TTS for out-of-vocabulary words and out-of-domain text, especially in languages with highly nonphonemic orthography such as English and ideographic languages such as Chinese. It is reported that phoneme input gives better results than character input [49, 28] for sequence-to-sequence based TTS and many recent sequence-to-sequence based TTS studies avoid character input and use phonemes instead. Some studies use both character and phoneme [34, 50] or a random mix of them [51]. It is also reported that pre-training an encoder to learn

grapheme-to-phoneme mapping is also an effective approach [29]. Nevertheless, the sequence-to-sequence based TTS is convenient to construct a multi-lingual model by handling various inputs with a single model, and it has been investigated with character, phoneme, and even byte input [24, 27, 52, 28].

There are also attempts to use richer linguistic information as additional inputs for sequence-to-sequence based TTS for tonal languages such as Chinese [53], and pitch accent languages such as Japanese [54], where tone or accent and prosody information are indispensable yet hard to learn from graphemes or phonemes implicitly. However, some studies report excessive information negatively affects their results [30, 33]. Simpler linguistic features such as pinyin with tone [31], phones with tone [25, 30], or phonemes with accentual type [2] seems to be good choices for these languages.

2.2.3 Alignment modeling

As another difference from DNN-based pipeline TTS, the sequence-to-sequence TTS does not use the external aligner or duration model. Instead, many sequence-to-sequence TTS methods use an attention mechanism [55] to implicitly align source and target sequences. An attention mechanism computes attention score or alignment probability $p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)$ which indicates how likely it is that the alignment z_t at t-th time step corresponds to input position u given the previous acoustic feature $\mathbf{x}_{t-1}$ and u-th linguistic feature $\mathbf{y}_u$. An acoustic decoder decodes output $\mathbf{x}_t$ from a context vector $\sum_{u=1}^{U} p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\mathbf{y}_u$, which is an expectation of the encoded linguistic features by the attention scores, as in $\mathbf{x}_t = \text{Decoder}\left(\sum_{u=1}^{U} p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\mathbf{y}_u, \mathbf{x}_{t-1}\right)$.

There are many types of attention mechanisms. Basic ones include Gaussian mixture model (GMM) [56], additive [55], dot-product [57], and location-sensitive [58] attention mechanisms, which were designed for other tasks but applied to TTS as in [21, 20, 1, 34, 39, 19, 22]. Several attention mechanisms have also been proposed specifically for TTS, where the alignment between source and target is monotonic and the target sequence is much longer than the source. They include forward attention [25], stepwise monotonic attention [59], a mixture of logistic attention [60], and dynamic convolution attention [61], all of which use a time relative approach to enforce monotonicity.

In addition, attention regularization loss has also been proposed for robust and fast

alignment learning [38, 62]. Furthermore, instead of using an advanced attention mechanism, systems based on CNN or self-attention that enable parallel training have advanced to fast non-autoregressive inference using attention distribution distillation [50] or duration model distillation [63]. Meanwhile, it is found that jointly modeled alignment by attention is not the source of the high naturalness of sequence-to-sequence based TTS [46].

2.2.4 Acoustic modeling

Typically, decoders in the sequence-to-sequence TTS adopt deep autoregressive neural networks to produce the target acoustic feature sequence on the basis of the attention and encoder outputs as in $\mathbf{x}_t = \text{Decoder}\left(\sum_{u=1}^{U} p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\mathbf{y}_u, \mathbf{x}_{t-1}\right)$.

The autoregressive structure is introduced to model its statistical dependency across frames and is very important for improving the prediction accuracy [46]. However, the autoregressive modeling has a common problem called *exposure bias* – the autoregressive decoder is inclined to excessively rely on feedback acoustic features while ignoring the encoder output to predict the next acoustic feature frame. Furthermore, the inaccurate predictions accumulate across the frames and lead to alignment errors and unacceptable outputs –. To alleviate this problem, most architectures apply dropout [64] regularization to feedback in layers called pre-net [1, 34, 22, 39]. There are also methods using a Generative Adversarial Network (GAN) [65], forward-backward decoding [23], and teacher-student training [31] to alleviate the exposure bias.

As for acoustic features, mel-scale spectrogram is widely used. The fine-grained vocoder parameter is less commonly used since it increases the length of output sequences and make decoding more difficult.

2.3 Pipeline TTS framework

2.3.1 Front-end

All TTS methods in the pipeline framework have a front-end $\mathbf{y}_{1:U} = \text{TextProcessor}(\text{text})$ that uses multiple knowledge-based or statistical models to extract linguistic features

$\mathbf{y}_{1:U}$ from the input text. These linguistic features are usually represented as full-context labels, which encode the segmental (e.g., phoneme identity) and suprasegmental (e.g., phrase-level intonation) information to utter the text. For example, a typical front-end for English may use a pronunciation dictionary and a decision-tree-based model to conduct the letter-to-sound (or grapheme-to-phoneme) conversion [66]. It may use other hand-crafted rules or statistical models to infer the phase breaks [67] and pitch accents [68]. These modules in the front-end are usually language dependent, and the statistical ones require supervised training and hand-annotated datasets.

Note that there are also methods to build a front-end with less language-specific knowledge or supervised training. For example, the vector space model may be used to derive linguistic features from input text in an unsupervised way [41, 42]. However, such a method is not applicable to ideographic languages such as Japanese due to the sparse distribution of written characters. On the other hand, there are also methods to jointly optimize statistical models in the front-end with the acoustic model in the back-end [69]. However, this method was only proposed for HMM-based SPSS and cannot be applied to DNN-based SPSS directly.

2.3.2 Back-end

The back-end in pipeline based TTS has two models: acoustic and waveform. The acoustic model has been advanced by replacing the hidden Markov models (HMMs) with DNNs [10]. On the basis of the vanilla feed-forward and recurrent NNs, researchers have further proposed many NN-based acoustic models with improved probabilistic modeling capability.

in particular, the vanilla feed-forward and recurrent NNs assume statistical independence among the acoustic feature frames [43]. To amend the independence assumption, autoregressive NN models were proposed to model the acoustic feature distribution of one frame conditioned on the previous frames, which can be implemented by feeding back the previous frames linearly [70] or non-linearly [43, 17]. A similar idea can be used to capture the causal dependence between different acoustic feature dimensions [71]. Meanwhile, unlike the autoregressive NNs, there are also other NN-based models using intractable or implicit density functions. Examples of the former case include the post-filtering approach using a restricted Boltzmann machine

(RBM) [44] and the trajectory DNN [72]. The latter case is the generative adversarial network (GAN) [45]. All the recently proposed NN-based acoustic models alleviate the over-smoothing effect in the vanilla NNs.

There are two types of waveform model in the back-end: algorithmic method or neural method. The conventional vocoder is an algorithmic method that has long been a core component of SPSS. However, the artifacts in generated waveforms from the vocoder limit the quality of SPSS, which motivates the development of waveform generation methods such as Griffin–Lim [73] in vocoder-free TTS and the neural waveform models [8]. Compared with the Griffin-Lim and related algorithms, the neural waveform models show better performance [74]. They are also more flexible as they can work on various types of input acoustic features including mel-spectrogram and vocoder parameters, be implemented with different types of NNs such as GAN [75] and normalizing flow [76], and incorporate classical signal processing algorithms like linear prediction coefficients (LPC) [77] and the harmonic-plus-noise model [78].

2.3.3 Duration model

HSMM-based SPSS models state duration with Gaussian distribution to align source and target [5]. In this case, the duration model is inherent to HSMM, and there is no external duration model.

DNN-based SPSS uses an independent phoneme duration model for alignment. A phoneme duration model is trained with duration labels obtained with a force aligner. The force aligner can be either HMM or neural recognition models. With phoneme duration, linguistic features $y_{1:U}$ can be upsampled up to the length of acoustic feature as in $\hat{y}_{1:T}$.

Phoneme duration models can be used to replace soft-attention in end-to-end TTS systems. In this case, phoneme duration models are normally external models, so it is not regarded as an end-to-end TTS framework but similar to vocoder-free TTS in a pipeline TTS framework. The force aligner used to train the duration model can be based on HMM [79, 80] or DNN [81] based recognition model. Some systems use a generative model for force aligner instead of a recognition model, such as a TTS model using hard-attention [82] or soft-attention [63].

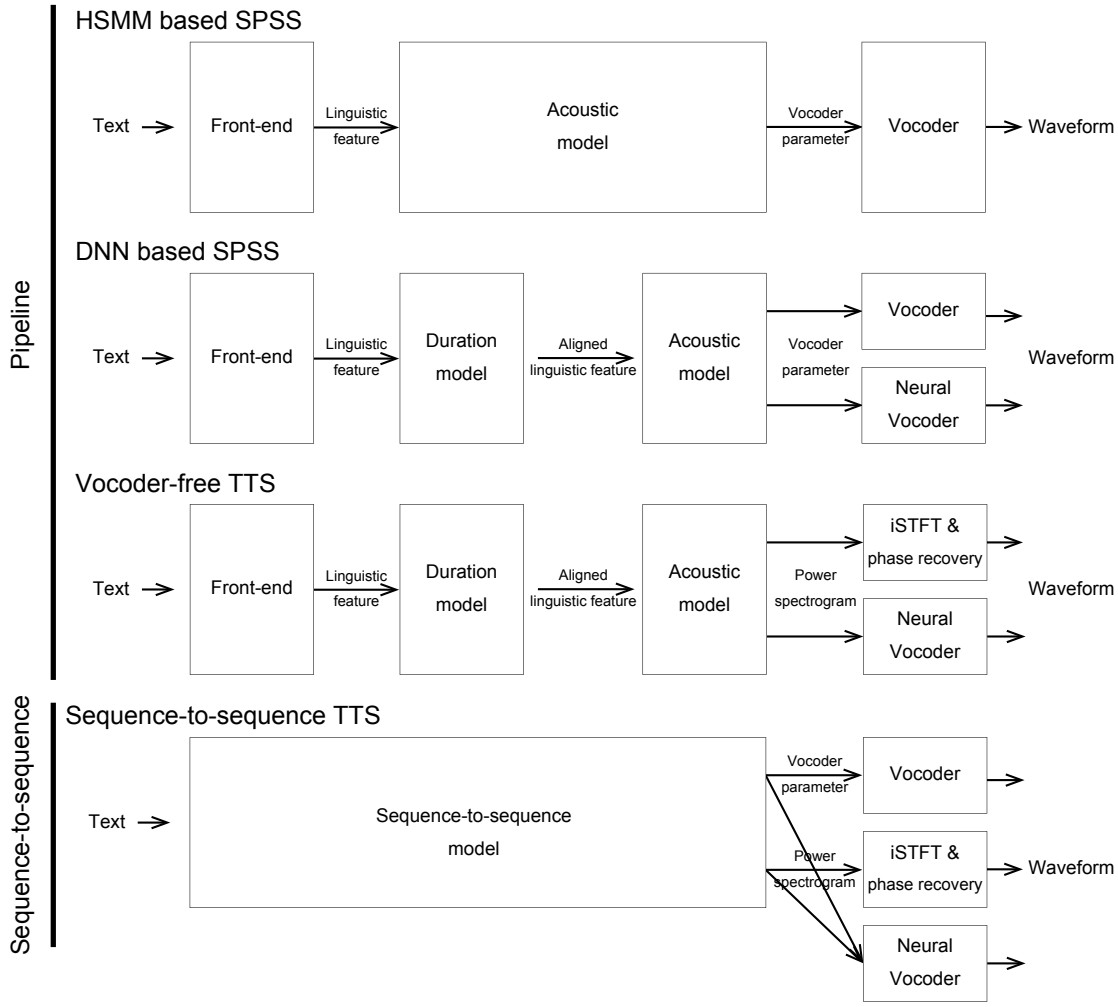


Figure 2.1: TTS frameworks appearing in this thesis. Pipeline TTS framework consists of independent front-end and back-end models. Pipeline TTS framework includes HSMM based SPSS, DNN based SPSS, and vocoder-free TTS. HSMM based SPSS and DNN based SPSS predict vocoder parameter with HSMM or DNN, respectively, and they use conventional vocoder for waveform generation. Vocoder-free TTS predicts power spectrogram with DNN, and it uses invert STFT and phase recovery to generate waveform. Neural vocoder can also be used for waveform generation. Sequence-to-sequence TTS has a single model, and it may use either vocoder parameter or power spectrogram, and corresponding waveform generation method or neural vocoder.

3

Analysis and issues of end-to-end TTS

3.1 Targets of analysis

3.1.1 Pronunciation

Learning pronunciation directly from texts is one of the most important characteristics of end-to-end TTS framework. Resolving pronunciation is possible in alphabetical languages, but it is not easy, as we described in Section 2.2.2 in Chapter 2. Phonemes are an alternative feature to characters for end-to-end TTS to avoid pronunciation ambiguity.

In this chapter we focus on factors affecting the ability of an end-to-end TTS model to disambiguate pronunciation of texts. The factor we cover here is neural network architecture. The encoder's network structure is especially important for pronunciation learning. We compare two encoder structures to see how much pronunciation learned from texts can be improved by encoder structure: one structure is specialized for character inputs and the other is a simple structure.

3.1.2 Alignment

End-to-end TTS has an internal alignment module to align linguistic features and acoustic features which have different lengths. Soft-attention is the most popular method for the alignment module in end-to-end TTS. Soft-attention provides a simple and computationally efficient alignment mechanism which avoids marginalization. Most end-to-end TTS methods adopt soft-attention.

It is known that soft-attention has a major issue in the case of TTS. In TTS, alignment between linguistic and acoustic features must be monotonic because transcription is read aloud from left to right. However, alignments predicted with soft attention may have fatal errors. Soft-attention can see the whole transcript at any time of decoding even though only small part of transcript should correspond to a frame of acoustic features. This feature of soft-attention is suitable to tasks such as machine translation in which source and target relation may require reordering. This is however not ideal for TTS in which the relationship between source and target is always monotonic, and the target has much longer length than the source, which makes predicting alignments challenging. Fatal alignment errors caused by soft-attention may result in muffling, repetition, skip, or ill termination.

Several soft-attention methods are proposed to stabilize alignments. They include forward attention [25], stepwise monotonic attention [59], a mixture of logistic attention [60], and dynamic convolution attention [61], all of which use a time relative approach to enforce monotonicity.

We focus on factors affecting alignment error rates other than attention mechanism itself in this chapter. The factors include 1) model parameter size, and 2) neural network architecture. We choose these factors because 1) they affect the learning ability of the TTS model in general including alignment, and 2) modifying these factors is not complicated. We use forward attention [25] as the attention mechanism in our experiment, which provides robust alignment and fast learning of alignment. We treat alignment methods different from soft-attention to force monotonicity by design in Chapters 6 and 7.

Table 3.1: Comparison of structure and configuration between Tacotron and Tacotron2. Numbers in brackets are unit size of layers.

	Tacotron [1]	Tacotron2 [22]
Embedding	(256)	(512)
Encoder pre-net	FFN (256, 128)	-
Encoder	CBHG (256)	CNN + LSTM (512)
Decoder pre-net	FFN (256, 128)	FFN (256, 256)
Attention RNN	GRU (256)	LSTM (1024, 1024)
Attention	Additive [83] (256)	Location-aware [58] (128)
Decoder RNN	GRU (256, 256)	-
Post-net	CBHG (256)	CNN (512)
Total parameters	6.9×10^6	27.3×10^6
Vocoder	Griffin-Lim [73]	WaveNet [12]

3.2 End-to-end TTS methods used for analysis

3.2.1 Tacotron and Tacotron2

We review Tacotron [1] and Tacotron2 [22] as representative of end-to-end TTS methods in this section. Tacotron is one of the earliest end-to-end TTS methods, and Tacotron2 is successor method of Tacotron which successfully improved the naturalness of synthetic speech to the human level. We construct TTS systems for our analysis in this chapter based on Tacotron and Tacotron2.

Tacotron and Tacotron2 are based on a sequence-to-sequence [18] conversion framework which consists of an encoder and decoder, as described in Chapter 2.

Tacotron and Tacotron2 share basic mechanisms to convert linguistic features to acoustic features. First, linguistic inputs are encoded by the encoder. Then at the decoder, encoded values are processed with an attention framework [55, 16, 84] to align with decoder outputs. The output from the decoder pre-net is first concatenated with the context vector from the previous time step and processed by the attention RNN. The attention mechanism relates the encoder output and the output from the attention RNN by assigning probability for which input label corresponds to the output acoustic feature in the current time step. Then the context vector is computed by the weighted sum of encoder output with the probabilities the attention mechanism

assigns as weights. The final output of the attention framework is the concatenation of the context vector and the output from the attention RNN. The output from the attention network is converted to acoustic features by RNN or linear transform.

Table 3.1 summarizes components of Tacotron and Tacotron2. Tacotron and Tacotron2 build the encoder and decoder with RNNs. Tacotron uses bidirectional GRU [85] RNN with convolution banks and highway network [86] whose structure is referred to as CBHG for its encoder, where CBH stands for Convolution Banks, Highway networks [86] and G stands for bidirectional-Gated recurrent unit (GRU) [85] RNN. Tacotron2 uses LSTM RNN encoder with CNN. Tacotron used GRU RNN decoder, and Tacotron2 uses LSTM RNN decoder. For alignment, Tacotron and Tacotron2 use soft attention with a different attention score calculation method. Tacotron uses additive attention mechanism [83], and Tacotron2 uses a location-sensitive attention mechanism [58].

Differences between Tacotron and Tacotron2 provides insights to improve the quality of synthetic speech. Changes about transition from Tacotron to Tacotron2 can be classified into three groups of changes: 1) simplified neural network modules, 2) increased model parameter size, and 3) introduction of a neural vocoder. Table 3.2 summarizes the network structure and parameter sizes of Tacotron and Tacotron2. Tacotron uses an encoder that consists of pre-net [1] and CBHG modules. Tacotron2 generally has simpler network structures than Tacotron: its encoder does not have pre-net [1], and the CBH module is replaced with simple CNN layers. Unlike Tacotron, Tacotron2 does not have the decoder RNN and instead has one additional layer in the attention RNN, so Tacotron2 has two layers in the attention RNN. The CBHG module in post-net [1] is also replaced with CNN layers, and the role of post-net is to improve possibly oversmoothed mel-spectrograms from the autoregressive decoder, instead of converting the scale of the spectrogram from mel into linear. Meanwhile, Tacotron2 has an advanced attention mechanism. Tacotron2 has adopted location-sensitive attention [58], an extended version of additive attention [55]. Location-sensitive attention can consider location in addition to content to align the source and target, so it is suitable for speech tasks and was originally tested in speech recognition. In contrast, additive attention was proposed for machine translation.

In addition to the simplified network structure, the model parameter size is increased significantly in Tacotron2. Table 3.2 also compares the parameter sizes of neural

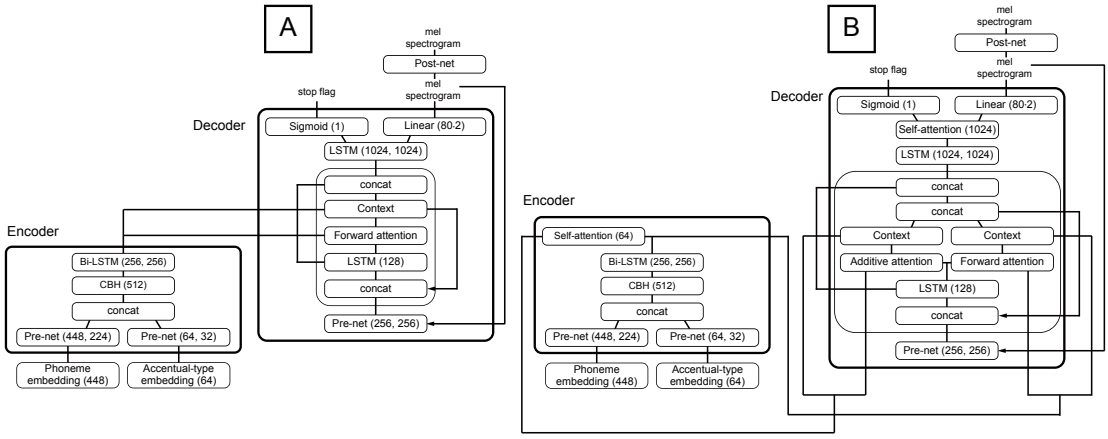


Figure 3.1: A: Neural network architecture of baseline Tacotron. B: Neural network architecture of self-attention Tacotron. Numbers in brackets are parameter size for layers. Arrows indicate feedback from the previous time step. Note that displayed parameter sizes are adjusted to "large" configuration for experiments conducted in this paper. Note that post-net on mel-spectrogram is added for the experiments in this study (see section 4.4.3) and was not used in the original work [1, 2].

network layers in Tacotron and Tacotron2. Along with the expansion of parameter size, many regularization techniques have been introduced, probably because models with huge capacity are more likely to suffer from overfitting. Dropout [64] is applied to every CNN layer in the encoder and post-net. Zoneout [87] regularization is applied to long short-term memory (LSTM) layers in the encoder and decoder. The L2 regularization term is added in the loss term. In contrast, in Tacotron, regularization is conducted only in the encoder pre-net and decoder pre-net through dropout.

Tacotron2 uses WaveNet [12] for waveform synthesis instead of the Griffin-Lim algorithm [73]. Human-level naturalness of synthetic speech was achieved by training WaveNet with a predicted spectrogram from Tacotron2. In other words, WaveNet can correct errors in a predicted spectrogram from Tacotron2 during waveform generation in this configuration.

3.2.2 Baseline Tacotron

We use a modified version of Tacotron [2] for our experiments and refer to it as the baseline Tacotron. Figure 3.1-A illustrates its network structure, and Table 3.2 lists its

details.

The baseline Tacotron encodes characters or phonemes with CBH networks [86] and bidirectional-LSTM [88] modules. Note that we replace the GRU cell [85] in the CBHG module with a LSTM cell to apply zoneout regularization [87]. Zoneout regularization along with the LSTM cell is introduced in Tacotron2 [89, 22], and we use LSTM with the zoneout regularization for all RNN layers including the attention RNN and decoder RNN. We refer to the modified encoder module as "CBHL" from here on.

For an attention mechanism, we use forward attention without a transition agent [25] instead of additive attention [55]. As mentioned in [25], the forward attention accelerates the alignment learning speed. In addition, forward attention is proved to give fewer alignment errors than additive and location-sensitive attention, which are used in Tacotron and Tacotron2 [59], and we observed the same trends in our preliminary experiments.

The output of the attention framework is processed by the decoder RNN. We use the LSTM cell for the decoder RNN. The output of the decoder RNN is projected to the mel-spectrogram with a linear layer as the final output of the network. In addition to the mel-spectrogram, the baseline Tacotron predicts stop flags [34, 22] to determine when to stop the prediction loop.

3.2.3 Self-attention Tacotron

We use the self-attention Tacotron proposed in [2] as a system to be investigated. The motivation to use self-attention in Tacotron is to better capture long-term dependency to model pitch accent, which is phrase level information. It is known that by directly connecting distant states, self-attention relieves the high burden placed on LSTM to learn long-term dependencies to sequentially propagate information over long distances [90].

Figure 4.5-B shows the network structure of the self-attention Tacotron, and Table 3.2 summarizes the network configuration. The difference between the self-attention Tacotron and the baseline Tacotron is the existence of self-attention layers in both the encoder and decoder. We used a self-attention block based on [91]. The self-attention has multiple heads, and a residual connection is placed after the self-attention layer. As in [90], the self-attention block is placed after the LSTM layers. The

encoder has two outputs: output of the bidirectional LSTM layer and output of the self-attention layer. The two encoder outputs are aligned to output acoustic features with different attention mechanisms in dual source attention [92]. The output from the bidirectional LSTM layer is fed to forward attention [25], and the output from the self-attention layer is fed to additive attention [55]. By using attention for each encoder output, we can check how the encoder outputs are used from the decoder by visualizing the attention probability distribution. In our previous study, we discovered that the forward attention captured the source-target relationship because it showed monotonic alignment, and the additive attention captured the phrase structure. The outputs from two attention layers are concatenated and processed by the decoder RNN followed by self-attention.

3.3 Pipeline TTS framework

3.3.1 Comparison between end-to-end and pipeline TTS framework

The pipeline TTS framework is one of the classical TTS frameworks as described in Chapter 2. We include the pipeline TTS system in our analysis to know the performance of end-to-end systems compared with the classical approach. The pipeline TTS framework is interesting to compare to in terms of pitch accents because it relies on labels related to pitch accents predicted by text processing front-end, unlike end-to-end TTS. This section clarifies conditions for fair comparison between conventional pipeline TTS and end-to-end TTS to avoid obscuring interesting differences by other factors.

The end-to-end TTS and DNN-based pipeline TTS frameworks have a few aspects in common. Both methods can benefit greatly from autoregressive modeling of the acoustic features sequences [74, 46] and neural waveform generators [74, 34, 22]. Although it is agreed that end-to-end TTS can generally generate more natural speech than traditional methods such as pipeline SPSS, few studies have fully investigated this under the same conditions of density modeling and a waveform generation method. For example, Tacotron [1] and Tacotron2 [22] were evaluated against a HMM-based unit selection system [93] and a DNN-based pipeline system without

either a neural waveform generator or an autoregressive acoustic model [94]. Some recent studies included a DNN-based pipeline using neural waveform generators, but the non-autoregressive acoustic models still affect the fairness of the comparison [95, 34].

Therefore, if we want to compare pipeline based TTS and sequence-to-sequence based TTS from a framework point of view, it is desirable to select comparable pipeline systems that use an autoregressive decoder with the same waveform generation method as sequence-to-sequence based TTS. Otherwise, differences in the experimental results for the two different approaches can be expected to mainly be caused by the difference in waveform generation techniques and the assumption of probabilistic modeling.

The largest scale experiment to compare various TTS systems including sequence-to-sequence based methods and pipeline methods is perhaps the Blizzard challenge 2019 [96], in which the task is single-speaker Chinese TTS. Among 24 participating systems, at least 14 were sequence-to-sequence based TTS methods, 5 were SPSS methods, and 2 were unit selection. In addition, one system was natural speech and one was Merlin benchmark [97]. The challenge did not intend fair comparison, because the participants could use various methods of waveform generation and even manual labeling or data argumentation with other corpora. Nevertheless, sequence-to-sequence based TTS systems using a neural vocoder obtained relatively high ranks, and the top score was achieved by a DNN-based SPSS system using a neural vocoder, BERT (Bidirectional Encoder Representations from Transformers) based front-end, autoregressive duration model, and non-autoregressive acoustic model with a GAN post-filter[98].

3.3.2 Pipeline TTS method used for analysis

We need a pipeline TTS system which utilizes an autoregressive density acoustic model and a neural vocoder to compare with our end-to-end TTS systems, as described in Section 3.3.1. We thus use a DNN-based SPSS system proposed by [16] for a pipeline TTS method. This pipeline TTS system uses an acoustic model with autoregressive density modeling and a neural vocoder, so we can compare it fairly with our end-to-end TTS systems.

Figure 3.2 shows the architecture of acoustic models in the DNN-based SPSS system

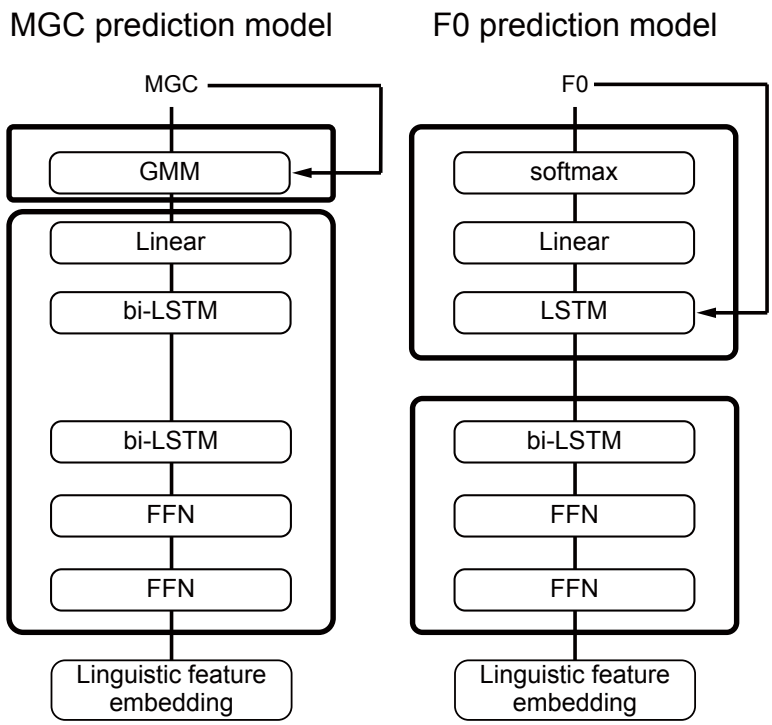


Figure 3.2: Acoustic models of pipeline TTS systems used in our experiments. A: DNN-based SPSS; left is MGC prediction model based on autoregressive recurrent mixture density network; right is F_0 prediction model based on deep autoregressive recurrent neural network. B: Mel-spectrogram prediction model of Vocoder-free TTS.

used in this study. This system consists of two acoustic models to predict two vocoder parameters. The first model in the DNN-based SPSS system is responsible for modeling the mel-generalized cepstral coefficient (MGC). It is based on a bidirectional recurrent mixture density network (RMDN) [99], but autoregressive feedback connection is introduced at the output layer to improve density modeling with AR [70]. The autoregressive feedback from past adjacent frames is linearly transformed to compute output density parameters. The second model in the DNN-based SPSS system is a F_0 model. The F_0 output modeled in this model is discrete, based on a just-noticeable difference in pitch in speech perception [100]. Thus, its output is a softmax distribution. This model also has also autoregressive feedback, but the feedback is nonlinearly transformed by a stack of neural network layers. In addition, this feedback scheme has an architecture consisting of two network modules. One module is bidirectional RNN

and additional network layers that take linguistic input, and the other is a unidirectional RNN layer that runs an autoregressive loop. The former module corresponds to an encoder and the latter corresponds to a decoder in a sequence-to-sequence framework [40]. However, unlike a sequence-to-sequence framework, this architecture's input is a frame level linguistic feature aligned by a duration model, so source and target mapping is frame-by-frame.

3.4 Experiments

In experiments, we investigate how the following configurations affect prediction accuracy of alignment and pitch accent in end-to-end TTS systems.

1) Model parameter size
2) Encoder network structures (CBHL [1] / CNN-based encoder [22])
3) Presence of self-attention
4) Linguistic feature (character / phones)

Model parameter size affects model's capacity, the amount of information a model can memorize. Increasing model parameter size is expected to improve alignment stability and quality of synthetic speech, as Tacotron2 shows. Encoders are a module of neural network layers which encode linguistic features to extract an acoustic representation. Resolving pronunciations from surface from is challenging, and encoder structure is expected to affect the level of disambiguating pronunciation including intonation. For example, a CBHG encoder is designed to extract pronunciations from surface form well [101]. Self-attention is a neural network structure known to be good at capturing long-term information. We expect capturing long term information with self-attention will improve alignment and intonation. We use phone sequences in addition to characters as linguistic features to clarify levels of pronunciation disambiguation from characters, and its effect on alignment and intonation.

Additionally, we compare end-to-end TTS systems with pipeline TTS system. Pipeline TTS uses full-context labels which include pitch accent information predicted by front-end system. Including a comparable pipeline TTS system enables us to

evaluate how much end-to-end TTS systems can infer pronunciation from limited linguistic inputs such as characters and phones.

3.4.1 Experimental conditions

Database

We chose the Blizzard 2011 corpus [102] as an English speech corpus. This corpus contains 12,092 utterances or around 17 hours in total duration from a female speaker. We used characters or phones for input. The Blizzard 2011 corpus is suitable for comparing character and phone input because it has a wider variety of unique word types (18,695 unique words) [48], so we can thoroughly check the generalizability of our systems using character input. We obtained phone labels using Flite [103]. All characters were converted into lowercase, but no elaborate text normalization was performed. We trimmed the beginning and end silences from the utterances, after which the duration of the corpus was reduced to 15.7 hours. We used 11,092 utterances for training, 500 for validation, and 500 for testing.

Tacotron configurations

We built 16 English Tacotron models in order to investigate the effect of configurations at the beginning of this section: each model is either small (equivalent to Tacotron) or large (equivalent to Tacotron2) in model parameter size for 1) model parameter size, uses CBHL or CNN-based encoder for 2) encoder structure, has or does not have self-attention for 3) presence of self-attention, and takes phone or raw text as input for 4) linguistic feature.

Waveforms were generated with μ-law WaveNet [12] trained with the ground-truth mel-spectrogram.

Objective evaluation

We measured fatal alignment error rates for objective metrics about alignment stability. There are three types of fatal alignment errors (discontinuous alignment, incomplete alignment, and overestimated duration) that can be easily detected with software or human eyes by examining the attention probability distribution. Specifically, the

discontinuous alignment error is indicated by a skip or repeat of a transcription and can be detected by finding non-monotonic jumps of the attention distribution mode. The incomplete alignment error is indicated by a premature termination of a transcription and can be detected by finding an alignment distribution mode that does not reach the final input position at the final time step. The overestimated duration error is signified by an excessively prolonged phoneme and can be detected by finding an attention distribution mode staying in the same position for a long time.

We measured the error rates of the 16 Tacotron models on the test set with 500 utterances. The 16 Tacotron systems are baseline Tacotron and self-attention Tacotron with combinations of the two parameter sizes: small or large; the two encoder structures: CBHL or CNN; and the two linguistic features: phones or characters.

We measured alignment error rates on each system several times. Each time, we initialized the system with a different random seed and trained it from scratch. After generating the 500 test set utterances, we counted the alignment errors using in-house software.

Subjective evaluation

We conducted a listening test about the naturalness of synthetic speech. We recruited 49 native English listeners via crowdsourcing. Listeners were asked to evaluate 36 samples using the five-grade MOS metric in one set. One listener could evaluate at most 20 sets. We collected 17,928 evaluations in total. We checked the statistical significance of the scores between systems with a Mann-Whitney rank test [104]. For the subjective evaluation, we excluded Tacotron systems with the small parameter size since they have obviously worse objective evaluation results as described later and may cause the ceiling effect that disturbs the analysis of other factors. Therefore, the listening test included 12 systems in total: eight Tacotron systems, one DNN based SPSS system using vocoder parameters [16], two analysis-by-synthesis systems from WaveNet used for the pipeline and Tacotron, and natural samples.

3.4.2 Experimental results

Tables 3.3a and 3.3b shows the number of samples containing fatal alignment errors from systems using phones and characters as an input, respectively. All systems with

the small model parameter size showed a relatively large alignment error rate, ranging from 7.6 % to 23.4 % on average. Among the systems with the small parameter size, systems using self-attention had especially high alignment error rates. One possible reason is that the combination of post-net and self-attention made alignment unstable. All systems with the large model parameter size had only small alignment errors. The alignment error rates from these systems were consistent across multiple runs with different random initial seeds. Differences of encoder and linguistic input did not impact the fatal alignment error rates. Self-attention in models with large parameter size had no impact on the fatal alignment error rates as well.

Overall, 1) for model parameter size, the large-parameter-size configuration helped to improve and stabilize alignment and its learning, whereas the small-parameter-size configuration made it unstable and sensitive to initial parameters and network structure. 2) Encoder network structures and 4) linguistic feature did not affect alignment stability. 3) Self-attention made alignments unstable for models with small parameter size, but did not effect models with large parameter size.

We conducted a listening test to investigate the impact of configurations other than the parameter size by using systems with the large parameter size. Figure 3.3 shows the results of the listening test in English. Refer to Table 3.4 for the statistical significance of the test. First, we can see that when the encoder was based on CNN, both systems with and without self-attention using phones had higher scores than corresponding systems using characters. The baseline Tacotron with the CNN based encoder had a score of 3.08 ± 0.05 when characters were used and 3.32 ± 0.05 when phones were used. Self-attention Tacotron with the CNN based encoder had 3.01 ± 0.05 when characters were used and 3.19 ± 0.05 when phones were used.

Second, interestingly, we can see that when the CBHL encoder was used, the score gap caused by the difference in input features vanished. The baseline Tacotron with the CBHL encoder had 3.50 ± 0.04 when characters were used and 3.49 ± 0.04 when phones were used. Self-attention Tacotron with the CBHL encoder had 3.57 ± 0.04 both when characters and phones were used.

Third, self-attention did not show statistical significance except for systems using the CNN encoder and phone inputs, and their score difference seemed to represent just their alignment error rates. Finally, the Tacotron systems did not match the pipeline system in terms of naturalness.

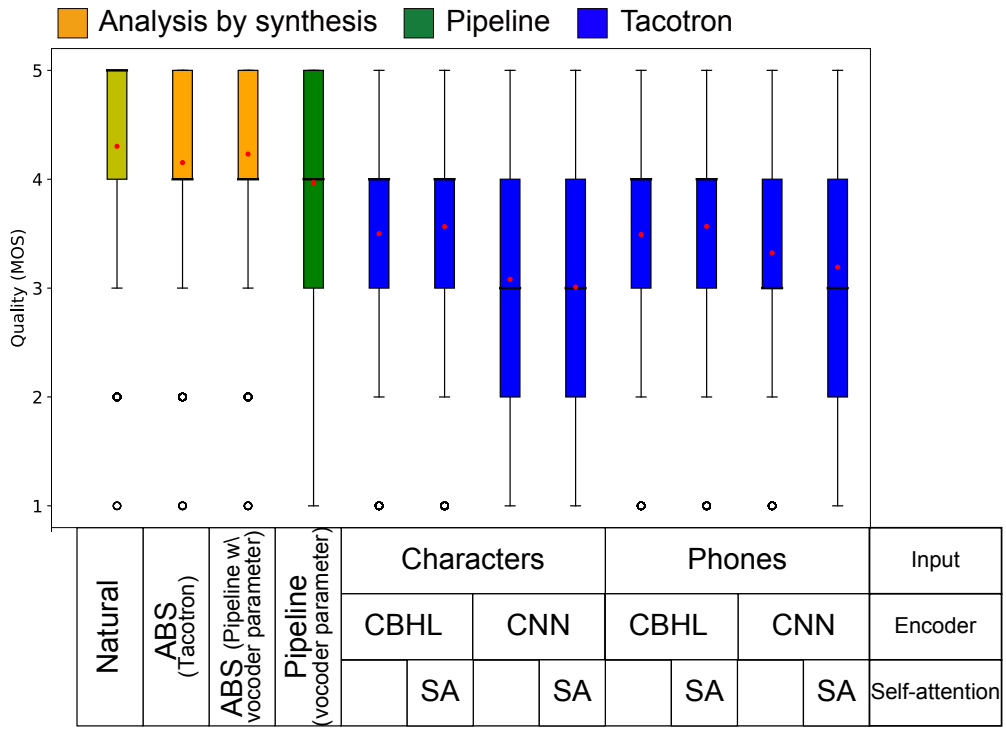


Figure 3.3: Box plot for the results of English listening test. Red dots indicate mean, and black bars indicate median.

We investigated variation of pitch from the two best Tacotron systems: self-attention Tacotron using characters and self-attention Tacotron using phones. Figure 3.4 shows standard deviations of F_0 against the output waveform length for samples from our Tacotron systems and the pipeline system. We can clearly see that samples predicted from our Tacotron system have smaller standard deviations than samples predicted from the pipeline across the whole output length. The average standard deviations of F_0 in the voiced region of generated samples in a test set were 48 Hz for self-attention Tacotron using characters, 46 Hz for self-attention Tacotron using phones, and 59 Hz for the pipeline system. We investigated samples from the two Tacotron systems. We listened to their samples that had relatively low average mean opinion score (MOS) whereas corresponding samples from the pipeline had high MOS of over 4.5. We found that the samples from the Tacotron systems had relatively flat pitch change, which resulted in unnatural prosody. In addition, we found that the low-score samples contain errors such as mispronunciations or wrongly stressed

syllables.

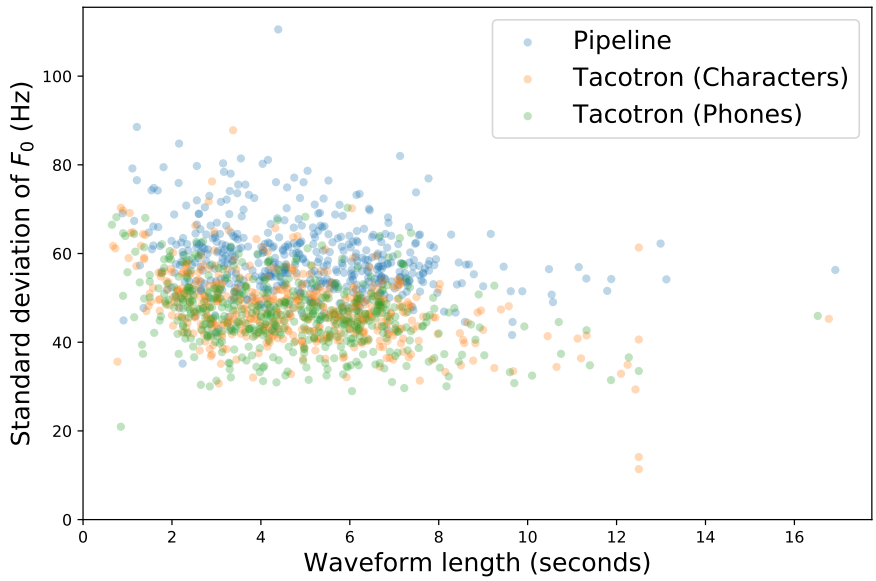


Figure 3.4: Standard deviation of F_0 against output waveform length for samples predicted from our Tacotron systems and pipeline system.

3.4.3 Conclusion

This experiment showed: 1) Increasing model parameter size helped to learn and predict alignments between the source and target more robustly. 2 & 4) Encoder structure had a great impact on the naturalness of the synthetic speech, but the naturalness came from disambiguating phoneme pronunciation from surface form, not improving alignment stability or intonation. 3) Self-attention did not affect naturalness or alignment quality. Our Tacotron systems were outperformed by the pipeline system probably due to poor pitch accents.

A few fatal alignment errors were observed, although increasing model parameter size improved the fatal alignment error rates. This is because soft-attention can not guarantee monotonic structure of alignment. It is desirable to avoid fatal alignment errors completely, because fatal alignment errors are unacceptable in terms of intelligibility of synthetic speech. If we want to avoid fatal alignment errors completely, we need a new alignment method other than soft-attention, and we cover it in Chapters 6

and 7.

The reason our Tacotron systems were outperformed by the pipeline system is unnatural prosody. It indicates predicting pitch accent from surface form is quite difficult, and increasing parameter size can not improve the prosody up to the pipeline TTS level. Similar problems are reported from a few studies using Tacotron based systems. According to [22], unnatural prosody including unnatural pitch was the most common error in their manual analysis for 100 challenging English sentences. This result motivates us to introduce pitch accent modeling to end-to-end TTS model, and we cover this in Chapter 5. Before we tackle pitch accent modeling, we the investigate learning of pitch accents in Chapter 4.

Table 3.2: Comparison of structure and configuration between Tacotron and Tacotron2 in literature and modified Tacotron models in this study. Baseline Tacotron and Self-attention Tacotron correspond to models A and B illustrated in figure 3.1. Numbers in bracket tuple indicate size of hidden layer(s). For modified Tacotrons, number tuples before and after "/" denote configuration for small and large model parameter sizes, respectively. Note that CBH block in encoders of modified Tacotron can be replaced with CNN. Pho., and accen., and para. denote phoneme, accentual type, and parameter, respectively.

Model		Tacotron in literature		Modified Tacotron in this study	
		Tacotron [1]	Tacotron2 [22]	Baseline Tacotron	Self-attention Tacotron
Vocoder		Griffin-Lim [73]	WaveNet [12]	WaveNet, open-source version	
Decoder	Post-net	CBHG (256)	CNN (512)	CNN (512)	CNN (512)
	Self-attention	-	-	-	Self-attention (256) / (1024)
	Decoder RNN	GRU (256, 256)	-	LSTM (256, 256) / (1024, 1024)	LSTM (256, 256) / (1024, 1024)
	Attention	Additive (256)	Location-aware (128)	Forward (256) / (128)	Additive & Forward (256) / (128)
	Attention RNN	GRU (256)	LSTM (1024, 1024)	LSTM (256) / (128)	LSTM (256) / (128)
	Pre-net	FFN (256, 128)	FFN (256, 256)	FFN (256, 128) / (256, 256)	FFN (256, 128) / (256, 256)
Encoder	Self-attention	-	-	-	Self-attention (32) / (64)
	Encoder core	CBHG (256)	Bi-LSTM (512)	Bi-LSTM (256) / (512)	Bi-LSTM (256) / (512)
			CNN (512)	CBH (128) / (256)	CBH (128) / (256)
	Pre-net	FFN (256, 128)	-	FFN Pho. (224, 112) / (448, 224) FFN Accen. (32, 16) / (64, 32)	FFN Pho. (224, 112) / (448, 224) FFN Accen. (32, 16) / (64, 32)
	Embedding	(256)	(512)	Pho. (224) / (448) Accen. (32) / (64)	Pho. (224) / (448) Accen. (32) / (64)
# Para. w/o vocoder		6.9×10^6	27.3×10^6	11.3×10^6 / 35.8×10^6	11.6×10^6 / 41.6×10^6

Table 3.3: Alignment error rate on English test set. Each system was evaluated for three times, and each time it was trained from scratch with a different initial random seed. Ave. indicates average error rate. Values in bold correspond to those evaluated in a listening test.

Para. size	Encoder	Self-attention	# Para. (1×10^6)	Alignment error rate (%) Ave.	1	2	3
Small	CBHL	-	7.9	7.6	6.2	8.6	9.0
		✓	12.1	17.9	10.0	10.4	33.2
	CNN	-	9.2	9.3	5.0	5.2	17.8
		✓	9.9	15.6	13.0	16.6	17.2
Large	CBHL	-	36.7	1.0	**0.6**	0.8	1.6
		✓	47.6	0.6	**0.2**	0.6	1.0
	CNN	-	27.2	1.1	**0.2**	1.4	1.6
		✓	38.1	1.0	0.8	1.0	**1.2**

(a) Systems using phone as input

Para. size	Encoder	Self-attention	# Para. (1×10^6)	Alignment error rate (%) Ave.	1	2	3
Small	CBHL	-	11.3	14.9	7.4	8.6	28.8
		✓	12.1	23.4	18.8	21.8	29.6
	CNN	-	9.2	11.1	6.6	10.8	15.8
		✓	9.9	18.0	15.0	16.8	22.2
Large	CBHL	-	36.7	1.1	0.8	**1.0**	1.6
		✓	47.6	0.8	0.4	**1.0**	1.0
	CNN	-	27.2	0.5	**0.2**	0.6	0.6
		✓	38.1	0.7	**0.4**	0.6	1.0

(b) Systems using character as input

Table 3.4: Mann-Whitney rank test for English listening test. Cell in this color denotes statistical significance ($p \leq 0.01$) while this color denotes $p > 0.01$.

				Natural	ABS		Pipeline Vocoder	Tacotron (Large parameter size)								
								Charater				Phone				Input
								CBHL		CNN		CBHL		CNN		Encoder
									✓		✓		✓		✓	Self-att.
Natural																
ABS	Tacotron Mel. spec.															
ABS	Vocoder para.															
Pipeline vocoder para.																
Tacotron (Large parameter size)	Character	CBHL	-													
Tacotron (Large parameter size)	Character	CBHL	✓													
Tacotron (Large parameter size)	Character	CNN	-													
Tacotron (Large parameter size)	Character	CNN	✓													
Tacotron (Large parameter size)	Phone	CBHL	-													
Tacotron (Large parameter size)	Phone	CBHL	✓													
Tacotron (Large parameter size)	Phone	CNN	-													
Tacotron (Large parameter size)	Phone	CNN	✓													

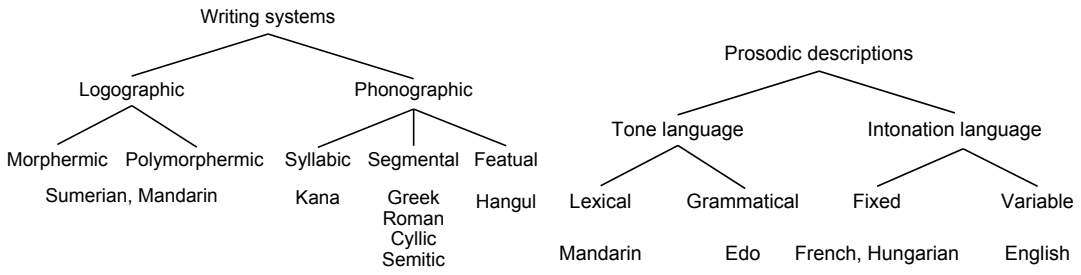
4

Learning pitch accent in end-to-end TTS

4.1 Pitch accent

Pitch accent is accent involving pitch change. Generating speech with correct pitch accents is challenging especially for end-to-end TTS because end-to-end TTS does not rely on elaborated linguistic features such as part-of-speech, which is helpful information to infer types of pitch accents.

Pitch accent is one of the most important features for prosody. Prosodic description is different among languages. Figure 4.1b shows a classification of languages based on prosodic description. There are two major groups in prosodic description: tone and intonation languages. Prosodic description of tone languages such as Mandarin and Japanese are lexical, which means pitch accents (tones) are determined based on linguistic units such as syllables or words. Intonation languages are further classified into fixed or variable. Fixed intonation languages have fixed accent position within words. Variable intonation languages do not have fixed accent position and pitch accents can not be determined straightforwardly. English has variable intonation



(a) Writing systems (b) Prosodic descriptions

Figure 4.1: Classification of languages based on writing systems and prosodic descriptions, based on [3].

of prosodic description, and we saw end-to-end TTS systems struggled to generate natural pitch change in English in Chapter 3.

The relationship between prosodic description and writing system is what makes the learning of pitch accents difficult for end-to-end TTS. Figure 4.1a shows classification of languages based on writing system. Logographic writing systems use logograph characters which represent meaning. Phonographic writing systems use characters which represent sound. Tone languages such as Mandarin and Japanese adopt logographic writing systems. Intonation languages such as English adopt phonographic writing systems. Surface form, and lexical information derived from it, is necessary to infer pitch accent, but using surface form as input for TTS is normally difficult in logographic languages because large the diversity of logographic characters results in data sparsity. Phonemes are normally used as input for TTS for logographic languages, and lack of surface form information makes predicting pitch accents difficult. Surface form can be used as input for TTS in phonographic languages, but intonation is not necessarily lexical, and is difficult to infer.

4.2 TTS for pitch accent languages: Japanese as an example

We overview Japanese as a pitch accent language and topics involving Japanese TTS in this section. Japanese is classified as a tonal language in terms of prosodic description,

but is usually referred to as a pitch accent language because of its limited tone patterns. In this chapter, we focus on Japanese as a tone language. We cover English as an intonation language in Chapter 3.

Japanese pitch accents are lexical, which depends on words or combinations of words. Japanese is a ”mora-timed” pitch-accent language; this means there is an accent nucleus position counted in mora units within an accentual phrase. Japanese pitch accent thus directly affects the meaning of words and the perceived naturalness of the speech, unlike the pitch accent in English. Although the pitch accent of individual words can be encoded in a lexicon, it is affected by adjacent words in an accentual phrase, i.e., pitch accent sandhi.

Word segmentation is one of the Japanese specific features of front-end systems in pipeline TTS. Because the Japanese orthographic system does not have clear word boundaries, a Japanese front-end system uses morphological analysis to resolve word segmentation. A look up lexicon is used in word segmentation task to resolve word pronunciation, word-level accentual type, and other information such as part-of-speech.

Accent sandhi estimation is another Japanese specific feature in front-end systems. Japanese has pitch accents which can be defined with the position of pitch decline in an accentual phrase, which is referred to as an accent nucleus position or accentual type. Although accent nucleus position in a word can be resolved with accent information in a lexicon entry from a dictionary, the position may change depending on adjacent words, caused by accent sandhi, in which words are concatenated to create larger accentual phrases. There are accent sandhi estimation based on rules [105] and estimation based on machine learning [106, 107]. For a concrete example, Open JTalk [108], a widely used front-end tool for Japanese TTS, uses a morphological analyzer; a pronunciation dictionary including accentual type, accent modification type, and accent concatenation type; and rule based accent sandhi estimation.

Japanese presents two difficulties to achieving end-to-end TTS. The first problem is character diversity. Japanese uses thousands of ideograms called kanji (i.e., logographic Chinese characters) alongside two syllabic scripts called hiragana and katakana, which prevent it from being used as direct input for end-to-end TTS. The second is pitch accent, which is an accent involving pitch change. Japanese pitch accent greatly impacts the naturalness of speech because it affects the meaning of words, yet it cannot be determined with simple rules. In this chapter, we ignore the first problem by using

phonemes as inputs to avoid data sparsity. Accordingly, the second problem can not be solved properly because surface form information is lost by using phonemes as input. We thus provide accentual type labels as minimum inputs for end-to-end TTS to produce pitch accents, and investigate various factors to affect pitch accent prediction capability. Figure 4.2 shows an example format of linguistic feature consisting of accentual types and phonemes which are what we use as inputs for our end-to-end TTS systems in this chapter.

Accentual type	1							1			
Phoneme	f	e	N	s	Cu	w	Ca	h	o	ry	Co

Figure 4.2: An example format of linguistic feature sequence consisting of accentual types and phonemes.

4.3 TTS systems used in this investigation

4.3.1 Baseline Tacotron using phoneme and accentual type

In this section, we describe Japanese Tacotron which is the baseline Tacotron in Section 3.2.2 in Chapter 3 which is slightly modified to handle Japanese accentual-type labels. We refer to this system as *JA-Tacotron*. Figure 4.3-A shows its architecture. Tacotron is a sequence-to-sequence architecture [18] that consists of encoder and decoder networks. Unlike classical pipeline systems with explicit duration models, Tacotron uses an attention mechanism [55] that implicitly learns alignments between source and target sequences. In this paper, we use phoneme and accentual-type sequences as a source and mel-spectrogram as a target as our first investigation towards end-to-end Japanese speech synthesis. This baseline architecture is inspired from [25], which applied Tacotron to the Chinese language. On the encoder side, phoneme and accentual-type sequences are embedded to separate embedding tables with different dimensions, and the embedding vectors are bottle-necked by their corresponding pre-nets [1]. The two inputs are then concatenated and encoded by Convolution Banks, Highway networks,

bidirectional-LSTM (CBH-LSTM) with zoneout regularization [87].

At the decoder, encoded values are decoded with attention based LSTM decoder. We use forward attention [25] as an attention mechanism, which is an extension of additive attention mechanism [55] used by the original Tacotron to enforce monotonic alignment. As suggested in [25], the forward attention accelerates the alignment learning speed and provides distinct and robust alignment with less training time than the original Tacotron. The decoder LSTM is regularized with zoneout as well as the encoder since it is expected that the zoneout regularization will reduce alignment errors. We set the reduction factor to be two so that the decoder outputs two frames at each time step. A predicted mel-spectrogram is converted to an audio waveform with WaveNet [109]. We use a frame shift of 12.5 ms for the mel-spectrogram to train the *JA-Tacotron* model as in [22][1]

4.3.2 Self-attention Tacotron using phoneme and accentual type

A pitch-accent language like Japanese uses lexical pitch accents that involve a f_0 drop within an accentual phrase. The accentual phrases in Japanese normally have mora of varying lengths. Since the length of an accentual phrase could be very long, we hypothesize that long-term information plays a significantly important role in TTS for pitch accent languages.

Therefore, we include self-attention Tacotron, which is also used in experiments in Chapter 3. Self-attention Tacotron introduces "self-attention" after LSTM layers at the encoder and decoder as illustrated in Figure 4.3-B. It is known that by directly connecting distant states, self-attention relieves the high burden placed on LSTM to learn long-term dependencies to sequentially propagate information over long distances [90]. This extension is inspired from a sequence-to-sequence neural machine translation architecture proposed by [91]. We refer to this architecture as *SA-Tacotron*.

To recap, the self-attention block consists of self-attention, followed by a fully connected layer with tanh activation and residual connection. We use multi-head dot product attention [91] as an implementation of self-attention. This block is inserted

[1]Our WaveNet model for *JA-Tacotron* is trained by fine-tuning using a ground truth mel-spectrogram with a frame shift of 12.5 ms starting with an existing model trained with a mel-spectrogram with a frame shift of 5 ms in order to make comparison with TTS systems using vocoder parameters fairer. We use softmax distribution as an output layer of WaveNet.

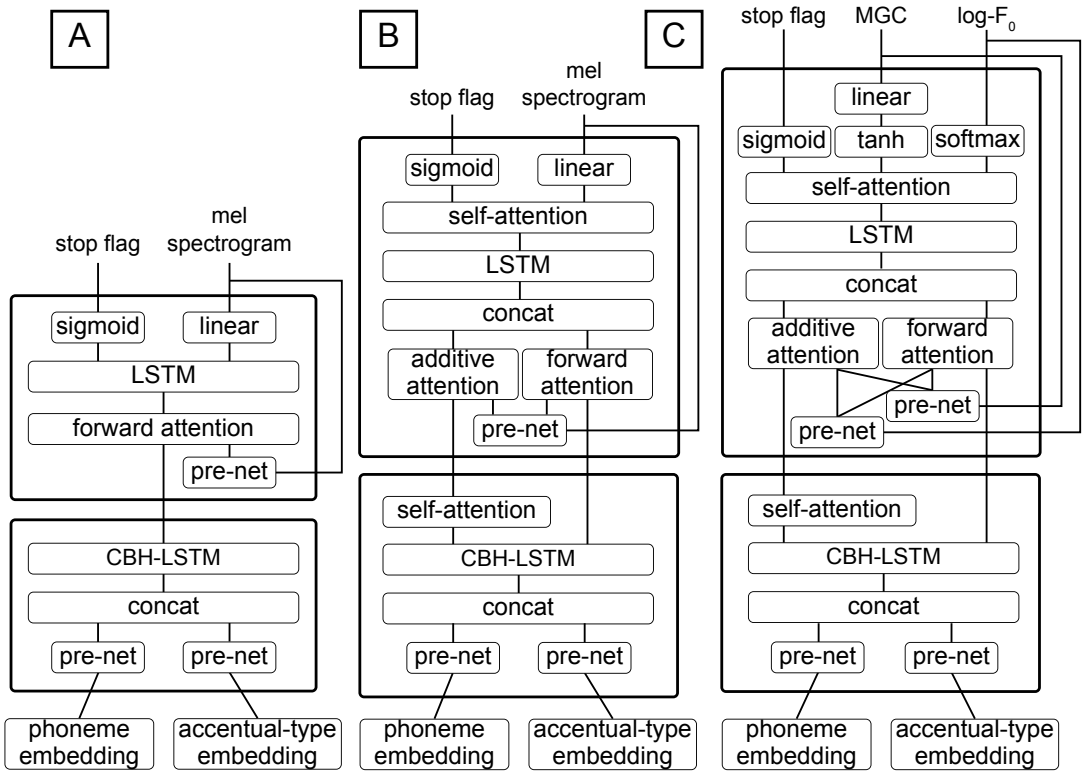


Figure 4.3: Architectures of proposed systems with accentual-type embedding. A: *JA-Tacotron*. B: *SA-Tacotron*, C: *SA-Tacotron* using vocoder parameters.

after LSTM layers at the encoder and decoder. At the encoder, the output of CBH-LSTM layers is processed with the self-attention block. Since LSTM can capture the sequential relationships of inputs, we do not use positional encoding [35]. Both self-attended representation and the original output of the CBH-LSTM layers are final outputs of the encoder.

At the decoder, the two outputs from the encoder are attended with a dual source attention mechanism [92]. We choose a different attention mechanism for each source, forward attention for the output of CBH-LSTM and additive attention for the self-attended values. This is because we want to utilize the benefits of both: forward attention accelerates alignment construction, and additive attention provides flexibility to select long-term information from any segment. In addition, we can visualize both alignments. Unlike the encoder, self-attention works autoregressively at the decoder. At each time step of decoding, the self-attention layer attends all past frames of LSTM

outputs and outputs only the latest frames as a prediction output. The predicted frames are fed back as input for the next time step.[2]

4.3.3 Tacotron using vocoder parameters

Explicitly modeling the fundamental frequency (f_0) might be a more appropriate choice for TTS systems for pitch-accent languages. To incorporate F_0 into the proposed systems, we further developed a variant of *SA-Tacotron* by using vocoder parameters as targets. We use mel-generalized cepstrum coefficients (MGC) and discretized $\log f_0$ as vocoder parameters, and we predict these parameters with Tacotron. We choose 5 ms for the frame shift to extract MGC and f_0 as such fine-grained analysis conditions are typically required for reliable speech analysis based on vocoders. However, note that this condition is not a natural choice for training Tacotron, which typically uses coarse-grained condition, usually 12.5 ms frame shifts and 50 ms frame lengths, to reduce input and output mismatch. With a frame shift of 5 ms, the length of target vocoder parameter sequences becomes 2.5 times longer than the normal 12.5 ms condition. In other words 2.5 times longer autoregressive loop iteration is required to predict a target, so this task is much more challenging. To alleviate the difficulty, we set the reduction factor to be 3 in order to reduce the target length. This setting results in 5/3 times longer target length compared to *SA-Tacotron* in the previous section.[3]

Figure 4.3-C shows the modified architecture of the *SA-Tacotron* using MGC and $\log f_0$ as targets. To handle the two types of vocoder parameters, we introduce two pre-nets and three output layers at the decoder. The output layers include a MGC prediction layer that consists of two fully connected layers followed by tanh and linear activations, a $\log f_0$ prediction layer which is a fully connected layer followed by softmax activation, and a stop flag prediction layer, which is a fully connected layer followed by sigmoid activation. We represented discretized $\log f_0$ as one-hot labels at training time, but feed back predicted probability values at inference time [43]. We use

[2]At training time, since all target frames are available, this computation can be parallelized by applying a step mask. Since the decoder depends on LSTM, the whole computation cannot be parallelized, but this optimization decreases memory consumption because all past LSTM outputs do not need to be preserved at each time step to calculate gradients on a backward path in backpropagation algorithm. Thanks to this optimization, we can train the extended architecture with a negligible increase in training time.

[3]We tried larger reduction factors, but the audio quality deteriorated as the reduction factor increased.

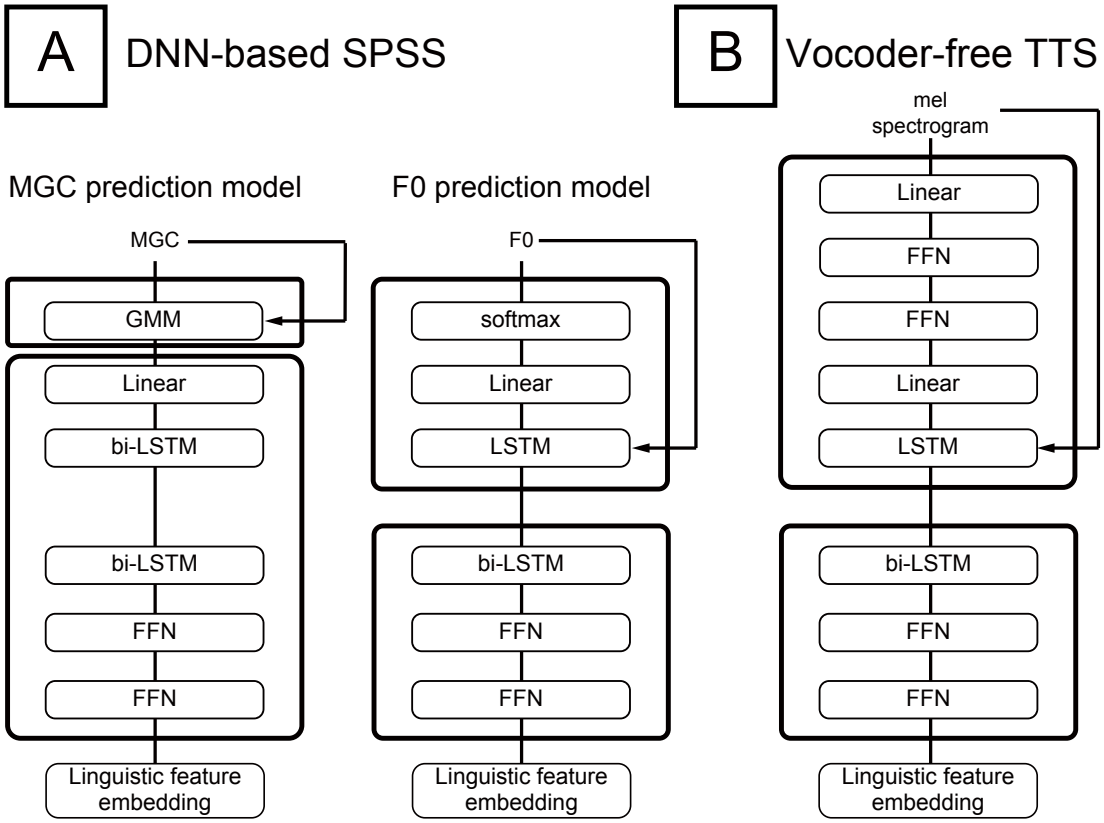


Figure 4.4: Acoustic models of pipeline TTS systems used in our experiments. A: DNN-based SPSS; left is MGC prediction model based on autoregressive recurrent mixture density network; right is F_0 prediction model based on deep autoregressive recurrent neural network. B: Mel-spectrogram prediction model of Vocoder-free TTS.

L1 loss for MGC and cross entropy error for discretized log f_0 and stop flag, and we optimize the model by using the weighted sum of the three losses. The cross entropy error of log f_0 is scaled by 0.45 to adjust its order to the other two loss terms.

4.3.4 Pipeline systems

We used two DNN-based pipeline systems: DNN-based SPSS and vocoder-free TTS. The DNN-based SPSS method is the same system which we used for the English experiment in Chapter 3.

Figure 4.4-A shows the architecture of acoustic models in the DNN-based SPSS system used for our Japanese experiments. This system is described in Section 3.3.2

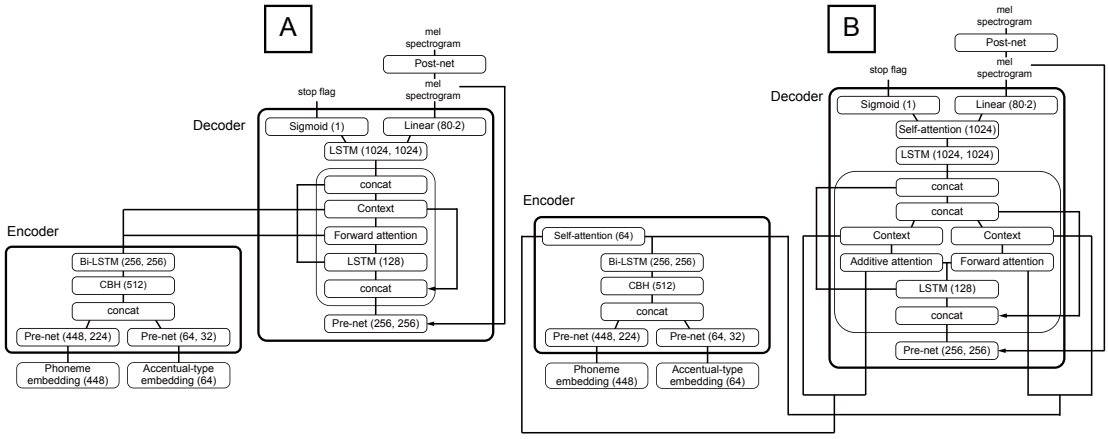


Figure 4.5: A: Neural network architecture of baseline Tacotron. B: Neural network architecture of self-attention Tacotron. Numbers in brackets are parameter size for layers. Arrows indicate feedback from the previous time step. Note that displayed parameter sizes are adjusted to "large" configuration for experiments conducted in this paper. Note that post-net on mel-spectrogram is added for the experiments in this study (see section 4.4.3) and was not used in the original work [1, 2].

in Chapter 3. To recap, it consists of two acoustic models to predict two vocoder parameters. The first model is responsible for modeling the MGC with RMDN [99]. The second model is a F_0 model using autoregressive density [43].

Figure 4.4-B shows the architecture of an acoustic model in the vocoder-free TTS system used in this study. This system was used for a voice cloning study in English [17], its target acoustic feature is the mel-spectrogram. The architecture of the acoustic model consists of two network modules: a bidirectional RNN based module that processes linguistic features, and a unidirectional RNN based module with a deep autoregressive feedback scheme of nonlinear transformation, as well as the F_0 model in the DNN-based SPSS system.

Both the DNN-based SPSS and vocoder-free TTS systems use HSMMs to predict phoneme duration and a WaveNet neural vocoder to generate waveforms. The WaveNet predicts a quantized waveform in 10 bits with μ-law coding at a 16 kHz sampling rate.

4.4 Experiments

We conducted two experiments about Japanese end-to-end TTS. We investigated how various factors affected alignments and pitch accents to Japanese end-to-end TTS in the two experiments. The first experiment is about the following configurations under small model parameter size and CBHL encoder configurations:

1) Presence or absence of pitch accent label
2) Presence or absence of self-attention
3) Predicted or forced alignments
4) Mel-spectrogram or vocoder parameters as acoustic features

The first factor is pitch accent labels: to measure how much end-to-end TTS can produce correct pitch accent if labels are given, and how much pitch accent degrades if accent labels are not given. The second factor is self-attention: to measure how much self-attention helps to improve pitch accents and alignments by capturing long-term information. The third factor is alignment: to measure how much soft-attention can predict correct alignments. Note that forced alignment is expected to produce correct alignments by teacher forcing. The fourth factor is an acoustic feature to measure if f_0 in vecoder parameter is beneficial to predict pitch accents or if mel-spectrogram is enough to produce correct pitch accents.

The second experiment is about the following configurations under the presence of pitch accent labels and natural alignment configurations:

1) Model parameter size (small or large)
2) Encoder network structure (CBHL or CNN)
3) Presence or absence of self-attention

The first factor is model parameter size: to see how much increasing the model parameter size improve alignments and pitch accents. The second factor is encoder network structure: to see how encoder structure affects alignments and pitch accents. Here, we chose CBHL and CNN encoder as elaborated and simple network structure,

respectively. The third factor is self-attention: to see how much self-attention helps to improve pitch accents and alignments by capturing long-term information.

Pipeline TTS systems are included in both experiments to be compared with end-to-end TTS. Pipeline TTS systems are interesting because they rely on full context labels which are beneficial to predict pitch accents. Performance of pipeline TTS systems is an indicator of how much end-to-end TTS can perform without complex labels.

4.4.1 Japanese speech corpus

We used the ATR Ximera [110] as a Japanese speech corpus for the two experiments. This corpus contains 28,959 utterances or around 46.9 hours in total duration from a female speaker. We also used the same settings as in the previous research. For linguistic features, we used phonemes and accentual type labels from manually annotated labels [16]. We trimmed the beginning and ending silences from the utterances, after which the duration of the corpus was reduced to 33.5 hours. We used 27,999 utterances for training, 480 for validation, and 480 for testing. All 480 samples in the test set were used for an objective evaluation, and 142 samples in the test set were used for a subjective evaluation.

4.4.2 Experimental conditions (1)

Systems

For the first experiment, we built several TTS systems as listed in Table 4.1. The *JA-Tacotron* and *SA-Tacotron* with and without accentual-type labels were built to show whether the investigated architectures can learn lexical pitch accents in an unsupervised manner. We also built a *SA-Tacotron* that uses vocoder parameters instead of mel-spectrogram as the acoustic features. In addition, we included *JA-Tacotron* with forced alignment instead of predicted alignment to understand the accuracy of alignment modeling better. With forced alignment, alignments are calculated with teacher forcing, and target acoustic parameters are predicted with the alignments obtained by teacher forcing. Note that, in this setting, even though forced alignments are calculated with teacher forcing, acoustic parameter prediction itself does not use

teacher forcing.

For *JA-Tacotron* and *SA-Tacotron*, we allocated 32 dimensions for accentual-type embedding and 224 dimensions for phoneme embedding. For the models without accentual-type embedding, 256 dimensions were allocated to phoneme embedding. We set the reduction factor to be two for the models using mel-spectrogram as a target and three for the models using vocoder parameters. All the predicted frames of the acoustic features were fed back as the next input. At inference time, the inference was stopped on the basis of a binary stop flag as in [22]. The network was optimized with Adam optimizer [111]. We used exponential learning decay with an initial rate 0.0005 for the models using mel-spectrogram, and 0.002 for the models using vocoder parameters.

For pipeline TTS systems, we included two systems that use vocoder parameters or mel-spectrogram [16], [17], [74]. Unlike the architecture of our proposed systems, these pipeline systems used full context labels as linguistic features and needed to have duration prediction models. To test how the accuracy of duration prediction affects the naturalness of synthetic speech, we compared phone duration predicted by a hidden semi-Markov model (HSMM) with oracle alignments obtained by forced alignments. Finally, as a reference for how much listeners are sensitive to incorrect lexical pitch accents, a baseline with slightly corrupted accentual labels was also included.[4]

Two types of WaveNet models were trained for the experiment, one taking the mel-spectrograms as the input and the other using the MGC and F_0 (vocoder parameters). These two WaveNets had the same network structure as that in our previous study [74].

Subjective evaluation

We recruited 236 native Japanese speakers as listeners by crowdsourcing. The listeners evaluated 32 samples from 16 systems in a single test set. This includes natural speech and analysis by synthesis (copy synthesis). One listener can evaluate at most 10 test sets. One sample was evaluated 20 times and we got 45,440 data points in total. Statistical significance was analyzed using the two-sided Mann-Whitney statistical test.

[4]This system is named MOC in [16].

Table 4.1: TTS systems used for our analysis. Notations are V: vocoder parameters, M: mel spectrogram, A: accentual type label, N: no accentual type label, P: predicted alignment, F: forced alignment.

System	Architecture	Acoustic feature	Accent label	Alignment
SATVAP	*SA-Tacotron*	MGC & F_0	✓	predicted
SATMAP		Mel-spec. 12.5 ms	✓	predicted
SATMNP		Mel-spec. 12.5 ms	N/A	predicted
TACMAP	*JA-Tacotron*	Mel-spec. 12.5 ms	✓	predicted
TACMAF			✓	force-aligned
TACMNP			N/A	predicted
TACMNF			N/A	force-aligned
PIPVAF	Pipeline [16, 74]	MGC & F_0	✓	force-aligned
PIPVAP		MGC & F_0	✓	predicted
PIPVCF		MGC & F_0	corrupted	force-aligned
PIPMAF		Mel-spec. 5 ms	✓	force-aligned
PIPMAP		Mel-spec. 5 ms	✓	predicted

Objective evaluation

We did some visual investigations for alignments and predicted mel-spectrogram. We visualized attention distributions from the *SA-Tacotron* system to investigate what kind of information the attention layers attend. The *SA-Tacotron* system gives two attention distributions from the dual source attention. The *SA-Tacotron* system has also two self-attention layers in encoder and decoder. We interpreted attention distributions from dual source attention and self-attention layers in decoder by visualization.

In addition, we conducted a statistical analysis for alignment scores of self-attention in a test set, because alignments of self-attention at encoder in the *SA-Tacotron* system are hard to interpret at sample level. We calculate mean alignment scores of phoneme pairs that frequently occur (more than 30 times).

We also compared mel-spectrograms from systems with and without accentual-type labels.

As metrics for pitch, we measured root mean square errors (RMSE), correlation, and unvoiced/voiced (U/V) errors of f_0. We measured the metrics of f_0 from *Ja-Tacotron*, *SA-Tacotron*, *SA-Tacotron* using vocoder parameters, the pipeline system using mel-

spectrogram, and the pipeline system using vocoder parameter. For systems using mel-spectrogram, we measured f_0 from their synthetic waveforms.

4.4.3 Experimental conditions (2)

Systems

We constructed Tacotron models with configuration combining three factors: 1) model parameter size (small or large); 2) encoder network structure (CBHL or CNN); 3) presence or absence of self-attention. The configurations of these eight models are also listed in the three left columns of Table 4.3.

For model parameter size configurations, we followed what we used in Section 3.2.1 in Chapter 3. To recap, the small parameter size is based on configurations used for the original experiment of Tacotron [1] and self-attention Tacotron [2]. For specific consideration for Japanese, we used 32-dimensional accentual type embedding besides the phoneme embedding with 224 dimensions, to incorporate accentual type as input to Tacotron. The encoder pre-net on the input phoneme embeddings has two layers: one with 224 dimensions and the other with 112 dimensions. The pre-net for accentual type input has 32 and 16 dimensions for its two layers. The CBHL encoder has 128 units, and its output from the bidirectional RNN has 256 dimensions. The decoder pre-net has 256 and 128 dimensions for each layer. The attention and decoder RNNs have 256 dimensions each. Location features in forward attention have 5 dimensions for filter and 10 dimensions for kernel. For the self-attention layer in self-attention Tacotron, we used 32 and 256 dimensions for the encoder and the decoder, respectively.

For the large parameter size, we followed configurations of Tacotron2 [22]. We used 64 and 448 dimensions for accentual type embedding and phoneme embedding, respectively. The encoder pre-net on phoneme increases the sizes of two layers to 448 and 224 dimensions, respectively. The layer sizes in pre-net on accentual type are increased to 64 and 32 dimensions. The CBHL encoder has 256 units, and its output from the bidirectional RNN is 512 dimensions. The decoder pre-net has 256 dimensions for each layer. The attention and decoder RNN have 128 and 1024 dimensions, respectively. For the self-attention layer in self-attention Tacotron, we used 64 and 1024 dimensions for the encoder and decoder, respectively. These parameter configurations are also shown in Figure 4.5-A for baseline Tacotron and Figure 4.5-B

for self-attention Tacotron.

For models using the CNN-based encoder, we set the hidden layers of the encoder to 256 and 512 dimensions for the small and large parameter sizes, respectively.

For all the eight experimental models, we further added CNN-based post-net of Tacotron2 for a fair comparison. Examples of post-net for large baseline and self-attention Tacotron are plotted in Figure 4.5.

We used L1 loss for the mel-spectrogram, and binary cross entropy loss for the stop flag. The models were optimized with an Adam optimizer [111], with exponential learning rate decay from 10^{-4} to 10^{-5}. We applied L2 regularization with a weight of 10^{-6} for baseline Tacotron and 10^{-7} for self-attention Tacotron with large parameter size. To avoid increasing training time, we used a reduction factor of two, the same value as in the previous work [2], instead of the reduction factor of one used in Tacotron2 [22]. We did not enable dropout during inference unlike Tacotron2 [22].

For waveform synthesis, we used the same μ-law WaveNet [12] model for both of the experiments, which was trained with ground truth mel-spectrograms with the same condition as acoustic features used by our Tacotron systems: 50 ms frame length and 12.5 ms frame shift.

Subjective evaluation

We conducted a listening test about naturalness of synthetic speech [5]. We recruited 213 native Japanese listeners via crowdsourcing. Listeners were asked to evaluate 28 samples in one set, using a five-grade mean opinion score (MOS) metric. One listener could evaluate at most ten sets. We collected 19,880 data points in total. We checked the statistical significance of the scores between systems with a Mann-Whitney rank test [104].

The listening test contained 14 systems including natural samples, three analysis-by-synthesis systems[6], two pipeline TTS systems, and the eight Tacotron systems. The three analysis-by-synthesis systems were WaveNet models used for the DNN based SPSS system using the vocoder parameter, for the vocoder-free TTS system using

[5] Audio samples for our experiments can be found at https://nii-yamagishilab.github.io/yasuda-csl-tacotron-audio-samples/.

[6] Analysis-by-synthesis systems are the reference systems using neural vocoders given natural acoustic features extracted from the test set utterances for waveform generation. They simulate TTS systems that can perfectly convert input text into acoustic features.

the mel-spectrogram with 5 ms frame shift, and for the Tacotron systems using the mel-spectrogram with 12.5 ms frame shift. The two pipeline systems were the DNN based SPSS and vocoder-free TTS systems using an autoregressive acoustic model as described earlier. These pipeline systems used manually annotated full-context labels [16], of which the subsets are the phoneme and accentual type labels used for all the Tacotron systems. With these pipeline systems, the experiment is expected to answer Q4 posed at the beginning of this section.

Since we trained the eight Tacotron models multiple times during objective evaluation, we selected the instance with the lowest alignment error rate and used the corresponding generated utterances for the listening test.

Objective evaluation

We measured the alignment error rate on the eight experimental Japanese Tacotron systems. There are three types of fatal alignment errors: discontinuous alignment, incomplete alignment, and overestimated duration, as described in Section 3.4.1 in Chapter 3.

We measured alignment error rates on each system several times. Each time, we initialized the system with a different random seed and trained it from scratch. After generating the 480 test set utterances, we counted the alignment errors using in-house software.

4.4.4 Experimental results (1)

Objective evaluation

Figure 4.6 shows a visualization of the attention layers of *SA-Tacotron* learned on the Japanese corpus. The first figure from the top shows the alignment of an encoder LSTM source and mel-spectrogram target for dual source attention. We can clearly see a sharp monotonic alignment formed by the forward attention. The second figure from the top shows the alignment of an encoder self-attention source and mel-spectrogram target. It seems to be related to accentual phrase segments and phrase breaks divided by pauses.

In statistical analysis of self-attention at encoder, we found some alignments based

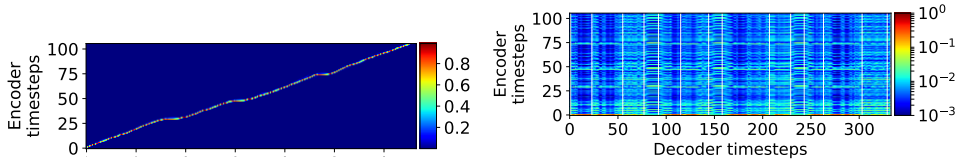


Figure 4.6: Alignment obtained by dual source attention in SATMAP. Left figure shows alignment between output of encoder's LSTM layer and target mel-spectrogram (forward attention). Right figure shows alignment between output of encoder's self-attention block and target mel-spectrogram (additive attention). Vertical white lines indicate accentual phrase boundaries obtained by forward attention.

on similarity in head 1. For example, top three phonemes with high score values are identical phoneme pairs and there are 11 identical phoneme pairs within the top 100. In addition, we found 13 phoneme pairs that belong to the same group (e.g. long vowels) within the top 100. The head 2 showed strong affinity to silence and pauses; we found 88 pairs that include pause or silence out of top the 100 pairs with high alignment scores.

Figure. 4.7 shows alignments of two heads from decoder self-attention in *SA-Tacotron*. Note that the upper triangular part above diagonal cannot be attended because it is the future information. Although both heads of the self-attention layer attend mostly the first frame which is silence with no useful information, they weakly attend a broad range of past frames. Both heads attend all past pauses once pauses are encountered.

Figure 4.8 shows predicted mel-spectrograms from *SA-Tacotron* with and without accentual-type labels. Accentual phrase boundaries predicted by the attention mechanism are also shown in the figure. From this figure, through comparison with a natural spectrogram, we see that the predicted spectrogram from *SA-Tacotron* without labels has wrong accentual positions and harmonics, whereas that from *SA-Tacotron* with labels does not. From informal listening, we also noticed that *SA-Tacotron* without labels had incorrect accent nucleus positions.

The alignment between source phoneme and target spectrogram frames should monotonically increase. Non-monotonic alignment may result in mispronunciation, some phonemes being skipped, repetition, the same phoneme continuing, and intermediate termination. We therefore manually counted abnormal alignment errors included in the test set. We observed no alignment errors for *JA-Tacotron* and *SA-Tacotron* using mel-spectrograms as a target. However, alignment errors were found for *SA-Tacotron*

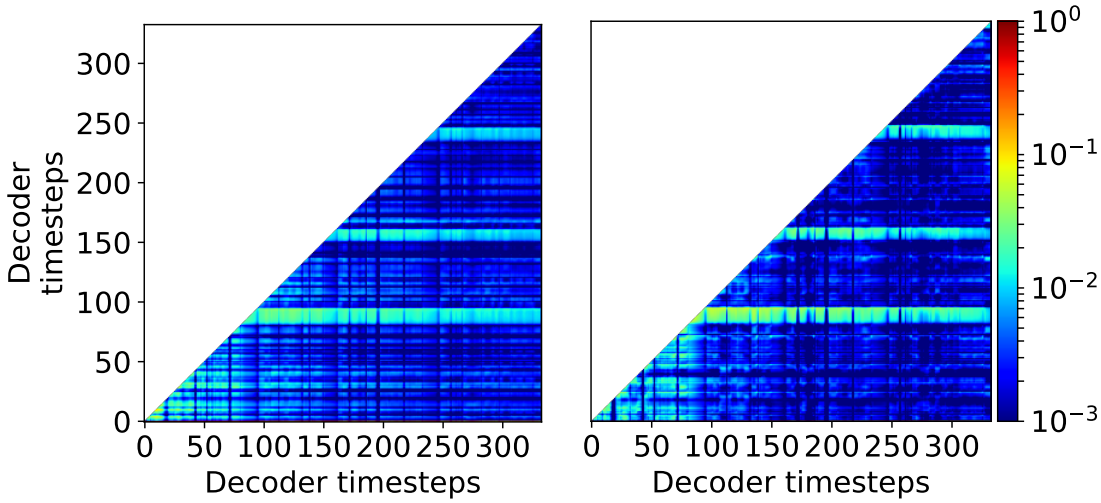


Figure 4.7: Alignment visualization of two heads from self-attention layer at decoder in *SA-Tacotron*. The vertical and horizontal axis is time steps of outputs from decoder RNN. Three horizontal bands with relatively high activation correspond to pause positions.

using vocoder parameters due to the length being longer than the corresponding mel-spectrogram. We found 19 alignment errors out of 142 test utterances.

Furthermore, we conducted an objective evaluation for four variant systems by using vocoder parameters such as JA-Tacotron and SA-Tacotron with and without accentual-type labels. To evaluate the f_0 prediction capability of JA-Tacotron and SA-Tacotron, we evaluated the objective metrics of f_0. To calculate the metrics we adjusted the frames between the predicted and ground truth f_0 using forced alignment. JA-Tacotron using vocoder parameters without labels failed to learn alignments between the source and target, so we did not include it.

Table 4.2: Objective evaluation of F_0 predicted by JA-Tacotron and SA-Tacotron.

System	accent	RMSE	CORR	U/V
TACVAF	✓	31.81	0.88	7.10 %
SATVAF	✓	31.77	0.88	7.20 %
SATVNF	N/A	39.30	0.79	7.04 %
PIPVAF	✓	23.31	0.94	3.25 %
PIPVCF	corrupted	31.09	0.89	3.29 %

Table 4.2 shows RMSE, correlation and U/V errors of F_0. Both JA-Tacotron and

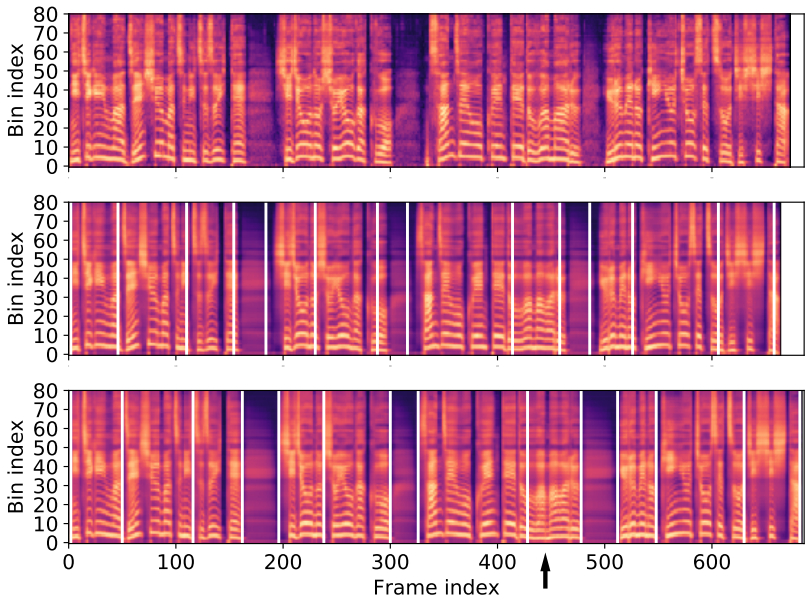


Figure 4.8: Natural mel-spectrogram (top figure), mel-spectrogram predicted from *SA-Tacotron* with accentual-type labels (middle figure), and mel-spectrogram predicted from *SA-Tacotron* without labels (bottom figure). Black arrow in the bottom figure points wrong harmonics that result in wrong accent. White lines show accentual phrase boundaries acquired from attention's output.

SA-Tacotron with accentual-type labels had an F_0 correlation value of 0.88. This indicates that self-attention had no effect on F_0 prediction accuracy. The systems without labels show lower correlation of 0.79 compared to the systems with labels, because of wrong accents as illustrated in Figure 4.8. Although these values were still lower than a baseline pipeline system that had 0.94 correlation [16], we think they are good enough considering that a frame shift of 5 ms is not the best condition for Tacotron. In addition, forced alignment itself has a negative effect on audio quality in Tacotron as we described in Section 4.4.2. The baseline system with noisy accentual-type labels had a correlation of 0.89. The noisy baseline had artificial accent errors with a probability of 50 %. Even though this is almost the same as the correlation values of our proposed systems with accentual-type labels, we do not think our proposed systems has accent errors with a probability of 50 %. As can be seen in the listening test result in Section 4.4.2, our proposed systems using mel-spectrogram with labels outperform the noisy baseline, so the relatively low correlation of F_0 was caused

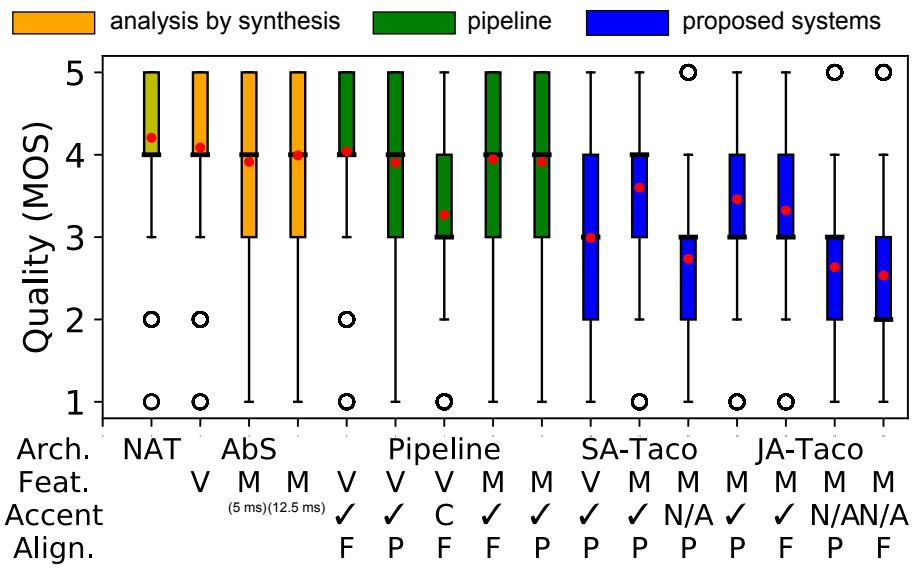


Figure 4.9: Box plots of MOS scores of each system regarding naturalness of synthetic speech. Red circles represent average values. NAT indicates natural speech. Refer to Table 4.1 for notations.

by the unsuitable conditions of acoustic features for Tacotron.

Subjective evaluation

Figure 4.9 shows five-point mean opinion scores of the proposed and baseline systems for the listening test results. All proposed systems without accentual-type labels got significantly lower scores than the corresponding systems with labels; for example, JA-Tacotron without labels had a score of 2.63 ± 0.03 whereas *JA-Tacotron* with labels got 3.46 ± 0.03. This means that the architectures of the proposed systems cannot learn lexical pitch accents in an unsupervised fashion and require additional inputs. The pipeline system with corrupted labels also showed a significant drop with a score of 3.27 ± 0.03. This shows that incorrect accents affected listener's judgments towards the naturalness of the synthetic speech.

SA-Tacotron had better scores than *JA-Tacotron* for each condition with or without accentual-type labels. This indicates that self-attention layers have a positive effect on the naturalness. Among our proposed systems, *SA-Tacotron* with labels (SATMAP) got the highest score of 3.60 ± 0.03.

SA-Tacotron using vocoder parameters got a relatively low score, 2.99 ± 0.03, even if it used accentual-type labels and self-attention layers. This is because this system

generated alignment errors due to the prediction of longer sequences as we described in the previous section. Among the baseline systems, the systems using MGC and F_0 had higher scores than the systems using mel-spectrogram under both the forced and predicted alignment conditions.

Interestingly, *JA-Tacotron* using forced alignment got lower scores than that using predicted alignment under both conditions with and without accentual-type labels. This result is surprising because, in traditional pipelines, forced alignment is used as an oracle alignment and normally leads to better perceptual quality than that of the predicted case. Since Tacotron learns both spectrograms and alignments simultaneously, it seems to produce the best spectrograms when it infers both of them. Among the baseline pipeline systems, as expected, a forced alignment gave higher scores than predicted alignment for both systems using vocoder parameters and mel-spectrogram. In the case of predicted alignment, the score has a long tail variance towards the low score region.

The best proposed system still does not match the quality of the best pipeline system. *SA-Tacotron* with accentual-type labels and the pipeline system using mel-spectrogram and predicted alignment had 3.60 ± 0.03 and 3.90 ± 0.03, respectively. These are not the same results as for the English experiments reported in [22]. One major difference of our proposed systems from pipeline systems other than architecture is input linguistic features; our proposed systems use phoneme and accentual-type labels only, but the baseline pipeline systems use various linguistic labels including word-level information such as inflected forms, conjugation types, and part-of-speech tags. In particular, an investigation on the same Japanese corpus found that the conjugation type of the next word is quite useful for F_0 prediction [112].

4.4.5 Experimental results (2)

Objective evaluation

Table 4.3 shows the alignment error rates of each system. From this table, we can first see that all systems with the large parameter size showed low alignment error rates and were stable over different runs. Increasing parameter size improved and stabilized the learning of alignments. Second, systems with self-attention and small parameter size made more alignment errors and were quite unstable across different

Table 4.3: Alignment error rate on Japanese test set. Each system was evaluated three times, and each time it was trained from scratch with a different initial random seed. Ave. indicates average error rate. Values in bold correspond to those evaluated in a listening test.

Para. size	Encoder	Self-attention	# Para. (1×10^6)	Alignment error rate (%) Ave.	1	2	3
Small	CBHL	-	11.3	2.4	**0.4**	2.7	4.2
		✓	11.6	11.5	**6.0**	7.9	20.6
	CNN	-	9.2	2.4	**0.6**	1.7	5.0
		✓	9.6	18.2	**4.8**	14.2	35.6
Large	CBHL	-	35.8	0.2	**0.2**	0.2	0.2
		✓	41.6	0.3	**0.2**	0.2	0.4
	CNN	-	27.2	0.2	**0.2**	0.2	0.2
		✓	32.8	0.2	**0.2**	0.2	0.2

runs. Through further investigation, we found that a combination of post-net and self-attention made the alignments unstable in the small-parameter-size configurations. In general, when systems have small parameter sizes, their learning of the alignment is sensitive to the initial parameters and network structures.

Subjective evaluation

Figure 4.10 shows the results of the listening test in Japanese, and Table 4.4 lists the outcome from the statistical significance test. We can see a significant gap between Tacotron systems with small and large parameter sizes. All systems with the small parameter size had low scores: baseline Tacotron had scores of 3.04 ± 0.04 and 2.55 ± 0.04 for the CBHL and CNN encoders, respectively, and self-attention Tacotron systems had 2.85 ± 0.04 and 2.23 ± 0.04 for the CBHL and CNN encoders, respectively. For both baseline Tacotron and self-attention Tacotron, the CBHL encoder performed better than the CNN encoder in the small parameter size conditions. On the other hand, Tacotron systems with the large parameter size had high MOSs of about 4.0. We listened to the samples that had average MOSs of less than 2.5 from the small-parameter-size systems and found incorrect pitch accents in them. Furthermore, samples from systems using the CNN encoder with the small parameter size generally sounded flatter in pitch than

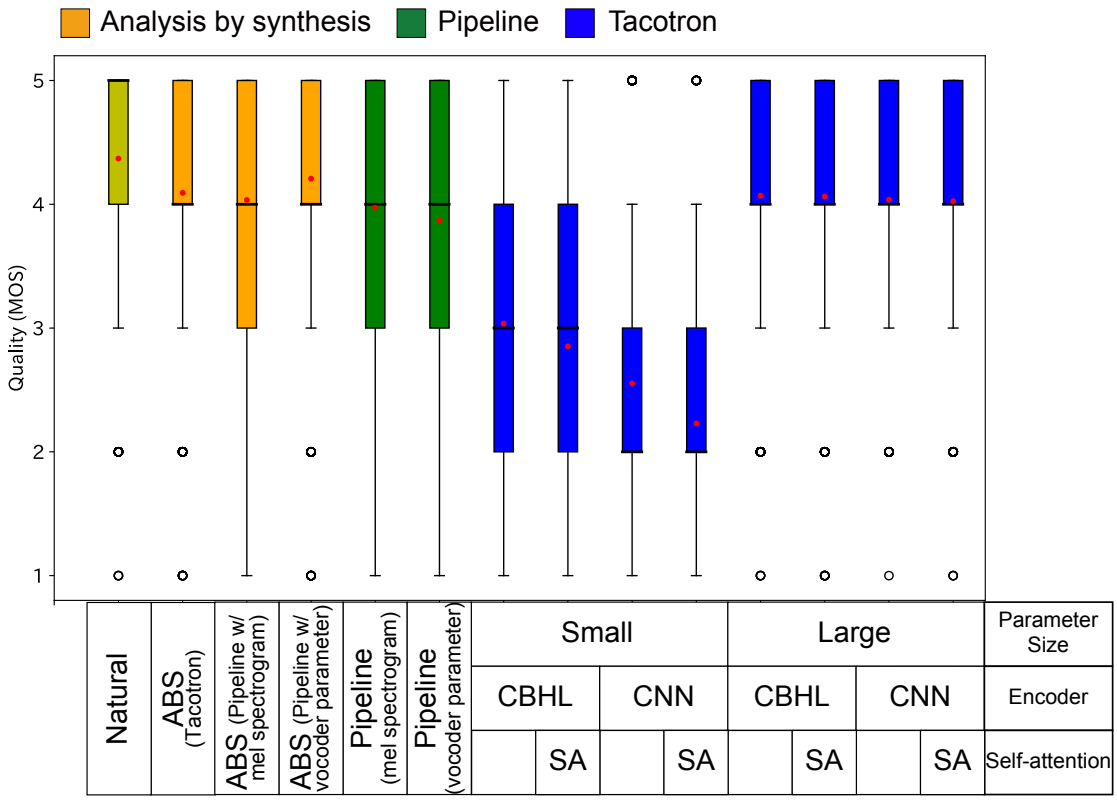


Figure 4.10: Box plot for a results of Japanese listening test. Red dots indicate mean, and black bars indicate median.

corresponding systems using the CBHL encoder, which is probably why listeners rated them negatively. Some lowly rated samples from the small self-attention Tacotron also contained fatal alignment errors, which were identified by the alignment error detector.

The difference among Tacotron systems with the large parameter size is not statistically significant as Table 4.4 shows, so the presence of self-attention and difference in encoder network structure did not significantly affect the naturalness of synthetic speech. Furthermore, these Tacotron systems had slightly higher scores than the pipeline system using the mel-spectrogram, which had an MOS of 3.97 ± 0.04. The Tacotron systems using the CBHL encoder had statistically significant differences to the pipeline system using mel-spectrogram, but the Tacotron systems using the CNN based encoder did not. The pipeline system using the vocoder parameter had a lower MOS (3.87 ± 0.04) than the pipeline system using the mel-spectrogram and the Tacotron system with the large parameter size.

Table 4.4: Mann-Whitney rank test for Japanese listening test. Cell in this color denotes statistical significance ($p \leq 0.01$) while this color denotes $p > 0.01$.

				Natural	ABS			Pipeline		Tacotron								
										Small				Large				Para. size
										CBHL		CNN		CBHL		CNN		Encoder
											✓		✓		✓		✓	Self-att.
Natural					Tacotron Mel.	Pipeline Mel.	Vocoder Para.	Pipeline Mel.	Vocoder Para.									
ABS	Tacotron Mel. spec.																	
	Pipeline Mel. spec.																	
	Vocoder para.																	
Pipline	Pipeline Mel. spec.																	
	Vocoder para.																	
Tacotron	Small	CBHL	-															
			✓															
		CNN	-															
			✓															
	Large	CBHL	-															
			✓															
		CNN	-															
			✓															

4.4.6 Conclusion

The first experiment showed: 1) Accent labels helped to produce correct pitch accents, while removing pitch accent labels degraded pitch accents significantly. 2) Self-attention improved naturalness of synthetic speech. 3) Predicted alignments were stable enough, while forced alignment slightly degraded naturalness of synthetic speech. 4) Mel-spectrogram was a suitable choice for acoustic features, while vocoder parameters were challenging for the end-to-end TTS model to predict. Finally, the Tacotron systems did not match the pipeline systems in terms of the quality of synthetic speech.

The second experiment showed: 1) Increasing parameter size greatly improved naturalness. 2) The CBHL encoder performed better than the CNN encoder for systems with the small parameter size. The difference in encoder structure however did not make much difference for systems with the large parameter size. 3) Self-attention did not improve naturalness for models with the large parameter size. Self-attention was expected to improve naturalness for models with the small parameter size in

accordance with the first experiment, but any improvement was undercut by alignment errors caused by poor training in this experiment. The Tacotron system with the large parameter size using the CBHL encoder slightly outperformed the pipeline system using the same mel-spectrogram and the same neural vocoder.

When a model has sufficient parameter capacity, we confirmed that the Tacotron systems can learn to compensate for the lack of other rich information and accurately predict mel-spectrogram without additional input contextual labels such as part-of-speech tags and syntactic information that are used in the pipeline systems. The score of the best Tacotron system (large parameter size and CBHL encoder) in our experiment was higher than that of the pipeline system, but, as expected, the difference is smaller than that reported in the literature since both systems use autoregressive modeling and the same neural vocoder.

There are many differences between Tacotron and Tacotron2 apart from the use of neural vocoders as we described earlier. Our result, however, reveals that such changes are not fundamental differences and do not bring about significant improvements, at least, for our Japanese TTS systems using phonemes. Our analysis results indicate that the major improvements simply come from the model capacity.

However, we also observed that the linguistic feature limitation affected accuracy of pitch accent prediction for a few systems. More specifically, we found incorrect pitch accent from audio samples were generated using the systems with the small parameter size. For pitch accent information, we fed accentual type labels, which indicate the accentual phrase boundary and type of pitch accent but do not explicitly indicate where to drop pitch. Therefore, we think that the systems with the small parameter size were occasionally unable to resolve the position to drop pitch given accentual type labels alone. This is consistent with results reported in [33], where an abstract pitch contour pattern is used such as high and low labels plus an accentual phrase boundary.

5 Modeling pitch accents in end-to-end TTS

5.1 From learning to modeling pitch accents

In Chapter 4, we confirmed that end-to-end TTS models could produce correct pitch accents when we fed pitch accent labels. It means learning pitch accents from labels is possible for end-to-end TTS framework. In this chapter, we aim to predict pitch accents along with acoustic features without relying on pitch accent labels during prediction to reduce dependency on linguistic features.

Figure 5.1 shows schematic comparison of the two approaches for pitch accents. The left diagram in Figure 5.1 shows the approach to learning pitch accents we investigated in Chapter 4. In this case, an end-to-end TTS model generates mel-spectrogram physically representing pitch accents given pitch accent and phoneme labels. The right diagram in Figure 5.1 shows an approach we take in this chapter. In this case, an end-to-end TTS model predicts pitch accent labels from phonemes, and then generates

mel-spectrogram based on the predicted pitch accents. To do so, the end-to-end TTS model has to model pitch accents as a latent variable.

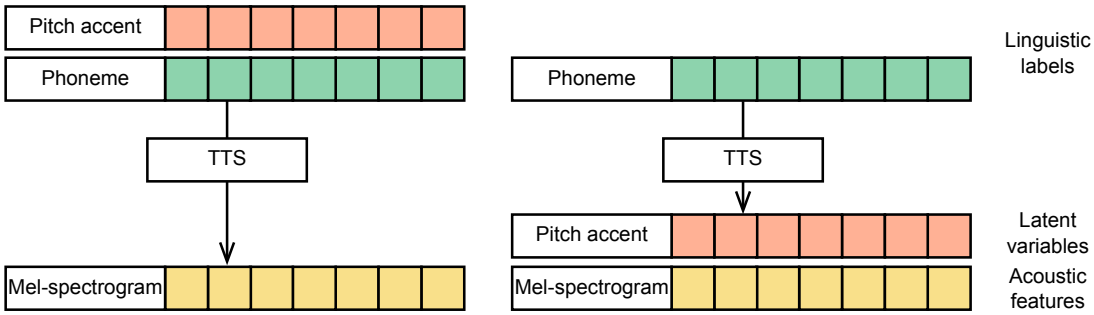


Figure 5.1: Schematic comparison of learning and modeling pitch accents.

5.2 Pitch accent modeling

Pitch accent modeling has been investigated in two forms: 1) abstract pitch contour modeling [113, 114]; 2) f_0 modeling [115, 43, 116]. We touched f_0 modeling in end-to-end TTS and DNN-based SPSS in Chapter 4. We work on the former, abstract pitch contour modeling in this chapter.

End-to-end TTS does not model pitch accent explicitly. It is ideal that end-to-end TTS learns and generates speech representing natural pitch accent without modeling it explicitly, but it is difficult, especially for pitch accent languages. As shown in Chapter 4, Tacotron-based end-to-end TTS systems can not produce correct pitch accents for Japanese, which is a pitch accent language, when labels are not given.

One way to model pitch accents in an end-to-end TTS framework is using latent variables to represent pitch accent. This approach enables prediction of pitch accents along with speech without depending on labels during test phase. To incorporate the pitch accent latent variable in end-to-end TTS, the following problems must be considered:

- Design of latent space;
- Framework to model and optimize latent variable;

- How to enforce latent variable to represent pitch accent.

5.2.1 Design of latent space

Pitch accents are perceived as pitch change. Pitch contour can thus be represented with f_0. f_0 is frame-level acoustic representation rather than linguistic-unit level representation. Abstract tone contour is linguistic-unit level representation of pitch accent which represents simplified pattern of pitch change for a linguistic unit. For example, tone of Mandarin can be represented with six pitch contour patterns: LL (low), HH (high), LH (rising), HL (falling), L (neutral low), and H (neutral high), and are compiled as C-ToBI format [117]. Japanese pitch accent in Tokyo dialect has more limited pattern, which consist of three pitch contour patterns: HL (falling), L (neutral low), and H (neutral high), and compiled as X-JToBI format [118].

Pitch accents are lexical accents. This means that pitch accents depends on linguistic units such as syllables. Pitch accent modeling is thus to predict pitch accent from linguistic features. Abstract pitch contour pattern is the natural choice of targets to model because it is based on linguistic units. It can be represented with discrete symbols. As a result, we design our latent space discrete.

5.2.2 Framework to model and optimize latent variable

There are several methods to model latent variables. Here we adopt variational autoencoder [119] for the following reasons:

- The latent variable for pitch accent is a sequence. It is daunting to marginalize all possible pitch accent sequences. Variational inference provides tractable optimization method by maximizing variational lower bound instead of marginal likelihood.

- Variational autoencoder has encoder to learn latent representations by encoding inputs into latent space. The inputs are acoustic features in our case. Pitch accent is inherent to acoustic information. Variational autoencoder is a promising method to obtain good pitch accent representation in latent space.

5.2.3 How to enforce latent variable to represent pitch accent

Latent variable may represent any features in speech. Latent information is anything aside from linguistic content which is given as input in TTS: it may represent speaking rate, intonation, accent, emotion, or loudness. In order to enforce latent variable to represent pitch accent, we utilize supervision by feeding pitch accent labels in abstract tone contour pattern format. Because we use discrete latent space to represent pitch accent in abstract pitch contour pattern, we can relate pitch accent labels to latent variable one-by-one.

5.3 Variational autoencoder

Variational autoencoder (VAE) is one of the deep latent variable models [119]. VAE is used extensively for unsupervised representation learning, which aims to learn statistically meaningful representations from data. VAE consists of encoder and decoder, which is a recognition and generative model, respectively. The encoder encodes inputs by modeling parameters to relate inputs to latent variables. VAE can be optimized as variational method-based autoencoder with evidence lower bound objective, which consists of loss for the decoder to reconstruct inputs from latent variables and loss to regularize latent variables.

The evidence lower bound objective can be derived as follows:

$$
\begin{aligned}
\log p(\mathbf{x}) &= \int q(\mathbf{z} \mid \mathbf{x}) \log p(\mathbf{x}) d\mathbf{z} && (5.1)\\
&= \int q(\mathbf{z} \mid \mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} && (5.2)\\
&= \int q(\mathbf{z} \mid \mathbf{x}) \log \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} && (5.3)\\
&= \int q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x}) p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{x}) p(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} && (5.4)\\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x} \mid \mathbf{z}) + \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} - \log \frac{p(\mathbf{z} \mid \mathbf{x})}{q(\mathbf{z} \mid \mathbf{x})}] && (5.5)\\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})] + \mathrm{KL}[p(\mathbf{z} \mid \mathbf{x}) \| q(\mathbf{z} \mid \mathbf{x})] && (5.6)\\
&\geq \mathbb{E}_{q_\lambda(\mathbf{z}|\mathbf{x})} [\underbrace{\log p_\theta(\mathbf{x} \mid \mathbf{z})}_{\text{Decoder}}] - \mathrm{KL}[\underbrace{q_\lambda(\mathbf{z} \mid \mathbf{x})}_{\text{Approximate posterior}} \| \underbrace{p_\phi(\mathbf{z})}_{\text{Prior}}]. && (5.7)
\end{aligned}
$$

The first term is reconstruction loss for decoder, and the second term is Kullback–Leibler divergence between approximate posterior and prior, where KL denotes Kullback–Leibler divergence. The expectation of the reconstruction loss is estimated with sampling. VAE uses a reparametrization trick to reduce noise induced by the sampling.

VAE has been used in TTS for speaking style and prosody modeling. Motivation of speaking style modeling ranges from diverse speech generation to speaking style control and prosody transfer. Early works of speaking style modeling were based on plain autoencoder [120] or style tokens [89, 121], and later works formulated based on VAE [122, 123, 124]. Normal VAE models global style for a whole utterance. In contrast, conditional VAE is used for local speaking style modeling by conditioning latent variable with linguistic features [125, 126, 127]. Probability distribution of latent variable is designed mainly as Gaussian [125, 123, 127, 124], but sometimes hierarchical latent space is adopted, for example, with combinations of categorical and mixture of Gaussian distribution [122], or Gaussian and indicator distribution [126].

5.4 VQ-VAE

VAE may use discrete distribution over latent variables. It is however not straightforward to use discrete distribution in VAE because the reparametrisation trick to propagate gradients from decoder to encoder is not possible for sampling discrete variables which stops gradient propagation. There are few methods enabling discrete latent variables for VAE [128, 129, 130].

Among the discrete VAEs, vector quantized autoencoder (VQ-VAE) [128] is a method based on vector quantization, and it has been applied to speech synthesis [131, 132, 133, 134, 135, 136, 137, 126, 138].

VQ-VAE defines latent embedding vectors $\mathbf{e}_i \in R^D, i \in \{1 \dots K\}$. An output of encoder $z_e(\mathbf{x})$ is quantized to the nearest neighbour of the embedding via $z_q(\mathbf{x}) = \mathbf{e}_k$ where $z_q(\mathbf{x})$ is a quantized representation and $k = \text{argmin}_j \|z_e(\mathbf{x}) - \mathbf{e}_j\|^2$. The quantized representation $z_q(\mathbf{x})$ is passed to decoder to reconstruct data $\mathbf{x}$.

For learning, VQ-VAE optimizes a reconstruction loss $-\log p(\mathbf{x}|z_q(\mathbf{x}))$ in evidence lower bound. To propagate a gradient from decoder to encoder, a gradient of the reconstruction loss with respect to the encoder output $z_e(\mathbf{x})$ is estimated with straight through estimator. The gradient with respect to the decoder input $z_q(\mathbf{x})$ is used as the gradient with respect to the encoder output $z_e(\mathbf{x})$.

VQ-VAE does not need to optimize KL divergence in evidence lower bound. VQ-VAE assumes prior distribution is uniform, so its KL divergence is $\log K - \mathrm{H}[q(z|\mathbf{x})]$ where H is entropy. Furthermore, it designs its posterior distribution as one-hot which has unity density at k, which means its entropy is zero. Therefore the KL divergence is constant, that is $\log K$.

In addition to the reconstruction loss, VQ-VAE use vector quantization objective by minimizing a square distance between encoder output $z_e(\mathbf{x})$ and the nearest embedding $\mathbf{e}_k$ to learn the latent representation. Besides, commitment loss is used to prevent the encoder output to grow arbitrarily. Overall, loss function of VQ-VAE is the following: $L = -\log p(\mathbf{x}|z_q(\mathbf{x})) + \|\text{sg}[z_e(\mathbf{x})] - \mathbf{e}_k\|^2 + \beta\|z_e(\mathbf{x}) - \text{sg}[\mathbf{e}_k]\|^2$, where the first term is reconstruction loss, the second term is the vector quantization loss, and the third term is the commitment loss with weight β.

Optionally, VQ-VAE can use prior distribution other than uniform, for example, autoregressive prior [128, 126]. In this case, a nonuniform prior is introduced at fitting

phase after training.

5.5 Pitch accent modeling with conditional VQ-VAE

Among variants of variational autoencoder, we formulate our TTS using pitch accent latent variable based on conditional VQ-VAE [139]. VQ-VAE is variational autoencoder that enables the use of a discrete latent variable. The conditional VQ-VAE is an extension of VQ-VAE to allow a nonuniform prior distribution. The conditional VQ-VAE is a suitable method to incorporate pitch accent latent variable to TTS because 1) we represent pitch accent as discrete symbols with abstract tone patterns, and 2) prior probability of pitch accent given linguistic feature is a nonuniform distribution and it is what we want to model.

We introduce discrete latent variables $\mathbf{z}^{(q)}_{1:U}$ to TTS model $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$ to represent pitch accents, where T and U denote length of acoustic features $\mathbf{x}_{1:T}$ and linguistic features $\mathbf{y}_{1:U}$, respectively. We assume a pattern of pitch accent or tone belongs to a linguistic unit such as phoneme or syllables. We therefore denote the latent accent variable sequence as $\mathbf{z}^{(q)}_{1:U}$, where $u = \{1, \cdots, U\}$ represents index of linguistic units in linguistic feature sequence $\mathbf{y}_{1:U}$.

Our aim is two fold: 1) we want to predict pitch accents from linguistic features $\mathbf{y}_{1:U}$ without depending on accent labels $\mathbf{l}_{1:U}$ during inference; 2) we want to utilize accent labels $\mathbf{l}_{1:U}$ during training to enforce latent variable to capture pitch accent information.

We follow a TTS framework based on conditional VQ-VAE [139] to formulate our proposed TTS method for latent accents. This framework is originally proposed for duration modeling using latent variable, not pitch accent, as we will cover it for duration modeling in Chapter 7. Figure 5.2 shows a brief overview of the idea to incorporate pitch accent model to end-to-end TTS based on VQ-VAE. The tone recognizer works as encoder by encoding speech into tone representation to optimize latent variables. The decoder reconstructs speech from tone latent variable given linguistic labels. The tone predictor predicts tones from linguistic features. The tone labels defines approximate posterior distribution of VQ-VAE to enforce latent space to represent tone information. We modify the framework of [139] to handle pitch accents in latent variable. Our modifications include, 1) we define approximate posterior

$Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$ with accent labels to enforce latent variable to capture accent information; 2) we use soft-attention to align acoustic features with linguistic features in acoustic encoder, and to align linguistic features and pitch accents with acoustic features in acoustic decoder.

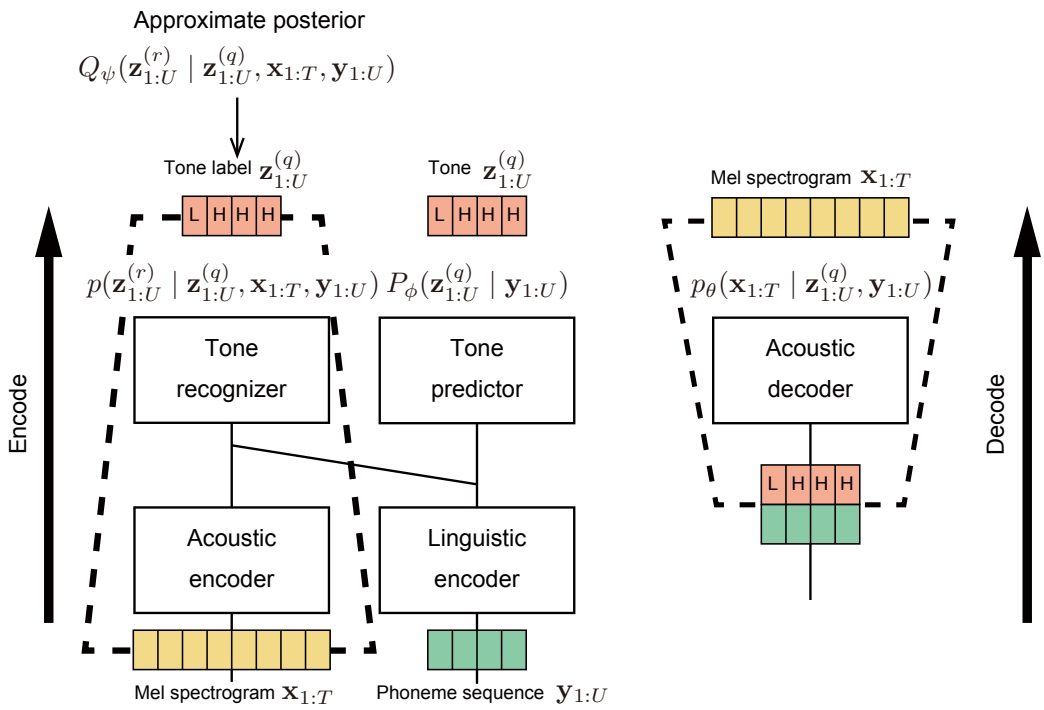


Figure 5.2: Brief overview of idea to incorporate pitch accent model to end-to-end TTS based on VQ-VAE.

5.5.1 Model definition and variational lower bound

Let us write the input sequence of discrete tokens as $\mathbf{y}_{1:U} = (y_1, \cdots y_U)$, where y_u is the u-th token (e.g., letter or phone). We then use $\mathbf{x}_{1:T} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ to denote an output sequence of acoustic features, where $\mathbf{x}_t \in \mathbb{R}^O$ is for the t-th output time step or *frame*. Our goal is to learn a good model $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$.

We use $\mathbf{l}_{1:U} = (l_1, \cdots, l_U)$ to denote the latent pitch accent for $\mathbf{y}_{1:U}$, where the value of l_u is equal to a kind of abstract pitch contour pattern of the u-th input token. Since l_u is discrete for TTS systems, we can parameterize $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$ with VQ-VAE. Let $\mathcal{Z} = \{\mathbf{e}_1, \cdots, \mathbf{e}_K\}$ be a codebook with K code words, where $\mathbf{e}_k \in \mathbb{R}^D$.

We then introduce a quantized latent vector $z_u^{(q)} \in \mathcal{Z}$ and an un-quantized latent vector $z_u^{(r)} \in \mathbb{R}^D$ for the u-th input token. Because $z_u^{(q)} \in \mathcal{Z}$, we can link $z_u^{(q)}$ with l_u by setting $z_u^{(q)} = \mathbf{e}_{l_u}, \forall u \in \{1, \cdots, U\}$, which avoids modeling $z_u^{(q)}$ and l_u independently. We additionally introduce $z_u^{(r)}$ because it allows us to derive the VQ-VAE training criteria in a theoretically sound manner [140].

Model training through direct optimization of $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:U})$ is daunting, but the above definition allows us to find a tractable variational lower bound (ELBO):

$$\log p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$$

$$= \log \sum_{\forall \mathbf{z}_{1:U}^{(q)}} \int_{\mathbf{z}_{1:U}^{(r)}} p(\mathbf{x}_{1:T}, \mathbf{z}_{1:U}^{(q)}, \mathbf{z}_{1:U}^{(r)} \mid \mathbf{y}_{1:U}) d\mathbf{z}_{1:U}^{(r)} \tag{5.8}$$

$$\geq \mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)}|\mathbf{x}_{1:T},\mathbf{y}_{1:U})} \underbrace{[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U})]}_{\text{Decoder}} \tag{5.9}$$

$$- \mathrm{KL}[Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| \underbrace{P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})}_{\text{Prior}}] \tag{5.10}$$

$$- \mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)})|\mathbf{x}_{1:T},\mathbf{y}_{1:U}} \left\{ \mathrm{KL}[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| \underbrace{p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{Vector quantization}}] \right\}. \tag{5.11}$$

$\mathbf{x}_{1:T}$ in the decoder is assumed to be independent of $\mathbf{z}_{1:U}^{(r)}$ given $\mathbf{z}_{1:U}^{(q)}$ and $\mathbf{y}_{1:U}$. Furthermore, $\mathbf{z}_{1:U}^{(r)}$ is assumed to depend only on $\mathbf{z}_{1:U}^{(q)}$. KL denotes the Kullback–Leibler divergence.

5.5.2 Model parameterization

Each term in the ELBO of Equations (5.9–5.11) can be parameterized using neural networks and calculated in a closed form.

Approximate posteriors

We first define the approximate posterior for $\mathbf{z}_{1:U}^{(q)}$ as

$$
\begin{aligned}
&Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) \\
&= \prod_{u=1}^{U} P(\mathbf{z}_u^{(q)} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{I}(\mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}),
\end{aligned} \tag{5.12}
$$

where $\mathcal{I}(\cdot)$ is an indicator function and l_u is the code word index for the u-th input token. We then define

$$
\begin{aligned}
&Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) \\
&= \prod_{u=1}^{U} p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}),
\end{aligned} \tag{5.13}
$$

where $\mathcal{N}$ is a multivariate Gaussian, σ^2 is a hyper-parameter, and $\mathbf{d}_u$ is computed by a neural network as $\mathbf{d}_u = \text{LatentNet}_\psi(\mathbf{z}_{u-1}^{(q)}, \bar{\mathbf{x}}_u, y_u)$.

Since $\mathbf{x}_{1:T}$ has T frames while $\mathbf{y}_{1:U}$ has U tokens, we aggregate $\mathbf{x}_{1:T}$ as $\bar{\mathbf{x}}_{1:U} = (\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_U)$ before feeding it to $\text{LatentNet}_\psi(\cdot)$. The aggregation is conducted by soft-attention, which averages $\mathbf{x}_t$ with weights indicating how likely acoustic features $\mathbf{x}_t$ correspond to the u-th token of linguistic feature $\mathbf{l}_{1:u}$.

We let Q_λ be an indicator function, following the idea of VQ-VAE. The value of l_u during training is given by an accent label. A Gaussian posterior Q_ψ is inspired by the original VAE [119]. It is used to compute the KL divergence in Eq. (5.11), which is explained in Section 5.5.2.

Decoder

Similar to other models, our model uses an auto-regressive decoder. Given $\mathbf{y}_{1:U}$, $\mathbf{l}_{1:U} = (l_1, \cdots, l_U)$, and $\mathbf{z}_{1:U}^{(q)} = (\mathbf{z}_1^{(q)} = \mathbf{e}_{l_1}, \cdots, \mathbf{z}_U^{(q)} = \mathbf{e}_{l_U})$, the PDF of $\mathbf{x}_{1:T}$ is defined as

$$
\begin{aligned}
&p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) \\
&= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t) = \prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_t; \mu_t, \sigma_d^2\mathbf{I}),
\end{aligned} \tag{5.14}
$$

where $\mu_t = \text{Decoder}_\theta(\mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t)$. Note that $\widehat{\mathbf{y}}_{1:T} = \{\widehat{y}_1, \cdots, \widehat{y}_T\}$ and $\widehat{\mathbf{z}}_{1:T}^{(q)} = \{\widehat{\mathbf{z}}_1^{(q)}, \cdots, \widehat{\mathbf{z}}_T^{(q)}\}$ are upsampled from $\mathbf{y}_{1:U}$ and $\mathbf{z}_{1:U}^{(q)}$ with soft-attention by averaging $\mathbf{y}_{1:U}$ and $\mathbf{z}_{1:U}^{(q)}$ with weights indicating how likely it is that they correspond to a frame of acoustic features $\mathbf{x}_t$ at each step during decoding.

Prior

The prior for $\mathbf{z}_{1:U}^{(q)}$ is defined as

$$P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U}) = \prod_{u=1}^{U} P(\mathbf{z}_u^{(q)} \mid \mathbf{z}_{u-1}^{(q)}, y_u), \tag{5.15}$$

where the probability of observing $\mathbf{z}_u^{(q)} = \mathbf{e}_l$ is computed as

$$P(\mathbf{z}_u^{(q)} = \mathbf{e}_l \mid \mathbf{z}_{u-1}^{(q)}, y_u) = \frac{\exp(-\|\mathbf{c}_u - \mathbf{e}_l\|_2^2)}{\sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2)}. \tag{5.16}$$

The activation vector $\mathbf{c}_u$ is computed through a neural network as $\mathbf{c}_u = \text{LatentNet}_\phi(\mathbf{z}_{u-1}^{(q)}, y_u)$. Note that the prior calculates the probability of selecting l-th codeword $\mathbf{e}_l$ for each input token y_u.

Given Eqs. (5.15) and Eq. (5.12), we compute Eq. (5.10) in the ELBO as

$$\begin{aligned}
&\text{KL}[Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})] \\
&= -\log P(\mathbf{z}_{1:U}^{(q)} = (\mathbf{e}_{l_1}, \cdots, \mathbf{e}_{l_U}) \mid \mathbf{y}_{1:U}) && (5.17) \\
&= \sum_{u=1}^{U} \left\{ \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2) \right\}. && (5.18)
\end{aligned}$$

Since Q_λ is an indicator function, the value of the KL divergence is equal to the prior models' likelihood of the codeword $\{\mathbf{e}_{l_1}, \cdots, \mathbf{e}_{l_U}\}$ produced by the approximate posterior Q_λ.

Vector quantization

For the vector quantization term in Eq. (5.11), we factorize it as

$$p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) = \prod_{u=1}^{U} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)}, \sigma^2\mathbf{I}), \tag{5.19}$$

and we assume that $\mathbf{z}_u^{(r)} = \mathbf{z}_u^{(q)} + \eta$, where $\eta \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. In other words, the quantization loss is Gaussian-distributed. The hyper-parameter σ is the same as that in Section 5.5.2.

Given Eqs. (5.19) and Eq. (5.13), we compute Eq. (5.11) in the ELBO as

$$\begin{aligned} &\mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)})}\left\{\mathrm{KL}\left[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U})\right]\right\} \\ &= \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2. \end{aligned} \tag{5.20}$$

Note that Q_λ is an indicator function, and the KL divergence between two Gaussian distributions has a closed form.

5.5.3 Attention regularization

The acoustic inputs $\mathbf{x}_{1:T}$ and linguistic inputs $\mathbf{y}_{1:U}$ have different lengths. To adjust their length, the tone recognizer performs aggregation of the acoustic inputs as $\bar{\mathbf{x}}_{1:U}$ and the decoder performs upsampling of the linguistic inputs as $\hat{\mathbf{y}}_{1:T}$. The aggregation is based on soft attention as follows.

$$\bar{\mathbf{x}}_{1:U} = \mathrm{Aggregate}(\mathbf{x}_{1:T}, \mathbf{y}_{1:U}) = \left\{ \bar{\mathbf{x}}_u \middle| \bar{\mathbf{x}}_u = \sum_{t=1}^{T} p(z_u = t \mid \mathbf{y}_{u-1}, \mathbf{x}_t)\mathbf{x}_t \right\}, \tag{5.21}$$

$$\hat{\mathbf{y}}_{1:T} = \mathrm{Upsample}(\mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \left\{ \hat{\mathbf{y}}_t \middle| \hat{\mathbf{y}}_t = \sum_{u=1}^{U} p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\mathbf{y}_u \right\} \tag{5.22}$$

In order to force the tone recognizer attention to attend at the acoustic frame where decoder currently decodes, we introduce a regularization term $\mathrm{KL}[p(z_t \mid$

$\mathbf{x}_{t-1}, \mathbf{y}_u)\|\text{Normalize}\,(p(z_u \mid \mathbf{y}_{u-1}, \mathbf{x}_t))]$ to match the two attention distributions at encoder and decoder.

5.5.4 Training criteria in summary

With all the components explained in Section 5.5.2, we summarize the training criterion of the proposed model as

$$
\begin{aligned}
&\mathcal{L}(\theta, \psi, \phi, \lambda, \mathcal{Z}) \\
=&\mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U})}[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})] && (5.23)\\
&- \sum_{u=1}^{U} \left\{ \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2) \right\} && (5.24)\\
&- \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2 && (5.25)\\
&+ \gamma \mathrm{KL}[p(z_t \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\|\text{Normalize}\,(p(z_u \mid \mathbf{y}_{u-1}, \mathbf{x}_t))], && (5.26)
\end{aligned}
$$

Equations (5.23 - 5.25) correspond to the ELBO in Section 5.5.1, and Equation 5.25 is attention regularization term described in 5.5.3. The vectors $\mathbf{d}_u$ and $\mathbf{c}_u$ are computed as $\mathbf{d}_u = \text{LatentNet}_\psi(\mathbf{z}^{(q)}_{u-1}, \bar{\mathbf{x}}_u, y_u)$ and $\mathbf{c}_u = \text{LatentNet}_\phi(\mathbf{z}^{(q)}_{u-1}, y_u)$.

The decoder parameter θ is trained with Equation (5.23); the LatentNet_ϕ in prior is trained Equation (5.24); the LatentNet_ψ in approximate posterior is trained with Equation (5.25). The codebook $\mathcal{Z} = \{\mathbf{e}_1, \cdots, \mathbf{e}_K\}$ is trained with Equation (5.24 -5.25). All the components are jointly trained. During inference, only the prior and decoder network are used.

5.6 Experiments

In the experiment, we tested how much our accent latent model can predicts accent information from linguistic features. We targeted Japanese pitch accent, which has pitch accent in a unit of a word or group of words called accentual phrase. Because TTS directly from the Japanese writing system is not feasible due to diversity of its logographic characters called kanji [2], we used phoneme sequence as linguistic

feature.

Architecture

We constructed our proposed method based on conditional VQ-VAE [139]. The TTS framework proposed in [139] treats phoneme duration as a latent variable by using conditional VQ-VAE. We modify the framework to use pitch accents instead of phoneme duration. Alignment is modeled with soft-attention by following fine-grained prosody transfer [125].

Figure 5.3 shows a network architecture of the proposed system. We used a CBHG encoder [1] for the network structure of the linguistic encoder, and a LSTM decoder [22] with forward attention [25] for the acoustic decoder.

We constructed the posterior and prior network based on LSTM. They share the same network structure, and their difference is input: the posterior takes acoustic features in addition to linguistic features. For an acoustic encoder in the posterior network, we used the same network structure with the reference encoder [125]. We used location-sensitive attention [57] for secondary attention to align acoustic features to phoneme sequences in the posterior network.

Systems

First, we investigate if accent related information can be learned in an unsupervised way. We build a model that does not use any accent labels. We refer to this model as an unsupervised model (U-). In this case, we can not fully expect that latent representations are related to accents. We measure the degree of matching between predicted latent indices and accent labels as tone error rate to check if the learned latent representation is related to accents. We feed ground truth speech to measure the tone error rate and denote this system as US.

Second, we investigate how much accents can be predicted from linguistic features. We build a model using accent labels during training. We refer to this model as a supervised model (S-). During inference, latent accents are predicted from linguistic features by the prior network without any accent labels. In addition, we also investigate predicting latent accent from acoustic and linguistic features by the posterior network (SS). This system validates reconstruction capability of our proposed method as

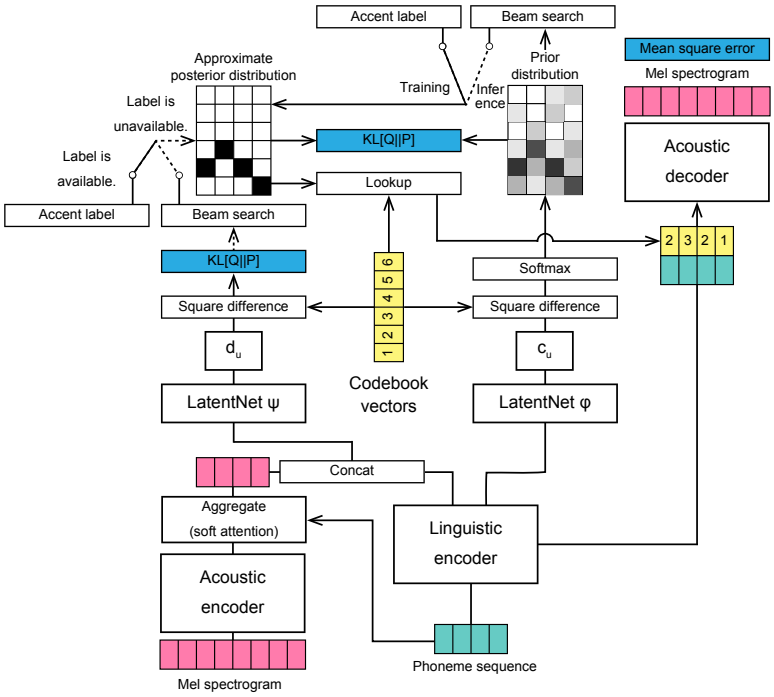


Figure 5.3: Architecture of end-to-end TTS system using pitch accent modeling based on VQ-VAE.

autoencoder.

Third, we investigate effect of phrase information to predict accents, because accents are phrase level information. We refer to this model as phrase model (PP). We include phrase labels as linguistic features in addition to phonemes for this model. The phrase labels are converted to phrase embedding vectors and concatenated with phoneme embedding vectors as an input to the linguistic encoder. During training, accent labels are given for this model as for the supervised model. During inference, the prior network predicts latent accents from the linguistic features containing phoneme and phrase information.

Fourth, we investigate if accent latent space can be shared between tonal languages. We refer to this model as mix model (M-). We mix Chinese speech data with Japanese and use the same accent latent embedding for both languages. We check if there is any improvement on the Japanese side by the mixing.

Accent and phrase labels:

For the accent label, we use abstract tone contour labels aligned to phoneme sequences. We use tone labels in X-JToBI format [118] for Japanese, which consist of three pitch contour patterns in Tokyo dialect: HL (falling), L (neutral low), and H (neutral high). For Chinese, we use tone labels in C-ToBI format [117], which consist of six pitch contour patterns: LL (low), HH (high), LH (rising), HL (falling), L (neutral low), H (neutral high).

For the phrase label, we use hierarchical phrase boundary information. The phrase boundary has three types: no boundary, accent phrase boundary, and breath phrase boundary.

Data

For the Japanese speech corpus we used ATR Ximera dataset [110]. This corpus contains 28,959 utterances from a female speaker and is around 46.9 hours in duration. We used manually annotated linguistic features including phoneme and phrase boundaries [16]. Tone labels in X-JToBI format were obtained by converting the linguistic features based on accentual type. We used 27,999 utterances for training, 476 for validation, and 474 for testing.

For a Chinese speech corpus, we used a Chinese Mandarin speech corpus [1], which contains 12 hours or 10,000 sentences of reading speech from a female speaker. Tone labels are obtained from pinyin annotations. We used 9,000 utterances for training, 500 for validation, and 500 for testing.

As acoustic features, we used mel spectrogram in 80 bins. We downsampled waveforms in 24 kHz sampling rate and trimmed the beginning and ending silence. Waveforms were generated from mel spectrogram with WaveNet [12].

Training and prediction conditions:

During training and prediction, we used a beam width of three to sample pitch accent sequences. We set a reduction factor of two. We used the Adam optimizer [111] with learning rate of 5.0×10^{-5} to optimize the models.

[1] https://www.data-baker.com/open_source.html

Evaluations

We evaluated our proposed systems with objective and subjective evaluations.

As objective evaluation metrics, we measured tone error rate (TER). The tone error rate is measured at the phoneme level. It is not possible to measure tone error rate for the unsupervised model, because it is uncertain if latent space represents accent-related information. To check if the unsupervised model's latent space is accent-related, we report TER between predicted latent indices and tone labels with minimum TER from all combinations of the indices and labels. We fed ground-truth speech to measure TER for this system to ignore prediction errors given linguistic features alone.

For subjective evaluation, we conducted a listening test. We included eight systems in the test: the six proposed systems, analysis-by-synthesis, and natural samples. Listeners were asked two questions for each sample: the first question was about naturalness of speech in five grade MOS score (Very bad, Bad, Acceptable, Good, Very good), and the second question was accuracy of pitch accents in four grade MOS score (Totally inaccurate, Likely inaccurate, Likely accurate, Totally accurate). Each sample was evaluated five times in total. Listeners evaluated 32 samples per set, and were allowed to evaluate at most 10 sets. We recruited 156 listeners with crowdsourcing, and obtained 18,880 evaluations for each question.

5.6.1 Experimental results

Figure 5.1 shows tone error rate (TER) from each system. First, a system trained without any labels (system US) had large TER (56.7 %) even when ground-truth speech was given during inference. It means that latent variables probably do not represent pitch accent information in this setting. Second, a system trained with tone label (system S-) had relatively low TER (25.1 %) compared to unsupervised model (US). This model reduced TER to 6.1 % when ground truth speech was given during inference (SS). A system given phrase information (PP) had TER of 13.5 %, which is low compared to the system without phrase information (system S-), indicating importance of phrase information when predicting tones. Finally, data argumentation by mixing with Chinese speech corpus did not improve TER much (system M-, TER: 23.0 %) compared to equivalent system without data augmentation (system S-).

Figure 5.4a shows a result of listening test about naturalness. A system trained

Table 5.1: Tone error rate (TER) from each TTS system using pitch accent modeling.

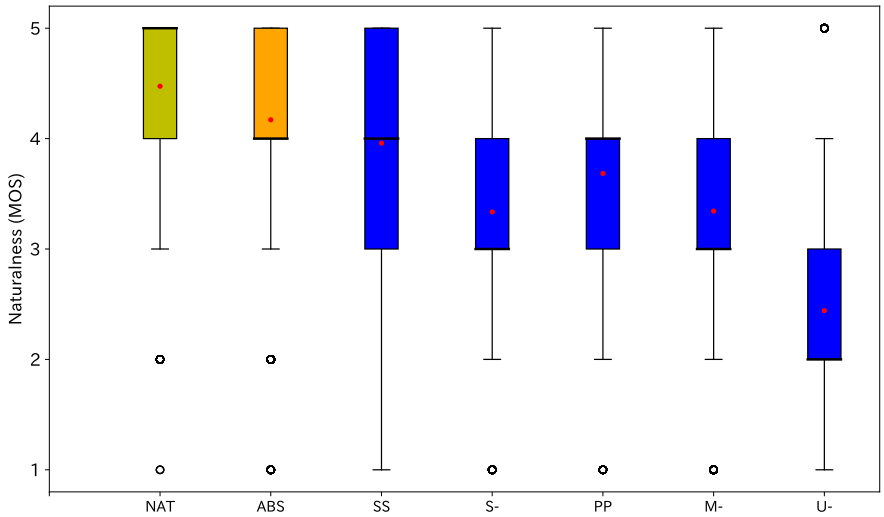
System	Accent label		TER (%)
	Training	Inference	Test
U-	-	-	-
US	-	Speech	56.7
S-	Tone	-	25.1
SS	Tone	Speech	6.1
PP	Tone & Phrase	Phrase	13.5
M-	Tone	-	23.0

without any labels (system U-) had very low MOS of 2.44 ± 0.03, which indicated that latent variables did not capture accent related information under training without accent labels. A system trained with tone label (system S-) had MOS of 3.34 ± 0.04, which is a higher score than the unsupervised model, indicating accent labels were necessary to capture accent information in latent space. The model trained with accent labels showed high MOS of 3.96 ± 0.03 when ground truth speech was given during inference (system SS), which indicated it performed well as an autoencoder. A system given phrase information (system PP) had MOS of 3.68 ± 0.03, which was higher than the system without phrase information, indicating phrase information was a good hint to predict pitch accents. A system with data argumentation by mixing with Chinese speech corpus (system M-) had MOS of 3.34 ± 0.03, showing no improvement from the system without data augmentation (system S-).

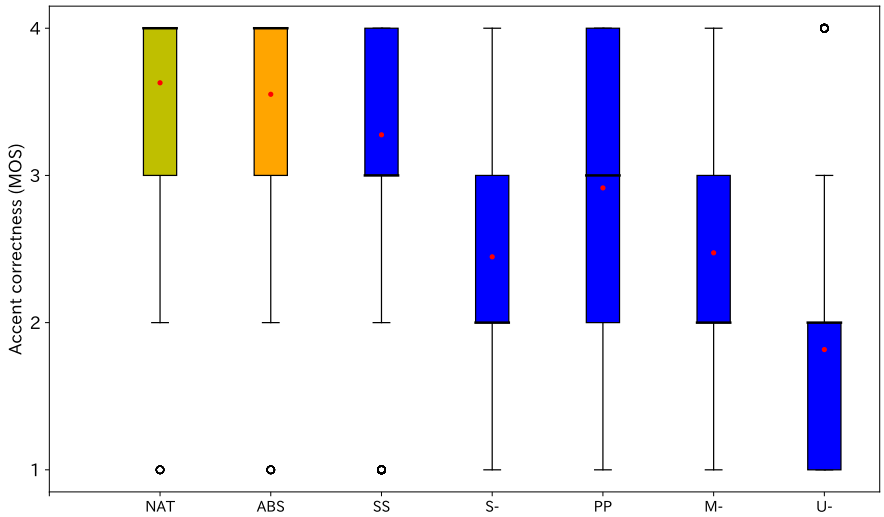
Figure 5.4a shows a result of listening test about pitch accent accuracy. A system trained without any labels (system U-) was rated as inaccurate accents (MOS: 1.82). A system trained with tone label (system S-) and data argumentation (system M-) were rated between likely accurate and likely inaccurate (MOS: 2.45 for system B, 2.47 for system F). A system trained with accent labels and given ground-truth speech during inference (system SS) was rated as accurate accent (MOS: 3.28). A system given phrase information (system PP) was rated likely accurate on average (MOS: 2.92).

5.7 Conclusion

Our pitch accent latent model incorporated in TTS successfully modeled pitch accents, as shown by the high MOSs and low tone error rates when ground truth speech was given. It indicates pitch accent recognition model or encoder can predict tone class from speech properly. In TTS settings where ground truth speech is unavailable, pitch accent predictor showed degradation of accuracy as we can see mediocre MOSs and relatively high TER. This is somewhat expected because lexical pitch accents are not fully predictable from phoneme sequences alone which lose word-level information. Still, our pitch accent latent model greatly advanced the naturalness of synthetic speech in the TTS setting compared to the unsupervised model. The unsupervised model was not able to produce correct pitch accents. This is because pitch accents can not be learned in an unsupervised way, as the very high TER from the unsupervised model suggested.



(a) Naturalness



(b) Accent correctness

Figure 5.4: Results of a listening test.

6 Modeling monotonic alignment in end-to-end TTS

6.1 Motivation of discrete latent alignments

From here on, we focus on phoneme duration as a lexical speech feature. Phoneme duration is an important feature for TTS to construct alignments between source texts and target speech. We focused on pitch accents in Chapters 4 and 5. In Chapter 5, we introduced latent pitch accents to the end-to-end TTS model. Representing with a latent variable is beneficial for alignment modeling as well, so we cover it in this chapter.

Alignment structure of TTS is monotonic, because transcripts are read a-loud left-to-right. It is therefore reasonable to constrain alignment structure to be monotonic by design for robustness and efficiency. Discrete representation of alignments allow us to design monotonic structures much more easily than probability distribution representation of soft-attention. Furthermore, modeling alignments as latent variables

allow us to optimize TTS models based on various probabilistic methods. In this chapter, we formulate our end-to-end TTS method using latent alignment by marginalizing all possible alignments. In Chapter 7, we formulate our end-to-end TTS method using latent duration based on variational inference.

6.2 Alignment in end-to-end TTS and its issues

Sequence-to-sequence [18] is a framework used to convert input sequences to output sequences which have different lengths to each other. All end-to-end TTS methods are based on this framework with different alignment methods.

The original sequence-to-sequence framework [18] does not use alignment. Output sequence is generated from a single context $\mathbf{y}_U$ which is the final encoded input from the encoder $\mathbf{y}_U = \text{Encoder}(y_{1:U})$. This method is not suitable to be applied to long sequences because output sequence has to be generated from a single context no matter how long the output length is. This is not suitable for TTS because output speech is quite long.

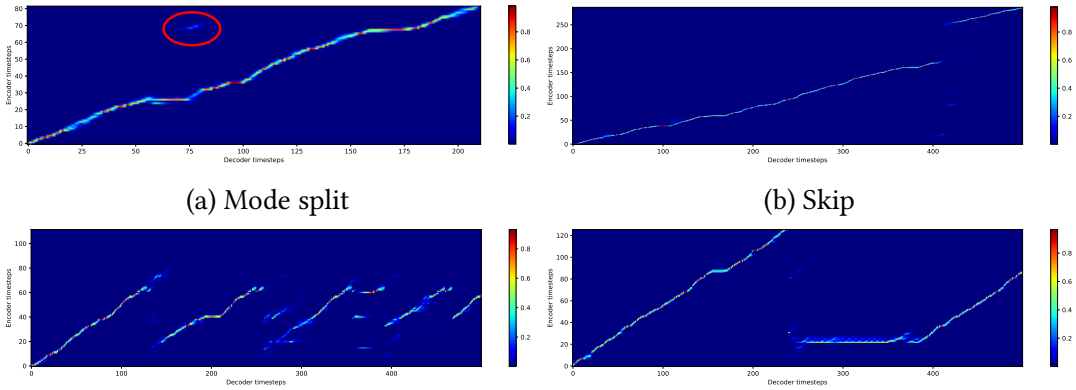
Soft-attention is an alignment method for the sequence-to-sequence framework. Unlike the original sequence-to-sequence, it produces the context for each output. The context is computed by taking expectation of encoder outputs $\mathbf{y}_{1:U}$ by alignment probabilities $p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)$ as in $\sum_{u=1}^{U} p(z_t = u \mid \mathbf{x}_{t-1}, \mathbf{y}_u)\mathbf{y}_u$, where z_t is alignment to relate u and t.

The method to calculate the alignment probabilities is called attention mechanism, and many attention mechanisms are proposed for TTS. Additive attention [141] and dot-product attention [57] are content based families [58] that consider input content to align input to output features. GMM attention [84] is a location based attention [58] that considers input location only. Location sensitive attention [58] extends the additive attention by considering the previous alignment, so it has both properties of content-based and location-based attention. The systems using vocoder parameters [21, 20] seem to choose GMM attention. GMM attention has monotonic progress properties for modes of attention distributions for input location, so it is suitable for predicting long sequences like vocoder parameters. The systems based on CNN or self-attention to enable parallel training [34, 35] seem to use dot-product attention which can be combined with positional encoding [91] for constructing

sequential relations and an initial monotonic alignment in parallel. The systems based on RNN seem to use additive attention and its extension [1, 22, 25, 2].

One reason so many attention mechanisms have been used and proposed for TTS is to improve alignment stability of attention mechanism for decoding long speech sequences. Alignment for TTS must be monotonic, but soft-attention does not guarantee monotonic alignment structure. Figure 6.1 shows typical fatal alignment errors in soft-attention. All content-based attentions and their extensions do not guarantee forward progress of alignment from the previous position. They include forward attention [25], which extends the location sensitive attention by incorporating monotonic transition formulation. The GMM attention [84] has monotonic progress properties for the mode position of each Gaussian component. However, it sometimes gives broad, mode-free, or multi-modal distributions that result in muffled speech as shown in Figure 6.1a, possibly because it does not consider the content of the input. Monotonic attention [142, 59] is the only attention mechanism that guarantees monotonic progress by switching to hard attention during inference from soft attention during training [34].

Failing the stop condition of prediction causes alignment error as well. Figure 6.1d shows alignment errors caused by ill stop condition. Recent end-to-end TTS methods predict binary stop flags to determine when to stop prediction [22, 34, 35]. As opposed to predicting the fixed length output, the stop flag enables the avoidance of unnecessary computation. The stop flag introduces an additional loss term in the objective function. Learning stop flags seems to be trivial because it is just a binary classification task and because only one boundary has a flag turn from not stopping to stopping. However, it tends to overfit because its loss has an order of magnitude lower than the acoustic feature loss term. To alleviate this problem, [35] introduced higher weighting at the boundary where a flag reaches a stop state in the binary loss term. The stop flag is an extra complexity to implement such a trivial feature.



(a) Mode split (b) Skip

(c) Repeat (d) Late termination

Figure 6.1: Common fatal alignment errors from soft-attention.

6.3 SSNT-based TTS: TTS using hard latent alignment

6.3.1 Model definition and learning of SSNT-based TTS

Let us denote the input letter or phone sequence as $y_{1:U} = \{y_1, \cdots y_U\}$, where y_u is a u-th letter or phoneme of the input sequence. We then use $\boldsymbol{x}_{1:T} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$ and $\boldsymbol{x}_u \in \mathbb{R}^D$ to denote an output acoustic feature sequence and acoustic features at time t, respectively. Our approach is to model the output acoustic feature sequence $\boldsymbol{x}_{1:T}$ given a letter or phoneme sequence $y_{1:U}$ by marginalizing all possible alignments over a trellis consisting of the input and output sequences:

$$p(\boldsymbol{x}_{1:T} \mid y_{1:U}) = \sum_{\forall \mathbf{z}_{1:T}} p(\boldsymbol{x}_{1:T}, \mathbf{z}_{1:T} \mid y_{1:U}), \tag{6.1}$$

where $\mathbf{z}_{1:T} = \{z_1, \cdots z_T\} = \{1, 1, , \cdots U-1, U\}$ represents one of the possible paths of the trellis. Figure 6.2 shows the trellis structure of our model.

We then use the concept of SSNT [143, 144]. More specifically, we factorize the joint probability of Eq. (6.1) into an alignment transition probability and emission

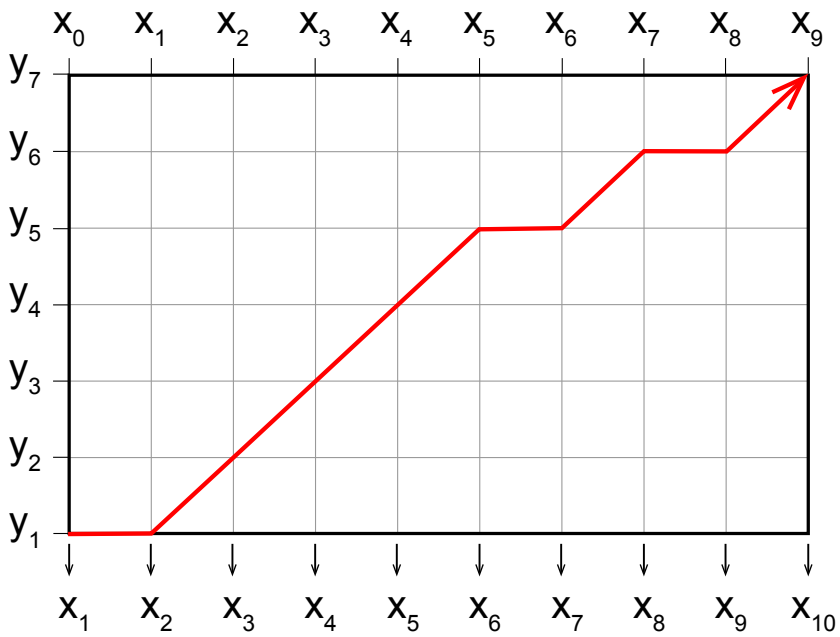


Figure 6.2: Trellis structure of our model. A path that connects $\boldsymbol{x}$ and $\boldsymbol{y}$ represents alignment.

probability for acoustic features with the 1st-order Markov assumption of $\mathbf{z}$:

$$p(\boldsymbol{x}_{1:T}, \mathbf{z}_{1:T} \mid y_{1:U}) \approx \prod_{t=1}^{T} p(z_t \mid z_{t-1}, \boldsymbol{x}_{1:t-1}, y_{1:U}) p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1}, z_t, y_{1:U}) \tag{6.2}$$

and we use neural networks to compute the two probabilities.

To constrain the alignment probability as left-to-right with a self transition, an alignment transition variable with two possible values, $a_{t,u} = \{\texttt{Emit}, \texttt{Shift}\}$ is

further introduced for the alignment probability of Eq. (6.2):

$$p(z_t = u \mid z_{t-1}, \boldsymbol{x}_{1:t-1}, y_{1:U}) = \begin{cases} 0 & z_{t-1} > u \\ p(a_{t,u} = \texttt{Emit}) & z_{t-1} = u \\ p(a_{t,u-1} = \texttt{Shift}) & z_{t-1} = u-1 \\ 0 & z_{t-1} < u-1 \end{cases} \tag{6.3}$$

Note that the `Emit` transition keeps the input position, while `Shift` transition proceeds and reads one more input. Please also note that $p(a_{t,u} = \texttt{Emit}) = 1 - p(a_{t,u} = \texttt{Shift})$ and that a neural network predicts only one of them.

The emission probability was a discrete distribution because SSNT was originally proposed for NLP tasks such as abstractive sentence summarization, morphological inflection generation, and machine translation, and because its outputs are words, the emission probability was a discrete distribution. In our case, the output is continuous, and hence we have to define our own emission probability distribution function. We simply use a multivariate Gaussian distribution as the emission probability of the acoustic features:

$$p(\boldsymbol{x}_t \mid \boldsymbol{t}_{1:t-1}, z_t, y_{1:T}) = \mathcal{N}(\boldsymbol{x}_t; \boldsymbol{\mu}, \sigma^2\mathbf{I}) \tag{6.4}$$

Note that $\boldsymbol{\mu}$ is predicted by an encoder-decoder network with autoregressive feedback similar to Tacotron.

Our model can be trained by minimizing the negative log likelihood:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= -\log p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:U}; \boldsymbol{\theta}) \\ &= -\log \alpha(T, U) \end{aligned} \tag{6.5}$$

Here $\alpha(T, U)$ is a forward variable of the forward-backward algorithm at the final input position U and the final time step T. The final forward variable can be calculated

recursively:

$$
\begin{aligned}
&\text{For } t = 1, \\
&\quad \alpha(1, u) = p(z_1 = 1 \mid y_{1:U}) p(\boldsymbol{x}_1 \mid z_1, y_{1:U}) \\
&\text{For } t > 1, \\
&\quad \alpha(t, u) = p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1}, z_t, y_{1:U}) \cdot \\
&\quad \big\{\alpha(t-1, u-1) p(z_t = u \mid z_{t-1} = u-1, \boldsymbol{x}_{1:t-1}, y_{1:U}) \\
&\quad + \alpha(t-1, u) p(z_t = u \mid z_{t-1} = u, \boldsymbol{x}_{1:t-1}, y_{1:U})\big\}
\end{aligned}
\tag{6.6}
$$

The gradients of the negative log likelihood with respect to $\boldsymbol{\theta}$ can be computed using the standard back-propagation algorithm. For more efficient gradient computation, the gradient can be computed with the following form:

$$
\begin{aligned}
&\frac{\partial \log p(\boldsymbol{x}_{1:T} \mid y_{1:U}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \\
&\frac{1}{p(\boldsymbol{x}_{1:T} \mid y_{1:U}; \boldsymbol{\theta})} \sum_{t=1}^{T} \sum_{u=1}^{U} \frac{\partial p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:U}; \boldsymbol{\theta})}{\partial \alpha(t, u)} \frac{\partial \alpha(t, u)}{\partial \boldsymbol{\theta}}
\end{aligned}
\tag{6.7}
$$

Then the back-propagation can be combined via backward operation of the forward-backward algorithms by introducing a backward variable $\beta(t, u)$ and utilizing the relation $\frac{\partial p(\boldsymbol{x}_{1:T} \mid y_{1:U}; \boldsymbol{\theta})}{\partial \alpha(t, u)} = \beta(t, u)$ [143, 145].

6.3.2 Network structure of SSNT-based TTS

Figure 6.3 shows the detailed network structure of the system for a SSNT-based speech synthesis system. The network consists of an encoder and decoder. The encoder processes either a letter or phone sequence. We then use the CNN-based encoder [22], which consists of a stack of convolutional layers, and a bidirectional LSTM layer [88].

The decoder processes the acoustic feature sequence in an autoregressive manner. The predicted acoustic features from the previous time step are first processed by the pre-net [1], which consists of fully connected layers with ReLU activation [146] and dropout regularization [64]. The pre-net's output is passed through a stack of LSTM layers [147]. The LSTM stack's output is concatenated with the encoder's output and

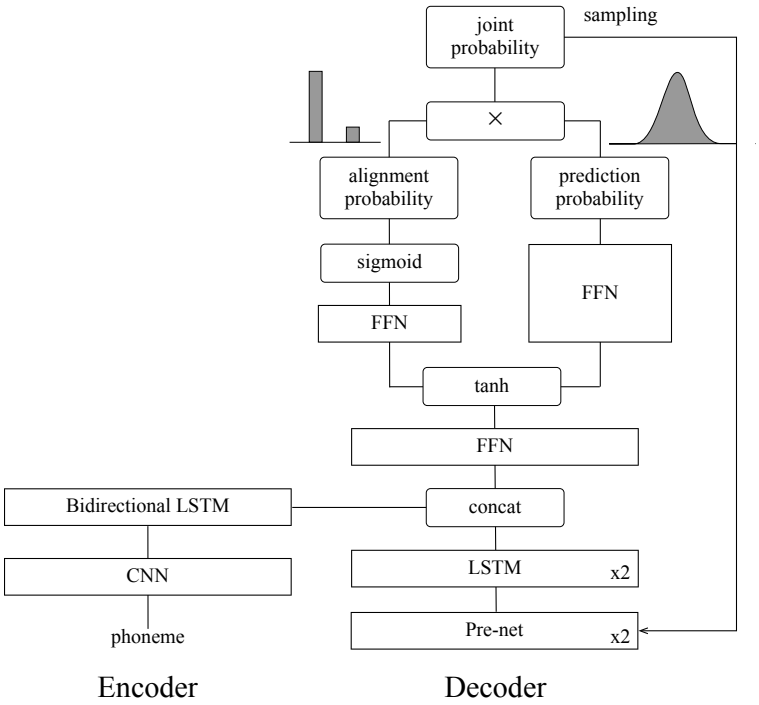


Figure 6.3: Detailed network structure of the SSNT-based TTS system.

then transformed by a fully connected layer with tanh activation. The output of tanh nonlinearity is passed to two networks. One is a fully connected layer with sigmoid activation to compute the alignment transition probability $p(a_{t,u})$ of Eq. (6.3). The other is a linear layer to compute the emission probability of Eq. (6.4).

6.4 Sampling methods of hard alignment

Because SSNT-TTS is based on hard alignment, it not only predicts the probability values of alignment but also determines the best discrete alignment path during inference. This is quite different from soft attention, which predicts only probability values and uses them as the alignment to do a weighted sum over the encoder's output. For the determination of the best discrete alignment path, SSNT-TTS samples alignment transition variable $a_{t,u} \in \{\texttt{Emit}, \texttt{Shift}\}$ from a learned probability distribution at each time step.

One basic example of alignment sampling methods is sampling from the Bernoulli

distribution. In Bernoulli sampling, the alignment variable z_t is incremented by sampling from the Bernoulli distribution with a parameter obtained by normalizing the two nonzero cases of Eq. (6.3), namely, $z_t = z_{t-1} + k$, where $k \sim \text{Bernoulli}(a_{t,u} = \texttt{Emit})$.

Note that this is different from a transition matrix in HMM synthesis which models duration with exponential distribution. Our system models transition of alignment as a latent variable so our system does not define distribution of duration like HMM synthesis.

In the original SSNT, its prediction stops when the EOS token is the output. In our case, its prediction stops when the alignment variable reaches the final input position[1] causing all of the input sequence to be used.

In the following sections, we describe various alignment sampling methods for SSNT-TTS. We investigate conditions for the alignment sampling methods in experiments.

6.4.1 Alignment sampling from a discrete distribution

A well-known method to sample a random variable from a discrete distribution is the Gumbel-Max trick [148]. The algorithm of the Gumbel-Max trick consists of two steps: one, sampling a continuous random variable from a learned distribution, and two, performing the argmax operation to output a final symbol. Let us denote the density of the binary transition variables as $p(a_{t,u} = \texttt{Emit}) = \alpha_1$ and $p(a_{t,u} = \texttt{Shift}) = \alpha_2$, where $\alpha_1 + \alpha_2 = 1$. To randomly sample a discrete alignment transition variable at input u and time step t with the Gumbel-Max trick, Gumbel noise is first added to each logit of the alignment transition densities, which can be written as

$$\mathbb{P}(a_{t,u} = \texttt{Emit}) = \mathbb{P}(G_1 + \log \alpha_1 > G_2 + \log \alpha_2) \tag{6.8}$$

$$= \mathbb{P}(L + \log \alpha_1 > \log \alpha_2), \tag{6.9}$$

where G_1 and G_2 are the Gumbel noises and $L = G_1 - G_2$. Since the difference L is known to follow a Logistic distribution, a Logistic noise can be sampled by $L = \log(U) - \log(1 - U)$, where U is a sample from the standard Uniform distribution. Finally, the argmax operation is applied to obtain a discrete sample of the alignment

[1] [144] uses the same criteria to stop prediction to consume full input.

transition variable

$$a_{t,u} = \text{argmax}(L + \log \alpha_1, \log \alpha_2). \tag{6.10}$$

This is a typical approach to drawing random samples from the Bernoulli distribution, and we refer to this condition as "Logistic condition" because it involves sampling from the Logistic distribution in continuous space. We will introduce another continuous distribution for the alignment transition distribution in Section 6.4.3.

In implementation, because $\alpha_1 + \alpha_2 = 1$, we parameterize them as $\alpha_1 = \text{sigmoid}(\log \alpha, \lambda)$ and $\alpha_2 = 1 - \text{sigmoid}(\log \alpha, \lambda)$, where $\text{sigmoid}(x, \lambda) = \frac{1}{1+\exp(-x/\lambda)}$ and λ is a temperature hyper-parameter to control the shape of the sigmoid function (see Figure 6.4). Accordingly, SSNT only needs to predict a single value of α at each time step. It can be shown that $\alpha = \alpha_1/\alpha_2$.

6.4.2 Stochastic alignment search

Since SSNT-TTS treats the alignment as a latent variable, it must decode a single alignment path from all possible alignments during inference. From here on we refer to the process of selecting the best alignment as *search*.

Greedy search is a search method that chooses a single path with a higher alignment transition probability at each time step, as

$$a_{i,j} = \text{argmax}(\log \alpha_1, \log \alpha_2) = \begin{cases} \texttt{Emit} & \text{if } \alpha_1 > \alpha_2 \\ \texttt{Shift} & \text{if } \alpha_1 < \alpha_2 \end{cases}. \tag{6.11}$$

From equations (6.10) and (6.11), we can see that the difference between the greedy search and random sampling from a Bernoulli distribution is just the Logistic noise L. For this reason, we refer to random sampling from a Bernoulli distribution as *stochastic greedy search* for convenience.

A generalized version of greedy search is the beam search method. The beam search method considers several candidates of alignments at each time step of decoding and selects the overall best candidate alignment path. The number of candidates is called beam width. Greedy search is a special case of beam search with beam width equal to one. Furthermore, by combining the sampling from the Bernoulli distribution and beam search, or more concretely, by adding the Logistic noise to the argmax

operations during beam search, *stochastic beam search* can be easily implemented.

6.4.3 Continuous relaxation of discrete alignment variables

As we see in equation (6.9), the conventional random sampling from the Bernoulli distribution uses the Logistic distribution. The use of the Logistic distribution is, however, an oversimplified assumption since it has a symmetric shape and a single mode. More specifically, under this condition, similarity between the sample distribution and the actual discrete distribution mainly depends on the prediction accuracy of the α_1 parameter.

To model the hard alignment more appropriately, we propose using the *binary Concrete distribution* [129] for the alignment transition probability. The Concrete distribution, which is also known as the Gumbel-Softmax distribution [149], is a continuous distribution that relaxes the discrete argmax operation. Here, we focus on the binary case of the Concrete distribution because our alignment transition variable $a_{t,u}$ has two states only.

Using a concept similar to the Gumbel-Max trick, we can achieve random sampling from the binary Concrete distribution. Like the procedure described in Section 6.4.1, random sampling from the binary Concrete distribution starts with adding the Logistic noise L to a logit of the density of the alignment transition variable. Here, we introduce parameter $\alpha = \alpha_1/\alpha_2$, and consider the logit of the α instead. First, we add the Logistic noise L to a sigmoid function:

$$\mathbb{P}(a_{t,u} = \texttt{Emit}) = \frac{1}{1 + \exp\left(-(\log \alpha + L)/\lambda\right)} \tag{6.12}$$

where λ is a temperature parameter. This sigmoid function is used as a continuous version of the argmax function and a discrete sample can be obtained by applying the argmax operator to the output of the sigmoid function. The resulting distribution has the following form:

$$\text{BinConcrete}(x|\alpha, \lambda) = \frac{\lambda \alpha x^{-\lambda-1}(1-x)^{-\lambda-1}}{\left(\alpha x^{-\lambda} + (1-x)^{-\lambda}\right)^2}. \tag{6.13}$$

Figure 6.4 shows the binary Concrete distribution with α and λ of various values. The

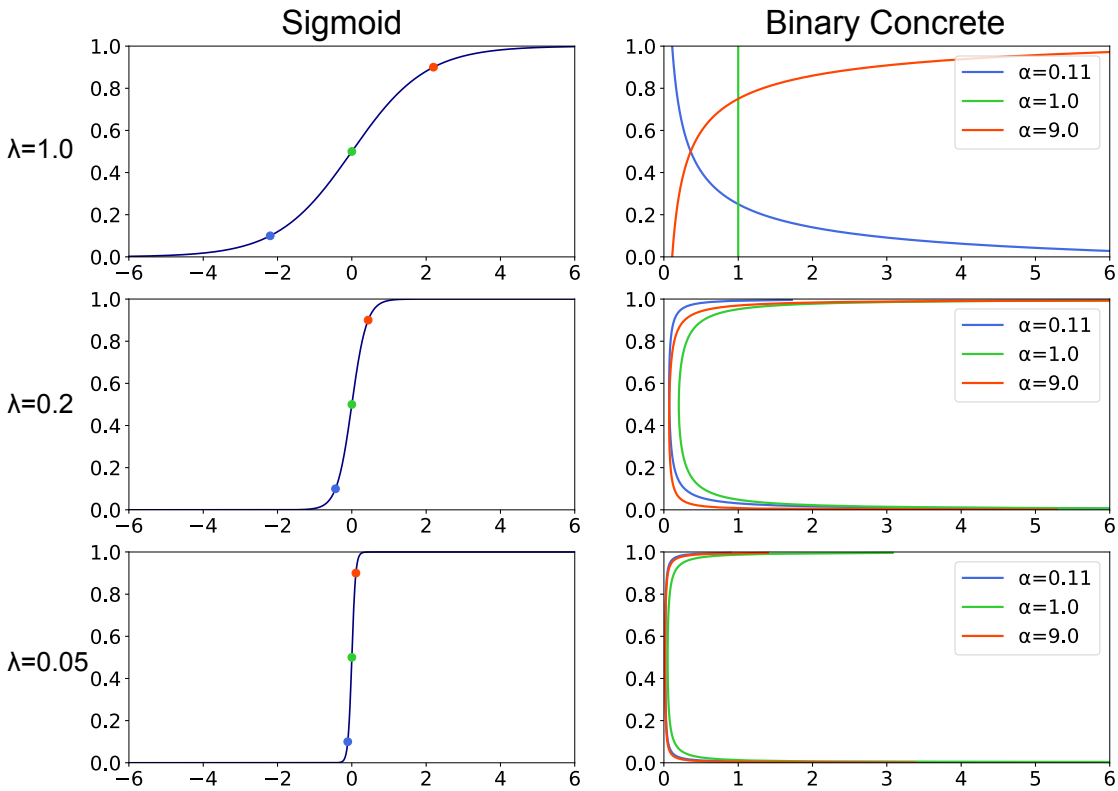


Figure 6.4: Left: Sigmoid function. Right: Binary Concrete distribution. Points on sigmoid function indicate α parameters for binary Concrete distribution.

parameter α can be learned by using the Logit noise during training. We refer to the condition using the binary Concrete distribution as the "binary Concrete condition".

Figure 6.5 summarizes the sampling process of the Logistic and binary Concrete conditions implemented with neural networks. We can see that both conditions have a similar network structure, and the difference is the location where Logistic noise is added: the Logistic condition has the Logistic noise after the sigmoid function, while the binary Concrete condition has the Logistic noise before the sigmoid function. Note that the use of Logistic noise is mandatory during training for the binary Concrete condition, as the randomness makes a model learn the parameter α of the binary Concrete condition. In contrast, the Logistic condition does not use Logistic noise during training, since during training a model computes likelihood, not samples. At inference time, the use of the Logistic noise can be optional in both conditions. If the Logistic noise is omitted during inference, it is equivalent to a deterministic search

using mean value α_1 for both the Logistic and binary Concrete conditions.

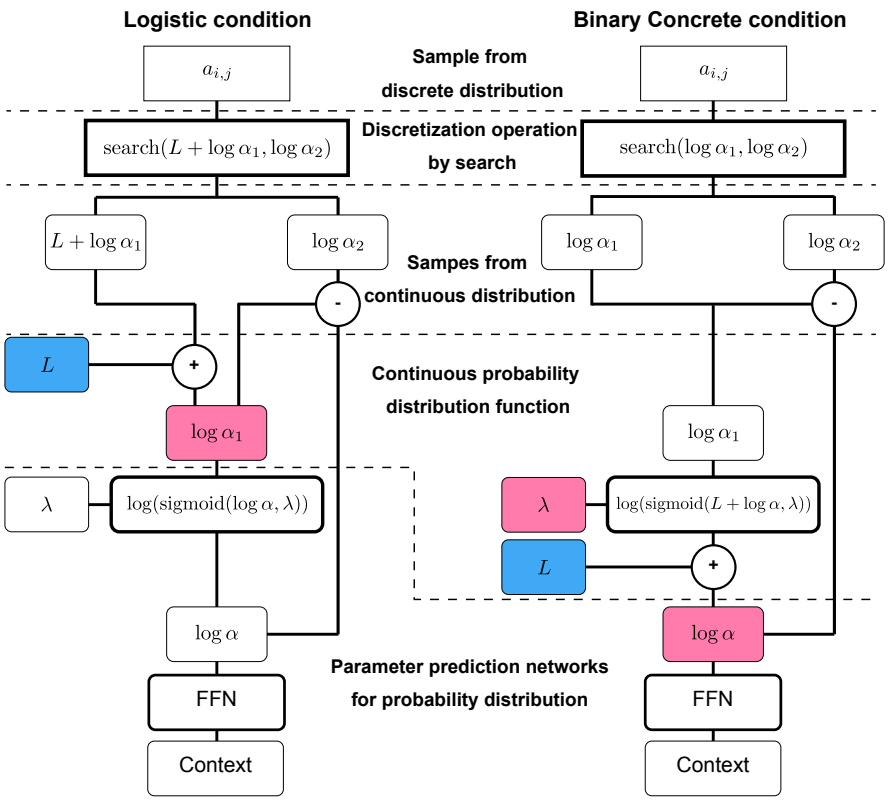


Figure 6.5: Sampling process of discrete alignment transition variable. Left: Logistic condition. Right: Binary Concrete condition. Bold boxes are functions, and normal boxes are values. Round boxes are continuous, and square boxes are discrete functions/values. Blue boxes are random noise, and red boxes are parameters of probability distribution.

6.5 Experiments

We conducted two experiments using SSNT-TTS. The first experiment is to compare SSNT-TTS with a major end-to-end TTS method (Tacotron), using basic alignment sampling method for inference (Bernoulli sampling). The second experiment is to compare alignment sampling methods described in 6.4.

6.5.1 Experimental conditions (1)

We used the same conditions as in our previous experiment [2]. A Japanese speech corpus from the ATR Ximera dataset [110] was used. This corpus contains a total of

28,959 utterances from a professional female speaker and is around 46.9 hours in duration. We used manually annotated phonemes labels [16]. The phoneme labels had 58 classes, including silences, pauses, and short pauses. All sentences start and end with a silence symbol. Although Japanese is a pitch-accented language, we did not use accentual type labels in this experiment. To train our proposed systems, we trimmed the beginning and ending silence from the utterances, after which the duration of the corpus was 33.5 hours. Fixed length silences were prepended and appended to target mel spectrogram. These data preparations made it inappropriate to skip any phoneme symbols. The frame size and shift used for the mel spectrogram were 50 ms and 12.5 ms, respectively. We used 27,999 utterances for training, 480 for validation, and 480 for testing.

Phoneme embedding vectors have 256 dimensions. For the encoder, we used the same conditions as [22]. For the decoder, we used two pre-net layers with 256 and 128 dimensions, two LSTM layers with 256 dimensions each, and two fully connected layers for context projection with 256 dimensions each. We applied zoneout regularization [87] to all LSTM layers with probability 0.1 as in [22]. We use random sampling from the Bernoulli distribution for the alignment sampling method. We set the reduction factor [1] to be 2. The Adam optimizer was used for training [111]. The validation loss was steady so we stopped training at 510 epochs by checking the quality of predicted spectrogram in validation set. Finally, we used μ-law WaveNet for the waveform generation [109].

6.5.2 Subjective evaluations

We conducted a listening test to measure the quality of synthetic speech. We chose Japanese Tacotron and self-attention Tacotron without accentual type labels [2] as baselines in this experiment. All the synthetic speech waveforms were generated using the identical WaveNet model, which was trained using natural mel spectrograms with a 12.5 ms frame shift and 16 kHz sampling frequency.

We recruited 104 native speakers of Japanese as listeners by crowdsourcing. The listeners evaluated 40 audio samples that contained eight randomly selected sentences generated by each of five systems in a random order in a single test set. The systems included natural speech and analysis-by-synthesis in addition to our system and the

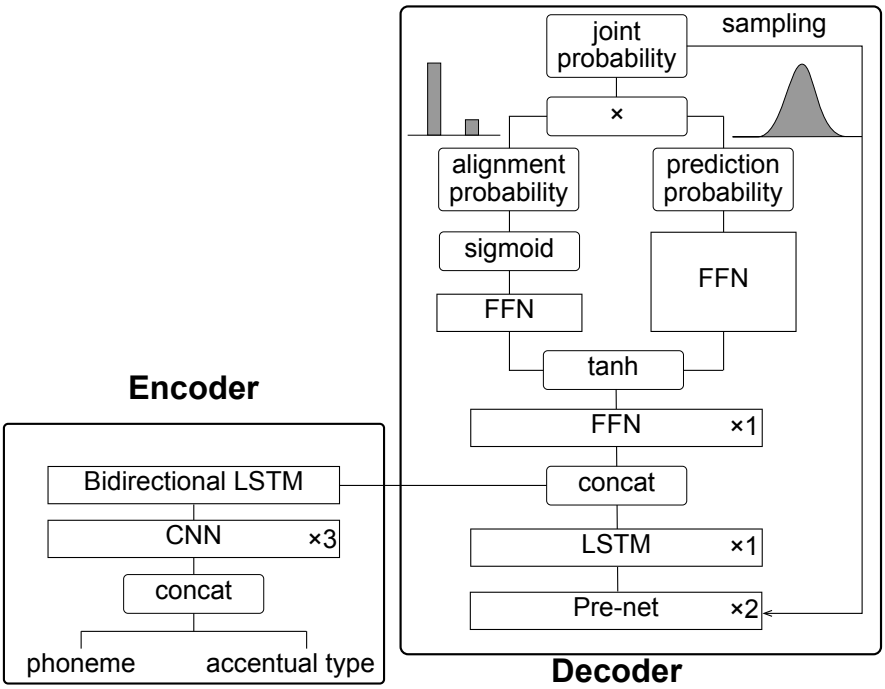


Figure 6.6: Network architecture of SSNT-TTS.

two aforementioned baseline systems. One listener could evaluated at most ten test sets. One audio sample was evaluated five times, and we got a total of 19,200 data points.

6.5.3 Experimental conditions (2)

We performed an experiment to compare the Logistic and binary Concrete conditions for SSNT-TTS. For each condition, we trained multiple SSNT-TTS models with different values of the hyper-parameter λ, which can be 1.0, 0.2, or 0.05. The effect of λ on the Logistic and binary Concrete distributions is shown in Figure 6.4.

Given a trained SSNT-TTS model, we further compared combinations of different alignment search methods during inference. First, we compared the greedy and beam search methods. In the case of beam search, we used a beam width equal to 10. For both greedy and beam searches, we further compared deterministic and stochastic ways to determine the alignment transition action $a_{i,j}$ at each time step. The deterministic way means absence of the Logistic noise while the stochastic way uses the Logistic noise during inference.

We used the ATR Ximera dataset [110], a Japanese speech corpus containing 28,959 utterances totaling around 46.9 hours from a professional female speaker. We used manually annotated phoneme and accentual type labels [16]. The phoneme labels consist of 58 classes, including silences, pauses, and short pauses. All sentences start and end with a silence symbol. To train our proposed systems, we trimmed the beginning and ending silence from the utterances, after which the duration of the corpus was 33.5 hours. Fixed length silences were prepended and appended to target mel spectrograms. The frame size and shift used for the mel spectrogram were 50 ms and 12.5 ms, respectively. We used 27,999 utterances for training, 480 for validation, and 480 for testing.

We used the same SSNT-TTS implementation as [150], except for the configuration of network parameters. Figure 6.6 shows the network architecture of SSNT-TTS. Phoneme embedding vectors had 512 dimensions, and the accentual type embedding had 64 dimensions. For the encoder, we used the same conditions as [22]. For the decoder, we used two pre-net [1] layers with 256 and 128 dimensions, a single LSTM layer with 512 dimensions, and a single fully connected layer for context projection with 512 dimensions. We applied zoneout regularization [87] to all LSTM layers with probability 0.1, as in [22]. We used the Adam optimizer [111] and trained the models with a batch size of 100 and a reduction factor equal to 2 [1]. To reduce training time, we used mixed-precision training during which the computation was mainly done in half precision, but the model parameters were kept in normal precision. Furthermore, to prepare models in various conditions, we first created a seed model in the Logistic condition with temperature 0.05 running up to 270 K steps. Second, we derived models with temperature 1.0 and 0.2 by fine-tuning the seed model by 100 K steps. Finally, models with the BinConcrete condition were further trained by fine-tuning the models with the Logistic noise by 100 K steps. We used μ-law WaveNet for the waveform generation [109][2].

6.5.4 Subjective evaluation

We organized a listening test to evaluate the proposed methods and recruited 193 native Japanese listeners by crowdsourcing. Listeners were presented with 40 samples

[2] Audio samples can be found at our web page https://nii-yamagishilab.github.io/sample-ssnt-sampling-methods

from 20 systems and asked to judge the naturalness of speech using a five-point MOS score. The 20 systems included natural and analysis-by-synthesis samples in addition to the 18 SSNT-TTS models with different training and inference configurations, which are summarized at the bottom of Figure 6.9. One listener could evaluate at least three and at most ten sets. Each sample was evaluated three times, and we obtained 28,800 evaluations in total. We checked the statistical significance of MOS scores with a pairwise t-test.

6.5.5 Experimental result (1)

Figure 6.7 shows the results of the subjective evaluation. Among the baseline systems, self-attention Tacotron got 3.13±0.03, and Japanese Tacotron got 3.05±0.03. The scores of the baseline systems are consistent with the previous work [2]. The scores were relatively low because we did not provide any pitch accentual type labels, even though Japanese is a pitch-accented language. Our system got a MOS score of 2.33±0.03. Unfortunately, it was rated lower than the baseline systems.

To understand the reasons, we did a simple investigation of generated audio files and discovered that our model overestimated the phoneme duration, especially for pauses within a sentence. Pauses had much longer duration than other phonemes, but, its acoustic features had less useful information; hence, deciding when `Shift` transition should be made would be difficult. Figure 6.8 shows an example sample that has alignment error of overestimation of pause duration. In fact, the MOS score of sentences that did not include pauses was 2.75 ± 0.06 and 3.24 ± 0.05 for SSNT and Tacotron, respectively, and the score difference was smaller. We also discovered that our method needed longer training time due to the marginalization process. The performance might be improved after using sufficiently long training periods.

6.5.6 Experimental result (2)

Figure 6.9 shows the results of the listening test about alignment sampling methods. If we compare sampling conditions such as randomness and search method with the same temperature parameter λ, we see a few interesting findings.

First, deterministic search conditions always outperformed stochastic search conditions.

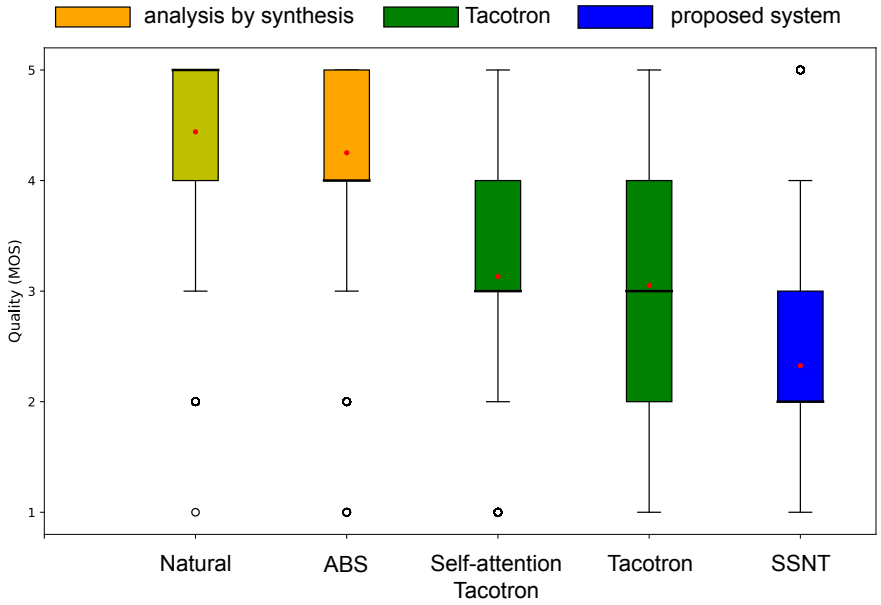


Figure 6.7: MOS scores of subjective evaluation.

Second, the effect of the search methods on the MOS scores depended on randomness, whether using deterministic search or stochastic search. Under the deterministic condition, systems with beam search always performed better than those with greedy search. On the other hand, under the stochastic and Logistic conditions, beam search performed worse than greedy search. For the BinConcrete condition, neither of the two search methods showed much difference.

If we focus on probability distribution variations while fixing temperature, we see that their scores depended on the use of randomness as well. With deterministic search, the Logistic conditions had slightly higher scores than the BinConcrete conditions, but their difference is not statistically significant. In contrast, stochastic search in the Logistic condition performed much worse than in the BinConcrete condition. The poor performance of stochastic search in the Logistic condition was mitigated by decreasing the temperature parameter.

Because the Logistic and BinConcrete conditions had similar scores under the deterministic search, we can infer that both conditions were capable of estimating the alignment transition boundary. However, we also see that the Logistic condition does not parameterize a proper alignment transition distribution because it performed

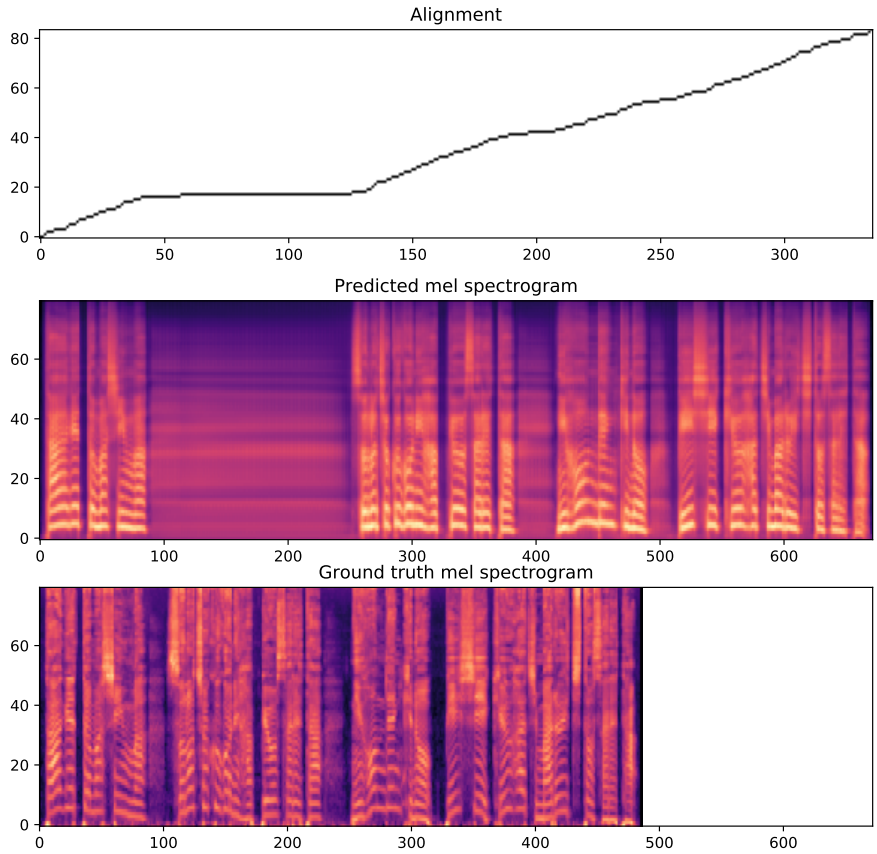


Figure 6.8: A sample that shows overestimation of pause duration.

very badly under stochastic search, especially in stochastic beam search. Meanwhile, the BinConcrete conditions were relatively robust to stochastic search conditions, presumably because it can fill the gap between continuous and discrete distributions.

6.5.7 Conclusion

We designed the alignment structure to be hard and monotonic, which enabled us to avoid some alignment errors that are commonly observed in soft-attention-based approaches. Such alignment errors include muffling, skipping, and repeating. Muffling error is caused by an attention distribution without a sharp mode, skipping error is caused by discontinuous attention, and repeating error is caused by a repeat of backward jumps of the mode of attention. These errors were not observed in our method because they could not happen by definition.

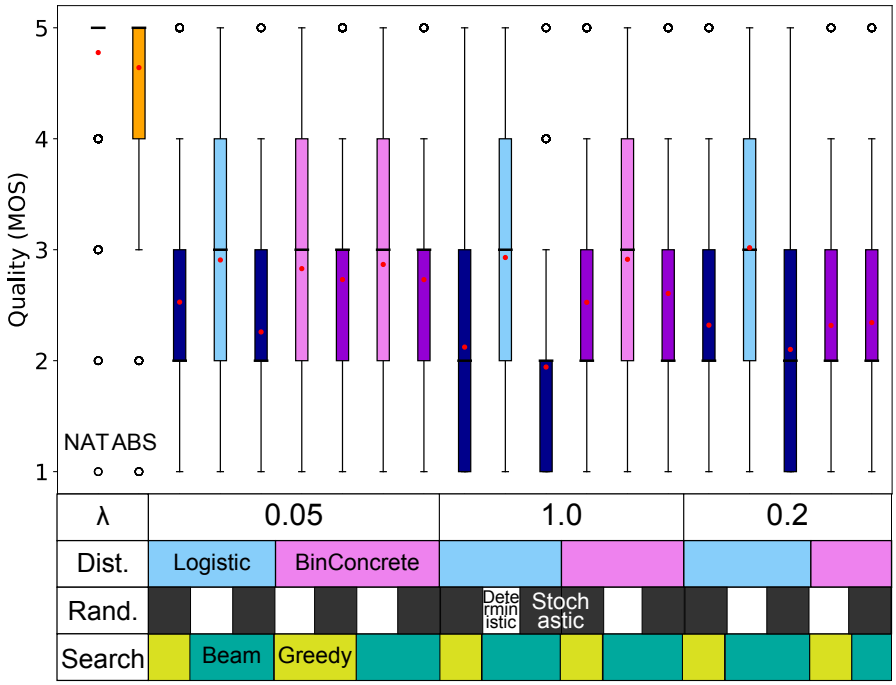


Figure 6.9: Results of listening test. Red circles indicate mean values. Bars show maximum, median, and minimum. NAT is natural sample, and ABS is analysis-by-synthesis.

However, we still observed different types of alignment errors in our samples, such as prolongations of vowel duration. Although we found the best combination of alignment sampling methods, the best system only showed average naturalness of synthetic speech, which was not competitive to end-to-end TTS methods based on soft-attention. Modeling alignments in frame level alignment transition may not be ideal for TTS, and it motivates us to model alignment as phoneme duration in Chapter 7.

7

Modeling duration in end-to-end TTS

7.1 Phoneme duration in TTS

TTS has been using phoneme duration instead of frame-level alignment transitions for alignment, as described in Section 2.1 in Chapter 2. HSMM-based SPSS uses phoneme duration as a latent variable to align source linguistic features with target vocoder parameters. DNN-based SPSS uses duration predictor to align linguistic features with vocoder parameters, and the duration predictor is trained with duration labels predicted by a forced aligner. Phoneme duration is the natural unit for alignment of TTS because, from a speech production perspective, each phoneme is expected to have intrinsic duration [151], and from an engineering perspective, phoneme-level alignment modeling is more efficient than frame-level alignment modeling because of the shorter sequence length.

In Chapter 6, we constructed a TTS method using a latent monotonic alignment called SSNT-TTS. Alignment of SSNT-TTS is expressed as frame-level alignment transition to design monotonic alignment structure. The next step to extend the latent

alignment TTS model is designing phoneme duration as a latent variable, and this is what we cover in this chapter. By using phoneme duration as latent alignment, we can achieve monotonic alignment more efficiently. In addition, we adopt a different probabilistic model from SSNT-TTS for computationally lightweight optimization. We formulate our end-to-end TTS method based on variational inference. Instead of marginalizing all possible alignments like SSNT-TTS, variational inference estimates the expectation term with sampling.

Another topic to be covered in this chapter is how to incorporate the duration model and forced aligner used in the DNN-based pipeline TTS framework to end-to-end TTS. Most works give up on end-to-end approach when they incorporate duration modeling, which replaces soft-attention or hard attention in end-to-end TTS. Instead they decompose the single model into multiple models similar to vocoder-free pipeline TTS framework [79, 80, 81, 82, 63], as described in Section 2.3.3 in Chapter 2. We consider the mathematical role of a duration predictor and forced aligner in end-to-end TTS to construct an end-to-end TTS method using latent duration.

7.2 Related works using duration for alignments

Explicit duration modeling was recently incorporated into neural TTS systems as an alternative to soft- and hard-attention for more robust alignment modeling. For example, FastSpeech [63] trains a duration predictor with alignment obtained from a soft-attention-based teacher model, AlignTTS [82] extracts duration using an aligner based on hard and monotonic alignment, and DurIAN [32] and FastSpeech2 [79] use external forced aligners to extract the duration. Alignments obtained from soft attention are not accurate enough, so JDI-T [32] introduces CTC loss and guided attention loss to the attention distribution of the teacher model. DurIAN [32] and FastSpeech2 [79] replace their alignment method with an external forced aligner from unstable soft attention, and they simplify the training process by removing teacher-student training. AlignTTS [82] and Glow-TTS [152] use an aligner based on hard and monotonic alignment to train a duration predictor to improve the duration label.

FastSpeech [63], FastSpeech2 [79], DurIAN [32], and AlignTTS [82] use independent duration and aligner models, and accordingly they require multiple training phases

to construct a whole TTS system. In contrast, recent models such as JDI-T [32] and Glow-TTS [152] can train the TTS and duration model jointly. Glow-TTS [152] treats duration as a latent variable, although it is not formulated as a VAE. Note that all methods above treat duration as a continuous value, even though duration in those TTS systems corresponds to the number of frames, which is intrinsically discrete.

7.3 TTS using latent duration

7.3.1 Overview of our approach

We construct our end-to-end TTS method using latent duration based on conditional VQ-VAE [139]. Figure 7.1 shows an overview of our idea to integrate end-to-end TTS with a forced aligner and duration model based on VAE. Phoneme duration is latent information inherent in phonemes, so it can be modeled with a conditional prior and the prior corresponds to a duration predictor. Approximate posterior in VAE can be defined with duration derived from a recognition model, and the recognition model corresponds to the forced aligner. Phoneme duration can be represented in the number of acoustic frames, and thus VQ-VAE is suitable to handle discrete latent variables in VAE. Note that this method is what we use for our latent pitch accent model in Chapter 5.

7.3.2 Model definition and variational lower bound

Let us write the input sequence of discrete tokens as $\mathbf{y}_{1:U} = (y_1, \cdots y_U)$, where y_u is the u-th token (e.g., letter or phone). We then use $\mathbf{x}_{1:T} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ to denote an output sequence of acoustic features, where $\mathbf{x}_t \in \mathbb{R}^O$ is the t-th output time step or *frame*. Our goal is to learn a good model $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$.

We use $\mathbf{l}_{1:U} = (l_1, \cdots, l_U)$ to denote the latent duration for $\mathbf{y}_{1:U}$, where the value of l_u is equal to the duration of the u-th input token. For example, in Figure 7.2, $l_2 = 3$ indicates that the second input token y_2 is aligned with three output frames $(\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5)$. For an output sequence with a total duration of T, we constrain $T = \sum_u l_u$. We further constrain $1 \leq l_u \leq K$, where K is a hyper-parameter that decides the maximum value of duration.

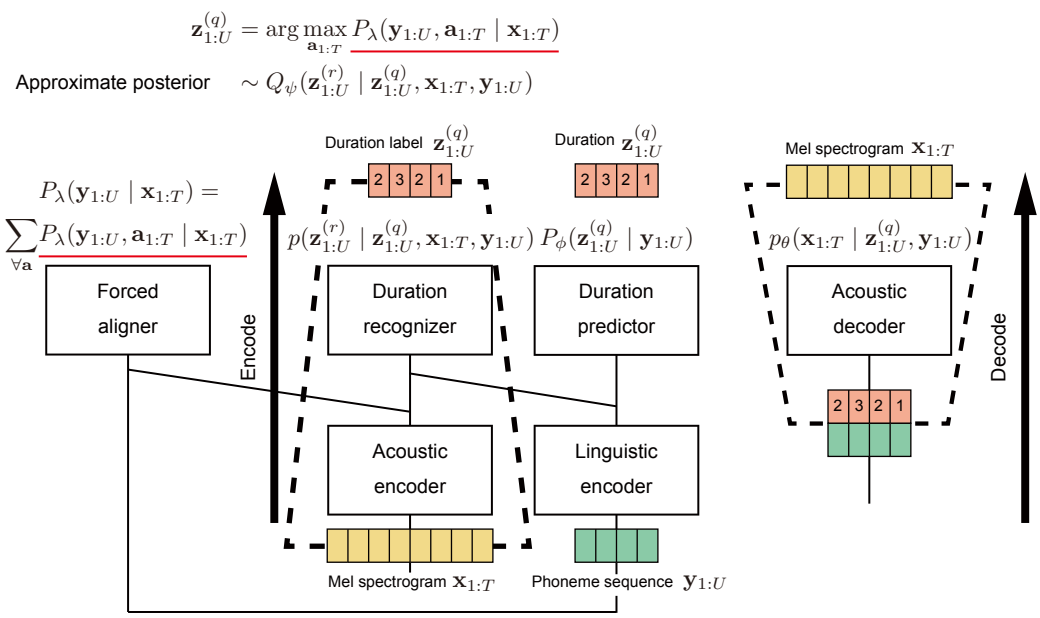


Figure 7.1: Brief overview of idea to incorporate forced aligner and duration predictor to end-to-end TTS based on VAE.

Since l_u is discrete for TTS systems, we can parameterize $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$ with VQ-VAE. Let $\mathcal{Z} = \{\mathbf{e}_1, \cdots, \mathbf{e}_K\}$ be a codebook with K codewords, where $\mathbf{e}_k \in \mathbb{R}^D$. We then introduce an quantized latent vector $z_u^{(q)} \in \mathcal{Z}$ and an un-quantized latent vector $z_u^{(r)} \in \mathbb{R}^D$ for the u-th input token. Because $z_u^{(q)} \in \mathcal{Z}$, we can link $z_u^{(q)}$ with l_u by setting $z_u^{(q)} = \mathbf{e}_{l_u}, \forall u \in \{1, \cdots, U\}$, which avoids modeling $z_u^{(q)}$ and l_u independently. We additionally introduce $z_u^{(r)}$ because it allows us to derive the VQ-VAE training criteria in a theoretically sound manner [140].

Model training through direct optimization of $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:U})$ is daunting, but the

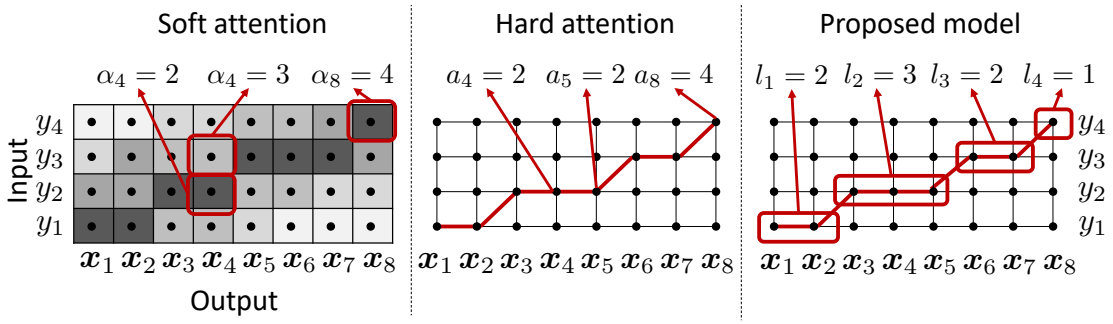


Figure 7.2: Illustration of alignment in soft-attention (left), hard-attention (middle), and proposed models (right). In soft and hard attention, alignment $\alpha_n = m$ and $a_n = m$ denote that output $\mathbf{y}_n$ is aligned with input x_m. In proposed model, $z_m = l$ indicates that x_m emits $l \in \mathbb{N}$ consecutive output steps.

above definition allows us to find a tractable variational lower bound (ELBO)[1]:

$$\log p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U})$$

$$= \log \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} p(\mathbf{x}_{1:T}, \mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{y}_{1:U}) d\mathbf{z}^{(r)}_{1:U} \tag{7.1}$$

$$\geq \mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U})} [\underbrace{\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})}_{\text{Decoder}}] \tag{7.2}$$

$$- \mathrm{KL}[Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| \underbrace{P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}_{\text{Prior}}] \tag{7.3}$$

$$- \mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U})} \Big\{ \mathrm{KL}[Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| \underbrace{p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U})}_{\text{Vector quantization}}] \Big\}. \tag{7.4}$$

$\mathbf{x}_{1:T}$ in the decoder is assumed to be independent of $\mathbf{z}^{(r)}_{1:U}$ given $\mathbf{z}^{(q)}_{1:U}$ and $\mathbf{y}_{1:U}$. Furthermore, $\mathbf{z}^{(r)}_{1:U}$ is assumed to depend only on $\mathbf{z}^{(q)}_{1:U}$. KL denotes the Kullback–Leibler divergence.

7.3.3 Model parameterization

Each term in the ELBO of Equations (7.2-7.4) can be parameterized using neural networks and calculated in a closed form.

[1]Details can be found in appendix on arXiv.

Approximate posteriors

We first define the approximate posterior for $\mathbf{z}_{1:U}^{(q)}$ as

$$
\begin{aligned}
&Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) \\
&= \prod_{u=1}^{U} P(\mathbf{z}_u^{(q)} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{I}(\mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}),
\end{aligned} \tag{7.5}
$$

where $\mathcal{I}(\cdot)$ is an indicator function, and l_u is the codeword index (and duration) for the u-th input token. We then define

$$
\begin{aligned}
&Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) \\
&= \prod_{u=1}^{U} p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}),
\end{aligned} \tag{7.6}
$$

where $\mathcal{N}$ is a multivariate Gaussian, σ^2 is a hyper-parameter, and $\mathbf{d}_u$ is computed by a neural network as $\mathbf{d}_u = \text{LatentNet}_\psi(\mathbf{z}_{u-1}^{(q)}, \bar{\mathbf{x}}_u, y_u)$.

Since $\mathbf{x}_{1:T}$ has T frames while $\mathbf{y}_{1:U}$ has U tokens, we aggregate $\mathbf{x}_{1:T}$ as $\bar{\mathbf{x}}_{1:U} = (\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_U)$ before feeding it to $\text{LatentNet}_\psi(\cdot)$. The aggregation is conducted by averaging $\mathbf{x}_t$ that correspond to the u-th token, given the value of $\mathbf{l}_{1:u}$. For example in Figure 7.2, we get $\bar{\mathbf{x}}_1 = \frac{\mathbf{x}_1+\mathbf{x}_2}{2}$ and $\bar{\mathbf{x}}_2 = \frac{\mathbf{x}_3+\mathbf{x}_4+\mathbf{x}_5}{3}$ given $l_1 = 2$ and $l_2 = 3$.

We let Q_λ be an indicator function, following the idea of VQ-VAE. The value of l_u during training is produced by another surrogate model that is explained in Section 7.3.4. A Gaussian posterior Q_ψ is inspired by the original VAE [119]. It is used to compute the KL divergence in Eq. (7.4), which is explained in Section 7.3.3.

Decoder

Similar to other models, our model uses an auto-regressive decoder. Given $\mathbf{y}_{1:U}$, $\mathbf{l}_{1:U} = (l_1, \cdots, l_U)$, and $\mathbf{z}_{1:U}^{(q)} = (\mathbf{z}_1^{(q)} = \mathbf{e}_{l_1}, \cdots, \mathbf{z}_U^{(q)} = \mathbf{e}_{l_U})$, the PDF of $\mathbf{x}_{1:T}$ is defined as

$$
\begin{aligned}
&p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) \\
&= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t) = \prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_t; \mu_t, \sigma_d^2\mathbf{I}),
\end{aligned} \tag{7.7}
$$

where $\mu_t = \text{Decoder}_\theta(\mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t)$. Note that $\widehat{\mathbf{y}}_{1:T} = \{\widehat{y}_1, \cdots, \widehat{y}_T\}$ and $\widehat{\mathbf{z}}_{1:T}^{(q)} = \{\widehat{\mathbf{z}}_1^{(q)}, \cdots, \widehat{\mathbf{z}}_T^{(q)}\}$ are upsampled from $\mathbf{y}_{1:U}$ and $\mathbf{z}_{1:U}^{(q)}$ by duplicating each y_u and $z_u^{(q)}$ for l_u times.

Prior

The prior for $\mathbf{z}_{1:U}^{(q)}$ is defined as

$$P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U}) = \prod_{u=1}^{U} P(\mathbf{z}_u^{(q)} \mid \mathbf{z}_{u-1}^{(q)}, y_u), \tag{7.8}$$

where the probability of observing $\mathbf{z}_u^{(q)} = \mathbf{e}_l$ is computed as

$$P(\mathbf{z}_u^{(q)} = \mathbf{e}_l \mid \mathbf{z}_{u-1}^{(q)}, y_u) = \frac{\exp(-\|\mathbf{c}_u - \mathbf{e}_l\|_2^2)}{\sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2)}. \tag{7.9}$$

The activation vector $\mathbf{c}_u$ is computed through a neural network as $\mathbf{c}_u = \text{LatentNet}_\phi(\mathbf{z}_{u-1}^{(q)}, y_u)$. Note that the prior calculates the probability of selecting l-th codeword $\mathbf{e}_l$ for each input token y_u, which also decides its duration $l_u = l$.

Given Eq. (7.8) and Eq. (7.5), we compute Eq. (7.3) in the ELBO as

$$\begin{aligned}
&\text{KL}[Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})] \\
&= -\log P(\mathbf{z}_{1:U}^{(q)} = (\mathbf{e}_{l_1}, \cdots, \mathbf{e}_{l_U}) \mid \mathbf{y}_{1:U}) && (7.10)\\
&= \sum_{u=1}^{U} \left\{ \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2) \right\}. && (7.11)
\end{aligned}$$

Since Q_λ is an indicator function, the value of the KL divergence is equal to the prior models' likelihood on the codeword $\{\mathbf{e}_{l_1}, \cdots, \mathbf{e}_{l_U}\}$ produced by the approximate posterior Q_λ.

Vector quantization

For the vector quantization term in Eq. (7.4), we factorize it as

$$p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) = \prod_{u=1}^{U} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)}, \sigma^2 \mathbf{I}), \tag{7.12}$$

and we assume that $\mathbf{z}_u^{(r)} = \mathbf{z}_u^{(q)} + \eta$, where $\eta \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. In other words, the quantization loss is Gaussian-distributed. The hyper-parameter σ is the same as that in Section 7.3.3.

Given Eq. (7.12) and Eq. (7.6), we compute Eq. (7.4) in the ELBO as

$$\begin{aligned}&\mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)}|\mathbf{x}_{1:T},\mathbf{y}_{1:U})}\Big\{\mathrm{KL}\big[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U})\big]\Big\} \\ &= \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2. \end{aligned} \tag{7.13}$$

Notice that Q_λ is an indicator function, and the KL divergence between two Gaussian distributions has a closed form.

7.3.4 CTC-based alignment search

In Section 7.3.3, we define the approximate posterior $Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$ as an indicator function. This corresponds to the ideal case where, given $\mathbf{x}_{1:T}$ and $\mathbf{y}_{1:U}$, Q_λ can perfectly infer the value of $\mathbf{l}_{1:U}$ and assign probability of zero to other values. In practice, we use a CTC-based recognition model $P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T})$ to produce $\mathbf{l}_{1:U}$.

This is done by first searching the optimal alignment $\mathbf{a}_{1:T}^*$ from the CTC trellis, i.e., $\mathbf{a}^*{}_{1:T} = \arg\max_{\mathbf{a}_{1:T}} P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T})$. The alignment variable a_t is similar to that in hard-attention in Figure 7.2. We then directly convert $\mathbf{a}_{1:T}^*$ into $l_{1:U}$ as the example on right side of Figure 7.2 shows.

The sampled $l_{1:U}$ must satisfies $T = \sum_u l_u$ and $1 \leq l_u \leq K$. This can be achieved by excluding the invalid alignment path from the CTC trellis. This is detailed in appendix.

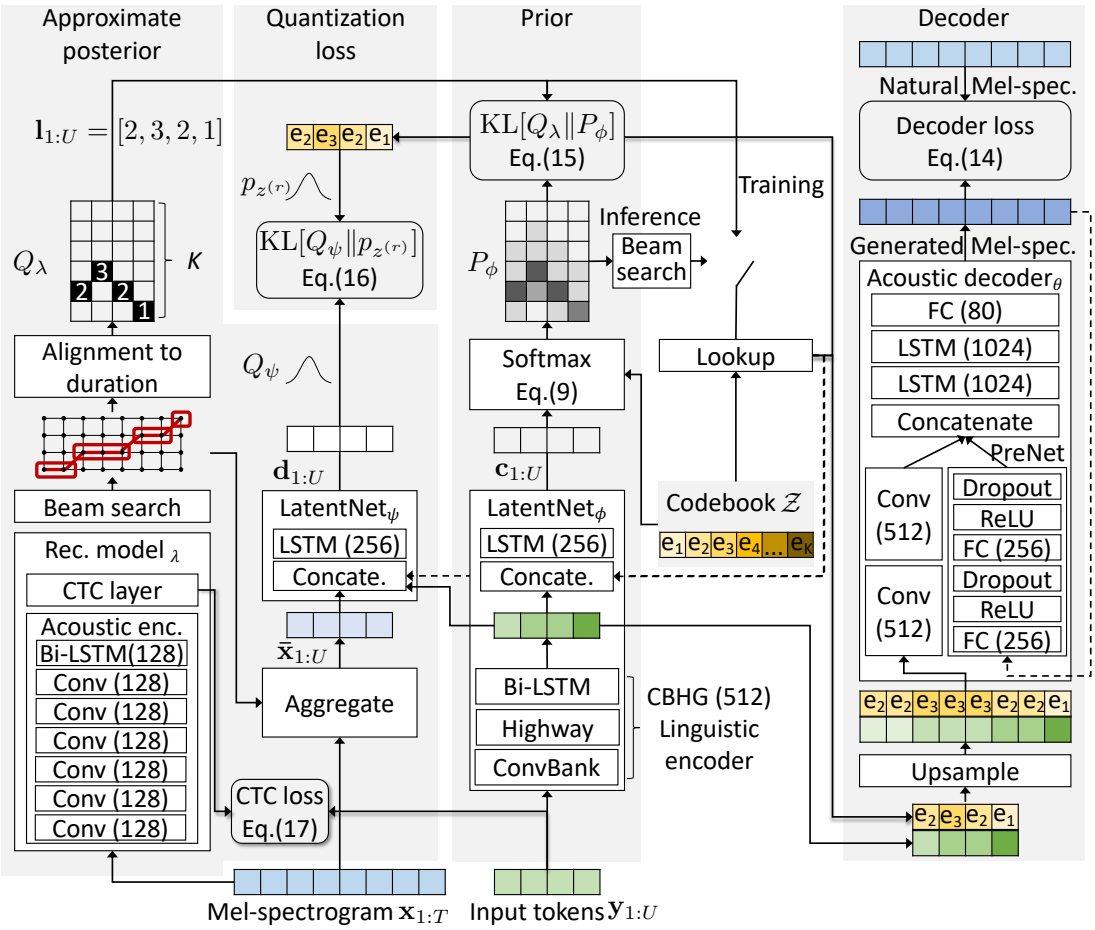


Figure 7.3: Proposed TTS system. Dashed line denotes feedback loop. Number in bracket denotes neural layer size. FC denotes a fully connected layer. During inference, only prior and decoder are used.

7.3.5 Training criterion in summary

With all the components as explained in Section 7.3.3 and 7.3.4, we summarize the training criterion of the proposed model as

$$
\begin{aligned}
&\mathcal{L}(\theta, \psi, \phi, \lambda, \mathcal{Z}) \\
=&\mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U})}[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})] && (7.14)\\
&- \sum_{u=1}^{U} \left\{ \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2) \right\} && (7.15)\\
&- \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2 && (7.16)\\
&+ \gamma \log \sum_{\forall \mathbf{a}_{1:T}} P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}), && (7.17)
\end{aligned}
$$

Equations (7.14 - 7.16) correspond to the ELBO in Section 7.3.2, and the last line is for the CTC model. The vectors $\mathbf{d}_u$ and $\mathbf{c}_u$ are computed as $\mathbf{d}_u = \text{LatentNet}_\psi(\mathbf{z}^{(q)}_{u-1}, \bar{\mathbf{x}}_u, y_u)$ and $\mathbf{c}_u = \text{LatentNet}_\phi(\mathbf{z}^{(q)}_{u-1}, y_u)$.

The decoder parameter θ is trained with Eq. (7.14); the LatentNet_ϕ in the prior is trained by Equation (7.15); the LatentNet_ψ in the approximate posterior is trained with Eq. (7.16). The codebook $\mathcal{Z} = \{\mathbf{e}_1, \cdots, \mathbf{e}_K\}$ is trained with Eqs. (7.15 -7.16). All the components are jointly trained. During inference, only the prior and decoder network are used.

7.3.6 Implementation

The implemented TTS system is illustrated in Figure 7.3. The acoustic encoder in the CTC model consists of convolution layers followed by a bidirectional LSTM, which is similar to that in [125]. The LatentNet_ϕ in the prior consists of a CBHG-based linguistic encoder [1] and an additional LSTM layer. LatentNet_ψ has only one LSTM layer. The decoder is similar to that in [1], which consists of pre-net bottleneck layers, convolutional layers, LSTM, and linear output layers. The decoder receives input from the output of the linguistic encoder.

7.4 Experiments

7.4.1 Experimental condition

In these experiments, we treated duration in three frames as a unit. This grouping allowed us to reduce the codebook size K to 13. Accordingly, the value of the latent duration can be $l_u = k$, where $k \in \{3, 6, \cdots, 39\}$. Each codebook vector e_k has 32 dimensions.

We configured $\sigma_d = 3.0$ in Eq. (7.7), and $\sigma = 0.4$ in Eq. (7.13). Square distance in Eq. (7.11) and (7.13) were decomposed into $f(a, b) = \alpha\|a - \text{sg}[b]\|_2^2 + \beta\|\text{sg}[a] - b\|_2^2$, where $\text{sg}[\cdot]$ is a stop gradient operator. We configured $\alpha = 1.0, \beta = 0.0$ for Eq. (7.11), and $\alpha = 2.0, \beta = 1.0$ for Eq. (7.13). We set $\gamma = 0.5$ for CTC loss in Eq. (7.17). We used Adam optimizer [111] with fixed learning rate of 5×10^{-5} to optimize the objective. Durations were sampled with beam width of 3 during training and of 10 during inference.

We used LJSpeech[2], an English speech corpus containing about 24 hour recordings from a female speaker. We used 12,600 utterances for training, 250 for validation, and 250 for testing.

We built three versions of the proposed systems that use character or phoneme as the input tokens $\mathbf{y}_{1:U}$:

- CC uses characters as input tokens;
- PP uses phonemes as input tokens;
- CP uses characters as input of the linguistic encoder and character-aligned phonemes as CTC target.

The phoneme labels are obtained with a G2P module[3]. All systems uses 80-dimensional Mel-spectrogram. Waveforms were generated with WaveNet [12][4].

We conducted a listening test to evaluate the performance of our proposed TTS system. We included nine systems in the listening test: natural samples

[2]https://keithito.com/LJ-Speech-Dataset/

[3]https://github.com/Kyubyong/g2p

[4]For fair comparison with TTS systems from ESPNet, we used a public WaveNet called ljspeech.wavenet.mol.v1 provided by ESPNet.

(`Natural`), analysis-by-synthesis (ABS), Transformer-TTS [39], FastSpeech [63], and two Tacotron2 models [22] as reference systems, and the three proposed systems[5]. The reference systems are public models built by the ESPNet-TTS team [153], and we used these specific models: `transformer.v3`, `fastspeech.v3`, `tacotron2.v2`, and `tacotron2.v3`. We selected these reference systems in order to compare our proposed system with TTS systems that use different duration modeling approaches: FastSpeech uses an external duration model, Tacotron-based systems uses soft-attention, and Transformer-TTS uses soft-attention with positional encoding.

We recruited 493 Japanese listeners with crowdsourcing. In each listening set, one listener was asked to evaluate the quality of 36 audio samples in five grade mean-opinion-score (MOS) scales. We collected 17,856 evaluation scores in total.

7.4.2 Experimental results

Figure 7.4 shows the results of listening test. Our proposed systems got 2.99 ± 0.04 for character based system (CC), 3.04 ± 0.04 for phoneme based system (PP), and 3.16 ± 0.04 for character and phoneme combination system (CP). For reference systems, `transformer.v3`, `tacotron2.v3`, `tacotron2.v2`, and `fastspeech.v3` got 4.03 ± 0.03, 3.77 ± 0.03, 3.71 ± 0.03, 2.66 ± 0.04, respectively.

Overall, our proposed systems were rated between soft-attention-based systems (Transformer-TTS and Tacotron2) and FastSpeech that uses an external duration model instead of soft attention. Among the proposed systems, the character and phoneme combination system (CP) showed the best score. We observed that phoneme labels estimated by the G2P module contained errors, which resulted in mispronunciations of the synthesized speech in the PP condition.

Although our proposed systems were rated worse than the soft-attention-based systems, we argue that this is not surprising. This is because there was a strong assumption in our model. More specifically, the current framework automatically estimates duration of given input symbols, but it does not insert any short pauses between the symbols. It does not determine duration of the short pauses, either. In other words, we constrain that the sum of duration of each input character is equal to speech duration, that is, $T = \sum_u l_u$, this constraint does not consider the duration

[5] Audio samples are available at https://nii-yamagishilab.github.io/sample-tts-latent-duration/

of any short pauses, which may be inserted into any phrase boundaries. Obviously this makes synthesized speech unnatural perceptually since synthetic speech of the proposed system have neither short pauses nor phrase breaks. Our model needs to have a more appropriate constraint where we exclude total duration of short pauses and an additional mechanism to insert short pauses at appropriate phrase boundaries. This is our next step.

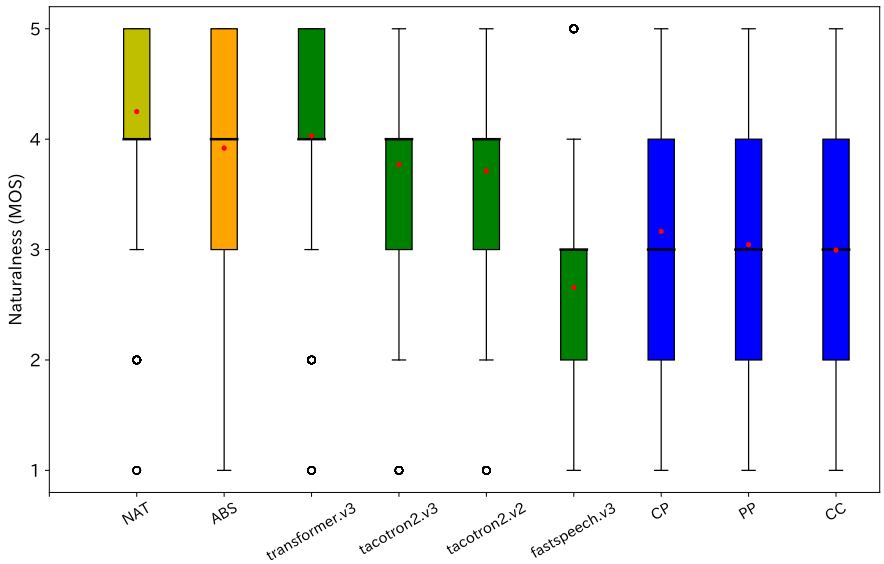


Figure 7.4: Result of listening test. Red dots denote mean MOS values.

We detected character error rates (CER) from synthetic speech with automatic speech recognition [6] to analyse the results of the subjective evaluation. Table 7.1 shows CER from each systems. Our systems show larger CER compared with baseline systems, indicating more pronunciation errors. The pronunciation errors were mitigated by using characters instead of phonemes. Interestingly, Fastspeech had the least CER although was rated at the bottom in MOS.

7.5 Conclusion

We proposed an end-to-end TTS method using phoneme duration as a latent variable. We construct our method based on VQ-VAE, representing phoneme duration in

[6] We used a public model based on Transformer trained with LibriSpeech provided by ESPNet.

Table 7.1: Character error rates of synthetic speech detected with automatic speech recognition.

System	CER (%)
CP (Proposed)	5.76
CC (Proposed)	6.46
PP (Proposed)	6.53
`transformer.v3`	3.47
`tacotron2.v3`	2.78
`tacotron2.v2`	2.49
`fastspeech.v3`	2.49

discrete symbols as the number of acoustic frames. We incorporated a forced aligner as approximate posterior, and phoneme duration predictor as prior in VAE. All the components are jointly trainable from scratch in end-to-end manner by maximizing a theoretically derived variational lower bound. Our method of latent duration was almost same as what we used for latent pitch accent, which indicates generalizability of the TTS framework based on conditional VQ-VAE.

We experimentally compared three versions of our proposed TTS method with other sequence-to-sequence TTS systems, including FastSpeech, Tacotron2, and Transformer-TTS, and found that our systems had better naturalness than FastSpeech but were rated lower than Transformer-TTS and Tacotron2. We infer that it is challenging to predict phoneme duration from texts or phonemes without pause label. We presume that the inferior naturalness was caused by the lack of appropriate modeling of short pauses.

8 Conclusion

8.1 Addressed issues

In Chapter 1, we listed issues to be addressed in this thesis from the perspective of pitch accents and phoneme duration. For pitch accents, we worked on the following issues:

1) Methods to treat pitch accents in tonal or pitch accent languages are not established in end-to-end TTS.

2) Pitch accent prediction from text is separately handled and is not incorporated as a part of end-to-end TTS.

For the first issue about pitch accent languages, we showed that accentual type labels could be used as an additional input for end-to-end TTS to produce correct pitch accents. Furthermore, our experiment showed that, in contrast to full-context labels, just phoneme and accentual types are enough information for linguistic features of Japanese end-to-end TTS if we increase model parameter size sufficiently.

For the second issue about pitch accent prediction, we incorporated pitch accent modeling in end-to-end TTS based on conditional VQ-VAE in Chapter 5. We showed that abstract tone contour labels can be used to represent pitch accents in latent variables, and the prior in VAE can work as a tone predictor. Our proposed method achieved moderately good tone error rate and naturalness of synthetic speech without relying on pitch accent labels during prediction.

For phoneme duration, we tackled the following two issues:

1) Alignments modeled with the current scheme of end-to-end TTS are unstable and error-prone.

2) Alignment is not represented with phoneme duration in end-to-end TTS, which is a more natural, robust, and efficient form for TTS.

For the first issue, our approach to improve alignment stability was to use hard monotonic alignment instead of soft attention. We proposed SSTN-TTS in Chapter 6 which used hard latent alignments, and our experiment showed it could avoid fatal alignment errors.

For the second issue, we proposed an end-to-end TTS method using latent phoneme duration based on conditional VQ-VAE in Chapter 7. Our method incorporated the forced aligner and duration model used in DNN-based pipeline frameworks to end-to-end TTS to handle duration as a latent variable. Our method could realize hard monotonic alignment more efficiently and eliminate overestimation of duration alignment errors observed in SSNT-TTS.

8.2 Remaining issues

In Chapter 3, our experiment for English revealed that an end-to-end TTS system showed flat pitch compared to pipeline TTS system. This thesis does not handle methods to improve the pitch accent in end-to-end TTS for English. One promising method is pitch accent modeling proposed in Chapter 5. We investigated the pitch accent modeling for Japanese and Chinese with tone contour labels, but it also allowed unsupervised learning of latent variables as shown in Chapter 5. We can use semi-supervised learning for English, for which tone contour labels are hard to annotate. To

enforce the latent space to capture pitch accents, we may use mixed language corpus as we did in the experiment in Chapter 5.

In Chapter 5, we introduced pitch accent modeling to end-to-end TTS based on conditional VQ-VAE. We achieved moderately accurate pitch accents for Japanese TTS without relying on pitch accent labels during prediction. The achieved naturalness of synthetic speech however did not reach the level we achieved by using accentual type labels and large model size in Chapter 4. We think this is because pitch accent can not be resolved from phoneme sequences only, without any surface form information. The next step of Japanese TTS is end-to-end TTS from text inputs including logographic characters. It is interesting how much tone error rate can be improved by using character inputs once methods for Japanese end-to-end TTS using characters is established.

In Chapter 7, we proposed an end-to-end TTS method using latent phoneme duration based on VQ-VAE. This method could solve the problem of overestimation of duration observed in the SSNT-TTS method which is based on frame-level hard alignment in the experiment in Chapter 6. The TTS method using latent duration however did not show high enough naturalness of synthetic speech expected from end-to-end TTS. We think this is because the formulation of our method is less capable of automatic labels which do not represent exact phonemes of corresponding speech. We used character sequence and phoneme sequence without pause labels derived with G2P in the experiment. These kinds of labels are frequently used in end-to-end TTS, so we have to handle these label characteristics in our method in the future.

VAE shines when it is used for unsupervised representation learning, although we used it with supervision on latent variables in Chapters 5 and 7. Semi-supervised learning is promising for pitch accent and duration modeling. Our proposed methods is advantageous because it can utilize speech corpora which is partially labeled. Pitch accents are especially tedious to label so semi-supervised learning of pitch accents is a realistic scenario. Applying our method to unsupervised settings is our next step.

Furthermore, speaking style control is a huge application of VAE in TTS. We think speaking style control is a good application of our methods for pitch accent modeling and duration modeling investigated in Chapters 5 and 7. Pitch accent control can be used for dialect transfer. Duration control can manipulate speaking rate. Speaking style control based on pitch accents and phoneme duration is also our future work.

8.3 Final remark

End-to-end TTS is an approach continuing to evolve in the time of this writing. This approach is promising in terms of language versatility. Expanding the applicability of end-to-end TTS for many languages is still on its way and will continue. This thesis contributed to this research direction by expanding the applicability of end-to-end TTS to pitch accent languages. In addition, we improved alignment stability of end-to-end TTS by using the same method for pitch accent modeling. Our method based on conditional VQ-VAE has potential applicability to lexical latent information in speech other than pitch accent and phoneme duration. Unifying lexical latent information modeling based on our approach may advance the applicability of end-to-end TTS for many languages further.

9
List of publications

- Yusuke Yasuda, Xin Wang, Junichi Yamagishi, “Investigation of learning abilities on linguistic features in sequence-to-sequence text-to-speech synthesis,” Computer Speech and Language, Volume 67, 2021, 101183, doi: 10.1016/j.csl.2020.101183.
- Yusuke Yasuda, Xin Wang, Junichi Yamagishi, “End-to-End Text-to-Speech using Latent Duration based on VQ-VAE”. Proc. ICASSP 2021
- Y. Yasuda, X. Wang and J. Yamagishi, "Effect of Choice of Probability Distribution, Randomness, and Search Methods for Alignment Modeling in Sequence-to-Sequence Text-to-Speech Synthesis Using Hard Alignment," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 6724-6728, doi: 10.1109/ICASSP40776.2020.9053546.
- Y. Yasuda, X. Wang, J. Yamagishi, “Initial investigation of encoder-decoder end-to-end TTS using marginalization of monotonic hard alignments”. Proc. 10th ISCA Speech Synthesis Workshop, 211-216, 2019, doi: 10.21437/SSW.2019-38.

- Y. Yasuda, X. Wang, S. Takaki and J. Yamagishi, "Investigation of Enhanced Tacotron Text-to-speech Synthesis Systems with Self-attention for Pitch Accent Language," ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, United Kingdom, 2019, pp. 6905-6909, doi: 10.1109/ICASSP.2019.8682353.

Bibliography

[1] Yuxuan Wang, R.J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. In *Proc. Interspeech*, pages 4006–4010, 2017.

[2] Yusuke Yasuda, Xin Wang, Shinji Takaki, and Junichi Yamagishi. Investigation of enhanced tacotron text-to-speech synthesis systems with self-attention for pitch accent language. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*, pages 6905–6909. IEEE, 2019.

[3] Schultz & Kirchhoff. *Multilingual Speech Processing*. Elsevier, 2006.

[4] Heiga Zen, Keiichi Tokuda, and Alan W Black. Statistical parametric speech synthesis. *Speech Communication*, 51:1039–1064, 2009.

[5] Heiga Zen, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura. Hidden semi-markov model based speech synthesis. In *INTERSPEECH 2004 - ICSLP, 8th International Conference on Spoken Language Processing, Jeju Island, Korea, October 4-8, 2004*. ISCA, 2004.

[6] Hideki Kawahara, Ikuyo Masuda-Katsuse, and Alain de Cheveigne. Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds. *Speech Communication*, 27:187–207, 1999.

[7] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. WORLD: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Trans. on Information and Systems*, 99(7):1877–1884, 2016.

[8] Akira Tamamori, Tomoki Hayashi, Kazuhiro Kobayashi, Kazuya Takeda, and Tomoki Toda. Speaker-dependent WaveNet vocoder. In *Proc. Interspeech*, pages 1118–1122, 2017.

[9] Heiga Zen, Alan Senior, and Martin Schuster. Statistical parametric speech synthesis using deep neural networks. In *Proc. ICASSP*, pages 7962–7966, 2013.

[10] Oliver Watts, Gustav Eje Henter, Thomas Merritt, Zhizheng Wu, and Simon King. From HMMs to DNNs: where do the improvements come from? In *Proc. ICASSP*, pages 5505–5509, 2016.

[11] Keiichi Tokuda, Kei Hashimoto, Keiichiro Oura, and Yoshihiko Nankaku. Temporal modeling in neural network based statistical parametric speech synthesis. In *9th ISCA Speech Synthesis Workshop*, pages 106–111, 2016.

[12] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[13] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. SampleRNN: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[14] Shinji Takaki, Hirokazu Kameoka, and Junichi Yamagishi. Direct modeling of frequency spectra and waveform generation based on phase recovery for DNN-based speech synthesis. In *Proc. Interspeech*, pages 1128–1132, 2017.

[15] Yuki Saito, Shinnosuke Takamichi, and Hiroshi Saruwatari. Vocoder-free text-to-speech synthesis incorporating generative adversarial networks using low-/multi-frequency stft amplitude spectra. *Computer Speech & Language*, 58:347 – 363, 2019.

[16] Hieu-Thi Luong, Xin Wang, Junichi Yamagishi, and Nobuyuki Nishizawa. Investigating accuracy of pitch-accent annotations in neural-network-based speech synthesis and denoising effects. In *Proc. Interspeech*, pages 37–41, 2018.

[17] Jaime Lorenzo-Trueba, Fuming Fang, Xin Wang, Isao Echizen, Junichi Yamagishi, and Tomi Kinnunen. Can we steal your vocal identity from the internet?: Initial investigation of cloning obama' s voice using gan, wavenet and low-quality found data. In *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*, pages 240–247, 2018.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, 2014.

[19] Wenfu Wang, Shuang Xu, and Bo Xu. First step towards end-to-end parametric tts synthesis: Generating spectral parameters with neural attention. In *Interspeech 2016*, pages 2243–2247, 2016.

[20] Yaniv Taigman, Lior Wolf, Adam Polyak, and Eliya Nachmani. VoiceLoop: Voice fitting and synthesis via a phonological loop. In *Proc. ICLR*, 2018.

[21] Jose Sotelo, Soroush Mehri, Kundan Kumar, João Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. In *Proc. ICLR (Workshop Track)*, 2017.

[22] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ-Skerrv Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu. Natural TTS synthesis by conditioning WaveNet on Mel spectrogram predictions. In *Proc. ICASSP*, pages 4779–4783, 2018.

[23] Y. Zheng, J. Tao, Z. Wen, and J. Yi. Forward–backward decoding sequence for regularizing end-to-end tts. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2067–2079, Dec 2019.

[24] Liumeng Xue, Wei Song, Guanghui Xu, Lei Xie, and Zhizheng Wu. Building a Mixed-Lingual Neural TTS System with Only Monolingual Data. In *Proc. Interspeech 2019*, pages 2060–2064, 2019.

[25] Jing-Xuan Zhang, Zhen-Hua Ling, and Li-Rong Dai. Forward attention in sequence-to-sequence acoustic modeling for speech synthesis. In *Proc. ICASSP*, pages 4789–4793. IEEE, 2018.

[26] Kyubyong Park and Thomas Mulc. CSS10: A Collection of Single Speaker Speech Datasets for 10 Languages. In *Proc. Interspeech 2019*, pages 1566–1570, 2019.

[27] Younggun Lee and Taesu Kim. Learning pronunciation from a foreign language in speech synthesis networks. *CoRR*, abs/1811.09364, 2018.

[28] Yu Zhang, Ron J. Weiss, Heiga Zen, Yonghui Wu, Zhifeng Chen, R.J. Skerry-Ryan, Ye Jia, Andrew Rosenberg, and Bhuvana Ramabhadran. Learning to Speak Fluently in a Foreign Language: Multilingual Speech Synthesis and Cross-Language Voice Cloning. In *Proc. Interspeech 2019*, pages 2080–2084, 2019.

[29] Jingbei Li, Zhiyong Wu, Runnan Li, Pengpeng Zhi, Song Yang, and Helen Meng. Knowledge-Based Linguistic Encoding for End-to-End Mandarin Text-to-Speech Synthesis. In *Proc. Interspeech 2019*, pages 4494–4498, 2019.

[30] Y. Lu, M. Dong, and Y. Chen. Implementing prosodic phrasing in chinese end-to-end speech synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7050–7054, May 2019.

[31] Rui Liu, Berrak Sisman, Jingdong Li, Feilong Bao, Guanglai Gao, and Haizhou Li. Teacher-student training for robust tacotron-based TTS. *CoRR*, abs/1911.02839, 2019.

[32] Dan Lim, Won Jang, Gyeonghwan O, Hyeyeong Park, Bongwan Kim, and Jesam Yoon. JDI-T: jointly trained duration informed transformer for text-to-speech without explicit alignment. *CoRR*, abs/2005.07799, 2020.

[33] Takato Fujimoto, Kei Hashimoto, Keiichiro Oura, Yoshihiko Nankaku, and Keiichi Tokuda. Impacts of input linguistic feature representation on Japanese end-to-end speech synthesis. In *Proc. 10th ISCA Speech Synthesis Workshop*, pages 166–171, 2019.

[34] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan Ömer Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep Voice 3: Scaling text-to-speech with convolutional sequence learning. In *Proc. ICLR*, 2018.

[35] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, Ming Liu, and Ming Zhou. Close to human quality TTS with transformer. *CoRR*, abs/1809.08895, 2018.

[36] Wei Ping, Kainan Peng, and Jitong Chen. Clarinet: Parallel wave generation in end-to-end text-to-speech. *CoRR*, abs/1807.07281, 2018.

[37] Ron J. Weiss, R. J. Skerry-Ryan, Eric Battenberg, Soroosh Mariooryad, and Diederik P. Kingma. Wave-tacotron: Spectrogram-free end-to-end text-to-speech synthesis. *CoRR*, abs/2011.03568, 2020.

[38] H. Tachibana, K. Uenoyama, and S. Aihara. Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4784–4788, April 2018.

[39] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 6706–6713. AAAI Press, 2019.

[40] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[41] Oliver Samuel Watts. *Unsupervised learning for text-to-speech synthesis*. PhD thesis, University of Edinburgh, 2013.

[42] Heng Lu, Simon King, and Oliver Watts. Combining a vector space representation of linguistic context with a deep neural network for text-to-speech synthesis. In *Proc. 8th ISCA Speech Synthesis Workshop*, pages 261–265, 2013.

[43] Xin Wang, Shinji Takaki, and Junichi Yamagishi. Autoregressive neural f0 model for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(8):1406–1419, Aug 2018.

[44] L. Chen, T. Raitio, C. Valentini-Botinhao, Z. Ling, and J. Yamagishi. A deep generative architecture for postfiltering in statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):2003–2014, Nov 2015.

[45] Takuhiro Kaneko, Hirokazu Kameoka, Nobukatsu Hojo, Yusuke Ijima, Kaoru Hiramatsu, and Kunio Kashino. Generative adversarial network-based postfilter for statistical parametric speech synthesis. In *Proc. ICASSP*, pages 4910–4914, 2017.

[46] Oliver Watts, Gustav Eje Henter, Jason Fong, and Cassia Valentini-Botinhao. Where do the improvements come from in sequence-to-sequence neural TTS? In *Proc. 10th ISCA Speech Synthesis Workshop*, pages 217–222, 2019.

[47] Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J. Weiss, Ye Jia, Zhifeng Chen, and Yonghui Wu. LibriTTS: A Corpus Derived from LibriSpeech for Text-to-Speech. In *Proc. Interspeech 2019*, pages 1526–1530, 2019.

[48] Jason Taylor and Korin Richmond. Analysis of Pronunciation Learning in End-to-End Speech Synthesis. In *Proc. Interspeech 2019*, pages 2070–2074, 2019.

[49] Jason Fong, Jason Taylor, Korin Richmond, and Simon King. A Comparison of Letters and Phones as Input to Sequence-to-Sequence Models for Speech Synthesis. In *Proc. 10th ISCA Speech Synthesis Workshop*, pages 223–227, 2019.

[50] Kainan Peng, Wei Ping, Zhao Song, and Kexin Zhao. Parallel neural text-to-speech. *CoRR*, abs/1905.08459, 2019.

[51] Kyle Kastner, João Felipe Santos, Yoshua Bengio, and Aaron C. Courville. Representation mixing for TTS synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*, pages 5906–5910. IEEE, 2019.

[52] B. Li, Y. Zhang, T. Sainath, Y. Wu, and W. Chan. Bytes are all you need: End-to-end multilingual speech recognition and synthesis with bytes. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5621–5625, May 2019.

[53] Shan Liu Qiao Tian, Jing Chen. The tencent speech synthesis system for blizzard challenge 2019. In *Proc. Blizzard Challenge Workshop*, 2019.

[54] Takuma Okamoto, Tomoki Toda, Yoshinori Shiga, and Hisashi Kawai. Real-Time Neural Text-to-Speech with Sequence-to-Sequence Acoustic Model and WaveGlow or Single Gaussian WaveRNN Vocoders. In *Proc. Interspeech 2019*, pages 1308–1312, 2019.

[55] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[56] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[57] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proc. EMNLP*, pages 1412–1421, 2015.

[58] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 577–585, 2015.

[59] Mutian He, Yan Deng, and Lei He. Robust Sequence-to-Sequence Acoustic Modeling with Stepwise Monotonic Attention for Neural TTS. In *Proc. Interspeech 2019*, pages 1293–1297, 2019.

[60] Sean Vasquez and Mike Lewis. Melnet: A generative model for audio in the frequency domain. *CoRR*, abs/1906.01083, 2019.

[61] Eric Battenberg, R. J. Skerry-Ryan, Soroosh Mariooryad, Daisy Stanton, David Kao, Matt Shannon, and Tom Bagby. Location-relative attention mechanisms for robust long-form speech synthesis. *CoRR*, abs/1910.10288, 2019.

[62] X. Zhu, Y. Zhang, S. Yang, L. Xue, and L. Xie. Pre-alignment guided attention for improving training efficiency and model stability in end-to-end speech synthesis. *IEEE Access*, 7:65955–65964, 2019.

[63] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech: Fast, robust and controllable text to speech. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3165–3174. Curran Associates, Inc., 2019.

[64] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[65] Haohan Guo, Frank K. Soong, Lei He, and Lei Xie. A New GAN-Based End-to-End TTS Training Algorithm. In *Proc. Interspeech 2019*, pages 1288–1292, 2019.

[66] Alan W Black, Kevin Lenzo, and Vincent Pagel. Issues in building general letter to sound rules. In *Proc. SSW3*, pages 77–80, 1998.

[67] Paul Taylor and Alan W Black. Assigning phrase breaks from part-of-speech sequences. *Computer Speech & Language*, 12(2):99–117, 1998.

[68] Julia Hirschberg. Pitch accent in context predicting intonational prominence from text. *Artificial Intelligence*, 63(1):305–340, 1993.

[69] K. Oura, Y. Nankaku, T. Toda, K. Tokuda, R. Maia, S. Sakai, and S. Nakamura. Simultaneous acoustic, prosodic, and phrasing model training for tts conversion systems. In *2008 6th International Symposium on Chinese Spoken Language Processing*, pages 1–4, Dec 2008.

[70] Xin Wang, Shinji Takaki, and Junichi Yamagishi. An autoregressive recurrent mixture density network for parametric speech synthesis. In *Proc. ICASSP*, pages 4895–4899, 2017.

[71] Benigno Uria, Iain Murray, Steve Renals, Cassia Valentini-Botinhao, and John Bridle. Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. In *Proc. ICASSP*, pages 4465–4469, 2015.

[72] Kei Hashimoto, Keiichiro Oura, Yoshihiko Nankaku, and Keiichi Tokuda. Trajectory training considering global variance for speech synthesis based on neural networks. In *Proc. ICASSP*, pages 5600–5604, 2016.

[73] Daniel Griffin and Jae Lim. Signal estimation from modified short-time Fourier transform. *IEEE Trans. ASSP*, 32(2):236–243, 1984.

[74] Xin Wang, Jaime Lorenzo-Trueba, Shinji Takaki, Lauri Juvela, and Junichi Yamagishi. A comparison of recent waveform generation and acoustic modeling methods for neural-network-based speech synthesis. In *Proc. ICASSP*, pages 4804–4808, 2018.

[75] Lauri Juvela, Bajibabu Bollepalli, Junichi Yamagishi, and Paavo Alku. GELP: GAN-Excited Linear Prediction for Speech Synthesis from Mel-spectrogram. *Proc. Interspeech*, pages 694–699, 2019.

[76] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. WaveGlow: A Flow-based Generative Network for Speech Synthesis. In *Proc. ICASSP*, pages 3617–3621, 2019.

[77] Jean-Marc Valin and Jan Skoglund. LPCNet: Improving Neural Speech Synthesis Through Linear Prediction. In *Proc. ICASSP*, pages 5891–5895, 2019.

[78] Xin Wang and Junichi Yamagishi. Neural Harmonic-plus-Noise Waveform Model with Trainable Maximum Voice Frequency for Text-to-Speech Synthesis. In *Proc. SSW*, pages 1–6, 2019.

[79] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech 2: Fast and high-quality end-to-end text to speech. *CoRR*, abs/2006.04558, 2020.

[80] Jonathan Shen, Ye Jia, Mike Chrzanowski, Yu Zhang, Isaac Elias, Heiga Zen, and Yonghui Wu. Non-attentive tacotron: Robust and controllable neural TTS synthesis including unsupervised duration modeling. *CoRR*, abs/2010.04301, 2020.

[81] Stanislav Beliaev, Yurii Rebryk, and Boris Ginsburg. Talknet: Fully-convolutional non-autoregressive speech synthesis model, 2020.

[82] Zhen Zeng, Jianzong Wang, Ning Cheng, Tian Xia, and Jing Xiao. AlignTTS: Efficient feed-forward text-to-speech system without explicit alignment. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 6714–6718. IEEE, 2020.

[83] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[84] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[85] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[86] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *Proc. International Conference on Machine Learning: Deep Learning Workshop*, 2015.

[87] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville,

and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *Proc. ICLR*, 2017.

[88] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45(11):2673–2681, 1997.

[89] Yuxuan Wang, Daisy Stanton, Yu Zhang, R. J. Skerry-Ryan, Eric Battenberg, Joel Shor, Ying Xiao, Ye Jia, Fei Ren, and Rif A. Saurous. Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5167–5176. PMLR, 2018.

[90] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *Proc. ICLR*, 2017.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. NIPS*, pages 6000–6010, 2017.

[92] Barret Zoph and Kevin Knight. Multi-source neural translation. In *Proc. NAACL-HLT*, pages 30–34, 2016.

[93] Xavi Gonzalvo, Siamak Tazari, Chun-an Chan, Markus Becker, Alexander Gutkin, and Hanna Silén. Recent advances in google real-time hmm-driven unit selection synthesizer. In Nelson Morgan, editor, *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2238–2242. ISCA, 2016.

[94] Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemyslaw Szczepaniak. Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices. In *Proc. Interspeech*, pages 2273–2277, 2016.

[95] Andrew Gibiansky, Sercan Ömer Arik, Gregory Frederick Diamos, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2962–2970, 2017.

[96] Zhizheng Wu et al. The blizzard challenge 2019. In *The Blizzard Challenge workshop*, 2019.

[97] Zhizheng Wu, Oliver Watts, and Simon King. Merlin: An open source neural network speech synthesis system. In *9th ISCA Speech Synthesis Workshop*, pages 202–207, 2016.

[98] Yuan Jiang, Ya-Jun Hu, Li-Juan Liu, Hong-Chuan Wu, Zhi-Kun Wang, Yang Ai, Zhen-Hua Ling, and Li-Rong Dai. The USTC system for blizzard challenge 2019. In *Proc. Blizzard Challenge Workshop*, 2019.

[99] Mike Schuster. Better generative models for sequential data problems: Bidirectional recurrent mixture density networks. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 589–595. The MIT Press, 1999.

[100] Brian C J Moore. *An Introduction to the Psychology of Hearing.* Brill, 6th edition, 2012.

[101] Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. Character-aware neural language models. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2741–2749. AAAI Press, 2016.

[102] Simon King and Vasilis Karaiskos. The blizzard challenge 2011. In *The Blizzard Challenge workshop*, 2011.

[103] Alan W. Black and Kevin A Lenzo. Flite: a small fast run-time synthesis engine. In *SSW4-2001*, page 204, 2001.

[104] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[105] Y. Sagisaka and H. Sato. Accentuation rules for japanese word concatenation. *IEICE Transactions on Information and Systems*, J66-D(7):849–856, 1983.

[106] Masayuki SUZUKI, Ryo KUROIWA, Keisuke INNAMI, Shumpei KOBAYASHI, Shinya SHIMIZU, Nobuaki MINEMATSU, and Keikichi HIROSE. Accent sandhi estimation of tokyo dialect of japanese using conditional random fields. *IEICE Transactions on Information and Systems*, E100.D(4):655–661, 2017.

[107] Minematsu Nobuaki, Kuroiwa Ryo, Hirose Keikichi, and Watanabe Michiko. CRF-based statistical learning of Japanese accent sandhi for developing Japanese text-to-speech synthesis systems. In *Proc. 6th ISCA Speech Synthesis Workshop*, pages 148–153, 2007.

[108] The HTS Working Group. The Japanese TTS System 'Open JTalk', 2015.

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

[110] H Kawai, T Toda, J Yamagishi, T Hirai, J Ni, N Nishizawa, M Tsuzaki, and K Tokuda. Ximera: A concatenative speech synthesis system with large scale corpora. *IEICE Transactions on Information and System (Japanese Edition)*, pages 2688–2698, 2006.

[111] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2014.

[112] Xin Wang. *Fundamental Frequency Modeling for Neural-Network-Based Statistical Parametric Speech Synthesis.* PhD thesis, Department of Informatics, SOKENDAI, 2018.

[113] Kenneth N. Ross and Mari Ostendorf. Prediction of abstract prosodic labels for speech synthesis. *Comput. Speech Lang.*, 10(3):155–185, 1996.

[114] Tohru Nagano, Shinsuke Mori, and Masafumi Nishimura. A stochastic approach to phoneme and accent estimation. In *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*, pages 3293–3296. ISCA, 2005.

[115] Kai Yu and Steve J. Young. Continuous F0 modeling for HMM based statistical parametric speech synthesis. *IEEE Trans. Speech Audio Process.*, 19(5):1071–1079, 2011.

[116] Adrian Lancucki. Fastpitch: Parallel text-to-speech with pitch prediction. *CoRR*, abs/2006.06873, 2020.

[117] Li Aijun. Chinese prosody and prosodic labeling of spontaneous speech. In *Speech Prosody 2002*, pages 39–46, 2002.

[118] K. MAEKAWA. X-jtobi : an extended j-tobi for spontaneous speech. *Proc. 7th ICSLP, 2002*, pages 1545–1548, 2002.

[119] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[120] R. J. Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Rob Clark, and Rif A. Saurous. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4700–4709. PMLR, 2018.

[121] Daisy Stanton, Yuxuan Wang, and R. J. Skerry-Ryan. Predicting expressive speaking style from text in end-to-end speech synthesis. In *2018 IEEE Spoken*

Language Technology Workshop, SLT 2018, Athens, Greece, December 18-21, 2018, pages 595–602. IEEE, 2018.

[122] Wei-Ning Hsu, Yu Zhang, Ron J. Weiss, Heiga Zen, Yonghui Wu, Yuxuan Wang, Yuan Cao, Ye Jia, Zhifeng Chen, Jonathan Shen, Patrick Nguyen, and Ruoming Pang. Hierarchical generative modeling for controllable speech synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[123] Raza Habib, Soroosh Mariooryad, Matt Shannon, Eric Battenberg, R. J. Skerry-Ryan, Daisy Stanton, David Kao, and Tom Bagby. Semi-supervised generative modeling for controllable speech synthesis. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[124] Vatsal Aggarwal, Marius Cotescu, Nishant Prateek, Jaime Lorenzo-Trueba, and Roberto Barra-Chicote. Using vaes and normalizing flows for one-shot text-to-speech synthesis of expressive speech. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 6179–6183. IEEE, 2020.

[125] Viacheslav Klimkov, Srikanth Ronanki, Jonas Rohnke, and Thomas Drugman. Fine-grained robust prosody transfer for single-speaker neural text-to-speech. In Gernot Kubin and Zdravko Kacic, editors, *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 4440–4444. ISCA, 2019.

[126] G. Sun, Y. Zhang, R. J. Weiss, Y. Cao, H. Zen, A. Rosenberg, B. Ramabhadran, and Y. Wu. Generating diverse and natural text-to-speech samples using a quantized fine-grained vae and autoregressive prosody prior. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6699–6703, 2020.

[127] Sri Karlapati, Alexis Moinet, Arnaud Joly, Viacheslav Klimkov, Daniel Sáez-Trigueros, and Thomas Drugman. Copycat: Many-to-many fine-grained prosody transfer for neural text-to-speech. In Helen Meng, Bo Xu, and Thomas Fang

Zheng, editors, *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 4387–4391. ISCA, 2020.

[128] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6306–6315, 2017.

[129] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[130] Jason Tyler Rolfe. Discrete variational autoencoders. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[131] Gustav Eje Henter, Xin Wang, and Junichi Yamagishi. Deep encoder-decoder models for unsupervised learning of controllable speech synthesis. *CoRR*, abs/1807.11470, 2018.

[132] Cristina Gârbacea, Aäron van den Oord, Yazhe Li, Felicia S. C. Lim, Alejandro Luebs, Oriol Vinyals, and Thomas C. Walters. Low bit-rate speech coding with VQ-VAE and a wavenet decoder. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*, pages 735–739. IEEE, 2019.

[133] Shaojin Ding and Ricardo Gutierrez-Osuna. Group Latent Embedding for Vector Quantized Variational Autoencoder in Non-Parallel Voice Conversion. In *Proc. Interspeech 2019*, pages 724–728, 2019.

[134] Jan Chorowski, Ron J. Weiss, Samy Bengio, and Aäron van den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE ACM Trans. Audio Speech Lang. Process.*, 27(12):2041–2053, 2019.

[135] Andros Tjandra, Berrak Sisman, Mingyang Zhang, Sakriani Sakti, Haizhou Li, and Satoshi Nakamura. VQVAE Unsupervised Unit Discovery and Multi-Scale Code2Spec Inverter for Zerospeech Challenge 2019. In *Proc. Interspeech 2019*, pages 1118–1122, 2019.

[136] X. Wang, S. Takaki, J. Yamagishi, S. King, and K. Tokuda. A vector quantized variational autoencoder (vq-vae) autoregressive neural f_0 model for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:157–170, 2020.

[137] Yi Zhao, Haoyu Li, Cheng-I Lai, Jennifer Williams, Erica Cooper, and Junichi Yamagishi. Improved prosody from learned F0 codebook representations for VQ-VAE speech waveform reconstruction. *CoRR*, abs/2005.07884, 2020.

[138] Tomoki Hayashi and Shinji Watanabe. Discretalk: Text-to-speech as a machine translation problem. *CoRR*, abs/2005.05525, 2020.

[139] Yusuke Yasuda, Xin Wang, and Junichi Yamagishi. End-to-end text-to-speech using latent duration based on VQ-VAE. *CoRR*, abs/2010.09602, 2020.

[140] Gustav Eje Henter, Jaime Lorenzo-Trueba, Xin Wang, and Junichi Yamagishi. Deep encoder-decoder models for unsupervised learning of controllable speech synthesis. *arXiv preprint arXiv:1807.11470*, 2018.

[141] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *Proc. ICLR*, 2015.

[142] Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. Online and linear-time attention by enforcing monotonic alignments. In Doina Precup and Yee Whye Teh, editors, *Proc. ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2837–2846. PMLR, 2017.

[143] Lei Yu, Jan Buys, and Phil Blunsom. Online segment to segment neural transduction. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,*

EMNLP 2016, Austin, Texas, USA, November 1-4, 2016, pages 1307–1316. The Association for Computational Linguistics, 2016.

[144] Lei Yu, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Tomás Kociský. The neural noisy channel. In *Proc. ICLR*, 2017.

[145] Jason Eisner. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In Kai-Wei Chang, Ming-Wei Chang, Alexander M. Rush, and Vivek Srikumar, editors, *Proceedings of the Workshop on Structured Prediction for NLP@EMNLP 2016, Austin, TX, USA, November 5, 2016*, pages 1–17. Association for Computational Linguistics, 2016.

[146] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proc. ICML*, pages 807–814, 2010.

[147] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[148] John I. Yellott. The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109 – 144, 1977.

[149] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proc ICLR (conference track)*. OpenReview.net, 2017.

[150] Yusuke Yasuda, Xin Wang, and Junichi Yamagishi. Initial investigation of encoder-decoder end-to-end TTS using marginalization of monotonic hard alignments. In *Proc. 10th ISCA Speech Synthesis Workshop*, pages 211–216, 2019.

[151] Alice Turk and Stefanie Shattuck-Hufnagel. Timing in talking: what is it used for, and how is it controlled? *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1658):20130395, 2014.

[152] Jaehyeon Kim, Sungwon Kim, Jungil Kong, and Sungroh Yoon. Glow-tts: A generative flow for text-to-speech via monotonic alignment search. *CoRR*, abs/2005.11129, 2020.

[153] Tomoki Hayashi, Ryuichi Yamamoto, Katsuki Inoue, Takenori Yoshimura, Shinji Watanabe, Tomoki Toda, Kazuya Takeda, Yu Zhang, and Xu Tan. Espnet-tts: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 7654–7658. IEEE, 2020.

A

Conditional VQ-VAE

A.1 ELBO in detail

To recap, we use $\mathbf{x}_{1:T}$ and $\mathbf{y}_{1:U}$ to denote the output acoustic feature and input linguistic (e.g., phoneme and character) sequences, respectively. We use $\mathbf{z}_u^{(q)}$ to denote the quantized latent and $\mathbf{z}_u^{(r)}$ to denote the un-quantized latent vectors for the u-th input token, $\forall u \in \{1, \cdots, U\}$. We further assume $\mathbf{z}_u^{(r)} \in \mathbb{R}^D$, where D is the dimension of the latent vector. For the quantized latent variable, we let $\mathbf{z}_u^{(q)} \in \mathcal{Z} = \{\mathbf{e}_1, \cdots, \mathbf{e}_K\}$, where $\mathcal{Z}$ is the codebook with K code words, and $\mathbf{e}_k \in \mathbb{R}^D, \forall k \in \{1, \cdots, K\}$.

We treat $\mathbf{z}_{1:U}^{(q)} = (\mathbf{z}_1^{(q)}, \cdots, \mathbf{z}_U^{(q)})$ and $\mathbf{z}_{1:U}^{(r)} = (\mathbf{z}_1^{(r)}, \cdots, \mathbf{z}_U^{(r)})$ as latent variables. With

Jensen's inequality, we have

$$\log p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:U}) \tag{A.1}$$

$$= \log \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} p(\mathbf{x}_{1:T}, \mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{y}_{1:U}) d\mathbf{z}^{(r)}_{1:U} \tag{A.2}$$

$$= \log \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} \underbrace{p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})}_{(\mathbf{x}_{1:T} \perp\!\!\!\perp \mathbf{z}^{(r)}_{1:U}) | \mathbf{z}^{(q)}_{1:U}} p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}) P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U}) d\mathbf{z}^{(r)}_{1:U} \tag{A.3}$$

$$\geq \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} \underbrace{Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{approximate posterior}} \log \frac{p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})\, p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}) P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}{\underbrace{Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{approximate posterior, factorize it to } Q(z^{(q)})Q(z^{(r)}|z^{(q)})}} d\mathbf{z}^{(r)}_{1:U} \tag{A.4}$$

$$= \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \underbrace{\log \frac{p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})\, p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}) P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}{Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}}_{\text{decompose this term}} d\mathbf{z}^{(r)}_{1:U} \tag{A.5}$$

$$= \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \underbrace{\int_{\mathbf{z}^{(r)}_{1:U}} Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{integrate out } \mathbf{z}^{(r)}_{1:U}} \log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}) d\mathbf{z}^{(r)}_{1:U} \tag{A.6}$$

$$+ \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \underbrace{\int_{\mathbf{z}^{(r)}_{1:U}} Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{integrate out } \mathbf{z}^{(r)}_{1:U}} \log \frac{P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}{Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} d\mathbf{z}^{(r)}_{1:U} \tag{A.7}$$

$$+ \sum_{\forall \mathbf{z}^{(q)}_{1:U}} \int_{\mathbf{z}^{(r)}_{1:U}} \underbrace{Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{Q(\mathbf{z}^{(q)}_{1:U}) Q(\mathbf{z}^{(r)}_{1:U} | \mathbf{z}^{(q)}_{1:U})} \log \frac{p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})}{Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} d\mathbf{z}^{(r)}_{1:U} \tag{A.8}$$

$$= \mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U} | \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} [\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})] \tag{A.9}$$

$$+ \sum_{\forall \mathbf{z}^{(q)}_{1:U}} Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \log \frac{P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}{Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} d\mathbf{z}^{(r)}_{1:U} \tag{A.10}$$

$$+ \sum_{\forall \mathbf{z}^{(q)}_{1:U}} Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \int_{\mathbf{z}^{(r)}_{1:U}} Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \log \frac{p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})}{Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} d\mathbf{z}^{(r)}_{1:U} \tag{A.11}$$

$$= \mathbb{E}_{Q_\lambda(\mathbf{z}^{(q)}_{1:U} | \mathbf{x}_{1:T}, \mathbf{y}_{1:U})} \underbrace{[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})]}_{\text{Decoder}} \tag{A.12}$$

$$- \mathrm{KL}[Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| \underbrace{P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})}_{\text{Prior}}] \tag{A.13}$$

Equations (A.12)–(A.14) denote the evidence lower bound (ELBO). In Eq. (A.3), we assume $(\mathbf{x}_{1:T} \perp\!\!\!\perp \mathbf{z}^{(r)}_{1:U})|\mathbf{z}^{(q)}_{1:U}$, which means that $\mathbf{x}_{1:T}$ is conditionally independent of $\mathbf{z}^{(r)}_{1:U}$, given $\mathbf{z}^{(q)}_{1:U}$ and $\mathbf{y}_{1:U}$. We use the chain rule to factorize the distribution $p(\mathbf{x}_{1:T}, \mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{y}_{1:U}) = p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U})\, p(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}) P_\phi(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U})$. The definition of each part of the ELBO is explained in the following subsections.

A.2 Approximate posterior

We define $Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T})$ as

$$Q(\mathbf{z}^{(q)}_{1:U}, \mathbf{z}^{(r)}_{1:U} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}), \tag{A.16}$$

where

$$Q_\lambda(\mathbf{z}^{(q)}_{1:U} \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} P(\mathbf{z}^{(q)}_u \mid \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{I}(\mathbf{z}^{(q)}_u = \mathbf{e}_{l_u}), \tag{A.17}$$

and

$$Q_\psi(\mathbf{z}^{(r)}_{1:U} \mid \mathbf{z}^{(q)}_{1:U}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} p(\mathbf{z}^{(r)}_u \mid \mathbf{z}^{(q)}_{u-1}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \prod_{u=1}^{U} \mathcal{F}(\mathbf{z}^{(r)}_u - \mathbf{d}_u), \tag{A.18}$$

$$\mathbf{d}_u = \text{LatentPredictor}_\psi(\mathbf{z}^{(q)}_{u-1}, \bar{\mathbf{x}}_u, y_u). \tag{A.19}$$

In the above equations, l_u is the codeword index for the u-th input token, $\mathcal{I}(\cdot)$ is an indicator function, and $\mathcal{F}(\cdot)$ is any fixed unimodal probability density function centered on the origin, such as the isotropic Gaussian $\mathcal{F}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; 0, \mathbf{I})$. The LatentPredictor is a neural network with a parameter set ψ.

Note that $\bar{\mathbf{x}}_u$ is from an aggregated acoustic features sequence $\bar{\mathbf{x}}_{1:U} = (\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_U)$, which has the same length U as the linguistic feature input $\bar{\mathbf{y}}_{1:U}$.

A.3 Decoder

Just to repeat the definition of the decoder, given $\mathbf{y}_{1:U}$, $\mathbf{l}_{1:U} = (l_1, \cdots, l_U)$ and $\mathbf{z}_{1:U}^{(q)} = (\mathbf{z}_1^{(q)} = \mathbf{e}_{l_1}, \cdots, \mathbf{z}_U^{(q)} = \mathbf{e}_{l_U})$, the PDF of $\mathbf{x}_{1:T}$ is defined as

$$p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t) = \prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_t; \mu_t, \sigma_d^2 \mathbf{I}), \tag{A.20}$$

where $\mu_t = \text{Decoder}_\theta(\mathbf{x}_{t-1}, \widehat{\mathbf{z}}_t^{(q)}, \widehat{y}_t)$. Note that $\widehat{\mathbf{y}}_{1:T} = \{\widehat{y}_1, \cdots, \widehat{y}_T\}$ and $\widehat{\mathbf{z}}_{1:T}^{(q)} = \{\widehat{\mathbf{z}}_1^{(q)}, \cdots, \widehat{\mathbf{z}}_T^{(q)}\}$ are upsampled from $\mathbf{y}_{1:U}$ and $\mathbf{z}_{1:U}^{(q)}$.

A.4 Prior

We assume the prior probability of observing the l-th codeword $\mathbf{e}_l$ to be

$$P_\phi(\mathbf{z}_{1:U}^{(q)} = \mathbf{e}_l \mid \mathbf{y}_{1:U}) = \prod_{u=1}^{U} P(\mathbf{z}_u^{(q)} = \mathbf{e}_l \mid \mathbf{z}_{u-1}^{(q)}, y_u) = \prod_{u=1}^{U} \frac{\exp(-\|\mathbf{c}_u - \mathbf{e}_l\|_2^2)}{\sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2)}, \tag{A.21}$$

$$\mathbf{c}_u = \text{LatentPredictor}_\phi(\mathbf{z}_{u-1}^{(q)}, y_u), \tag{A.22}$$

where $\text{LatentPredictor}_\phi(\mathbf{z}_{u-1}^{(q)}, y_u)$ is a neural network with parameter ϕ.

With Eqs. (A.21) and Eq. (A.17), the KL divergence in Eq. (A.13) can be computed as

$$\mathrm{KL}[Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})] = \mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)} | \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}[\log \frac{Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}{P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})}], \tag{A.23}$$

$$= - \mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)} | \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}[\log P_\phi(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{y}_{1:U})] - \underbrace{H(Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}))}_{=0 \text{ for indicator function}} \tag{A.24}$$

$$= - \log P_\phi(\mathbf{z}_{1:U}^{(q)} = (\mathbf{e}_{l_1}, \cdots, \mathbf{e}_{l_U}) \mid \mathbf{y}_{1:U}), \tag{A.25}$$

$$= \sum_{u=1}^{U} \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2). \tag{A.26}$$

Note that the $H(\cdot)$ term calculates the entropy of the indicator function, which is equal to 0.

A.5 Vector quantization

In the final term in Eq. (A.14), let us define

$$p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) = \prod_{u=1}^{U} p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_u^{(q)}) = \prod_{u=1}^{U} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)}, \sigma^2 \mathbf{I}), \tag{A.27}$$

where $\mathbf{I}$ is an identity matrix and σ^2 is a fixed parameter. For the first equality, we assume that $\mathbf{z}_u^{(r)}$ is conditionally independent from $\mathbf{y}_{1:U}$, $\mathbf{z}_v^{(q)}$, and $\mathbf{z}_v^{(r)}, v \neq u$. This second equality assumes that $\mathbf{z}_u^{(r)} = \mathbf{z}_u^{(q)} + \eta$, where $\eta \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$. In this case, we assume the quantization loss is from a Gaussian distribution.

With Eqs. (A.18) and Eq. (A.27), we can compute Eq. (A.14) in ELBO as

$$\underbrace{\mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}}_{\text{indicator function}} \Big\{ \mathrm{KL}\big[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U})\big] \Big\}, \tag{A.28}$$

$$= \mathrm{KL}\big[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)} = \mathbf{e}_{1:U}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)} = \mathbf{e}_{1:U}, \mathbf{y}_{1:U})\big], \tag{A.29}$$

$$= \sum_{u=1}^{U} \mathrm{KL}\big[p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)} = \mathbf{e}_{l_{u-1}}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) \| \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}, \sigma^2 \mathbf{I})\big]. \tag{A.30}$$

In Eq. (A.18), we defined that $p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \mathcal{F}(\mathbf{z}_u^{(r)} - \mathbf{d}_u)$, where $\mathcal{F}(\cdot)$ can be a unimodal distribution and $\mathbf{d}_u = \mathrm{LatentPredictor}_\psi(\mathbf{z}_{u-1}^{(q)}, \bar{\mathbf{x}}_u, y_u)$. Now let us define $\mathcal{F}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; 0, \sigma^2 \mathbf{I})$, whose covariance matrix is the same as that for $\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)}, \sigma^2 \mathbf{I})$. Accordingly, we have $p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2 \mathbf{I})$. The KL divergence between the two multivariate Gaussians can be analytically computed as

$$\mathrm{KL}\big[\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2 \mathbf{I}) \| \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}, \sigma^2 \mathbf{I})\big] = \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2. \tag{A.31}$$

By plugging the KL divergence into Eq. (A.30), we have

$$\underbrace{\mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}}_{\text{indicator function}} \Big\{ \mathrm{KL}\big[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U})\big] \Big\} = \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2. \tag{A.32}$$

A.6 In summary

With Eqs. (A.26) and (A.32), we get the final form for the ELBO:

$$\mathrm{ELBO}(\mathbf{x}_{1:T}, \mathbf{y}_{1:U}) = \mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)})}[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U})], \tag{A.33}$$

$$- \sum_{u=1}^{U} \Big\{ \|\mathbf{c}_u - \mathbf{e}_{l_u}\|_2^2 + \log \sum_{k=1}^{K} \exp(-\|\mathbf{c}_u - \mathbf{e}_k\|_2^2) \Big\}, \tag{A.34}$$

$$- \sum_{u=1}^{U} \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2, \tag{A.35}$$

where $\mathbf{d}_u = \text{LatentPredictor}_\psi(\mathbf{z}_{u-1}^{(q)}, \bar{\mathbf{x}}_u, y_u)$ and $\mathbf{c}_u = \text{LatentPredictor}_\phi(\mathbf{z}_{u-1}^{(q)}, y_u)$.

Our derivation is similar to [140]. However, we further take into account the condition $y_{1:U}$ and define a parametric form for the prior in Eq. (A.21) rather than assuming it to be uniform. This parametric prior leads to the loss in Eq. (A.34), which is not included in unconditional models. We further assume a Gaussian for $p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_{u-1}^{(q)}, \mathbf{y}_{1:U}, \mathbf{x}_{1:T}) = \mathcal{F}(\mathbf{z}_u^{(r)} - \mathbf{d}_u)$, rather than assuming it to be a Dirac delta function.

Note how Eqs. (A.33) and (A.35) are similar to the original training criteria of VQ-VAE (with $\beta = 1$). Our derivation can thus be used to interpret conditional VQ-VAE.

A.7 Alternative to Gaussian quantization noise

As an alternative to the procedure in Section A.5, we may also define the vector quantization part as

$$p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) = \prod_{u=1}^{U} p(\mathbf{z}_u^{(r)} \mid \mathbf{z}_u^{(q)}) = \prod_{u=1}^{U} Z \frac{\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)}, \sigma^2\mathbf{I})}{\frac{1}{K}\sum_{k=1}^{K} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{e}_k, \sigma^2\mathbf{I})}, \tag{A.36}$$

where Z is a scalar to normalize the PDF so that $\int p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) d\mathbf{z}_{1:U}^{(r)} = 1$.

Then, the KL divergence can be computed as

$$\mathbb{E}_{\underbrace{Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})}_{\text{indicator function}}} \left\{ \text{KL}\left[Q_\psi(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{x}_{1:T}, \mathbf{y}_{1:U}) \| p(\mathbf{z}_{1:U}^{(r)} \mid \mathbf{z}_{1:U}^{(q)}, \mathbf{y}_{1:U}) \right] \right\}, \tag{A.37}$$

$$= \sum_{u=1}^{U} \text{KL}\left[\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}) \| \frac{\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}, \sigma^2\mathbf{I})}{\frac{1}{K}\sum_{k=1}^{K} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{e}_k, \sigma^2\mathbf{I})} Z \right], \tag{A.38}$$

$$= \sum_{u=1}^{U} \left\{ \text{KL}\left[\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}) \| \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{z}_u^{(q)} = \mathbf{e}_{l_u}, \sigma^2\mathbf{I}) \right] - \text{KL}\left[\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}) \| \frac{1}{ZK} \sum_{k=1}^{K} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{e}_k, \sigma^2\mathbf{I}) \right] \right\}, \tag{A.39}$$

$$= \sum_{u=1}^{U} \left\{ \frac{1}{2\sigma^2} \|\mathbf{d}_u - \mathbf{e}_{l_u}\|_2^2 - \text{KL}\left[\mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{d}_u, \sigma^2\mathbf{I}) \| \frac{1}{ZK} \sum_{k=1}^{K} \mathcal{N}(\mathbf{z}_u^{(r)}; \mathbf{e}_k, \sigma^2\mathbf{I}) \right] \right\}. \tag{A.40}$$

For the KL divergence between Gaussian and the mixture of Gaussian, there is no

closed form. If we use approximation (variational approximation in [?]), we get

$$\mathrm{KL}\Big[\mathcal{N}(\mathbf{z}_u^{(r)};\mathbf{d}_u,\sigma^2\mathbf{I})\|\frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(\mathbf{z}_u^{(r)};\mathbf{e}_k,\sigma^2\mathbf{I})\Big] \approx -\log\frac{1}{K}\sum_{k=1}^{K}\exp(-\frac{1}{2\sigma^2}\|\mathbf{d}_u-\mathbf{e}_k\|_2^2). \tag{A.41}$$

With Eqs. (A.41) and (A.40), we have

$$\underbrace{\mathbb{E}_{Q_\lambda(\mathbf{z}_{1:U}^{(q)}\mid\mathbf{x}_{1:T},\mathbf{y}_{1:U})}}_{\text{indicator function}}\Big\{\mathrm{KL}\big[Q_\psi(\mathbf{z}_{1:U}^{(r)}\mid\mathbf{z}_{1:U}^{(q)},\mathbf{x}_{1:T},\mathbf{y}_{1:U})\|p(\mathbf{z}_{1:U}^{(r)}\mid\mathbf{z}_{1:U}^{(q)},\mathbf{y}_{1:U})\big]\Big\}, \tag{A.42}$$

$$\approx\sum_{u=1}^{U}\Big\{\frac{1}{2\sigma^2}\|\mathbf{d}_u-\mathbf{e}_{l_u}\|_2^2+\log\sum_{k=1}^{K}\exp(-\frac{1}{2\sigma^2}\|\mathbf{d}_u-\mathbf{e}_k\|_2^2)\Big\}. \tag{A.43}$$

Here we ignore the constant term on $\frac{1}{ZK}$. While it is also possible to use Eq. (A.43) for the ELBO, in this paper we use the form in Eq. (A.32).

B

Connectionist Temporal Classification

B.1 Sampling of duration with CTC

Sampled duration $\mathbf{l}_{1:U}$ from $Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$ must satisfy two constraints: $1 \leq l_u \leq K$ and $T = \sum_{u=1}^{U} l_u$. Furthermore, the sampled $\mathbf{l}_{1:U}$ should be reasonably accurate so that we do not have to draw many samples during model training. Note that the sampled duration is used in $\text{Aggregate}(\mathbf{x}_{1:T}, \mathbf{z}_{1:U}^{(q)} = \mathbf{l}_{1:U})$ and $\text{Upsample}(\mathbf{y}_{1:U}, \mathbf{z}_{1:U}^{(q)} = \mathbf{l}_{1:U})$ to align the acoustic features $\mathbf{x}_{1:T}$ with linguistic feature sequence $\mathbf{y}_{1:U}$.

The above requirements motivate us to implement $Q_\lambda(\mathbf{z}_{1:U}^{(q)} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$ with the help of a recognition model $P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T})$ based on connectionist temporal classification (CTC) [?]. Since CTC parameterizes the alignment as a trellis (see Fig. B.1b), it is straightforward to convert the monotonic CTC alignment into the duration $\mathbf{z}_{1:U}$. Furthermore, the constraint $T = \sum_{u=1}^{U} z_u$ can be satisfied by only considering the CTC alignments that start from $(t, u) = (1, 1)$ and end at $(t, u) = (T, U)$. Alignment that does not satisfy $1 \leq z_u \leq K$ can also be directly excluded from the trellis, as Fig. B.1a illustrates. Last but not least, if $P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T})$ is well trained, we can select

the alignment $\mathbf{a}^*{}_{1:T}$ that maximizes $P_\lambda(\mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$, and $\mathbf{z}_{1:U}$ derived from $\mathbf{a}^*{}_{1:T}$ is expected to be sufficiently accurate.

To recap, CTC predicts linguistic features $\mathbf{y}_{1:U}$ from acoustic features $\mathbf{x}_{1:T}$ by marginalizing all possible alignments $\mathbf{a}_{1:T}$. The monotonic alignment is represented in alignment transition variable $a_t \in \{\varnothing, \mathbb{I}\}$, where the blank symbol $\varnothing$ means keeping the current linguistic label position, and the shift symbol $\mathbb{I}$ means transition to the next linguistic label position[1]. Accordingly, the probability of observing $\mathbf{y}_{1:U}$ given $\mathbf{x}_{1:U}$ is defined as

$$\{\hat{y}_1 \cdots, \hat{y}_T\} = \text{Upsample}(\mathbf{y}_{1:U}, \text{AlignmentToDuration}(\mathbf{a}_{1:T})) \tag{B.1}$$

$$\mathbf{z}_{1:U} = \{z_1, \cdots, z_U\} = \text{AlignmentToDuration}(\mathbf{a}_{1:T}) = \left\{ z_u \middle| \sum_{t \in \mathcal{T}(z_{1:u})} \delta(a_t = \varnothing) + 1 \right\}. \tag{B.2}$$

During training, the likelihood in Eq. (??) can be computed efficiently with a forward-backward algorithm (as shown in Fig. B.1b) or written as

$$P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T}) = \sum_{t=1}^{T} \sum_{u=1}^{U} \alpha(t,u)\beta(t,u), \tag{B.3}$$

$$\alpha(t,u) = \alpha(t-1,u)p(a_t = \varnothing|x_t)p(\hat{y}_t|x_t) + \alpha(t-1,u-1)p(a_t = \mathbb{I}|x_t)p(\hat{y}_t|x_t) \tag{B.4}$$

$$\beta(t,u) = \beta(t+1,u)p(a_{t+1} = \varnothing|x_{t+1})p(\hat{y}_t|x_{t+1}) + \beta(t+1,u+1)p(a_{t+1} = \mathbb{I}|x_{t+1})p(\hat{y}_{t+1}|x_{t+1}). \tag{B.5}$$

This recognition model is jointly trained with other components of the proposed model, as explained in the next section.

To sample a good alignment from the CTC model, note that the $\mathbf{a}^*_{1:T}$ that maximizes $P_\lambda(\mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$ also maximizes the joint probability $P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T})$. This can

[1]Because we are interested in alignment rather than output labels, instead of including blank label $\varnothing$ to output symbols as one class, we separate conditional probability at each time step into alignment transition probability $P_\lambda(a_t|x_t)$ and output probability $P_\lambda(\bar{y}_t|x_t)$.

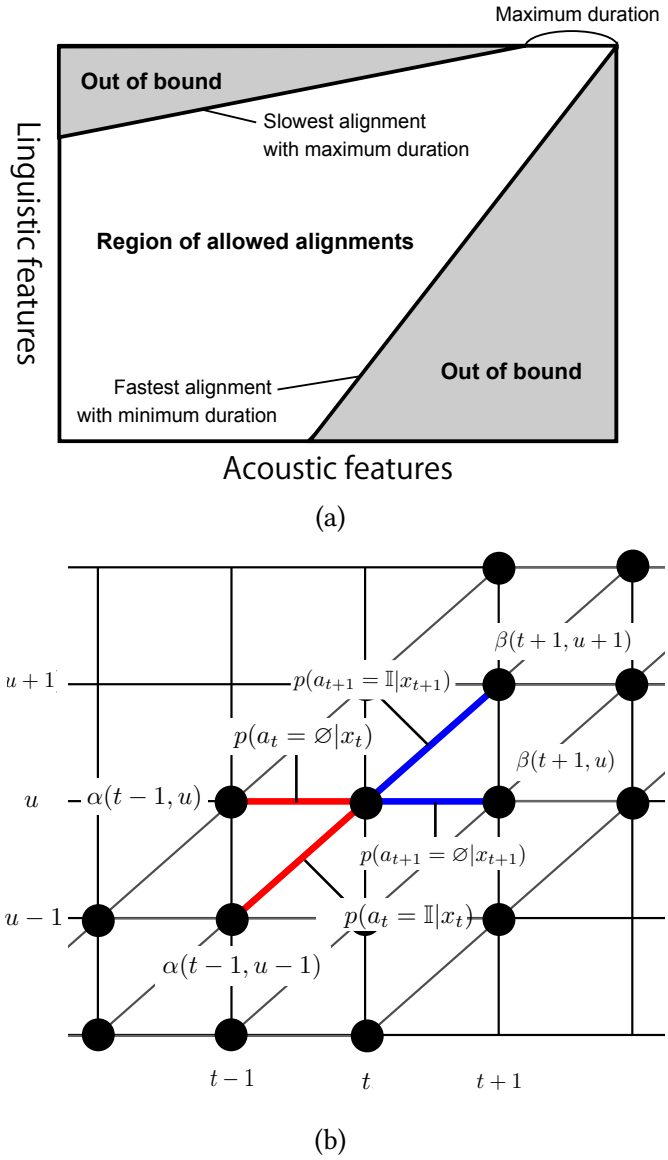


Figure B.1: (a) Example trellis of CTC-based recognition model and (b) constraints on possible path.

be shown by

$$
\begin{aligned}
\mathbf{a}^*{}_{1:T} &= \arg\max_{\mathbf{a}_{1:T}} P_\lambda(\mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}), \\
&= \arg\max_{\mathbf{a}_{1:T}} P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T}) P_\lambda(\mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U}), \\
&= \arg\max_{\mathbf{a}_{1:T}} P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}),
\end{aligned} \tag{B.6}
$$

where $P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T})$ is constant against $\mathbf{a}_{1:T}$. Since the recognition model can optimize the joint probability for all possible alignments by marginalization $P_\lambda(\mathbf{y}_{1:U} \mid \mathbf{x}_{1:T}) = \sum_{\forall \mathbf{a}} P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T})$, the $\mathbf{a}^*_{1:T}$ that maximizes $P_\lambda(\mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:U})$, or equivalently $P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T})$, is expected to be sufficiently accurate for evaluating the ELBO of the proposed model.

Accordingly, we use the following criterion to acquire $\mathbf{a}^*_{1:T}$ and then convert it into the duration sequence $\mathbf{z}_{1:T}$:

$$\mathbf{a}^*_{1:T} = \arg\max_{\mathbf{a}_{1:T}} P_\lambda(\mathbf{y}_{1:U}, \mathbf{a}_{1:T} \mid \mathbf{x}_{1:T}), \tag{B.7}$$

$$\mathbf{l}_{1:U} = \text{AlignmentToDuration}(\mathbf{a}^*{}_{1:T}). \tag{B.8}$$

While we could use a simple greedy search for Eq. (B.7), the outcome might be inferior due to the independence assumption assumed by CTC. In practice, we search for the N best duration by beam search. The search is conducted on the trellis produced by CTC, where the score of each lattice point $\alpha(t, u)\beta(t, u)$ is computed by using its statistics through the forward-backward algorithm.