ENFORCING CONSTRAINTS FOR MULTI-LINGUAL AND CROSS-LINGUAL SPEECH-TO-TEXT SYSTEMS

BY

JUNRUI NI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Adviser:

Professor Mark Hasegawa-Johnson

ABSTRACT

The recent development of neural network-based automatic speech recognition (ASR) systems has greatly reduced the state-of-the-art phone error rates in several languages. However, when an ASR system trained on one language tries to recognize speech from another language, such a system usually fails, even when the two languages come from the same language family. The above scenario poses a problem for low-resource languages. Such languages usually do not have enough paired data for training a moderately-sized ASR model and thus require either cross-lingual adaptation or zero-shot recognition.

Due to the increasing interest in bringing ASR technology to low-resource languages, the cross-lingual adaptation of end-to-end speech recognition systems has recently received more attention. However, little analysis has been done to understand how the model learns a shared representation across languages and how language-dependent representations can be fine-tuned to improve the system's performance. We compare a bi-lingual CTC model with language-specific tuning at earlier LSTM layers to one without such tuning. This is to understand if having language-independent pathways in the model helps with multi-lingual learning and why. We first train the network on Dutch and then transfer the system to English under the bi-lingual CTC loss. After that, the representations from the two networks are visualized. Results showed that the consonants of the two languages are learned very well under a shared mapping but that vowels could benefit significantly when further language-dependent transformations are applied before the last classification layer. These results can be used as a guide for designing multilingual and cross-lingual end-to-end systems in the future.

However, creating specialized processing units in the neural network for each training language could yield increasingly large networks as the number of training languages increases. It is also unclear how to adapt such a system to zero-shot recognition. The remaining work adapts two existing constraints

to the realm of multi-lingual and cross-lingual ASR. The first constraint is cycle-consistent training. This method defines a shared codebook of phonetic tokens for all training languages. Input speech first passes through the speech encoder of the ASR system and gets quantized into discrete representations from the codebook. The discrete sequence representation is then passed through an auxiliary speech decoder to reconstruct the input speech. The framework constrains the reconstructed speech to be close to the original input speech. The second constraint is regret minimization training. It separates an ASR encoder into two parts: a feature extractor and a predictor. Regret minimization defines an additional regret term for each training sample as the difference between the losses of an auxiliary language-specific predictor with the real language I.D. and a fake language I.D. This constraint enables the feature extractor to learn an invariant speech-to-phone mapping across all languages and could potentially improve the model's generalization ability to new languages.

To my parents, for their love and support.

ACKNOWLEDGMENTS

All the work completed in the thesis would not have been possible without the ongoing support of my graduate advisor, Prof. Mark Hasegawa-Johnson. He has provided me with guidance and insights throughout all my projects and connected me with other researchers and funding opportunities. For the first project (Chapter 3), I would additionally like to additionally thank Prof. Odette Scharenborg from the Delft University of Technology for her ideas, suggestions, and collaboration. For the second project (Chapter 4), supported by NSF IIS 19-10319 (RI: Small: Automatic creation of new phone inventories), I would like to additionally thank Prof. Odette Scharenborg, Dr. Siyuan Feng from the Delft University of Technology, Prof. Najim Dehak from Johns Hopkins University, Dr. Piotr Zelasko from Johns Hopkins University and Dr. Laureano Moro-Velázquez from Johns Hopkins University for their insights, and especially Dr. Piotr Zelasko, Dr. Laureano Moro-Velázquez, and Dr. Siyuan Feng for helping me with pre-processing and training scripts in ESPnet and Kaldi. For the third project (Chapter 5), I would like to thank Dr. Yang Zhang from IBM Research, Dr. Kaizhi Qian from IBM Research, and Prof. Shiyu Chang from the University of California, Santa Barbara, for their ideas, ongoing support, and guidance, as well as the IBM Center for Cognitive Systems Research for ongoing funding. I would also like to thank Heting Gao for his collaboration for implementations related to the third project. Lastly, I would like to thank all co-authors in [1] for allowing me to include sections from our in-review manuscript into Section 1.3, Section 2.4, Section 6.3, and Chapter 5 of the thesis.

Last but not least, I would like to thank my family and friends for their ongoing emotional support, especially during the past two years.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Different languages have different phoneme inventories, and the relationship between them is complicated. The first project compared how a language-independent and a language-dependent ASR model process information differently for two closely related languages. We analyzed the hidden representations of the consonant and vowel inventories from two languages at multiple layer outputs. We found that our language-independent model did not work as well as the language-dependent model with additional language-specific tuning layers. This is partly because the layers in the language-independent model are forced to share the representations from the two languages, and deeper layers in the network are less effective.

Language-specific tuning layers do not generalize well as the number of training languages increases. Therefore, we would like to solve the problem differently. One way to do so is to enforce some constraints when optimizing the recognition loss during multi-lingual training. We hope that constraining the search space during neural network optimization could guide the optimization process to possibly better local optima. We also hope that the constraints will help the model achieve lower error rates on unseen languages. In the following two projects, we experimented with two constraints for multi-lingual training:

1. cycle consistency constraint, where we constrain another text-to-speech model to correctly reconstruct the speech input into the ASR model, conditioned on decoded text transcripts from the same ASR model;

2. regret minimization constraint, where we define a regret term as the difference between the risk of an out-of-environment predictor and a within-environment predictor and constrain the regret to be small.

In the following three sections, we will discuss the motivations for the three projects individually and give a quick preview of the goals, methods,

and results.

1.1 Analyzing the Effect of Language-specific Tuning for a Bi-lingual CTC System

When a neural network-based automatic speech recognition (ASR) system tries to learn more than one language, it will have to deal with the similarities and differences between the phoneme inventories of those languages. One traditional approach is to create language-dependent softmax layers, where the number of output layers equals the number of training languages [2, 3, 4]. For example, a GMM-HMM system was trained on the bottleneck features from a multilingual artificial neural network (ANN) phoneme classifier, with or without language-dependent output mapping [4]. They found that the bottleneck features learned through language-dependent output mapping give superior performance over bottleneck features learned through a shared phone output mapping. Other approaches involve finding a global phone set for all the languages and using a shared phone output layer across the languages [5, 6]. However, due to differences in language-dependent realizations [7] of the phones and contextual information, systems that use a shared output layer usually try to apply language adaptive training techniques, such as the Language Feature Vector [5] and Learning Hidden Unit Contribution [6]. More recently, a language-independent phone layer has been transformed into language-dependent phonemes, with the result optimized using CTC [8].

However, except for the reported phone error rates, none of the work above analyzed what their networks had learned about the phoneme inventories of each language, or the relationships between languages. This may be due to the large number of training languages within their systems. With many languages, it is difficult to test for language independence of each layer: the parameters of the models could easily grow out of control if more than one or two layers are permitted to be language-dependent. In our work, we, therefore, limited the number of training languages to two. By doing so, we can analyze what the network has learned about the relationship between the phoneme sets of the two languages and the degree to which each phone benefits from language-dependent layers. We also hope to better characterize changes in the representation of the first language (L1) after the acquisition

of a second language (L2), a phenomenon that has been studied in human beings but not in neural networks.

We created two models, illustrated in Figure 1.1, to investigate these questions. All layers in the single pathway are shared between L1 and L2 except the softmax, while the dual-pathway model contains two language-dependent LSTM layers. We first train a CTC network on Dutch and then inject English utterances into the training loop to learn a bi-lingual model. We visualized the phoneme categories at the last shared layer of the dual pathway model (orange layer in Figure 1.1) and compared the learned representations with those from the last non-shared layer (gray layers in the dual-pathway model). By doing so, we hope to discover which speech sound representations require further language-dependent tuning. Likewise, we compared the activations from the last layer of the single pathway model with those from the last layer of the dual pathway model. By doing so, we hope to further understand what benefit, if any, language-specific tuning at layers before the softmax might bring to a bi-lingual system and how the relationships between L1 and L2 of a neural net resemble those of a human. In Chapter 3, we first present our experiment set-up, including a brief introduction to the chosen English and Dutch corpora and the phonetic differences of the two languages, our model architecture, and the training stages. We then describe the three experiments conducted to investigate language-specific tuning and propose several hypotheses for each experiment. Finally, we prove or disprove our hypotheses with either qualitative or quantitative results, such as visualization and error rates.

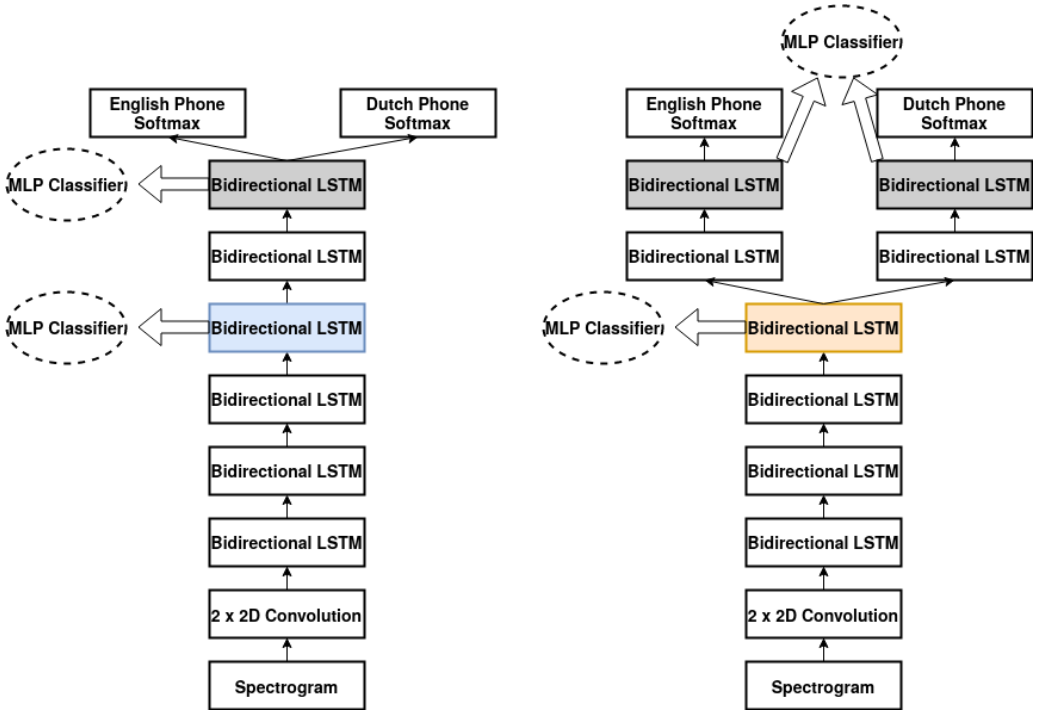

Figure 1.1: Model architecture; on the left: single pathway model; on the right: dual pathway model.

1.2 Cycle Consistency For Speech

Humans produce speech by first organizing their thoughts in their brain and then controlling their vocal organs to produce the message in linguistic form. The linguistic form is then propagated through the media via acoustic waves into the listener's ear, and the sensory nerves of the listener transmit the nerve impulses into the brain. The listener's brain then recognizes the message. The speaker also acts as a second listener via a feedback loop, comparing the sound they produce with the sound they intend to produce and adjusting if necessary. This entire process is denoted as the speech chain [9]. Figure 1.2, which is extracted from a recent work on machine speech chain [10], displays the above process and its connections to speech technology.

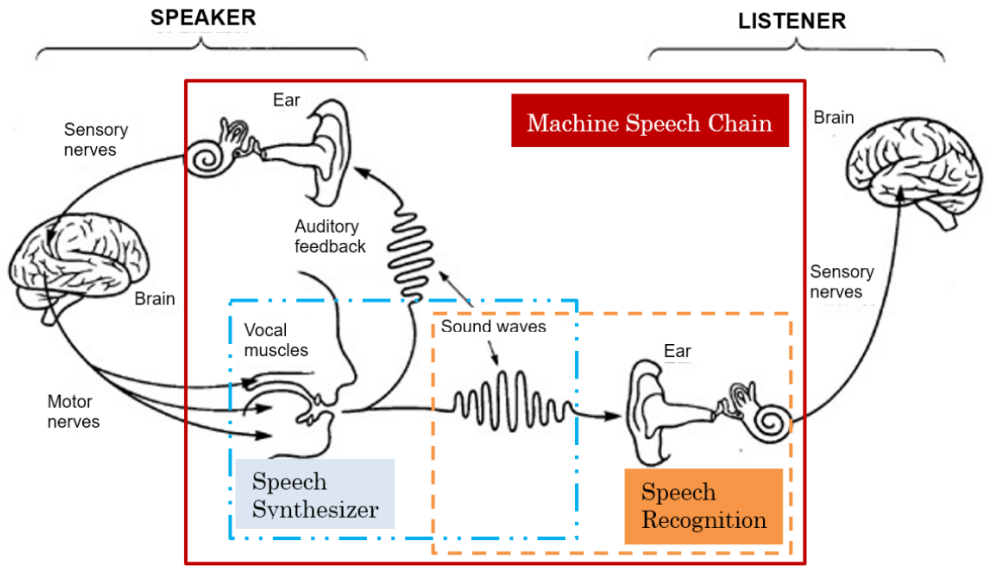

Figure 1.2: Speech chain [9] and the machine speech chain [10]. The figure is directly extracted from [10].

In speech technology, one could think of a text-to-speech (TTS) system as the speaker and an automatic speech recognition (ASR) system as the listener. In the neural network era, while ASR and TTS systems may borrow the overall architecture and system designs from each other, they were largely trained and evaluated independently. This relative "independence" is vastly different from how humans learn to communicate.

In recent years, semi-supervised learning with unpaired speech and text became increasingly popular, especially as the community generalizes neural ASR and TTS systems to low-resource languages with limited paired data [11, 12]. The unpaired speech and text data were utilized in multiple ways, including

1. de-noising auto encoder (DAE) training by concatenating the signal paths from the speech encoder (ASR) to the speech decoder (TTS), and the text encoder (TTS) with the text decoder (ASR);

2. dual transformation, where the ASR system transcribed unpaired speech into a pseudo-corpus for use by TTS, and the TTS system synthesized speech from unpaired text for use by ASR;

3. knowledge-distillation, where unpaired speech and text was used for self-training.

While the DAE loss can be considered an application for enforcing cycle consistency, it still solves a slightly different problem from speech recognition or speech synthesis. Other work focused on more literal adaptations of the speech chain, especially the feedback loop where the speaker serves as his own listener. These works usually cascaded a TTS system on top of the ASR decoder and backpropagated the gradient from the TTS system to the ASR system as an end-to-end feedback loop [10, 13, 14].

We can view the formulation of cascaded ASR-TTS as a constrained minimization problem. Given a speech recognizer $\mathcal{G}_{ASR}$ with parameters θ_{ASR} and a speech synthesizer as $\mathcal{G}_{TTS}$ with parameters θ_{TTS}, we could view the recognition loss (such as the CTC loss [15]) as the main objective function. If we denote the i-th speech input as $\mathbf{X}_i$, and the reconstruction from the cascaded ASR-TTS system as $\hat{\mathbf{X}}_i := \mathcal{G}_{TTS}\left(\mathcal{G}_{ASR}\left(\mathbf{X}_i\right)\right)$, we would like to solve the following problem:

$$\begin{aligned} &\min_{\theta_{ASR}} -\sum_{i=1}^{n} \log \mathbf{P}_{\theta_{ASR}}\left(\mathbf{z}_i \mid \mathbf{X}_i\right) \\ \text{s.t.} \quad &\left\|\mathcal{G}_{TTS}\left(\mathcal{G}_{ASR}\left(\mathbf{X}_i\right)\right) - \mathbf{X}_i\right\| = 0 \quad \forall i \in \{1, \ldots, N\} \end{aligned} \tag{1.1}$$

where N is the number of training samples and $\mathbf{z}_i$'s are the target sequences. This can be easily transformed into a Lagrangian optimization problem as:

$$\min_{\theta_{ASR}} \sum_{i=1}^{n} \left[-\log \mathbf{P}_{\theta_{ASR}}\left(\mathbf{z}_i \mid \mathbf{X}_i\right) + \lambda \left\|\mathcal{G}_{TTS}\left(\mathcal{G}_{ASR}\left(\mathbf{X}_i\right)\right) - \mathbf{X}_i\right\|\right] \tag{1.2}$$

The experiments in Chapter 4 differ from previous work in some aspects. First, we would like to explore directly applying the cycle-consistency constraint to multi-lingual, multi-speaker speech recognition without additional unpaired data. Second, the multi-lingual ASR corpora contain both speaker and language modalities and are relatively small in size. There are three languages from the same language family, and each language only has around 20 to 30 hours of speech but around 60 to 80 training speakers. Therefore, it can be challenging for a TTS system to model the language and speaker modalities with limited data. We chose to condition the TTS system on both

speaker and language embedding extracted from the input utterance, using a pre-trained speaker encoder and a language encoder. We trained the two encoders directly on the training set for the three languages, which could increase the specificity of the speaker and language vectors. Third, we opt to replace recurrent networks with Transformer architecture, which allows the entire system to model additional complexities and train more quickly.

We defer a detailed explanation of the algorithms, including the architecture and loss functions for all model components, the data, model and training settings, and the results, including some visualizations, to Chapter 4. As a preview, the results in Chapter 4 did not demonstrate the ASR-TTS feedback system as successful, as it failed to beat the baseline for all training and test languages. We hypothesize the reasons for the inferior performance and future directions in Section 6.2.

1.3 Tackling Language-dependent Phoneme Inventories With Invariant Embedding

Speech production is a nonlinear process. Any given articulatory movement—say, a shift of 1 cm in the position of the tongue tip—may cause a huge change in the produced acoustic spectrum or a minuscule change in the spectrum, depending on the articulatory position from which the movement started. Let us use the words "unstable" vs. "stable," respectively, to denote articulations from which small deviations cause large vs. small acoustic consequences. A learner imitating adult speech tends to have greater success in imitating stable rather than unstable articulations because stability permits accurate acoustic imitation despite it being imprecise. For this reason, phonemes tend to correspond to stable articulations, and unstable articulations tend to mark the boundaries between pairs of phonemes [16]. The number of unstable configurations is larger than the number of phoneme distinctions in any known language; therefore, each language chooses a subset to use as phoneme boundaries, e.g., some languages treat the phones /θ/ (as in "thin") and /s/ (as in "sin") as distinct phonemes, while in other languages, they are both considered to be acceptable pronunciations of the same phoneme. A language-independent ASR is an automatic speech recognizer trained to recognize all of the articulatory features that may be used to signal phoneme

distinctions in any of the world's languages.

The relationships among phoneme inventories of different languages are complicated, however, by tremendous cross-lingual divergence in the use of redundant features [17]. No language uses all of the available articulatory features to define phonemes; hence, every language has some extra articulatory features that can be used to add redundancy to its phoneme code. In modern English, for example, the feature of plosive voicing (/d/ vs. /t/) is often enhanced by the feature of aspiration (/d/ vs. /t^h/), while the tense-lax vowel distinction (/i/ vs. /ɪ/) is often enhanced by the feature of lengthening (/iː/ vs. /ɪ/). In both of these cases, it is possible to identify one feature as *phonemic* and another as *redundant* because, in each case, the redundant feature can be modified without changing the meaning of the word (/biːt^h/ and /bit/ are both "beat"). Redundant features add robustness to speech in much the same way that an error-correcting code adds robustness to digital communication systems: imprecise production or noisy perception are less likely to cause communication errors if every phoneme is redundantly specified. Because redundant features improve the efficiency of speech communication, they are ubiquitous.

Because redundant features are defined separately for every language, however, they cause significant problems in training language-independent ASR. A typical ASR training corpus is a set of labeled examples, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \sim X$ is a speech waveform, and $y_i \sim Y$ is the corresponding text transcript (the notation $x_i \sim X$ means that x_i is an instance of the random variable X). We can safely assume that certain transformations are information-preserving, e.g., a waveform can be converted to or from a spectrogram without loss of information [18]; therefore, we can consider both to be equivalent representations of the random variable X. Similarly, in any well-resourced language, a pronunciation lexicon can be used to convert text transcripts to phoneme transcripts encoded using the international phonetic alphabet (IPA [19]); therefore, we can consider text transcripts and IPA phonemic transcripts to be equivalent representations of the random variable Y. The key obstacle to language-independent ASR is that phonemic transcripts are not the same as language-independent phonetic transcripts. The English word "beat," for example, has the same phonemic transcript (y_i =[bit]), regardless of whether or not the vowel is lengthened (/biːt/ vs. /bit/), and regardless of whether the final consonant is aspirated,

unreleased, glottalized, or replaced by a glottal stop (/bit$^{\text{h}}$/, /bit$^{\urcorner}$/, /bit$^{\text{ʔ}}$/, or /biʔ/). The phonetic sequences /biːt$^{\text{h}}$ət/ and /biʔət/ are different words in Arabic ("home" and "environment," respectively), but an ASR trained using English data would be unable to distinguish them. Similarly, a plosive voicing detector trained on English fails to correctly recognize Spanish unvoiced plosives, which are not aspirated [20], or Hindi voiced aspirated and unvoiced unaspirated plosives [21]. A vowel classifier trained on English is able to recognize the duration differences of some Japanese vowel pairs but not others [22]. A Mandarin vowel classifier, applied to English vowels, finds American English /u/ to be closer to the Mandarin central unrounded vowel /ɨ/ than to the Mandarin /u/ [23].

Apparently, what is needed is some intermediate representation capable of compensating for language-dependent differences in the use of redundant features. The work in Chapter 5 proposes the use of an **invariant embedding**, $z \sim Z$, defined to be a high-dimensional signal representation with no information about the language-dependent redundant articulatory features. The invariant embedding allows us to train a language-independent ASR using a large number of language-dependent training corpora. Each language-dependent training corpus contains a number of tuples of the form $\mathcal{D} = \{(x_1, e_1, y_1), \ldots, (x_n, e_n, y_n)\}$, where $e_i \in \mathcal{E}$ specifies the language and dialect being spoken, and the transcriptions are language-dependent phonemic transcripts rather than language-independent phonetic transcripts: $y_i = f(x_i, e_i)$. The invariant embedding is trained to ignore language-dependent redundant features in X and encode only the features that correspond to Y in a language-independent way. The resulting mapping $w : Z \rightarrow Y$ thus becomes language-independent ASR. This is shown in Figure 1.3.

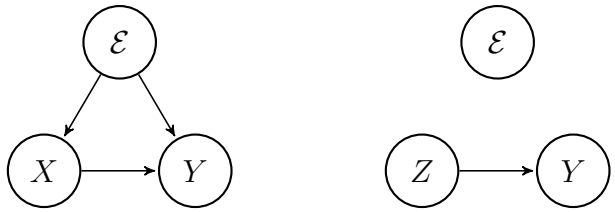

Figure 1.3: The phonemic transcript, Y, captures a limited set of information about the speech signal, X. The limits of the transcription process are dependent on the language environment, $\mathcal{E}$. Language-independent ASR finds a feature embedding, $Z = \phi(X)$, such that the relationship between Z and Y is independent of $\mathcal{E}$.

For example, suppose that x_1 and x_2 are two different waveforms, each examples of the English word "beat," meaning that they both have exactly the same label sequence, $y_1 = y_2 =$[bit]. Suppose that fine phonetic transcriptions of these two waveforms would detect some differences, e.g., perhaps x_1 sounds like /biːtʰ/, while x_2 sounds like /biʔ/. The purpose of the invariant embedding is to eliminate these fine phonetic differences so that if one were to convert the invariant embedding back into an acoustic signal, the language-independent fine phonetic transcription of that acoustic signal would be exactly the sequence /bit/. In this way, a language-independent speech recognizer, capable of mapping $f : X \to Y$, is decomposed into two subsystems: (1) a feature extraction system computes features $Z = \phi(X)$ such that (2) the mapping $w : Z \to Y$ is independent of the language environment.

Recognition based on invariant features has a long history in machine learning. In ASR, speaker normalization of the acoustic features, or of an invariant embedding, can be performed using methods such as cepstral mean and variance normalization [24], vocal tract length normalization [25], feature-based maximum likelihood linear regression [26], or auxiliary networks [27]. In computer vision, signal processing methods were developed that resulted in features invariant to changes in an object's scale, rotation, and illumination [28]. Early proponents of 2D convolutional neural networks (CNNs) argued that CNNs provide a degree of shift-invariance in computer vision [29] and a degree of speaker-invariance in ASR [30]. Recent work has attempted to characterize invariant representations in a more general way to make it possible to learn a representation that is invariant to any type of environment

mismatch [31, 32]. The goal of domain invariance has recently been called into question, however, by a line of research suggesting that machine learning algorithms trained using standard empirical risk minimization procedures automatically learn domain-invariant embeddings, even when not explicitly trained to do so [33].

To our knowledge, the tools of domain-invariant machine learning have not yet been applied to the task of language-independent ASR. A sequence of experiments were carried out to test the following hypotheses regarding how well regret minimization [32], a particular domain invariance method, works for multilingual and cross-lingual speech recognition. Not all of these hypotheses were experimentally verified; the experimental truth or falsehood of each hypothesis is noted briefly here and is supported by the evidence presented in Chapter 5.

- **H1:** Regret minimization, a domain-invariant machine learning method (RGM, [32]) can be used to optimize an end-to-end (E2E) neural network ASR so that it more effectively generalizes from fifteen training languages to five novel test languages, as compared to a baseline ASR trained using a standard training criterion called empirical risk minimization (ERM). **Experimental result:** this hypothesis is demonstrated to be false by the experiments presented in Chapter 5.

- **H2:** RGM, as compared to ERM, can be applied to optimize an E2E ASR so that it more effectively generalizes from training languages in one language family to test languages in a different language family. **Experimental result:** partially true.

- **H3:** The optimal training regimen for phone token classification (given known phone token boundary times) is different from the optimal training regimen for phone token recognition (with unknown boundary times). **Experimental result:** true. Experiments described in Chapter 5 find that empirical risk minimization (ERM) is superior to regret minimization (RGM) for both multilingual recognition and classification, but under certain scenarios and validation schemes, RGM can be superior for cross-lingual recognition and classification.

As shown in Section 2.4.2, for a two-environment case, regret minimization

can be formulated as the following constrained minimization problem:

$$f_{RGM} = \min_{w,\phi} \mathcal{R}(w \circ \phi)$$
$$\text{s.t.} \quad \mathcal{R}^e(w^{-e} \circ \phi) - \mathcal{R}^e(w^e \circ \phi) = 0 \quad \forall e \tag{1.3}$$

where $\mathcal{R}(\cdot)$ denotes the risk over all environments, ϕ is a feature extractor, and w is an environment-independent classifier. w^{-e} is the classifier trained without access to environment e, w^e is trained on environment e only, and $\mathcal{R}^e(\cdot)$ is the risk on environment e. This in turn can be transformed into the following Lagrangian minimization problem:

$$f_{RGM} = \min_{w,\phi} \mathcal{R}(w \circ \phi) + \lambda \sum_e [\mathcal{R}^e(w^{-e} \circ \phi) - \mathcal{R}^e(w^e \circ \phi)] \tag{1.4}$$

We defer the algorithm modifications to suit multi-lingual ASR with more than two training languages to Chapter 5. After that, we will also discuss data and model settings for the experiments designed to prove or disprove the above hypotheses.

CHAPTER 2

BACKGROUND

In the background chapter, we will introduce end-to-end neural speech recognition, focusing on relevant architectures and loss functions. This is relevant throughout Chapter 3, 4, and 5. We will then give an introduction to end-to-end neural speech synthesis, which is relevant to Chapter 4. We will then talk about the concept of cycle consistency and its applications, focusing on prior work in speech technology. Finally, we will discuss the theoretical formulation of invariant risk minimization and regret minimization.

2.1 End-to-end Neural Speech Recognition

In this section, we will give a brief introduction to end-to-end neural speech recognition. End-to-end neural speech recognition involves training a neural network that takes unsegmented speech representations as input and directly outputs text transcripts. These text transcripts can be either characters, words, or phones. We will first review connectionist temporal classification, which allows end-to-end training with unsegmented input sequences. We will then review two architectures, Deep Speech 2 and transformer. These two architectures are widely used in academics and industry alike and form the backbone of the experiments in later chapters.

2.1.1 Connectionist Temporal Classification

Connectionist temporal classification (CTC) [15] allows end-to-end training with unsegmented input sequences. Suppose we are given an input sequence with T frames $\mathbf{X} := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, a target sequence of length U $\mathbf{z} := (z_1, z_2, \ldots, z_U)$, where $U \leq T$, and a neural network $\mathcal{N}_w$ that outputs a softmax probability over $K+1$ labels every time-step. K is the output token

inventory size, and the additional label is for a special output token called the blank label. Let $\mathbf{Y} := \mathcal{N}_w(\mathbf{X}) = (\mathbf{y}_1, \ldots, \mathbf{y}_k)$, where the output vector at time t is defined as $\mathbf{y}_t := \left(y_t^0, \ldots y_t^K\right)$. y_t^k is the output probabilities at time step t for the k-th output token in the softmax layer. Suppose we are given a frame-wise alignment $\pi := (\pi_1, \pi_2, \ldots, \pi_T)$, then assuming that the per-frame output probability are independent, the joint probability of such an alignment given input $\mathbf{X}$ can be defined as

$$\mathbf{P}(\pi \mid \mathbf{X}) = \prod_{t=1}^{T} y_t^{\pi t} \tag{2.1}$$

Let L be the set of output labels in the target sequences $\mathbf{z}$'s, and let $L^{'} := L \cup \{blank\}$. The CTC algorithm first defines a many-to-one mapping $\mathcal{B} : {L^{'}}^T \mapsto L^{\leq T}$ that simply removes all blanks and repeated tokens not separated by blanks. With such a mapping, the conditional probability of the target sequence $\mathbf{z}$ given input $\mathbf{X}$ can be calculated as:

$$\mathbf{p}(\mathbf{z} \mid \mathbf{X}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} \mathbf{P}(\pi \mid \mathbf{X}) \tag{2.2}$$

Similar to the forward-backward algorithm for hidden Markov models, the CTC algorithm calculates a forward probability $\alpha_t(s)$ and a backward probability $\beta_t(s)$ on the modified label sequence $\mathbf{z}^{'}$ with length $2U+1$, which is obtained from $\mathbf{z}$ by inserting blank tokens at the start and end, as well as between every two non-blank tokens as:

$$\alpha_t(s) := \sum_{\substack{\pi \in {L^{'}}^T \\ \mathcal{B}(\pi_{1:t}) = \mathbf{z}^{'}_{1:s}}} \prod_{t'=1}^{t} y_{t'}^{\pi_{t'}} \tag{2.3}$$

which is the total probability of the first s tokens of the target sequence being $\mathbf{z}^{'}_{1:s}$ at time t, and

$$\beta_t(s) := \sum_{\substack{\pi \in {L^{'}}^T \\ \mathcal{B}(\pi_{t:T}) = \mathbf{z}^{'}_{s:U}}} \prod_{t'=t}^{T} y_{t'}^{\pi_{t'}} \tag{2.4}$$

which is the total probability of the last $2U - s + 2$ tokens of the target sequence being $\mathbf{z}^{'}_{s:2U+1}$ at time t.

To initialize $\alpha_t(s)$, CTC allows all prefixes to start with either a blank (denoted by b onwards) or the first symbol in $\mathbf{z}$. To initialize $\beta_t(s)$, CTC allows all suffixes to end with either a blank or the last symbol in $\mathbf{z}$. To recursively update $\alpha_t(s)$ and $\beta_t(s)$, the CTC algorithm defines valid transitions as those between blank and non-blank tokens, as well as those between any pair of distinct non-blank tokens. If we imagine the modified sequence $\mathbf{z}^{'}$ as a state machine, where every arc corresponds to advancing a time step, we would have self loops on blank and non-blank tokens, advance-by-one arcs between consecutive tokens, and advance-by-two arcs only between non-blank tokens that are not the same. This leads to the following initialization and recursive updates for $\alpha_t(s)$ and $\beta_t(s)$:

$$\alpha_1(1) = y_1^b \tag{2.5}$$

$$\alpha_1(2) = y_1^{\mathbf{z}_1} \tag{2.6}$$

$$\alpha_1(s) = 0, \forall s > 2 \tag{2.7}$$

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1))\, y_t^{\mathbf{z}'_s} & \text{if } \mathbf{z}'_s = b \text{ or } \mathbf{z}'_{s-2} = \mathbf{z}'_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2))\, y_t^{\mathbf{z}'_s} & \text{otherwise} \end{cases} \tag{2.8}$$

$$\beta_T(2U+1) = y_T^b \tag{2.9}$$

$$\beta_T(2U) = y_T^{\mathbf{z}_U} \tag{2.10}$$

$$\beta_T(s) = 0, \forall s < 2U \tag{2.11}$$

$$\beta_t(s) = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1))\, y_t^{\mathbf{z}'_s} & \text{if } \mathbf{z}'_s = b \text{ or } \mathbf{z}'_{s+2} = \mathbf{z}'_s \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2))\, y_t^{\mathbf{z}'_s} & \text{otherwise} \end{cases} \tag{2.12}$$

For back-propagation, it suffices to calculate $\frac{\partial \ln(\mathbf{P}(\mathbf{z}|\mathbf{X}))}{\partial y_t^k}$. Note that

$$\frac{\alpha_t(s)\beta_t(s)}{y_t^{\mathbf{z}'_s}} = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{z}): \\ \pi_t = \mathbf{z}'_s}} \mathbf{P}(\pi \mid \mathbf{X}) \tag{2.13}$$

Comparing with Equation (2.2)

$$\mathbf{P}(\mathbf{z} \mid \mathbf{X}) = \sum_{s=1}^{2U+1} \frac{\alpha_t(s)\beta_t(s)}{y_t^{\mathbf{z}'_s}} \tag{2.14}$$

we would get

$$\frac{\partial \ln(\mathbf{P}(\mathbf{z} \mid \mathbf{X}))}{\partial y_t} = \frac{1}{\mathbf{P}(\mathbf{z} \mid \mathbf{X})} \frac{\partial \mathbf{P}(\mathbf{z} \mid \mathbf{X})}{\partial y_t^k} \tag{2.15}$$

$$= \frac{\sum_{s \in \text{lab}(\mathbf{z},k)} \alpha_t(s)\beta_t(s)}{(\alpha_T(2U+1) + \alpha_T(2U))\, y_t^{k^2}} \tag{2.16}$$

where $\text{lab}(\mathbf{z}, k) = \{s : \mathbf{z}'_s = k\}$.

After the model converges, we may simply apply greedy decoding with $\pi^* = \arg\max_{\pi \in L'^T} \mathbf{P}(\pi \mid \mathbf{X})$ followed by $\mathbf{z}^* = \mathcal{B}(\pi^*)$. This might not give the best possible labeling, so usually beam search [34, 35] is used.

2.1.2 Deep Speech 2

Deep Speech 2 [36] is one of the successful neural architectures for end-to-end neural speech recognition. In general terms, the model applies several convolutional layers on top of a spectrogram, followed by several recurrent and fully connected layers. The entire network is optimized under the CTC loss with character targets. In more details, let h^l denote the output from the l-th layer, where h^0 denotes the input. The first several layers of convolution operators can be expressed as:

$$h^l_{t,i} = f\left(w^l_i \circ h^{l-1}_{t-c_l:t+c_l}\right) \quad \text{1D convolution} \tag{2.17}$$

$$h^l_{t,r,i} = f\left(w^l_i \circ h^{l-1}_{t-c_l:t+c_l,r-c'_l:r+c'_l}\right) \quad \text{2D convolution} \tag{2.18}$$

where t is the time index and r is the generalized frequency index (as in "image height" in 2D convolution). i denotes the i-th output channel, and $f(\cdot)$ denotes the non-linearity. Note that the element-wise multiplication operator $\circ$ operates over all input channels of h^{l-1} within the respective receptive field of the filters w^l_i. The authors used convolutional layers to model translational invariance in spectrograms and speaker variability. They experimented with one to three layers of 1D or 2D convolutions under different strides.

Following the convolutional layers are bidirectional recurrent layers. They experimented with regular bidirectional RNN layers [37], which they define as

$$\overrightarrow{h}_t^l = f\left(\overrightarrow{W}^l h_t^{l-1} + \overrightarrow{U}^l \overrightarrow{h}_{t-1}^l + \overrightarrow{b}^l\right) \tag{2.19}$$

$$\overleftarrow{h}_t^l = f\left(\overleftarrow{W}^l h_t^{l-1} + \overleftarrow{U}^l \overleftarrow{h}_{t-1}^l + \overleftarrow{b}^l\right) \tag{2.20}$$

$$h^l = \overrightarrow{h}^l + \overleftarrow{h}^l \tag{2.21}$$

as well as GRU layers [38], where the forward direction $\overrightarrow{h}_t^l$ is calculated as

$$\overrightarrow{z}_t^l = \sigma\left(\overrightarrow{W}_z^l h_t^{l-1} + \overrightarrow{U}_z^l \overrightarrow{h}_{t-1}^l + \overrightarrow{b}_z^l\right) \tag{2.22}$$

$$\overrightarrow{r}_t^l = \sigma\left(\overrightarrow{W}_r^l h_t^{l-1} + \overrightarrow{U}_r^l \overrightarrow{h}_{t-1}^l + \overrightarrow{b}_r^l\right) \tag{2.23}$$

$$\overrightarrow{\tilde{h}}_t^l = f\left(\overrightarrow{W}_h^l h_t^{l-1} + \overrightarrow{r}_t^l \circ \overrightarrow{U}_h^l \overrightarrow{h}_{t-1}^l + \overrightarrow{b}_h^l\right) \tag{2.24}$$

$$\overrightarrow{h}_t^l = \left(1 - \overrightarrow{z}_t^l\right) \overrightarrow{h}_{t-1}^l + \overrightarrow{z}_t^l \overrightarrow{\tilde{h}}_t^l \tag{2.25}$$

and the backward direction is calculated in a similar fashion, with its individual weight matrices and taking h_{t+1}^l instead of h_{t-1}^l. $\sigma(\cdot)$ is the sigmoid function. z and r represent the update and reset gates, respectively.

They also experimented with a unidirectional variant that uses a lookahead operator with trainable weights W to incorporate some future information:

$$r_{t,r}^l = \sum_{j=1}^{\tau+1} W_{r,j}^l h_{t+j-1,r}^l \tag{2.26}$$

where r denotes the row dimension, and h^l denotes the output from a unidirectional recurrent layer.

The recurrent layers they use incorporate sequence-wise Batch Normalization [39]. This simply means that when calculating $\mathcal{B}(x) = \gamma \frac{x - \mathrm{E}[x]}{(\mathrm{Var}[x] + \epsilon)^{1/2}} + \beta$, the mean and variance statistics for $x := h_t^l$ is calculated over all samples in the current batch and over all valid time steps in each sample. The recurrent layers are followed by fully-connected layers with $h_t^l = f\left(W^l h_t^{l-1} + b^l\right)$ and then a softmax over the characters and a blank token $\mathbf{P}\left(z_t = k \mid X\right) = \frac{\exp\left(w_k^L \cdot h_t^{L-1}\right)}{\sum_j \exp\left(w_j^L \cdot h_t^{L-1}\right)}$.

During training, they incorporated SortaGrad, which simply means that

in the first epoch, they sorted the utterances by length and iterated through the training samples in increasing utterance lengths. This is a heuristic that feeds utterances that are shorter and easier to align to the CTC loss before longer and harder ones.

During inference, they find the most probable transcript y by maximizing over

$$Q(y) = \log\left(\mathbf{P}_{\text{ctc}}(y \mid x)\right) + \alpha \log\left(\mathbf{P}_{\text{lm}}(y)\right) + \beta \,\text{word\_count}(y) \tag{2.27}$$

under a fixed number of beams [35].

2.1.3 Transformer

The Transformer [40] architecture has been widely used in various sequence modeling tasks since its proposal. The transformer encoder-decoder structure is solely based on the attention mechanism. Compared to recurrent architectures, transformers do not rely on obtaining the hidden state h_{t-1}^{l} from the previous time step before calculating the hidden state h_{t}^{l} for the current time step. This enables better parallelism and makes the model less susceptible to vanishing gradients. Compared to convolutional architectures, transformers do not depend on stacking multiple layers or large dilation factors to capture long-range dependencies. A detailed comparison of the computation complexity between different architectures can be found in Table 1 of the original proposal for the transformer architecture [40].

Transformers use a special type of attention called multi-head attention. Multi-head attention extends the notion of scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.28}$$

where Q is the query, K, V denotes a key-value pair and d_k denotes the feature dimension of Q and K (i.e., the dimension that the dot-product operates on). Given this operation, the multi-head attention is simply defined as a

concatenation of h different scaled dot-product attentions:

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \tag{2.29}$$

$$\text{where head}_\text{i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2.30}$$

where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$,$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. Usually $d_k = d_v = d_{\text{model}}/h$.

Figure 2.1, which is extracted from the original proposal for the transformer architecture [40], shows the transformer encoder-decoder architecture for sequence transduction tasks. First, the input and output are converted to their respective representations. The output is shifted to the right to preserve auto-regressive property and to prevent the model from "seeing the answer." Positional encoding is added to both input and output embeddings to infuse relative or absolution positional information. The encoding is simply defined by sine and cosine functions:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \tag{2.31}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \tag{2.32}$$

where i is the feature dimension and pos is the position in the time dimension. The encoder consists of self-attention layers, where queries, keys, and values all come from the outputs of the previous encoder layer. For the first layer, they correspond to the input embedding with positional encoding. The decoder contains two types of attention, self-attention, and encoder-decoder attention. Each self-attention layer in the decoder only allows each time step in the decoder to attend to all previous time steps. This can be done by masking the attention matrices $\frac{QK^T}{\sqrt{d_k}}$. Following every single decoder self-attention module is an encoder-decoder cross-attention module, where the keys and values come from the encoder outputs, and queries come from the preceding self-attention layer. Queries, keys, and values for decoder self-attention layers all come from the output of the previous decoder layer. For the first layer, they are the output embedding (shifted to the right) with positional encoding. Each encoder or decoder layer, in addition to their respective attention modules, contains a position-wise feed-forward network

(FFN), defined as

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{2.33}$$

where x is from the last attention module of the current layer.

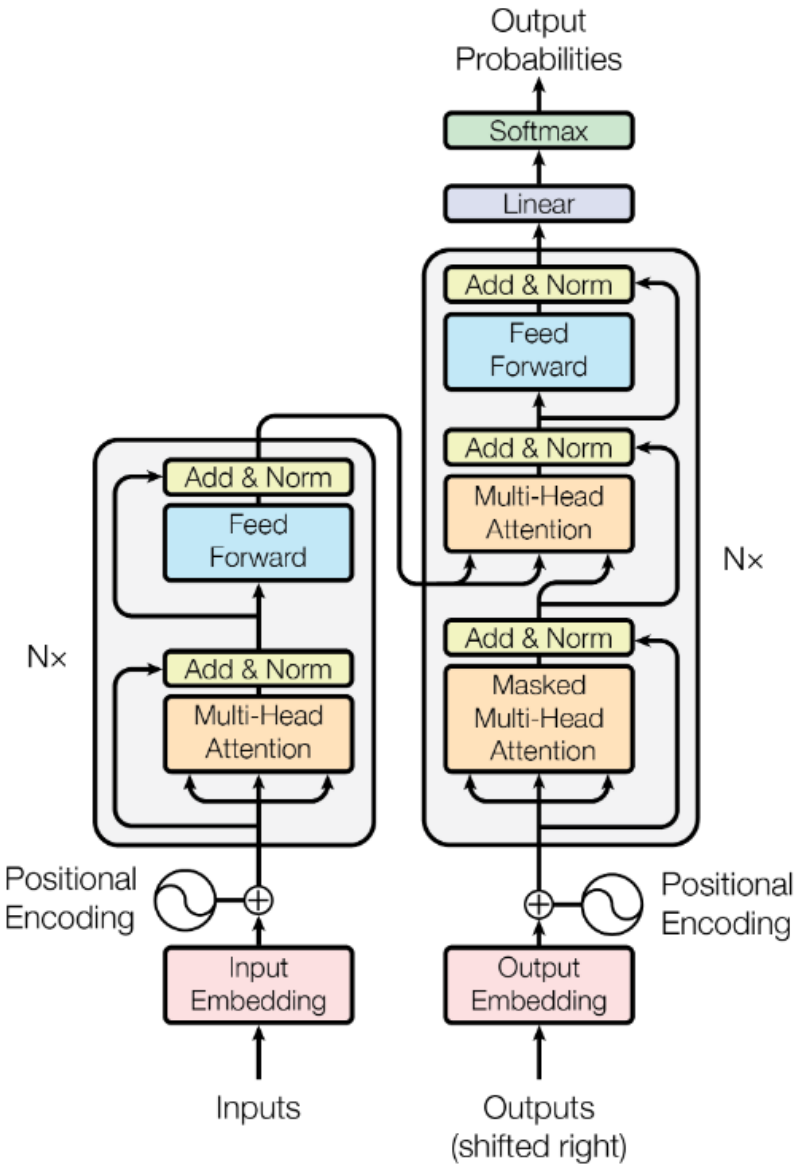

Figure 2.1: The transformer encoder-decoder architecture from the original proposal for the transformer architecture [40].

The Speech-Transformer [41] was one of the first adaptations of transformers to speech recognition. The model achieves similar performance to other

seq2seq architectures but with much reduced computational requirements. A faster and more accurate ASR system that combines the transformer architecture and the advances in RNN-based ASR [42] added a CTC loss for the transformer encoder for multi-task training with decoder cross-entropy loss. They also implemented joint CTC + attention decoding with an external language model [43, 44]. A multi-lingual transformer network trained on 13 languages achieved significantly improved phonetic token error rates over the mono-lingual baselines [45]. The model was trained with a CTC loss on the transformer encoder and a cross-entropy loss on the decoder. During inference, joint CTC + attention decoding [42] was used. Transformers also allow the development of powerful self-supervised models that learn from raw speech directly [46]. The speech representations from such models are shown to be better than traditional speech features such as MFCCs.

2.2 End-to-end Neural Speech Synthesis

This section will review two architectures for end-to-end neural speech synthesis, Tacotron 2 and Transformer-TTS. Tacotron 2 is based on Long Short-Term Memory (LSTM) networks and provides a general encoder-decoder framework for predicting mel-spectrograms conditioned on text. Transformer-TTS uses transformers instead of LSTMs. The discussion in this section is necessary for understanding cycle consistency in speech, and the Transformer-TTS architecture is used in Chapter 4 for building up the modified ASR-TTS framework.

2.2.1 Tacotron 2

Tacotron 2 uses a sequence-to-sequence recurrent architecture to directly predict a mel-spectrogram given text. The mel-spectrogram then serves as input into a modified WaveNet [47] to produce raw speech waveform. The overall architecture is shown in Figure 2.2, which is extracted from the original proposal of Tacotron 2 [48].

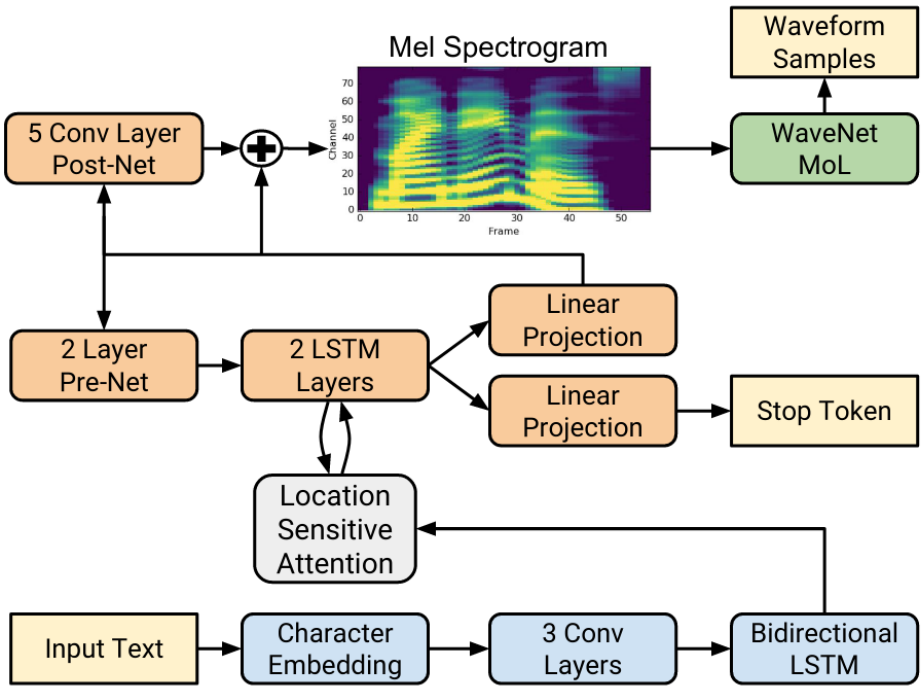

Figure 2.2: Tacotron 2 system architecture from the original proposal of Tacotron 2 [48]..

The model takes an input text sequence and produces a sequence of learnable 512-dimensional character embeddings. The embeddings are then fed into a stack of three 1D convolutional layers, each with a filter size of five and 512 output channels. The convolutional layers are followed by Batch Normalization and ReLU activations. The output of the convolutional layers serves as input into a bidirectional LSTM layer, where each direction has a dimension of 256, and the final encoder output is the concatenation of the output from the two directions. A forward LSTM layer can be defined with the following set of operations:

$$i_t = \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) \tag{2.34}$$

$$f_t = \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) \tag{2.35}$$

$$g_t = \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right) \tag{2.36}$$

$$o_t = \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) \tag{2.37}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.38}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{2.39}$$

where i_t, f_t, g_t, o_t are the input, forget, cell and output gates, respectively. c_t is the cell state and h_t is the output hidden state of the current time step. x_t is either the input or the output of the previous layer, and h_{t-1} is replaced by h_{t+1} for the backward direction.

The output from the bidirectional LSTM is used to calculate location-sensitive attention [49]. Denote the encoder output as $h := (h_1, \ldots, h_L)$, location-sensitive attention can be computed as:

$$\alpha_i = \text{Attend}\left(s_{i-1}, \alpha_{i-1}, h\right) \tag{2.40}$$

$$g_i = \sum_{j=1}^{L} \alpha_{i,j} h_j \tag{2.41}$$

where $\alpha_i \in \mathbb{R}^L$ are the attention weights at time step i, s_i are the recurrent hidden states and g_i is the attended representation. The Attend$(\cdot, \cdot, \cdot)$ operation is calculated as:

$$f_i = F * \alpha_{i-1} \tag{2.42}$$

$$e_{i,j} = w^\top \tanh\left(W s_{i-1} + V h_j + U f_{i,j} + b\right) \tag{2.43}$$

$$\alpha_{i,j} = \exp\left(e_{i,j}\right) / \sum_{j=1}^{L} \exp\left(e_{i,j}\right) \tag{2.44}$$

where $f_{i,j} \in \mathbb{R}^k$ are the results of convolving α_{i-1} with $F \in \mathbb{R}^{k \times r}$, w, b are learnable vectors and W, V, U are learnable matrices. In Tacotron 2, the attention dimension is chosen to be 128, and the convolution has 32 output channels with a filter size of 31.

During training, each predicted frame $\hat{y}_t$ is conditioned on the encoder output h and y_{t-1} from the ground-truth mel-spectrogram. y_{t-1} goes through two layers of a feedforward network with size 256 and ReLU activations, which are denoted as prenet. The prenet output and the attended representation g_t go through two layers of uni-directional LSTM with 1024 hidden size, which serves as the decoder. The decoder output and g_t are concatenated and projected to the mel-spectrogram dimension to predict the frame at time t. The coarse spectrogram is fine-tuned with a residual, which is calculated by feeding the coarse representation into five layers of 1D convolution with batch normalization, each with 512 output channels, a filter size

of 5, and *tanh* activations in between. The coarse and fine spectrogram representations are both used to calculate a mean-squared error, and the error terms are summed. The concatenation of decoder output and g_t are also projected down to a scalar to calculate the binary cross-entropy of the stop token. Dropout with a rate of 0.5 is used to regularize convolutional layers, and zoneout with a rate of 0.1 is used to regularize recurrent layers.

A modified WaveNet vocoder is trained for inference. The WaveNet vocoder contains 30 dilated convolutions, and the dilation factor at layer k is $2^{k(\bmod 10)}$. Two upsampling layers are used, and the parameters for the mixture of logistic distributions (MoL) are calculated by feeding the convolution stack output through a ReLU and a linear projection. MoL generates the 16-bit samples, and the network weights are updated via maximum likelihood. During inference, the predicted frame $\hat{y}_{t-1}$ goes into the decoder prenet to auto-regressively predict $\hat{y}_t$. The predicted mel-spectrogram then goes into the trained WaveNet vocoder to produce the speech waveform. The synthetic speech from Tacotron 2 achieves a mean opinion score (MOS) of 4.53, which is very close to the ground-truth MOS of 4.58.

2.2.2 Transformer-TTS

The Transforer-TTS model [50] replaced the recurrent encoder-decoder structures in Tacotron 2 with a transformer encoder-decoder framework to improve training efficiency and better capture long-term dependency. As shown in Figure 2.3, which is directly extracted from the original proposal of Transformer-TTS [50], the vanilla transformer architecture can be readily adapted for synthesizing mel-spectrograms conditioned on text.

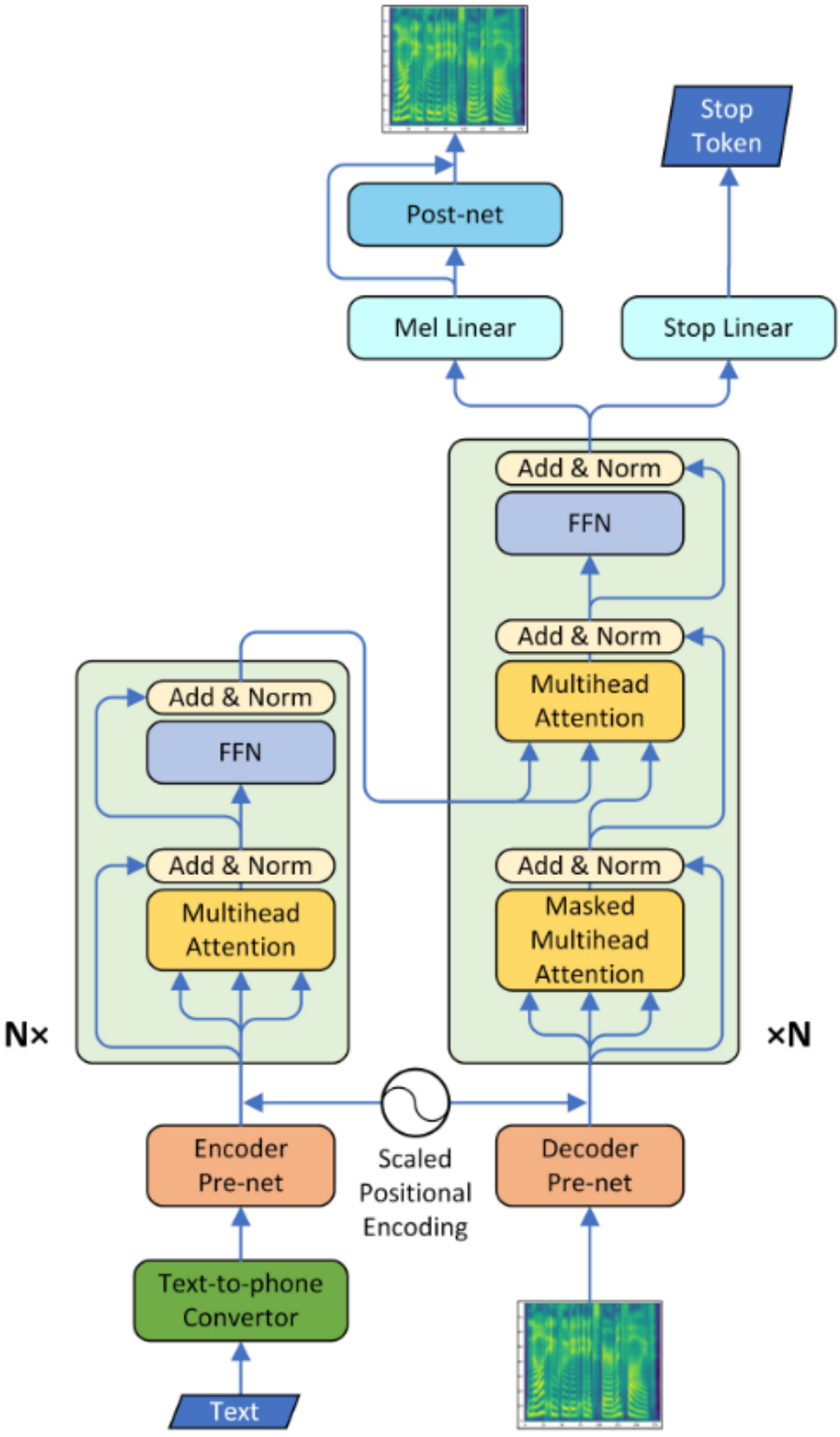

Figure 2.3: Transformer-TTS system architecture from the original proposal of Transformer-TTS [50].

Instead of character inputs as in Tacotron 2, Transformer-TTS first converts text into phonemes and then phoneme embeddings of dimension 512. The input goes through the same three-layer CNN as in Tacotron 2, except for another linear layer after the final ReLU for zero-centering. The CNN and the additional linear layer are denoted as encoder pre-net in Figure 2.3. The decoder pre-net has the same architecture as in Tacotron 2, which projects the mel-spectrogram representation and phoneme representation to the same space. The author also found that the ReLU non-linearity in the decoder pre-net is important for learning encoder-decoder attention for Transformer-TTS. Increasing the dimension of the decoder pre-net further does not improve performance. The decoder pre-net also contains an additional linear layer for zero-centering. While they use the same positional encoding representation as in Expression 2.1.3, they added a learnable scale parameter α so that the encodings could better fit the source domain and the target domain individually. The input for the transformer encoder layers can be expressed as:

$$x_i = \text{encoder\_prenet}\left(\text{phoneme}_i\right) + \alpha\,\text{PE}(i) \tag{2.45}$$

The input for the decoder, constructed from mel-spectrogram, can be similarly defined. The authors found that such scalable positional encoding is necessary as the final scale for the decoder is indeed smaller than the scale for the encoder.

The encoder and decoder layers are the same as the original transformer architecture [40]. The self-attention modules allow efficient capturing of long-term dependencies in text and spectrograms, and multi-head attention could capture different perspectives between input and output. Transformer-TTS also uses the decoder post-net from Tacotron 2 to calculate a residual for fine-tuning the final mel-spectrogram. The MSE loss is similarly defined as in Tacotron 2. A positive weight is used for the stop token loss when calculating the binary cross-entropy, as there is only one positive stop token per utterance.

While the Transformer-TTS and Tacotron 2 achieve the same MOS score on their internal US English female dataset, there is still a preference for the output from the transformer model shown by the comparison mean option score test. The authors also found that their 6-layer Transformer-TTS re-

constructed the details better than Tacotron 2 and their three-layer variant, which blurred the texture in the high-frequency region.

2.3 Cycle Consistency for Speech

Cycle consistency relies on the following assumption: given a forward model M that converts from $\mathcal{X}$ to $\mathcal{Y}$ and another reverse model N that converts from $\mathcal{Y}$ to $\mathcal{X}$, the reconstruction $N(M(\mathbf{x})) \in \mathcal{X}$ should be close to the original input $\mathbf{x} \in \mathcal{X}$ (and $M(N(\mathbf{y})) \in \mathcal{Y}$ should be close to $\mathbf{y} \in \mathcal{Y}$). For speech, $\mathcal{X}$ is the domain for speech representations, such as mel-spectrograms, and $\mathcal{Y}$ is the domain for text, such as phones or characters. The model M is a speech recognizer while the model N is a speech synthesizer.

Three works will be reviewed here, the comprehensive machine speech chain model and two other variants. The first variant uses encoder states as the output for a speech synthesizer, and the other variant combines a speech encoder with CTC loss, a vector-quantized variational auto-encoder (VQVAE), and a speech decoder from Tacotron 2.

2.3.1 Machine Speech Chain

As introduced in Section 1.2, a speaker produces sound waves for the listener, who converts the auditory signals into information. The speaker also acts as their own listener. This concept for the human speech chain [9] can be adapted to speech recognition and speech synthesis, as shown in the proposal for the machine speech chain [10]. Figure 2.4, which is extracted from the proposal for the machine speech chain [10], explains the above concept as a "chain" between an ASR system and a TTS system.

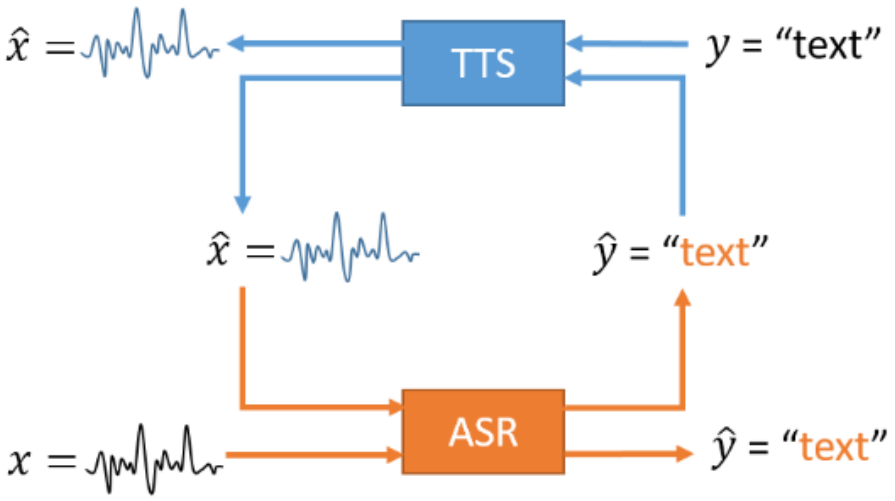

Figure 2.4: Machine speech chain architecture from the proposal for the machine speech chain [10].

The machine speech chain architecture consists of a seq2seq ASR system, a seq2seq TTS system, and a loop connection that feeds the output of ASR to TTS and the output of TTS to ASR. The entire system leverages three types of data:

1. paired speech and text for training ASR and TTS under their regular formulation;

2. unpaired speech, which serves as input to ASR, where the ASR text prediction is fed into TTS for reconstruction;

3. unpaired text, which serves as input to TTS, where the TTS output is fed into ASR for recognition.

Given a paired speech-text sample $\left(\mathbf{x}^P, \mathbf{y}^P\right) = \left(\left[x_1^P, \ldots, x_{S_P}^P\right], \left[y_1^P, \ldots, y_{T_P}^P\right]\right)$, a text probability and a speech frame prediction can be calculated by the ASR encoder-decoder network and the TTS encoder-decoder network (via teacher-forcing) as:

$$p_{y_t} = P\left(y_t \mid y_{<t}^P, \mathbf{x}^P; \theta_{ASR}\right), \forall t \in [1..T_P] \tag{2.46}$$

$$\hat{x}_s^P = \text{argmax}\, P\left(z \mid \mathbf{x}_{<s}^P, \mathbf{y}^P; \theta_{TTS}\right); \forall s \in [1..S_P] \tag{2.47}$$

upon which the paired ASR loss and TTS loss can be calculated as

$$\ell_{ASR}^{P} = \mathcal{L}_{ASR}\left(\mathbf{y}^{P}, \mathbf{p}_{y}; \theta_{ASR}\right) \tag{2.48}$$

$$\ell_{TTS}^{P} = \mathcal{L}_{TTS}\left(\mathbf{x}^{P}, \hat{\mathbf{x}}^{P}; \theta_{TTS}\right) \tag{2.49}$$

Given an unpaired text $\mathbf{y}^{U} = \left[y_1^U, \ldots, y_{T_U}^U\right]$, speech is generated by the TTS model, and the synthetic speech is used as input into the ASR system to calculate an ASR loss, with $\mathbf{y}^{U}$ as the target sequence. The procedure can be expressed as:

$$\hat{\mathbf{x}}^{U} \sim P_{TTS}\left(\cdot \mid \mathbf{y}^{U}; \theta_{TTS}\right) \tag{2.50}$$

$$p_{y_t} = P\left(y_t \mid \mathbf{y}_{<t}^{U}, \hat{\mathbf{x}}^{U}; \theta_{ASR}\right); \quad \forall t \in [1..T_U] \tag{2.51}$$

$$\ell_{ASR}^{U} = \mathcal{L}_{ASR}\left(\mathbf{y}^{U}, \mathbf{p}_{i}; \theta_{ASR}\right) \tag{2.52}$$

Given an unpaired speech $\mathbf{x}^{U} = \left[x_1^U, \ldots, x_S^U\right]$, text is generated by the ASR model, and the pseudo text is used as input into the TTS system to calculate a TTS loss, with $\mathbf{x}^{U}$ as the target. The procedure can be expressed as:

$$\hat{\mathbf{y}}^{U} \sim P_{ASR}\left(\cdot \mid \mathbf{x}^{U}; \theta_{ASR}\right) \tag{2.53}$$

$$\hat{x}_s^U = \underset{z}{\operatorname{argmax}} P_{TTS}\left(z \mid \mathbf{x}_{<s}^{U}, \hat{\mathbf{y}}^{U}; \theta_{TTS}\right); \quad \forall s \in [1..S] \tag{2.54}$$

$$\ell_{TTS}^{U} = \mathcal{L}_{TTS}\left(\mathbf{x}^{U}, \hat{\mathbf{x}}^{U}; \theta_{TTS}\right) \tag{2.55}$$

The final loss can be expressed as

$$\ell_{ALL} = \alpha * \left(\ell_{TTS}^{P} + \ell_{ASR}^{P}\right) + \beta * \left(\ell_{TTS}^{U} + \ell_{ASR}^{U}\right) \tag{2.56}$$

The gradient with respect to θ_{TTS} and θ_{ASR} are calculated and back-propagated through the respective systems.

The ASR encoder-decoder network consists of an encoder with fully connected layers followed by stacked bidirectional LSTMs, and a decoder that consists of a unidirectional LSTM with an attention mechanism. The ASR loss is calculated as

$$\mathcal{L}_{ASR}\left(\mathbf{y}, \mathbf{p}_{y}\right) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C}\mathbf{1}\left(y_t = c\right) * \log p_{y_t}[c] \tag{2.57}$$

where C is the number of classes. The TTS model uses an older Tacotron model architecture [51]. Different from Tacotron 2, Tacotron uses a fully-connected + 1D Convolution Bank + Highway + bidirectional-GRU (CBHG) structure for the encoder and a fully-connected + LSTM structure with an attention mechanism for the decoder. The decoder reconstructs mel-spectrograms first, and another CBHG structure reconstructs linear spectrograms from mel-spectrograms. The TTS loss is thus

$$\begin{aligned}\mathcal{L}_{TTS}(\mathbf{x}, \hat{\mathbf{x}}) =& \frac{1}{S} \sum_{s=1}^{S} \left\| x_s^M - \hat{x}_s^M \right\|_2^2 + \left\| x_s^R - \hat{x}_s^R \right\|_2^2 \\ & - \left(b_s \log \left(\hat{b}_s \right) + (1 - b_s) \log \left(1 - \hat{b}_s \right) \right) \end{aligned} \tag{2.58}$$

where $x_s^M, \hat{x}_s^M$ are in the mel scale and $x_s^R, \hat{x}_s^R$ are in the linear (regular) scale. The last term is the stop-token binary cross-entropy loss. During inference, the predicted linear spectrogram is fed into the well-known Griffin-Lim algorithm [52] to estimate the phase component and reconstruct the speech waveform.

Speaker embeddings extracted from speech utterances are used as additional conditioning for the TTS decoder in their multi-speaker experiments. The authors proposed a DeepSpeaker model, where the speech features first go through several convolutional layers, followed by pooling over time and L2 normalization. The final speaker embeddings z could be learned with a speaker classification loss where

$$p_y = \text{Softmax}\left(zW_z\right) \tag{2.59}$$

$$\ell_{NLL} = -\sum_{n=1}^{N} \mathbf{1}(y = n) * \log p_y[n] \tag{2.60}$$

or a triplet loss

$$\ell_{TRI} = \sum_{\substack{a,p,n \\ y_a = y_p \neq y_n}} \max\left(\|z_a - z_p\|_2^2 + \|z_a - z_n\|_2^2, 0 \right) \tag{2.61}$$

where a, p are from the same speaker and n is from a different speaker.

The TTS loss is also modified to incorporate an additional term that minimizes the cosine distance between the speaker embeddings z and $\hat{z}$, which are extracted from the ground-truth speech and the predicted speech, respec-

tively:

$$\begin{aligned}\ell_{TTS} =& \mathcal{L}_{TTS}(\mathbf{x}, \hat{\mathbf{x}}, z, \hat{z}) \\ =& \left(\sum_{s=1}^{S} \gamma_1 \left(\left\| x_s^M - \hat{x}_s^M \right\|_2^2 + \left\| x_s^R - \hat{x}_s^R \right\|_2^2 \right) \right. \\ & \left. -\gamma_2 \left(b_s \log \left(\hat{b}_s \right) + (1 - b_s) \log \left(1 - \hat{b}_s \right) \right) \right) \\ & + \gamma_3 \left(1 - \frac{\langle \hat{z}, z \rangle}{\|\hat{z}\|_2 \|z\|_2} \right) \end{aligned} \tag{2.62}$$

Note that for the machine speech chain model discussed earlier, the gradient is cut-off between the ASR module and the TTS module when unpaired data is involved. The authors also proposed an end-to-end feedback loss for improving an ASR system without paired data, with an additional TTS reconstruction loss $\ell_{TTS}^{rec} = \mathcal{L}_{TTS}^{rec}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{S} \sum_{s=1}^{S} \left\| x_s^M - \hat{x}_s^M \right\|_2^2$ added to ℓ_{ASR}. However, as the intermediate text generation for calculating ℓ_{TTS}^{rec} is discrete, back-propagating the term to θ_{ASR} may pose a problem. The authors proposed two strategies. The first strategy is the vanilla straight-through estimation, with the forward pass (starting from $p_{y_t}[c] = \frac{\exp\left(h_t^d[c]/\tau\right)}{\sum_{i=1}^{C} \exp\left(h_t^d[i]/\tau\right)}$, where h_t^d's are ASR decoder states) defined as:

$$\tilde{z}_t = \text{argmax}_c \, p_{y_t}[c] \tag{2.63}$$

$$\tilde{y}_t = \text{onehot} \left(\tilde{z}_t \right) \tag{2.64}$$

and the backward pass defined as

$$\frac{\partial \tilde{y}_t}{\partial p_{y_t}} = \mathbf{1} \tag{2.65}$$

The second strategy is called straight-through Gumbel-softmax. This strategy modifies $p_{y_t}[c]$ from a regular softmax into

$$p_{y_t}[c] = \frac{\exp\left(\left(h_t^d[c] + g_c\right)/\tau\right)}{\sum_{i=1}^{C} \exp\left(\left(h_t^d[i] + g_i\right)/\tau\right)}, \quad \forall c \in [1..C] \tag{2.66}$$

where τ is the temperature parameter and g_i are i.i.d. random variables

sampled with

$$u_i \sim \text{Uniform}(0, 1) \tag{2.67}$$

$$g_i = -\log\left(-\log\left(u_i\right)\right) \tag{2.68}$$

The strategy then continues the same forward and backward pass as the vanilla straight-through estimator.

2.3.2 From ASR-TTE to ASR-TTS + TTS-ASR

The ASR-TTE model [13] is an end-to-end cycle-consistent ASR system based on a seq2seq ASR encoder-decoder network with attention and a Tacotron 2 trained with ASR encoder outputs as targets instead of mel-spectrograms. The motivation for using ASR encoder outputs is that such hidden states are relatively speaker-independent, and the reconstruction from the Tacotron 2 would be less prone to mismatches caused by missing para-linguistic information.

Let $\mathbf{X}$ denote input speech features and $\mathbf{C}$ denote output characters, the ASR encoder-decoder goes through the following set of operations

$$\mathbf{h}_t^{\text{asr}} = \text{Encoder}^{\text{asr}}(\mathbf{X}), \tag{2.69}$$

$$a_{lt}^{\text{asr}} = \text{Attention}^{\text{asr}}\left(\mathbf{q}_{l-1}^{\text{asr}}, \mathbf{h}_t^{\text{asr}}, \mathbf{a}_{l-1}^{\text{asr}}\right), \tag{2.70}$$

$$\mathbf{r}_l^{\text{asr}} = \Sigma_{t=1}^{T} a_{lt}^{\text{asr}} \mathbf{h}_t^{\text{asr}}, \tag{2.71}$$

$$\mathbf{q}_l^{\text{asr}} = \text{Decoder}^{\text{asr}}\left(\mathbf{r}_l^{\text{asr}}, \mathbf{q}_{l-1}^{\text{asr}}, c_{l-1}\right), \tag{2.72}$$

$$p_{\text{asr}}\ \left(c_l \mid c_{1:l-1}, \mathbf{X}\right) = \text{Softmax}\left(\text{LinB}\left(\mathbf{q}_l^{\text{asr}}\right)\right), \tag{2.73}$$

with the objective trained under teacher-forcing as

$$\mathcal{L}_{\text{asr}} = -\log p_{\text{asr}}(\mathbf{C} \mid \mathbf{X}) \tag{2.74}$$

$$= -\Sigma_{l=1}^{L} \log p_{\text{asr}}\left(c_l^{\text{asr}} \mid c_{1:l-1}^{\text{asr}}, \mathbf{X}\right) \tag{2.75}$$

The TTE module takes input characters $\mathbf{C}$ and reconstructs $\mathbf{h}_t^{\text{asr}}$, as follows

$$\mathbf{h}_l^{\text{tte}} = \text{Encoder}^{\text{tte}}(\mathbf{C}) \tag{2.76}$$

$$a_{tl}^{\text{tte}} = \text{Attention}^{\text{tte}}\left(\mathbf{q}_{t-1}^{\text{tte}}, \mathbf{h}_l^{\text{tte}}, \mathbf{a}_{t-1}^{\text{tte}}\right) \tag{2.77}$$

$$\mathbf{r}_t^{\text{tte}} = \Sigma_{l=1}^{L} a_{tl}^{\text{tte}} \mathbf{h}_l^{\text{tte}} \tag{2.78}$$

$$\mathbf{v}_{t-1} = \text{Prenet}\left(\mathbf{h}_{t-1}^{\text{asr}}\right) \tag{2.79}$$

$$\mathbf{q}_t^{\text{tte}} = \text{Decoder}^{\text{tte}}\left(\mathbf{r}_t^{\text{tte}}, \mathbf{q}_{t-1}^{\text{tte}}, \mathbf{v}_{t-1}\right) \tag{2.80}$$

$$\hat{\mathbf{h}}_t^{b,\,\text{asr}} = \tanh\left(\text{LinB}\left(\mathbf{q}_t^{\text{tte}}\right)\right) \tag{2.81}$$

$$\mathbf{d}_t = \text{Postnet}\left(\mathbf{q}_l^{\text{te}}\right) \tag{2.82}$$

$$\hat{\mathbf{h}}_t^{a,\,\text{asr}} = \tanh\left(\text{LinB}\left(\mathbf{q}_l^{\text{tte}}\right) + \mathbf{d}_t\right) \tag{2.83}$$

$$\hat{s}_t = \text{Sigmoid}\left(\text{LinB}\left(\mathbf{q}_t^{\text{tte}}\right)\right) \tag{2.84}$$

where $\hat{s}_t$ is for the stop-token. The TTE module is trained under a similar objective as regular TTS:

$$\begin{aligned}\mathcal{L}_{\text{tte}} &= \text{MSE}(\hat{\mathbf{h}}_t^{a,\,\text{asr}}, \mathbf{h}_t^{\text{asr}}) + \text{MSE}\left(\hat{\mathbf{h}}_t^{b,\,\text{asr}}, \mathbf{h}_t^{\text{asr}}\right) \\ &+ \text{L1}\left(\hat{\mathbf{h}}_t^{a,\,\text{asr}}, \mathbf{h}_t^{\text{asr}}\right) + \text{L1}\left(\hat{\mathbf{h}}_t^{b,\,\text{asr}}, \mathbf{h}_t^{\text{asr}}\right) \\ &+ \frac{1}{T}\Sigma_{t=1}^{T}\left(s_t \ln \hat{s}_t + (1 - s_t)\ln\left(1 - \hat{s}_t\right)\right)\end{aligned} \tag{2.85}$$

For end-to-end feedback training, the TTE module needs to take

$$\hat{\mathbf{C}} = \underset{\mathbf{C}\in\mathcal{U}^+}{\text{argmax}} \log p_{\text{asr}}\left(\mathbf{C} \mid \mathbf{X}\right)$$

as input. However, the argmax operator is non-differentiable. They introduced the expected loss for $\mathcal{L}_{\text{tte}}$, defined as

$$\mathcal{L}_{\text{ette}} = \mathbb{E}_{\hat{\mathbf{C}}|\mathbf{X}}\left[\mathcal{L}_{\text{tte}}\left(\hat{\mathbf{H}}^{\text{asr}}\left(\hat{\mathbf{C}}\right), \mathbf{H}^{\text{asr}}\left(\mathbf{X}\right)\right)\right] \tag{2.86}$$

where $\hat{\mathbf{H}}^{\text{asr}}\left(\hat{\mathbf{C}}\right)$ denotes the TTE model predictions given $\hat{\mathbf{C}}$, and $\mathbf{H}^{\text{asr}}\left(\mathbf{X}\right)$ denotes the ASR encoder state targets produced by input speech $\mathbf{X}$. The gradient of $\mathcal{L}_{\text{ette}}$ with respect to the ASR model parameters is then estimated over N samples of $\mathbf{C}^n$ from the ASR model, with the REINFORCE algorithm

as

$$\nabla \mathcal{L}_{\text{ette}} \approx \frac{1}{N} \sum_{\substack{\mathbf{C}^n \sim p_{\text{asr}}(\cdot|\mathbf{X}), \\ n=1,\ldots,N}} \mathrm{T}\left(\mathbf{C}^n, \mathbf{X}\right) \nabla \log p_{\text{asr}}\left(\mathbf{C}^n \mid \mathbf{X}\right) \tag{2.87}$$

where

$$\mathrm{T}\left(\mathbf{C}^n, \mathbf{X}\right) = \mathcal{L}_{\text{tte}}\left(\hat{\mathbf{H}}^{\text{asr}}\left(\mathbf{C}^n\right), \mathbf{H}^{\text{asr}}(\mathbf{X})\right) - B\left(\mathbf{X}, \mathbf{C}^n\right) \tag{2.88}$$

and $B\left(\mathbf{X}, \mathbf{C}^n\right)$ is a baseline for reducing variance, which is simply chosen as the mean value of $\mathbf{H}^{\text{asr}}\left(\mathbf{C}^n\right)$ over N samples.

A later semi-supervised model derived from ASR-TTE proposed to combine an end-to-end differentiable ASR→TTS loss and another non-end-to-end TTS→ASR loss [14]. The first loss leverages unpaired speech, while the second loss leverages unpaired text. For calculating ASR→TTS loss from unpaired speech, the model proposes to additionally condition Tacotron 2 on speaker vectors obtained from an x-vector network [53]. The motivation is that the earlier TTE model could weaken the consistency constraint. The ASR→TTS loss is still back-propagated to ASR model parameters via the REINFORCE algorithm. To leverage unpaired text, the TTS→ASR is calculated by first feeding the text sequence + a randomly sampled x-vector into the TTS module under inference mode, and then feeding the generated $\hat{\mathbf{X}}$ into the ASR module to calculate the ASR loss with the unpaired text sequence as the target. Note that there is no need to backpropagate the gradient into the TTS module. The above processes is shown in Figure 2.5, which is directly extracted from the proposal for cycle-consistency training with unpaired speech and text [14].

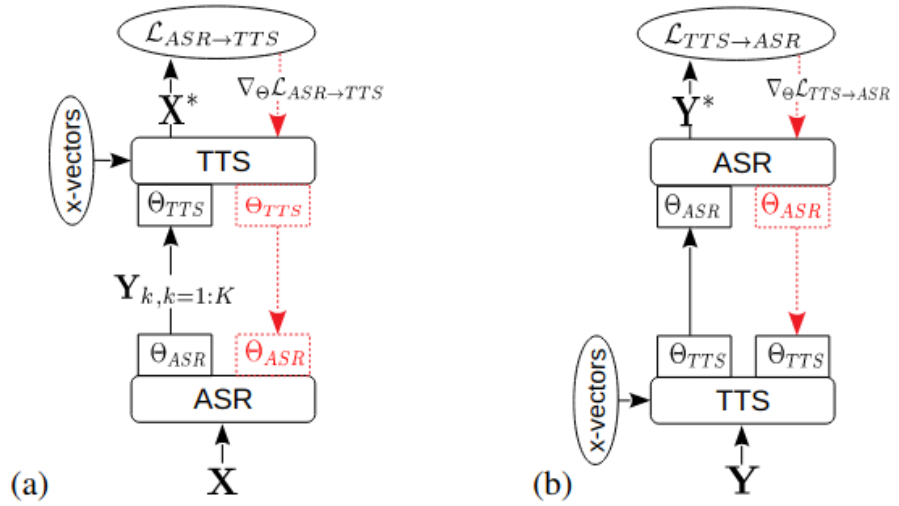

Figure 2.5: Cycle-consistency training with unpaired speech and text [14].

2.3.3 Sequential Representation Quantization Auto Encoder

A Sequential Representation Quantization AutoEncoder (SeqRQ-AE) [54] shares the intermediate representation between an ASR encoder and TTS decoder with a VQVAE. The overall system is shown in Figure 2.6, which is extracted directly from the original proposal for SeqRQ-AE [54].

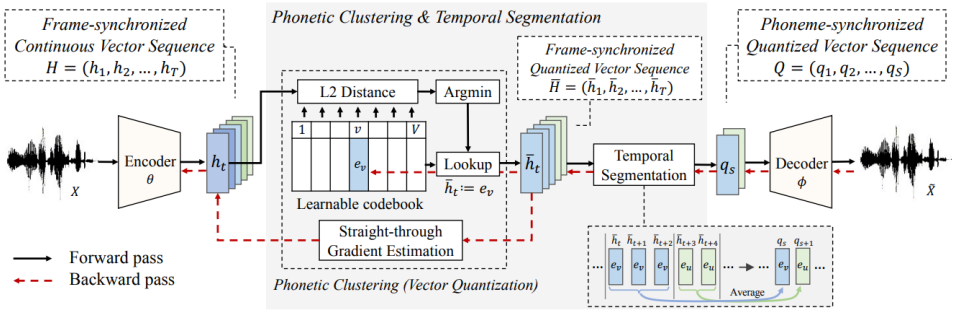

Figure 2.6: The system architecture for SeqRQ-AE [54].

The frame-wise speech features $X = (x_1, x_2, \ldots, x_T)$ first go through an ASR encoder with parameters θ to produce frame-synchronized hidden states

$H = \text{Encoder}_\theta(X) = (h_1, h_2, \ldots, h_T)$. With a learnable codebook $E = \{e_1, e_2, \ldots, e_V\}$ of V different embeddings, the frame-synchronized quantization $\bar{h}_t$ can be calculated for each h_t as

$$\bar{h}_t = h_t + e_v - \text{sg}(h_t), \qquad \text{where } v = \arg\min_k \|h_t - e_k\|_2 \tag{2.89}$$

where $sg(\cdot)$ is the stop-gradient operator. The above expression allows straight-through gradient back-propagation from $\bar{h}_t$ to h_t, as shown by the dashed red line. The temporal segmentation module takes $\bar{H} = \left(\bar{h}_1, \bar{h}_2, \ldots, \bar{h}_T\right)$, groups consecutive $\bar{h}_t$ with the same value together, and averages over each group to produce a phoneme-synchronized quantization sequence $Q = (q_1, q_2, \ldots, q_S)$. The TTS decoder with parameters ϕ takes Q as input and reconstructs frame-level speech features as $\tilde{X} = \text{Decoder}_\phi(Q)$.

The model is trained with both unpaired speech X and a limited amount of paired speech and text (X_{pair}, Y_{pair}). The codebook size V is set to the number of phonemes plus an additional blank token so that the CTC loss can be calculated over the paired data. The frame-wise phoneme posteriors are defined as:

$$P(v \mid h_t) = \frac{\exp\left(-\|h_t - e_v\|_2\right)}{\sum_{k \in V} \exp\left(-\|h_t - e_k\|_2\right)} \tag{2.90}$$

The final loss function on both paired data and unpaired speech is thus

$$\begin{aligned} L_{\text{total}} &= \text{MSE}(\tilde{X}, X) \\ &\quad - \lambda_1 \log P\left(Y_{\text{pair}} \mid H\right) \\ &\quad + \lambda_2 \,\text{MSE}\left(\text{Dec}_\phi\left(Q_{\text{pair}}\right), X_{\text{pair}}\right) \end{aligned} \tag{2.91}$$

The experiments were performed on a single-speaker corpus, LJSpeech [55]. The model achieves competitive phoneme error rates for as little as 5-20 minutes of paired speech. With 20 minutes of paired data, synthetic speech from the TTS module also outperformed the machine speech chain [10].

2.4 Invariant Risk Minimization

This section reviews the original proposal for invariant risk minimization [31], and a stronger version for enforcing invariance called regret minimization [32]. We will use the following notations throughout the two sections.

First, the random variable X is used to represent input data, and each sample is denoted as $x \sim X$. The random variable Y is used to represent output data, and each sample is denoted as $y \sim Y$. We use $\mathcal{E}$ to denote the set of environments in the training data. We use $f : X \to Y$ to denote the predictor $Y \approx f(X)$ that we aim to learn. The predictor f can be viewed as a composition of two parts $f := w \circ \phi$, where $\phi : X \to Z$ is the feature extractor that maps input features X into a latent representation space Z, and $w : Z \to Y$ is a classifier that maps the latent representation Z to the output Y. We use $\mathcal{R}$ to denote the empirical risk over the entire dataset and use $\mathcal{R}^e$ to denote the empirical risk over the subset of data from environment e.

Risk is defined to be the expected value of loss [56]. The loss we incur when the the paired sample (x_i, y_i) is predicted as $f(x_i)$ is measurable using a loss function $\mathcal{L}(f(x_i), y_i)$. Risk is therefore computed as the average over all $(x, y) \sim (X, Y)$:

$$\mathcal{R}(f(X), Y) = \mathbb{E}[\mathcal{L}(f(X), Y)]. \tag{2.92}$$

Empirical risk minimization (ERM) minimizes the average loss on the training set, with the goal of achieving high accuracy on an independent and identically distributed test set. ERM is formulated as follows:

$$f_{ERM} = \arg\min_{f} \mathcal{R}(f(X), Y) \tag{2.93}$$

In the limit of infinite training data, ERM provably minimizes the expected risk on the test corpus, provided that the test corpus and training corpus are drawn from the same distribution [57]. In many practical settings, however, the test corpus and training corpus are not drawn from the same distribution.

For the application of speech, input samples are sequences of raw acous-

tic feature vectors and output samples are sequences of IPA symbols. Each language is viewed as an environment, which determines the distributions of acoustic features and phonemes. $f : X \to Y$ is then a speech recognition model that takes acoustic features as input and transcribes them into phoneme transcriptions. Available ASR training corpora are heavily biased in favor of a few well-resourced languages. During testing, the mixture of languages may be quite different: some languages that were badly underrepresented during training may be somewhat more frequent during testing. We can characterize the problems with ERM by separately measuring the risk for each language, $e \in \mathcal{E}$, as

$$\mathcal{R}^e(f(X), Y) = \mathbb{E}_e[\mathcal{L}(f(X), Y)], \tag{2.94}$$

where $\mathbb{E}_e[\cdot]$ denotes expectation over data drawn from environment e. A number of different approaches have been developed to address the problem of ERM solutions with unacceptably high risk for one or more particular environments.

2.4.1 Invariant Risk Minimization

Invariant risk minimization [31] seeks to find a better balance among the many different languages in the training corpus, by computing an invariant embedding $Z = \phi(X)$ such that the optimal speech recognizer, $Y = w(Z)$, is the same in all languages.

Invariant risk minimization finds an environment-dependent classifier $f = w \circ \phi$ that is the composition of a feature extractor, $\phi : X \to Z$, and a classifier, $w : Z \to Y$. The feature extractor is judged to achieve invariant risk if the minimum-risk classifier sets for all of the environments, $\arg\min \mathcal{R}^e(w)$, overlap by at least one element: there is at least one classifier that is simultaneously optimal in all environments. Invariant risk minimization finds a (w, ϕ) that minimize the overall risk, subject to the constraint that ϕ achieves invariant risk:

$$f_{IRM} = \arg\min_{w,\phi} \sum_{e \in \mathcal{E}} \mathcal{R}^e(w \circ \phi(X), Y),$$
$$\text{s.t.} \qquad w \in \arg\min_{\bar{w}} \mathcal{R}^e(\bar{w} \circ \phi(X), Y) \quad \forall e \in \mathcal{E}. \tag{2.95}$$

Equation (2.95) defines IRM, but is difficult to implement. The constrained optimization in Equation (2.95) requires that, in order to update the feature extractor, one must determine the update's effect on the set of optimal classifiers in every environment. A more practical version of IRM [31] states that finding $w \in \arg\min \mathcal{R}^e$ is equivalent to minimizing the L2-norm of the gradient, $\|\nabla_w \mathcal{R}^e\|^2$, for every environment, which can be performed using a multi-task learning framework with a weighting coefficient of λ:

$$f_{IRM} = \arg\min_{w,\phi} \sum_{e \in \mathcal{E}} \mathcal{R}^e(w(\phi(X)), Y) + \lambda \|\nabla_w \mathcal{R}^e(w(\phi(X)), Y)\|_2^2. \tag{2.96}$$

The division of f into two subsystems, ϕ and w, is somewhat arbitrary; in an end-to-end neural network, any particular layer could be arbitrarily chosen to be trained as the invariant embedding. The subsequent formulation of IRM [31] takes inspiration from the observation that, when the loss function is either mean squared error or cross entropy, the optimal classifier is the conditional expectation of Y given $\phi(X)$ [58]. In this case, the feature extractor ϕ is optimal across environments if and only if we have:

$$\mathbb{E}_{e_i}[Y|\phi(X) = z] = \mathbb{E}_{e_j}[Y|\phi(X) = z] \ \forall e_i, e_j \in \mathcal{E}, \tag{2.97}$$

The final form of IRM (IRMv1) [31] observes that Equation (2.97) is most simply satisfied if $\phi(x) = z = y$. In order to guarantee that $\phi(x) = y$ satisfies the condition in Equation (2.96), they propose fixing $w(z) = w \cdot z$, and fixing the coefficient to $w = 1.0$, thus:

$$f_{IRM} = \arg\min_{\phi} \sum_{e \in \mathcal{E}} \mathcal{R}^e(\phi(X), Y) + \lambda \|\nabla_{w:w=1.0} \mathcal{R}^e(w \cdot \phi(X)), Y)\|_2^2. \tag{2.98}$$

2.4.2 Regret Minimization

The framework of regret minimization was originally proposed in economics, in order to explain the tendency of human actors to consistently make choices that lead to suboptimal expected rewards [59]. The framework of regret minimization proposes that rational actors have reason to doubt their own estimates of the probabilities of future events. One way to compensate for lack of knowledge is by minimizing expected regret, where regret is an increasing convex function of foregone income, such that potential events that lead to a great deal of foregone income are overweighted relative to their estimated probability. Recently, regret minimization was applied to the task of domain adaptation in machine learning [32]. They proposed that the distribution of environments in a test corpus is often badly matched to the distribution of environments in a training corpus, and that it is therefore rational to learn a classifier that minimizes the regret incurred by training on the wrong subset of environments.

Denote $\mathcal{R}^e(w \circ \phi)$ as the risk computed over environment e, and $\mathcal{R}^{-e}(w \circ \phi)$ as the risk computed over all environments other than environment e, *i.e.*,

$$\mathcal{R}^{-e}(w \circ \phi) = \mathbb{E}_{e' \neq e}[\mathcal{L}^{e'}(w \circ \phi)] \tag{2.99}$$

Further, define w^e and w^{-e} as the minimizers of the corresponding risks

$$w^e = \arg\min_h \mathcal{R}^e(h \circ \phi), \quad w^{-e} = \arg\min_h \mathcal{R}^{-e}(h \circ \phi) \tag{2.100}$$

The regret minimization criterion [32] is then

$$f_{RGM} = \min_{w,\phi} \mathcal{R}(w \circ \phi) + \lambda \sum_e [\mathcal{R}^e(w^{-e} \circ \phi) - \mathcal{R}^e(w^e \circ \phi)] \tag{2.101}$$

The first term in Equation (2.101) is the empirical risk averaged over all environments. The second term measures the sum, across all environments, of the regret, $R_e(\phi)$, that would be incurred by training and testing on different environments:

$$R_e(\phi) = \mathcal{R}^e(w^{-e} \circ \phi) - \mathcal{R}^e(w^e \circ \phi) \tag{2.102}$$

Since w^{-e} and w^e are minimizers, $R_e(\phi)$ is a function of ϕ. Since w^e is

the minimizer of $\mathcal{R}^e(w \circ \phi)$, $R_e(\phi)$ is guaranteed to be non-negative. The minimizer of $R_e(\phi)$, therefore, is a feature extractor that eliminates all information about the environment, in the sense that the cross-environment classifier, w^{-e}, performs exactly as well as the optimum environment-dependent classifier, w^e.

IRM (Section 2.4.1) requires that the globally optimum classifier, w, must also be a minimizer of the environment-dependent risk for every particular environment: $\mathcal{R}^e(w \circ \phi) = \mathcal{R}^e(w^e \circ \phi)$. If the constrained optimization of Equation (2.95) is solved using a Lagrangian optimization technique, the Lagrangian form is

$$f_{IRM} = \min_{w,\phi} \mathcal{R}(w \circ \phi) + \lambda \sum_e [\mathcal{R}^e(w \circ \phi) - \mathcal{R}^e(w^e \circ \phi)] \tag{2.103}$$

The similarities and differences between IRM and RGM may be understood by comparing Equation (2.101) and Equation (2.103). Like IRM, regret minimization uses a Lagrangian constraint term to enforce invariance. Unlike IRM, the classifier w^{-e} is trained without access to samples in environment e, so that RGM in theory enforces a stronger invariance constraint on the feature extractor ϕ than IRM: in the terminology of a recent transformer-based multi-lingual speech recognition system [45], IRM minimizes the difference between multi-lingual and mono-lingual error rates, while RGM minimizes the difference between cross-lingual and mono-lingual error rates.

The procedure for regret minimization is schematized in Figure 2.7. As shown, even with only two distinct training environments (X^1 and X^2), five distinct classifiers must be trained (the globally optimum classifier w, the environment-dependent classifiers w^1 and w^2, and the cross-environment classifiers w^{-1} and w^{-2}).

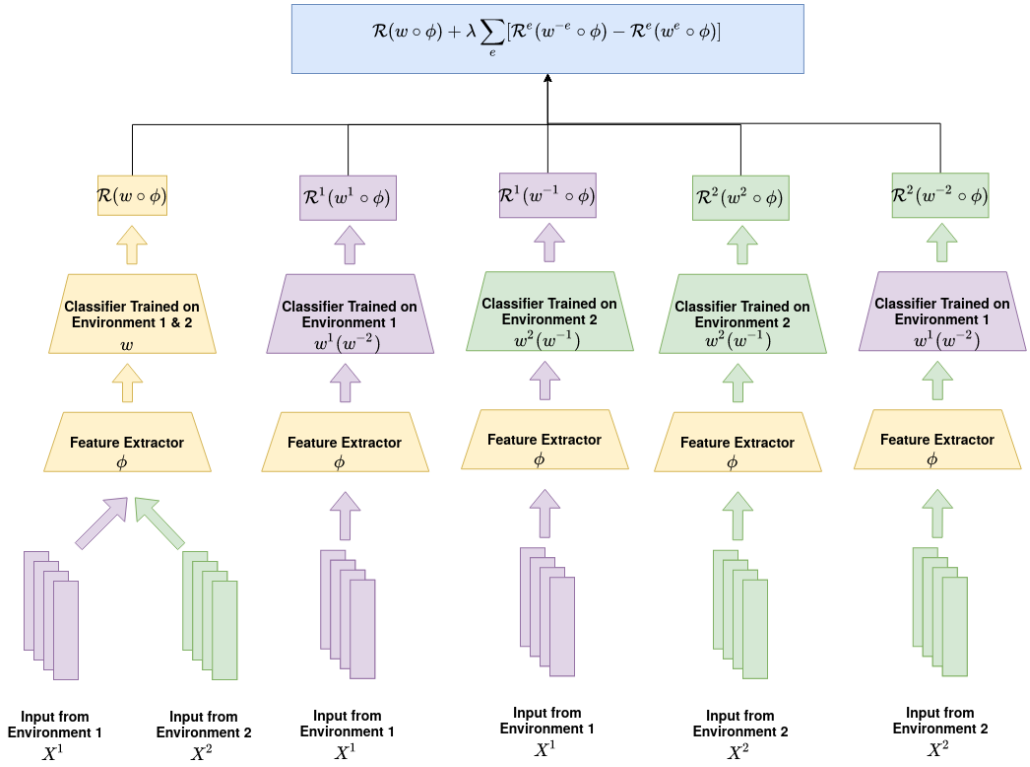

Figure 2.7: How to compute the risk for regret minimization in a two-environment setting. In addition to the ERM risk calculated on inputs from both environment under the shared feature extractor and classifier, regret minimization inserts an additional regret term for each environment into the total risk. The regret for environment 1, for example, can be calculated by first feeding the inputs from environment 1 into the feature extractor, and then into the classifier trained on environment 1, as well as the classifier trained on environment 2. The difference calculated from the corresponding loss term for the out-of-environment classifier and the within-environment classifier is the regret. The same calculation can be done to calculate the regret for environment 2.

CHAPTER 3

ANALYZING THE EFFECT OF LANGUAGE-SPECIFIC TUNING FOR A BI-LINGUAL CTC SYSTEM

In this chapter, we present our experiment set-up for a bi-lingual CTC system in 3.1. We will discuss the shared and non-shared phone inventories of our English and Dutch corpora, data and model settings, as well as training methods. In 3.2, we will then describe the three experiments designed for visualizing and analyzing language-specific tuning. We focus on understanding the differences of consonant and vowel spaces of the two languages, the differences in the shared or non-shared layers in the two models, and the differences between the two models. For each experiment, we proposed some hypotheses following our intuition about L2 acquisition and model architecture. We finally prove or disprove the hypotheses for each experiment in 3.3 with visualizations and classification tests.

3.1 Experimental Set-up

We used Dutch and English as the pair of languages to investigate the effect of language-specific tuning, because these languages share many consonants but few vowels. In the two corpora we used (see below), only four of the 23 consonants in Dutch (/ʋ, x, ɲ, ɤ/) do not exist in English, and only five of the 24 consonants in English do not exist in Dutch. For vowels, 11 of 19 the Dutch vowels (ɔu, eː, iː, oː, œɪ, yː, ʉ, ɒː, ɛɪ, øː, aː) do not exist in the English corpus, while 10 of the 17 English vowels (/aʊ, eɪ, oʊ, ʌ, æ, ʊ, ɝ, aɪ, ɔɪ, ɚ/) do not exist in the Dutch corpus. Therefore, we expected language-specific tuning to behave differently when processing the consonants and the vowels of the two languages.

For Dutch, we used 64 hours of read speech from the Spoken Dutch Corpus (CGN; [60]), of which 80% was used for training, 10% for validation, and 10% for the test set. For English, we used The Flickr Audio Caption Corpus [61],

which consists of 40,000 spoken captions of 8,000 natural images from the Flickr8k image-caption dataset. We selected 30 hours of English utterances by discarding those that are too long/too short, as well as those with too high/too low forced alignment score. We first trained a CTC phone recognizer on Dutch only (L1 training stage), then used the obtained weights to initialize the single pathway model and the dual pathway model. The two models were then adapted on both English and Dutch (joint training stage). The architecture is a modified Deep Speech 2 [36]. The model starts with two layers of 2D convolution on the raw input spectrogram. The output from the convolutional layers is fed into six layers of bidirectional LSTM, each with 200 hidden units. The model was optimized under phone CTC [15] loss, and converged after 12 epochs of training.

In the joint training stage, the single pathway model was first initialized as the baseline CTC phone recognizer. A new softmax for English phones is created and appended to the last bLSTM layer. The dual pathway model was created by splitting the two final bLSTM layers (one for each language) and the final softmax (one for each language). New softmax units for the shared phones in English were initialized as their counterparts in Dutch, while softmax units for unseen English phones were initialized as linear combinations of several Dutch phones [62]. For the dual pathway model, the earlier shared layers were initialized with weights from corresponding layers in the baseline CTC recognizer. Both branches of bLSTM layers (for English and Dutch) were initialized by copying the weights of the corresponding layers in the baseline CTC recognizer as well. The two models have been illustrated in Figure 1.1. Both the single pathway and dual pathway models were then adapted using utterances from Flickr8k interleaved with those from the read speech section of CGN, using a multi-task phone CTC loss that simply sums up the individual loss from the two languages. Convergence was achieved in 12 epochs for each model.

3.2 Visualization and Analysis Methods

Three experiments were conducted to investigate language-specific tuning for the (largely shared between languages) consonants and the (largely unshared) vowels of the two languages. The first experiment investigated consonant rep-

resentations learned by the dual pathway model. Theories of human second language learning (PAM-L2 [63]; SLM [64]) suggest that, when L1 and L2 phonemes have similar articulatory or acoustic correlates, the learner may treat them as identical. Since most consonants in English and Dutch share the same IPA symbols, we hypothesize that the hidden layer activations of those consonants in the two languages may be the same, and that therefore, consonants may be well represented at the language-independent shared layer of the dual pathway model (i.e., the fourth bLSTM layer, shaded in orange in Figure 1.1). To that end, we jointly visualized the consonant representations in the two languages at the last shared layer. IPA symbols for each consonant are visualized, as are their manners of articulation, since previous research suggests that DNNs learn manner of articulation more easily than place [65]. We propose two hypotheses regarding experiment 1:

Hypothesis 1 (H1): *Consonants in Dutch and English with the same IPA symbols share similar hidden layer activations in the shared layer (analogy to the shared phonological space of human L2 learners [63]) and display a clear grouping in terms of manner of articulation (as suggested by previous experiments with DNNs), both within and between languages.*

Hypothesis 2 (H2): *Due to the large overlap between Dutch and English consonants, a model can learn the consonants in both L1 and L2 very well, even without the help of language-specific tuning.*

In the second experiment, we analyzed whether the dual pathway model has benefited from separate language-specific tuning at the later layers, by comparing the learned representations at sixth LSTM layers of the dual pathway model with those from the single pathway model. We focus on Dutch because joint training on English and Dutch leads to a strong deterioration on the recognition of Dutch, compared to the Dutch-only baseline, for the single pathway model but not the dual pathway model. While L2 interference on L1 is also observed for humans [64], the effect is usually minor and mostly happens on the production level. Thus, we wish to see why the dual pathway model better approximates human speech processing mechanisms than the single pathway model in this regard. We also focused on vowels instead of consonants in this experiment due to the large proportion of non-shared vowels between the two languages. Experiment 2 addresses the following hypothesis:

Hypothesis 3 (H3): *The L1 vowel space is poorly matched to the L2 without language-specific tuning. Compared to representations learned by the single pathway model, vowel representations of the dual pathway model are greatly improved.*

The third experiment further analyzed what the dual pathway model learns at the shared and non-shared levels by comparing the learned representations at the last shared layer (orange layer in Figure 1.1) with those at the last non-shared layer (i.e., after separate pathways for language-specific tuning; gray layers) separately for the two languages. Following our hypothesis for the second experiment, we believe that it will again be the vowel space that benefits from such further tuning, while following our hypotheses for the first experiment, we expect to see only marginal benefits for consonants. We calculated the average classification error rates (for consonants and vowels separately) using the activation frames extracted from the fourth and sixth layer of the dual pathway model (see dashed ovals in Figure 1.1). For comparison, we also included the classification error rates at the same layers for the single pathway model (see dashed ovals in Figure 1.1). We proposed the following hypothesis regarding experiment 3:

Hypothesis 4 (H4): *Activations from the sixth layer of the dual pathway model provide a better representation for the classifier, for English and Dutch vowels, than the fourth layer. However, no significant improvement should be observed by going deeper in the single pathway model.*

Hypothesis 5 (H5): *Consonants do not depend on language-dependent tuning (sixth layer error rates are as high as fourth layer), and show much lower error rates for both Dutch and English compared to vowels, due to the large overlap between Dutch and English consonant categories.*

All three experiments extract LSTM output activations from the corresponding bLSTM layer of the corresponding model. Phone alignments of the hidden activation frames for Dutch are obtained using a Kaldi [66] CGN triphone model with a recipe found online[1]. English alignments were kindly provided by the authors of a recent proposal that builds a low-resource language ASR from a high-resource language ASR [62]. For experiment 1, 10%

[1]https://github.com/laurensw75/kaldi_egs_CGN

of activation frames for each phone with the highest corresponding softmax probabilities are selected, jointly standardized, reduced to a dimension of 50 using PCA, and then jointly visualized using t-sne [67]. Experiment 2 is similar, but uses activations from a single language (Dutch) extracted from the single pathway and the dual pathway models. For experiment 3, we trained several MLP phone classifiers using the extracted activations at the fourth and sixth layers from both the single and dual pathway models as input feature vectors, separately for the two languages. The MLP classifiers all consist of one hidden layer of dimension 1024, followed by a softmax layer for phone outputs.

3.3 Visualization and Analysis Results

L1 training achieves 12.1% PER on Dutch. After Dutch-English joint training, the single pathway model obtains Dutch PER of 19.9%, which is 7.8% PER higher than the monolingual baseline, and English PER of 15.5%. The dual pathway model suffers less degradation than the single-pathway model: Dutch PER is 12.9% (only 0.8% higher than monolingual baseline), and English PER is also 12.9%.

Experiment 1 visualizes the space of consonants at the last shared bLSTM layer of the dual pathway model. Figures 3.1 and 3.2 show the hidden space of all consonants, where each symbol in the figure corresponds to an IPA symbol, and each color corresponds to a manner of articulation. Hypothesis 1 is confirmed: except for a few cases, the English consonants, after learning, get assimilated to almost the same location as their counterparts in Dutch. Groups of consonants with the same manner of articulation across the two languages also reside in roughly the same region of space. Hypothesis 2 is also confirmed: new consonants in the L2 language (English) learn representations close to phonetically similar neighbors. In Figures 3.1 and 3.2, English affricates not present in Dutch (/ʤ/, /ʧ/ near [-40, 0] in Figure 3.2) are placed between a group of plosives and a group of fricatives, crowding the phonetic space [64] by leveraging language-invariant articulatory gestures. Similarly, /θ/ (near [-40, -20] in Figure 3.2) and /ð/ (near [0, 0] in Figure 3.2) lie close to /t/ and /d/, with which they share a similar place of articulation.

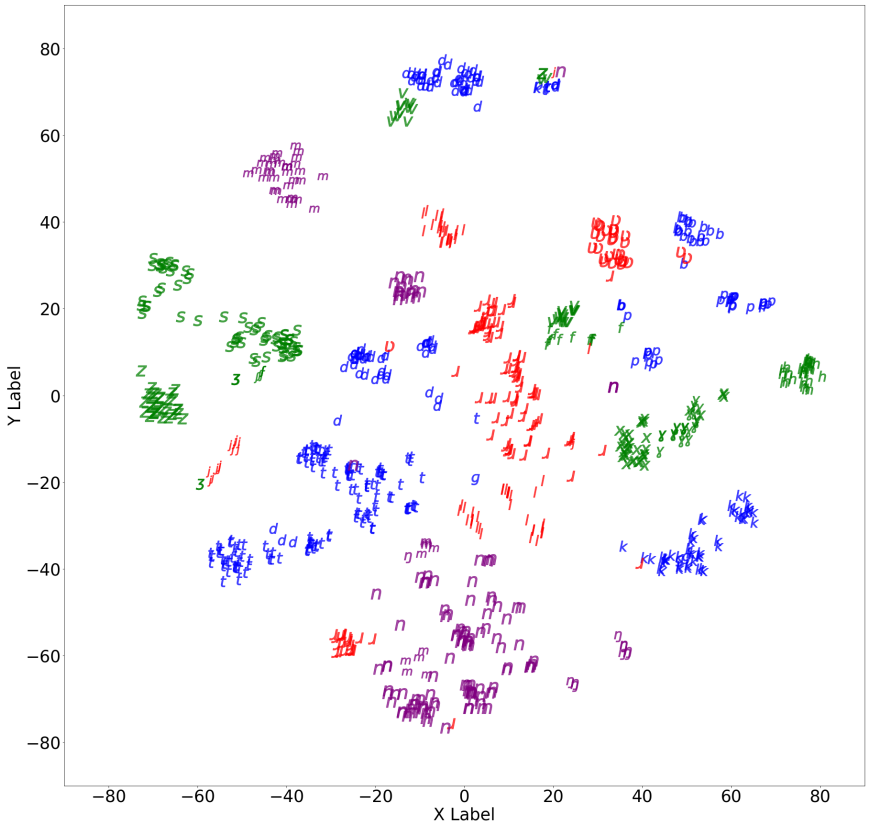

Figure 3.1: Dutch consonants after joint training; dual pathway model, fourth bLSTM.

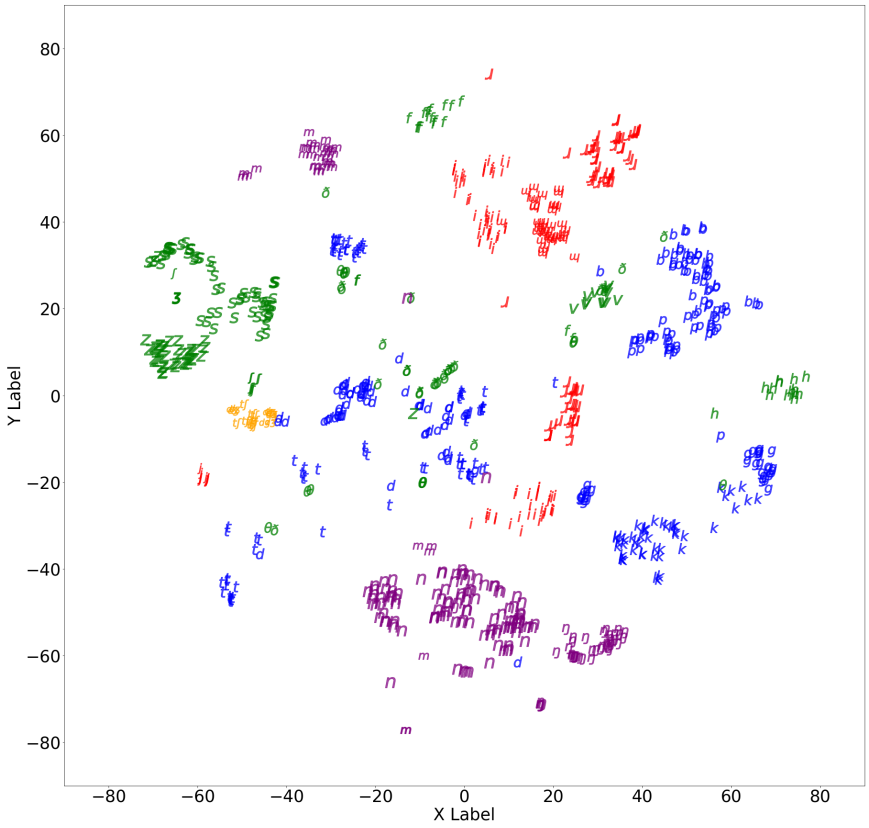

Figure 3.2: English consonants after joint training; dual pathway model, fourth bLSTM.

Experiment 2 investigates the vowel space in the penultimate activations from the dual pathway model for Dutch vowels (Figures 3.3 and 3.4: Figure 3.3 is for the single pathway model and Figure 3.4 is for the dual pathway model). Dutch is visualized because Dutch vowels suffered a larger number of errors than English vowels after joint training. Each color represents a distinct tongue height (low, low-mid, mid, high-mid or high). Hypothesis 3 is confirmed: activations in Figure 3.2 but not Figure 3.3 are crowded together, especially for the region of space that contains /eː,ɛɪ,øː/. The dual pathway model gives better clustering of individual vowels and of vowel height categories.

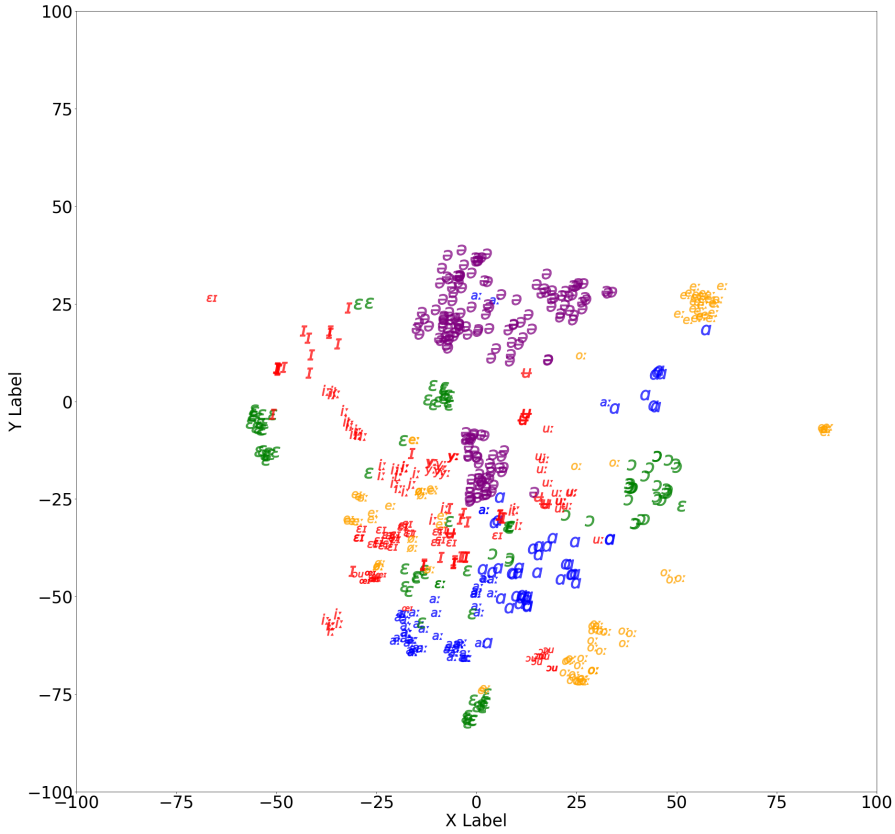

Figure 3.3: Dutch vowels; single pathway model, sixth bLSTM.

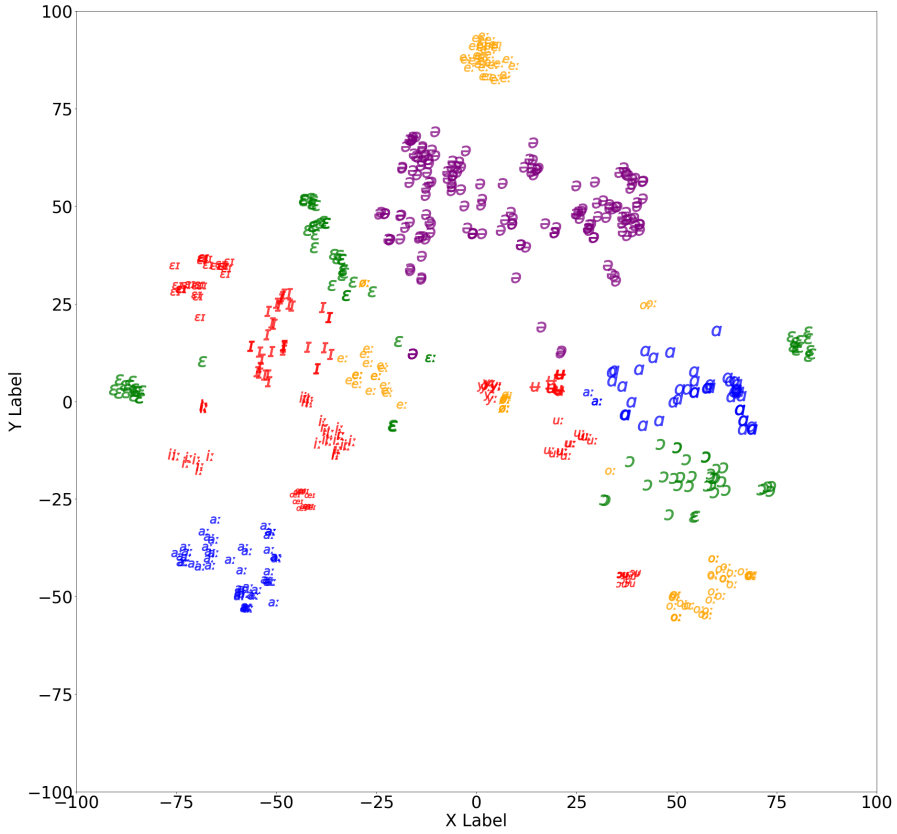

Figure 3.4: Dutch vowels; dual pathway model, sixth bLSTM.

Experiment 3 further investigates language-specific tuning in the dual pathway model by training two separate classifiers that use the activations before (fourth layer) and after (sixth layer) language-specific tuning. For comparison, two additional classifiers use activations from the single pathway model at the same layers. Figure 3.5 shows the average classification error rates for English and Dutch consonants and vowels, in the single vs. dual pathway models, as functions of the layer index. Hypothesis 4 is confirmed: vowel error rate decreases drastically for both Dutch and English if activations from the sixth layer of the dual pathway model are used instead of those from the fourth layer. On the other hand, going deeper into the single pathway model does not reduce error rates. Language-specific tuning

(the sixth layer of the dual-pathway model) is therefore demonstrated to give better representations (lower classification error rates) of non-shared phones (vowels), compared to an earlier layer of the same model (fourth layer), and compared to an equally deep layer of a language-independent model (sixth layer of the single pathway model).

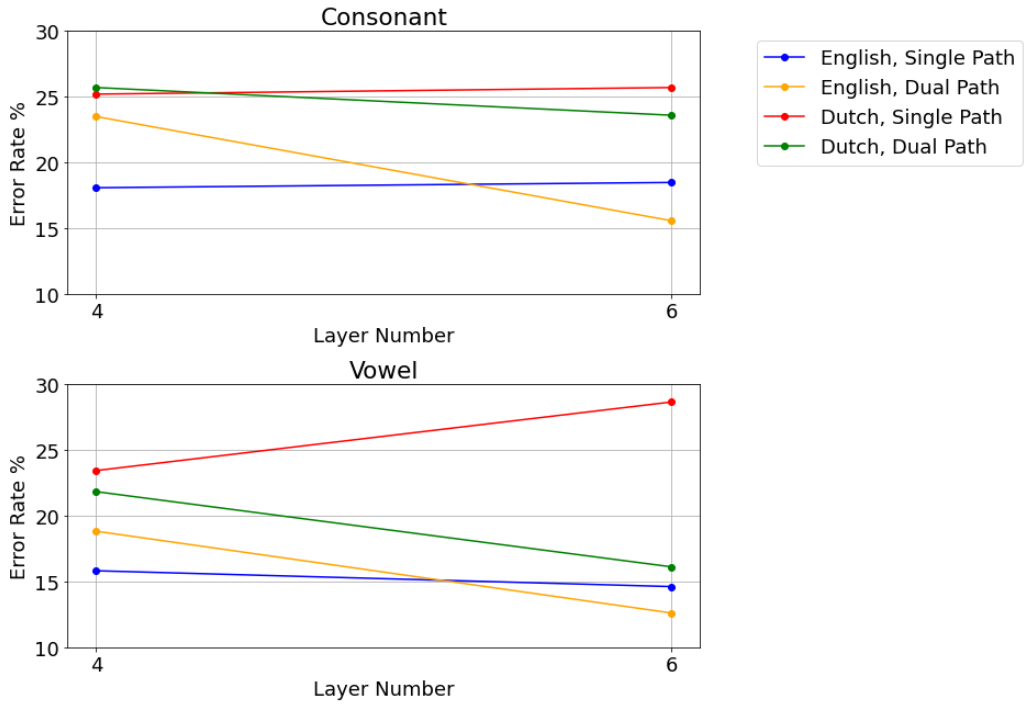

Figure 3.5: Classification error rates for consonants and vowels: MLP classification of the fourth vs. sixth layers of the single vs. dual pathway models, for English and Dutch.

Hypothesis 5 is partially confirmed for Dutch, but not for English: Dutch consonant error rates do not decrease much when moving from the fourth to the sixth layer of the dual pathway model (increased language dependence), but English consonant error rates decrease drastically. Apparently English consonants, despite displaying well-learned representations in Experiment 1, still benefit from language-specific tuning. Considering the fact that English is effectively the second language for our system, it is not surprising that English consonants benefit more from language-specific tuning than Dutch consonants. Error rates from the single pathway model are layer-independent for both Dutch and English, showing that the change in the dual pathway model is not a simple effect of having more layers. Contrary to Hypothesis 5,

consonant error rates are higher than vowels, possibly because there are more consonants than vowels in both languages.

CHAPTER 4

CYCLE-CONSISTENCY CONSTRAINT FOR MULTI-LINGUAL SPEECH RECOGNITION

In this chapter, we wish to apply the Sequential Representation Quantization Au- toEncoder (SeqRQ-AE) [54] to multi-lingual speech recognition with multiple speakers in each language. Under the current setting, we did not assume the availability of unpaired data for each language. Rather, the reconstruction loss calculated from the TTS module, conditioned on text predictions from the ASR module, was treated as an additional regularization term when training the ASR system. By doing so, we hope that the additional reconstruction loss could more promptly correct the errors produced by the ASR system.

This chapter is divided into three sections: algorithms, experiment settings, and experiment results. In the first section, we will first describe how the current model differs from the SeqRQ-AE model [54]. Specifically, we will describe how we trained and extracted speaker and language embeddings as conditional input for the TTS module and how we modified its architecture to reflect the predominant use of transformer [40] modules for sequence-to-sequence tasks. The second section will describe the specific corpora and model parameters used in this set of experiments. In the third section, we will start by validating our choice of speaker and language embedding through visualization and compare our baseline multi-lingual system's phonetic token error rates without cycle-consistency constraint to our modified system with such constraint.

4.1 Algorithm

The first subsection will describe how we trained a speaker encoder and a language encoder to provide conditional input for the TTS module. In the second subsection, we will describe our modified ASR system with the cycle-

consistency constraint. Different from the SeqRQ-AE model [54], which used convolutional layers and recurrent layers, we chose to use transformers as essential building blocks for the ASR system, as they work well for multilingual speech recognition [45]. Then speech reconstruction module is extended to a complete auto-regressive TTS system, which is also transformer-based.

4.1.1 Speaker and Language Embedding

In this subsection, we will discuss the algorithm for training speaker and language embeddings. The SeqRQ-AE model [54] was trained on a single-language, single-speaker corpus, LJSpeech [55], for their experiments. In our work, we would like to extend the method to a multi-lingual, multi-speaker training setting. Therefore, we pre-trained two additional modules, a language encoder, and a speaker encoder. The language encoder takes a raw mel-spectrogram and generates a language embedding for the utterance. The speaker encoder takes the same raw mel-spectrogram and generates a speaker embedding. Note that we chose to use dynamic embeddings instead of fixed embeddings for each language and each speaker. We also chose to train the language and speaker encoder from scratch on the multi-lingual corpora for better specificity.

For the language encoder, we used a simple four-layer convolutional neural network that mimics the architectural designs of VGG16 [68]. To train the language encoder, we fed the mel-spectrogram of the utterance into the network, averaged the encoded representation across time, and calculated the cross-entropy loss using its language id as the target:

$$L_{lid} = \sum_{i=1}^{n} -\log \text{Softmax}\left[\frac{1}{F_{le}^{(i)}} \sum_{f=1}^{F_{le}^{(i)}} \text{Language\_Encoder}\left(\mathbf{X}^{(i)}\right)\right]_{l^{(i)}} \tag{4.1}$$

where the Softmax$(\cdot)$ layer projects the output from the language encoder into logits and calculates a softmax distribution on top of the logits, $\mathbf{X}^{(i)} := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T'}\}$ denotes the input mel-spectrogram, $F_{le}^{(i)}$ denotes the number of output frames for the language encoder, and $l^{(i)} \in [L]$ denotes the language id label over L different languages. After the language encoder was trained,

the language embedding $\mathbf{w}^{(i)}$ of an input utterance $\mathbf{X}^{(i)}$ was calculated as

$$\mathbf{w}^{(i)} = \frac{1}{F_{le}^{(i)}} \sum_{f=1}^{F_{le}^{(i)}} \text{Language\_Encoder}\left(\mathbf{X}^{(i)}\right) \tag{4.2}$$

For the speaker encoder, while we could have directly extracted the x-vectors [53], we would like to explore other methods of obtaining dynamic speaker vectors. In particular, we trained a modified version of the self-expressing autoencoder [69] to extract speaker embeddings. We used a weighted combination of the triplet margin loss and the speaker classification loss as the objective [10]. The modified architecture is shown in Figure 4.1.

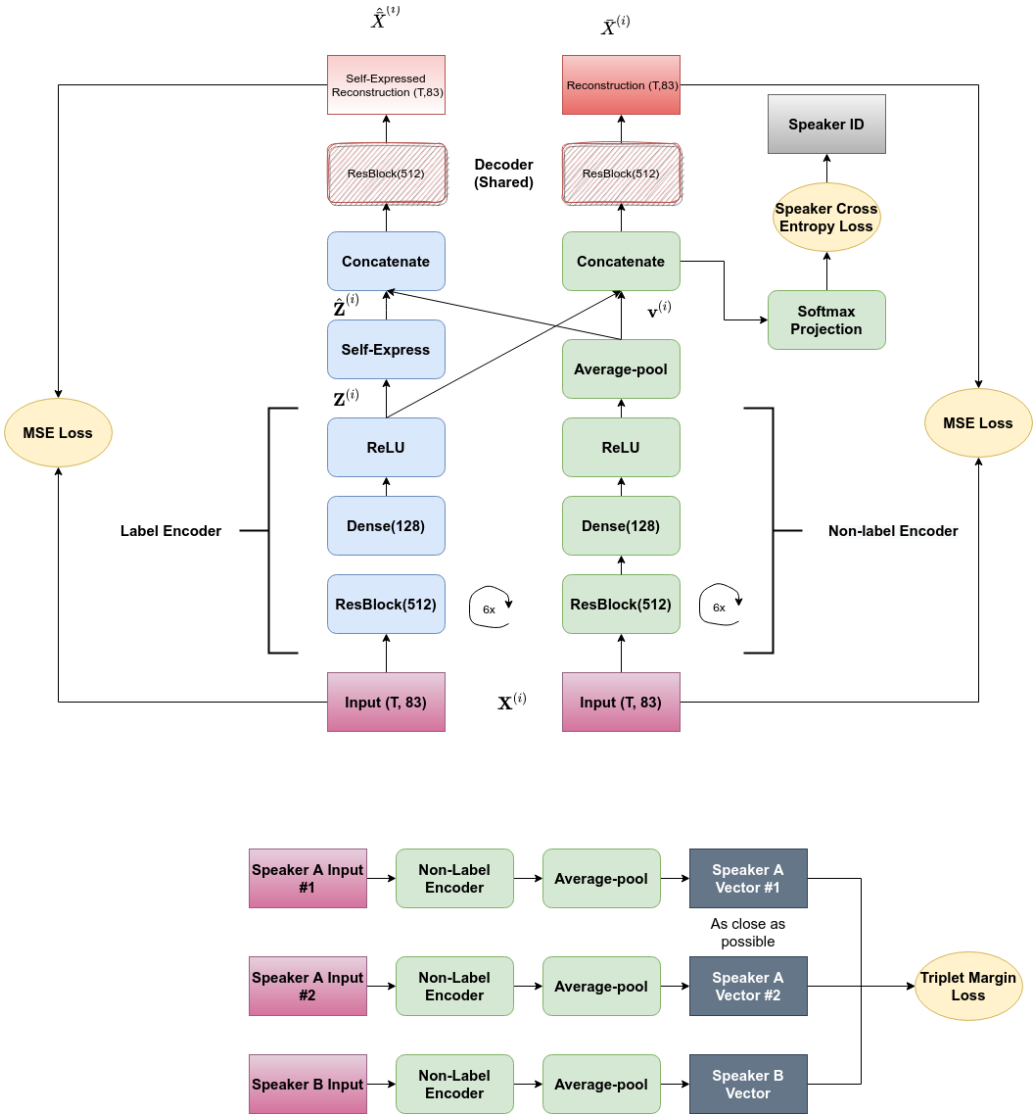

Figure 4.1: Self-expressing autoencoder for training speaker embeddings.

Denote the input as $\mathbf{X}^{(i)}$. The model consists of two encoders, a label encoder and a non-label encoder. Both encoders encode $\mathbf{X}^{(i)}$ through several blocks of residually-connected layers, followed by a final dense layer with non-linearity. Different from the label encoder, the non-label encoder averages the output across the time-dimension and obtains a single vector $\mathbf{v}^{(i)}$. The output $\mathbf{Z}^{(i)} := \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{T'}\}$ from the label encoder is used to calculate a self-expressed version $\hat{\mathbf{Z}}^{(i)} := \{\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \ldots, \hat{\mathbf{z}}_{T'}\}$. By using two encoders instead of one, we hope that we could separate out dynamic, phonetic information from static, speaker information more effectively.

The self-expressing module works as follows:

$$\mathbf{S}_{ij}^{(k)} = \frac{\mathbf{z}_i^{(k)T}\mathbf{z}_j^{(k)}}{\left\|\mathbf{z}_i^{(k)}\right\|\left\|\mathbf{z}_j^{(k)}\right\|} \tag{4.3}$$

$$\hat{\mathbf{S}}^{(k)} = \mathbf{S}^{(k)} - \mathbf{I} \tag{4.4}$$

$$\mathbf{D}_{ii}^{(k)} = \text{diag}\left(\sum_j \hat{\mathbf{S}}_{ij}^{(k)}\right) \tag{4.5}$$

$$\hat{\mathbf{Z}}^{(k)} = \left(\mathbf{D}^{(k)^{-1}}\hat{\mathbf{S}}^{(k)}\right)\mathbf{Z}^{(k)} \tag{4.6}$$

where $\mathbf{S}^{(k)}$ is the cosine similarity matrix calculated on the encoded sample $\mathbf{Z}^{(k)}$, and we subtract an identity matrix from $\mathbf{S}^{(k)}$ to obtain $\hat{\mathbf{S}}^{(k)}$. This is because the goal of the self-expressing module is to express a frame $\mathbf{z}_i^{(k)}$ with similar frames, such as those nearby that correspond to the same phoneme, thereby pushing the representations of frames under the same phoneme class closer to one another. Without subtracting the identity matrix, however, a diagonal $\mathbf{S}_{ij}^{(k)}$ could be learned without enforcing the self-expressing constraint [69].

After obtaining the pooled vector vector $\mathbf{v}^{(i)}$, the raw representation from the encoder $\mathbf{Z}^{(i)}$ and the self-expressed representation $\hat{\mathbf{Z}}^{(i)}$, each frame of $\mathbf{Z}^{(i)}$ and $\hat{\mathbf{Z}}^{(i)}$ are concatenated with $\mathbf{v}^{(i)}$. A single decoder, consisting of several residual blocks, calculates an output for the two types of representation individually, leading to two reconstructed mel-spectrograms $\bar{\mathbf{X}}^{(i)}$ and $\hat{\bar{\mathbf{X}}}^{(i)}$ (respective to $\mathbf{Z}^{(i)}$ and $\hat{\mathbf{Z}}^{(i)}$).

We believe that the speaker classification loss and the triplet margin loss would force the non-label encoder branch to focus on speaker-discriminative information. We thus denote the non-label encoder (right branch in Figure 4.1) as the speaker encoder, and $\mathbf{v}^{(i)}$'s as the speaker vectors. Note that as we chose to train a joint speaker encoder model on all training languages, we would like to avoid sampling two utterances from different languages when calculating the triplet margin loss, as the underlying language information would make it very easy to tell them apart. Therefore, we revised the batching scheme to contain at least four utterances from the same language: two from one speaker and two from another speaker. We also calculated the speaker classification loss in each language individually under similar concerns. As shown in the oval modules in Figure 4.1, the loss function is a sum

of three components:

1. the mean-squared error calculated on top of $\bar{X}^{(i)}$ and $\hat{\bar{X}}^{(i)}$:

$$L_{mse} = \sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \bar{\mathbf{X}}^{(i)}\|^2 + \sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \hat{\bar{\mathbf{X}}}^{(i)}\|^2; \tag{4.7}$$

2. the speaker classification loss calculated from $\mathbf{v}^{(i)}$:

$$L_{lid} = \sum_{i=1}^{n} -\log \text{Softmax}_{lid^{(i)}} \left(\mathbf{v}^{(i)}\right)_{s^{(i)}_{lid^{(i)}}} \tag{4.8}$$

   where Softmax_{lid} indexes the correct softmax projection layer to use based on the sample's language id, and $s^{(i)}_{lid^{(i)}}$ is the speaker target of that sample within the corresponding language;

3. the triplet margin loss calculated from $\mathbf{v}^{(i)}$, $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(k)}$, where $\mathbf{v}^{(j)}$ is calculated from a randomly sampled utterance of the same speaker as $\mathbf{v}^{(i)}$ (with $i \neq j$), while $\mathbf{v}^{(k)}$ is a randomly sampled utterance from a different speaker (but in the same language):

$$L_{margin} = \sum_{i=1}^{n} \max\left(\text{sim}\left(\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\right) + \lambda - \text{sim}\left(\mathbf{v}^{(i)}, \mathbf{v}^{(k)}\right), 0\right) \tag{4.9}$$

   The loss enforces the speaker embedding from the same speaker to be close, and at least a separation of λ for speaker embeddings from different speakers.

4.1.2 ASR-TTS with VQVAE

In this subsection, we will describe the modified ASR-TTS model we use for multilingual phonetic token recognition. The overall approach follows the SeqRQ-AE model formulation [54]. The model starts with an ASR encoder that takes a mel-spectrogram as input and predicts a sequence of hidden states. A vector quantization module (with a codebook size equal to the number of distinct phonetic tokens plus a blank token) takes the hidden states from the ASR encoder and quantizes them into discrete codebook vectors. Given the ASR encoder output and the set of codebook vectors,

per-frame output probability is calculated based on the distance from the encoded states to each codebook vector, which is then used to calculate the CTC loss [15]. The TTS encoder takes ground-truth phonetic transcripts as input (during TTS pre-training) or the symbol-level quantized ASR encodings (during ASR-TTS training; obtained by averaging over repeated frame-level codebook vectors) as input and outputs a sequence of text embeddings. The auto-regressive TTS decoder takes such embeddings to reconstruct speech, conditioned on the ground-truth shifted mel-spectrogram, the speaker embedding, and the language embedding (Section 4.1.1). The new multi-lingual, multi-speaker model is displayed in Figure 4.2.

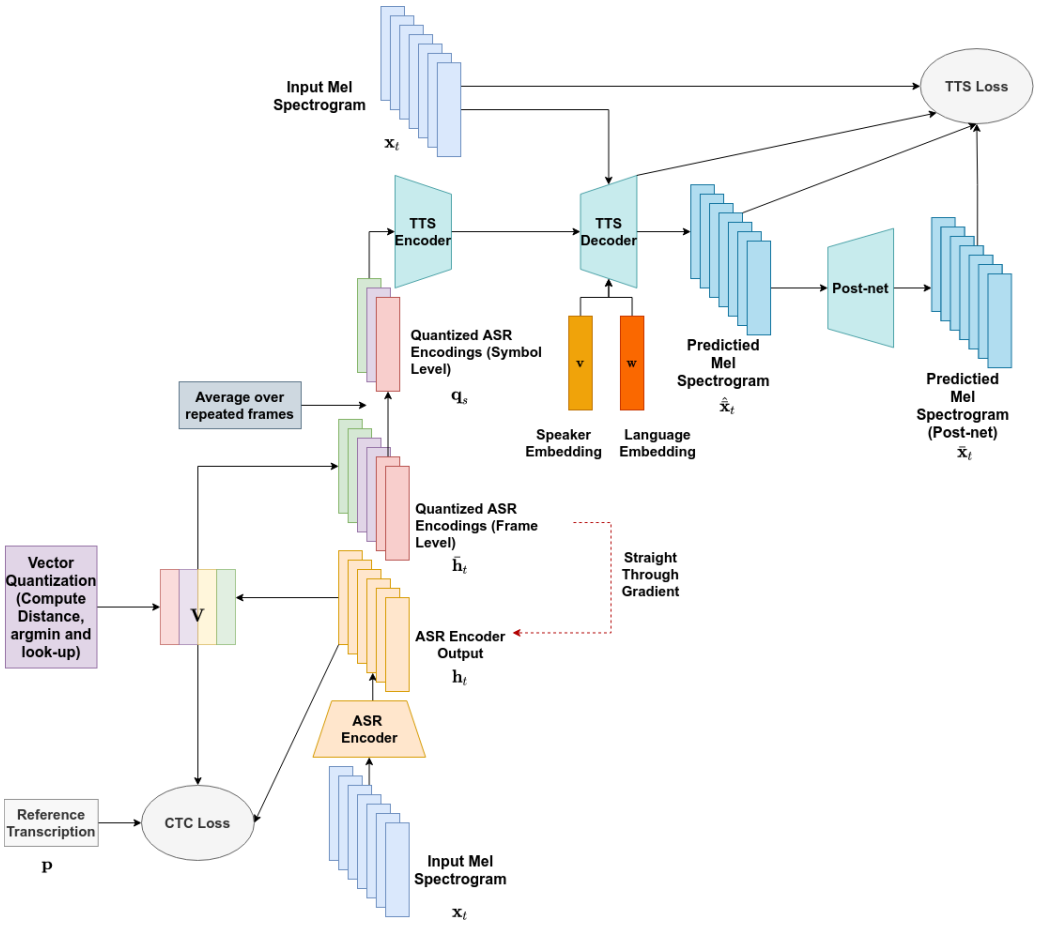

Figure 4.2: A multi-lingual, multi-speaker ASR-TTS model.

Before training the joint ASR-TTS model, we first pre-trained the TTS module, as we would like the cascaded TTS system to provide stable gradients to the ASR encoder during subsequent ASR training. The TTS model

adopts a recent transformer-based auto-regressive TTS model [50]. During pre-training, a ground-truth phonetic token index sequence $\mathbf{p} := \{p_1, \ldots, p_L\}$ is used to index a codebook of embeddings as $\mathbf{V}(\mathbf{p}) := \{\mathbf{V}_{p_1}, \ldots, \mathbf{V}_{p_L}\}$. The resulting $\mathbf{V}(\mathbf{p})$ is then fed into the Transformer-TTS encoder as

$$\mathbf{S} := \{\mathbf{s}_1, \ldots, \mathbf{s}_L\} = \text{Projection}\left(\text{Concat}\left(\text{TTS-Encoder}\left(\mathbf{V}\left(\mathbf{p}\right)\right), \mathbf{w}, \mathbf{v}\right)\right)$$

where $\mathbf{w}$ denotes the language embedding, and $\mathbf{v}$ denotes the speaker embedding. The concatenation and the re-projection are repeated over the time-dimension. Conditioned on $\mathbf{S}$, the decoder reconstructs the original mel-spectrogram via teaching-forcing. More details of the generic transformer-based encoder-decoder architecture can be found in Section 2.1.3, and more details of the Transformer-TTS architecture can be found in Section 2.2.2.

The final TTS loss contains three loss functions:

1. A reconstruction loss (L1 loss + MSE loss) between ground-truth and predicted mel-spectrograms, before and after the postnet in the decoder (with output denoted as $\bar{\mathbf{X}}^{(i)}$ and $\hat{\bar{\mathbf{X}}}^{(i)}$, respectvely):

$$\begin{aligned} L_{recon} = &\sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \bar{\mathbf{X}}^{(i)}\|_2^2 + \sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \hat{\bar{\mathbf{X}}}^{(i)}\|_2^2 \\ &+ \sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \bar{\mathbf{X}}^{(i)}\|_1 + \sum_{i=1}^{n} \|\mathbf{X}^{(i)} - \hat{\bar{\mathbf{X}}}^{(i)}\|_1 \end{aligned} \tag{4.10}$$

2. A guided attention loss [70] for the encoder-decoder attention matrix. The guided attention loss enforces a diagonal constraint for some attention matrix A:

$$L_{att}(A) = \sum_{n} \sum_{t} A_{nt} W_{nt} \tag{4.11}$$

with

$$W_{nt} = 1 - \exp\left\{-(n/L - t/T')^2 / 2\sigma^2\right\} \tag{4.12}$$

where for a specific attention matrix A (associated with a certain sample, decoder layer and attention head), n is the text position, t is the frame position, L is the total text length, and T' is the total number of frames. For a Transformer-TTS decoder, the constraint is only enforced over a selected number of layers and heads.

3. A frame-wise weighted binary cross-entropy loss, with a larger weight for positive stop tokens:

$$L_{bce} = \sum_{i=1}^{n} \sum_{t=1}^{T'^{(i)}} \alpha o_{i,t} \log P_{i,t} + (1 - o_{i,t}) \log(1 - P_{i,t}) \tag{4.13}$$

where $P_{i,t}$ is the probability that the output of the i-th sample should stop at the t-th frame, $o_{i,t}$ is the stop token target for that frame, and α is the weight for positive samples. Setting a large weight for the positive samples alleviates the imbalance between positive and negative samples.

After pre-training the TTS module, we trained the cascaded ASR-TTS model. The signal path of the model is shown in Figure 4.2. Denote the input mel-spectrogram as $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T'}\}$ and the intermediate hidden states after the ASR encoder as $\mathbf{H} := \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T\} := \text{Encoder}_\theta(\mathbf{X})$. For each time step $t \in [T]$, the quantized state vector $\bar{\mathbf{h}}_t$ is calculated from $\mathbf{h}_t$ as

$$\bar{\mathbf{h}}_t = \mathbf{h}_t + \mathbf{e}_v - \text{sg}(\mathbf{h}_t) \tag{4.14}$$

where the codebook vector $\mathbf{e}_v$ at time t is selected as

$$\mathbf{e}_v = \arg\min_{\mathbf{e}_k \in V} \|\mathbf{h}_t - \mathbf{e}_k\| \tag{4.15}$$

Here, V is a codebook with size equal to the size of phone token inventory plus an additional blank token, and $\text{sg}(\cdot)$ denotes the stop-gradient operator. Note that the above calculation of $\bar{\mathbf{h}}_t$ allows straight-through gradient during back-propagation, as shown in the dashed red path in Figure 4.2. If the input speech $\mathbf{X}$ has paired transcription, the frame-wise logits used for calculating CTC loss (Section 2.1.1) can be expressed as

$$P(v \mid \mathbf{h}_t) = \frac{\exp\left(-\|\mathbf{h}_t - \mathbf{e}_v\|_2\right)}{\sum_{\mathbf{e}_k \in V} \exp\left(-\|\mathbf{h}_t - \mathbf{e}_k\|_2\right)} \tag{4.16}$$

To reconstruct the input speech, $\mathbf{H} := \left\{\bar{\mathbf{h}}_1, \ldots, \bar{\mathbf{h}}_T\right\}$ is first collapsed in the time dimension by averaging over consecutive frames with the same codebook vector index, leading to $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_S\}$. Finally, we feed $\mathbf{Q}$ into the TTS module. The TTS loss is calculated in the same fashion as described above

for pre-training. The only difference is that we use predicted sequences of phonetic codebook vectors from the ASR module, instead of ground-truth transcripts, as inputs to the TTS encoder. The final loss function for ASR-TTS training is a weighted combination of the CTC loss and the TTS loss.

In this set of experiments, we do not use unpaired data and only wish to evaluate cycle-consistency as a constraint for the multilingual system.

4.2 Experiment Settings

We use ESPnet [71] as our ASR framework, which offers a complete ASR pipeline including data preprocessing, transformer [40] network implementation, network training and decoding. We choose Bulgarian, Czech, and Polish for multilingual training and Croatian for zero-shot recognition. All four languages come from the Slavic language family. The data for the languages come from GlobalPhone [72].

Using Kaldi [66], we extract 80-dim log Mel spectral coefficients with 25 ms frame size and 10 ms shift between frames, and augmented the frame vectors with three extra dimensions for pitch features. The transcriptions are converted to IPA symbols, or language-universal phonetic tokens [45], using LanguageNet grapheme-to-phone (G2P) models [73]. We treat diacritics (such as palatalization [ʲ]), suprasegmentals (such as length mark [ː] and primary stress mark [ˈ]), and tones (such as high tone [˥] and low tone [˩] symbols) as separate tokens. We also split diphthongs and affricates (if any) into individual symbols. For example, the G2P output /d a m vʲ i k/ would be transcribed into seven individual tokens instead of six. The resulting inventory size is 46, including four diacritics ([ʲ], [◌̩], [◌̪], [◌̃]) and two suprasegmentals ([ː], [ˈ]). During evaluation, we measure the phonetic token error rate (PTER) [45]. PTER is calculated by counting the insertion, substitution and deletion errors between reference phonetic token transcription and the CTC prediction.

The speaker encoder, which is trained as a self-expressing autoencoder with speaker classification loss and triplet margin loss, uses the same hyperparameters as in Figure 4.1. The label encoder, non-label encoder, and decoder all consist of six fully-connected residual blocks. Each residual block consists of three layers. The first and last layers are dense layers with size

512, followed by ReLU activation, and the middle bottleneck layer is dense with size 128. The input of the residual block is simply added to the final output of the three dense layers. The corpus for each language consists of around 60 to 80 speakers. The triplet loss margin is fixed to 0.1, and $\text{sim}(\cdot, \cdot)$ is chosen as the cosine similarity. The speaker embeddings come from the output of the non-label encoder after average-pooling of the time dimension.

The language encoder consists of four 2D convolutional layers. The number of channels for each layer is 64, 64, 128, and 128, respectively. All layers have a kernel size of three by three. A max-pooling layer of stride two comes after every two convolutions. The first max-pooling layer has a kernel size of three by three while the second one has a kernel size of two by two. The final tensor is reshaped into $(Batch\_Size, Frame\_Size, Feature\_Size)$. For each utterance, the output is then averaged over the time dimension to obtain the language embedding.

The baseline ASR encoder network, trained with vanilla CTC loss, starts with four 2D convolutional layers. The four convolutional layers have exactly the same hyperparameters as the language encoder. 12 self-attention encoder layers follow the convolutional layers, each having four heads, an attention dimension of 256, and a 2048-dim position-wise feed-forward layer. We use fixed positional encoding for the transformers. The encoder output is followed by a dense layer to compute frame-wise phonetic token posteriors and the CTC loss. All dropout rates are set to 0.1.

The TTS encoder-decoder network is based on Transformer-TTS [50]. The phonetic embedding layer has a dimension of 256, with 47 distinct codes (46 phonetic tokens and an additional blank token for use by the cascaded ASR-TTS system). The transformer encoder contains four self-attention encoder layers, each with four heads, an attention dimension of 384, and two layers of 1536-dim 1D convolutions with a kernel size of 1 and ReLU activation in between. The decoder pre-net contains two layers of 1D convolution and a reduced feature dimension of 256, with ReLU activation in between. The decoder also contains four layers. The self-attention modules of the decoder and the encoder-decoder attention modules all use the same hyperparameters as the encoder's layers. For both the encoder and the decoder, scaled positional encoding with a learnable scale parameter is used instead of fixed encoding. All dropout rates are set to 0.1 except for the decoder prenet and postnet, which are set to 0.5.

The joint ASR-TTS concatenates the baseline ASR encoder network with the TTS encoder-decoder network, as shown in Figure 4.2. The TTS module is initialized with pre-trained weights on the same multilingual, multi-speaker corpora. The codebook $\mathbf{V}$ is initialized to be the same as the phonetic embedding layer of the TTS encoder-decoder network. The cascading of the ASR system and the TTS system is achieved by feeding the symbol-level quantized ASR encodings $\{\mathbf{q}_1, \ldots, \mathbf{q}_S\}$ directly into the TTS encoder as a phonetic embedding sequence. For the ASR encoder network, the only difference is that the frame-wise phonetic token posterior is calculated via 4.16.

We run a total of ten epochs for the language encoder, 14 epochs for the speaker encoder, 100 epochs for TTS pre-training, and 30 epochs for the baseline ASR and the cascaded ASR-TTS system. All transformer models were warmed up for 4000 steps.

4.3 Experiment Results

In this section, we first visualize the extracted speaker and language embeddings to validate our design choices. After that, we will display the main results of this section, which is the phonetic token error rates of the baseline ASR system and the cascaded ASR-TTS system.

4.3.1 Speaker and Language Embedding Visualization

The language encoder scores a 99.7% accuracy on the training set and a 94.9% accuracy on the validation set. The speaker encoder obtains a 99.75% accuracy on the training set. The validation set contains a different set of speakers, so the accuracy calculation is not applicable.

For each language, we randomly sampled 1000 utterances and extracted language embeddings for those utterances. Figure 4.3 displays those language embeddings after reducing the dimension to two via T-SNE [67]. The plot shows well separation between embeddings from different languages and relatively well clustering of embeddings from the same language.

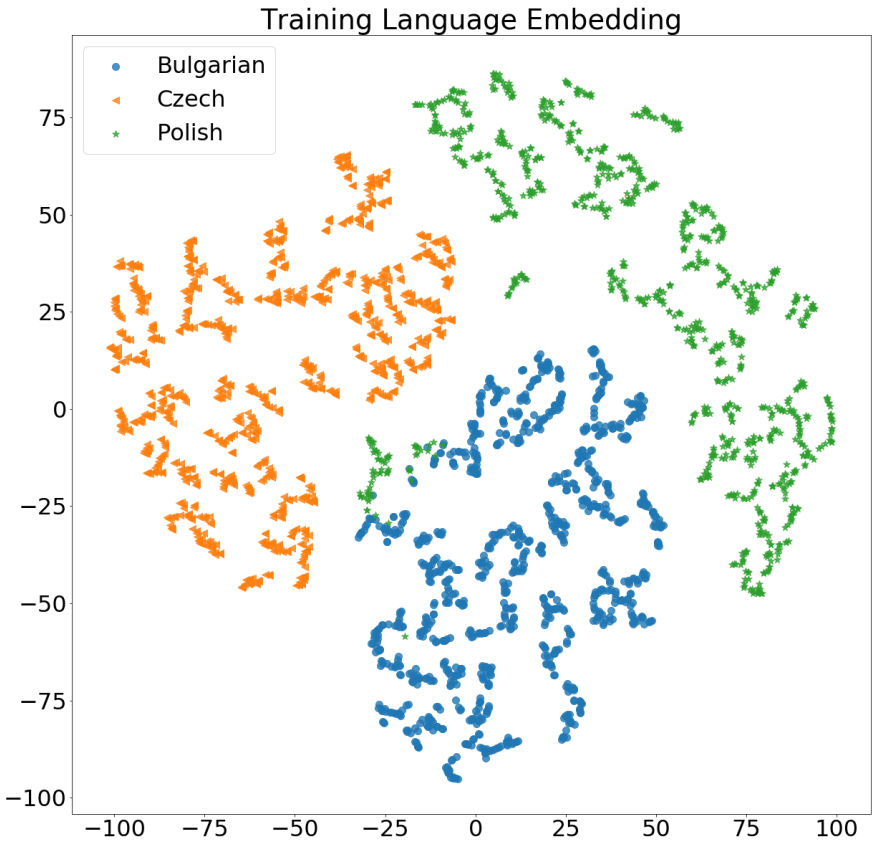

Figure 4.3: Language embeddings plotted via T-SNE.

To visualize the speaker embeddings, we chose five training speakers in Bulgarian and another three unseen speakers. We fed all the utterances of the chosen speaker into the speaker encoder and extracted speaker embeddings for those utterances. Figures 4.4 and 4.5 show that the embeddings from the same speakers are well cluster together and well separated from other speakers, even for unseen ones (although to a lesser degree). The autoencoder also achieved good reconstruction for both the vanilla reconstruction and the self-expressed reconstruction (although to a lesser degree), as shown in Figure 4.6.

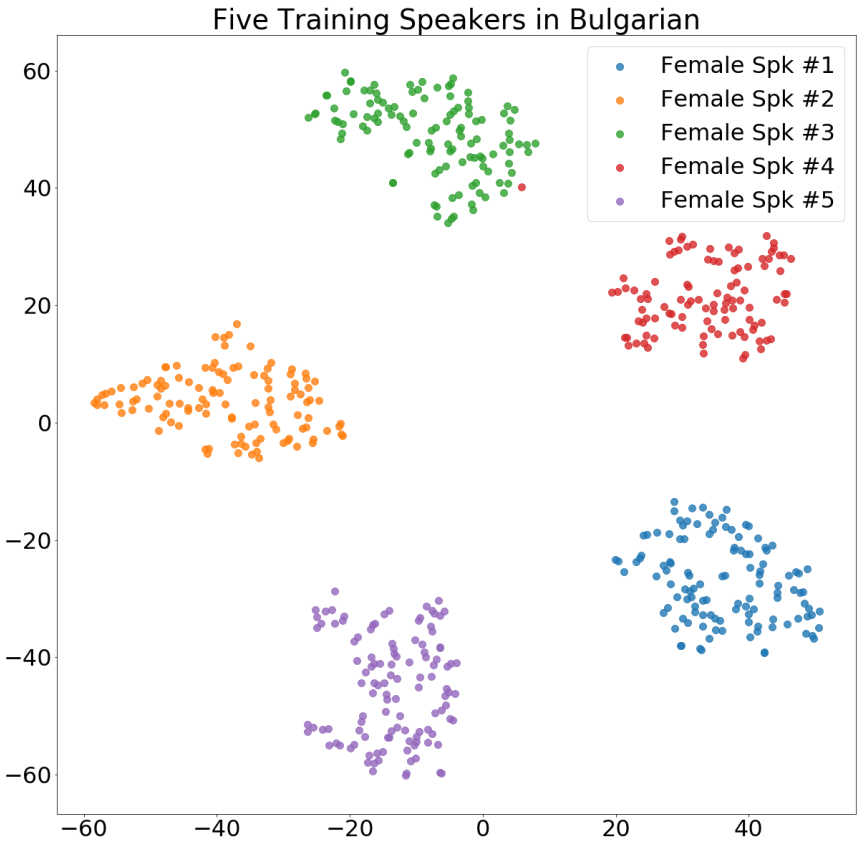

Figure 4.4: Speaker embeddings for five of the training speakers in Bulgarian.

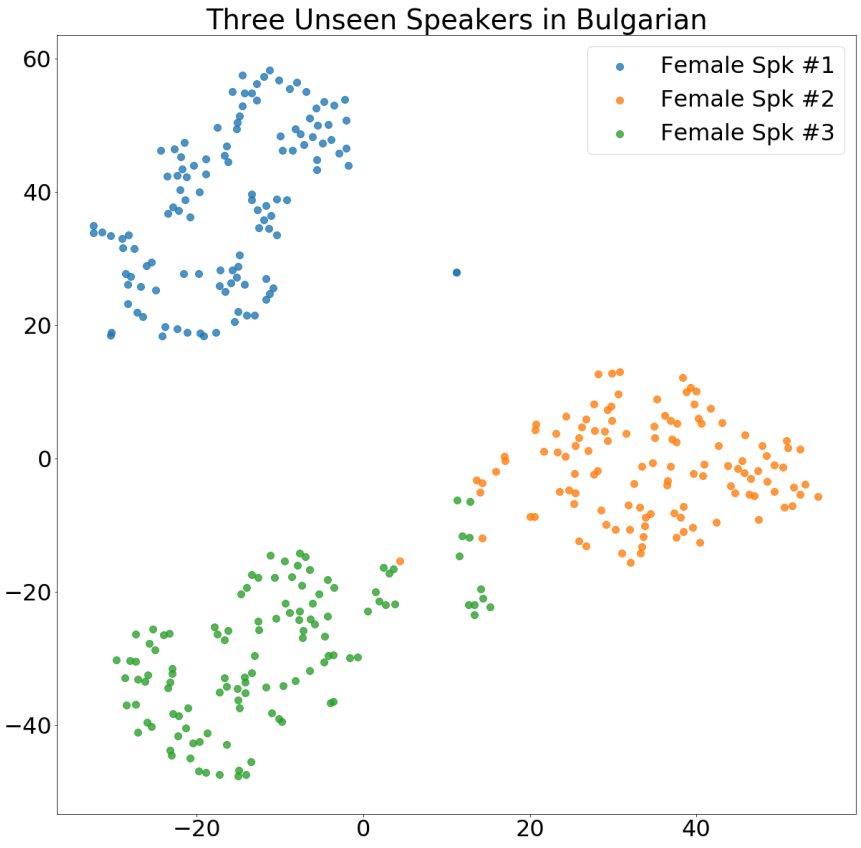

Figure 4.5: Speaker embeddings for three of the unseen speakers in Bulgarian.

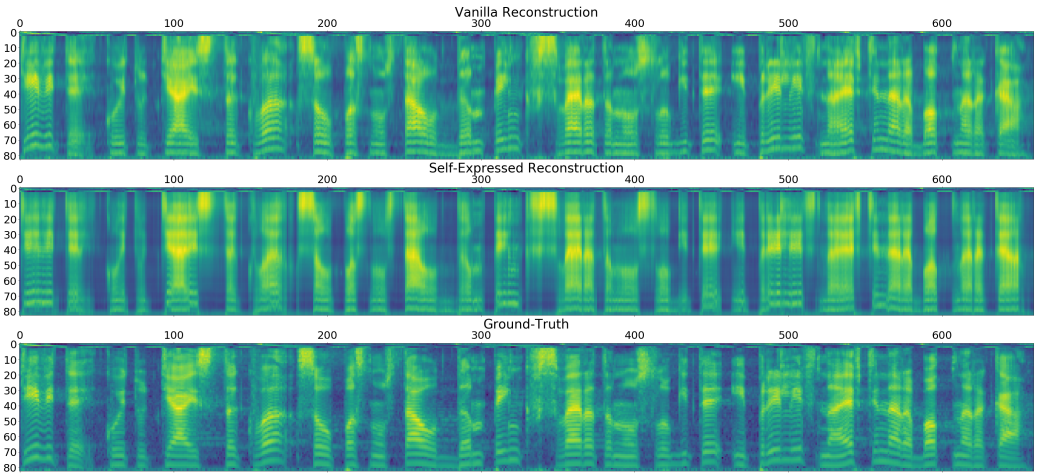

Figure 4.6: Reconstruction of the self-expressing autoencoder.

4.3.2 PTER Comparison

We trained three different models to understand if joint ASR-TTS training provides any benefit over a baseline CTC system. The first model is the vanilla CTC encoder network, where the frame-wise phonetic token posterior is calculated by projecting the encoder hidden states into phonetic token logits. The second model is almost the same as the vanilla CTC network. However, the frame-wise phonetic token posterior is calculated by first projecting the encoder hidden states into the codebook dimension and then invoking 4.16. The codebook in the second system is initialized with the codebook from the pre-trained TTS system and fixed during ASR training. The third system is the cascaded ASR-TTS system, where the codebook and all other TTS modules are initialized with weights from the pre-trained TTS system and fixed during ASR training. Table 4.1 displays the multi-lingual PTER calculated over the evaluation set for Bulgarian, Czech, and Polish, together with the zero-shot cross-lingual PTER calculated over the evaluation set for Croatian. The first system is denoted as "Linear-CTC," the second system is denoted as "Codebook-CTC," and the third is denoted as "ASR-TTS."

Surprisingly, the second system, intended to serve as an ablation study, gives the best PTER for all training languages. The second system performs better than the linear-CTC and the ASR-TTS systems on an unseen test

Table 4.1: Phone token error rates (PTER, %) of the "Linear-CTC," "Codebook-CTC," and "ASR-TTS" systems trained on three Slavic languages (Czech, Bulgarian and Polish). Early-stopping and other hyperparameters of each algorithm were selected based on development test data in the three training languages. Numbers reported are from the evaluation test data in each of the three training languages, and in the unseen Slavic language (Croatian).

	Training Languages				Test Language
Algorithm	Czech	Bulgarian	Polish	**Average**	Croatian
Linear-CTC	15.6	33.6	37.5	28.9	56.6
Codebook-CTC	**14.8**	**32.8**	**35.9**	**27.8**	**55.9**
ASR-TTS	16.2	34.0	37.9	29.4	57.8

language from the same language family. Comparing the first and second systems, we conclude that the pre-trained codebook can guide the phonetic token posterior much better than a randomly initialized logit projection layer. In other words, while the first system needs to learn a phonetic representation from scratch, the second system only needs to match such representations. The proposed ASR-TTS system, unfortunately, lags behind both the linear-CTC baseline and the codebook-CTC ablation study. This implies that the straight-through gradient provided by the TTS system is not very useful to the ASR system. A reason for the subpar performance is shown in Figure 4.7, which plots the encoder-decoder attention map from the last layer of the pre-trained TTS system for a training utterance. This attention map is far from ideal as only a selected number of inputs (horizontal axis) are attended by the decoder to produce speech (vertical axis). Training a multilingual, multi-speaker TTS system on this ASR corpus is a relatively difficult task, as there are a significantly large number of silence periods within most of the utterances. This is further complicated by the fact that are only 100 to 150 utterances for the 60 to 80 speakers in each training language.

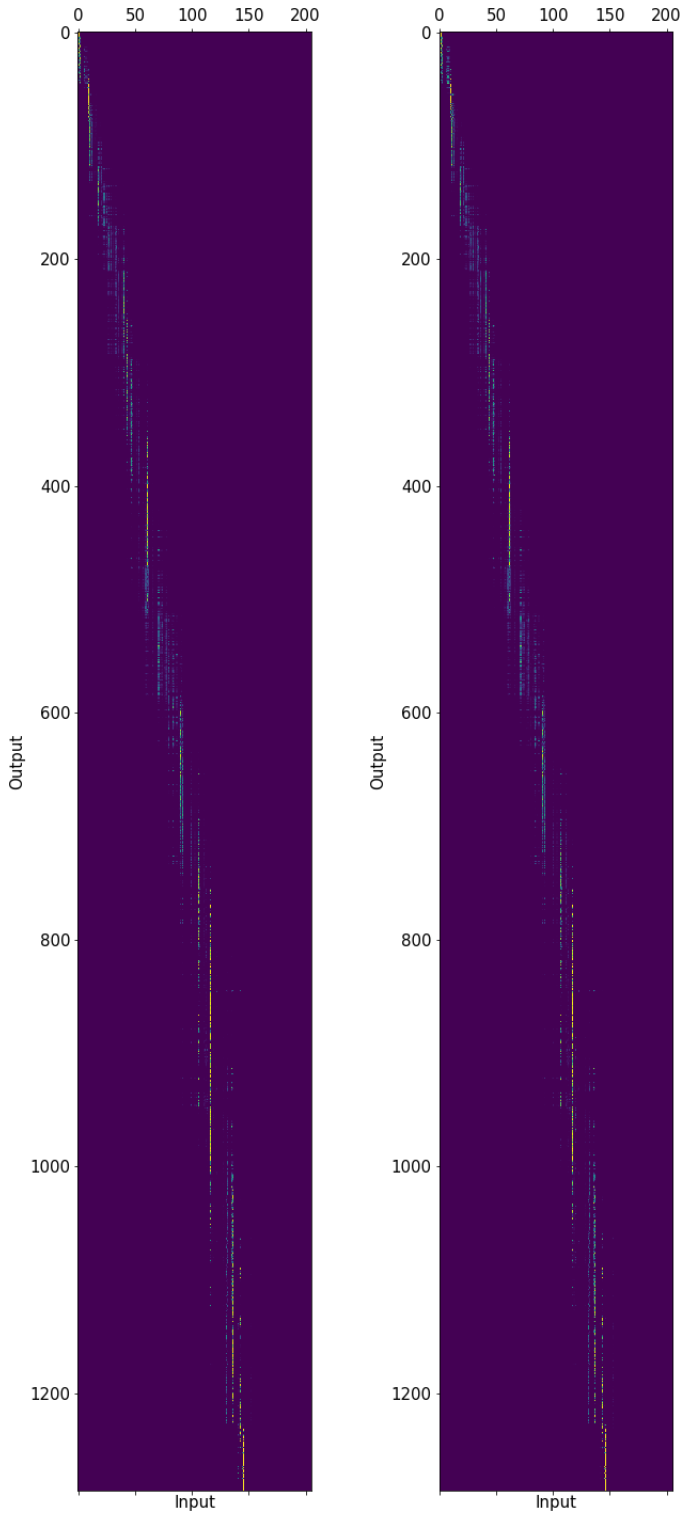

Figure 4.7: The encoder-decoder attention map from the last layer of the pre-trained TTS system for a training utterance.

CHAPTER 5

REGRET MINIMIZATION FOR MULTI-LINGUAL SPEECH RECOGNITION

In this chapter, we first present a modification of the original regret minimization method so that it is practical for multi-lingual ASR training (Section 5.1). We modified the regret term by replacing the leave-one-out classifier with a language-specific classifier on a different language. We then present our model, data, and training settings for phone token recognition and classification experiments in Section 5.2. The ERM and RGM results of the experiments on the 15-language training set and the Slavic training set (Section 5.3 will be used to prove or disprove the hypotheses in Section 6.3.

5.1 Algorithms

Language-independent ASR was first trained using empirical risk minimization (ERM). The regret minimization method proposed in Equation (2.101), however, is computationally impractical for ASR, because it requires optimizing an ASR separately for every leave-one-language-out subcorpus, $w^{-e} = \arg\min_h \mathcal{R}^{-e}(h \circ \phi)$; doing so is impractical when the subcorpus for each training language contains many hours of labeled speech.

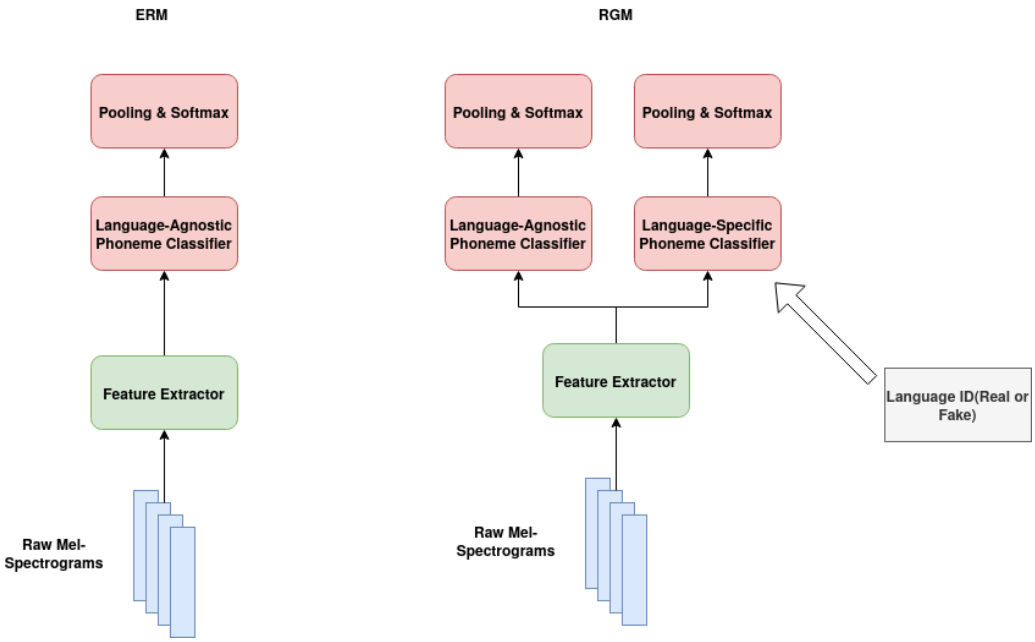

Figure 5.1: The modified architecture for regret minimization (RGM) versus the original architecture for empirical risk minimization (ERM). Both methods train a language-agnostic phone token classifier; RGM also trains language-specific phone token classifiers.

In order to make regret minimization practical for ASR, we modify Equation (2.101) into

$$\min_{w,\phi} \mathcal{R}(w \circ \phi) + \lambda \sum_{e,e':e \neq e'} [\mathcal{R}^e(w^{e'} \circ \phi) - \mathcal{R}^e(w^e \circ \phi)] \tag{5.1}$$

which is essentially replacing the leave-one-out classifier with the single-language classifier on *a different language*. This leaves us with one feature extractor, $\phi(\boldsymbol{X})$, $|\mathcal{E}|$ different single-language phone token classifiers $w^e(\boldsymbol{Z})$ (each of which is obtained by conditioning a common model on a specific language ID), and one language-agnostic phone token classifier $w(\boldsymbol{Z})$, as shown in Figure 5.1. Figure 5.1 compares empirical risk minimization (ERM), which trains only the language-agnostic classifier, to the modified RGM of Equation (5.1), which also trains language-specific phone token classifiers for each language in the training corpus. Each iteration of training consists of three steps:

1. Feed $\{\boldsymbol{X}^e\}$ into the single-language classifier, and perform K steps of

gradient descent to find $w^e = \arg\min_w \mathcal{R}^e(w \circ \phi)$.

2. Feed $\{\boldsymbol{X}\}$ into the language-agnostic classifier, and perform K steps of gradient descent to find $w = \arg\min_w \mathcal{R}(w \circ \phi)$.

3. Append a fake language label, $e' \neq e$, to each utterance. Train ϕ by performing one step of gradient descent on

$$\mathcal{R}(w \circ \phi) + \lambda[\mathcal{R}^e(w^{e'} \circ \phi) - \mathcal{R}^e(w^e \circ \phi)] \tag{5.2}$$

When the classifier $f = w \circ \phi$ is used as a phone token recognizer, its per-frame softmax outputs are scored using connectionist temporal classification [15]; when used as a phone token classifier, its per-frame logits are mean-pooled and then passed through a softmax nonlinearity, as stated in Figure 5.1.

5.2 Experimental Methods

We perform end-to-end phone token recognition experiments with unsegmented speech and phone token classification experiments with short speech segments. Sections 5.2.1 and 5.2.2 give details of our model parameters, data settings, as well as training and evaluation methods for all the experiments.

5.2.1 Phone Token Recognition

We use ESPnet [71] as our ASR framework which offers a complete ASR pipeline including data preprocessing, transformer network implementation [40], network training and decoding. We choose 15 languages as the multilingual set and an additional five languages as the cross-lingual set. Models are trained, validated, and tested using languages in the multilingual set; languages in the cross-lingual set are used only for testing. The details of our dataset are listed in Table 5.1.

Table 5.1: Sources of data used in our cross-lingual experiment. The upper part is the multilingual set and the lower part is the cross-lingual set. "Corpus" is GlobalPhone, corpus of spoken Dutch, or Babel. "Type" column denotes whether the corpus contains spontaneous (Sp.) or read speech. "Len" column shows the total duration of all utterances in hours. "Family" column shows the language family.

Language	**Abbr**	**Corpus**	**Type**	**Family**	**Len**
Portuguese	por	GP	Read	Romance	26
Turkish	tur	GP	Read	Turkic	17
German	deu	GP	Read	Germanic	18
Bulgarian	bul	GP	Read	South Slavic	21
Thai	tha	GP	Read	Tai	22
Mandarin	cmn	GP	Read	Sinitic	31
French	fra	GP	Read	Romance	25
Czech	ces	GP	Read	West Slavic	29
Dutch	nld	CGN	Read	Germanic	64
Georgian	kat	Babel	Sp.	Kartvelian	190
Javanese	jav	Babel	Sp.	Austronesian	204
Amharic	amh	Babel	Sp.	Ethiopic	204
Zulu	zul	Babel	Sp.	Bantu	211
Vietnamese	vie	Babel	Sp.	Vietic	215
Bengali	ben	Babel	Sp.	Indo-Aryan	215
Croatian	hrv	GP	Read	South Slavic	16
Polish	pol	GP	Read	West Slavic	24
Spanish	spa	GP	Read	Romance	22
Lao	lao	Babel	Sp.	Tai	207
Cantonese	yue	Babel	Sp.	Sinitic	215

Data are extracted from three publicly available corpora: GlobalPhone [72], the corpus of spoken Dutch [60], and Babel [74]. The former two corpora contain read speech, while Babel contains primarily spontaneous speech.

Due to the sampling rate differences among corpora, we first upsample all audio signals to 16 kHz. Using Kaldi, we then extract 80-dimensional log Mel spectral coefficients with 25 ms frame size and 10 ms shift between frames, and augment the frame vectors with three extra dimensions for pitch

features. The transcriptions are converted to IPA symbols using LanguageNet grapheme-to-phone (G2P) models [73]. Following the approach of a recent transformer-based multilingual speech recognition system [45], ASR is trained end-to-end with an output vocabulary consisting of *phone tokens* instead of phones. A phone token is defined to be any single character in the IPA transcription, including base phones, diacritics, and tone symbols; the Cantonese syllable nucleus [aː˥], for example, is decomposed into four phone tokens: /a/, /ː/, /˥/, and /˥/. The resulting phone token inventory contains the 95 distinct IPA characters present in phoneme transcriptions of the 15 training languages. IPA characters present in the test languages, but not in the training languages, are each mapped to the closest token in the phone token inventory.

The encoder part of our transformer network starts with two 2D convolutional layers with a subsampling factor of four, followed by 12 self-attention encoder layers, each having four heads, an attention dimension of 256 and a 2048-dim position-wise feed-forward layer. The encoder output is passed through a dense layer to compute frame-wise phone token posteriors, which are scored using connectionist temporal classification (CTC, [15]).

For the experiments involving the Slavic subset, we chose the four Slavic languages (Bulgarian, Czech, and Polish for multilingual training and Croatian for cross-lingual testing) out of the 20-language set. The features used are the same as those from the 20-language experiment, but the label set contains only the phone tokens from the three multilingual training languages. This results in a total of 46 phonetic tokens. For recognition scoring purposes, OOV IPA characters in Croatian are each mapped to the closest token in the phone token inventory. Two additional test languages, French and German, are also used for further evaluation, but any OOV tokens are mapped to UNK instead.

All transformer models were trained for 30 epochs, and warmed up for 25000 steps.

5.2.2 Phone Token Classification

In addition to recognition experiments, we also tested all training algorithms in a phone token classification experiment, using training and test data from

only Polish, Bulgarian, Czech, Croatian, French and German. Grapheme-to-phoneme transducers were first applied to the original text transcriptions to obtain IPA transcriptions. IPA transcriptions were then split into individual phone tokens. There are no lexical tones in these six languages, but several of them use other diacritics: the IPA lengthening symbol (/ː/) composed 3.2% of all phone tokens, and other diacritics composed 2.3% of the remaining phone tokens (0.3–0.6% each). Triphone hidden Markov models (HMMs) were trained for each language individually, with phone tokens as targets; for example, the triphone /a-ː+p/ denotes the sound made by the IPA lengthening symbol (/ː/) when it follows the vowel /a/, and precedes the consonant /p/. Kaldi was used to train and cluster the triphones, and to force-align them to audio, in order to find physical segment boundaries for each triphone. Based on the forced alignment, we then segmented variable-length phone token utterances from the audio to construct a multilingual phone token classification dataset. The training set was further subsampled by a factor of three, leading to 688 k training pairs. We then trained a model consisting of six transformer encoder layers (instead of 12 as in the previous experiments; all other architectural details are the same when applicable) and mean-pooled the time steps to obtain phone token logits, which are fed forward to a single softmax nonlinearity for the entire phone token segment. Phone tokens that appear in the test languages (Croatian, French and German) but not in the training languages (Czech, Bulgarian and Polish) were excluded from the evaluation corpus.

5.3 Results

Table 5.2 lists phone token error rates (PTER, %) of an ASR trained using fifteen languages, and tested on five additional languages. The fifteen training languages were chosen to span ten language families; the five test languages were chosen to be members of five of the same families. Parameters of the ASR were trained using training data in the fifteen languages shown in the left column. Each neural network was trained until PTER reached a minimum on development test data in the fifteen training languages (a strategy sometimes called *early stopping* [75]). Other hyperparameters, including multi-task training weights for RGM, were also optimized for minimum error

on development test data in the training languages. The results reported in Table 5.2 were then measured using evaluation test data in both training and test languages. As shown, ASR trained using empirical risk minimization (ERM, Equation (2.93)) gave the best results on average, and for every language individually.

Table 5.2: Phone token error rates (PTER, %) of an ASR trained on 15 languages, tested on five additional languages. Early-stopping epoch and other hyperparameters of each algorithm were selected based on development test data in the training languages. Numbers reported are from the evaluation test data in each language.

Training Languages			Test Languages		
Language	ERM	RGM	Language	ERM	RGM
Portuguese	18.4	22.1	Croatian	47.8	50.9
Turkish	21.3	25.0	Polish	62.5	65.5
German	26.1	29.4	Spanish	38.1	40.6
Bulgarian	27.0	30.2	Lao	78.2	78.8
Thai	26.1	34.5	Cantonese	77.0	77.7
Mandarin	30.0	46.3	-	-	-
French	13.7	16.8	-	-	-
Czech	11.0	13.7	-	-	-
Dutch	21.3	27.6	-	-	-
Georgian	38.0	41.5	-	-	-
Javanese	47.0	49.6	-	-	-
Amharic	44.7	49.7	-	-	-
Zulu	42.4	46.3	-	-	-
Vietnamese	52.3	58.5	-	-	-
Bengali	40.2	43.4	-	-	-
Average	30.6	35.6	**Average**	60.7	62.7

Table 5.3 lists PTER (%) of an ASR trained using three languages from the Slavic language families. The ASR was also tested on one Slavic test language (Croatian), and two non-slavic Indo-European languages (French and German). Early stopping and hyperparameter optimization were performed using development test data in the training languages. Table 5.3 reports PTER measured using evaluation test data in all six languages. The results are quite different from those shown in Table 5.2. ERM achieves the lowest

error rates on the three training languages, and on the test language that is drawn from the same language family (Croatian), but both French and German achieve lower error rates using regret minimization.

Table 5.3: Phone token error rates (PTER, %) of an ASR trained on three Slavic languages (Czech, Bulgarian and Polish). Early-stopping and other hyperparameters of each algorithm were selected based on development test data in the three training languages. Numbers reported are from the evaluation test data in each of the three training languages, and in each of three previously unseen test languages.

	Training Languages			
Algorithm	Czech	Bulgarian	Polish	**Average**
ERM	**26.4**	**41.7**	**44.9**	**37.7**
RGM	32.3	46.0	48.2	42.2
	Test Languages			
Algorithm	Croatian	French	German	**Average**
ERM	**56.4**	71.5	65.4	64.4
RGM	57.1	**69.2**	**65.3**	**63.9**

Table 5.4 lists phone token classification error rates (PTCER) for the same six languages listed in Table 5.3. As described in Section 5.2.2, these experiments were performed by segmenting each audio file using forced alignment with a monolingual phone-token HMM ASR. The resulting phone token segments were then classified using a transformer-based phone token classifier, whose parameters, hyperparameters, and early-stopping schedule were optimized using training data and development test data from Bulgarian, Polish, and Czech.

Table 5.4: Phone token classification error rates (PTCER, %) of an ASR trained on three Slavic languages (Czech, Bulgarian and Polish). Early-stopping and other hyperparameters of each algorithm were selected based on development test data in the three training languages. Numbers reported are from the evaluation test data in each of the three training languages, and in each of three previously unseen test languages.

	Training Languages			
Algorithm	Czech	Bulgarian	Polish	**Average**
ERM	**29.7**	**46.2**	**42.9**	**39.6**
RGM	32.0	49.0	45.7	42.2
	Test Languages			
Algorithm	Croatian	French	German	**Average**
ERM	**48.3**	**56.6**	**59.3**	**54.7**
RGM	48.8	57.3	64.3	56.8

Table 5.5 lists phone token classification error rates (PTCER) for transformer-based phone classifiers trained exactly as in Table 5.4, except that training is stopped in a different manner. In Table 5.4, training was stopped when PTCER reached a minimum on development test data in the training languages. In Table 5.5, however, training was stopped when PTCER reached a minimum on development test data in one of the test languages. Numbers in boldface in Table 5.5 highlight the best results achieved when parameters are trained in (three) training languages, early-stopping is timed using a (fourth) development-test language, and then the system is evaluated in a (fifth) evaluation-test language. As shown, early-stopping using a development-test language outperforms early-stopping using a training language in all of the three languages.

Table 5.5: Phone token classification error rates (PTCER, %) of an ASR trained on three Slavic languages (Czech, Bulgarian and Polish) and tested on one Slavic language (Croatian) and two other Indo-European languages (French and German). In this table, the epoch for early stopping was chosen using development-test data from one of the three test languages: Croatian in rows 1-2, French in rows 3-4, German in rows 5-6. PTCER was then measured using evaluation-test data from each test language. Numbers reported using early-stopping on the test language are considered oracle; boldface shows the lowest non-oracle error rate.

	Early-Stopping	Eval Languages		
Algorithm	Language	Croatian	French	German
ERM	Croatian	46.6	59.4	63.4
RGM	Croatian	44.9	**56.2**	**57.2**
ERM	French	46.8	58.4	60.6
RGM	French	47.5	56.0	60.5
ERM	German	48.9	62.2	59.3
RGM	German	**44.9**	**56.2**	57.2

Inspired by the success of the cross-evaluation method in Table 5.5, we re-calculated the PTER for the recognition task performed for Table 5.3 using the same process. For the transformer-based end-to-end phone recognizer trained exactly as in Table 5.3, we stopped training when PTER reached a minimum on development test data in one of the test languages instead of in the training languages. The results are displayed in Table 5.6. Numbers in boldface in Table 5.5 highlight the best results achieved when parameters are trained in (three) training languages, early-stopping is timed using a (fourth) development-test language, and then the system is evaluated in a (fifth) evaluation-test language. Under this method, ERM was able to outperform RGM on two out of the three test languages. Compared to Table 5.3, ERM was able to reduce its non-oracle error rates on all three test languages significantly with cross-evaluation. At the same time, RGM could only reduce the non-oracle error rate on one test language. The average non-oracle error rate for RGM on the three test languages is 63.6%, while the average non-oracle error rate for ERM is 63.3%.

Table 5.6: Phone token error rates (PTER, %) of an ASR trained on three Slavic languages (Czech, Bulgarian and Polish) and tested on one Slavic language (Croatian) and two other Indo-European languages (French and German). In this table, the epoch for early stopping was chosen using development-test data from one of the three test languages: Croatian in rows 1-2, French in rows 3-4, German in rows 5-6. PTER was then measured using evaluation-test data from each test language. Numbers reported using early-stopping on the test language are considered oracle; boldface shows the lowest non-oracle error rate.

	Early-Stopping	Eval Languages		
Algorithm	Language	Croatian	French	German
ERM	Croatian	55.7	69.1	**65.1**
RGM	Croatian	54.7	**68.4**	65.9
ERM	French	**55.7**	69.1	**65.1**
RGM	French	56.7	68.4	65.8
ERM	German	**55.7**	69.1	65.1
RGM	German	57.1	69.1	65.3

Figures 5.2 and 5.3 displays the confusion matrices for ERM and RGM calculated on the evaluation-test data from the unseen language Croatian. The model for ERM is chosen as the one that gives a 46.8% PTCER in the third row of Table 5.5, and the model for ERM is chosen as the one that gives a 44.9% PTCER in the sixth row of 5.5. These correspond to the best non-oracle models for ERM and RGM when evaluated on Croatian using the evaluation scheme that generates Table 5.5. The <oov> entry in the confusion matrix corresponds to all phone tokens in the inventory of the training languages that do not exist in Croatian. The rest of the entries in the matrix corresponds to a Croatian phone token. As shown, the confusion matrix for RGM has a more diagonal structure and is less likely to output a phone token that is not in Croatian.

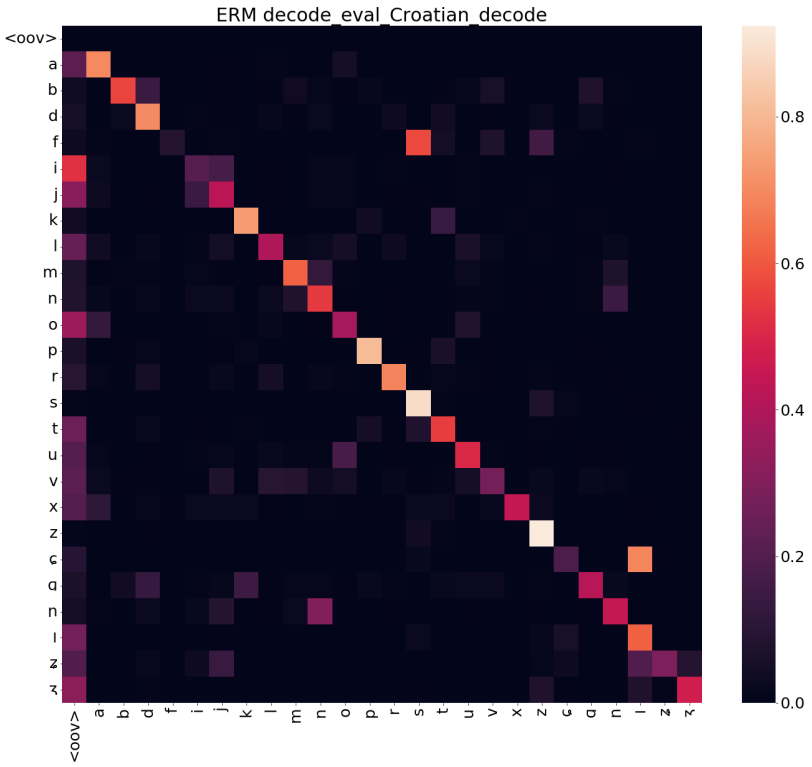

Figure 5.2: The confusion matrix for the ERM model that gives a 46.8% PTCER on Croatian in the third row of Table 5.5. The rows are labeled with ground-truth phone tokens, and the columns are labeled with predicted phone tokens. Each entry in the confusion matrix is the output probability of a phone token X in the inventory of the training languages (Czech, Bulgarian and Polish), given that the speech segment has a ground-truth label of Y. Brighter colors represent a higher probability and darker ones represent a lower probability. Each entry in the <oov> column aggregates the probabilities of all output phone tokens in the inventory that do not exist in Croatian.

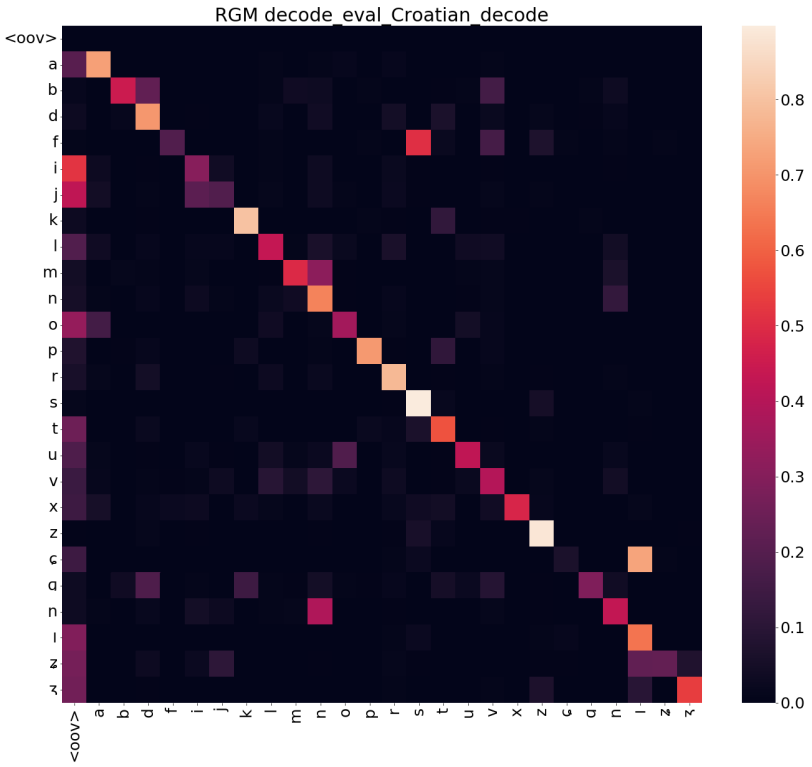

Figure 5.3: The confusion matrix for the RGM model that gives a 44.9% PTCER on Croatian in the sixth row of Table 5.5. The rows are labeled with ground-truth phone tokens, and the columns are labeled with predicted phone tokens. Each entry in the confusion matrix is the output probability of a phone token X in the inventory of the training languages (Czech, Bulgarian and Polish), given that the speech segment has a ground-truth label of Y. Brighter colors represent a higher probability and darker ones represent a lower probability. Each entry in the <oov> column aggregates the probabilities of all output phone tokens in the inventory that do not exist in Croatian.

CHAPTER 6

DISCUSSION

In this chapter, some further thoughts are provided for the three individual experiments. In the first section, we re-iterate some of the findings related to language-specific tuning for a bi-lingual CTC network and discuss the implications of designing a multi-lingual model. In the second section, we discuss how our ASR-TTS model compares with previous models, why it failed to improve the phonetic token error rates (PTERs) on multi-lingual speech recognition, and how we could improve the model in the future. In the third section, we revisit the hypotheses proposed at the end of Section 1.3 regarding the comparison between empirical risk minimization (ERM) and regret minimization (RGM) in phone token classification and recognition.

6.1 Further Thoughts on Language-specific Tuning

The results in Chapter 3 show that, while a neural network respects the commonalities between some phonetic information among two related languages, it may still benefit significantly from language-specific tuning modules to learn better representations for the two languages individually. Visualizations at the shared layer of the dual pathway model show that consonant representations are highly consistent and well-shared between the two languages. Visualizations of the single and dual pathway models at the last bLSTM layer show that the dual pathway model learns a better representation of the vowel space. Experiment three summarizes our findings by using the activations from the 4th and 6th layers of both models as input features of a phone classifier. The results show that language-specific tuning layers of our dual pathway model provide better features for both Dutch and English vowels and English consonants. Using this simplified model and on a restricted set of two languages, we have unveiled some of the inner details

of how a multi-lingual system deals with shared and non-shared phonetic information across languages.

Our results suggest that even a multilingual system should incorporate language-specific tuning. However, we have to note that simply creating different branches for language-dependent processing, as we did for Dutch and English in the dual pathway model, may not work well for a typical multilingual system that has a large set of training languages. As the number of languages increases, so does the total number of parameters in those different branches. Therefore, the language-dependent parameters will quickly overwhelm the language-independent parameters at earlier layers, thus limiting the system's capability of extracting language-independent features, which we have shown to be helpful for learning most of the consonants in English and Dutch. In the future, we aim to design a new architecture that can more efficiently separate language-independent information from language-dependent information in layers deeper than the softmax and investigate how much a multilingual system can benefit from such decoupling.

Another important insight from our result, in terms of designing a multilingual system, is that some specific groups of phonemes may depend on language-specific tuning more than other groups for the purpose of achieving good cluster separation and low classification error rates. In our case, the vowels in English and Dutch require language-specific tuning, apparently because few of them are shared. Most consonants are shared between the two languages, and perhaps, for this reason, consonants receive far less benefit from language-specific tuning. Both Dutch and English consonants achieve good cluster separation without language-specific tuning (Experiment 1); English consonant error rates benefit from language-specific tuning, but Dutch consonants do not (Experiment 3). End-to-end ASR has achieved tremendous success because it can achieve low ASR monolingual error rates without consideration of a language's phone inventory [36], and low multilingual error rates without consideration of the similarity of the phone inventories [76]. However, our analysis suggests that better cluster separation and lower classification error rates can be achieved by incorporating model components that specifically target the differences and similarities between the L1 and L2 phone inventories.

6.2 Lessons and Future Directions for Cycle-consistency in Multilingual Speech Recognition

In Chapter 4, we conducted an experiment to extend the Sequential Representation Quantization AutoEncoder (SeqRQ-AE) [54] to multi-lingual and multi-speaker speech recognition. While we incorporated the sequential representation quantization module with a VQVAE, our model contains a full TTS encoder-decoder instead of just a decoder. Like the end-to-end feedback models in machine speech chain [10] and another cycle-consistency model fine-tuned on unpaired speech and text [14], we conditioned the TTS module on dynamic speaker embeddings. Unlike these two models, we modified a self-expressing autoencoder [69] for learning speaker embeddings and additionally conditioned the TTS module on dynamic language embeddings. We also did not use an ASR decoder as for the SeqRQ-AE model [54] and calculated the CTC loss based on the distance of the hidden encoder states to the codebook vectors.

As shown in Table 4.1, the end-to-end feedback ASR-TTS model did not beat the baseline with a linear-layer CTC loss on top of Transformer encoders, even when we pre-trained the TTS model. Interestingly, however, the ablation study with a modified CTC loss calculated with phonetic token codebook vectors (according to the frame-wise emission probability in Equation (4.16) gave lower PTERs on all training and test languages. Note that for this system, the codebook vectors were still initialized with the phonetic token embeddings from the pre-trained TTS system, so in essence, ASR learning is still guided with TTS pre-training. Unfortunately, the straight-through gradients from the pre-trained TTS module failed to guide the learning of the ASR systems further. Future work would explore different gradient back-propagation schemes from TTS back to ASR, such as the policy-gradient approach used to train the ASR-TTE model [13].

Another possible reason the TTS gradients did not help ASR learning is that TTS pre-training did not converge with good encoder-decoder attention. Note that the TTS model was trained with teacher-forcing for both the pre-training stage and the ASR-TTS joint training stage. Therefore, without a good attention map to gain useful information for reconstructing speech,

the decoder pre-net would instead pass through much information from the ground-truth spectrogram to drive down the loss, even if the spectrogram is right-shifted. Later work not directly related to the experiments in Chapter 4 shows that Tacotron 2 [48] and Transformer-TTS [50] still could suffer from less-than-ideal convergence of encoder-decoder attention when trained on a multi-lingual, single-speaker corpus. On the other hand, the work identified that a simpler, fully-convolutional architecture [70] provided better convergence and less erroneous inference on the same multi-lingual, single-speaker corpus. Future work will focus on incorporating this fully-convolutional architecture with both speaker and language embedding.

Unlike most previous work, we did not assume the availability of additional unpaired data and directly used the end-to-end feedback loss on available paired data. Our results indicated that the use of unpaired speech may have indeed played a massive part in the success of previous ASR-TTS systems. We want to utilize multi-lingual unpaired speech data for the end-to-end ASR-TTS feedback system for future work. We could also utilize unpaired text data, which are easier to collect, once we designed a good enough TTS model for multi-lingual, multi-speaker scenarios.

Other directions for future exploration include trying out alternative ways for extracting speaker and language embeddings. This includes trying out pre-trained speaker vectors from large speaker recognition tasks [53]. Note that while these speaker vectors may not provide the same specificity as the speaker vectors obtained on the training set, they could allow few-shot adaptation of the system to unseen speakers in a new language. Such few-shot adaptation also requires training the dynamic language embedding on raw speech from many more different languages, possibly spanning across different language families as well.

6.3 Revisiting the Hypotheses for Regret Minimization

Empirical risk minimization (ERM) is provably optimal, in the limit of infinite training data, if the test data are drawn from the same distribution as the training data, e.g., when training and test data are drawn from the same set of languages. RGM seeks to compensate, during training, for possible

differences between the training languages and the test languages. In more detail, RGM seeks to enforce generalizability by minimizing the differences between cross-lingual and monolingual error rates (termed the "regret").

Three hypotheses were proposed in Section 1.3; this section discusses the status and interpretation of those hypotheses, in light of the experimental results in Section 5.3.

- **H1:** RGM can be used to optimize E2E ASR so that it generalizes from fifteen training languages to five novel test languages more effectively than if it were trained using ERM. **Status:** False.

Hypothesis **H1** is falsified by the experimental results in Table 5.2. The conclusion suggested by this result is that the training data and the test data are drawn from the same distributions. For example, we might (speculatively) conclude that the distribution of speech sounds in these five test languages is reasonably well represented by the set of fifteen training languages. We believe that our choice of output symbols as universal phonetic tokens plays a small part in the result. Further, the language ID may have provided too little information about the environment. For example, it could not have provided much information about the sequence-level phonotactics in the training languages.

- **H2:** RGM, as compared to ERM, can be applied to optimize an E2E ASR so that it more effectively generalizes from training languages in one language family to test languages in a different language family. **Status:** Partially true.

Experimental results in Table 5.3 suggest that hypothesis **H2** is true, but the results in Table 5.6 are somewhat contradictory. In the experiment described in Table 5.3, regret minimization (RGM) is used to minimize the difference between cross-lingual and monolingual error rates of languages in the same family (Slavic). The resulting trained parameters can be applied to languages from other language families (French and German) with better results than the results achieved using ERM. However, in Table 5.6, if we could perform cross-evaluation with another unseen language, the error rates for ERM would improve for all three test languages, while the error rates for RGM would improve for only one. In this case, RGM only performed better than ERM on French but not German.

- **H3:** The optimal training regimen for phone token classification (given known phone token boundary times) is different from the optimal training regimen for phone token recognition (with unknown boundary times). **Status:** True.

Experimental results in Tables 5.3, 5.4, 5.5 and 5.6 suggest that hypothesis **H3** is true. First, note that for the training languages, no matter what the task is, ERM always outperforms RGM. For the three test languages, the recognition error rates shown in Table 5.3 are optimized by ERM if the test language is in the same family as the training languages and by RGM otherwise. The classification error rates in Table 5.4 for the test languages, on the other hand, are optimized using ERM.

Note that the results in Tables 5.3 and 5.4 are obtained by applying early-stopping on the development test data in the three training languages. In Tables 5.5 and 5.6, the early-stopping schedule was governed by a test language rather than by the training languages. Test-based early-stopping improved the performance of RGM in Table 5.5 relative to Table 5.4 for the classification task, and was able to outperform ERM on all three test languages. However, for the recognition task, test-based early-stopping improved the performance of ERM more significantly than RGM, as shown by comparing Tables 5.6 to Table 5.3. We note that the improvement achieved by RGM over ERM in Table 5.5 is indeed significant, as shown by Figures 5.2 and 5.3, which are the confusion matrices calculated on Croatian for the phone token classification experiment.

CHAPTER 7

CONCLUSION

Even for two closely related languages, English and Dutch, a language-independent CTC network still lags behind a language-dependent CTC network. Our study of the hidden representations from the two models suggests that while the consonant hidden representations are well-shared across the two languages, vowel representations aren't and benefit significantly from language-specific tuning. However, designing language-specific tuning layers for a multi-lingual end-to-end system is difficult, especially when different groups of the phoneme inventory react differently to representation sharing.

Therefore, we instead try to develop constraints to guide network training to more suitable local optima. The first constraint is the cycle consistency constraint. We first pre-trained a multi-lingual, multi-speaker TTS system. We fed the TTS system with pseudo phone token sequences during ASR training and constrained the TTS reconstruction to be close to the original speech input. However, results showed that the cycle consistency constraint was not useful in our setting, as the TTS system itself suffered from convergence issues under overwhelming modalities. Surprisingly, simply initializing the representations of the phone tokens in the ASR system with those from the pre-trained TTS system offered a boost in performance.

The second constraint involves minimizing a regret term defined as the difference in risk between an out-of-environment classifier and a within-environment classifier. Both classifiers use the same feature extractor trained on all environments. We hope that minimizing the regret could allow the system to learn an intermediate invariant embedding for seen and unseen languages. Results show that regret minimization (RGM) failed to outperform empirical risk minimization (ERM) in recognition tasks when the training and test languages span across a large number of languages from different families. However, under certain conditions and evaluation schemes, RGM may outperform ERM on the test languages.

REFERENCES

[1] "Domain generalization for language-independent automatic speech recognition," *Frontiers*, in review.

[2] S. Dalmia, R. Sanabria, F. Metze, and A. W. Black, "Sequence-based multi-lingual low resource speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/ICASSP.2018.8461802 pp. 4909–4913.

[3] J. Huang, J. Li, D. Yu, L. Deng, and Y. Gong, "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6639081 pp. 7304–7308.

[4] K. Veselý, M. Karafiát, F. Grézl, M. Janda, and E. Egorova, "The language-independent bottleneck features," in *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012*. IEEE, 2012. [Online]. Available: https://doi.org/10.1109/SLT.2012.6424246 pp. 336–341.

[5] M. Müller, S. Stüker, and A. Waibel, "Language adaptive multilingual CTC speech recognition," in *Speech and Computer - 19th International Conference, SPECOM 2017, Hatfield, UK, September 12-16, 2017, Proceedings*, ser. Lecture Notes in Computer Science, A. Karpov, R. Potapova, and I. Mporas, Eds., vol. 10458. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-66429-3_47 pp. 473–482.

[6] S. Tong, P. N. Garner, and H. Bourlard, "Cross-lingual adaptation of a ctc-based multilingual acoustic model," *Speech Commun.*, vol. 104, pp. 39–46, 2018. [Online]. Available: https://doi.org/10.1016/j.specom.2018.09.001

[7] P.-S. Huang and M. Hasegawa-Johnson, "Cross-dialectal data transferring for gaussian mixture model training in arabic speech recognition," *Constraints*, vol. 1, 01 2012.

[8] X. Li, S. Dalmia, J. Li, M. Lee, P. Littell, J. Yao, A. Anastasopoulos, D. R. Mortensen, G. Neubig, A. W. Black, and F. Metze, "Universal phone recognition with a multilingual allophone system," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ICASSP40776.2020.9054362 pp. 8249–8253.

[9] P. B. Denes and E. N. Pinson, "The speech chain : the physics and biology of spoken language," 1963.

[10] A. Tjandra, S. Sakti, and S. Nakamura, "Machine speech chain," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 28, pp. 976–989, 2020. [Online]. Available: https://doi.org/10.1109/TASLP.2020.2977776

[11] Y. Ren, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T. Liu, "Almost unsupervised text to speech and automatic speech recognition," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/ren19a.html pp. 5410–5419.

[12] J. Xu, X. Tan, Y. Ren, T. Qin, J. Li, S. Zhao, and T. Liu, "Lrspeech: Extremely low-resource speech synthesis and recognition," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403331 pp. 2802–2812.

[13] T. Hori, R. F. Astudillo, T. Hayashi, Y. Zhang, S. Watanabe, and J. L. Roux, "Cycle-consistency training for end-to-end speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. IEEE, 2019. [Online]. Available: https://doi.org/10.1109/ICASSP.2019.8683307 pp. 6271–6275.

[14] M. K. Baskar, S. Watanabe, R. F. Astudillo, T. Hori, L. Burget, and J. Cernocký, "Semi-supervised sequence-to-sequence ASR using unpaired speech and text," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, G. Kubin and Z. Kacic, Eds. ISCA, 2019. [Online]. Available: https://doi.org/10.21437/Interspeech.2019-3167 pp. 3790–3794.

[15] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, ser. ACM International Conference Proceeding Series, W. W. Cohen and A. W. Moore, Eds., vol. 148. ACM, 2006. [Online]. Available: https://doi.org/10.1145/1143844.1143891 pp. 369–376.

[16] K. N. Stevens, "The quantal nature of speech: evidence from articulatory-acoustic data," in *Human communication—a unified view*, P. B. Denes and E. E. David, Eds. New York: McGraw-Hill, 1972, pp. 51–56.

[17] K. N. Stevens, S. J. Keyser, and H. Kawasaki, "Toward a phonetic and phonological theory of redundant features," in *Invariance and Variability in Speech Processes*, J. S. Perkell and D. H. Klatt, Eds. Hillsdale, NJ US: Lawrence Erlbaum Associates, 1986, pp. 426–463.

[18] S. H. Nawab, T. F. Quatieri, and J. S. Lim, "Signal reconstruction from the short-time fourier transform magnitude," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '82, Paris, France, May 3-5, 1982*. IEEE, 1982. [Online]. Available: https://doi.org/10.1109/ICASSP.1982.1171712 pp. 1046–1048.

[19] I. P. Association, "Handbook of the international phonetic association," Cambridge, 1999.

[20] L. Lisker and A. S. Abramson, "A cross-language study of voicing in initial stops: Acoustical measurements," *Word*, vol. 20, no. 3, pp. 384–422, 1964.

[21] V. Patil and P. Rao, "Detection of phonemic aspiration for spoken hindi pronunciation evaluation," *J. Phonetics*, vol. 54, pp. 202–221, 2016. [Online]. Available: https://doi.org/10.1016/j.wocn.2015.11.001

[22] K. Nishi, W. Strange, R. Akahane-Yamada, R. Kubo, and S. A. Trent-Brown, "Acoustic and perceptual similarity of Japanese and American English vowels," vol. 124, no. 1, pp. 576–588, 2008.

[23] S. Shi, C. Shih, and J. Zhang, "Capturing L1 influence on L2 pronunciation by simulating perceptual space using acoustic features," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, G. Kubin and Z. Kacic, Eds. ISCA, 2019. [Online]. Available: https://doi.org/10.21437/Interspeech.2019-3183 pp. 2648–2652.

[24] O. Viikki and K. Laurila, "Cepstral domain segmental feature vector normalization for noise robust speech recognition," *Speech Commun.*, vol. 25, no. 1-3, pp. 133–147, 1998. [Online]. Available: https://doi.org/10.1016/S0167-6393(98)00033-8

[25] E. Eide and H. Gish, "A parametric approach to vocal tract length normalization," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, ICASSP '96, Atlanta, Georgia, USA, May 7-10, 1996*. IEEE Computer Society, 1996. [Online]. Available: https://doi.org/10.1109/ICASSP.1996.541103 pp. 346–348.

[26] M. J. F. Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Comput. Speech Lang.*, vol. 12, no. 2, pp. 75–98, 1998. [Online]. Available: https://doi.org/10.1006/csla.1998.0043

[27] L. Sari, M. Hasegawa-Johnson, and S. Thomas, "Auxiliary networks for joint speaker adaptation and speaker change detection," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 29, pp. 324–333, 2021. [Online]. Available: https://doi.org/10.1109/TASLP.2020.3040626

[28] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision, Kerkyra, Corfu, Greece, September 20-25, 1999*. IEEE Computer Society, 1999. [Online]. Available: https://doi.org/10.1109/ICCV.1999.790410 pp. 1150–1157.

[29] Y. LeCun, "Generalization and network design strategies," Department of Computer Science, University of Toronto, Tech. Rep. CRG-TR-89-4, 1989.

[30] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6639347 pp. 8614–8618.

[31] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," *CoRR*, vol. abs/1907.02893, 2019. [Online]. Available: http://arxiv.org/abs/1907.02893

[32] W. Jin, R. Barzilay, and T. Jaakkola, "Enforcing predictive invariance across structured biomedical domains," *arXiv preprint arXiv:2006.03908*, 2020.

[33] I. Gulrajani and D. Lopez-Paz, "In search of lost domain generalization," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=lQdXeXDoWtI

[34] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, ser. JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014. [Online]. Available: http://proceedings.mlr.press/v32/graves14.html pp. 1764–1772.

[35] A. L. Maas, A. Y. Hannun, D. Jurafsky, and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns," *CoRR*, vol. abs/1408.2873, 2014. [Online]. Available: http://arxiv.org/abs/1408.2873

[36] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, A. Y. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, C. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016. [Online]. Available: http://proceedings.mlr.press/v48/amodei16.html pp. 173–182.

[37] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997. [Online]. Available: https://doi.org/10.1109/78.650093

[38] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[39] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*. IEEE, 2016. [Online]. Available: https://doi.org/10.1109/ICASSP.2016.7472159 pp. 2657–2661.

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html pp. 5998–6008.

[41] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/ICASSP.2018.8462506 pp. 5884–5888.

[42] S. Karita, N. E. Y. Soplin, S. Watanabe, M. Delcroix, A. Ogawa, and T. Nakatani, "Improving transformer-based end-to-end speech recognition with connectionist temporal classification and language model integration," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, G. Kubin and Z. Kacic, Eds. ISCA, 2019. [Online]. Available: https://doi.org/10.21437/Interspeech.2019-1938 pp. 1408–1412.

[43] T. Hori, S. Watanabe, and J. R. Hershey, "Joint ctc/attention decoding for end-to-end speech recognition," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, R. Barzilay and M. Kan, Eds. Association for Computational Linguistics, 2017. [Online]. Available: https://doi.org/10.18653/v1/P17-1048 pp. 518–529.

[44] T. Hori, S. Watanabe, Y. Zhang, and W. Chan, "Advances in joint ctc-attention based end-to-end speech recognition with a deep CNN encoder and RNN-LM," *CoRR*, vol. abs/1706.02737, 2017. [Online]. Available: http://arxiv.org/abs/1706.02737

[45] P. Zelasko, L. Moro-Velázquez, M. Hasegawa-Johnson, O. Scharenborg, and N. Dehak, "That sounds familiar: An analysis of phonetic representations transfer across languages," in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, H. Meng, B. Xu, and T. F. Zheng, Eds. ISCA, 2020. [Online]. Available: https://doi.org/10.21437/Interspeech.2020-2513 pp. 3705–3709.

[46] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html

[47] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*. ISCA, 2016. [Online]. Available: http://www.isca-speech.org/archive/SSW_2016/abstracts/ssw9_DS-4_van_den_Oord.html p. 125.

[48] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS synthesis by conditioning wavenet on MEL spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/ICASSP.2018.8461368 pp. 4779–4783.

[49] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/1068c6e4c8051cfd4e9ea8072e3189e2-Abstract.html pp. 577–585.

[50] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33016706 pp. 6706–6713.

[51] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: A fully end-to-end text-to-speech synthesis model," *CoRR*, vol. abs/1703.10135, 2017. [Online]. Available: http://arxiv.org/abs/1703.10135

[52] D. W. Griffin and J. S. Lim, "Signal estimation from modified short-time fourier transform," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '83, Boston, Massachusetts, USA, April 14-16, 1983*. IEEE, 1983. [Online]. Available: https://doi.org/10.1109/ICASSP.1983.1172092 pp. 804–807.

[53] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/ICASSP.2018.8461375 pp. 5329–5333.

[54] A. H. Liu, T. Tu, H. Lee, and L. Lee, "Towards unsupervised speech recognition and synthesis with quantized speech representation learning," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ICASSP40776.2020.9053571 pp. 7259–7263.

[55] K. Ito and L. Johnson, "The lj speech dataset," https://keithito.com/LJ-Speech-Dataset/, 2017.

[56] V. Vapnik, *Statistical learning theory*. Wiley, 1998.

[57] V. N. Vapnik, "Chervonenkis: On the uniform convergence of relative frequencies of events to their probabilities," 1971.

[58] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford University Press, 1996.

[59] D. E. Bell, "Regret in decision making under uncertainty," *Oper. Res.*, vol. 30, no. 5, pp. 961–981, 1982. [Online]. Available: https://doi.org/10.1287/opre.30.5.961

[60] N. Oostdijk, W. Goedertier, F. V. Eynde, L. Boves, J. Martens, M. Moortgat, and R. H. Baayen, "Experiences from the spoken dutch corpus project," in *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*. European Language Resources Association, 2002. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2002/sumarios/98.htm

[61] D. F. Harwath and J. R. Glass, "Deep multimodal semantic embeddings for speech and images," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*. IEEE, 2015. [Online]. Available: https://doi.org/10.1109/ASRU.2015.7404800 pp. 237–244.

[62] O. Scharenborg, F. Ciannella, S. Palaskar, A. Black, F. Metze, L. Ondel, and M. Hasegawa-Johnson, "Building an asr system for a low-resource language through the adaptation of a high-resource language asr system: Preliminary results," 2017.

[63] C. Best and M. Tyler, *Nonnative and second-language speech perception: Commonalities and complementarities*, 01 2007, pp. 13–34.

[64] J. Flege, "Language contact in bilingualism: Phonetic system interactions," *Laboratory Phonology*, vol. 9, 01 2007.

[65] O. Scharenborg, N. van der Gouw, M. A. Larson, and E. Marchiori, "The representation of speech in deep neural networks," in *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, I. Kompatsiaris, B. Huet, V. Mezaris, C. Gurrin, W. Cheng, and S. Vrochidis, Eds., vol. 11296. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-05716-9_16 pp. 194–205.

[66] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[67] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[68] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[69] S. Bhati, J. Villalba, P. Zelasko, and N. Dehak, "Self-expressing autoencoders for unsupervised spoken term discovery," in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, H. Meng, B. Xu, and T. F. Zheng, Eds. ISCA, 2020. [Online]. Available: https://doi.org/10.21437/Interspeech.2020-3000 pp. 4876–4880.

[70] H. Tachibana, K. Uenoyama, and S. Aihara, "Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/ICASSP.2018.8461829 pp. 4784–4788.

[71] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai, "Espnet: End-to-end speech processing toolkit," in *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, B. Yegnanarayana, Ed. ISCA, 2018. [Online]. Available: https://doi.org/10.21437/Interspeech.2018-1456 pp. 2207–2211.

[72] T. Schultz, N. T. Vu, and T. Schlippe, "Globalphone: A multilingual text & speech database in 20 languages," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6639248 pp. 8126–8130.

[73] M. Hasegawa-Johnson, L. Rolston, C. Goudeseune, G. Levow, and K. Kirchhoff, "Grapheme-to-phoneme transduction for cross-language ASR," in *Statistical Language and Speech Processing - 8th International Conference, SLSP 2020, Cardiff, UK, October 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, L. E. Anke, C. Martín-Vide, and I. Spasic, Eds., vol. 12379. Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-59430-5_1 pp. 3–19.

[74] M. Harper, "The IARPA BABEL multilingual speech database," 2011.

[75] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. B. Orr and K. Müller, Eds. Springer, 1996, vol. 1524, pp. 55–69. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_3

[76] J. Cho, M. K. Baskar, R. Li, M. Wiesner, S. H. R. Mallidi, N. Yalta, M. Karafiát, S. Watanabe, and T. Hori, "Multilingual sequence-to-sequence speech recognition: architecture, transfer learning, and language modeling," *CoRR*, vol. abs/1810.03459, 2018. [Online]. Available: http://arxiv.org/abs/1810.03459