Doctoral Dissertation

Machine Speech Chain

Andros Tjandra

February 20, 2020

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Andros Tjandra

Machine Speech Chain[*]

Andros Tjandra

Abstract

Despite the close relationship between speech perception and production, research in automatic speech recognition (ASR) and text-to-speech synthesis (TTS) has progressed more or less independently without exerting much mutual influence. In human communication, on the other hand, a closed-loop speech chain mechanism with auditory feedback from the speaker's mouth to her ear is crucial.

We take a step further and develop a closed-loop machine speech chain model based on deep learning. The sequence-to-sequence model in closed-loop architecture allows us to train our model on the concatenation of both labeled and unlabeled data. While ASR transcribes the unlabeled speech features, TTS attempts to reconstruct the original speech waveform based on the text from ASR. In the opposite direction, ASR also attempts to reconstruct the original text transcription given the synthesized speech. To the best of our knowledge, this is the first deep learning framework that integrates human speech perception and production behaviors. Our experimental results show that the proposed approach significantly improved performance over that from separate systems that were only trained with labeled data.

In this thesis, first I present a study about end-to-end speech modeling in general and followed by their application for ASR and TTS. Later, the basic of machine speech chain is described in detail in Chapter 3. Next, we integrate speech chain with speaker embedding model in Chapter 4 to achieve multi-speaker speech chain and improve the ASR and TTS performance on multi-speaker dataset settings. In Chapter 5, we identify the issue where the output of ASR is discrete variables, therefore we proposed a way to fully backpropagate the loss from TTS

[*]Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, February 20, 2020.

to the ASR model by using the straight-through estimator. In Chapter 6, we propose an alternative ASR training with reinforcement learning to solve the discrepancy between training and inference stage.

Keywords:

speech chain, deep learning, machine learning, semi-supervised, speech recognition, speech synthesis

Acknowledgements

Firstly, I would like to express my highest gratitude to Professor Satoshi Nakamura for giving me the opportunity to study and research under his excellent supervision. I attribute the extent of my researches and participation in many conferences to his incredible guidance and immense support; his discipline and tenacity inspire me to become a better researcher.

Secondly, I would like to thank Research Associate Professor Sakriani Sakti for her continuous guidance and recommendations to improve every aspect of my research proficiency. She has provided me with plenty of advice on writing papers, making research presentations, and solving new research problems. Additionally, I thank all the staff in the AHCLab, especially Associate Professor Katsuhito Sudoh, Assistant Professor Koichiro Yoshino, Assistant Professor Hiroki Tanaka, and Research Associate Professor Keiji Yasuda for their feedback during group meetings and seminars. Besides my professors and colleagues at the AHCLab, I would like to thank Manami Matsuda, for her invaluable help with a wide range of tasks, such as administrative work and adjusting to life in Japan as an international student.

My sincere appreciation also goes to all my co-authors, students, and former students of the AHCLab for maintaining a conducive and encouraging environment for conducting research and discussion. Furthermore, thank you to Kano-san and Shinagawa-san for their assistance during my stay in Japan. I thank all my dear friends in Japan who have made my life here cherishable, vibrant, and adventurous. Likewise, thank you to all my friends outside of Japan who have kept in touch with me after all these years.

Last but not least, I want to convey my deepest gratitude to my family for their endless support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Speech Chain on Human Speech Communication

Speech chain, a concept introduced by Denes et al. [1], describes the basic mechanism involved in speech communication when a spoken message travels from the speaker's mind to the listener's mind (Fig. 1.1). It consists of a speech production mechanism in which the speaker produces words and generates speech sound waves, transmits the speech waveform through a medium (i.e., air), and creates a speech perception process in a listener's auditory system to perceive what was said. Over the past few decades, researchers have struggled to understand the principles underlying natural speech communication. Many attempts have also been made to replicate human speech perception and production with machines to support natural modality in human-machine interactions. In the following subsection, we will describe several technologies that allow human-machine interactions over speech information.

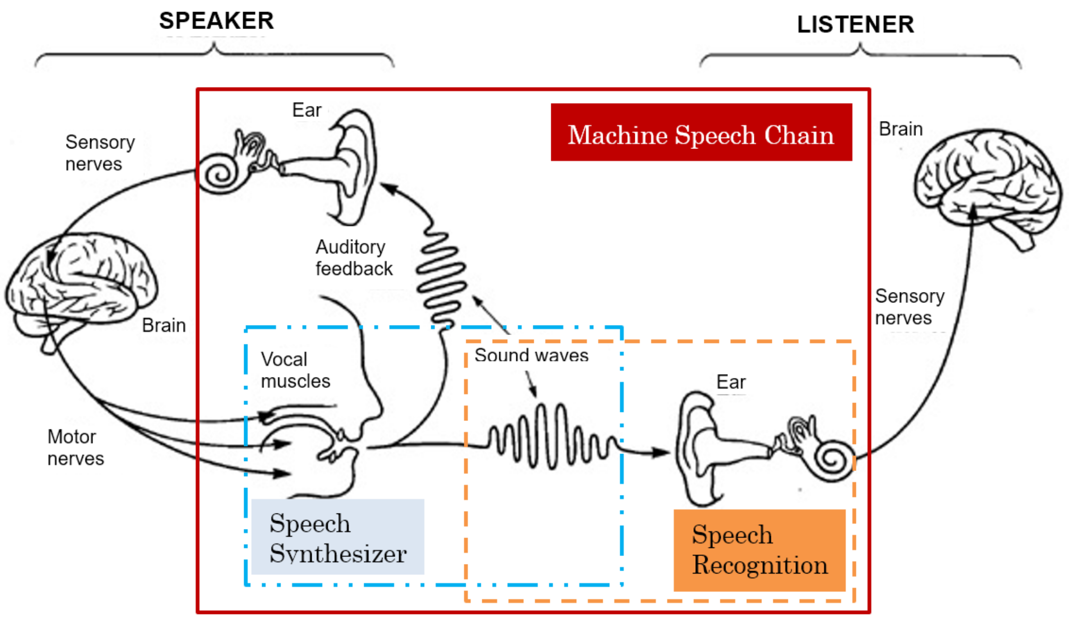


Figure 1.1. Speech chain [1] and the connection with spoken language technologies.

1.2. Technology for Speech Production and Perception

1.2.1 Speech Recognition

Automatic speech recognition (ASR) is an advanced spoken language technologies that enabled machines to process and transcript the speech into human readable format. This process mimics the human auditory system where they listen with their ears and process the sound waves into a concept inside their brain. Various ASR approaches have relied on acoustic-phonetics knowledge [4] in earlier works to template-based schemes with dynamic time warping (DTW) [5, 6]. Later on, the data-driven approaches with rigorous statistical modeling of a hidden Markov model-Gaussian mixture model (HMM-GMM) [7, 8] reached better performance and scaled better with the amount of data and vocabulary size. Since the deep learning [9] become a popular choice for many machine learning tasks, the role of GMM for acoustic modeling was slowly replaced with various deep neural network architecture such as deep stacked recurrent neural networks (RNNs) [10, 11], convolutional layers [12], etc.

1.2.2 Speech Synthesis

Text-to-speech (TTS) or speech synthesis is an advanced speech generation technologies that enabled machines to produce natural sound waves based on the text input. This process mimics the human vocal system where our brain produces some context and our vocal chord start moving and produces sound waves. In a similar direction as ASR, TTS technology has gradually shifted from the foundation of a rule-based system using waveform coding and an analysis-synthesis method [13, 14, 15] to waveform unit concatenation [16, 17] and a more flexible approach using the statistical modeling of a hidden semi-Markov model-GMM (HSMM-GMM) [18, 19]. Recently, after the resurgence of deep learning, interest has also surfaced in the possibility of utilizing a neural approach for ASR and TTS systems. Many state-of-the-art performances in ASR [10, 20, 21] and TTS [22, 23, 24] tasks have been successfully constructed based on neural network frameworks. Later on, end-to-end speech synthesis model [25, 26, 27] generates high quality and natural speech.

1.2.3 Limitation

Despite the close relationship between speech perception and production, ASR and TTS research has progressed more or less independently without exerting much influence on each other. Training an accurate speech recognition system usually requires large amount of paired speech and text dataset. In the process, gathering a large amount of paired speech and text is costly and requires a lot of effort and time consuming. It might be a difficult to have a large paired data especially for low-resource languages. However, unlabeled data is cheap and easy to obtain with minimal effort. Unfortunely, the current ASR or TTS system are still cannot use the unlabeled data to improve their performance. In human communication, on the other hand, a closed-loop speech chain mechanism has a critical auditory feedback mechanism from the speaker's mouth to her ear (Fig. 1.1).

Unfortunately, investigating the inherent links between these two processes is very challenging. Difficulties arise because methodologies and analyses are necessarily quite different when they are extracting the underlying messages from speech waveforms, as in speech perception, or generating an optimum dynamic speaking style from the intended message, as in speech production. Until recently, it was impossible in a joint approach to reunite the problems shared by both modes. However, due to deep learning's representational power, many complicated hand-engineered models have been simplified by letting deep neural nets (DNNs) learn their way from input to output spaces [28]. With this newly emerging approach to sequence-to-sequence mapping tasks, a model with a common architecture can directly learn the mapping between variable-length representations of different modalities: text-to-text sequences [29, 2], speech-to-text sequences [30, 31], text-to-speech sequences [25], and image-to-text sequences [32], etc.

1.3. Thesis Contribution

The main goals of this thesis is to proposed a mechanism for the computer-based speech processing system to handle data limitation inspired by human speech chain mechanism. Specifically, we develop a closed-loop machine speech chain model based on deep learning and construct a sequence-to-sequence model for both ASR and TTS tasks, as well as a loop connection between these two processes. The sequence-to-sequence model in closed-loop architecture allows us to train our model on the concatenation of both labeled and unlabeled data. While ASR transcribes the unlabeled speech features, TTS attempts to reconstruct the original speech waveform based on text from ASR. In the opposite direction, ASR also reconstructs the original text transcription given the synthesized speech. To the best of our knowledge, this is the first deep learning model that integrates human speech perception and production behaviors.

As we describe in the outline before, we have made several novel ideas during our studies:

1. Basic machine speech chain that integrates ASR and TTS and performs on single-speaker task.

2. Multi-speaker speech chain with a speaker-embedding network for handling speech with different voice characteristics.

3. Machine speech chain with a straight-through estimator to allow end-to-end feedback loss through discrete units or subwords.

We shows that our proposed methods jointly train both ASR and TTS parameters together. Speech chain also greatly improved the performance of ASR and TTS in the semi-supervised and supervised settings. In the latter part of this thesis, I also proposed an alternative training mechanism to minimize the discrepancy during training and inference stage for ASR. By using reinforcement learning, we got a significant improvement compared to the model trained with teacher forcing mechanism.

Chapter 2

End-to-End Speech Modeling

In this chapter, we cover some basic knowledges of sequence-to-sequence model. After that, we explain about sequence-to-sequence ASR and TTS. These information will be used extensively over many following chapters and served as the main framework for our researches.

2.1. Sequence-to-Sequence

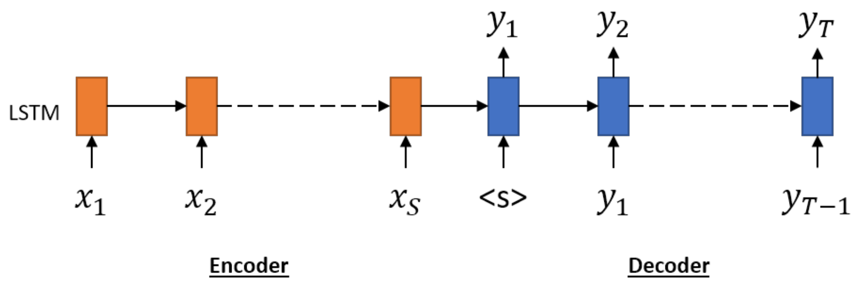


Figure 2.1. Sequence-to-sequence architecture [2].

Sequence-to-sequence model is a neural network that directly models conditional probability between two dynamic length sequence $p_\theta(\mathbf{y}|\mathbf{x}) = p_\theta(y_1, .., y_T|x_1, .., x_S)$ where $\mathbf{x} = [x_1, ..., x_S]$ is the source sequence with length S and $\mathbf{y} = [y_1, ..., y_T]$ is the target sequence with length T and parameterized by θ. Sequence-to-sequence model [2, 33] consists of two components: encoder and decoder. The encoder tasks is to represent a variable length source sequence $\mathbf{x}$ into fixed lengths vector representation v. The decoder tasks is to generate the output target sequence y_t step-by-step, conditioned to the previous output $y_1, ..., y_{<t}$ and fixed encoder representation v. Both encoder and decoder are consisted of LSTM [11] layers. To calculate the conditional probability of generating target sequence, we formulate these operations as:

$$p_\theta(y_1, .., y_T | x_1, .., x_S) = \prod_{t=1}^{T} p_\theta(y_t | y_{<t}, v). \tag{2.1}$$

However, encoder-decoder model has difficulty when the source or target sequence are too long. The reason is the decoder depends only to a fixed vector v from the encoder side, creating an information bottleneck and limit the capabilities to find relevant information in the specific time-step. Therefore, attention mechanism was introduced by [29] to overcome the issue. Attention mechanism creates a “bridge” between encoder hidden states with the decoder hidden states. It allows the decoder to focus only the relevant information from the encoder states and ignore the other information. Attention modules produce context information c_t at time t based on the encoder hidden states h_e^s and decoder hidden states h_t^d:

$$\begin{aligned}
\mathbf{h^e} &= [h_1^e, .., h_S^e] = \text{Enc}(\mathbf{x}) && (2.2)\\
h_t^d &= \text{DecState}(y_{t-1}, h_{t-1}^d, \mathbf{h^e}) && (2.3)\\
a_t(s) &= \text{Align}(h_s^e, h_t^d)\\
&= \frac{\exp(\text{Score}(h_s^e, h_t^d))}{\sum_{s=1}^{S} \exp(\text{Score}(h_s^e, h_t^d))}. && (2.4)\\
c_t &= \sum_{s=1}^{S} a_t(s) * h_s^e && (2.5)
\end{aligned}$$

There are several variations for score functions [34]:

$$\text{Score}(h_s^e, h_t^d) = \begin{cases} \langle h_s^e, h_t^d \rangle, & \text{dot product}\\ h_s^{e\intercal} W_s h_t^d, & \text{bilinear}\\ V_s^\intercal \tanh(W_s[h_s^e, h_t^d]), & \text{MLP} \end{cases} \tag{2.6}$$

where Score : $(\mathbb{R}^M \times \mathbb{R}^N) \to \mathbb{R}$, M is the number of hidden units for the last layer of encoder and N is the number of hidden units for the decoder. We illustrates a sequence-to-sequence with attention mechanism in Figure 2.2.

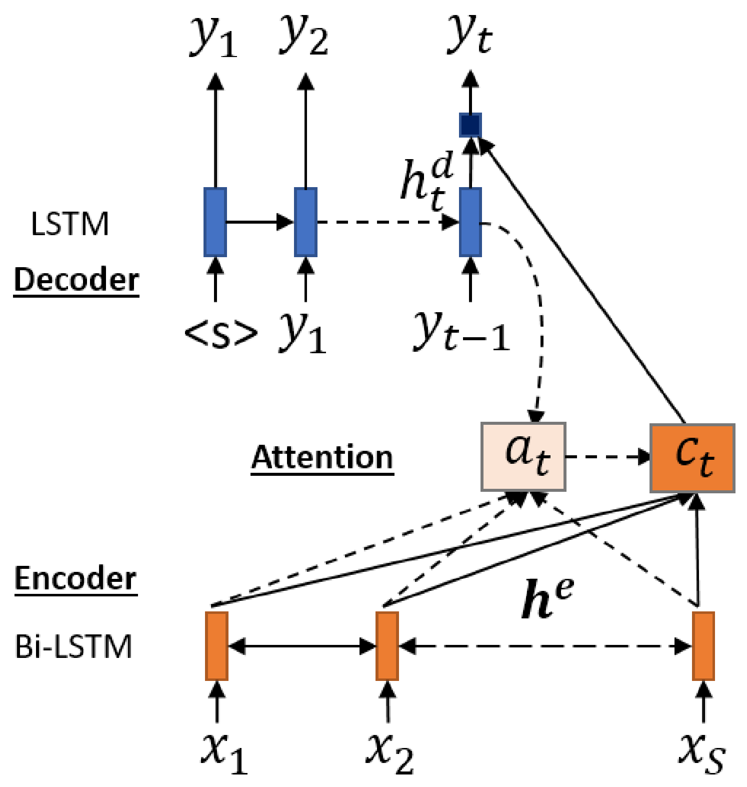


Figure 2.2. Sequence-to-sequence architecture with attention mechanism.

After we get the context information c_t, we concatenated c_t with h_t^d and predict the target output y_t:

$$y_t = W_o\,[h_t^d, c_t] \tag{2.7}$$

where W_o is a matrix for linear projection in this example. The encoder and decoder layers are not limited to LSTM, it could be replaced with GRU [35] or convolution layer [36].

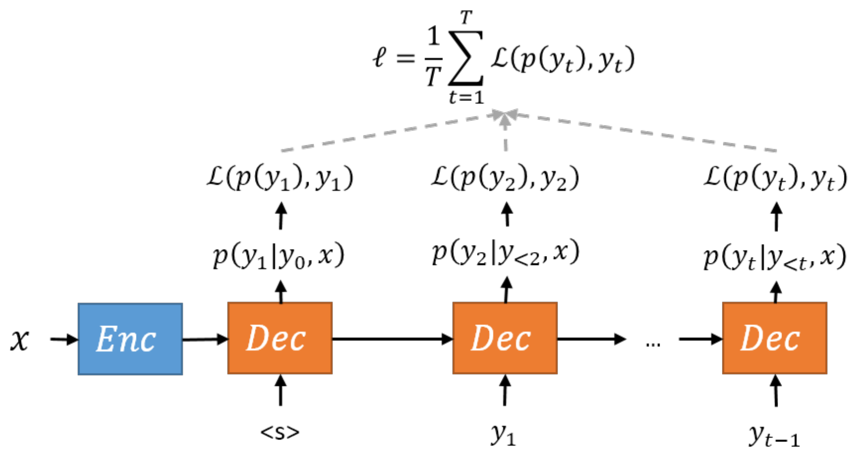

$\ell = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}(p(y_t), y_t)$
$\mathcal{L}(p(y_1), y_1)$
$\mathcal{L}(p(y_2), y_2)$
$\mathcal{L}(p(y_t), y_t)$
$p(y_1|y_0, x)$
$p(y_2|y_{<2}, x)$
$p(y_t|y_{<t}, x)$
x
Enc
Dec
Dec
...
Dec
<s>
y_1
y_{t-1}

Figure 2.3. Teacher forcing training strategy: loss function $\mathcal{L}(\cdot,\cdot)$ denotes the loss between predicted $p(y_t)$ and ground-truth label y_t.

2.2. Training and Decoding Sequence-to-Sequence Model

In the sequence-to-sequence model, the decoder usually generate the output step-by-step, not whole sequences at the same time. This type of decoder are usually described as autoregressive decoder. Here, we formulate the autoregressive decoder that output probability over class as the prediction:

$$p(y_t), h^d_t = \text{Dec}(y_{t-1}, h^d_{t-1}, \mathbf{x}), \tag{2.8}$$

which calculate the output probability p_{y_t} by conditioning on previous output y_{t-1}, previous hidden state h^d_{t-1} and input source sequence $\mathbf{x}$. Function Dec is a simplified decoder operation from Eqs. 2.3-2.5 which output decoder hidden states h^d_t and predicted output $p(y_t)$.

Training autoregressive decoders are mostly done with teacher-forcing strategy. Basically, teacher-forcing conditioned the input with ground-truth during the training stage. In the Figure 2.3, we show the example of teacher forcing. For each time step, we calculate a loss (the loss function itself depends on the nature of the task and their output representation) and accumulate the loss across multiple time-step. However, teacher-forcing training sometimes fail to account

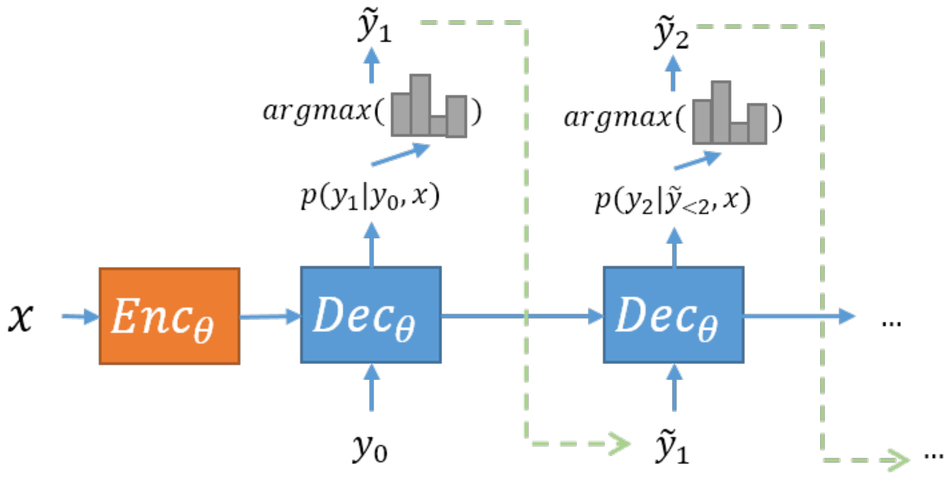


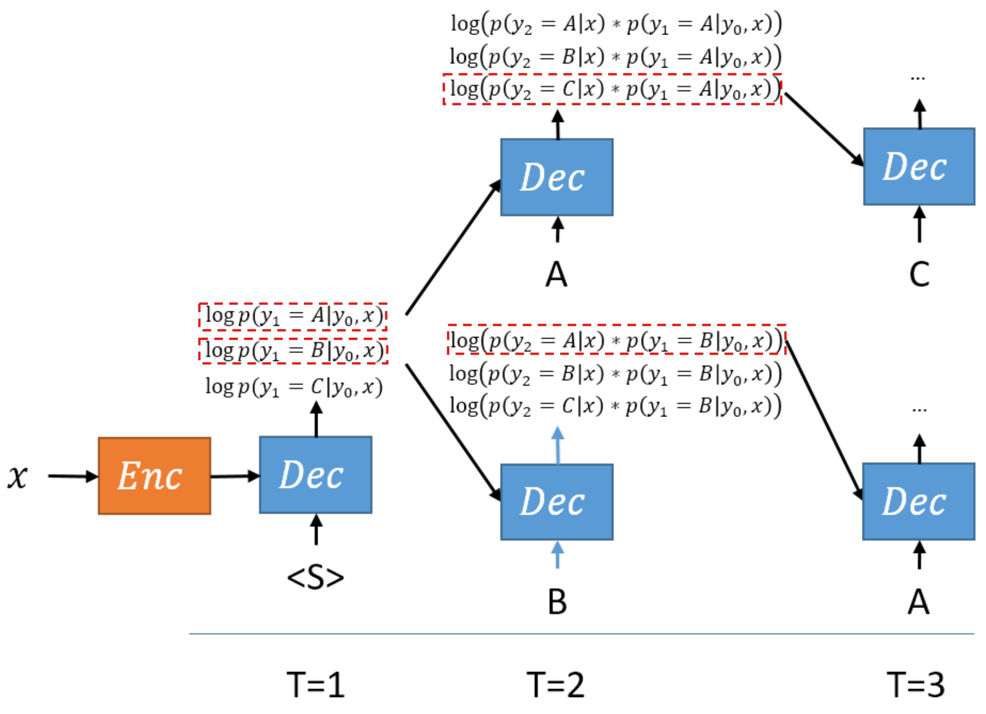
Figure 2.4. Greedy decoding during inference stage.

the ground-truth are not available during inference. Thus, any errors during the early time-step might accumulate in the later time-step and decrease the model robustness against longer utterances. I will revisit this issue and propose a solution in Chapter 6

During the inference stage, we do not have any ground-truth for the decoder input. Therefore, we replace previous output $\mathbf{y}_{<t}$ with predicted $\tilde{\mathbf{y}}_{<t}$. There are two most common strategies for decoding during the inference stage: greedy and beam-search decoding.

Greedy decoding is the simplest decoding strategy and runs with linear complexity $O(T)$ where T is the length of the target sequence. Based on the previous hypothesis, we only need to pick the index with the largest probability as the predicted hypothesis at the current time-step. We show the process in Figure 2.4.

However, most of the time greedy decoding does not generate optimal hypotheses because we only consider one hypothesis in the current time-step instead of expanding all search space. On the other hand, expanding the search space $\mathcal{S}$ with size $V = \|\mathcal{S}\|$ exhaustively also not a good option since the space and time complexity growth exponentially for each time-step with $O(V^T)$ complexity. Therefore, beam-search decoding constraints the search space by only keeping a limited number of good hypotheses and prune out hypotheses with low scores. In the implementation, we could use the priority queue to sort the hypotheses and



$\log(p(y_2 = A|x) * p(y_1 = A|y_0, x))$
$\log(p(y_2 = B|x) * p(y_1 = A|y_0, x))$
$\log(p(y_2 = C|x) * p(y_1 = A|y_0, x))$
...
Dec
Dec
A
C
$\log p(y_1 = A|y_0, x)$
$\log p(y_1 = B|y_0, x)$
$\log p(y_1 = C|y_0, x)$
$\log(p(y_2 = A|x) * p(y_1 = B|y_0, x))$
$\log(p(y_2 = B|x) * p(y_1 = B|y_0, x))$
$\log(p(y_2 = C|x) * p(y_1 = B|y_0, x))$
...
x
Enc
Dec
Dec
Dec
<S>
B
A
T=1
T=2
T=3

Figure 2.5. Beam-search decoding during inference stage with size $k = 2$.

take only top-k hypothesis to be expended in the future steps and discard the rest to avoid memory issues. In the end, beam-search time and space complexity reduced into $O(kT)$. In Figure 2.5, we simulate a beam search with search space size 3 and beam size $k = 2$.

2.3. Sequence-to-Sequence ASR

Based on the sequence-to-sequence model on Section 2.1, sequence-to-sequence ASR follows a similar architecture as Figure 2.2. We model the conditional probability $p_\theta(\mathbf{y}|\mathbf{x})$ where the input source $\mathbf{x}$ for speech recognition tasks is a sequence of feature vectors like the MFCC or Mel-spectrogram. Therefore, $\mathbf{x} \in \mathbb{R}^{S \times D}$, where D is the number of feature dimensions and S is the total frame length for an utterance. Output $\mathbf{y}$, which is a speech transcription sequence, can be either a phoneme or grapheme (character) sequence. Figure 2.6 shows the overall structure of the attention-based encoder-decoder model that consists of encoder, decoder, and attention modules.

The loss function for ASR can be formulated as:

$$\ell_{ASR} = L_{ASR}(\mathbf{y}, \mathbf{p}_y) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C} \mathbb{1}(y_t = c) * \log p_{y_t}[c], \tag{2.9}$$

where C is the number of output classes.

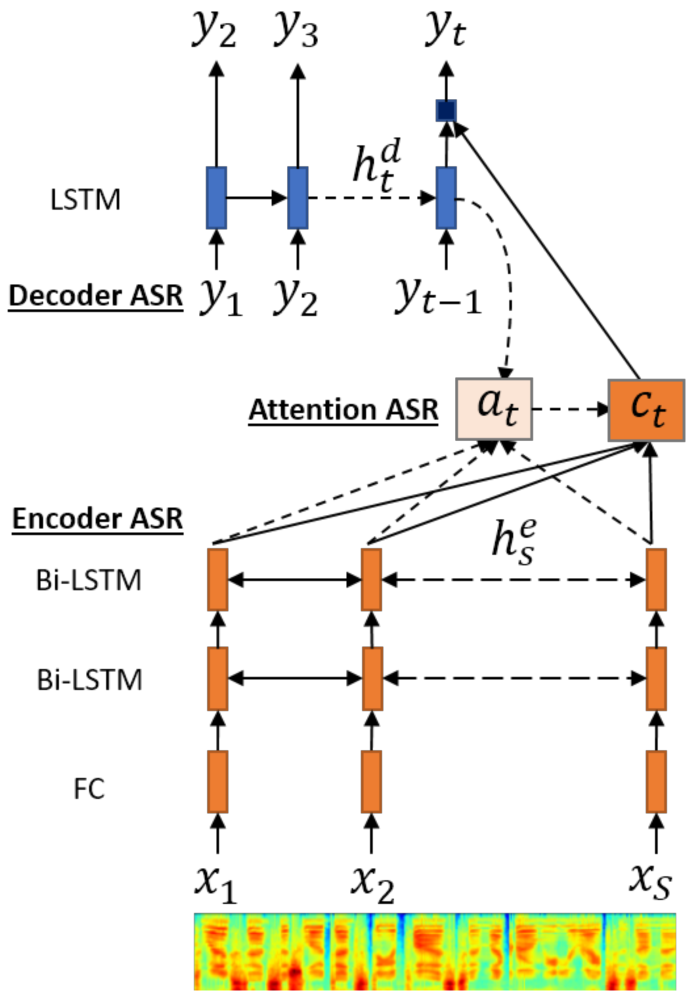


Figure 2.6. Sequence-to-sequence ASR architecture: the encoder consists of a fully connected layer + stack bidirectional LSTM and the decoder consists of a unidirectional LSTM with attention mechanism.

2.4. Sequence-to-Sequence Model for TTS

Parametric speech synthesis resembles a sequence-to-sequence task where we generate speech given a sentence. Using a sequence-to-sequence model, we model the conditional probability between $p_\theta(\mathbf{x}|\mathbf{y})$, where $\mathbf{y} = [y_1, ..., y_T]$ is the sequence of characters with length T and $\mathbf{x} = [x_1, ..., x_S]$ is the sequence of (framed) speech features with length S. From the sequence-to-sequence ASR model perspective, we now have an inverse model for reconstructing the original speech given the text.

In this work, our core architecture is based on Tacotron [25] with several structural modifications. Figure 2.7 illustrates our modified Tacotron. In the encoder sides, we project our input characters with an embedding layer. The character vectors are fed into several fully connected layers followed by a non-linear activation function. We pass the result into the CBHG block (1-D **C**onvolution **B**ank + **H**ighway + bidirectional **G**RU) with eight filter banks (filter size ranging from 1 to 8). The CBHG output is expected to produce representative information $h^e = [h^e_1, ..., h^e_T]$ for the decoder.

Our modified decoder has one input layer and three output layers (instead of two as in the original Tacotron). The first output layer generates a sequence of log Mel-scale spectrogram frames $\mathbf{x}^M = [x^M_1, ..., x^M_S]$. At the s-th step, the input layer is fed by a previous step-log Mel-scale spectrogram x^M_{s-1}, and then several fully connected layers and a non-linear activation function are processed. Next, we use a stacked LSTM with a multilayer perceptron (MLP) attention with alignment and context history [37] to extract the expected context c_s information based on the current decoder input and encoder states h^e. We project the context with a fully connected layer to predict the Mel-scale spectrogram.

The second output layer reconstructs log-magnitude spectrogram $\mathbf{x}^R = [x^R_1, ..., x^R_S]$ given the first layer generated output $\mathbf{x}^M$. After we get complete the sequences of the log Mel-scale spectrogram, we feed them into a CBHG block followed by a fully connected layer to predict the log magnitude spectrogram.

The third output layer generates binary prediction $b_s \in [0, 1]$ (1 if the s-th frame is the end of speech, otherwise 0) based on the current log-Mel spectrogram generated by the first output layer and expected context c_s from the decoder with the attention layer. We add the binary prediction layer because the output from the first and second decoder layers is a real value vector, and we cannot use an end-of-sentence (eos) token to determine when to stop the generation process. Based on our initial experiment, we found that our modification helped Tacotron determine the end of speech more robustly than forcing the decoder to generate frames with a 0 value at the end of the speech. We also enable our model to learn from multiple speakers by concatenating the projected speaker embedding into the input before the LSTM layer, first output regression layer, and second output regression layer.

For training the TTS model, we used the following loss function:

$$\begin{aligned}\ell_{TTS} = L_{TTS}(\mathbf{x}, \hat{\mathbf{x}}) = & \frac{1}{S}\sum_{s=1}^{S} \|x_s^M - \hat{x}_s^M\|_2^2 + \|x_s^R - \hat{x}_s^R\|_2^2 \\ & - (b_s \log(\hat{b}_s) + (1 - b_s)\log(1 - \hat{b}_s))\end{aligned}, \tag{2.10}$$

where $\hat{\mathbf{x}} = (\hat{\mathbf{x}}^M, \hat{\mathbf{x}}^R, \hat{b})$ are the predicted Mel-scale spectrogram, the magnitude spectrogram, and the end-of-frame probability, and $\mathbf{x} = (\mathbf{x}^{\mathbf{M}}, \mathbf{x}^{\mathbf{R}}, b)$ is the ground truth. In the decoding process, we use the Griffin-Lim [38] algorithm to iteratively estimate the phase spectrogram and reconstruct the signal with the inverse short-time Fourier transform (STFT) from the predicted magnitude and phase spectrogram.

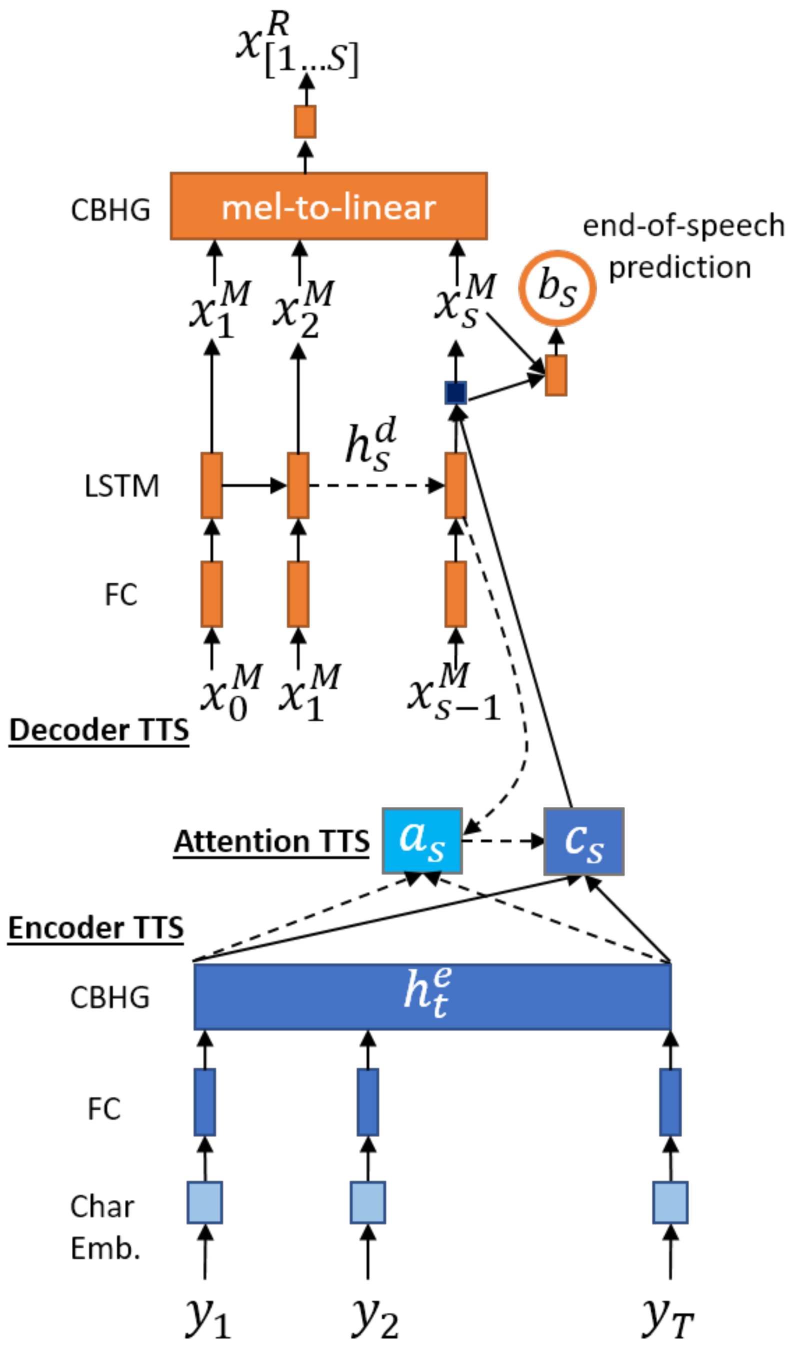


Figure 2.7. Sequence-to-sequence TTS (Tacotron) architecture with frame ending binary prediction. (FC = Fully Connected, CBHG = Convolution Bank + Highway + bi-GRU)

.

Chapter 3

Basic Machine Speech Chain

3.1. Overview

We start by explaining an overall basic machine speech chain mechanism. For a better understanding, we illustrated the speech chain loop in Fig. 3.1(a). Speech chain consists of a sequence-to-sequence ASR (see section 2.3), a sequence-to-sequence TTS (see section 2.4), and a loop connection from ASR to TTS and from TTS to ASR. The key idea is to jointly train both the ASR and TTS models. As mentioned above, the sequence-to-sequence model in closed-loop architecture allows us to train our model on the concatenation of both the labeled (paired) and unlabeled (unpaired) data.

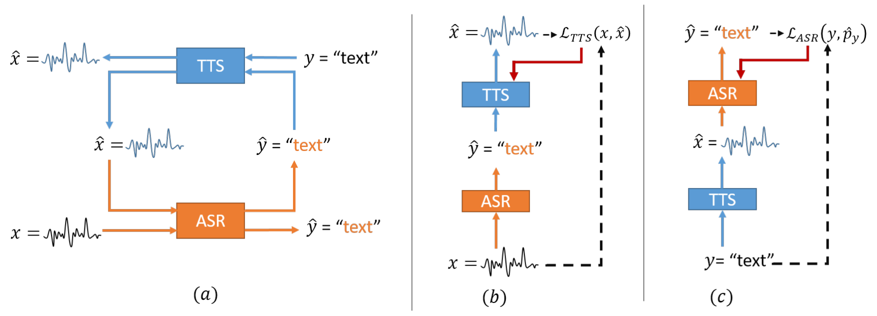


Figure 3.1. (a) Overview of machine speech chain architecture. Examples of unrolled process: (b) from ASR to TTS and (c) from TTS to ASR.

To further clarify the learning process during supervised and unsupervised training, we unrolled the architecture as follows:

1. **Paired speech-text training for ASR and TTS**
   Given the labeled data (speech-text paired data), both models can be trained independently by minimizing the loss between their predicted target sequence and the ground truth sequence via teacher forcing.

2. **Unpaired speech data only (ASR $\rightarrow$ TTS)**
   Given the unlabeled speech features, ASR transcribes the unlabeled input speech, while TTS reconstructs the original speech waveform based on the output text from ASR. Figure 3.1(b) illustrates the mechanism. We may also treat it as an autoencoder model, where the speech-to-text ASR serves as an encoder and the text-to-speech TTS as a decoder.

3. **Unpaired text data only (TTS $\rightarrow$ ASR)**
   Given only the text input, TTS generates speech waveform, while ASR also reconstructs the original text transcription given the synthesized speech. Figure 3.1(c) illustrates the mechanism. Here, we may also treat it as another autoencoder model, where the text-to-speech TTS serves as an encoder and the speech-to-text ASR as a decoder.

With such autoencoder models, ASR and TTS can teach each other by adding a reconstruction term of the observed unlabeled data to the training objective. Details of the algorithm can be found in Alg. 1.

3.2. Experiment on Single-Speaker Task

To verify our proposed method, we experimented on a corpus with a single speaker because, until recently, most TTS systems by deep learning are trained on a single speaker dataset.

We utilized a natural speech single-speaker dataset named LJSpeech [39] that contains about 13,100 utterances. Because there is no official dev and test split from this dataset, we shuffled it and randomly took 94% (total 12,314 utts) for training, 3% (total 393 utts) for dev, and 3% (total 393 utts) for the test set. Later, we split the train-set again to smaller ratio to compare the result between

Algorithm 1 Speech Chain Algorithm (part 1)

```
1: Input:Paired speech and text dataset $\mathcal{D}^P$, text-only dataset $\mathcal{Y}^U$, speech-only
   dataset $\mathcal{X}^U$, supervised loss coefficient $\alpha$, unsupervised loss coefficient $\beta$
2: REPEAT ...
3: A. Supervised training with speech-text data pairs
4: Sample paired speech and text $(\mathbf{x}^P, \mathbf{y}^P) = ([x_1^P, .., x_{S_P}^P], [y_1^P, .., y_{T_P}^P])$ from $\mathcal{D}^P$
   with speech length $S_P$ and text length $T_P$.
5: Generate a text probability vector by ASR using teacher forcing:
   $p_{y_t} = P(y_t | y_{<t}^P, \mathbf{x}^P; \theta_{ASR}), \forall t \in [1..T_P]$
6: Generate best predicted speech by TTS using teacher forcing:
   $\hat{x}_s^P = \underset{z}{\text{argmax}}\, P(z | \mathbf{x}_{<s}^P, \mathbf{y}^P; \theta_{TTS}); \forall s \in [1..S_P]$
7: Calculate the loss for ASR and TTS                                    ▷ Eq. 2.9 & 2.10
   $$\ell_{ASR}^P = \mathcal{L}_{ASR}(\mathbf{y}^P, \mathbf{p}_y; \theta_{ASR}) \quad (3.1)$$
   $$\ell_{TTS}^P = \mathcal{L}_{TTS}(\mathbf{x}^P, \hat{\mathbf{x}}^P; \theta_{TTS}) \quad (3.2)$$
8: B. Unsupervised training with unpaired speech and text
9: # Unpaired text data (TTS → ASR):
10: Sample text $\mathbf{y}^U = [y_1^U, .., y_{T_U}^U]$ from $\mathcal{Y}^U$
11: Generate speech by TTS: $\hat{\mathbf{x}}^U \sim P_{TTS}(\cdot | \mathbf{y}^U; \theta_{TTS})$
12: Generate text probability vector by ASR from TTS's predicted speech using
    teacher forcing: $p_{y_t} = P(y_t | \mathbf{y}_{<t}^U, \hat{\mathbf{x}}^U; \theta_{ASR}); \quad \forall t \in [1..T_U]$
13: Calculate the loss between original text $\mathbf{y}^U$ and reconstruction probability $\mathbf{p}_y$
    $$\ell_{ASR}^U = \mathcal{L}_{ASR}(\mathbf{y}^U, \mathbf{p}_y; \theta_{ASR}) \quad (3.3)$$
14: # Unpaired speech data (ASR → TTS):
15: Sample speech $\mathbf{x}^U = [x_1^U, .., x_{S_U}^U]$ from $\mathcal{X}^U$
16: Generate text by ASR: $\hat{\mathbf{y}}^U \sim P_{ASR}(\cdot | \mathbf{x}^U; \theta_{ASR})$
17: Generate speech by TTS from ASR's predicted text using teacher forcing:
    $\hat{x}_s^U = \underset{z}{\text{argmax}}\, P_{TTS}(z | \mathbf{x}_{<s}^U, \hat{\mathbf{y}}^U; \theta_{TTS}); \quad \forall s \in [1..S]$
18: Calculate the loss between original speech $\mathbf{x}^U$ and generated speech $\hat{\mathbf{x}}^U$
    $$\ell_{TTS}^U = \mathcal{L}_{TTS}(\mathbf{x}^U, \hat{\mathbf{x}}^U; \theta_{TTS}) \quad (3.4)$$
```

Algorithm 2 Speech Chain Algorithm (part 2)

```
19: # Loss combination:
20: Combine all weighted loss into a single loss variable

        $$\ell_{ALL} = \alpha * (\ell^P_{TTS} + \ell^P_{ASR}) + \beta * (\ell^U_{TTS} + \ell^U_{ASR}) \quad (3.5)$$

21: Calculate TTS and ASR parameters gradient with
    the derivative of $\ell_{ALL}$ w.r.t $\theta_{ASR}, \theta_{TTS}$

        $$G_{ASR} = \nabla_{\theta_{ASR}} \ell \quad (3.6)$$
        $$G_{TTS} = \nabla_{\theta_{TTS}} \ell \quad (3.7)$$

22: Update TTS and ASR parameters with gradient descent
    optimization (SGD, Adam, etc)

        $$\theta_{ASR} \leftarrow Optim(\theta_{ASR}, G_{ASR}) \quad (3.8)$$
        $$\theta_{TTS} \leftarrow Optim(\theta_{TTS}, G_{TTS}) \quad (3.9)$$

23: UNTIL convergence of parameter $\theta_{TTS}, \theta_{ASR}$
```

the paired only and paired + unpaired data. In Figure 3.2, we illustrate how we split the training data for supervised (baseline) with small ratio of paired data, supervised (upperbound) with full paired data and semi-supervised with paired, unpaired text and unpaired speech. For the unpaired speech and text, there are 70% remaining unused data. To get the unpaired text, we take it by randomly sample 35% (without replacement) from the unused data. For the unpaired speech, we take it from the remaining 35% data. Therefore, there is no overlap between unpaired speech and unpaired text dataset.

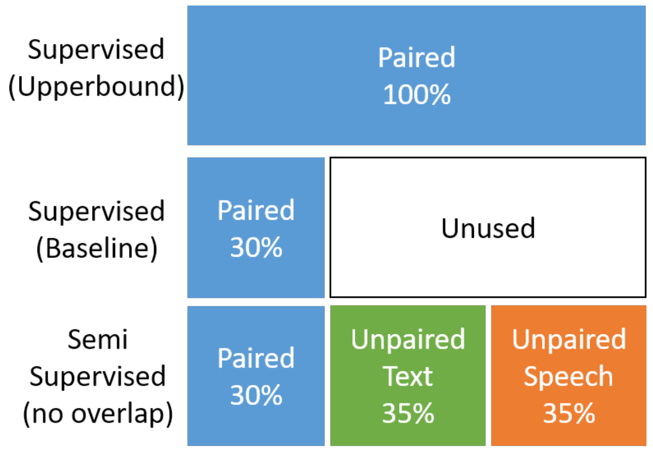


Figure 3.2. Illustration for training data split between three different scenarios.

3.2.1 Feature Extraction

For the speech features, we extracted two different sets of features: Mel spectrogram and magnitude spectrogram. Both the Mel spectrogram and magnitude spectrogram are extracted based on STFT with the librosa package [40]. All speech waveform were sampled at 16 kHz. Given the raw speech waveform, we applied pre-emphasis (coefficient 0.97) and extracted the spectrogram with STFT (50-ms frame length, 12.5-ms frame shift, 2048-points FFT). After getting the spectrogram, we applied absolute and log operation to extract the log magnitude spectrogram features. To generate the Mel spectrogram features, we extracted the 80-dims Mel-scale coefficient from the magnitude spectrogram followed by log operation. Our final set is comprised of an 80-dimension log-Mel spectrogram and 1025-dimension log magnitude spectrogram. The log magnitude spectrogram features are used by TTS and the log-Mel spectrogram features are used by both TTS and ASR.

For the text, we converted all of the sentences into lowercase and replaced some punctuation marks (for example, ” into ’). In the end, we have 26 letters (a-z), six punctuation marks (,:’?.-), and three special tags (<s>, </s>, <spc>) to denote the start, end of sentence, and spaces between words.

3.2.2 Model Details

Our ASR model is an encoder-decoder with an attention mechanism. On the encoder side, we used a log-Mel spectrogram as the input features (in unsupervised process, the log-Mel spectrogram was generated by TTS), which are projected by a fully connected layer and a LeakyReLU ($l = 1e - 2$) [41] activation function, and processed by three stacked bidirectional LSTM (BiLSTM) layers with 256 hidden units for each direction (512 hidden units). We applied sequence subsampling [33, 31] to reduce the memory usage and computation time on the each LSTM layer and reduced the length of the speech features eight times shorter. On the decoder side, the input character is projected with a 128-dims embedding layer and fed into a one-layer LSTM with 512 hidden units. We calculated the attention matrix with an MLP scorer (Eq. 2.6) followed by a fully connected layer and a softmax function. In the decoding phase, the transcription was generated by beam-search decoding (size = 5), and we normalized the log-likelihood score by dividing it by its own length to prevent the decoder from favoring the shorter transcriptions. We did not use any language model or lexicon dictionary in this work.

Our TTS model hyperparameters are generally the same as the original Tacotron, except that we used LeakyReLU instead of ReLU for most of the parts. On the encoder sides, the CBHG used $K = 8$ different filter banks instead of 16 to reduce our GPU memory consumption. For the decoder sides, we used a two-stacked LSTM instead of a GRU with 256 hidden units. Our TTS predicted four consecutive frames in one time-step to reduce the number of time-steps in the decoding process.

Both the ASR and TTS models are implemented with the PyTorch library [1].

3.2.3 Experiment Results

For the ASR, we compare the character error rate (CER) between different scenarios in Table 3.1. For the TTS experiment, we did both objective and subjective evaluations. In the objective evaluation, we compare the L2-norm squared between the predicted and ground truth log Mel-spectrogram in Table 3.2. We experimented with a different ratio between the paired and unpaired data from the LJSpeech dataset. In the subjective evaluation, based on the quality of the synthesized speech using mean opinion score (MOS) test based on five-point scale (5: very good - 1: very poor). We compare three systems: 1) baseline with paired speech-text 30%, 2) speech chain with paired speech-text 30%, unpaired text 35% and unpaired speech 35 (no overlap)% and 3) upperbound paired speech-text 100%. To generate the samples, we randomly picked 20 utterances from the test set. In total, we have 27 subjects and each subject evaluates 60 utterances. We report the subjective evaluation result in Figure 3.4.

The results show that after the ASR and TTS models are trained with a small paired dataset, they start to teach each other using unpaired data and generate useful feedback. Here, we improved both the ASR and TTS performance significantly compared to only using a portion of the paired dataset. We provided some samples from the single speaker speech chain TTS experiments on https://speech-chain-single-spk-demo.netlify.com/.

3.3. Discussion

In this section, we presented a basic speech chain mechanism and demonstrated the ability to train both ASR and TTS modules with paired and unpaired speech and a text dataset. However, there is a limitation in the unpaired training:

1. For training unpaired text, given an unpaired text, we can only generate speech with a specific speaking style. The speaking style is limited based on the speaker set that we used in the supervised TTS training.

[1] PyTorch `https://github.com/pytorch/pytorch`

Table 3.1. ASR experiment result for LJSpeech single-speaker natural speech dataset.

Model	Paired	Unpaired		CER (%)
		Text	Speech	
Supervised (Baseline)				
Enc-Dec Att	10%	-	-	31.7
Enc-Dec Att	20%	-	-	9.9
Enc-Dec Att	30%	-	-	6.8
Enc-Dec Att	40%	-	-	4.9
Enc-Dec Att	50%	-	-	4.1
Semi-supervised (Speech Chain)				
Enc-Dec Att	10%	45%	45%	12.3
Enc-Dec Att	20%	40%	40%	5.6
Enc-Dec Att	30%	35%	35%	4.7
Enc-Dec Att	40%	30%	30%	3.8
Enc-Dec Att	50%	25%	25%	3.5
Supervised (Upperbound)				
Enc-Dec Att	100%	-	-	3.1

2. In the unpaired speech training, the ASR transcribes a sentence. However, our TTS can only reconstruct the speech if the speaker identity from the unpaired speech is provided and the speaker embedding for that person has been seen during the supervised training.

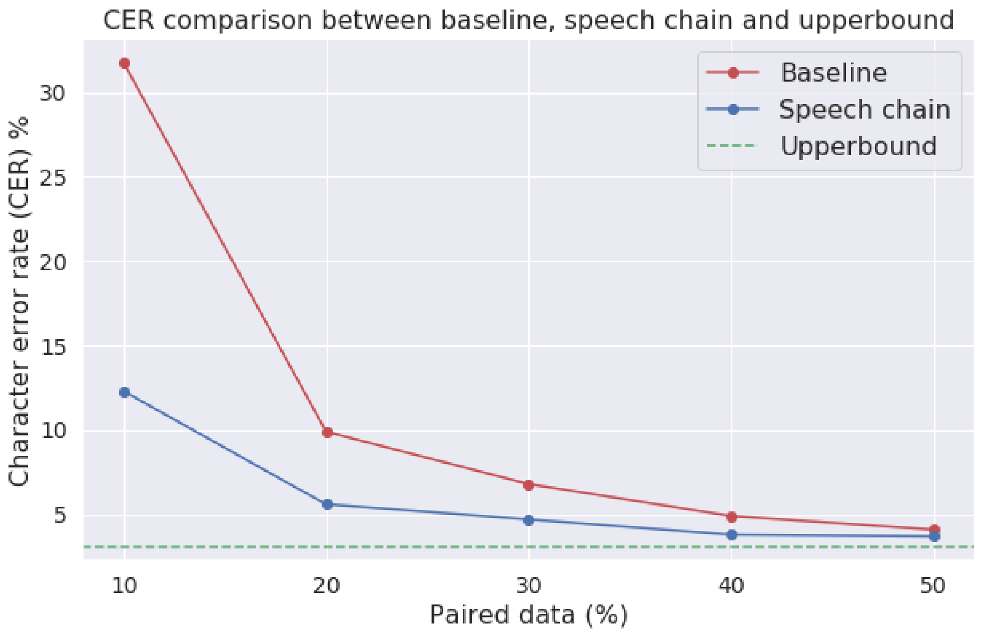


Figure 3.3. Side-by-side CER (%) comparison between baseline, speech chain and upperbound with different percentage of paired speech-text data.

Table 3.2. TTS experiment result for LJSpeech single-speaker natural speech dataset.

Supervised (Baseline)				
Model	Paired	Unpaired		L2-norm^2
		Text	Speech	
Enc-Dec Att	10%	-	-	1.05
Enc-Dec Att	20%	-	-	0.91
Enc-Dec Att	30%	-	-	0.71
Enc-Dec Att	40%	-	-	0.69
Enc-Dec Att	50%	-	-	0.66
Semi-supervised (Speech Chain)				
Enc-Dec Att	10%	45%	45%	0.87
Enc-Dec Att	20%	40%	40%	0.73
Enc-Dec Att	30%	35%	35%	0.66
Enc-Dec Att	40%	30%	30%	0.65
Enc-Dec Att	50%	25%	25%	0.64
Supervised (Upperbound)				
Enc-Dec Att	100%	-	-	0.606

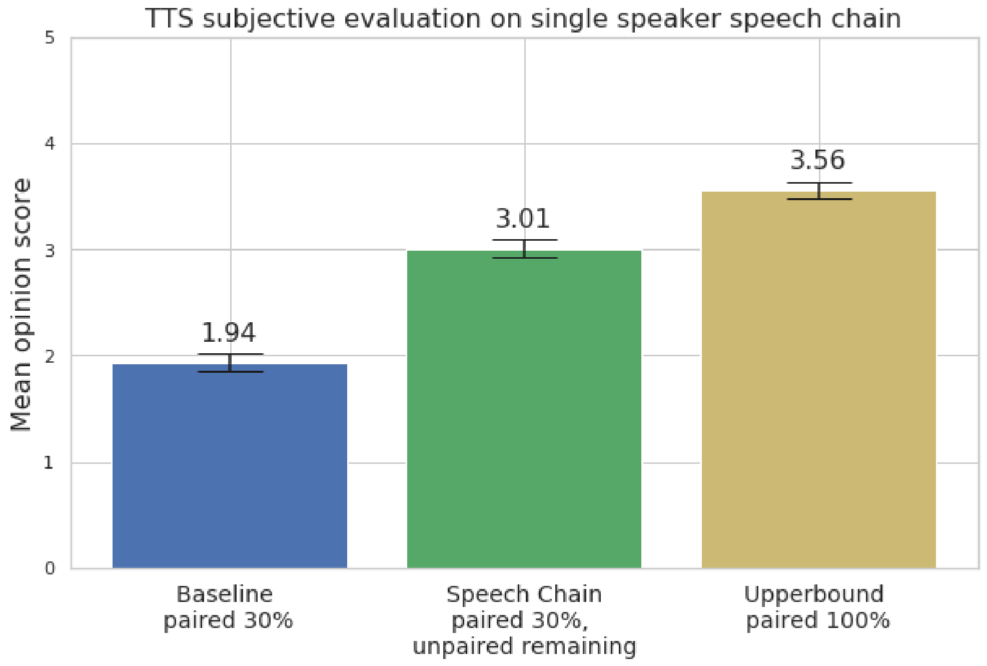


Figure 3.4. MOS with 95% confidence interval between baseline, speech chain and upperbound scenario.

Chapter 4

Multispeaker Machine Speech Chain with One-shot Speaker Adaptation

4.1. Overview

Figure 4.1 illustrates the updated speech chain mechanism. Similar to the earlier version, it consists of a sequence-to-sequence ASR [42, 31], a sequence-to-sequence TTS [25], and a loop connection from ASR to TTS and from TTS to ASR. The key idea is to jointly train the ASR and TTS models. The difference is that, in this version, we integrate a speaker recognition (SPKREC) model inside the loop illustrated in Fig. 4.1(a). As mentioned above, we can train our model on the concatenation of both labeled (paired) and unlabeled (unpaired) data. We describe the learning process below.

1. **Paired speech-text dataset (see Fig. 4.1(a))** Given the speech utterances $\mathbf{x}$ and the corresponding text transcription $\mathbf{y}$ from dataset $\mathcal{D}^P$, both the ASR and TTS models can be trained independently. Here, we can train ASR by calculating the ASR loss ℓ^P_{ASR} directly with teacher forcing. For TTS training, we generate a speaker-embedding vector $z = \text{SPKREC}(\mathbf{x})$, integrate z information with TTS, and calculate the TTS loss ℓ^P_{TTS} via teacher forcing.

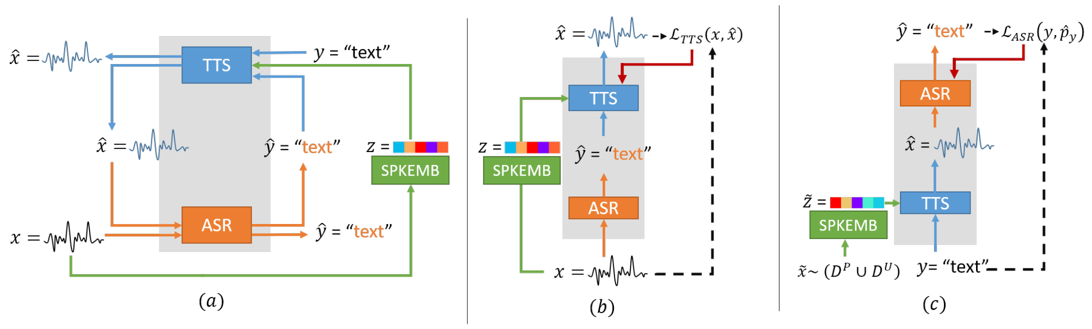


Figure 4.1. (a) Overview of proposed machine speech chain architecture with speaker recognition; (b) Unrolled process with only speech utterances and no text transcription (speech → [**ASR,SPKREC**] → [text + speaker vector] → **TTS** → speech); (c) Unrolled process with only text, but no corresponding speech utterance ([text + speaker vector by sampling **SPKREC**] → **TTS** → speech → **ASR** → text). Note: grayed box is the original speech chain mechanism.

2. **Unpaired speech data only (see Fig. 4.1(b))** Given only the speech utterances $\mathbf{x}$ from unpaired dataset $\mathcal{D}^U$, ASR generates the text transcription $\hat{\mathbf{y}}$ (with greedy or beam-search decoding) and SPKREC provides a speaker-embedding vector $z = \text{SPKREC}(\mathbf{x})$. Given the generated text and the original speaker vector z, TTS then reconstructs the speech waveform $\hat{\mathbf{x}} = TTS(\hat{\mathbf{y}}, z)$ via teacher forcing. We then calculate the loss ℓ^U_{TTS} between $\mathbf{x}$ and $\hat{\mathbf{x}}$.

3. **Unpaired text data only (see Fig. 4.1(c))** Given only the text transcription $\mathbf{y}$ from unpaired dataset $\mathcal{D}^U$, we need to sample speech from the available dataset $\tilde{\mathbf{x}} \sim (\mathcal{D}^P \cup \mathcal{D}^U)$ and generate a random speaker vector $\tilde{z} = \text{SPKREC}(\tilde{\mathbf{x}})$ from SPKREC. Then, TTS generates the speech utterance $\hat{\mathbf{x}}$ with greedy decoding. Given the generated speech $\hat{\mathbf{x}}$, ASR reconstructs the text $\hat{\mathbf{y}} = ASR(\hat{\mathbf{x}})$ via teacher forcing. We then calculate the loss ℓ^U_{ASR} between $\mathbf{y}$ and $\hat{\mathbf{y}}$.

We combine all losses together and update both the ASR and TTS model:

$$\ell = \alpha * (\ell^{P}_{ASR} + \ell^{P}_{TTS}) + \beta * (\ell^{U}_{ASR} + \ell^{U}_{TTS}) \tag{4.1}$$

$$\theta_{ASR} \leftarrow Optim(\theta_{ASR}, \nabla_{\theta_{ASR}}\ell) \tag{4.2}$$

$$\theta_{TTS} \leftarrow Optim(\theta_{TTS}, \nabla_{\theta_{TTS}}\ell), \tag{4.3}$$

where α, β are hyperparameters to scale the loss between the supervised (paired) and unsupervised (unpaired) loss, and $\nabla_{\theta_{ASR}}\ell$, $\nabla_{\theta_{TTS}}\ell$ are the gradient of combined loss ℓ w.r.t. ASR θ_{ASR} and TTS parameters θ_{TTS}.

4.2. Speaker Recognition and Embedding

Speaker recognition is a task to determine the identity of the speaker based on a spoken utterances. Another related tasks to speaker recognition is speaker identification, where the speaker identification model needs to predict if a pair of speech are come from same identity or not. By generating a embedding that correspond to the speaker identity, it can be used to predict both tasks. There are several traditional methods for speaker recognition such as i-vectors [43] and PLDA-based approach [44]. Since the deep learning approach become more popular, several deep learning architectures (DeepSpeaker [45], [46]) have been proposed to directly learn speaker representation from speech features. In Figure 4.2 we illustrate DeepSpeaker architecture in more details.

To generate a speaker representation for speaker recognition task, we assume our input is a speech feature $\mathbf{x} \in \mathbb{R}^{S \times d_{in}}$. Then, we construct a deep neural network by stacking convolution, recurrent, pooling, etc and generate a fixed size vector $z \in \mathbb{R}^{d_z}$. On the top of d_z, we attach a linear projection and softmax activation function to calculate the probability along all possible N speakers.

$$z = \text{SPKEMB}(\mathbf{x}) \tag{4.4}$$

$$p_y = \text{Softmax}(zW_z) \tag{4.5}$$

To optimize the speaker representation model, there are several loss functions such as negative log-likehood:

$$\ell_{NLL} = -\sum_{n=1}^{N} \mathbb{1}(y = n) * \log p_y[n], \tag{4.6}$$

or distance-based such as triplet loss [47, 48]:

$$\ell_{TRI} = \sum_{\substack{a,p,n \\ y_a = y_p \neq y_n}} \max(\|z_a - z_p\|_2^2 + \|z_a - z_n\|_2^2, 0) \tag{4.7}$$

where a, p, n are the anchor, positive and negative example and z_a, z_p, z_n are their embedding respectively. In the training stage, those losses could be combined together and improved the final model performance [45].

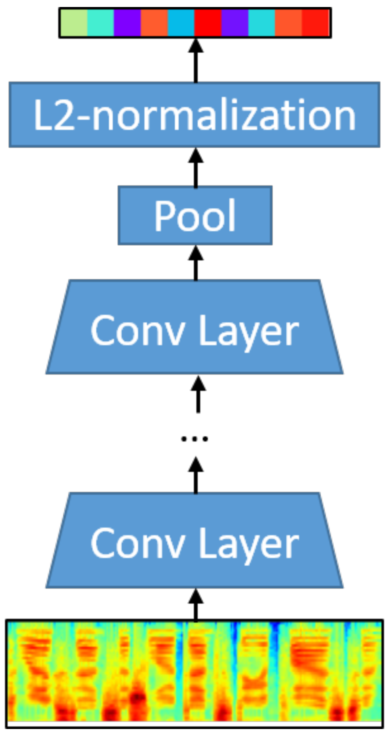


Figure 4.2. Deep learning based speaker embedding (DeepSpeaker) architecture.

4.3. Sequence-to-Sequence TTS with One-shot Speaker Adaptation

A parametric TTS can be formulated as a sequence-to-sequence model where the source sequence is a text utterance $\mathbf{y} = [y_1, .., y_T]$ with length T and the target sequence is a speech feature $\mathbf{x} = [x_1, .., x_S]$ with length S. Our model objective is to maximize $P(\mathbf{x}|\mathbf{y};\theta_{TTS})$ w.r.t. TTS parameter θ_{TTS}. We build our model upon the basic structure of the "Tacotron" TTS [25] and "Deep Speaker" [45] models.

The original Tacotron is a single-speaker TTS system based on a sequence-to-sequence model. Given a text utterance, Tacotron produces the Mel spectrogram and linear spectrogram followed by the Griffin-Lim algorithm to recover the phase and reconstruct the speech signal. However, the original model is not designed to incorporate speaker identity or to generate speech from different speakers.

On the other hand, Deep Speaker is a deep neural speaker-embedding system (here denoted as "SPKEMB"). Given a sequence of speech features $\mathbf{x} = [x_1, .., x_S]$, Deep Speaker generates an L2-normalized continuous vector embedding z. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are spoken by the same speaker, the trained Deep Speaker model will produce the vector $z_1 = \text{SPKEMB}(\mathbf{x}_1)$ and the vector $z_2 = \text{SPKEMB}(\mathbf{x}_2)$, which are close to each other. Otherwise, the generated embeddings z_1 and z_2 will be far from each other. By combining Tacotron with Deep Speaker, we can do "one-shot" speaker adaptation by conditioning the Tacotron with the generated fixed-size continuous vector z from Deep Speaker with a single speech utterance from any speaker.

Here, we adopt both systems by modifying the original Tacotron TTS model to integrate the Deep Speaker model. Figure 4.3 illustrates our proposed model. From the encoder module, the character embedding maps a sequence of characters into a continuous vector. The continuous vector is then projected by two fully connected (FC) layers with the LReLU[41] function. We pass the results to a CBHG module (1D **C**onvolution **B**ank + **H**ighway + bidirectional **G**RU) with K=8 (1 to 8) different filter sizes. The final output $\mathbf{h}^e = [h_1^e, ..h_T^e]$ from the CBHG module represents high-level information from input text $\mathbf{y}$.

On the decoder side, we have an autoregressive decoder that produces the current output Mel spectrogram $\hat{x}_s^M$ given the previous output x_{s-1}^M, the encoder context vector c_t, and the speaker-embedding vector z. First, at time-step s-th, the previous input x_{s-1}^M is projected by two FC layers with LReLU. Then, to tell our decoder which speaker style will be produced, we feed the corresponding speech utterance and generate speaker-embedding vector $z = SPKEMB(\mathbf{x}^M)$. This speaker embedding z is generated using only one utterance of the target speakers; thus it is called "one-shot" speaker adaptation. After that, we integrate speaker

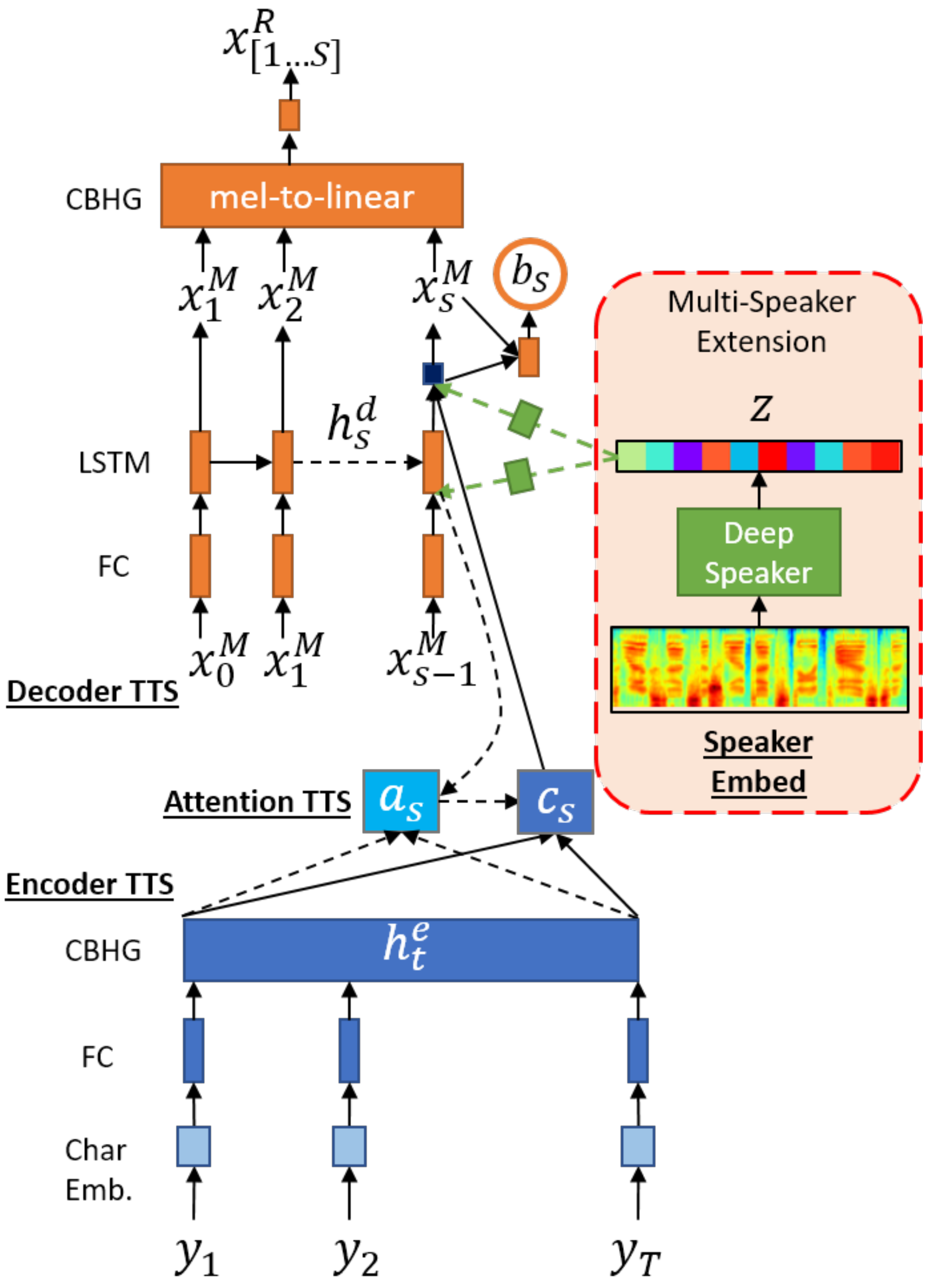


Figure 4.3. Proposed model: sequence-to-sequence TTS (Tacotron) + speaker information via neural speaker embedding (Deep Speaker).

vector z with a linear projection and sum it with the last output from the FC layer. Then, we apply two LSTM layers to generate current decoder state h_s^d. To retrieve the relevant information between the current decoder state and the entire encoder state, we calculate the attention probability $a_s(t) = Align(h_t^e, h_s^d); \forall t \in [1..T]$ and the expected context vector $c_s = \sum_1^T a_s(t) * h_t^e$. Then, we concatenate the decoder state h_s^d, context vector c_s, and projected speaker-embedding z together into a vector, followed by two fully connected layers to produce the current time-step Mel spectrogram output x_s^M. Finally, all predicted outputs of Mel spectrogram $\mathbf{x}^M = [x_1^M, .., x_S^M]$ are projected into a CBHG module to invert the corresponding Mel spectrogram into a linear spectrogram $\mathbf{x}^R = [x_1^R, .., x_S^R]$. Additionally, we have an end-of-speech prediction module to predict when the speech is finished. The end-of-speech prediction module reads the predicted Mel spectrogram $\hat{x}_s^M$ and the context vector c_s, followed by an FC layer and sigmoid function to produce a scalar $b_s \in [0..1]$.

In the training stage, we optimized our proposed model by minimizing the following loss function:

$$\begin{aligned}
\ell_{TTS} &= \mathcal{L}_{TTS}(\mathbf{x}, \hat{\mathbf{x}}, z, \hat{z}) \\
&= \left(\sum_{s=1}^{S} \gamma_1 \left(\|x_s^M - \hat{x}_s^M\|_2^2 + \|x_s^R - \hat{x}_s^R\|_2^2 \right) \right. \\
&\quad \left. -\gamma_2 \left(b_s \log(\hat{b}_s) + (1 - b_s) \log(1 - \hat{b}_s) \right) \right) \\
&\quad +\gamma_3 \left(1 - \frac{\langle \hat{z}, z \rangle}{\|\hat{z}\|_2 \ \|z\|_2} \right),
\end{aligned} \tag{4.8}$$

where $\gamma_1, \gamma_2, \gamma_3$ are our sub-loss hyperparameters, and $\mathbf{x}^M, \mathbf{x}^R, b, z$ are the ground-truth Mel spectrogram, linear spectrogram, and end-of-speech label and speaker-embedding vector from the real speech data, respectively. $\hat{\mathbf{x}}^M, \hat{\mathbf{x}}^R, \hat{b}$ represent the predicted Mel spectrogram, linear spectrogram, and end-of-speech label, respectively, and speaker-embedding vector $\hat{z} = \text{SPKEMB}(\hat{\mathbf{x}}^M)$ is the predicted speaker vector from the Tacotron output. Here, ℓ_{TTS} consists of three different loss formulations: Eq. 4.8 line 1 applies L2-norm squared error between the ground truth and predicted speech as a regression task, Eq. 4.8 line 2 applies binary cross entropy for end-of-speech prediction as a classification task, and Eq. 4.8

line 3 applies cosine distance between the ground-truth speaker-embedding z and predicted speaker-embedding $\hat{z}$, which is the common metric for measuring the similarity between two vectors; furthermore, by minimizing this loss, we also minimize the global loss of speaker style [49, 50].

4.4. Experiment on Multi-speaker Task

4.4.1 Corpus Dataset

In this study, we ran our experiment on the Wall Street Journal (WSJ) CSR Corpus [51]. The complete data are contained in an SI284 (SI84+SI200) dataset. We followed the standard Kaldi [52] s5 recipe to split the training set, development set, and test set. To reformulate the speech chain as a semi-supervised learning method, we prepared SI84 and SI200 as paired and unpaired training sets, respectively. SI84 consists of 7138 utterances (about 16 hours of speech) spoken by 83 speakers, and SI200 consists of 30,180 utterances (about 66 hours) spoken by 200 speakers (without any overlap with speakers of SI84). We use "dev93" to denote the development and "eval92" for the test set.

4.4.2 Feature and Text Representation on WSJ dataset

For the feature extraction, we use a same configuration as Section 3.2.1 The text utterances were tokenized as characters and mapped into a 33-character set: 26 alphabetic letters (a-z), 3 punctuation marks ('.-), and 4 special tags ⟨`noise`⟩, ⟨`spc`⟩,⟨`s`⟩, and ⟨`/s`⟩ as noise, space, start-of-sequence, and end-of-sequence tokens, respectively. Both the ASR input and TTS output shared the same text representation.

4.4.3 Model Details

For the ASR and TTS encoder-decoder, we use a same setting as Section 3.2.2. We set the sub-loss hyperparameter in Eq. 4.8 with $\gamma_1 = 1, \gamma_2 = 1, \gamma_3 = 0.25$.

For the speaker recognition model, we used the Deep Speaker model and followed the original hyperparameters in the original paper. However, our Deep

Table 4.1. Character error rate (CER (%)) comparison between results of supervised learning and those of a semi-supervised learning method, evaluated on *test_eval92* set (without any lexicon & language model on the decoding step)

Model	CER (%)
Supervised training: WSJ *train_si84* (paired) → Baseline	
Att Enc-Dec [54]	17.01
Att Enc-Dec [55]	17.68
Att Enc-Dec (ours)	17.35
Supervised training: WSJ *train_si284* (paired) → Upperbound	
Att Enc-Dec [54]	8.17
Att Enc-Dec [55]	7.69
Att Enc-Dec (ours)	7.12
Semi-supervised training: WSJ *train_si84* (paired) + *train_si200* (unpaired)	
Label propagation (greedy)	17.52
Label propagation (beam=5)	14.58
Proposed speech chain (Sec. 4)	**9.86**

Speaker is only trained on the WSJ SI84 set with 83 unique speakers. Thus, the model is expected to generalize effectively across all remaining unseen speakers to assist the TTS and speech chain training. We used Adam optimization with a learning rate of $5e{-}4$ for the ASR and TTS models and $1e{-}3$ for the Deep Speaker model. All of our models in this manuscript are implemented with PyTorch [53].

4.4.4 Experiment Results

Table 4.1 shows the ASR results from multiple scenarios evaluated on eval92. In the first block, we trained our baseline model by using paired samples from the SI84 set only, and we achieved 17.35% CER. In the second block, we trained our model with paired data of the full WSJ SI284 data, and we achieved 7.12% CER as our upperbound performance. In the last block, we trained our model with a semi-supervised learning approach using SI84 as paired data and SI200 as unpaired data. For comparison with other models trained with semi-supervised

Table 4.2. L2-norm squared on log-Mel spectrogram to compare the supervised learning and those of a semi-supervised learning method, evaluated on *test_eval92* set. Note: We did not include standard Tacotron (without SPKEMB) into the table since it cannot output various target speakers.

Model	**L2-norm^2**
Supervised training: WSJ *train_si84* (paired) $\rightarrow$ Baseline	
Proposed Tacotron (Sec. 4.3) (ours)	1.036
Supervised training: WSJ *train_si284* (paired) $\rightarrow$ Upperbound	
Proposed Tacotron (Sec. 4.3) (ours)	0.836
Semi-supervised training: WSJ *train_si84* (paired) + *train_si200* (unpaired)	
Proposed speech chain (Sec. 4 + Sec. 4.3)	**0.886**

learning, we carried out label-propagation [56]. Label propagation is a simple way to do semi-supervised learning. First, we train initial model with paired speech-text $\mathcal{D}^P$. The pre-trained model is used to generate the hypothesis from the unpaired speech $\mathcal{X}^U$. Later, we add the unpaired speech and their correspondent hypothesis into training set and treat them as a paired dataset. Our result showed that by using label-propagation with beam-size=5, we successfully reduced the CER to 14.58%. Nevertheless, our proposed speech-chain model could achieve a significant improvement over all baselines (paired only and label-propagation) with 9.89% CER, close to the upperbound results.

For the TTS experiment, we did both objective and subjective evaluations. In the objective evaluation, we calculated the difference with L2-norm squared between ground truth and the predicted log-Mel spectrogram and presented the result on Table 4.2. We observed similar trends with the ASR results, where the semi-supervised training with speech chain method improved significantly over the baseline and close to the upperbound result. In the subjective evaluation, based on the quality of the synthesized speech using mean opinion score (MOS) test based on five-point scale (5: very good - 1: very poor). To generate the samples, we randomly picked 20 utterances from the test set. In total, we have 26 subjects and each subject evaluates 60 utterances. We report the subjective eval-

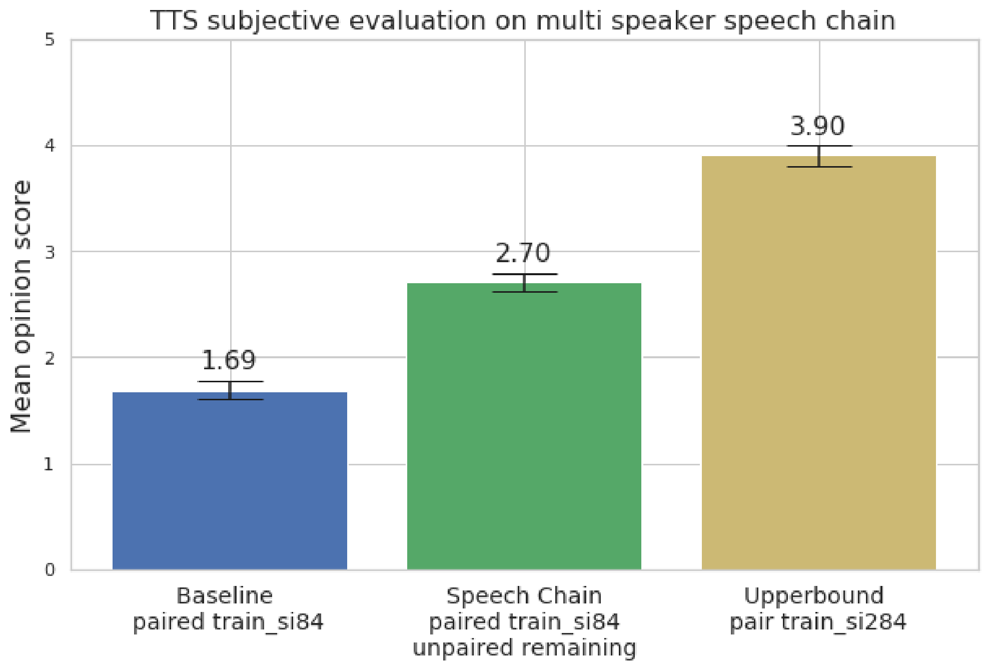


Figure 4.4. MOS with 95% confidence interval between baseline, speech chain and upperbound scenario.

uation result in Figure 4.4. We provided some samples from multi-speaker speech chain TTS experiments on https://speech-chain-multi-spk-demo.netlify.com/.

4.5. Discussion

In this section, we introduced an improved speech chain mechanism by integrating a speaker recognition model inside the loop. By using the new system, we eliminated the downside from our basic speech chain, where we are unable to incorporate the data from unseen speakers. We also extended the capability of TTS to generate speech from an unseen speaker by implementing the one-shot speaker adaptation. Thus, the TTS can generate speech with a similar voice characteristic only with a one-shot speaker example. Inside the speech chain loop, the ASR also gets new data from the combination between a text sentence and an arbitrary voice characteristic. Our results show that after we deployed the speech-chain loop, the ASR system achieved significant improvement compared to the baseline (supervised training only) and other semi-supervised technique (label propagation). Like the trends in ASR, the TTS system also showed improvement compared to the baseline (supervised training only).

However, there is a limitation from the single speaker speech chain (Chapter 3) and multispeaker speech chain. We could not backpropagate the loss from the TTS into the ASR modules because of the output from the ASR model are discrete variables. Therefore, we tried to mitigate this issue in the following chapter.

Chapter 5

End-to-end Feedback Loss on Speech Chain

5.1. Overview

In the speech chain mechanism, given speech features $\mathbf{x} = [x_1, .., x_S]$ (e.g., Mel spectrogram) and text $\mathbf{y} = [y_1, .., y_T]$, we fed the speech to the ASR module and the ASR decoder generated continuous vector h^d_t step by step. To calculate probability vector $\mathbf{p}_y = [p_{y_1}, .., p_{y_T}]$, we applied the softmax function $p_{y_t} = \texttt{softmax}(h^d_t)$ to decoder output h^d_t. For each class probability mass in p_{y_t}, $p_{y_t}[c]$ was defined as:

$$p_{y_t}[c] = \frac{\exp(h^d_t[c]/\tau)}{\sum_{i=1}^{C} \exp(h^d_t[i]/\tau)}, \quad \forall c \in [1..C]. \tag{5.1}$$

Here, C is the total number of classes, $h^d_t \in \mathbb{R}^C$ are the logits produced by the last decoder layer, and τ is the temperature parameters. Setting temperature τ using a larger value ($\tau > 1$) produces a smoother probability mass over classes [57].

For the generation process, we generally have two different methods:

1. Conditional generation given ground truth (teacher forcing):
If we have paired speech and text $(\mathbf{x}, \mathbf{y})$, we can generate p_{y_t} from autoregressive ASR decoder $Dec_{ASR}(y_{t-1}, \mathbf{h}^e)$, conditioned to ground-truth text

y_{t-1} in the current time-step and encoded speech feature $\mathbf{h}^e = Enc_{ASR}(\mathbf{x})$. At the end, the length of probability vector $\mathbf{p}_y$ is fixed to T time-steps.

2. Conditional generation given previous step model prediction:
   Another generation process to decode ASR transcription uses its own prediction to generate probability vector p_{y_t}. There are many different generation methods, such as greedy decoding (1-best beam-search) ($\tilde{y}_t = \text{argmax}_c p_{y_t}[c]$), beam-search, or stochastic sampling ($\tilde{y}_t \sim Cat(p_{y_t})$).

After the generation process, we obtained probability vector $\mathbf{p}_y$ and applied discretization from continuous probability vector p_{y_t} to $\tilde{y}_t$ either by taking the class with the highest probability or sampling from a categorical random variable. After getting a single class to represent the probability vector, we encoded it into vector $[0, 0, .., 1, .., 0]$ with one-hot encoding representation and gave it to the TTS as the encoder input. The TTS reconstructs Mel spectrogram $\hat{\mathbf{x}}$ with the teacher-forcing approach. The reconstruction loss is calculated by:

$$\ell_{TTS}^{rec} = \mathcal{L}_{TTS}^{rec}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{S} \sum_{s=1}^{S} \| x_s^M - \hat{x}_s^M \|_2^2, \tag{5.2}$$

where $\hat{x}_s^M$ is the predicted (or reconstructed) Mel spectrogram and x_s^M is the ground-truth spectrogram at s-th time-step.

We directly calculated the gradient from the reconstruction loss w.r.t. the TTS parameters ($\partial \ell_{TTS}^{rec} / \partial \theta_{TTS}$) because all the operations inside the TTS module are continuous and differentiable. However, we could not calculate the gradient from the reconstruction loss w.r.t. the ASR parameters ($\partial \ell_{TTS}^{rec} / \partial \theta_{ASR}$) because we have a discretization operation from $p_{y_t} \rightarrow \texttt{onehot}(\tilde{y}_t)$. Therefore, we applied a straight-through estimator to enable the loss from ℓ_{TTS}^{rec} to pass through discrete variable $\tilde{y}_t$.

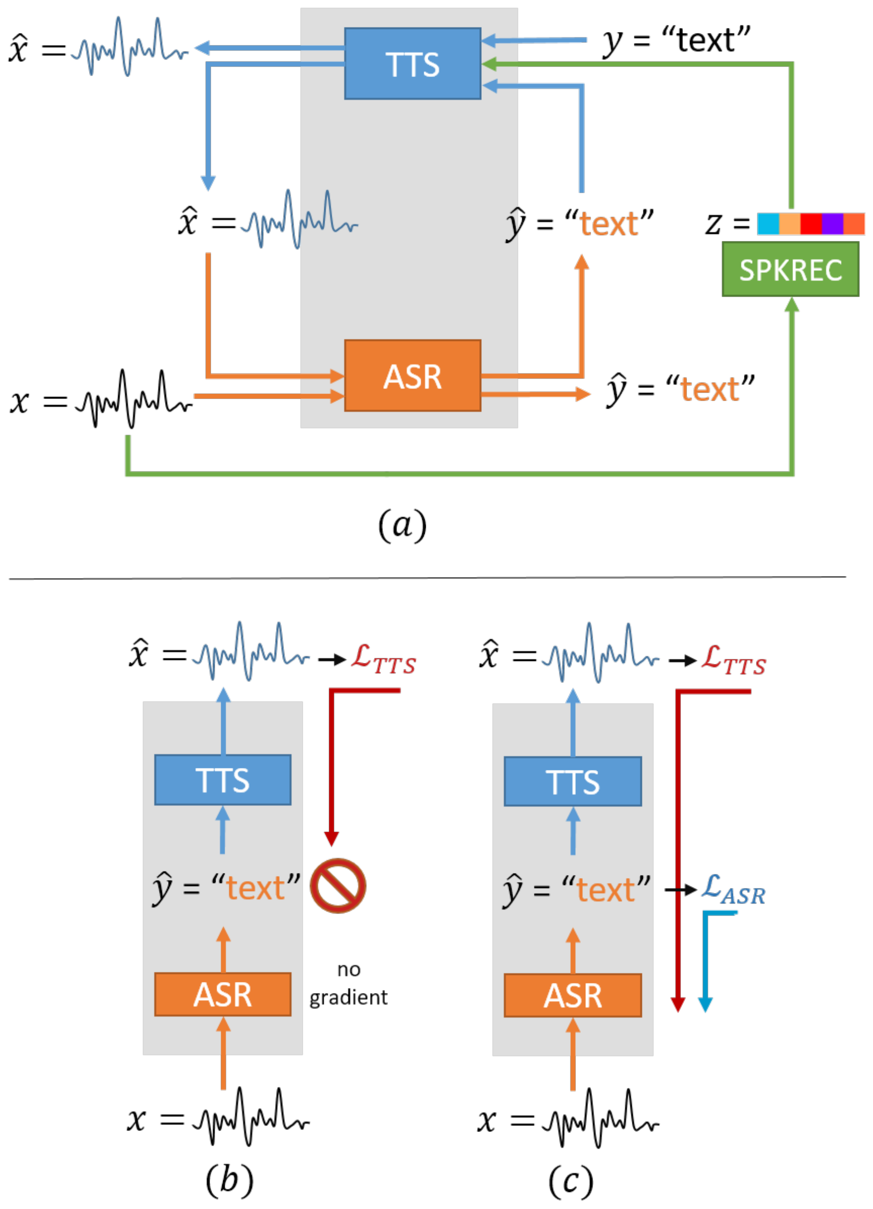


Figure 5.1. a) Multispeaker machine speech chain mechanism; b) Baseline ([3]): feedback loss from TTS is only backpropagated through the TTS module, and the ASR module is not updated because variable $\hat{y}$ is non-differentiable; c) **Proposal:** feedback loss from TTS is backpropagated through discrete variable $\hat{y}$, and ASR modules are updated based on the estimated gradient from the TTS module by a straight-through estimator.

5.2. End-to-End Feedback Loss

5.2.1 Straight-through Argmax

The straight-through estimator [58, 59] is a method for estimating or propagating gradients through stochastic discrete variables. Its main idea is to backpropagate through discrete operations (e.g., $\text{argmax}_c\, p_{y_t}[c]$ or sampling $\tilde{y}_t \sim Cat(p_{y_t})$) like an identity function. We describe the forward process and the gradient calculation with a straight-through estimator in Fig. 5.2.

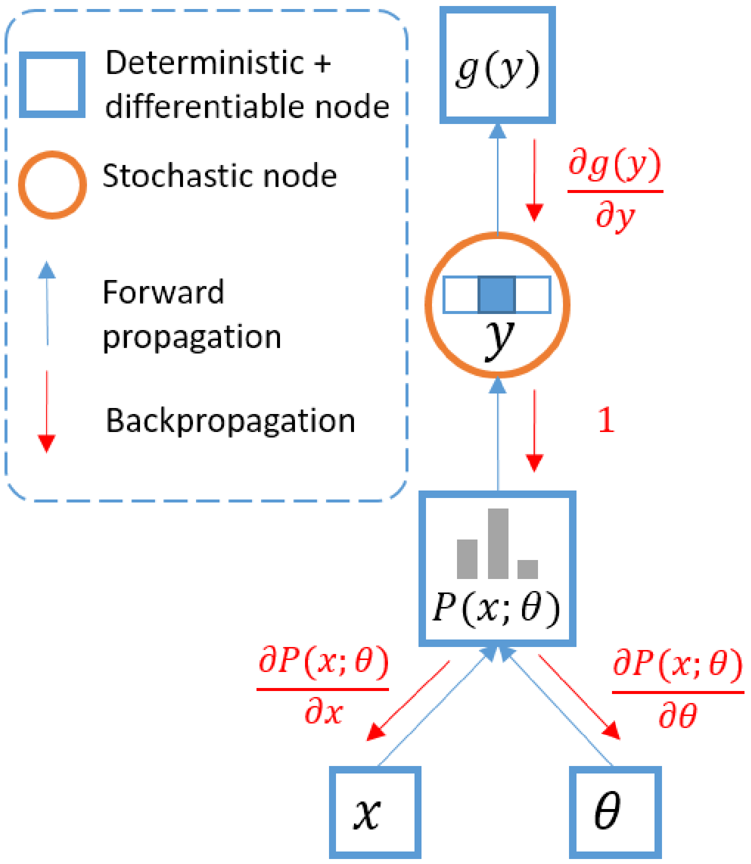

Deterministic + differentiable node
Stochastic node
Forward propagation
Backpropagation
$g(y)$
$\frac{\partial g(y)}{\partial y}$
y
1
$P(x;\theta)$
$\frac{\partial P(x;\theta)}{\partial x}$
$\frac{\partial P(x;\theta)}{\partial \theta}$
x
θ

Figure 5.2. **Straight-through estimator on** $\arg max$ **function**. Given input x and model parameters θ, we calculate categorical probability mass $P(x;\theta)$ and apply discrete operation argmax. In the backward pass, the gradient from stochastic node y to $P(x;\theta)$, $\partial y / \partial P(x;\theta) \approx \mathbb{1}$ is approximated by identity.

In the implementation, we created a function with different forward and backward operations. For the argmax one-hot encoding function, we formulated the forward operation:

$$\tilde{z}_t = \operatorname*{argmax}_c p_{y_t}[c] \tag{5.3}$$

$$\tilde{y}_t = \texttt{onehot}(\tilde{z}_t). \tag{5.4}$$

Here, we describe $\tilde{y}_t$ as a one-hot encoding vector with the same length as the p_{y_t} vector. When the loss is calculated and the gradients are backpropagated from loss ℓ_{TTS}^{rec}, we formulate the backward operation:

$$\frac{\partial \tilde{y}_t}{\partial p_{y_t}} \approx \mathbb{1}. \tag{5.5}$$

Therefore, when we backpropagate the loss from Eq. 5.2 with the straight-through estimator approach, we calculate the TTS reconstruction loss gradient w.r.t. θ_{ASR}:

$$\frac{\partial \ell_{TTS}^{rec}}{\partial \theta_{ASR}} = \sum_{t=1}^{T} \frac{\partial \ell_{TTS}^{rec}}{\partial \tilde{y}_t} \cdot \frac{\partial \tilde{y}_t}{\partial p_{y_t}} \cdot \frac{\partial p_{y_t}}{\partial \theta_{ASR}} \tag{5.6}$$

$$\approx \sum_{t=1}^{T} \frac{\partial \ell_{TTS}^{rec}}{\partial \tilde{y}_t} \cdot \mathbb{1} \cdot \frac{\partial p_{y_t}}{\partial \theta_{ASR}}. \tag{5.7}$$

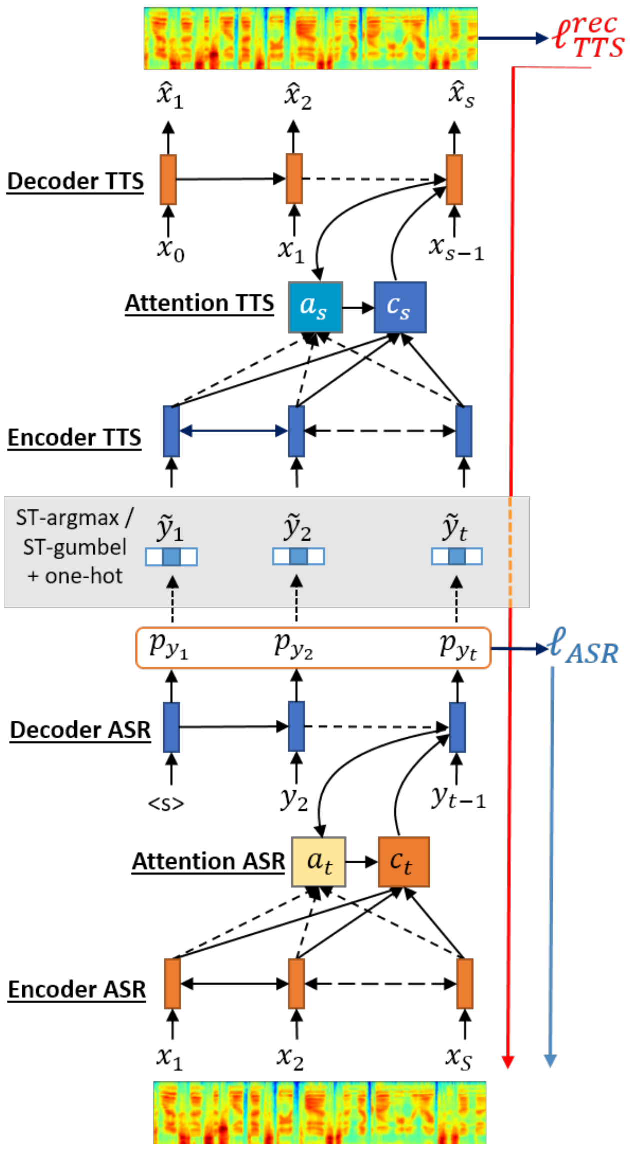


Figure 5.3. Given speech feature $\mathbf{x}$, ASR generates a sequence of probability $\mathbf{p}_y = [p_{y_1}, p_{y_2}, ..., p_{y_T}]$. If we have a ground-truth transcription, we can calculate $\mathcal{L}_{ASR}$ (Eq. 2.9). The TTS module generates speech features and we calculate reconstruction loss $\mathcal{L}^{rec}_{TTS}$ (Eq. 5.2). After that, the gradients based on $\mathcal{L}_{ASR}$ are propagated through the ASR module, and the gradients based on $\mathcal{L}^{rec}_{TTS}$ are propagated through the TTS and ASR modules by a straight-through estimator.

5.2.2 Straight-through Gumbel-Softmax

Besides taking argmax class from probability vector p_{y_t}, we also generated a one-hot encoding by sampling with the Gumbel-Softmax distribution [60, 61]. Gumbel-Softmax is a continuous distribution that approximates categorical samples and the gradients can be calculated with a reparameterization trick. For Gumbel-Softmax, we replaced the softmax formula for calculating p_{y_t} (Eq. 5.1):

$$p_{y_t}[c] = \frac{\exp((h_t^d[c] + g_c)/\tau)}{\sum_{i=1}^{C} \exp((h_t^d[i] + g_i)/\tau)}, \quad \forall c \in [1..C], \tag{5.8}$$

where $g_1, .., g_C$ are i.i.d. samples drawn from Gumbel (0, 1) and τ is the temperature. We sample g_c by drawing samples from the uniform distribution:

$$u_c \sim \texttt{Uniform}(0, 1) \tag{5.9}$$

$$g_c = -\log(-\log(u_c)), \quad \forall c \in [1..C]. \tag{5.10}$$

To generate a one-hot encoding, we define our forward operation:

$$\tilde{z}_t = \text{argmax}_c \, p_{y_t}[c] \tag{5.11}$$

$$\tilde{y}_t = \texttt{onehot}(\tilde{z}_t). \tag{5.12}$$

At the backpropagation time, we use the same straight-through estimator (Eq. 5.5) to allow the gradients to flow through the discrete operation.

5.2.3 Combined Loss for ASR

Our final loss function for ASR is a combination from negative likelihood (Eq. 2.9) and TTS reconstruction loss (Eq. 5.2) by sum operation:

$$\ell_{ASR}^{F} = \ell_{ASR} + \ell_{TTS}^{rec}. \tag{5.13}$$

To summarize our explanation in this section, we provide an illustration in Fig. 5.3 that explains how sub-losses ℓ_{ASR} and ℓ_{TTS}^{rec} are backpropagated to the rest of the ASR and TTS modules.

5.3. Experiment on Multi-speaker Task in Supervised Settings

5.3.1 Dataset

We evaluated the performance of our proposed method on the WSJ dataset [51]. Our settings for the training, development, and test sets are the same as the Kaldi s5 recipe [52]. We trained our model with WSJ-SI284 data. Our validation set was `dev_93` and our test set was `eval_92`. For the feature extraction and text tokenization, we use the same setting as Section 4.4.2.

5.3.2 Model Details

For the ASR model, we used a standard sequence-to-sequence model with an attention module (Section 2.3). We use the same encoder setting as Section 3.2.2. On the decoder sides, we projected one-hot encoding from the previous character into a 256-dims continuous vector with an embedding matrix, followed by one unidirectional LSTM with 512 hidden units. For the attention module, we used the content-based attention + multiscale alignment (denoted as "Att MLP-MA") [37] with a 1-history size. In the evaluation stage, the transcription was generated by beam-search decoding (size = 5), and we normalized the log-likelihood score by dividing it by its own length to prevent the decoder from favoring shorter transcriptions. We did not use any language model or lexicon dictionary in this work. In the training stage, we tried ST-argmax (Section 5.2.1) and ST-Gumbel softmax (Section 5.2.2). We also tried both teacher forcing and greedy decoding to generate ASR probability vectors $\mathbf{p}_y$ in the training stage. For each scenario, we treated temperature $\tau = [0.25, 0.5, 1, 2]$ as our hyperparameter and searched for the best temperature based on the CER on the development set.

For the TTS model, we used the modified Tacotron which is explained in Section 2.4 and we use same settings as Section 3.2.2.

Table 5.1. ASR experiment result on WSJ dataset test_eval92.

Baseline (ℓ_{ASR})			
Model			**CER (%)**
Att MLP [54]			11.08
Att MLP + Location [54]			8.17
Att MLP [55]			7.12
Att MLP-MA (ours) [37]			6.43
Proposed ($\ell_{ASR} + \ell_{TTS}^{rec}$)			
Model	**p_{y_t} generation**	**ST**	**CER (%)**
Att MLP-MA	Teacher forcing	argmax	5.75
Att MLP-MA		gumbel	**5.7**
Att MLP-MA	Greedy	argmax	5.84
Att MLP-MA		gumbel	5.88

5.3.3 Experiment Results

For our baseline, we trained an encoder-decoder with MLP + multiscale alignment with a 1-history size [37]. We also added several published results to our baseline. All of the baseline models were trained by minimizing negative log-likelihood ℓ_{ASR} (Eq. 2.9).

All the models in the proposed section were trained with a combination from two losses, $\ell_{ASR} + \ell_{TTS}^{rec}$, and the ASR parameters were updated based on the gradient from the sum of the two losses. We have four different scenarios, most of which provide significant improvement compared to the baseline model that is only trained on $\mathcal{L}_{ASR}$ loss. With teacher forcing and sampling from Gumbel-softmax, we obtained 11% relative improvement compared to our best baseline Att MLP-MA.

5.4. Discussion

In this chapter, we trained our ASR module by adding feedback from the TTS reconstruction loss. However, the ASR output is not differentiable because of the transcription generated by the discretization process. To address this problem, we used a straight-through estimator to enable the gradient from the TTS module to flow through discrete variables. We tried various scenarios with different decoding and discretization processes. From our experimental results, with teacher-forcing and sampling from Gumbel-Softmax, we improved the ASR performances by 11% relative CER reduction compared to our baseline.

Chapter 6

Improving End-to-End ASR via Reinforcement Learning

In this chapter, I will discuss about my additional research contribution on speech recognition. We revisit the problem of discrepancy between training and inference stage on sequence-to-sequence ASR (as we mentioned in Section 2.2).

6.1. Introduction

End-to-end sequence models are typically composed of three different components: encoder, decoder, and attention. The encoder part extracts features from the source sequence. The decoder part forms an autoregressive model, which conditionally generates the target sequence step-by-step based on the previous output, current state, and encoder features. The attention part is used to calculate the relevance between the current decoder state and encoder features. For training an autoregressive decoder model, the most popular approach is by using teacher forcing [62]. In teacher forcing, the decoder generates output prediction by using the ground-truth input for current time-step. However, in the inference stage, the decoder has no access to the ground-truth transcription. The decoder needs to rely on its own previous prediction as to the input. As the decoding steps going further, any mistakes from the decoder might be accumulated into the future and the predicted target sequence are diverging from the optimal solution.

Besides the difference between the generation method, the mismatch between the objective in the training and the metric for evaluation could also been problematic [63, 64]. In the training stage, the probability predicted by teacher forcing are trained via maximum likelihood estimation (MLE). Therefore, the loss are usually calculated based on the log-probability for each time-step. However, a models are usually evaluated with different objective or metric such as Levenshtein distance for speech recognition and BLEU [65] for machine translation. Therefore, optimizing the model parameters with the correct metric is necessary to obtain its best performance in the inference stage.

Here, we introduce an alternative method for optimizing the ASR model by utilizing the concept from reinforcement learning (RL). To be more precise, we apply one of the RL methods called a policy gradient (REINFORCE) [66] to solve the problem arising from teacher forcing and MLE objective. We assume the ASR autoregressive decoder as an RL agent that produces an action for each time-step, thus we could (1) generate the target sequence transcription with the model's own prediction instead of teacher forcing, thus simulates the prediction in the inference stage, and (2) construct a reward function that is highly correlated with Levenshtein distance and maximize the expected reward with respect to the agent. By incorporating the RL method for optimizing our model, the model is still able to be trained end-to-end and also optimized exactly towards ASR evaluation metric.

6.2. Related Works

Reinforcement learning is one of important types of machine learning where an agent that interacts with its environment learns how to maximize the rewards using feedback signals. Reinforcement learning have been successfully applied in many applications, including building an agent that can learn how to behave in environment and play a game without having any explicit knowledge [67, 68], control tasks in robotics [69], and dialogue system agents [70, 71].

Not limited to those areas, reinforcement learning has also been adopted for improving end-to-end deep learning architecture. To date, Ranzato et al. [63] proposed to combine REINFORCE with an MLE training objective called MIXER. In the early stage of training, the first s steps were trained with MLE and the remaining T-s steps with REINFORCE. They decreased s as the training progress over time. By using REINFORCE, they trained the model using non-differentiable task-related rewards (e.g., BLEU for machine translation). In this manuscript, we did not need to deal with any scheduling or mix any sampling with the teacher-forcing ground-truth. Furthermore, MIXER did not sample multiple sequences based on the REINFORCE Monte Carlo approximation.

In machine translation tasks, Shen et al. [72] could improve the neural machine translation (NMT) model using Minimum Risk Training (MRT). A Google NMT [64] system combined MLE and MRT objectives to achieve better results. Zhang et al. [73] also points out the gap between training and inference, they address this issue by sampling context words not only from the ground truth sequence but from the model prediction during training. In ASR tasks, Shanon et al. [74] performed WER optimization by sampling paths from the lattices that were used during sMBR training, which seemingly resembles the REINFORCE algorithm. But the work was only applied to a CTC-based model. From a probabilistic perspective, MRT formulation resembles the expected reward formulation used in reinforcement learning. Here, MRT formulation equally distributed the sentence-level loss into all of the time-steps in the sample. To the best of our knowledge, we are the first to publish the work on optimizing attention-based encoder-decoder ASR with reinforcement learning approach [75]. Later on, similar work is also published by Karita et al. [76]. The main difference between our work and their work is the design of the reward function and the sampling process.

6.3. Sequence-to-Sequence ASR

A sequence-to-sequence (seq2seq) is an end-to-end neural network model that map a dynamic length sequence $\mathbf{X} = [x_1, x_2, ..., x_S]$ with length S to another dynamic length sequence $\mathbf{Y} = [y_1, y_2, ..., y_T]$ with length T time-step [2]. In the basic form, seq2seq could be formulated as $P_\theta(\mathbf{Y}|\mathbf{X})$ parameterized by model parameters θ. In ASR case, we build a seq2seq model that generate a text transcription Y (e.g., character or phoneme) given a speech features X (e.g., MFCC or Mel-spectrogram). We show our complete structure for seq2seq ASR in Chapter 2.3.

The decoder task is to generate a target discrete sequence Y:

$$P(Y|X;\theta) = \prod_{t=1}^{T} P(y_t|c_t, h_t^D, y_{t-1};\theta), \tag{6.1}$$

where c_t is the relevant context generated by the attention module. This equation represent an conditional autoregressive model that produces current time-step target probability y_t given the previous time-step output y_{t-1}, a decoder state h_t^D (which consists of a compressed representation for decoder from time 1 to $t-1$) and a context vector c_t.

Training seq2seq model mostly done by using maximum likelihood estimation (MLE):

$$\begin{aligned} \theta^* &= \operatorname*{argmax}_\theta P(Y|X;\theta) \\ &= \operatorname*{argmax}_\theta \prod_{t=1}^{T} P(y_t|c_t, h_t^D, y_{t-1};\theta). \end{aligned} \tag{6.2}$$

Based on the maximum likelihood criterion, we obtained optimal model θ^* by minimizing the negative log-likelihood (NLL) calculated by the teacher-forcing generation method:

$$\begin{aligned} \mathcal{L}_{NLL} &= -\log P(\mathbf{y}|\mathbf{x};\theta), \\ &= -\log \prod_{t=1}^{T} P(y_t|c_t, h_t^D, y_{t-1};\theta), \\ &= -\sum_{t=1}^{T} \log P(y_t|c_t, h_t^D, y_{t-1};\theta). \end{aligned} \tag{6.3}$$

For each time-step, the teacher-forcing approach generates the label probability based on the ground-truth label at time-t. In Fig. 6.1, we illustrate the generation process based on the teacher-forcing method. Loss function NLL is described as follows:

$$NLL(y_t, p(y_t)) = -\sum_c \mathbb{1}\{y_t = c\} \log p(y_t = c), \tag{6.4}$$

where $p(y_t) = P(y_t|c_t, h_t^D, y_{t-1};\theta)$.

However, in the inference stage, since we have no access to the ground-truth transcription, our model must rely on its own previous prediction as input for the current time-step. We illustrated the decoding process with a greedy approach by taking the label index based on the highest probability mass on $p(y_t)$ in Fig. 6.2.

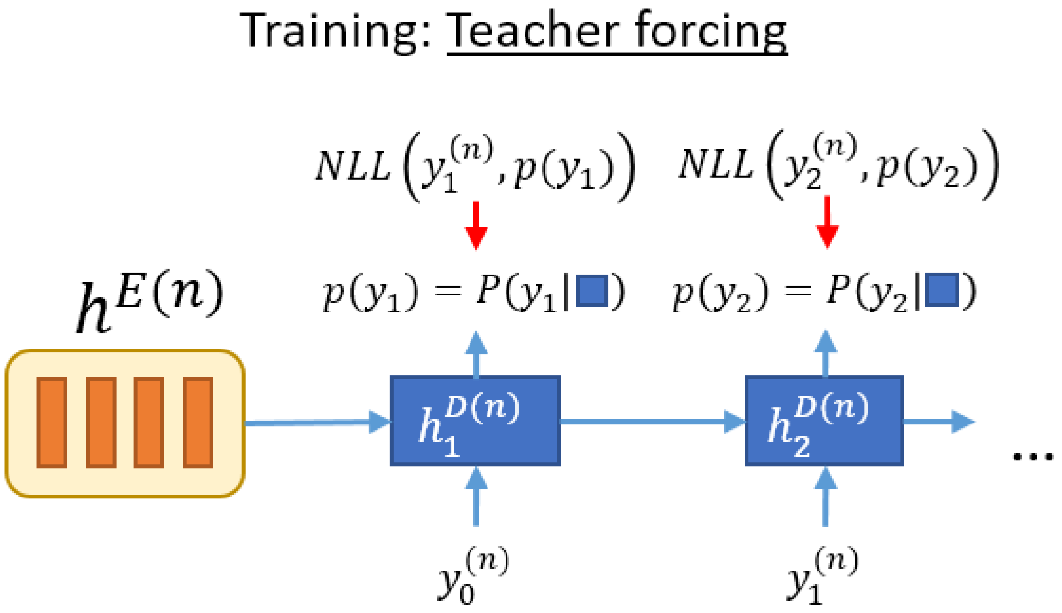


Figure 6.1. Training stage: generation via teacher-forcing method sets the model input with ground-truth transcription. For each time-step, decoder generates probability vector $p(y_t)$, and we calculate negative log-likelihood between $p(y_t)$ and ground-truth $y_t^{(n)}$.

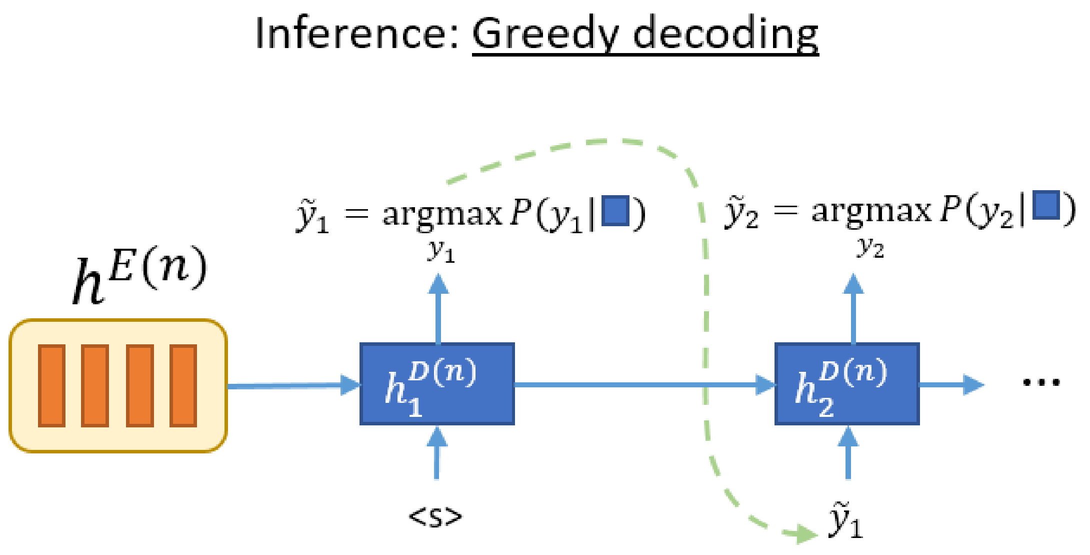

Inference: Greedy decoding
$h^{E(n)}$
$h_1^{D(n)}$
$h_2^{D(n)}$
$\tilde{y}_1 = \operatorname{argmax}_{y_1} P(y_1|\blacksquare)$
$\tilde{y}_2 = \operatorname{argmax}_{y_2} P(y_2|\blacksquare)$
<s>
$\tilde{y}_1$
...

Figure 6.2. Testing/inference stage: decoder doesn't have access to ground-truth transcription. Therefore, for each time-step t, decoder input depends on model prediction from previous time-step $t-1$. For greedy decoding (1-best search), we took the label from highest probability $\tilde{y}_{t-1} = \operatorname*{argmax}_{y_{t-1}} P(y_{t-1}|h_1^{D(n)})$ and use selected label $\tilde{y}_{t-1}$ for current decoder input.

6.4. Reinforcement Learning

In this section, we briefly discuss reinforcement learning, which is an area of machine learning where the agent learns by interacting inside a specific environment. In the learning stage, the agent receives a state and sequentially generates an action through multiple time-steps and eventually the environment returns a reward as a signal feedback for the agent. If agents get a high reward value, it means that they are doing a good job related to their given tasks. Our final goal is to make agents that can choose a series of optimal actions that maximize the reward in that environment.

The RL method can be described formally by the Markov Decision Process (MDP) [77]. Here the agent and environment interact in discrete time-steps $t = [1, 2, .., T]$. We formulate a MDP property as a tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where

- $\mathcal{S} = \{S_1, S_2, .., S_n\}$ is a set of the environment's states and $\forall t \in [1..T], s_t \in \mathcal{S}$;
- $\mathcal{A} = \{A_1, A_2, .., A_m\}$ is a set of possible actions for the agent and $\forall t \in [1..T], a_t \in \mathcal{A}$;
- $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a state transition probability where $\mathcal{P}(s'|s, a)$ is the probability of transitioning to state s' given state s and action a;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function that returns a value given a state and an action.

In Fig. 6.3, we illustrated the interaction between an agent and its environment within MDP notation. The MDP process starts from state s_1 as the initial agent's state. The initial state s_1 is defined by the environment (e.g., s_1 is the location of robot starting point inside certain arena). Based on the initial state s_1, the agent chooses actions $a_1 \in \mathcal{A}$. Given current state $s_1 \in \mathcal{S}$ and selected action a_1, new state s_2 is drawn or generated based on state transition probabilities $s_2 \sim \mathcal{P}(s_2|s_1, a_1)$ where $s_2 \in \mathcal{S}$. We repeat the process and generate a sequence of states and action from time $t \in [1..T]$:

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} ... \xrightarrow{a_{T-2}} s_{T-1} \xrightarrow{a_{T-1}} s_T. \tag{6.5}$$

For each trajectory $s_1, a_1, s_2, a_2, ..$, the environment returns a series of rewards as a signal feedback:

$$\mathcal{R}(s_1, a_1) + \gamma \mathcal{R}(s_2, a_2) + \gamma^2 \mathcal{R}(s_3, a_3) + .., \tag{6.6}$$

where $\gamma \in [0, 1)$ is the discount factor for future rewards. RL's main target is to optimize an agent that chooses the most optimal actions over time to maximize the expected reward:

$$\mathbb{E}_{a_t \sim \pi}[\mathcal{R}(s_1, a_1) + \gamma \mathcal{R}(s_2, a_2) + \gamma^2 \mathcal{R}(s_3, a_3) + ..] \tag{6.7}$$

Policy function $\pi : \mathcal{S} \to \mathcal{A}$ maps a state to an action. Given state s_t, the policy function returns feasible action $a_t = \pi(s_t)$. Value function $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ is defined:

$$V^\pi(s) = \mathbb{E}_{a_t \sim \pi}[\mathcal{R}(s_1, a_1) + \gamma \mathcal{R}(s_2, a_2) + ..|s_1 = s]. \tag{6.8}$$

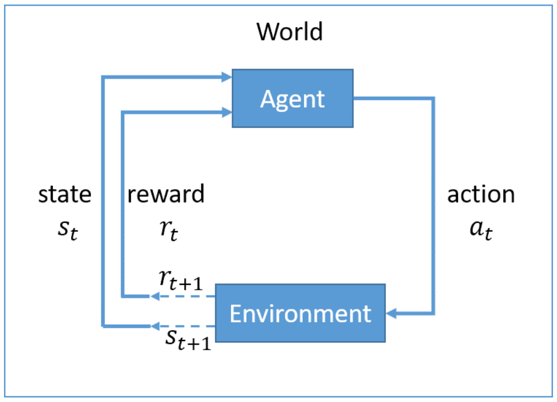


Figure 6.3. Interaction between agent and their environment inside an MDP. Given current state s_t, the agent choose an action a_t. The environment responds to the selected action and generates a new state s_{t+1} and a reward r_{t+1}.

The value function calculates the expected reward given state s and action $a_t \sim \pi$ taken from policy π. We got the following optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{6.9}$$

Given optimal value function $V^*(s)$, optimal policy π^* becomes

$$\pi^* = \operatorname*{argmax}_{\pi} V^*(s) \quad \forall s \in \mathcal{S}. \tag{6.10}$$

To extend the value function, a Q-function predicts the expected reward given state-action pair $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defined:

$$Q^{\pi}(s,a) = \mathbb{E}_{a_t \sim \pi}[\mathcal{R}(s_1,a_1) + \gamma\mathcal{R}(s_2,a_2) + ..|s_1 = s, a_1 = a]. \tag{6.11}$$

The optimal Q-function $Q^*(s,a)$ is the maximum action value-function over policies

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{6.12}$$

We retrieved best policy $\pi^*(s)$ given state s:

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a) \quad \forall s \in \mathcal{S}. \tag{6.13}$$

Reinforcement learning can be solved in several ways. First, we can directly optimize policy function π to maximize the expected reward in Eq. 6.7. Policy gradient [66] is one of the algorithm that optimizes parameterized policy π_θ with respect to the expected reward. Parameterized policy π_θ can also be represented with a neural network and optimized directly by first-order optimization such as stochastic gradient descent (SGD). Second, we can find the optimal policy based on Eq. 6.13 based on the Q-function. Q-learning [78] learns a policy and informs the agent of the expected reward given a certain state and action pair. If we have discrete states and actions, Q-learning can be implemented with a simple table where the state and action pairs are defined by columns and rows and the expected reward value is in the cell. However, when we have high-dimensional states and action spaces, we can replace the table with a function that approximates the Q-function, such as simple linear regression or a deep neural network [79].

6.5. Policy Gradient Training for Sequence-to-Sequence ASR

We present our proposed method to incorporate policy optimization with seq2seq ASR architecture. First, we present an overview about policy gradient (REINFORCE) optimization strategy. Later, we describe several reward functions that we used to optimize our agent in the reinforcement learning environment.

6.5.1 Policy Gradient

Policy gradient is a method based on policy function formulation. The policy $\pi_\theta(a|s)$ optimized directly by adapting the parameters θ to increase the expected reward $E[R_t|\pi_\theta]$ [77]. The parameters θ depends on the function that we use to approximate the policy. Here, we use deep neural network to parameterized the policy function and θ denotes a collections of neural network weight matrices. To bridge the ASR with reinforcement learning optimization, we need to formulate within an MDP tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability between a state to another state, and $\mathcal{R}$ is the reward function.

We define our RL agent as a seq2seq ASR model where the agent function is to predict the transcription given a sequence of speech features. We describe the state $s_t \in \mathcal{S}$ as a temporary state $s_t = [c_t, h_t^D]$ from seq2seq decoder at time $t \in \{1..T\}$. Action state $a_t \in \mathcal{A}$ is the discrete output token from the decoder such as character or phoneme symbols. The transition probability $\mathcal{P}$ are implied by the operation from RNN cell inside the decoder. Lastly, reward function $\mathcal{R}$ are designed to be highly correlated with the quality measure for an ASR system. We provide the detail in Section 6.5.2.

We assume $\left(X^{(n)}, Y^{(n)}\right)$ is a pair between speech features and their groundtruth transcription. The reward $R^{(n)}$ calculated between the groundtruth $Y^{(n)}$ and sampled transcription $\tilde{Y}^{(n,\cdot)}$. We are looking to maximize the expected reward $\mathbb{E}_Y[R^{(n)}|\pi_\theta]$ with respect to seq2seq parameters θ where $\pi_\theta(a_t|s_t) = P(y_t|h_t^{D(n)}, c_t^{(n)}; \theta) = P(y_t|y_{<t}, X^{(n)}; \theta)$. In order to optimize θ, we calculate the expected reward gradient with respect to the parameters θ:

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_Y \left[R^{(n)}|\pi_\theta\right] \\
&= \nabla_\theta \int P(Y|X^{(n)}; \theta) R^{(n)} \, dY \\
&= \int \nabla_\theta P(Y|X^{(n)}; \theta) R^{(n)} \, dY \\
&= \int P(Y|X^{(n)}; \theta) \frac{\nabla_\theta P(Y|X^{(n)}; \theta)}{P(Y|X^{(n)}; \theta)} R^{(n)} \, dY \\
&= \int P(Y|X^{(n)}; \theta) \nabla_\theta \log P(Y|X^{(n)}; \theta) R^{(n)} \, dY \\
&= \mathbb{E}_Y \left[\nabla_\theta \log P(Y|X^{(n)}; \theta) R^{(n)}\right] \\
&\approx \frac{1}{M} \sum_{m=1}^{M} R^{(n,m)} \nabla_\theta \log P(\tilde{Y}^{(n,m)}|X^{(n)}; \theta),
\end{aligned} \tag{6.14}$$

where M is the number of samples, $\tilde{Y}^{(n,m)} \sim P(Y|X^{(n)}; \theta)$ is the m-th sample from model θ conditioned on input $X^{(n)}$, and $R^{(n,m)}$ is the calculated reward between ground-truth $Y^{(n)}$ and sample $\tilde{Y}^{(n,m)}$. From another perspective, Eq. 6.14 is a bit identical with the gradient from Minimum Risk Training (MRT) [72].

Occasionally using only a single reward signal for a whole sequence of sample $\tilde{Y}^(m, n)$ is not sufficient. For example, Eq. 6.14 can be expanded as:

$$\frac{1}{M} \sum_{m=1}^{M} \sum_{m=1}^{M} R^{(n,m)} \nabla_\theta \sum_{t=1}^{T} \log P(\tilde{y}_t^{(n,m)}|X^{(n)}; \theta) \tag{6.15}$$

which is we distribute the sequence reward $R^{(n,m)}$ to all time-step equally. There might be a sub-optimal case where the reward is negative caused by several time-step action, but we penalize all time-step with negative reward instead. Therefore, we could substitute the reward $R^{(n)}$ with time-distributed reward $R_t^{(n)} \in \mathbb{R}, \forall t \in \{1..T\}$. The reward $R_t^{(n)}$ might have different value between different time-step, thus it could provide more informative feedback for each time-step. Mathematically, we substitute Eq. 6.14 $t = [1, .., T]$ with:

$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_Y \left[\sum_{t=1}^{T} R_t^{(n)} | \pi_\theta \right] \\
&= \nabla_\theta \int P(Y|X^{(n)};\theta) \left(\sum_{t=1}^{T} R_t^{(n)} \right) dY \\
&= \int P(Y|X^{(n)};\theta) \frac{\nabla_\theta P(Y|X^{(n)};\theta)}{P(Y|X^{(n)};\theta)} \left(\sum_{t=1}^{T} R_t^{(n)} \right) dY \\
&= \int P(Y|X^{(n)};\theta) \nabla_\theta \log P(Y|X^{(n)};\theta) \left(\sum_{t=1}^{T} R_t^{(n)} \right) dY \\
&= \mathbb{E}_Y \left[\left(\sum_{t=1}^{T} R_t^{(n)} \right) \nabla_\theta \log P(Y|X^{(n)};\theta) \right] \\
&= \mathbb{E}_Y \left[\left(\sum_{t=1}^{T} R_t^{(n)} \right) \sum_{t=1}^{T} \nabla_\theta \log P(y_t|y_{<t}, X^{(n)};\theta) \right] \\
&\approx \mathbb{E}_Y \left[\sum_{t=1}^{T} R_t^{(n)} \nabla_\theta \log P(y_t|y_{<t}, X^{(n)};\theta) \right] && (6.16) \\
&\approx \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T(m)} R_t^{(n,m)} \nabla_\theta \log P(\tilde{y}_t^{(n,m)}|\tilde{y}_{<t}^{(n,m)}, X^{(n)};\theta), && (6.17)
\end{aligned}
$$

where T is the length of transcription Y, $R_t^{(n)}$ is the generalized reward based on the current state and action at time-t. In Eq. 6.17, $R_t^{(n,m)}$ is the reward from m-th sample, time-step t-th and compared with n-th utterance groundtruth, and $T(m)$ denotes the sample $\tilde{Y}^{(n,m)}$ length. To calculate the expected reward from Eq. 6.14 and Eq. 6.17, we need to integrate all possible transcription across random variable Y. It is unrealistic because the search space are growing exponential for each time-step. Therefore, we do Monte-carlo sampling M times per sequence

$\tilde{Y}^{(n,m)} \sim P(Y|X^{(n)};\theta)$ for each utterance $X^{(n)}$ to get an approximated expected reward.

To summarize our explanation, we compared the differences between teacher-forcing and policy gradient loss calculation from Figs. 6.1 and 6.4. In the teacher-forcing method, the model predictions are generated based on the ground-truth transcription. However, in the policy gradient method, first we sample M sequences by Monte Carlo sampling and stop after getting an `</s>` symbol. Then we calculate discounted reward $R_t^{(n,m)}$ for each time-step based on the future rewards. We provide pseudocode to complete our explanation in Alg. 3.

6.5.2 Reward Construction for ASR Tasks

One important component for optimizing an agent using an reinforcement learning approach is to design a good reward function that closely corresponds to the metric that we used to evaluate our agent performance. In our case, our agent is ASR systems that were evaluated based on the edit-distance or the Levenshtein distance algorithm. Therefore, we composed our reward function with a modified edit-distance algorithm and divided the reward into two different types:

Sentence-level reward

Based on Eq. 6.14, we need to calculate the reward by comparing ground-truth transcription $Y^{(n)}$ and sampled transcription $\tilde{Y}^{(n,m)}$. In this case, we designed reward function $\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)})$ to calculate $R^{(n,m)}$:

$$R^{(n,m)} = \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}) = -\frac{ED(\tilde{Y}^{(n,m)}, Y^{(n)})}{|Y^{(n)}|}, \tag{6.18}$$

where $ED(\cdot,\cdot)$ is an edit-distance function. In practice, we would like to minimize the edit-distance between the sample and the ground-truth transcription. However, for the reinforcement learning environment, we design a reward function with the opposite output. For example, if our model produces two samples, $\tilde{Y}^{(n,1)}$ and $\tilde{Y}^{(n,2)}$, the first $\tilde{Y}^{(n,1)}$ is “closer” to $Y^{(n)}$ than the second $\tilde{Y}^{(n,2)}$, then the reward function must fulfill: $\mathcal{R}(\tilde{Y}^{(n,1)}, Y^{(n)}) > \mathcal{R}(\tilde{Y}^{(n,2)}, Y^{(n)})$. Therefore, we multiply the edit-distance result by -1 to fulfill the requirement of the reward function.

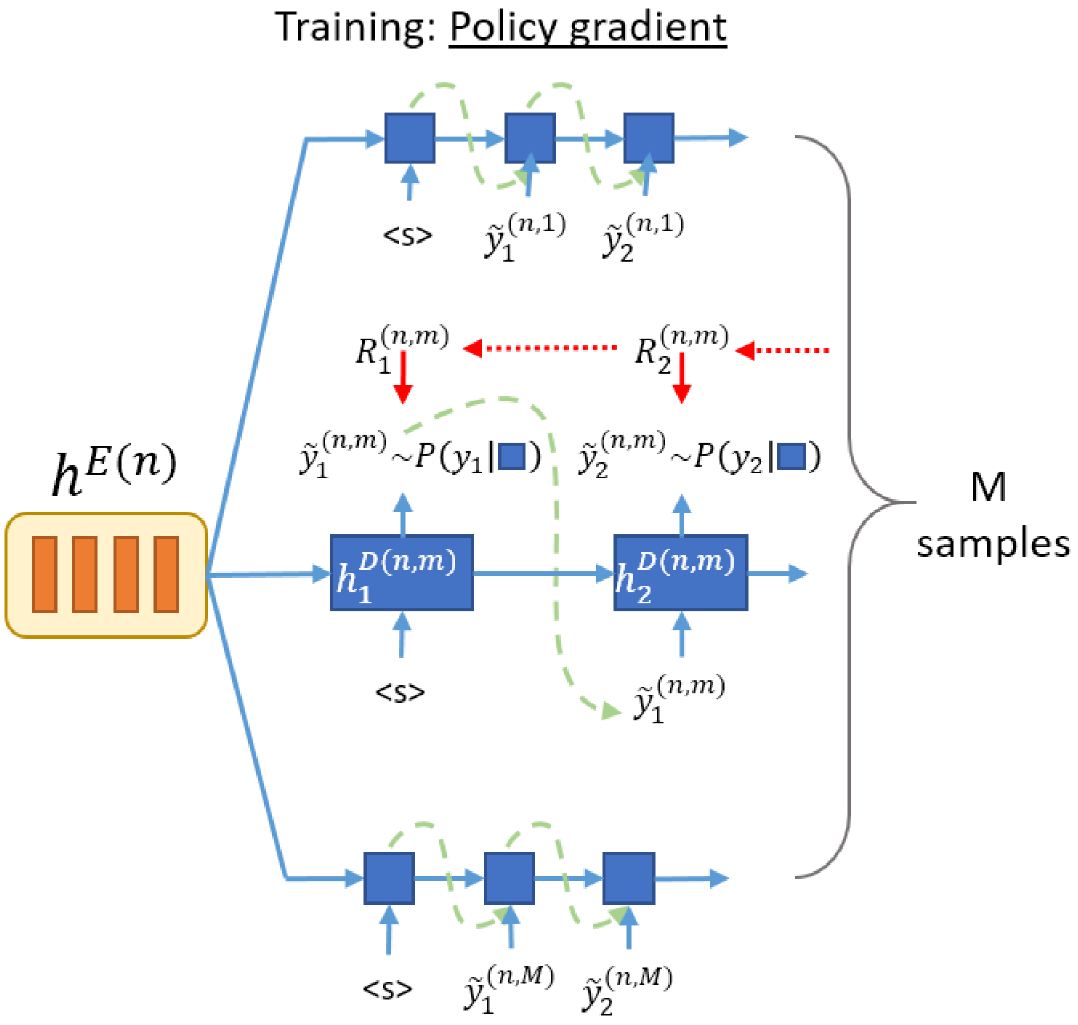


Figure 6.4. Policy gradient set decoder input to be conditioned on its own prediction sampled from previous time-step to predict current time-step output probability. Therefore, decoder doesn't rely on a ground-truth transcription like teacher-forcing method. Expected rewards for model transcription are approximated by the average from multiple sample trajectories.

Algorithm 3 Pseudocode for sampling text on sequence-to-sequence ASR

```
1: procedure SAMPLE(Speech features $\mathbf{x}$, sample size M, vocab size V)
2:     y_in = [<S>,..,<S>] $\in \mathbb{R}^M$          ▷ init with start token <S> $M$ times
3:     l_sample_logp = [[] for _ in [1..M]]
4:     l_sample_act = [[] for _ in [1..M]]
5:     l_sample_len = [-1,..,-1]                        ▷ init sample length
6:     tt = 0
7:     model.encode($x$)                                ▷ encode speech into $h^E$
8:     finished = False
9:     repeat
10:        p_y = model.decode(y_in) $\in \mathbb{R}^{M \times V}$
11:        log_p_y = log(p_y)
12:        for m in [1..M] do
13:            a_y ~ Categorical (p_y[m])       ▷ sample action from Categorical
               distribution
14:            y_in[m] = a_y                            ▷ set next decoder input
15:            if l_sample_len[m] == -1 then
16:                l_sample_logp[m].add(
                       log_p_y[m, a_y]))
17:                l_sample_act[m].add(a_y)
18:                if a_y == <\s> then
19:                    l_sample_len[m] = tt+1
20:                end if
21:            end if
22:        end for
23:        finished = all(l_sample_len $\neq$ -1)
24:    until finished == True                           ▷ all samples meet </s>
25:    return l_sample_logp, l_sample_act
26: end procedure
```

Since the REINFORCE gradient estimator is usually too noisy and might hinder our learning process, there are several tricks to reduce the variance [80, 81]. Here we normalize reward $R^{(n,m)}$:

$$\begin{aligned}
\mu_n &= \frac{1}{M}\sum_{m=1}^{M} \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}) \\
\sigma_n^2 &= \frac{1}{M}\sum_{m=1}^{M} \left(\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}) - \mu_n\right)^2 \\
R^{(n,m)} &= \frac{\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}) - \mu_n}{\sigma_n}.
\end{aligned} \tag{6.19}$$

We normalize our reward across M samples into zero mean and unit variance. We provide the pseudocode for calculating sentence-level reward in Algorithm 4.

Token-level reward

Rather than having only a single reward attributed to the whole sequence, we could also construct a better reward function which give a feedback for every time-step. Here we design a reward function that could provide an intermediate reward before the sample transcription finished. This reward function $\mathcal{R}(\tilde{Y}, Y^{(n)}, t)$ calculate $R_t^{(n)}$ by utilizing the edit-distance algorithm. We define reward $\mathcal{R}(\tilde{Y}, Y^{(n)}, t)$:

$$\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t) = \begin{cases} |Y^{(n)}| - ED(\tilde{Y}_{1:t}^{(n,m)}, Y^{(n)}) & \text{if} \quad t = 1 \\ ED(\tilde{Y}_{1:t-1}^{(n,m)}, Y^{(n)}) - ED(\tilde{Y}_{1:t}^{(n,m)}, Y^{(n)}) & \text{if} \quad 1 < t < T \\ -ED(\tilde{Y}^{(n,m)}, Y^{(n)}) & \text{if} \quad t = T \end{cases} \tag{6.20}$$

where $ED(\cdot, \cdot)$ is the edit-distance function between two transcriptions, $\tilde{Y}_{1:t}^{(n,m)}$ is a substring of $\tilde{Y}^{(n,m)}$ from index 1 to t, $|Y^{(n)}|$ is the ground-truth length, and T is the sample transcription $\tilde{Y}^{(n,m)}$ length. Intuitively, we calculate whether the current new transcription at time-t decreases the edit-distance compared to previous transcriptions and multiply it by -1 for a positive reward if our new edit-distance at time t is smaller than the previous $t-1$ edit-distance. Also, at the end-of-sentence at time-T, we give a penalty based on the final edit-distance between the sample and the ground-truth transcription. In Fig. 6.5, we illustrate our reward scoring at each time-step from different trajectory samples.

In most cases, the current selected action affects future states and actions as well. Therefore, we should also account for some of the future rewards in the current time-step. Reward $R_t^{(n)}$ can be written:

$$\begin{aligned} R_t^{(n,m)} =& \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t) \\ & + \gamma \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t+1) \\ & + \gamma^2 \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t+2) + ... \\ & + \gamma^{T-t} \mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, T), \end{aligned} \tag{6.21}$$

where γ is the discount factor.

Additionally, since the REINFORCE estimator has high variance and could cause instability in the training stage, we apply the following normalization for reward $R^{(n,m)}$:

$$R_t^{(n,m)} = \begin{cases} \dfrac{\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t) - \mu_{(n,t)}}{\sigma_{(n,t)}} & \text{if} \quad 1 \leq t < T \\ \dfrac{\mathcal{R}(\tilde{Y}^{(n,m)}, Y^{(n)}, t) - \mu_{(n,</s>)}}{\sigma_{(n,</s>)}} & \text{if} \quad t = T, \end{cases} \tag{6.22}$$

where $\mu_{(n,t)}, \sigma_{(n,t)}$ is the reward mean and standard deviation for all samples at the t-th timestep, $\mu_{(n,</s>)}, \sigma_{(n,</s>)}$ is the reward mean and standard deviation for all the samples at the end of the transcription (denoted with `<\s>`), and T is the sample transcription $\tilde{Y}^{(n,m)}$ length. We separate the mean and the standard deviations between `<\s>` and non-`<\s>` labels because the reward function (Eq. 6.20) has different ways to calculate the reward. We provide the pseudocode for calculating token-level reward in Algorithm 5.

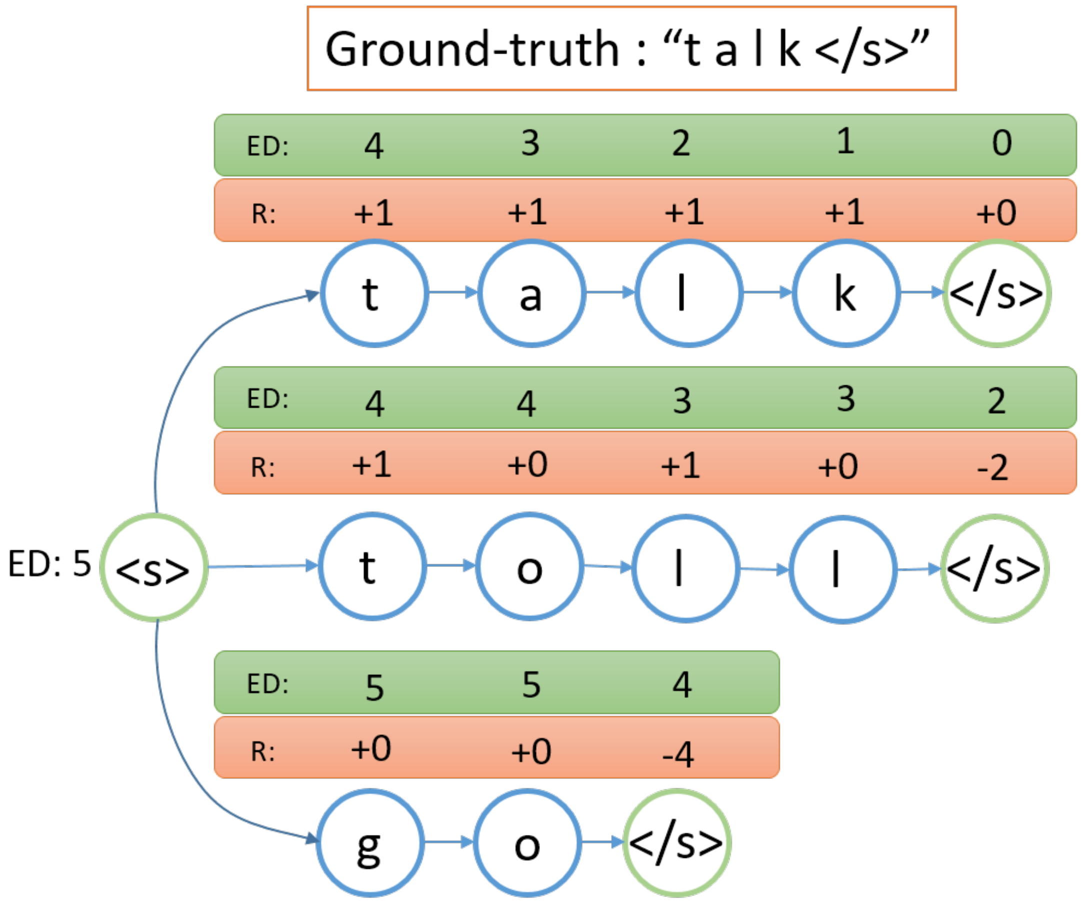


Figure 6.5. Based on Eq. 6.20, we provide an example for how to calculate the reward for each sample trajectory.

Algorithm 4 Pseudo-code for policy gradient with sentence-level reward R

```
1: procedure LossPGSentence(Speech features x, ground-truth text y_gold,
   sample size M, vocab size V)
2:     l_s_logp, l_s_act = Sample(x, M, V)
                                                        ▷ Algorithm 3
3:     l_r = []
4:     for m in [1..M] do
5:         # Calculate reward between ground-truth and each sample
6:         l_r.add($\mathcal{R}$(y_gold, l_s_act[m]))              ▷ Eq. 6.18
7:     end for
8:     # Reward normalization
9:     l_r = (l_r - mean(l_r)) / std(l_r)                 ▷ Eq. 6.19
10:    # Calculate loss and update $\theta_{ASR}$ model
11:    $\mathcal{L} = 0$
12:    for m in [1..M] do
13:        for t in [1..len(l_s_act[m])] do
14:            $\mathcal{L}$ += -l_s_logp[m, t] * l_r[m]
15:        end for
16:    end for
17:    $\theta_{ASR} = \text{Optim}(\theta_{ASR}, \nabla_{\theta_{ASR}}\mathcal{L})$      ▷ update ASR parameters
18: end procedure
```

Algorithm 5 Pseudocode for policy gradient with token-level reward R_t

```
1: procedure LossPGToken(Speech features x, ground-truth text y_gold,
   sample size M, discount factor $\gamma$, vocab size V)
2:     l_s_logp, l_s_act = Sample(x, M, V)
                                                            ▷ Algorithm 3
3:     l_r = [[] for _ in [0..M]]
4:     for m in [1..M] do
5:         for t in [1..len(l_s_act[m])] do
6:             # Calculate reward between ground-truth and each sam-
               ple at time-$t$
7:             l_r[m].add($\mathcal{R}$(l_s_act[m],
                   y_gold, t))                              ▷ Eq. 6.20
8:         end for
9:     end for
10:    # Calculate discounted reward
11:    for m in [1..M] do
12:        R = 0
13:        for t in [len(l_s_act[m]) .. 1] do
14:            R = l_r[m, t] + $\gamma$ * R
15:            l_r[m, t] = R
16:        end for
17:    end for
18:    # Reward normalization
19:    l_r = normalization(l_r)                             ▷ Eq. 6.22
20:    # Calculate loss and update $\theta_{ASR}$ model
21:    for m in [1..M] do
22:        for t in [1..len(l_s_act[m])] do
23:            $\mathcal{L}$ += -l_s_logp[m, t] * l_r[m, t]
24:        end for
25:    end for
26:    $\theta_{ASR}$ = Optim($\theta_{ASR}, \nabla_{\theta_{ASR}}\mathcal{L}$)    ▷ update ASR parameters
27: end procedure
```

Table 6.1. WSJ subset information

Subset	**Utterances**	**Duration**	**Speakers**
train_si84	7138	16 h	83
train_s284	38154	80 h	282
eval_dev93	503	65 m	10
eval_test92	333	42 m	8

6.6. Experiment

6.6.1 Speech Dataset and Feature Extraction

We evaluate our proposed method using Wall Street Journal dataset (WSJ) [82]. Following Kaldi s5 recipe [52], we use same training, validation and test sets partition. For the training, we a smaller set (train_si84) for preliminary and faster experiment, then later we use full set (train_si284). The speech features are computed with 80-dimension log Mel-filterbank with 25 ms window width and 10 ms window step. The text transcription are tokenized into characters, which contains alphabet, space, dashes, periods, apostrophes, noise and end-of-sentence (`</s>`). We describe the details for such as number of utterances, duration and unique speakers for each set on WSJ in Table 6.1.

6.6.2 Model Architecture

Our encoder input is a sequence of Mel-frequency spectrogram with 80 dimensions. For each frame, the input is projected by a dense linear layer with 512 output units and transformed by leaky rectifier unit (LeakyReLU) [41] as the non-linear activation function. Later, the output from dense linear layer was processed by three bi-directional LSTMs [11] (bi-LSTM) with 512 hidden units (256 hidden units for each direction). We apply hierarchical sub-sampling [83, 42] by a factor of 2 for all bi-LSTM output and the final encoder states has $T/8$ length compared to the original speech features. This trick is useful to reduce the computation time and memory usage for seq2seq model.

Our decoder has an autoregressive form which takes the character output from the previous time-step as the current time-step input. Every character is projected by a continuous vector via character embedding with 128 dimensions. Later, one uni-directional LSTM with 512 units project the character vector. The attention module with MLP scorer (256 units projection layer) calculates the context vector c_t, concatenated with the LSTM output and finally projected into a categorical probability distribution with a softmax layer. To optimize our seq2seq ASR model, we use Adam [84] with learning rate $lr = 0.0005$.

We have two steps of training seq2seq ASR. First, we pre-train seq2seq ASR by minimizing NLL criterion (Eq. 6.4) via teacher forcing generation until the loss is stable and converged. Later, we continue the training by summing the RL objective with the NLL criterion at the same time until the character error rate (CER) in the dev set stops decreasing.

We use beam-search (beam-size = 5) decoding to transcript the speech utterance in the testing step. Each beam score is calculated by their log probability $\log P(Y|X;\theta)$ and divided by the hypothesis length to prevent the top-K beams promoting shorter hypothesis. In this work, we did not utilize any lexicon dictionary or language model. We use Pytorch k[1] library to implement our model and loss function.

[1]PyTorch `https://github.com/pytorch/pytorch/`

6.6.3 Results and Discussion

Table 6.2. Character error rate (CER) report from WSJ train_si84 set (small set), comparing the result between baseline (without RL) and proposed method (NLL + RL). All decoding results were produced without additional language model or lexicon dictionary.

Models	**Results**
WSJ-SI84	**CER (%)**
NLL	
CTC [54]	20.34 %
Seq2Seq Content [54]	20.06 %
Seq2Seq Location [54]	17.01 %
Joint CTC+Att (MTL) [54]	14.53 %
Seq2Seq (ours)	17.68 %
NLL + RL	
Seq2Seq + RL (sentence-level R, $M = 5$)	16.88 %
Seq2Seq + RL (sentence-level R, $M = 10$)	15.38 %
Seq2Seq + RL (sentence-level R, $M = 15$)	15.21 %
Seq2Seq + RL (token-level R_t, $M = 5, \gamma = 0$)	15.17 %
Seq2Seq + RL (token-level R_t, $M = 5, \gamma = 0.5$)	15.34 %
Seq2Seq + RL (token-level R_t, $M = 5, \gamma = 0.95$)	14.75 %
Seq2Seq + RL (token-level R_t, $M = 10, \gamma = 0$)	15.08 %
Seq2Seq + RL (token-level R_t, $M = 10, \gamma = 0.5$)	14.45 %
Seq2Seq + RL (token-level R_t, $M = 10, \gamma = 0.95$)	14.29 %
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0$)	14.99 %
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0.5$)	14.25 %
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0.95$)	13.92 %

Table 6.2 shows the ASR performance on the WSJ-SI84. Here, we compare our proposed model (NLL + RL) with the baseline (without RL). Our baseline model is an attention encoder-decoder that was only trained with the NLL objective. In addition, we also compared our results with several published models, including CTC, standard seq2seq, and the Joint CTC-Attention model trained with the NLL objective. The main difference between our seq2seq model with others is that our decoder calculates the attention probability and context vector based on the current hidden state instead of the previous hidden state. Furthermore, we also reused the previous context vector by concatenating it with the input embedding vector.

We ran various experiments with different scenarios:

- Reward types:
  1. sentence-level reward (Sec. 6.5.2)
  2. token-level reward (Sec. 6.5.2)
- Sample sizes:
  1. $M = 5$
  2. $M = 10$
  3. $M = 15$
- Discount factors (for token-level reward):
  1. $\gamma = 0$
  2. $\gamma = 0.5$
  3. $\gamma = 0.95$

To show the effect of different sample sizes, we plotted the performances into different lines with respect to the CER in Fig. 6.6. From another perspective, we also provided Fig. 6.7 to compare the performances within different reward formulations and discount factors.

Based on the result in Table 6.2, we observed the following:

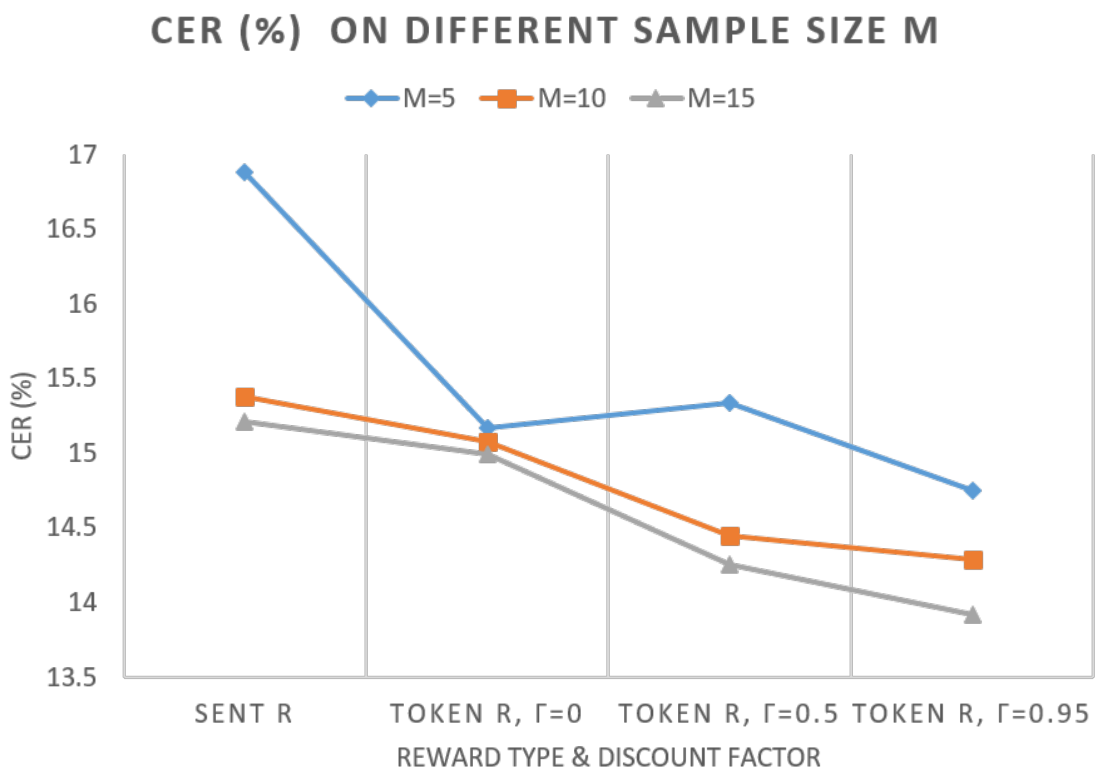


Figure 6.6. CER (%) comparisons between different sample sizes M

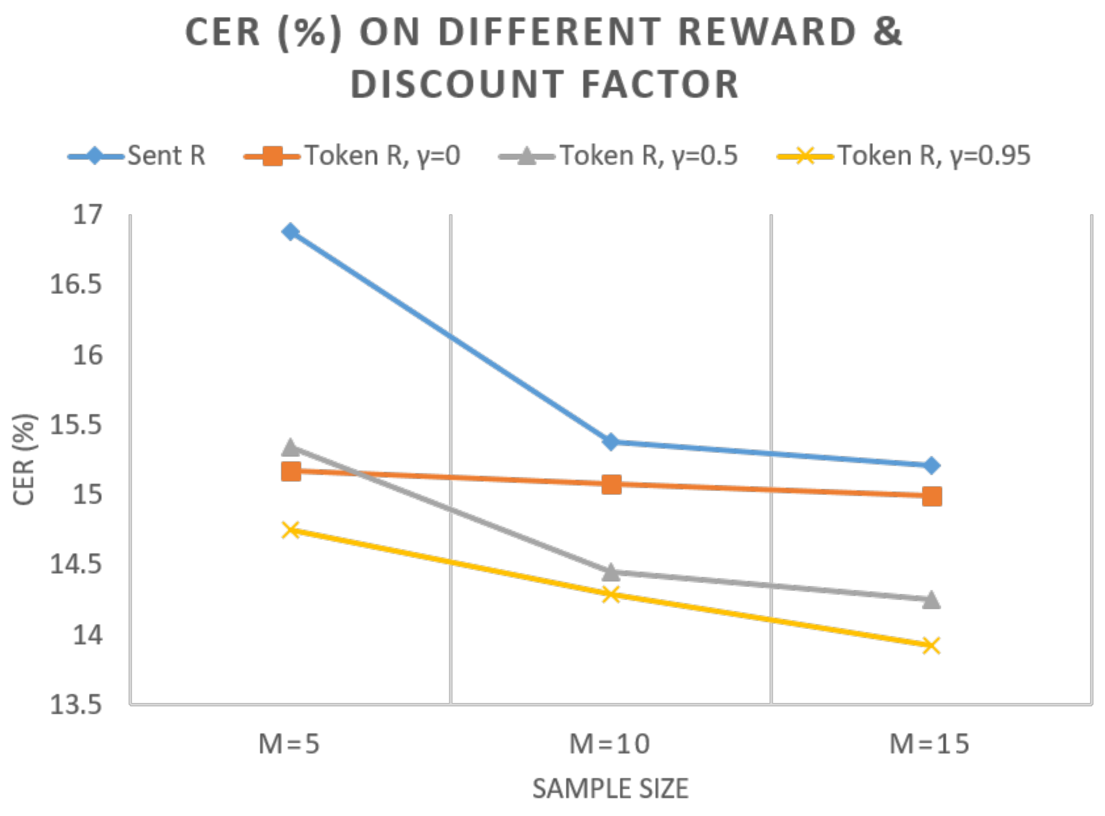


Figure 6.7. CER (%) comparison between different reward types and discount factors γ.

1. Increasing sample size M from 5 to 10 and 10 to 15 generally improved the performance. Unfortunately, the training time also increased linearly with sample size M.

2. Token-level reward improved the performance more than the model trained with the sentence-level reward.

3. Discount factor $\gamma = 0.95$ provided a better result than $\gamma = 0.5$ and $\gamma = 0.0$ in most cases.

Next we extended our experiment on WSJ train_si284, which is much larger than train_si84. Since our previous observation about the train_si84 dataset concluded that sample $M = 15$ gave a better result than any smaller sample size, we fixed our sample size to $M = 15$.

Table 6.3. Character error rate (CER) report from WSJ train_si284 set (large set), comparing the result between baseline (without RL) and proposed method (NLL + RL). All decoding results were produced without additional language model or lexicon dictionary.

Models	**Results**
WSJ-SI284	**CER (%)**
MLE	
CTC [54]	8.97 %
Seq2Seq Content [54]	11.08 %
Seq2Seq Location [54]	8.17 %
Joint CTC+Att (MTL) [54]	7.36 %
Seq2Seq (ours)	7.69%
MLE+RL	
Seq2Seq + RL (sentence-level R)	7.26%
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0$)	6.64 %
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0.5$)	6.37 %
Seq2Seq + RL (token-level R_t, $M = 15, \gamma = 0.95$)	6.10 %

We provide the result from WSJ train_si284 in Table 6.3. From the table, we could observe that the combination between NLL teacher forcing and RL objective significantly improve the seq2seq ASR performance compared to a model trained by NLL teacher forcing only. For both train_si84 and train_si284 dataset, the best discount factor for token-level reward is $\gamma = 0.95$.

6.7. Conclusion

This manuscript introduced an alternative strategy for training end-to-end ASR models by integrating an idea from reinforcement learning. Our proposed method integrates: (1) the power of sequence-to-sequence approaches to learn mapping between speech signals and text transcription; and (2) the strength of reinforcement learning to directly optimize the model with ASR performance metrics. Here, several different scenarios for training with RL-based objectives are explored with various reward functions, sample sizes, and discount factors. Experimental results reveal that by combining RL-based objectives with MLE objectives, our model performance could significantly improve in comparison to the model that just trained with MLE objectives. The best system achieved up to 6.10% CER in WSJ-SI284 using token-level rewards, sample size $M = 15$, and discount factor $\gamma = 0.95$.

Chapter 7

Discussion

7.1. Related Works

The learning process by using reconstruction loss as the criterion has been popularized by autoencoder-based models. For example, denoising autoencoder [85] learns robust representative features by learning to reconstruct from a noisy input. Constrastive autoencoder [86] improves the representation upon denoising autoencoder model by penalized the norm for the Jacobian matrix of encoder activation with respect to the input. Variational autoencoder (VAE) [87] introduced an stochastic autoencoder under Bayesian formulation with simple normal distribution as the prior. All of these model has either implicit or explicit objective that encourage the latent variable to be compact and contain enough information to reconstruct the input. It draws a similarity with speech chain during a scenario: (speech $\rightarrow$ ASR $\rightarrow$ text $\rightarrow$ TTS $\rightarrow$ speech) where the text represent compact representation of speech utterances.

In the computer vision field, image-to-image translation between different styles is a very hard problem since there is a limited number of pairs and it is hard to create the annotation. However, CycleGAN [88] is proposed to tackle the image-to-image translation with purely unsupervised data. They have two generators, where the first generator transforms the image from domain A to domain B, and the second generator inverse transforms the image from domain B to domain B. During the training stage, combining cycle consistency loss from the first generator and second generator with adversarial loss. However, there are some similarities between domain A and B (e.g, from horse images to zebra images) and CycleGAN use identity loss to preserve some similarity from image A and image B. Compared to our problem, speech and text doesn't contain the same amount of information and there is no intermediate loss such as identity loss to guide the generation in the middle. In the speech chain, we use standard loss functions such as regression loss L2-norm for TTS and classification loss negative log-likelihood (NLL) for ASR. Compared to GAN loss which involves minimax game between the generator and discriminator, standard reconstruction losses are less sensitive to the change of hyperparameters or architectures.

Approaches that utilize learning from source-to-target and vice-versa, as well as feedback links, remain scant. He et al. [89] quite recently published a work that addressed a mechanism called dual learning in neural machine translation. Their system has a dual task: source-to-target language translation (primal) versus target-to-source language translation (dual). The primal and dual tasks form a closed loop and generate informative feedback signals to train the translation models, even without the involvement of a human labeler. This approach was originally proposed to tackle training data bottleneck problems. With a dual-learning mechanism, the system can leverage monolingual data (in both the source and target languages) more effectively. First, they construct one model to translate from the source to the target language and another to translate from the target to the source language. After both the first and second models have been trained with a small parallel corpus, they start to teach each other using monolingual data and generate useful feedback with language model likelihood and reconstruction error to further improve the performance.

Another similar work in neural machine translation was introduced by Cheng et al. [90] and Senrich et al. [91]. This approach also exploited monolingual corpora to improve neural machine translation. Their system utilizes a semi-supervised approach for training neural machine translation (NMT) models on the concatenation of labeled (parallel corpora) and unlabeled (monolingual corpora) data. The central idea is to reconstruct monolingual corpora using an autoencoder in which the source-to-target and target-to-source translation models serve as the encoder and decoder, respectively.

In this manuscript, we addressed similar problems in spoken language processing tasks. This paper presents a novel mechanism that integrates human speech perception and production behaviors. With a concept that resembles dual learning in NMT, we utilize the primal model (ASR) that transcribes the text given the speech versus the dual model (TTS) that synthesizes the speech given the text. However, the main difference between NMT is that the domain between the source and the target here are different (speech versus text). While ASR transcribes the unlabeled speech features, TTS attempts to reconstruct the original speech waveform based on the text from ASR. In the opposite direction, ASR also attempts to reconstruct the original text transcription given the synthesized speech. Nevertheless, our experimental results show that the proposed approach also identified a successful learning strategy and significantly improved performance over that of separate systems that were only trained with labeled data.

After our preliminary work, several works have discussed our methods and built on top of them. Karita et al. [92] form a text and speech autoencoder and train unpaired data with reconstruction loss. Ren et al., [93] replaced the LSTM-based encoder-decoder with Transformer modules for both ASR and TTS and achieved good performance with small paired speech-text in single speaker dataset. Rosenberg et al. [94] explored the effect of data augmentation by using TTS on the larger experiment. Kurata et al. [95] improved ASR performance by adding feature reconstruction loss during training. Ueno et al. [96] use synthetic speeches from multi-speaker TTS to improve their Acoustic2Word speech recognition system. Baskar et al. [97] proposed an alternative to backpropagate through discrete variables by using a policy-gradient method, compared to our proposal using a straight-through estimator. Hori et al. [98] replaced TTS with text-to-encoder (TTE) to avoid the need for modeling the speaking style during the reconstruction.

7.2. Conclusions

This thesis addressed various issues of current speech processing technology such as the ASR and TTS research are independently progressed without exerting much influence on each other, limitation of current ASR and TTS where we required a large amount of paired speech and text to achieved high accuracy result. Inspired by the nature of human communication where there is an auditory feedback mechanism from the speaker's to their ear, called a speech chain mechanism. Here, we proposed a novel machine speech chain mechanism based on deep learning to emulate the feedback loop idea to the computer system.

In the Chapter 3, we present the basic speech chain with sequence-to-sequence model. The closed-loop architecture allows us to train our model on the concatenation of both labeled and unlabeled data. To test the feasibility of our idea, we run the experiment on the single-speaker dataset. Based on the experiment result, the unpaired data from speech and text can improve the performance of ASR and TTS on the single speaker task.

After the success of our preliminary experiment on the single-speaker dataset. We expand our single speaker speech chain into a multi-speaker speech chain in Chapter 4. However, there are some obstacles where most of the TTS systems are designed to generate voice from a single speaker. Therefore, we proposed a new end-to-end TTS architecture with one-shot speaker adaptation by conditioning the decoder with speaker embedding from a speaker embedding network. Our experimental results show improvement in the ASR and TTS models for the multi-speaker dataset.

During the development of the speech chain, we noticed that there is a discontinuity between the ASR and TTS module, caused by the output of ASR are represented with discrete variables. Because we represent the text with discrete variables, we could not do backpropagate TTS loss with respect to the ASR parameters. Therefore, we introduced an extension to allow backpropagation through the discrete output from the ASR module with a straight-through estimator in Chapter 5.

During my research period, I analyzed some problems in the training sequence-to-sequence ASR. Most of the sequence-to-sequence models are trained by the teacher forcing method because of its simplicity and effectiveness. However, there is a discrepancy between the training and the inference stage. In the inference stage, the model does not have access to the ground truth during the decoding and any small error during the early stage of decoding might propagate to the later stage. Another problem also arises from the objective function where the teacher forcing use negative log-likelihood instead of the directly minimizing ASR task metric such as word/character error rate. To solve both issues, we proposed another method for training ASR via reinforcement learning in Chapter 6. We showed that by optimizing the ASR model with reinforcement learning and deriving a reward function based on edit-distance, we could improve the ASR performance significantly.

7.3. Future Directions

At the moment, the machine speech chain has been extended for various applications such as:

1. Code-switching ASR [99]
   Creating a parallel code-switching corpus is hard and time-consuming. Therefore, we tried to develop a code-switching ASR by using paired non-code switching datasets and unpaired code-switching speech and text. This approach successfully improved the ASR performance on code-switching significantly in terms of word error rate on the code-switching speech.

2. Multimodal speech-chain [100]
   Extending speech chain capabilities into other modalities such as visual modality. This approach incorporates image captioning and image retrieval/generation into the speech chain loop and improves the performance of the ASR system.

3. Unsupervised subword discovery with speaker style disentanglement [101].
   Inspired by a similar architecture with a multi-speaker machine speech chain, we build a conditional autoencoder with the discrete variable to represent the context of the speech and disentangle the speaker's identity as well. The proposed model are successfully achieved low ABX discrimination score and high-quality voice conversion result in ZeroSpeech 2019 challenge [102].

In the future, is necessary to further validate the effectiveness of our approach to various languages and conditions (i.e., spontaneous, noisy, and emotion). Speech synthesis with one-shot adaptation (e.g. emotion, speaking rate, etc) also worth to be explored near the future.Lastly, we also interested to explore how to implement the speech chain mechanism to assist human communication, for example, provide assistance for a person who has a hearing impairment.

References

[1] P.B. Denes and E. Pinson. *The Speech Chain*. Anchor books. Worth Publishers, 1993.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence-to-Sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[3] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Machine speech chain with one-shot speaker adaptation. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018.*, pages 887–891, 2018.

[4] K. H. Davis, R. Biddulph, and S. Balashek. Automatic recognition of spoken digits. *Acoustic Society of America*, pages 627–642, 1952.

[5] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Kibernetika*, pages 81–88, 1968.

[6] H. Sakoe and S. Chiba. Dynamic programming algorithm quantization for spoken word recognition. *IEEE Transaction on Acoustics, Speech and Signal Processing*, ASSP-26(1):43–49, 1978.

[7] F. Jelinek. Continuous speech recognition by statistical methods. *IEEE*, 64:532–536, 1976.

[8] J. G. Wilpon, L. R. Rabiner, C. H. Lee, and E. R. Goldman. Automatic recognition of keywords in unconstrained speech using hidden Markov models. *IEEE Transaction on Acoustics, Speech and Signal Processing*, 38(11):1870–1878, 1990.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

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

[12] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[13] John N Holmes, Ignatius G Mattingly, and John N Shearme. Speech synthesis by rule. *Language and speech*, 7(3):127–143, 1964.

[14] J. P. Olive. Rule synthesis of speech from dyadic units. In *Proceedings of ICASSP*, pages 568–570, 1977.

[15] Y. Sagisaka. Speech synthesis by rule using an optimal selection of non-uniform synthesis units. In *Proceedings of ICASSP*, 1988.

[16] Y. Sagisaka, N. Kaiki, N. Iwahashi, and K. Mimura. Atr υ-talk speech. In *Proceedings of ICSLP*, pages 483–486, 1992.

[17] A. Hunt and A. Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *Proceedings of ICASSP*, pages 373–376, 1996.

[18] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura. Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis. In *Proceedings of Eurospeech*, pages 2347—-2350, 1999.

[19] K. Tokuda, T. Kobayashi, and S. Imai. Speech parameter generation from HMM using dynamic features. In *Proceedings of ICASSP*, pages 660—-663, 1995.

[20] Dimitri Palaz, Mathew Magimai Doss, and Ronan Collobert. Convolutional neural networks-based continuous speech recognition using raw speech signal. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4295–4299. IEEE, 2015.

[21] Tara N Sainath, Ron J Weiss, Andrew W Senior, Kevin W Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform cldnns. In *Interspeech*, volume 2015, 2015.

[22] Heiga Zen, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 7962–7966, 2013.

[23] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[24] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, S. Sengupta, and M. Shoeybi. Deep voice: Real-time neural text-to-speech. *arXiv preprint arXiv:1702.07825*, 2017.

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

[26] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783. IEEE, 2018.

[27] Wei-Ning Hsu, Yu Zhang, Ron J Weiss, Heiga Zen, Yonghui Wu, Yuxuan Wang, Yuan Cao, Ye Jia, Zhifeng Chen, Jonathan Shen, et al. Hierarchical generative modeling for controllable speech synthesis. *arXiv preprint arXiv:1810.07217*, 2018.

[28] Shinji Watanabe, Marc Delcroix, Florian Metze, and John R. Hershey, editors. *New Era for Robust Speech Recognition, Exploiting Deep Learning.* Springer, 2017.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[30] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent NN: First results. *arXiv preprint arXiv:1412.1602*, 2014.

[31] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[32] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.

[33] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[34] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[35] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[36] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

[37] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Multi-scale alignment and contextual history for attention mechanism in sequence-to-sequence model. *To appear in IEEE SLT 2018*, 2018.

[38] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[39] Keith Ito. The LJ speech dataset. `https://keithito.com/LJ-Speech-Dataset/`, 2017.

[40] Brian McFee, Matt McVicar, Oriol Nieto, Stefan Balke, Carl Thome, Dawen Liang, Eric Battenberg, Josh Moore, Rachel Bittner, Ryuichi Yamamoto, and et al. librosa 0.5.0. Feb 2017.

[41] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[42] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE, 2016.

[43] Yun Lei, Nicolas Scheffer, Luciana Ferrer, and Mitchell McLaren. A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1695–1699. IEEE, 2014.

[44] Pavel Matějka, Ondřej Glembek, Fabio Castaldo, Md Jahangir Alam, Oldřich Plchot, Patrick Kenny, Lukáš Burget, and Jan Černocky. Full-covariance ubm and heavy-tailed plda in i-vector speaker verification. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4828–4831. IEEE, 2011.

[45] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. Deep speaker: an end-to-end neural speaker embedding system. *arXiv preprint arXiv:1705.02304*, 2017.

[46] Yingke Zhu, Tom Ko, David Snyder, Brian Mak, and Daniel Povey. Self-attentive speaker embeddings for text-independent speaker verification. *Proc. Interspeech 2018*, pages 3573–3577, 2018.

[47] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[49] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International Conference on Machine Learning*, pages 1558–1566, 2016.

[50] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[51] Douglas B Paul and Janet M Baker. The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics, 1992.

[52] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian,

Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[53] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[54] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. Joint CTC-attention based end-to-end speech recognition using multi-task learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4835–4839. IEEE, 2017.

[55] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Attention-based wav2text with feature transfer learning. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017, Okinawa, Japan, December 16-20, 2017*, pages 309–315, 2017.

[56] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, 2002.

[57] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[58] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[59] Geoffrey Hinton. Neural networks for machine learning, Coursera video lectures. 2012.

[60] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017.

[61] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[62] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[63] Marc Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[64] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[65] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[66] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[68] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[69] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[70] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, pages 956–962, 2000.

[71] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[72] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[73] Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. Bridging the gap between training and inference for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4334–4343, Florence, Italy, July 2019. Association for Computational Linguistics.

[74] Matt Shannon. Optimizing expected word error rate via sampling for speech recognition. *arXiv preprint arXiv:1706.02776*, 2017.

[75] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Sequence-to-sequence asr optimization via reinforcement learning. *arXiv preprint arXiv:1710.10774*, 2017.

[76] S. Karita, A. Ogawa, M. Delcroix, and T. Nakatani. Sequence training of encoder-decoder model using policy gradient for end- to-end speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5839–5843, April 2018.

[77] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[78] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[79] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[80] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

[81] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning-Volume 32*, pages II–1791. JMLR. org, 2014.

[82] Douglas B. Paul and Janet M. Baker. The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 357–362, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[83] Alex Graves. Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer, 2012.

[84] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[85] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[86] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011.

[87] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[88] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In

Proceedings of the IEEE international conference on computer vision, pages 2223–2232, 2017.

[89] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tieyan Liu, and Wei-Ying Ma. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pages 820–828, 2016.

[90] Yong Cheng, Wei Xu, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Semi-supervised learning for neural machine translation. *arXiv preprint arXiv:1606.04596*, 2016.

[91] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 86–96, 2016.

[92] Shigeki Karita, Shinji Watanabe, Tomoharu Iwata, Atsunori Ogawa, and Marc Delcroix. Semi-supervised end-to-end speech recognition. *Proc. Interspeech 2018*, pages 2–6, 2018.

[93] Yi Ren, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Almost unsupervised text to speech and automatic speech recognition. *arXiv preprint arXiv:1905.06791*, 2019.

[94] Andrew Rosenberg, Yu Zhang, Bhuvana Ramabhadran, Ye Jia, Pedro Moreno, Yonghui Wu, and Zelin Wu. Speech recognition with augmented synthesized speech. *arXiv preprint arXiv:1909.11699*, 2019.

[95] Gakuto Kurata and Kartik Audhkhasi. Multi-task ctc training with auxiliary feature reconstruction for end-to-end speech recognition. *Proc. Interspeech 2019*, pages 1636–1640, 2019.

[96] Sei Ueno, Masato Mimura, Shinsuke Sakai, and Tatsuya Kawahara. Multi-speaker sequence-to-sequence speech synthesis for data augmentation in acoustic-to-word speech recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6161–6165. IEEE, 2019.

[97] Murali Karthick Baskar, Shinji Watanabe, Ramon Astudillo, Takaaki Hori, Lukáš Burget, and Jan Černockỳ. Self-supervised sequence-to-sequence asr using unpaired speech and text. *arXiv preprint arXiv:1905.01152*, 2019.

[98] Takaaki Hori, Ramon Astudillo, Tomoki Hayashi, Yu Zhang, Shinji Watanabe, and Jonathan Le Roux. Cycle-consistency training for end-to-end speech recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6271–6275. IEEE, 2019.

[99] Sahoko Nakayama, Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Speech chain for semi-supervised learning of japanese-english code-switching asr and tts. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 182–189. IEEE, 2018.

[100] Johanes Effendi, Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. From speech chain to multimodal chain: Leveraging cross-modal data augmentation for semi-supervised learning. *CoRR*, abs/1906.00579, 2019.

[101] 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.

[102] Ewan Dunbar, Robin Algayres, Julien Karadayi, Mathieu Bernard, Juan Benjumea, Xuan-Nga Cao, Lucie Miskic, Charlotte Dugrain, Lucas Ondel, Alan W Black, Laurent Besacier, Sakriani Sakti, and Emmanuel Dupoux. The Zero Resource Speech Challenge 2019: TTS without T. In *Interspeech 2019 - 20th Annual Conference of the International Speech Communication Association*, Graz, Austria, September 2019.

List of publications

For the latest publication list, please refer to: `https://scholar.google.com/citations?user=Bvox_f8AAAAJ&hl=en`.

Publications

Journal Paper (peer-reviewed)

1. *Machine Speech Chain*
   Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
   IEEE/ACM Transactions on Audio, Speech, and Language Processing Vol.28, 976-989

2. *End-to-End Speech Recognition Sequence Training With Reinforcement Learning*
   Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
   IEEE Access 7, 79758-79769

3. *Recurrent Neural Network Compression based on Low-Rank Tensor Representation*
   Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
   IEICE Transaction

Conference papers (peer-reviewed)

1. *Transformer VQ-VAE for Unsupervised Unit Discovery and Speech Synthesis: ZeroSpeech 2020 Challenge*

Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
INTERSPEECH, 2020

2. *Deja-vu: Double Feature Presentation in Deep Transformer Networks*
   Andros Tjandra, Chunxi Liu, Frank Zhang, Xiaohui Zhang, Yongqiang Wang, Gabriel Synnaeve, Satoshi Nakamura, Geoffrey Zweig
   IEEE International Acoustics, Speech and Signal Processing (ICASSP), 2020

3. *Transformer-based Acoustic Modeling for Hybrid Speech Recognition*
   Yongqiang Wang, Abdelrahman Mohamed, Duc Le, Chunxi Liu, Alex Xiao, Jay Mahadeokar, Hongzhao Huang, **Andros Tjandra**, Xiaohui Zhang, Frank Zhang, Christian Fuegen, Geoffrey Zweig, Michael L Seltzer.
   IEEE International Acoustics, Speech and Signal Processing (ICASSP), 2020

4. *Cross-Lingual Machine Speech Chain for Javanese, Sundanese, Balinese, and Bataks Speech Recognition and Synthesis*
   Sashi Novitasari, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
   Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages, 2020

5. *Speech-to-speech Translation between Untranscribed Unknown Languages*
   Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
   IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2019

6. *Zero-Shot Code-Switching ASR and TTS with Multilingual Machine Speech Chain*
   Sahoko Nakayama, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
   IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2019

7. *Listening while Speaking and Visualizing: Improving ASR through Multimodal Chain*
   Johannes Effendi, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
   IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2019

8. *VQVAE Unsupervised Unit Discovery and Multi-scale Code2Spec Inverter for Zerospeech Challenge 2019*
   Andros Tjandra, Berrak Sisman, Mingyang Zhang, Sakriani Sakti, Haizhou Li, Satoshi Nakamura.
   INTERSPEECH, 2019

9. *Sequence-to-Sequence Learning via Attention Transfer for Incremental Speech Recognition*
   Sashi Novitasari, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
   INTERSPEECH, 2019

10. *End-to-End Feedback Loss in Speech Chain Framework via Straight-Through Estimator*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE International Acoustics, Speech and Signal Processing (ICASSP), 2019

11. *Recognition and translation of code-switching speech utterances.*
    Sahoko Nakayama, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
    22nd Conference of the Oriental COCOSDA International Committee for the Co-ordination and Standardisation of Speech Databases and Assessment Techniques, 2019

12. *Multi-scale Alignment and Contextual History for Attention Mechanism in Sequence-to-sequence Model*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE Spoken Language Technology (SLT), 2018

13. *Speech Chain for Semi-Supervised Learning of Japanese-English Code-Switching ASR and TTS*
    Sahoko Nakayama, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
    IEEE Spoken Language Technology (SLT), 2018

14. *Machine Speech Chain with One-shot Speaker Adaptation*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    INTERSPEECH, 2018

15. *Compressing End-to-end ASR Networks by Tensor-Train Decomposition*
    Takuma Mori, **Andros Tjandra**, Sakriani Sakti, Satoshi Nakamura.
    INTERSPEECH, 2018

16. *Tensor decomposition for compressing recurrent neural network*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE International Joint Conference Neural Networks (IJCNN), 2018

17. *Sequence-to-Sequence ASR optimization via reinforcement learning*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE International Acoustics, Speech and Signal Processing (ICASSP), 2018

18. *Listening while speaking: Speech chain by deep learning*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2017 (**Best student paper award**)

19. *Attention-based Wav2Text with feature transfer learning*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2017

20. *Speech recognition features based on deep latent Gaussian models*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.

IEEE Workshop on Machine Learning for Signal Processing (MLSP), 2017

21. *Local monotonic attention mechanism for end-to-end speech recognition and language processing*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    International Joint Conference on Natural Language Processing (IJCNLP), 2017

22. *Compressing recurrent neural network with Tensor-Train*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
    IEEE International Joint Conference Neural Networks (IJCNN), 2017

23. *Gated recurrent neural tensor network*
    Andros Tjandra, Sakriani Sakti, Ruli Manurung, Mirna Adriani, Satoshi Nakamura.
    IEEE International Joint Conference Neural Networks (IJCNN), 2016

24. *Stochastic gradient variational Bayes for deep learning-based ASR*
    Andros Tjandra, Sakriani Sakti, Satoshi Nakamura, Mirna Adriani.
    IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2015

25. *A heuristic Hidden Markov Model to recognize inflectional words in sign system for Indonesian language known as SIBI (Sistem Isyarat Bahasa Indonesia)*
    Erdefi Rakun, Mohammad Ivan Fanany, I Wayan Wiprayoga, **Andros Tjandra**.
    International Conference on Technology, Informatics, Management, Engineering & Environment (TIME-E), 2015

26. *Combination of two-dimensional cochleogram and spectrogram features for deep learning-based ASR*
    Andros Tjandra, Sakriani Sakti, Graham Neubig, Tomoki Toda, Mirna Adriani, Satoshi Nakamura.

IEEE International Acoustics, Speech and Signal Processing (ICASSP), 2015

27. *Combining depth image and skeleton data from kinect for recognizing words in the sign system for Indonesian language (SIBI [Sistem Isyarat Bahasa Indonesia])*
Erdefi Rakun, Mirna Adriani, I Wayan Wiprayoga, Ken Danniswara, **Andros Tjandra**.
International Conference on Advanced Computer Science and Information Systems (ICACSIS), 2013

28. *Spectral domain cross correlation function and generalized learning vector quantization for recognizing and classifying Indonesian sign language*
Erdefi Rakun, Muhammad Febrian Rachmadi, **Andros Tjandra**, Ken Danniswara.
International Conference on Advanced Computer Science and Information Systems (ICACSIS), 2012

Workshop paper (peer-reviewed) + Pre-print (non peer-reviewed)

1. *End-to-End Speech Recognition with Local Monotonic Attention*
Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
NIPS Workshop Machine Learning for Audio Signal Processing (ML4Audio), 2017

2. *Compact recurrent neural network based on Tensor-Train for polyphonic music modelling*
Andros Tjandra, Sakriani Sakti, Satoshi Nakamura.
NIPS Workshop Machine Learning for Audio Signal Processing (ML4Audio), 2017