Improving End-to-End Neural Network Models for Low-Resource Automatic Speech Recognition

by

Jennifer Fox Drexler

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
August 28, 2020

Certified by .....
James Glass
Senior Research Scientist
Thesis Supervisor

Accepted by .....
Leslie Kolodziejski
Graduate Officer, EECS

Improving End-to-End Neural Network Models for Low-Resource Automatic Speech Recognition

by

Jennifer Fox Drexler

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

In this thesis, we explore the problem of training end-to-end neural network models for automatic speech recognition (ASR) when limited training data are available. End-to-end models are theoretically well-suited to low-resource languages because they do not rely on expert linguistic resources, but they are difficult to train without large amounts of transcribed speech. This amount of training data is prohibitively expensive to acquire in most of the world's languages.

We present several methods for improving end-to-end neural network-based ASR in low-resource scenarios. First, we explore two methods for creating a shared embedding space for speech and text. In doing so, we learn representations of speech that contain only linguistic content and not, for example, the speaker or noise characteristics in the speech signal. These linguistic-only representations allow the ASR model to generalize better to unseen speech by discouraging the model from learning spurious correlations between the text transcripts and extra-linguistic factors in speech. This shared embedding space also enables semi-supervised training of some parameters of the ASR model with additional text.

Next, we experiment with two techniques for probabilistically segmenting text into subword units during training. We introduce the n-gram maximum likelihood loss, which allows the ASR model to learn an inventory of acoustically-inspired subword units as part of the training process. We show that this technique combines well with the embedding space alignment techniques in the previous section, leading to a 44% relative improvement in word error rate in the lowest resource condition tested.

Thesis Supervisor: James Glass
Title: Senior Research Scientist

Acknowledgments

Thank you to my advisor, Jim Glass, for all of his encouragement, his insightful comments, and the space to find the right topic for me.

Thank you to everyone at MIT Lincoln Laboratory in the Human Language Technology Group and the Lincoln Scholars Program, especially Wade Shen, Joe Campbell, Doug Reynolds, Cliff Weinstein, and Ken Estabrook. My time at Lincoln Lab and the support of the Lincoln Scholars program were invaluable.

Thank you to my parents, grandparents, brother, and extended family for the love and support that got me both to and through graduate school.

Thank you to my husband, Stephen, for everything. It is no exaggeration to say that I could not have done this without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Automatic speech recognition (ASR) has recently become ubiquitous, fundamentally changing how we interact with technology. Advancements in machine learning techniques, particularly deep neural networks, have revolutionized ASR. The rapid improvement in ASR performance in recent years can be attributed both to these new models and to an explosion in available computing power and data. Neural network models can be used for ASR as one component in a hybrid HMM-based system, or as an end-to-end model that transcribes speech directly. Either way, modern ASR systems that can operate effectively in realistic noise conditions require, at minimum, word-level transcripts of thousands to hundreds-of-thousands of hours of speech for training.

End-to-end neural network models for ASR have become the focus of much academic research, and now represent the state-of-the-art in some scenarios (Chiu et al., 2018). These models are simpler to train than traditional ASR models: they comprise a single component trained end-to-end, instead of a series of models of increasing complexity that must be trained in succession (Graves and Jaitly, 2014). End-to-end neural network models can also be effectively trained from transcribed speech without requiring any additional resources. Traditional models, by contrast, are most often trained with a pronunciation dictionary that must be hand-crafted by linguists.

Traditional models can be trained without such a dictionary, but their performance suffers (Wang et al., 2018).

The ability to perform well without expert resources is especially attractive for low-resource languages in which acquiring these resources can be prohibitively expensive (Besacier et al., 2014). The development of ASR technologies in these languages is an important problem with widespread applicability. There are nearly 7000 languages spoken world-wide, but ASR technology is available in fewer than 100. While speech recognition transforms the lives of speakers of the world's most widely spoken languages, many others are being quickly left behind.

Despite the promise of end-to-end neural network models for low-resource languages, continued research has demonstrated a clear trade-off. While they do not require expert resources, these models require even more transcribed speech than traditional models to reach the same level of performance (Chiu et al., 2018; Rosenberg et al., 2017). Unfortunately, the collection of additional training data is expensive, limiting the usefulness of end-to-end neural network models for low-resource speech recognition.

In this thesis, we present several methods for improving end-to-end neural network-based ASR in low-resource scenarios. Broadly, these methods aim to make better use of small training corpora and improve generalization. Some of these methods also enable semi-supervised learning with non-parallel speech and text, which are much easier and cheaper to acquire than speech transcriptions.

1.2 Contributions

The contributions of this thesis are:

1. **A broad framework for semi-supervised training of end-to-end ASR models.** The core components of this framework are a text autoencoder that shares parameters with the ASR decoder and a method for pushing the speech and text encoder outputs to use the same embedding space. This framework allows us to learn representations of speech that contain only linguistic content

and not, for example, the speaker or noise characteristics in the speech signal. Our hypothesis, born out in the results, is that these linguistic-only representations allow the ASR model to generalize better to unseen speech by discouraging the model from learning spurious correlations between the text transcripts and extra-linguistic factors in speech. This model architecture also allows us to train some parameters of the ASR model with text only. This framework was originally introduced in Drexler and Glass (2018).

2. **A method for using adversarial training to create a shared embedding space for speech and text.** Adversarial training is typically used to match a generated distribution to an existing data distribution. Here, we present a modified version of adversarial training that pushes two generated distributions together. We show that this method effectively uses non-parallel speech and text to improve ASR performance, with greater improvements in lower-resource scenarios. We also demonstrate that this method can improve the performance of fully-supervised ASR models when trained with the same speech and text used for ASR training. These results were first published in Drexler and Glass (2018).

3. **An objective function and training procedure for supervised embedding space alignment.** Our adversarial alignment technique considers only individual speech and text embeddings rather than sequences. We develop an alternative technique that aligns the embeddings for full utterances by taking advantage of our ASR model's attention mechanism. This attention mechanism provides an alignment between parallel speech and text utterances. We show that this technique improves ASR performance in low-resource scenarios, although it works better when a slightly larger training corpus is available. This is in contrast to the adversarial alignment technique, which leads to bigger relative improvements on smaller parallel corpora. This work was first published in Drexler and Glass (2019a).

4. **The extension of subword regularization (Kudo, 2018b) to end-to-**

end ASR. Any sequence of one or more characters is called a subword unit. End-to-end neural network models are most commonly character-based, but the use of subword units has recently become quite popular. Here, we are the first to extend the subword regularization technique (Kudo, 2018b), developed for machine translation, to ASR. Our experiments demonstrate that this technique is effective for ASR generally and for low-resource corpora specifically. This work was first published in Drexler and Glass (2019b).

5. **A novel objective function for attention-based ASR that automatically learns an inventory of acoustically-inspired subword units.** We introduce n-gram maximum likelihood loss, a simple modification of the standard maximum likelihood loss. Instead of maximizing the likelihood of a single correct output sequence, we maximize the combined likelihood of all valid character n-grams at each time step. We show that the use of this loss function both improves ASR performance and pushes the model to use an acoustically-inspired subword inventory. We also develop a modified version of the latent sequence decomposition (LSD) framework to use with the n-gram loss which is both more efficient and better performing than the original. Parts of this work were published in Drexler and Glass (2020).

6. **Experiments demonstrating the effectiveness of acoustically-inspired subword units for the construction of a shared speech/text embedding space.** We present experiments for all possible combinations of the above embedding space alignment methods with the above subword segmentation techniques. Both types of subword units improve the construction of the shared embedding space, with the acoustically-inspired subword units working best in both cases. The overall best combined model achieves a 45.4% error rate reduction relative to the baseline in the lowest-resource case tested in this thesis.

1.3 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides the background for this thesis. It includes both an overview of the background necessary to understand the motivation for this work and a survey of prior work that is similarly motivated.

Chapter 3 defines the low-resource problem for end-to-end ASR models. We describe all of the datasets used for this thesis and present baseline results for each. We also provide comparable results from the literature where available.

Chapters 4 through 7 contain the novel ASR training methods developed for this thesis. Chapter 4 introduces our semi-supervised learning framework and adversarial embedding space alignment technique. In Chapter 5, we describe our supervised embedding space alignment technique. In Chapter 6, we apply subword regularization to ASR and analyze its impact on the transcripts produced by our ASR model. Chapter 7 presents the n-gram maximum likelihood loss function and a statistical model for acoustically-inspired subword segmentation of text.

In Chapter 8, we combine the techniques from Chapters 4 through 7, testing both embedding space alignment methods with both subword segmentation methods. While all four combinations work better than the individual techniques alone, we find that the acoustically-inspired subword units are most useful for constructing a shared embedding space for text and speech.

Finally, in Chapter 9, we summarize the contributions and results of this thesis. We also provide several possible avenues for further research.

Chapter 2

Background

2.1 Automatic Speech Recognition (ASR) Overview

Automatic speech recognition (ASR) is the task of automatically producing a sequence of words given an input speech signal. ASR is traditionally a supervised learning problem, meaning that our training data comprises a large number of paired speech and text examples that demonstrate the desired behavior of the trained model. This section provides a broad overview of the field of ASR. First, we give a high-level introduction to the traditional ASR architecture, which is based on Hidden Markov Models (HHMs). We then go into more detail about two different types of end-to-end neural network models for ASR. We focus in particular on the attention-based end-to-end model, which is the architecture used for all experiments in this thesis.

At the highest level, ASR models define the probability distribution $P(text|speech)$. Once the model is trained, we can search for the word sequence that maximizes $P(text|speech)$ for any new speech signal. In this section, we explore the forms that this probability distribution can take and the training methods we can use to learn this distribution from data. All of the models discussed here share one aspect, which is pre-processing of the continuous speech signal into frames of speech features. Some newer models operate directly on the waveform, but that is beyond the scope of this work. Pre-processing is done by taking small segments of the waveform, typically 25ms segments taken every 10ms, and converting that speech segment into a feature

vector.

The inputs to all ASR models used in this thesis are standard features called log-mel filterbank features. To compute these features, we first use a Fourier transform to convert each speech segment into the frequency space, then apply a series of filters. Each feature dimension represents the combined magnitude in a particular filterbank channel. The filters that we use are designed to be equally spaced along the mel-scale, which is based on human perceptions of pitch intervals (Stevens et al., 1937). Many HMM-based ASR systems perform a further step to produce a lower dimensional vector of features called MFCCs. This further step reduces the correlation across features, which is important for traditional models that use Gaussians with diagonal covariances, but is not necessary for neural network models.

2.1.1 ASR With Hidden Markov Models

HMM-based models for ASR were the first ASR models to work well enough for use in real-world systems. While end-to-end models have begun to outperform HMM-based models in some limited scenarios, HMM-based models have been researched and optimized over decades, and have remained the state-of-the-art on many tasks, especially when neural networks incorporated into some components. HMM-based models that include neural networks as components are often called hybrid models.

HMM-based models are designed to mimic the following generative process of speech. The first step in generating a speech utterance is to sample a word sequence. Then we sample a particular pronunciation for each word, such that the sequence of words is now a sequence of sounds. Finally, speech features are generated for each sound. The distribution $P(words|speech)$ can be re-written to reflect the separate components of this generative model:

$$P(words|speech) = P(speech|pronunciation)P(pronunciation|words)P(words)$$

where $P(speech|pronunciation)$ is called the acoustic model, $P(pronunciation|words)$ is the lexicon, and $P(words)$ is the language model.

The language model $P(words)$ is a common type of model that is useful for a wide variety of speech and language tasks. This can be trained from only text data. Rather than directly model the likelihood of an entire sequence of words, language models typically model the likelihood of each word given the previous words in the sentence:

$$P(w_1, w_2, ..., w_m) = P(w_1)P(w_2|w_1)...P(w_m|w_1, ..., w_{m-1})$$

HMM-based ASR models typically use n-gram language models, which make an additional assumption: the likelihood of the current word is dependent on only the previous n words. The choice of n is dependent in part on resource constraints; 3-gram and 4-gram models are most common and work reasonably well for English.

The lexicon, $P(pronunciation|words)$, breaks each word in the model's vocabulary down into a sequence of phonemes, the smallest sound units of language. The lexicon is most commonly crafted by expert linguists, but can also be learned from data. Another option is to use graphemes, or character-based units, instead of phonemes, although this usually results in worse performance (Wang et al., 2018). Most commonly, the sequence of phonemes for an utterance is converted to a sequence of context-dependent phonemes or triphones, which are simply three-phoneme sequences that help the model account for co-articulation effects between nearby sounds.

The acoustic model is the main component of this ASR system that is learned from transcribed speech. The acoustic model is actually a large number of individual HMMs, one for each context-dependent phoneme. These individual HMMs can be used to model an entire speech sequence by concatenating together the HMMs for the sequence of units. HMMs model a sequence of observations as being emitted by a sequence of unobserved latent states. In this case, the observations are the sequence of speech features. The parameters of an HMM are the transition probabilities between states and the emission probabilities for each state. The emission probabilities are the likelihoods of observing particular speech features at each state. A detailed explanation of HMMs is beyond the scope of this thesis; more information can be found in Rabiner (1989).

When HMM-based ASR models were first introduced, the emission probabilities were modeled with Gaussian Mixture Models (GMMs), and these models were referred to as HMM-GMM models. More recently, these have instead been modeled with deep neural networks; ASR models that use neural networks are called hybrid models or HMM-DNN models. The best architectures and training objectives for these neural networks are an active area of research.

A key feature of training an effective HMM-DNN acoustic model is using a recipe that trains a successive series of models with more and more complicated objective functions, using each model to bootstrap the training of the next one. The Kaldi toolkit (Povey et al., 2011) contains recipes for training state-of-the art ASR systems on a wide variety of corpora. The recipe for the Librispeech corpus, which we also use in this thesis, trains five successive acoustic models on a 100 hour subset, followed by another model trained on 460 hours, and then two more models trained on the full 960 hour corpus.

2.1.2 Connectionist Temporal Classification (CTC)

End-to-end models, which comprise a single neural network trained “end-to-end" for a particular task, have become popular for a variety of tasks due to their simplicity. They do not require the development of a complex generative model, as described in the previous section. They can generally be trained in a single step and make very few assumptions about the form of the task and its potential solutions. One particular advantage of using an end-to-end model for ASR is that these models do not require any understanding of the phonetics of a language. Instead, they directly model the process of converting speech input to text output.

Connectionist temporal classification (CTC) was the first method for training end-to-end neural network models for ASR (Graves et al., 2006). The input to this model, as in the previous section, is a sequence of speech feature frames. The output is a sequence of text units; the most common choice is characters but the model can be used with words or subword units with no modifications.

CTC models comprise a single recurrent network (often with many layers) that

produces one output for each input time-step. At the top of this network is a softmax layer, which converts its inputs to a probability distribution over the vocabulary. In this section, we will assume that the vocabulary is the set of Latin characters, plus a 'blank' unit, which is required for CTC and which will be explained shortly. We will represent the blank unit with an underscore in this section.

After we feed a sequence of speech features through the CTC model, the output is one probability distribution per input time-step. We will call the probability of observing label k at time t y_t^k. We define a path π as a sequence of labels of length T, where T is the length of the input and output sequences. The probability of path π according to the CTC model is $p(\pi|x) = \sum_{t=1}^{T} y_t^{\pi_t}$, or the total probability of observing the labels that make up π.

The key feature of CTC is how it handles the vast majority of cases where the desired transcript is not the same length as the input sequence. First, CTC makes an assumption about the ASR task: that the length of the desired output is less than or equal to the length of the input sequence. This assumption is a reasonable one for ASR when there are, as is most common, tens or hundreds of speech feature frames per second.

Second, CTC is premised on a simple function that maps the sequence of outputs to a 'labeling', which is the actual transcript of the input speech. This function handles cases in which the desired transcript is shorter than the input sequence. Consider a simple training example: a sequence of 4 speech feature frames and the word CAT. In CTC, there are several different ways that the model could take in these 4 frames and correctly produce the labeling CAT. They are: CAT_, CA_T, C_AT, _CAT, CATT, CAAT, and CCAT. The mapping removes all blanks and repeated labels so that each of these output sequences map to the same labeling, CAT.

CTC models are trained with the CTC loss function, which maximizes the likelihood of the desired labeling by marginalizing over every possible output sequence that condenses into that labeling. This is possible in part because of the way CTC models are structured: the ouputs are conditionally independent given the input. Given an input x and ground-truth labeling y, the CTC objective function which we want to

maximize is:

$$O(x, Y) = \sum_{\pi \in Y} p(\pi|x)$$

where Y is the set of paths that condense to the labeling y. Both this objective function and the best labeling for a new input can be computed using a variant of the forward-backward algorithm specifically designed for CTC (Graves et al., 2006).

2.1.3 Attention-Based End-to-End ASR

Attention-based models differ from CTC-based models in how they handle the difference between the input and output lengths. The attention-based model architecture is also called an encoder-decoder model with attention. While the parameters are all trained end-to-end with a single objective function, the model can be thought of as three components: an encoder network, a decoder network, and an attention mechanism. Attention-based models were originally developed for machine translation (Bahdanau et al., 2016; Chorowski et al., 2015) and make fewer assumptions than CTC about the ASR task. The attention-based model architecture used throughout this thesis is based on the Listen, Attend, and Spell (LAS) model (Chan, Jaitly, et al., 2016). This model architecture is shown in Figure 2-1. We will refer here to the LAS "listener" component as the encoder, and the LAS "speller" component as the decoder.

The encoder network is typically a bi-directional recurrent neural network (RNN) that takes the speech features as input and outputs a series of hidden states whose length is the input length down-sampled by some multiplicative factor. This down-sampling happens through the use of pyramidal layers, in which each unit receives input from some number of consecutive units in the previous layer. Figure 2-1 illustrates two pyramidal layers that each downsample their input by a factor of 2, so that the encoder network downsamples its input by a total factor of 4.

The decoder network is a recurrent neural network that functions similarly to an RNN language model, but conditioned on the input speech. At each time-step, the

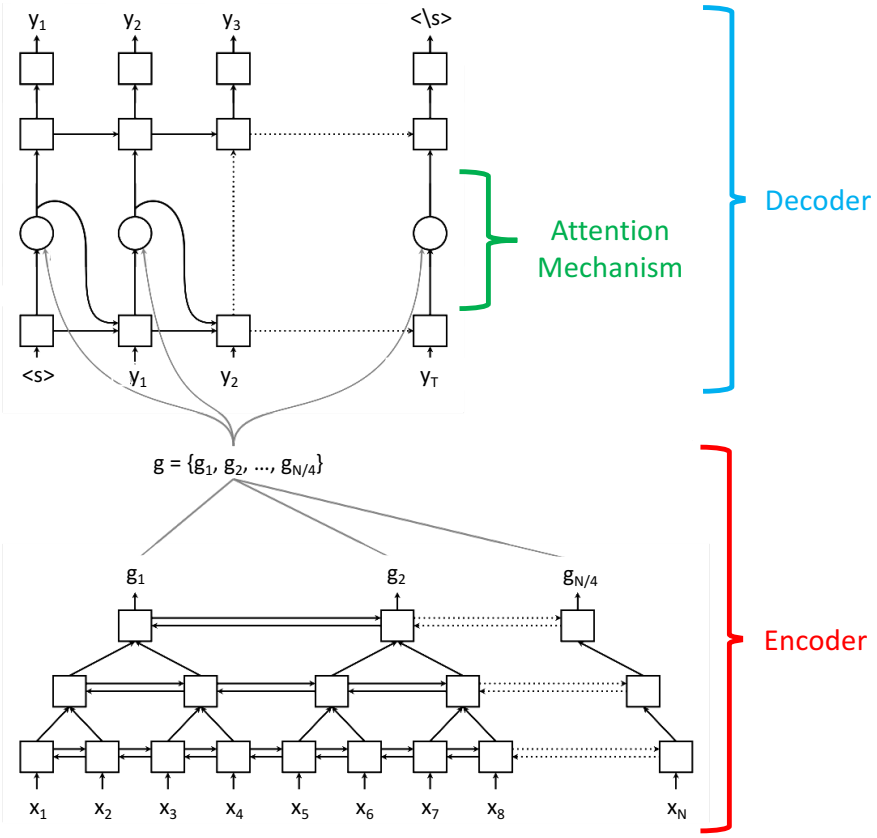


Figure 2-1: Listen, Attend, and Spell (LAS) model architecture; figure adapted from Chan, Jaitly, et al. (2016). In this work, we refer to the LAS "listener" component as the encoder, and the LAS "speller" component as the decoder.

decoder receives two inputs. The first is the previous unit in the text sequence, fed through an embedding layer so that it is converted to a continuous vector space. During training, the decoder receives the ground truth text as input. The decoder outputs a probability distribution over the set of output targets.

The decoder also receives input from the attention mechanism. The attention mechanism is a neural network that, at each decoding time-step, produces a weighted combination of the encoder outputs, based on the decoder input at that time-step and the attention output from the previous time-step. The attention mechanism tells the decoder which speech frames to focus on at each time-step. This allows the decoder to operate without making any assumptions about the relative lengths of the input and output sequences.

We use an attention mechanism that takes as input the sequence of encoder outputs, g, and the current state of the first decoder layer. These inputs are fed through a feed-forward layer to produce a set of attention weights which are used to create the attention context vector. The context vector is a weighted linear combination of the encoder outputs. Both the context vector and the first layer decoder state are fed through another feed-forward layer, then to the next decoder layer.

The model is trained end-to-end to maximize the likelihood of the ground truth text labels. More formally, each training example contains a speech utterance, x, and text label sequence, y, where $y = y_1, ..., y_T$, with T being the length of the label sequence. The training loss is:

$$L(x, y) = -\sum_{t=1}^{T} log(p_t(y_t|x, y_{1:t-1}))$$

where p_t is the decoder output distribution at time-step t.

2.1.4 Inference in ASR Models

"Inference" refers to the process of searching for the best transcription of a new speech utterance given a trained model. With HMM-based models, this process is split into two steps: decoding and rescoring. For end-to-end models, we generally only do the decoding step and thus use the word decoding interchangeably with inference.

With a model of the form $p(words|speech)$, we are searching for the sequence of words that maximizes this probability. For most ASR applications, the search space is far too large to do an exhaustive search of all possible word sequences. For all three model types discussed here, decoding is done with variants of the beam search algorithm (Sutskever et al., 2014). Broadly, the idea behind beam search is to do a breadth-first search in which we choose a beam size, b, and prune the search space after each time-step so that we are only exploring at most b paths.

For HMM-based models, decoding is done through a weighted finite state transducer (WFST) created by composing together the language model, lexicon, and acoustic model into a single graph. This graph is essentially a large HMM in which the

HMM states learned by the acoustic model have been mapped to words, so that the most likely path through the HMM corresponds to the most likely word sequence according to the combined model components. Viterbi search is an efficient algorithm for finding the best path through an HMM, but ASR decoding graphs are typically too large to do full Viterbi search. Beam search can be used to limit the search space and make decoding more efficient. For more details on WFSTs and decoding in HMM-based ASR models, see Mohri et al. (2002).

For CTC, we run the full neural network model to get a sequence of output distributions and then use beam search to explore the paths through those distributions. The CTC beam search algorithm looks not for the most likely sequence of outputs but for the most likely labeling, which means summing over many possible ways of producing the same final transcript. The beam that we maintain here is over b labelings, not output sequences.

For attention-based models, beam search with beam size b means feeding the full speech signal through the encoder and then maintaining b copies of the decoder, each with a different internal state. Attention-based ASR models are thus limited to a relatively small beam compared to other model architectures. It also makes the search process complicated to implement, because we frequently need to make copies of the decoder state if two different likely hypotheses branch off from the same path.

As mentioned previously, the outputs of a CTC model are conditionally independent given the input. This means that a CTC model is not doing any sort of language modeling as part of the ASR process. Attention-based models, meanwhile, perform some language modeling at the level of the output units, but often have trouble capturing the long-range dependencies necessary to do language modeling at the word level. Thus, in both cases, it can be beneficial to explicitly incorporate a word-level language model into the decoding process.

To use a word-level n-gram language model in end-to-end ASR decoding, we first use FST composition to convert the word-level language model to a character-level one as in Bahdanau et al. (2016). We need to do this so that we can consider the likelihood according to the language model every time we add a single character. For example,

the language model likelihood of choosing the letter T at the beginning of an utterance should be the combined likelihood of starting an utterance with any word that starts with T. Once we have correctly formatted our language model, it is straightforward to take the hypotheses in the beam and follow the same paths through the language model. This language model integration has two hyperparameters: a language model weight and a word bonus. We use the language model weight to find the combined score for a hypothesis according to both the ASR model and the language model (the ASR model likelihood always has a weight of 1). The word bonus is used to encourage the model to transcribe the full utterance when the language model is used; without this bonus, the language model can lead to truncated transcriptions because longer sentences are less likely than shorter ones.

2.2 Low-Resource Machine Learning

There have been many recent advances in machine learning techniques, mostly devoted to larger and larger models trained with larger and larger corpora. These state-of-the art techniques do not perform nearly as well when limited resources are available.

While ASR is the focus of this thesis, the low-resource problem has been explored for a variety of machine learning tasks. In this section, we will look at how others have approached the problem of data scarcity in neural network training, both for ASR and for other tasks. The work discussed here has broadly similar motivation to this thesis, whether or not the techniques are similar to those used here. All subsequent chapters will also have a “related work" section - we use those sections to discuss in more detail the specific techniques that are most relevant to the technical content of each chapter.

This section covers three topics. The first part focuses on explicit regularization of neural networks through regularization terms, dropout, and data augmentation. The next part looks at semi-supervised learning, or training with some labeled and some unlabeled data. In the case of ASR, semi-supervised learning means training with

untranscribed speech, stand-alone text, or both. Finally, the third part covers a range of other techniques that have been used specifically for low-resource ASR. These techniques all broadly aim to bring in more training data, performing either multilingual or multi-task learning with additional corpora. It is worth noting that many of the works discussed here could be placed in multiple sections, as the boundaries between the categories of low-resource training techniques are fuzzy. In particular, most of the work discussed in this chapter could be broadly categorized under the rubric of regularization.

2.2.1 Regularization in Neural Networks

Neural networks are extremely powerful learning machines. Depending on the model architecture and the training corpus, neural networks are sometimes so powerful that they memorize the training data rather than finding patterns in it. When a machine learning model performs well on the training data but does not generalize well to new data, this is called "overfitting." Regularization is an important part of neural network design and training that can greatly reduce this overfitting problem.

Regularization is often thought of specifically in the context of regularization terms, which are additional terms added to the training objective function that push the model to favor particular types of solutions. Regularization terms can also be thought of in a Bayesian sense as imposing a prior over the model parameters. Two classic regularization terms that can be used with a variety of machine learning models are L_1 and L_2 regularization. L_1 regularization terms penalize the sum of the model weights; this encourages sparse solutions with limited redundancy. L_2 regularization penalizes the sum of the squared model weights; by contrast, this encourages all weights to be as small as possible and encourages computations to be spread across multiple weights.

In this section, we will also consider dropout, a different way of achieving regularization in neural networks specifically. Dropout is a very simple technique in which some percentage of the model weights are randomly set to zero on each training iteration. This method is similar to L_2 regularization in that it encourages redundancy

and dense weight matrices.

Lastly, we will discuss data augmentation techniques. Some data augmentation techniques are very similar to dropout: rather than randomly setting weights to zero, they randomly set some parts of the speech input to zero. Others, like varying the speed of the input speech or adding noise, encourage the model to learn representations that are invariant to these factors in audio.

Regularization Terms

As mentioned above, the most common way of doing regularization is to add an explicit regularization term to the task loss function. This type of regularization is a long-standing technique that is applicable not just to neural networks but to most types of machine learning models.

Within the specific field of neural network models, regularization terms have been applied to a wide range of problems and model architectures (Dieng et al., 2018; Hu Liu et al., 2018; Ochiai et al., 2017). Here, we highlight two techniques that have been applied successfully to neural networks used for ASR.

C. Wu et al. (2017) introduce a method they call activation regularization, which imposes particular patterns on the activations of neurons in each layer of a neural network. This method enables them to use their knowledge of the task and the training data to design a custom regularization term. They test their method on a DNN acoustic model, using a map designed to have similar phonemes represented by neurons that are nearby in the network. They show that their technique outperforms both L_2 regularization and dropout on both the SI84 subset of the WSJ corpus and several Babel languages. Their analysis also demonstrates that this type of regularization can be used to improve the interpretability of neural network parameters.

Label smoothing (Szegedy et al., 2016) is a common technique to avoid overfitting in neural networks. Label smoothing involves a modification to the loss function; instead of asking the model to output the correct answer with 100% probability, the model is trained to output the correct answer with slightly less confidence, say 90%. This can help the model avoid memorization and thus improve generalization. Pereyra

et al. (2017) show that a similar effect can be achieved with a regularization term that imposes a penalty on overly confident output distributions. They demonstrate the effectiveness of this technique on a number of tasks in different domains, including ASR on the WSJ corpus.

Dropout

Dropout is a simple but effective technique. In the paper that first introduced dropout, Srivastava et al. (2014) suggest that many of the learned dependencies between neurons in a neural network are spurious correlations that exist in the training data but not in the larger data distribution, especially if the training set is small. They argue that if the neurons can be encouraged to represent meaningful concepts on their own, without over-reliance on other neurons, it will make the entire network more robust and reduce overfitting. Another way of thinking about dropout is that it allows a single neural network to mimic an ensemble of neural networks by training a random subset of the network on each iteration, and then essentially averaging the outputs of those subsets at test time. Srivastava et al. (2014) demonstrate the effectiveness of this method, producing, at the time, state-of-the art results in a range of tasks in computer vision, speech recognition, and computational biology. More recently, a number of researchers have devoted time to studying the best ways to apply dropout in recurrent neural networks (Gal and Ghahramani, 2016; Merity et al., 2017; Zaremba et al., 2014). The improvements that have resulted from this work are ones that we take advantage of; we use dropout in every model trained for this thesis, in both recurrent and feed-forward layers.

Zhou et al. (2017) provide an overview of the impacts of both dropout and data augmentation (discussed in the next section) on end-to-end neural network models for ASR. They show WER improvements of more than 20% relative on both the WSJ and Librispeech datasets from dropout alone.

Data Augmentation

Data augmentation refers to the process of perturbing the inputs to a machine learning model so as to create more variance in the data the model is exposed to. It can be viewed as a regularization method that pushes the model to be invariant to the dimensions along which the data are perturbed. In the case of speech recognition, a range of augmentation strategies have been employed, both for acoustic model and end-to-end model training.

In the case of data augmentation for acoustic model training, augmentation strategies that have been shown to work well include: the addition of noise, speed perturbation, and vocal track length perturbation (Cui et al., 2015; Jaitly and Hinton, 2013; Ko et al., 2015).

Hannun et al. (2014) was one of the first papers to demonstrate the ability of end-to-end models to compete with hybrid models on large ASR tasks. This model was trained on a corpus of more than 5000 hours of collected audio, augmented by the addition of recorded noise. Zhou et al. (2017) experiment with the addition of white noise, as well as pitch, tempo, and volume modulation. They demonstrate a 6% relative improvement over a strong baseline on WSJ from just noise augmentation, and a 21% relative improvement from all augmentation methods combined. Their results suggest, however, that these gains are not additive with the gains from regularization achieved through dropout; the combination of dropout and data augmentation produced similar gains to either dropout or data augmentation on its own.

SpecAugment (Park et al., 2019) is a simple data augmentation method that has produced state of the art results on the Librispeech and Switchboard corpora. SpecAugment operates on the input speech features to the neural network and includes several methods: time-warping, time-masking, and frequency-masking. While the best performing models used all three methods, time-warping contributed minimally to the results while also being more expensive to implement than the other two. In the process of generating these results, the authors found that using SpecAugment led to underfitting, but this issue was solved by using larger models and longer

training times.

2.2.2 Semi-Supervised Learning

Semi-supervised learning describes any method that trains with unlabeled data in addition to labeled data. In the context of ASR, this can mean either untranscribed speech or stand-alone text. Almost every ASR system uses semi-supervised learning in that it incorporates a language model trained on a large text-only corpus. In this section, we focus on work that uses unlabeled data to train the parameters of a neural network that is used for a supervised task. In the ASR context, this means either the acoustic model component of a hybrid DNN-HMM system or an end-to-end neural network model.

Pseudo-Labeling

Pseudo-labeling is simple technique for generating additional supervised training data from unlabeled data (Lee, 2013). In the ASR context, pseudo-labeling means automatically generating either text labels for untranscribed speech or speech features to go with stand-alone text, then using those new paired speech/text examples as part of ASR model training. Pseudo-labeling is most often used to label additional output data, as bad pseudo-outputs can hurt model performance rather than help. For example, if we pseudo-label speech and the generated text labels are un-grammatical, then the model will learn to produce un-grammatical output.

Labeling additonal output data requires training two models: one to perform the desired task and one in the reverse direction. This is a common technique in machine translation, where the necessary model architecture is the same in both directions. Replicating this idea for ASR is a bit more complicated: it means training a text-to-speech, or TTS, model, which is an entirely separate research area.

Up until very recently, it has been difficult to generate high-quality speech, especially if the training corpus is small. The performance of TTS models has been progressing rapidly thanks to similar developments as those seen in ASR. Rossen-

bach et al. (2020) use a very recent, state-of-the-art TTS model to generate synthetic spectrogram features from text.

Hayashi et al. (2018) introduce the concept of a text-to-encoder, or TTE, model used in place of a TTS model. The authors first train an attention-based ASR model, then use this model to get the encoder outputs for all of the training speech. The TTE model is trained using (text, encoder) pairs created from the original (text, speech) training data. Once the TTE model is trained, it can be used to generate encoder sequences from new text. These encoder sequences can be used along with the text for additional training of the decoder parameters in the ASR model. This method avoids the difficulty of generating realistic speech features as well as the possibility of training the encoder parameters on un-realistic speech.

It is also possible to train an ASR with pseudo-labels generated from speech by the model itself, but it requires care to not exacerbate the model's errors when doing this. The TTE model described above is used in Hori et al. (2019) to score pseudo-labels for untranscribed speech generated by the ASR model, using the notion of cycle-consistency. The idea is that the ASR model should generate transcripts of untranscribed speech that, when encoded with the TTE model, produce encodings similar to those of the original speech. The cycle-consistency loss is calculated between the speech encodings and the TTE encodings of several hypothesized transcripts from the ASR model. The best pseudo-label sequence is chosen not according to the ASR model likelihood but according to this cycle-consistency loss.

Kahn et al. (2020) also use pseudo-labels generated by the ASR model itself, but takes several steps to ensure that these labels are high-quality. First, they decode with an external language model when producing these labels, ensuring that the labels are better than what the end-to-end neural network model by itself is able to produce. Second, they develop a set of heuristic filters to remove pseudo-labels with errors common to end-to-end models - looping and early stopping. Finally, they use an ensembling method in which they train five models from different initializations and sample each set of pseudo-labels from a randomly selected model.

Alternative Objective Functions

In the previous section, pseudo-labeling was used to turn data in one domain (speech or text) into paired speech/text data that could be used to train the ASR model with the same maximum-likelihood loss function used for baseline training. In this section, we will look at alternative objective functions that can be used to train the model parameters based only on data from one modality, or on non-parallel speech and text data.

Baskar et al. (2019) use the notion of cycle-consistency to create new loss functions for training the model parameters with both untranscribed speech and additional text. Their framework consists of both an ASR model and a TTS model. Both are originally trained with a small corpus of parallel speech and text data. They feed untranscribed speech first through the ASR model, then feed that output through the TTS model. Cycle-consistency states that this should produce output that looks similar to the original speech. Similarly, when text is fed through the TTS model and then the ASR model, the output should be similar to the original text. The key innovation in this paper is to modify the models so that an end-to-end differentiable loss can be computed through both models at once. This essentially turns the combined models into two autoencoders that can be trained, respectively, with just speech and just text.

Liu et al. (2019) make use of adversarial training, a powerful technique that has been used most notably in computer vision to generate highly realistic fake images and videos. Adversarial training consists of two models: a discriminator and a generator. The generator is the model being trained to generate a particular kind of data, such as realistic images. In this case, the generator is the ASR model that is being trained to generate text from speech. The discriminator is a classifier that is trained to distinguish between true examples (here, text from a large text corpus) and examples produced by the generator. These models are trained iteratively: the discriminator is trained to classify the two types of examples, the generator is trained to 'trick' the discriminator, then the discriminator is re-trained, and so on. In this paper, the

discriminator is what the authors call a 'criticizing language model'. The authors run experiments using the 100 hour clean librispeech subset as paired speech/text data along with the text from either the remaining 360 hours of clean data or the remaining 860 hours from the whole corpus. They show that, in both cases, this technique is better than pseudo-labeling the text with a TTS model.

Hsu et al. (2020) use a language model as a way to score the linguistic plausibility of different potential transcripts of untranscribed speech. Rather than use these potential transcripts as pseudo-labels for further training, the authors introduce a new objective function, called local prior matching, that encourages the probability distribution over hypotheses in the beam to be proportional to the probabilities assigned by the language model. In this way, they use the language model as a prior over the ASR outputs. In addition to impressive results using a large language model as the prior, this paper demonstrates that this technique can be effective even when not much text is available for language model training.

Unsupervised Pre-Training

Unsupervised pre-training has been very successful in computer vision for many years (Erhan et al., 2010). More recently, several successful techniques (e.g. BERT (Devlin et al., 2018), ELMO (Peters et al., 2018)) have been introduced for building high-quality text representations from unlabeled text, then doing supervised fine-tuning for a particular task on top of those representations. These pre-training methods have produced state-of-the-art results on a number of NLP tasks (Devlin et al., 2018).

In a very recent work, unsupervised pre-training for speech recognition has begun to show the huge gains associated with unsupervised pre-training in other domains. The architecture for wav2vec 2.0 (Baevski et al., 2020) comprises a convolutional neural network encoder and a transformer network. The model encodes speech, then parts of those encodings are masked. The model is trained to distinguish between the true values of the masked encoding and several distractor candidates at each masked time-step. Using this technique, the authors demonstrate both competitive performance on the full Librispeech corpus with a much simpler ASR architecture and

very impressive semi-supervised learning results. They are able to achieve comparable results to a hybrid model trained on 100 hours of Librispeech data using only 10 minutes of labeled data. They are also able to outperform the previous state-of-the art on Librispeech with 100 hours labeled and 860 hours unlabeled using only 1 hour of the labeled data.

2.2.3 Other Approaches to Low-Resource ASR

Other work that has focused on ASR for low-resource languages has generally looked at finding additional data for training. This work differs from semi-supervised learning in that the extra data is not stand-alone speech or text but labeled data that can be used to train all or part of the model in a supervised fashion. A popular method is to train a multilingual ASR model using data from many languages, on the theory that languages have much in common and learning about one language can help with another. An alternative is to use the ASR training data to train the model to perform multiple tasks at once.

Multilingual Training

The motivation behind multilingual training is most clear when considering acoustic model training for hybrid HMM-DNN ASR systems. For these systems, it is possible to develop a shared phone set so that an acoustic model can be trained with many different languages using the same output targets (Lin et al., 2009). After the success of this type of multilingual training, researchers have also considered multilingual acoustic models that share everything except the output targets (Heigold et al., 2013; Karafiát et al., 2017). This type of training rests on the theory that low-level speech features will be shared among languages, even if the phone sets diverge.

The first models to use multilingual training for end-to-end models were incremental models that proceeded in several stages (Rosenberg et al., 2017). First, a multilingual acoustic model is trained using one of the two techniques described above. Features are extracted from a 'bottleneck' layer in this acoustic model for

every training utterance. A bottleneck layer is a layer with fewer units than the rest of the layers in the network, and is typically the second-to-last layer when used as a feature extractor. An end-to-end ASR model is then trained on the language of interest using these features as input.

More recently - as in Cho et al. (n.d.), Dalmia et al. (2018), and Kannan et al. (2019) - researchers have begun to train end-to-end models with multilingual corpora. Of particular note is Pratap et al. (2020), in which the authors trained a single model on data from 51 languages, with between 100 and 1000 hours of training data from each. With their best model architecture, they achieved an average 28.8% relative WER reduction across the languages tested compared to the monolingual baselines.

Multi-Task Training

.

Multi-task training has become popular for training low-resource DNN acoustic models. In that case, the typical output targets are triphones, but many other output targets are available within the same data because we have access to both a sequence of character labels and a sequence of phonetic labels. As in multilingual training, the idea is to share the entire model except for the output layer. Chen et al. (2014) do multi-task training by using both triphones and trigraphemes as output targets. In Chen and Mak (2015), the same authors combine that technique with multilingual training, so that there are two tasks for each language used. Bell and Renals (2015) use a similar approach, but with triphones and monophones.

A popular form of multi-task learning for end-to-end ASR is the joint CTC/attention model (Kim et al., 2017). In this model, the CTC loss is computed from a softmax layer directly on top of the attention-based encoder. The CTC loss is considered an auxiliary loss in this case; the attention-based decoder is ultimately used for inference. In Kim et al. (2017), this joint model performs better than either a CTC model or an attention-based model on both the WSJ SI284 set (81 hours) and the SI84 set (15 hours).

Gowda et al. (2019) extend the joint CTC/attention framework to also include

subword sequences produced with byte pair encoding (we will discuss subword sequences in more detail in Chapters 6 and 7). Instead of two losses, as in the above model, this work has four: one loss of each type for characters and one of each for subwords. Tested on the Librispeech corpus, this model improves the on the performance of the joint CTC/attention model.

Chapter 3

The Low-Resource Problem in Automatic Speech Recognition

3.1 Data and Evaluation

When we experiment with different ASR models, we divide each of our corpora into three parts: training data, validation data, and test data. The training data is the bulk of each corpus; this is the data that is used to teach the model how to perform ASR. The validation and test sets are small subsets of the original corpus. The validation data is used to determine when to stop training: we score the performance of the model after each training epoch and choose the model parameters that perform best on the validation set. All reported scores in this thesis use a set of test utterances that were not used for either training or validation.

Throughout this thesis, we will judge the performance of an ASR model by measuring the word error rate (WER) of its outputs compared to the ground-truth transcripts. To calculate WER, we first align the ASR output to the reference, then count up all of the errors in the output. These errors fall into three categories: substitutions, insertions, and deletions. The WER is the total number of errors divided by the number of words in the reference. Thus, if the output transcript is longer than the reference, it is possible to have a WER greater than 100%. While WER is the most common metric for ASR, we also frequently report character error rate (CER),

which is calculated the same way as WER but on the individual character level.

The experiments in this thesis use datasets from two categories: well-studied English benchmark datasets and datasets in true low-resource languages. The English datasets allow a full exploration of the low-resource problem: we can look at how each proposed technique performs across a range of settings by starting with the full corpus and incrementally reducing the amount of data used for training. The low-resource corpora then allow us to assess how the most promising of these ideas generalize across languages and what their usefulness might be in real-world low-resource applications.

Corpus	Speakers	# Utterances	Train Hours
WSJ SI284	284	37416	81
WSJ SI84	84	7138	15
WSJ 40	284	18708	40
WSJ 20	284	9354	20
WSJ 10	284	4677	10
WSJ 5	284	2338	5
WSJ 2.5	284	1169	2.5
Librispeech clean460	1172	132548	460
Librispeech clean100	251	28539	100
Librispeech clean20	251	5708	20

Table 3.1: Details on the two English copora, Wall Street Journal (WSJ) and Librispeech, used in this thesis. The WSJ SI284, WSJ SI84, Librispeech clean460, and Librispeech clean100 sets are standard training corpora used frequently for ASR research. The remaining subsets detailed here were created for this thesis to mimic low-resource scenarios.

We use two English datasets: the Wall Street Journal (WSJ) corpus (LDC, 1994) and the LibriSpeech corpus (Panayotov et al., 2015). The full WSJ corpus consists of 81 hours of read speech; each utterance is a single sentence read from the Wall Street Journal newspaper. This training corpus includes 37.4K utterances, read by 284 speakers. In all WSJ experiments, we use the standard dev93 and eval92 sets for validation and test, respectively. For results that incorporate a language model, we use the text corpus that accompanies the WSJ speech recognition corpus. To simulate lower-resource conditions, we divide the training data by powers of two, selecting the utterances so that all 284 speakers are represented in all training corpora. In particular, we focus on three low-resource conditions - $1/8^{\text{th}}$, $1/16^{\text{th}}$, and $1/32^{\text{nd}}$

of the original corpus, or approximately 2.5, 5, and 10 hours of transcribed speech, respectively.

The LibriSpeech corpus is a much larger corpus - 960 hours - encompassing both clean and noisy speech, read from books in the public domain. This comprises 460 hours of clean speech and 500 hours of noisy speech; there is also a standard 100 hour clean training set. We use the 460 hour set as our topline and the 100 hour training set as one low-resource condition. We also create our own sets of approximately 10, 20 and 50 hours by selecting 10, 20, or 50 percent of the 100-hour utterances, respectively. We again ensure that these subsets contain all speakers from the 100 hour set. The corpus includes standard clean and "other" validation and test sets; all results in this thesis use the clean validation and test sets unless otherwise indicated.

For both WSJ and Librispeech, we explore semi-supervised learning by using the audio we do not include in our low-resource training corpus as untranscribed speech. We also make use of the available language model training text as part of ASR training in some cases. More details on the sizes of the WSJ and Librispeech corpora and the subsets we created to emulate low-resource conditions can be found in Table 3.1.

The true low-resource corpora that are included in this thesis come from the Babel program (Harper, 2014). This program was an effort by IARPA to collect small ASR corpora in a range of low-resource languages and explore the possibilities for rapid development of ASR capabilities in these languages. The audio quality of these corpora tends to be much lower than for WSJ and Librispeech (where we largely focus on the 'clean' portion of the corpus). These recordings also largely capture conversational speech, which is well-known to be harder to recognize than read speech.

In this chapter, we present results for four languages: Dholuo, Igbo, Javanese, and Swahili. These languages were chosen for this work for two reasons. First, these languages use the Latin alphabet, which allows us to do qualitative analysis of the ASR models we train. Second, we were able to find published results on these languages using end-to-end models. The training corpora that we use for the Babel languages contain approximately 40 hours of transcribed speech per language.

3.2 Model Details

For our baseline attention-based model, we use long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997) for all recurrent layers in the encoder and the decoder. Our encoder has one standard bidirectional LSTM layer, and 3 pyramidal bidirectional LSTM layers that each downsample their input by a factor of 2. The total downsampling factor of the encoder is 8. The decoder contains two unidirectional LSTM layers. For all models, we use the same number of units in every hidden layer of both the encoder and the decoder.

In this chapter, we experiment with a few standard hyperparameters of the model to find the best baseline architecture for each corpus to be used in this thesis. First, we experiment with the size of the hidden layers. Next, we experiment with dropout; the amount of dropout is a single hyperparameter that applies to all layers of our model except for the text embedding layers. Finally, we experiment with scheduled sampling. This hyperparameter governs how often we sample the next decoder input from the decoder output distribution, instead of using the ground-truth. Scheduled sampling exposes the decoder to the types of mistakes it is likely to see during inference, allowing the model to learn to recover from those mistakes.

3.3 Baseline Results

In this section, we present results for the baseline attention-based models used as the starting point for all work in this thesis. Most results here are from the best model settings found for each corpus described above. These results demonstrate first the competitiveness of our model on the standard corpora that we work with here, and second the low-resource problem with end-to-end models. This section also provides example output that gives a qualitative understanding of the meaning of WER and discusses the settings that we explored in order to achieve the best performance on each corpus.

3.3.1 Standard English Corpora

Corpus	Model	WER	CER
WSJ SI284	this work	16.7	6.0
	Bahdanau et al. (2016)	18.6	6.4
WSJ SI84	this work	36.5	15.0
	Kim et al. (2017)		17.0
	Karita et al. (2018)		15.8
Librispeech clean460	this work	12.7	5.7
	Hori, Astudillo, et al. (2019)	11.8	4.6
Librispeech clean100	this work	23.3	11.2
	Hori, Watanabe, et al. (2017)	25.2	11.1

Table 3.2: Baseline results on the standard English corpora used in this thesis, with comparison to other published work.

Our baseline results for the standard subsets of the WSJ and Librispeech corpora, along with a selection of published results, are in Table 3.2. We have specifically chosen published results for model architectures that are similar to ours; this table demonstrates that our model implementation is in line with those used elsewhere in the field. For all four subsets in this table, other model architectures have been shown to outperform those listed here. In particular, HMM-DNN models still consistently outperform end-to-end models on all of these tasks. New end-to-end architectures like transformer models, which are beyond the scope of this thesis, have also become popular in the time since the majority of the research for this thesis was done.

3.3.2 Low-Resource English Subsets

Figure 3-1 clearly illustrates the low-resource problem in end-to-end ASR. It shows WER on the y-axis, as a function of the number of hours of training data on the x-axis. The two lines represent the two standard English corpora used here, WSJ and Librispeech. As we decrease the amount of training data, performance degrades drastically, to the point where the model has learned virtually nothing about performing ASR when trained with only 2.5 hours of WSJ data. It is also worth noting that Librispeech is a much harder corpus than WSJ for the same amount of training data.

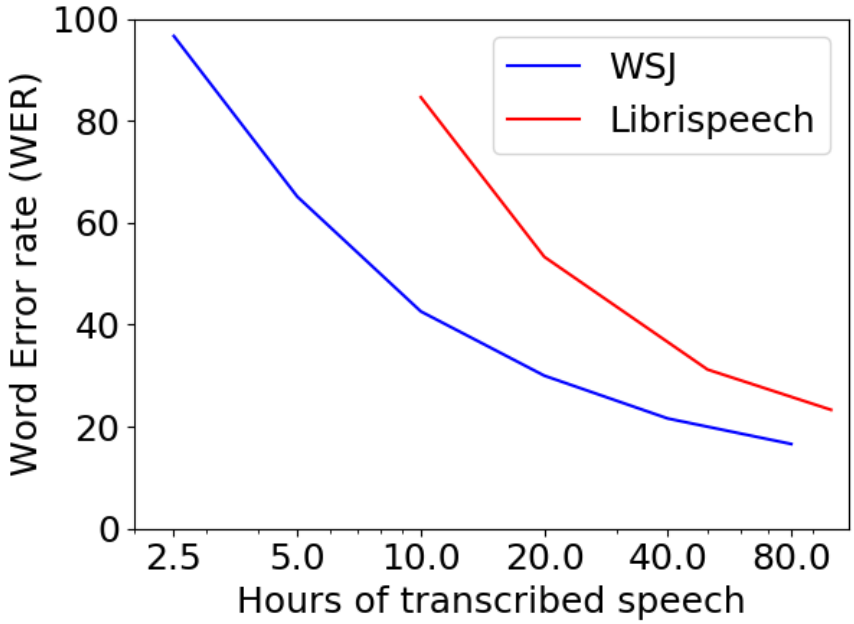


Figure 3-1: Word error rate as a function of number of hours of training data, for both the WSJ (blue line) and Librispeech (red line) corpora.

There is no clear line for when the available data for a language or domain makes the ASR problem "low-resource."

It is also difficult to judge exactly how WER correlates with the usefulness of an ASR transcript, but some research has explored this question. In one study, students were asked to take a quiz about the contents of a recorded lecture (Munteanu et al., 2006). Students who were given either a perfect transcript or a transcript with 25% WER did better on the quiz than those who did not have access to a transcript. However, students with no transcript outperformed students with access to a 45% WER transcript.

Table 3.3 gives qualitative examples of how the different low-resource models perform. WER is most useful averaged over many utterances - we see in this table that the outputs from the models trained on 5 and 10 hours of transcribed speech score the same WER on this one sentence, despite performing very differently on the full test set. Still, this table gives a sense of what different WERs mean, and how WER and CER interact. While the model trained with 20 hours of speech is clearly the

Hours	NOW CIBA IS TRYING TO MAKE AMENDS	WER	CER
2.5	**THE NINETEEN MILLION DOLLARS**	100.0	81.8
5	NOW **S. BARYS** TRYING TO **MADE MANAGE**	71.4	39.4
10	NOW **SEE BASED TRIED** TO MAKE **IMMENSE**	71.4	45.5
20	NOW **C. BE** IS TRYING TO MAKE **IMMENSE**	42.9	30.3

Table 3.3: Example outputs from models trained with different amounts of WSJ data, along with word error rate and character error rate. The ground truth transcript is "NOW CIBA IS TRYING TO MAKE AMENDS."

best performing, arguably none of these outputs is particularly useful in terms of conveying the meaning of the original utterance.

The results in Table 3.2 and Figure 3-1 have all been optimized for each individual training set by choosing the best performing out of a range of hyperparameter settings. Table 3.4 shows the impact of the different hyperparameters we experimented with in order to find the best baseline settings for each corpus on the 5 hour WSJ subset. We do not replicate the full suite of results for all corpora here; we just give the 5 hour WSJ subset as an illustrative example. For this corpus, the best configuration used 128 hidden units per recurrent layer, a dropout rate of 0.3 and a scheduled sampling rate of 0.1.

hidden	dropout	sched. samp.	WER	CER
128	0.3	0.1	65.1	35.3
64			80.4	50.9
256			93.9	78.1
	0.4		68.2	36.8
	0.2		81.3	52.7
	0.1		94.2	78.2
	0.0		96.6	79.1
		0.0	72.5	42.1

Table 3.4: Results from models trained on the 5 hour WSJ subset with a variety of hyperparameter options. The best performing configuration is in the first row of results; remaining rows are labeled with the hyperparameter that differs from the best performing model.

Unless otherwise mentioned, all models in this thesis use a dropout rate of 0.3 and scheduled sampling rate of 0.1. We found that the best number of hidden units was related to the amount of training data: 64 hidden units was best for the 2.5 hour

WSJ model, and 256 units was best for all WSJ models trained with at least 10 hours of transcribed speech. For librispeech, the best choice of hidden size was 128 for the 10 and 20 hour subsets, 256 for 50 and 100 hour subsets, and 512 units for the larger training corpora. 256 hidden units was best for all of the Babel languages, which each have 40 hours of training data.

3.3.3 Babel Languages

	Dholuo	Igbo	Javanese	Swahili
This work - Attention	58.9	85.5	83.8	74.6
Rosenberg et al. (2017) - HMM-DNN	41.6	61.4	57.3	
Rosenberg et al. (2017) - CTC	44.2	64.4	58.3	
Rosenberg et al. (2017) - Attention	48.2	62.1	64.6	
Cho et al. (n.d.) - Joint CTC/Attention				66.2

Table 3.5: Baseline results on selected Babel languages. All numbers in this table are word error rates.

Baseline results on selected Babel languages are in Table 3.5. Our baseline results are much worse than the results that we were able to find in the literature for these corpora. However, while we selected these languages because there were available end-to-end results for comparison, none of this other work is actually directly comparable to ours. All of the models in Table 3.5 except for ours are trained on the language of interest as well as training data from several other languages. We discuss multilingual ASR models in more detail in Chapter 2.2.3, and the joint CTC/attention model architecture in 2.2.3.

Despite the use of much larger training corpora than those used in this thesis, these Babel results give us a sense of how well the state-of-the-art techniques for low-resource languages can perform on these particular corpora. The techniques used in this thesis are orthogonal to those used in Rosenberg et al. (2017) and Cho et al. (n.d.), and hopefully can be productively combined with multilingual training techniques.

The results from Rosenberg et al. (2017) also give a sense of the relative performance of the different types of ASR models outlined in this chapter. For all of the languages tested in that paper, HMM-DNN models outperform end-to-end models.

The difference between CTC and attention-based models is less consistent; which one performs better is language-dependent.

3.4 Chapter Summary

In this chapter, we have demonstrated the low-resource problem in end-to-end ASR, setting up the motivation for this thesis. This problem is most clearly illustrated in our baseline results when training on subsets of standard English ASR training corpora. Even with hyperparameters selected specifically for each corpus, our models degrade significantly when the amount of training data is reduced. In the remainder of this thesis, these low-resource English subsets will be used for the bulk of our experiments, so that we can judge how our methods perform as the amount of training data is adjusted.

We have also presented results for two other categories of datasets. We use the full versions of the WSJ and Librispeech corpora to demonstrate that the attention-based ASR implementation used for this thesis is in line with those used elsewhere in the literature. We also present results on several corpora collected for the Babel program (Harper, 2014) in true low-resource languages. While our baselines are not directly comparable to other published results, all of the results in this section demonstrate the difficulty of training an effective end-to-end ASR model in these languages.

Chapter 4

Semi-Supervised ASR with Adversarial Training

4.1 Introduction and Motivation

In this chapter, we extend end-to-end models to smaller ASR corpora by focusing on a low-resource paradigm that is realistic for many low-resource languages: a small amount of transcribed speech along with much larger corpora of non-parallel speech and text. Both text and speech data are widely available on the Internet; we believe that this paradigm of semi-supervised ASR presents a clear path towards making ASR technology available in many currently underserved languages.

We approach semi-supervised ASR as a multi-task learning problem. Our goal is to train some of the parameters of our ASR model using tasks that require only speech or only text. The standard “task" that can be performed with unlabeled data is an autoencoder, which is a model that is trained to recreate its input. In this case, we train an autoencoder with text data that shares decoder parameters with the ASR model and an autoencoder with speech data that shares encoder parameters with the ASR model.

When the ASR model shares its decoder with a text autoencoder, that decoder must learn to decode the outputs of either the text encoder or the speech encoder. In order to ensure as much transfer as possible from the text autoencoder training

to the ASR model, we encourage the text and speech encoders to produce similar outputs. This way, the decoder can learn to handle both modalities the same way. To accomplish this, we use adversarial training. Adversarial training is a technique typically used for training a model to generate outputs that match a data distribution. Here, we instead use it to train two models - the speech and text encoders - to generate outputs from the same distribution.

Through a series of experiments on small subsets of the WSJ corpus, we demonstrate the effectiveness of this method. We show that this model architecture effectively uses non-parallel speech and text to improve performance, and that these improvements are biggest when the parallel training corpus is smallest. Additionally, we show that the creation of a shared embedding space for speech and text has a regularizing effect on the ASR model, improving performance over the baseline even when no extra data is used. We also analyze the factors contributing to these improvements, demonstrating in particular the importance of adversarial training to the performance of this model.

4.2 Related Work

The low-resource scenario explored in this paper closely resembles the one used in Tjandra et al. (2017) and Tjandra et al. (2018). In those papers, the authors train both ASR and text-to-speech (TTS) systems with their small transcribed speech corpus, and use both for pseudo-labeling. They use the ASR system to turn untranscribed speech into a synthetic training set for their TTS model and create a training set for the ASR model from stand-alone text using their TTS system. Once these synthetic datasets are used to further train their models, the new models are iteratively used to generate better synthetic training data. As in this work, Tjandra et al. (2018) create a semi-supervised corpus from the Wall Street Journal speech recognition corpus (LDC, 1994) - treating a portion of the original dataset as 'parallel' and the remainder as 'non-parallel'. Using this method, they achieve impressive results: their semi-supervised model closes 73% of the gap between the character error rate

of their low-resource baseline and high-resource topline results.

While these results are encouraging, the fact that the non-parallel speech and text come from the same dataset means that they are not truly independent from each other. In our experiments, we enforce independence between the non-parallel speech and text by ensuring that there is no overlap between the underlying utterances used for each modality. This creates a more difficult task, but one that must be addressed for real-world low-resource scenarios.

This work also shares many features with work in unsupervised machine translation (MT). Both Lample et al. (2017) and Artetxe et al. (2017) start with a common architecture for end-to-end supervised MT and add components to allow for unsupervised training. Specifically, non-parallel texts in the source and target languages are used to train separate encoder-decoder sequence to sequence autoencoder models. Adversarial training (see Goodfellow et al. (2014) for an overview) is used to push the hidden representations in the two different encoders to use the same embedding space. Thus, a decoder trained only as part of an autoencoder in the target language can also decode the outputs of the source language encoder.

The success of these unsupervised MT models relies on a key observation: word embedding spaces tend to have similar structure across languages (Mikolov et al., 2013a). Conneau et al. (2017) use this fact to learn an unsupervised mapping from the embedding space of one language to the embedding space of the other, then use that mapping to learn a dictionary. This dictionary can then be used to seed the training of a fully unsupervised MT system. We construct a similar model but use a small corpus of transcribed speech to seed the training of a semi-supervised ASR system.

4.3 Model Architecture

The architecture of our semi-supervised speech recognition model is shown in Figure 4-1. The components of the core speech recognition system are outlined in bold. This ASR model is the attention-based model presented in Chapter 2, with the attention

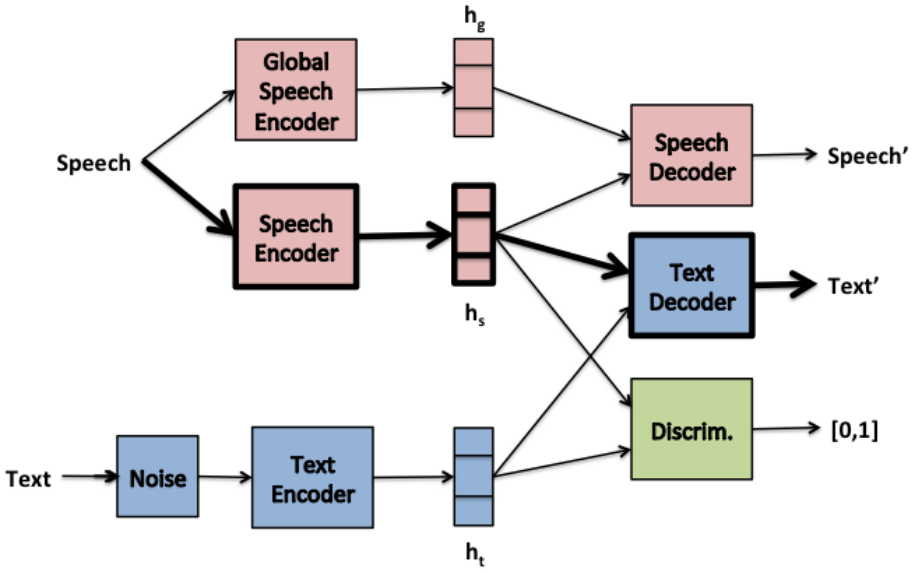


Figure 4-1: Semi-supervised ASR model architecture. Speech-to-text model is outlined in bold; text autoencoder is shaded blue; speech autoencoder is shaded red; discriminator for adversarial training is shaded green.

mechanism considered to be part of the decoder for the purposes of this figure. Our model architecture also contains a de-noising text autoencoder, shown in blue, which shares the ASR decoder and is trained in an unsupervised fashion using only text. The speech autoencoder is shown in red - it shares the ASR encoder and is trained from untranscribed speech. Finally, our model has a classifier, colored green, used as a discriminator for adversarial training of the outputs of the speech and text encoders. Each new component of this system - the text autoencoder, the speech autoencoder, and the adversarial training - is described in detail below.

4.3.1 Text Autoencoder

The text autoencoder has three components, shown in blue in Figure 4-1: a noise model, an encoder and a decoder. The decoder is shared with the ASR model - training this autoencoder thus also trains the parameters of our ASR decoder.

The text encoder takes one-hot character encodings as input. Its first layer is an embedding layer (Mikolov et al., 2013b) that converts these inputs to multi-

dimensional embedding vectors. It then has two bidirectional LSTM layers that do not downsample their input. As a character autoencoder is a relatively trivial learning task, we add noise to the input before feeding it the encoder, following Lample et al. (2017). Whil Lample et al. (2017) delete words and shuffle their order when adding noise for unsupervised MT, we delete characters but do not shuffle them. Monotonicity is an important feature of ASR and we want the decoder to be able to learn that. We drop characters with probability $p = 0.2$. The text autoencoder is trained with the same end-to-end maximum likelihood objective as the speech recognition model, with the same scheduled sampling procedure.

4.3.2 Speech Autoencoder

The speech autoencoder is shown in red in Figure 4-1. Speech requires a different autoencoder architecture than text, because the speech signal contains both linguistic and non-linguistic information. By pushing the speech and text encoders to use the same output embedding space, we will be encouraging the ASR encoder to contain only the linguistic information, but we also need to capture the non-linguistic information in order to train an autoencoder. As in Hsu, Zhang, et al. (2017), we build a hierarchical autoencoder: one encoder to capture the aspects of the signal that change over time (namely, linguistic content), and one encoder to capture the utterance-level properties of the signal (non-linguistic characteristics). Here, our original ASR encoder serves that first purpose, and a global speech encoder serves the second purpose. The output of the global encoder is appended to the output of the original speech encoder at every time step.

Our global encoder is a convolutional neural network (CNN) (LeCun and Bengio, 1998) with three layers. Each layer has a convolution, batch normalization, rectified linear unit (ReLU) non-linearities, and max-pooling. The first layer performs convolution in frequency, the next two perform convolution in time. The layers look at increasingly larger segments of speech - from a single frame at the lowest layer to 45 frames at the highest. The last layer pools over the entire utterance to generate a single utterance-level representation vector.

The speech decoder is a simple feed-forward neural network with two leaky ReLU layers and a linear layer on top. It takes as input a single vector - a concatenation of the output from the global encoder and a single output from the ASR encoder - and generates eight frames of output speech features. These are scored against the eight frames of input speech features that produced the given ASR encoder output.

The speech autoencoder is trained end-to-end using smoothed L1 loss, an element-wise loss which equals the L1 loss when the difference between the output and the label is greater than one and equals the L2 loss when that difference is less than one.

4.3.3 Adversarial Training

Adversarial training (Goodfellow et al., 2014) is a technique originally developed for training generative models to produce examples whose distribution matches a ground-truth data distribution. In the standard case, an adversarial model has two components - a generator and a discriminator - which are trained alternately. The discriminator is trained to differentiate between examples from the data distribution and outputs from the generator. The generator is trained to 'trick' the discriminator and generate examples that the discriminator will score as likely to have come from the data distribution.

Here, we instead use adversarial training to encourage the outputs of the speech and text encoders to share an embedding space: we have two generators (the encoders) and no data distribution. While there are a number of ways to potentially modify the original adversarial 'game' for our scenario, we chose to treat the output of the text encoder as the data distribution. This choice made training more effective because the text autoencoder, even with noise added, trains much faster than the ASR model and thus the text encodings are more stable over the course of training. We also theorized that this choice would further encourage the outputs of the speech encoder to contain only linguistic information. This adversarial training can be performed with all available text and speech data, as it does not assume any parallelism.

In our model, the discriminator is a simple feed-forward network with two fully-connected layers. It takes as input a single vector, and outputs a single real-valued

score in the interval $[0, 1]$. The discriminator is trained using binary cross-entropy loss to assign high scores to output vectors generated by the text encoder and low scores to those generated by the speech encoder. Adversarial training has been shown to benefit greatly from label smoothing (Goodfellow, 2016), so we set the desired output for vectors from the speech encoder to 0.1 and the desired output for vectors from the text encoder to 0.9. During discriminator training, the losses are not backpropagated to the encoders.

4.4 Experimental Details

For all experiments in this chapter, we use subsets of the Wall Street Journal (WSJ) corpus, as outlined in Chapter 2.

All recurrent layers in all components have 256 units. Character embedding layers have 64 units. The discriminator also has 256 units per layer. The convolutional layers in the global speech autoencoder have 32, 64, and 256 filters and kernels of size $(36, 1)$, $(1, 5)$, and $(1, 3)$. All have a stride of one. The first convolutional layer pools over three frames, the second over five inputs (15 frames), and the third over the entire utterance. We use batches of size 32 for discriminator training and 16 for all other training. For optimization, we use stochastic gradient descent (SGD) with momentum (Sutskever, Martens, et al., 2013) and a learning rate of 0.2. We stop training when WER on the validation set stops improving.

For all results in thid chapter, we use a beam of size 20 during decoding. For language model experiments, we used a 3-gram word-level language model. We use a language model weight of 0.5 and word bonus of 1 when decoding with a language model.

4.5 Results

In Figure 4-2, we compare the performance of our baseline attention-based ASR model (blue) with that of our proposed architecture (green and red). All three model

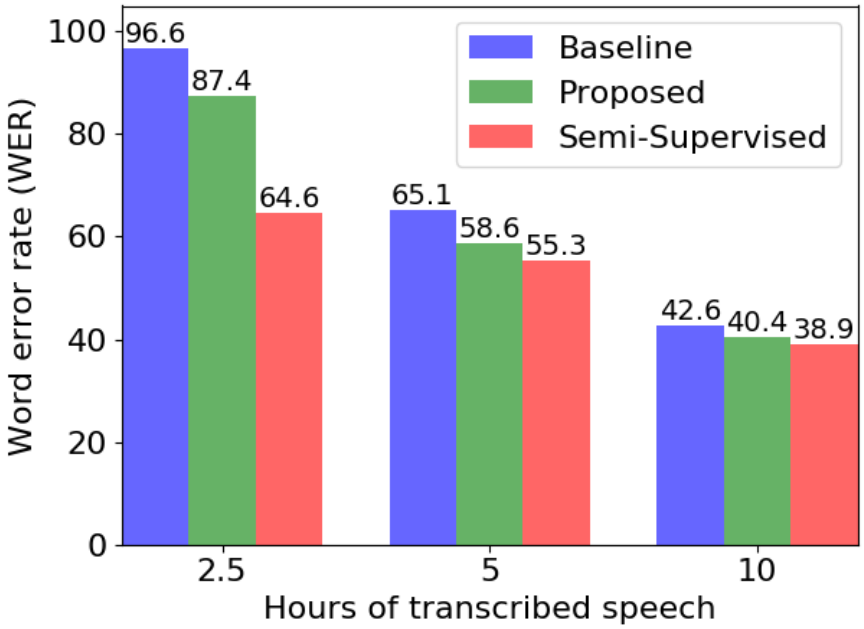


Figure 4-2: Word error rate by model architecture and hours of transcribed training data. The red and green bars both represent the results of our proposed architecture. The semi-supervised model (red) uses our larger non-parallel dataset in addition to the transcribed speech.

architectures were trained separately on 2.5, 5, and 10 hours of transcribed speech. The semi-supervised model (red bar) also used the larger non-parallel corpora for the autoencoders and adversarial training.

The baseline results denoted by the blue bars show clearly the impact of limited training data size. Trained on the full WSJ set, this baseline model achieves a word error rate (WER) of 16.6 and character error rate (CER) of 5.8, in line with previously reported similar models (for example, Bahdanau et al. (2016) reports a WER of 18.0). Using only 10 hours of transcribed speech degrades that performance to a WER of 42.6 and CER of 16.1. With only 2.5 hours of training data, the baseline model barely learns anything, resulting in a WER of 96.4 and CER of 83.0.

When trained on the same data as the baseline model, our architecture (green bars) produces improved results in all training conditions. Thus, this architecture allows us to more efficiently use very small ASR corpora in a fully-supervised scenario. We hypothesize multiple reasons for this. First, it is much easier to train the attention

mechanism as part of the text autoencoder, even with noise added to the input text. Second, adversarial training has a regularizing effect on the ASR model: there are many possible sets of parameters that would solve the ASR problem equally well on a small training corpus, but many fewer in which only linguistic information is represented at the output of the encoder. Even in the lowest resource case where the difference seems small, the model outputs are qualitatively much different. The baseline model trained on 2.5 hours of speech did not learn anything meaningful about the correspondence between speech and text; it essentially learned a language model and only generated 93 unique outputs for the 333 inputs in the validation set. Our new model architecture (without extra data), generated 330 unique sentences on that same set.

We see significant additional improvements through the use of the non-parallel data, as shown by the red bars. The impact of this semi-supervised training is highest when the least parallel training data is available; with 10 hours of transcribed speech, the improvements due to semi-supervised training are modest.

Our final set of experiments compares our model with a more traditional method of incorporating additional text data: the inclusion of an external language model during decoding. These results are in Figure 4-3 - the original results from Figure 4-2 are shown in lighter colors with the corresponding language model results superimposed on top. For reference, our baseline model trained on the full WSJ corpus achieves a WER of 10.5 when combined with a language model, in comparison to the WER of 10.8 reported in Bahdanau et al. (2016).

Somewhat surprisingly, adding a language model significantly hinders the performance of our model architecture trained on 2.5 hours of transcribed speech with no non-parallel data. On further inspection, we find that the language model overwhelms the speech recognition model in this case. Finding the optimal parameters to balance these models is beyond the scope of this work, but the issue is an important one for our suggested low-resource scenario and warrants further research. We get significant improvement with a language model on all models trained on either five or ten hours of transcribed speech; the improvement is greater - both relative and absolute - for

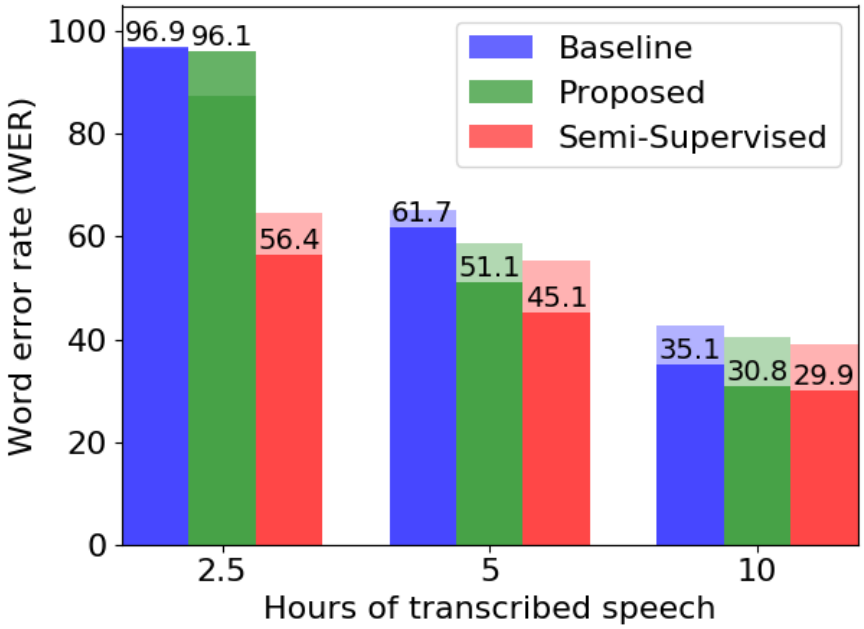


Figure 4-3: Word error rate by model architecture and hours of transcribed training data, when decoding with an external language model. The red and green bars both represent the results of our proposed architecture. The semi-supervised model (red) uses the full WSJ dataset as non-parallel text and speech.

all semi-supervised models compared to the baseline. Importantly, the gains achieved through our method of incorporating text data are complimentary with the gains due to the language model.

4.6 Ablation Study

We perform an ablation study to understand which components of our model have the most impact on performance. These results are in Table 4.1. All models used for this table were trained with 2.5 hours of transcribed speech. In addition to the three models from 4-2, we experiment with two semi-supervised models that each have a single feature of the complete model removed. We did not experiment with removing the text autoencoder because it is so integral to the model: without the text autoencoder, we cannot perform adversarial training or make any use of the additional text data.

Table 4.1: Ablation results. We compare the models tested in the previous section with two semi-supervised models that each have a single feature of the complete model removed. All models were trained on 2.5 hours transcribed speech.

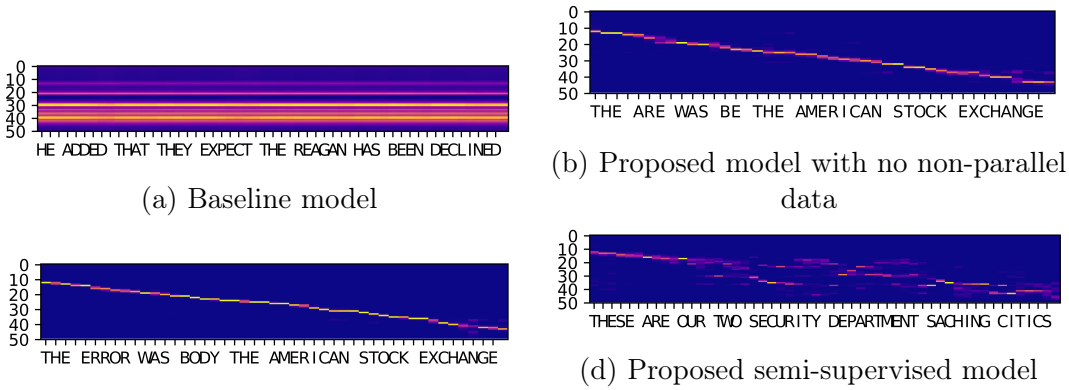
	WER	CER
Baseline model	96.4	83.0
Proposed model:		
with transcribed speech only	87.4	62.0
with parallel and non-parallel data	64.6	34.6
without speech autoencoder	69.0	42.0
without adversarial training	93.3	67.0

The first three rows of Table 4.1 mirror the left-hand side of Figure 4-2. The fourth row shows that removing the speech autoencoder degrades the performance of our model somewhat, but still allows for a significant improvement over the baseline. We use a relatively simple speech decoder here, and plan to experiment with more complex models in future work, which we hope will elicit further gains.

The final row of Table 4.1 shows that removing the adversarial training sharply reduces the performance of our model, demonstrating that having a shared embedding space for the outputs of the speech and text encoders is critical. Without adversarial training, the text corpus is still used to expose the decoder to a wider range of possible sentences, which likely accounts for the small improvement over the baseline using this model.

In order to further understand the improvements generated by the use of our model architecture, we inspect the attention mechanism during decoding. Figure 4-4 illustrates the activity of the attention mechanism during decoding of the same test sentence with four different models. In each subfigure, the y-axis represents the outputs from the speech encoder - one for every 8 frames, or 80ms, of speech - while the x-axis represents outputs from the decoder. The lowest weights are shown in blue, while the highest are shown in yellow.

When the baseline model is trained on only 2.5 hours of speech, the attention mechanism has the same weights at all time-steps (Figure 4-4a) - it has not learned anything about the correspondence between speech and text. When our proposed model architecture is trained with the same data, however, the attention mechanism



(a) Baseline model

(b) Proposed model with no non-parallel data

(c) Proposed semi-supervised model

(d) Proposed semi-supervised model without adversarial training

Figure 4-4: Attention weights during decoding for WSJ utterance 443c040c. In each subfigure, the y-axis represents the outputs from the speech encoder - one for every 8 frames, or 80ms, of speech - while the x-axis represents time-steps in the decoder. The lowest weights are shown in blue, while the highest are shown in yellow. The ground truth transcript is: THE ERROR WAS BY THE AMERICAN STOCK EXCHANGE.

has clearly learned quite a bit, as shown in Figure 4-4b.

Incorporating additional non-parallel data - as in Figure 4-4c - allows our model to learn more confident alignments between speech and text. However, when we remove the adversarial training (Figure 4-4d), the model struggles to find the correspondence between speech and text. The adversarial training is essential to this method of incorporating additional text data into ASR training.

4.7 Chapter Summary

In this chapter, we have proposed an effective model for semi-supervised ASR with limited transcribed speech and larger, separate speech and text corpora. This model architecture is premised on the idea of creating a shared embedding space for speech and text through adversarial training. This shared space allows us to regularize the model by forcing the outputs of the speech encoder to contain only linguistic information, while also enabling training of some parameters of the ASR model with only speech and others with only text.

We have demonstrated significant performance gains across 2.5-, 5-, and 10-hour subsets of the WSJ corpus, both with and without the use of additional unlabeled data

for training. The model architecture introduced in this chapter produces larger gains in lower-resource conditions, and much smaller gains as the amount of training data is increased. In all cases, we have shown that the gains from our model architecture are complementary with the use of an external language model during decoding.

Through an ablation study, we have demonstrated the relative contributions of the different components of our model. We find that the speech autoencoder adds only minimally to the model performance, while the adversarial training is vital.

Chapter 5

Supervised Alignment of Text and Speech Embedding Spaces

5.1 Introduction

In the previous chapter, we presented a model architecture and training method to extend end-to-end automatic speech recognition to a semi-supervised scenario in which large corpora of untranscribed speech and stand-alone text are available. While motivation for that work came from the availability of that non-parallel speech and text, the results demonstrated that this new model architecture was able to improve the performance of models trained on small ASR corpora even when no extra speech or text was used. That finding has motivated the work in this chapter, where we present a model architecture and training procedure inspired by the same underlying idea of a shared embedding space for speech and text but designed specifically for a fully supervised training scenario.

The results in this chapter further demonstrate the regularizing effect of the creation of a shared embedding space as part of ASR training. As discussed in Chapter 3, regularization is an important way of counteracting the overfitting that occurs when neural network models are trained on small corpora. There are many possible values of the neural network parameters that can perform well on a small training corpus. The idea behind this chapter is that a network that performs well on the training

corpus and only represents linguistic information at the output of the encoder will perform better on unseen data than one that only performs well on the training data. This additional constraint serves as a type of regularizer, as we are encouraging a particular type of solution to our problem.

The model architecture used here is a simplified version of the one presented in the previous chapter; it includes only the core attention-based ASR model and a text encoder. Our supervised encoding alignment method works as follows: we take a paired speech/text training example, separately encode the speech and text with our two encoders, then push these encoder outputs to be close together in the shared embedding space. The key contribution of this work is to handle the length discrepancies between the speech and text encoder outputs by comparing the attention-weighted speech encoder outputs to the text encoder outputs. We are thus using the attention mechanism to provide an alignment between speech and text and enable supervised learning of the shared embedding space.

We push the encodings of paired speech and text together with an objective function which we call the encoder loss. The model training proceeds in several stages. First, we train a baseline ASR model. Then, we use the encoder loss to train the parameters of the text encoder so that the text encoder outputs match the speech encoder outputs as closely as possible - by definition, this will capture the linguistic information in the current speech encoder outputs but nothing else. Next, we again use the encoder loss, but this time to retrain the speech encoder parameters to match the text encodings - removing the non-linguistic information from the speech encoder outputs. Finally, we retrain the decoder to effectively decode these linguistic-only encoder outputs.

While this chapter focuses on using parallel data to create the shared embedding space for speech and text, we can still take advantage of that shared embedding space for semi-supervised learning. In the last step of our fully-supervised training procedure described above, we train the decoder to decode the updated speech encoder outputs using the ASR corpus. We introduce a modified version of this last step in which we retrain the decoder parameters as part of a text autoencoder. This can be

done either in a fully-supervised way with only the text side of the ASR corpus, or in a semi-supervised way with additional text.

This chapter contains several sets of results. First, we use the Librispeech (Panayotov et al., 2015) corpus for experiments, training on the “clean 100" set or a smaller subset - 20 or 50 hours - of it. In this section, we show that, by following the procedure outlined above, we can significantly improve performance over the baseline and that the improvements are larger for models trained on less data. Additionally, we show that our fully-supervised model outperforms a semi-supervised model trained with a similar technique. We also explore the contribution of the different steps in our training procedure with several “early stopping" experiments in which certain stages of training are not trained to convergence.

In the next set of results, we confirm the effectiveness of this procedure for low-resource scenarios with several different subsets of the Wall Street Journal corpus (LDC, 1994). We compare results on the SI84 WSJ subset to related work in the literature. We also run experiments on the same WSJ subsets used in the previous chapter, enabling a direct comparison between the direct alignment technique explored here and adversarial alignment. Finally, we present results on the Babel Swahili corpus, demonstrating that this technique can be effectively extended to non-English languages.

5.2 Related Work

Our encoding loss objective function is inspired in part by the text-to-encoder (TTE) model in Hori, Astudillo, et al. (2019), which considers a semi-supervised scenario with a small ASR corpus and a larger corpus of untranscribed speech. In that work, the TTE model is a full encoder-decoder model trained to take text as input and mimic the speech encoder outputs that a trained ASR model produces when given the matching speech utterance. Then, the authors sample several possible text outputs from the ASR model for an untranscribed speech utterance and compute a cycle-consistency loss based on the difference between the output of the speech encoder and

the TTE model outputs. The idea is to train the ASR model to output transcripts for untranscribed speech whose encodings are close to the encodings of the original speech. This can be thought of as a way of generating pseudo-labels for untranscribed speech. Our model and training procedure share many similarities with Hori, Astudillo, et al. (2019), but we are able to significantly improve our baseline results using only the same data used to train the baseline model.

Within the traditional ASR framework, there is a thread of research that also bears some similarity to this work: adversarial training for removing non-linguistic factors from speech representations. This research has been largely focused on neural network acoustic models. Meng et al. (2018) uses adversarial training to push the outputs of an intermediate layer of a DNN acoustic model to be speaker-invariant; Serdyuk et al. (2016) uses a similar method to induce features that are invariant to noise conditions. In both cases, the authors reported significant improvements in performance. Here, we attempt to generate similar invariance without using explicit speaker or noise condition labels.

5.3 Methods

5.3.1 Encoder Loss

Figure 5-1 depicts the model used in this chapter, which is the same attention-based ASR model discussed in previous chapters, along with a text encoder. The text encoder architecture is quite simple: it has an embedding layer followed by two bidirectional LSTM layers that do not downsample their input. This is the same as the text encoder in Chapter 4. Of particular interest in this figure are the speech encoder outputs (g), text encoder outputs (h), and attention-weights speech encoder outputs (w). In the previous chapter, our adversarial training method pushed g and h to use the same shared embedding space. For this chapter, we use the attention-weighted speech encoder outputs, w, instead of g. These attention-weighted speech encoder outputs will necessarily be the same length as the text sequence, because one

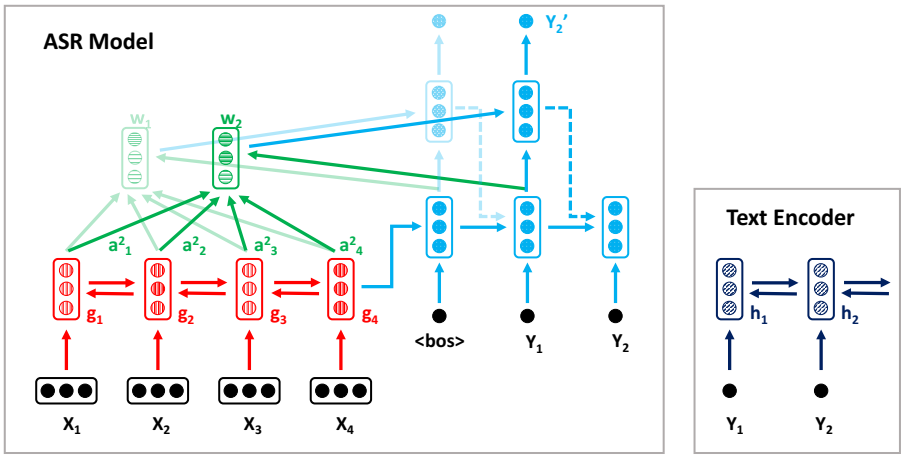


Figure 5-1: Schematic of the ASR model architecture when using teacher forcing, at time-step $t = 2$. Speech inputs (X) and text labels (Y) shown in black solid circles. Encoder outputs (g) shown in red vertical-striped circles. Decoder states and outputs shown in blue dotted circles. Attention-weighted encoder outputs (w) shown in green horizontal-striped circles, with attention weights (a) computed as part of the transformation from g to w. Calculations from $t = 1$ rendered in faded colors. Separate schematic shows the calculation of the text encoder outputs, h.

is generated for eery decoder time-step.

The inputs to our model are (X, Y) pairs, where X is a sequence of speech features and Y is a sequence of characters. $g(X)$ is the output sequence from the speech encoder given input X; it has length N, and $g_n(X)$ denotes the n^{th} element of $g(X)$. $h(Y)$ is the output sequence of the text encoder given input Y; its length is T. We use the MLP attention mechanism described in detail in Bahdanau et al. (2016); its output, $a(X, Y)$, is an N-dimensional score vector at each decoding timestep t, where $\sum_{n=1}^{N} a_n^t(X, Y) = 1$. We compute the attention-weighted speech encoder output, w_t, for timestep t as:

$$w_t(X, Y) = \sum_{n=1}^{N} a_n^t(X, Y) * g_n(X) \tag{5.1}$$

The variables in Equation 5.1 (X, g, a, and w) are labeled in Figure 5-1. We use the following equation to compute the encoder loss for the full utterance (X, Y):

$$L^{enc}(X, Y) = \sum_{t=1}^{T} SmoothL1(w_t(X, Y), h_t(Y)) \tag{5.2}$$

Equation 5.2 calculates the difference between the attention-weighted speech encodings, w, and the text encodings, $h(Y)$. We use an element-wise smoothed L_1 loss that is equivalent to the L_2 loss when the absolute difference is less than one and the L_1 loss otherwise.

$$SmoothL1(a, b) = \begin{cases} 0.5(a-b)^2, & \text{if } |a-b| < 1 \\ |a-b| - 0.5, & \text{otherwise} \end{cases}$$

This is conceptually similar to Hori, Astudillo, et al. (2019), which uses the sum of the L_1 loss and the L_2 loss when comparing speech and text encoder outputs.

5.3.2 Training Procedure

For all experiments in this paper, we follow a sequential training procedure with different loss functions for each step. The steps of this procedure are outlined in Table 5.1. Step 1 is baseline ASR training; this step impacts all of the parameters in the ASR model. In Step 2, we train the text encoder to match its outputs to those of the speech encoder; all parameters are fixed except for those of the text encoder. Step 3 uses the same loss as Step 2, but it is now used to train the parameters of the speech encoder, pushing the speech encodings closer to the text encoder outputs. All of these first three steps are always trained with the same corpus of paired text and speech.

Step	Description	Loss Function	Components Trained
1	ASR Training	L^{ML}	speech encoder, attention, decoder
2	Text Encoder Training	L^{enc}	text encoder
3	Speech Encoder Training	L^{enc}	speech encoder
4	Decoder Training	L^{ML}	attention, decoder

Table 5.1: Supervised encoding alignment training procedure.

We experiment with three variants of Step 4; for all three, we fix the parameters of the encoders and train only the decoder and attention mechanism. In the first, Step 4a, we simply do standard ASR training. In version 4b, we do combined ASR training and text autoencoder training, with the text drawn from the ASR corpus itself. In version 4c, we again do both ASR training and text autoencoder training, but the text comes from a larger text corpus with no associated speech.

For our main results, we train each step to convergence on the ASR validation set before moving on to the next step. Steps 1 and 4 are considered to have converged with the WER on the validation set stops improving. For Steps 2 and 3, we use the loss between the speech and text encodings of the validation set as our convergence criterion.

We also present results for a set of experiments in which we explore the impact of stopping one of the training steps early, while still training the rest to convergence. These experiments are designed with two questions in mind. First, whether it is possible to achieve similar performance improvements with fewer overall training iterations, and second, what the requirements are for an effective shared encoding space for text and speech.

We considered an alternate training procedure in which we replace Steps 1 and 2 with simply training a text autoencoder, followed by Step 3, in which we train the parameters of the speech encoder to minimize the loss between the speech and text encoder outputs. We found, however, that this was not a reliable method for training the attention mechanism in the decoder.

We also considered a number of ways of combining these steps together (as with the combined objective function we used for adversarial alignment), but were unable to achieve satisfactory results.

5.3.3 Experimental Details

In this chapter, we experiment with both Librispeech (Panayotov et al., 2015) and the Wall Street Journal (WSJ) corpus (LDC, 1994). For Librispeech, we present results for the standard clean460 training set as a 'topline' and treat the standard clean100

training set as a low-resource training regime. We also create two lower-resource scenarios with subsets of the clean100 set: clean50 and clean20 which comprise 50% and 20% of the utterances in the clean100 set, respectively. We use the standard clean validation and test sets for all Librispeech experiments.

For WSJ, one set of experiments uses the SI84 training set (also called WSJ0), which is a standard subset that contains approximately 15 hours of transcribed speech. We also present experiments using the 5- and 10-hour WSJ subsets created for the previous chapter. For all WSJ experiments, use the standard dev93 set for validation and all WSJ scores are reported on the eval92 set.

We use 80-dimensional log-mel filterbank features, computed using 25ms frames with a 10ms frame-rate for all speech input. Librispeech text was segmented using a unigram wordpiece model with a 500 unit vocabulary with a maximum unit length of four characters. WSJ models are character based.

All Librispeech models trained with at least 50 hours of speech used 512 units for all encoder layers and 1024 units for all decoder layers. Models trained on the clean20 set used 128 units for all encoder layers and 256 units for decoder layers - we find that a smaller model produces better results in very low-resource scenarios. For WSJ0, we used 320 units in all layers, to match comparable prior work. We used 128 units per layer for the 5-hour WSJ model and 256 units per layer for the 10-hour model. All word embedding layers in the Librispeech and WSJ0 models had 128 units; word embedding layers in the smaller WSJ models had 64 units.

5.4 Librispeech Results

5.4.1 Main Results

First, we establish a baseline for our low-resource conditions. These results are in the second and third columns of Table 5.2. We report both character error rate (CER) and word error rate (WER). For comparison, our topline model trained on the clean460 training set achieves a WER of 12.7% and CER of 5.7%. Our baseline results on the

Table 5.2: Comparison between baseline models and models trained with encoding alignment procedure, using different amounts of training data.

	Baseline		Aligned	
Train Set	CER	WER	CER	WER
clean100	11.2	23.3	9.9	21.3
clean50	16.0	31.2	14.6	29.2
clean20	30.4	53.3	27.7	49.7

standard 460 and 100 hour training sets are comparable to those reported in Hori, Astudillo, et al. (2019). In that work, the authors report 11.1% CER and 25.2% WER on the clean100 subset. THey report 4.6% CER and 11.8% WER on the clean460 subset. Our ASR encoder has fewer layers with more units each and our decoder has a different output vocabulary, so it is to be expected that our results would be slightly different from theirs. In particular, our slightly less powerful model performs better on the smaller dataset but worse on the larger dataset.

The last two columns of Table 5.2 show the results of our encoding alignment procedure. In all cases, we are able to improve on the baseline results, despite using exactly the same training data as the comparable baseline model. The relative improvement in WER is largest on the clean100 set (8.6%), while the absolute improvement is largest on the smallest training set (3.6%). It is also worth noting that our best result on the clean100 training set is comparable to the model presented in Hori, Astudillo, et al. (2019) (CER: 9.4%, WER: 21.5%), which uses an extra 360 hours of speech (without transcriptions).

5.4.2 Early Stopping Results

In this section, we experiment with early stopping of different steps of our training procedure. All experiments in this section use the clean20 training set. Training to several different objective functions in sequence can be time consuming - we want to understand to what extent each of these training steps is necessary and whether it is possible to stop intermediate steps early without sacrificing final performance.

Speech Encoder Training (Step 3)

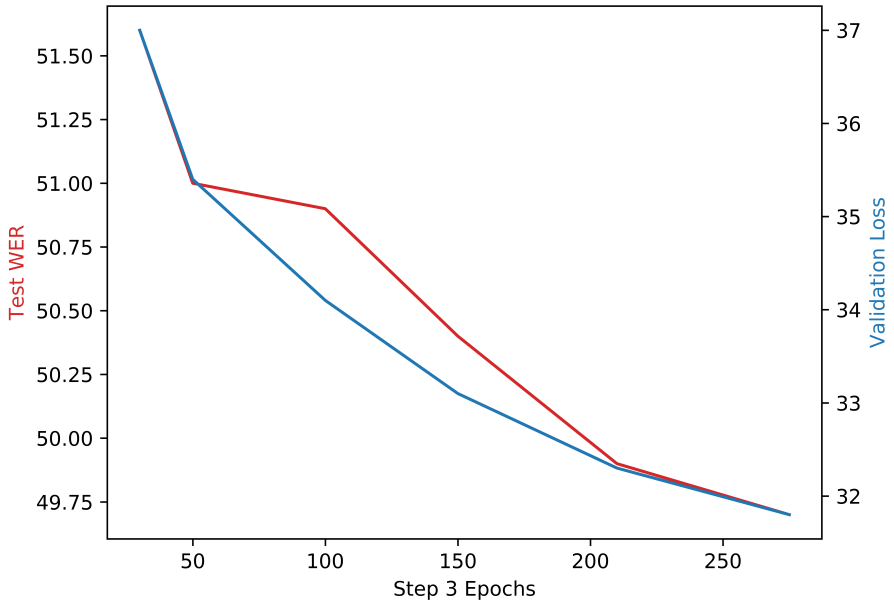


Figure 5-2: Validation set loss (blue) and test set word error rate (red) as a function of the number of Step 3 training epochs completed.

Stopping Step 3 early is a test of our core hypothesis - we expect that it will negatively impact our results if the speech encoding space is not fully aligned to the text encoding space. To perform this test, we train Steps 1 and 2 to convergence, then save models at several points during Step 3. From each of these models, we train Step 4 to convergence to get a final WER.

As shown in Figure 5-2, we see a direct relationship between the encoding loss on the validation set and the WER on the test set as training progresses. This confirms our hypothesis that directly matching the speech encoder outputs to a text encoding space will improve generalization performance of the model.

Text Encoder Training (Step 2)

There is no theoretical reason why the text encoding space must be as close to the original speech encoding space as possible, given that we will be re-training the speech

encoder to match whatever text encoder space is defined in Step 2. In these experiments, we stop Step 2 training early, then train Steps 3 and 4 to convergence using their respective stopping criteria. The results of these experiments are in Table 5.3, with models referenced based on how many epochs of Step 2 training were run before moving on to Step 3.

Table 5.3: Results of encoder alignment models with early stopping of Step 2. After Step 2, all models were trained to convergence on Steps 3 and 4.

Step 2 Epochs	Val L^{enc}	CER	WER
50	86.4	27.6	49.9
100	82.4	27.8	50.0
200	80.8	27.7	49.7

Table 5.3 shows that the final model performance does not depend on the number of epochs of Step 2 training. The second column, indicating the encoder loss on the validation set at the end of Step 2 training, shows that this training has not converged after 50 or 100 epochs; still, the final results are comparable across all three models.

It is not necessary to train Step 2 to convergence, but that does mean that we will be able to save computation by stopping Step 2 early; it is possible that we are simply trading fewer iterations of Step 2 for more iterations of Step 3 in order to achieve the same final result. As shown in Figure 5-3, this is not the case: regardless of how many epochs of Step 2 we complete, Step 3 training follows virtually the same path in terms of the validation set loss and the models converge at approximately the same number of Step 3 epochs.

It is also interesting to note the difference in the values of the validation loss after Step 2 (in Table 5.3) and after Step 3 (in Figure 5-3). These numbers confirm that the original speech encoder outputs contain much non-linguistic information that the text encoder cannot learn to represent, and that much of this extraneous information is removed from the speech encodings during Step 3 training.

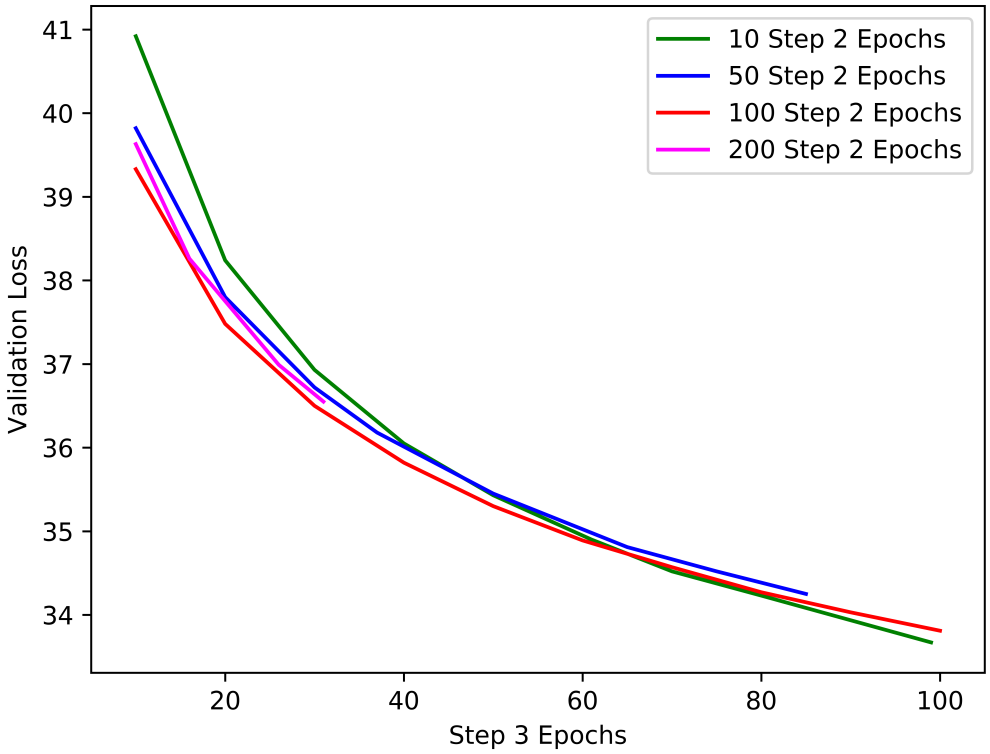


Figure 5-3: Validation set loss as a function of the number of Step 3 training epochs completed. Each line represents a different early stopping point for Step 2. Validation loss at $X = 0$ is the loss at the end of Step 2.

5.5 WSJ Results

5.5.1 WSJ0

Since the Librispeech clean50 and clean20 training sets were created for this work, we have no comparison in the literature for those results. We ran the same experiments on the WSJ0 corpus, to confirm both the competitiveness of our baseline model and the effectiveness of our proposed method. Our baseline model achieves a CER of 15.0% and WER of 36.5%, compared to similar attention-based models reporting a CER of either 17.0% (Kim et al., 2017) or 15.8% (Karita et al., 2018). The training procedure presented here reduces the CER to 13.8% and the WER to 35.0%. This represents a 4.1% relative improvement in WER.

5.5.2 Comparison to Adversarial Training

Train Corpus	Technique	WER	CER	WER (LM)
5 hours	Baseline	62.2	35.3	56.7
	Direct Alignment (4a)	61.0	33.3	54.9
	Direct Alignment (4b)	59.8	32.8	53.4
	Direct Alignment - semi-sup	57.6	30.9	50.0
	Adversarial Alignment	53.0	25.8	41.5
	Adversarial Alignment - semi-sup	**47.8**	**21.2**	**35.0**
10 hours	Baseline	39.5	18.9	29.3
	Direct Alignment (4a)	38.3	18.7	29.3
	Direct Alignment (4b)	37.1	18.5	28.9
	Direct Alignment - semi-sup	**34.6**	16.1	26.0
	Adversarial Alignment	37.4	16.1	25.8
	Adversarial Alignment - semi-sup	35.9	**14.8**	**24.9**

Table 5.4: Comparison of direct alignment and adversarial alignment on the 5 and 10 hour subsets of the WSJ corpus.

We also ran experiments on the 5 hour and 10 hour WSJ subsets; these results are in Table 5.4. In this table, we experiment with the three versions of step 4: no text autoencoder training (4a), text autoencoder training with only the text from the ASR training corpus (4b), and text autoencoder training with additional text (semi-sup). We see that we get small gains from the use of the text autoencoder, and slightly larger gains from the incorporation of additional text.

The results enable a comparison to the adversarial alignment method presented in the previous chapter. Without any non-parallel data, direct alignment does not work as well as adversarial alignment, with a particularly stark difference in the lowest-resource condition. We hypothesize that this is due to the attention mechanism; whereas the adversarial alignment technique improves the training of the attention mechanism, the direct alignment technique only impacts the training of the encoder.

When semi-supervised training is performed, adversarial training still works better on the 5 hour subset, but this direct alignment method works better on the 10 hour subset. This result suggests that the direct alignment method is able to form a better shared text/speech embedding space than adversarial training, as long as enough training data is available to build an adequate baseline ASR model, and to

do supervised training of the shared embedding space.

5.6 Babel Languages

We saw in the previous section that this technique works better with 10 hours of WSJ data compared to 5 hours, the reverse of the pattern in the previous chapter where we were able to get bigger gains from poorer-performing baseline models. With the Babel languages, we have more training data than we used for any of the WSJ experiments, but because of the nature of the data the baseline performance is worse than any of the baseline WSJ models.

We tested this technique on the Swahili corpus, where we had a baseline of 74.6% WER and 42.4% CER. After embedding space alignment, using ASR training only during Step 4, this is reduced to 69.7% WER and 39.8% CER. When we include text autoencoder training in step 4, this is further improved to 68.3% WER and 39.5% CER. This improvement gets us most of the way to the 66.2% WER reported in Cho et al. (n.d.) for a model that also made us of training data from several other languages. These results demonstrate that the theory behind this technique is sound even for non-English languages.

5.7 Chapter Summary

In this chapter, we developed a training procedure for attention-based ASR models designed to improve their generalization performance in low-resource settings. Our training strategy includes the addition of a text encoder network to a standard ASR model architecture, and a novel objective function designed to push the encoder component of the ASR model to represent only linguistic information. This is accomplished by encouraging the attention-weighted speech encoder outputs to match the outputs of the trained text encoder when the networks are fed paired speech/text inputs.

We experiment with several subsets of both the Librispeech corpus and the WSJ

corpus, and find that our procedure improves WERs in all cases. On the 100 hour Librispeech set, we achieve comparable improvements to those reported in a related paper that made use of an additional 360 hours of untranscribed speech (Hori, Astudillo, et al., 2019), while using only the same data used to train the baseline model. Additionally, we show through a series of early-stopping experiments on the Librispeech corpus that the second step in our training process does not need to be trained to convergence, which can reduce the overall training time needed.

We also experiment with the 5 and 10 hour subsets of the WSJ corpus, and compare these results to those in the previous chapter. We find that adversarial alignment of the text and speech embedding spaces works better in the lowest resource condition, but direct alignment works better with the larger subset. We hypothesize that this difference is related to both the quality of the baseline ASR model and the fact that the direct alignment method relies only on the parallel training data to create the shared embedding space.

Finally, we present results on the Babel Swahili corpus. The best model from this chapter achieves an 8.4% relative WER improvement over the baseline on that corpus.

Chapter 6

Subword Regularization for ASR

6.1 Introduction

Within the field of automatic speech recognition (ASR), determining the correct scale of units to use at various stages in the recognition pipeline is a key research area. The idea of finding and using a vocabulary of "subword units" - sequences of one-or-more phonemes or characters - has been explored extensively in the context of traditional HMM-based state-dependent acoustic models (Bazzi and Glass, 2000; Bulyko et al., 2012) and has become increasingly common at the output of end-to-end deep neural network models (Xiao et al., 2018; Zenkel et al., 2017; Zeyer et al., 2018). For recent end-to-end ASR models, subword units are most often discovered using the byte pair encoding (BPE) technique (Sennrich et al., 2016), which was originally developed for machine translation.

Subword regularization (Kudo, 2018b) is a more recent technique for both discovering and using subword units that has been shown to produce large gains over high-quality machine translation baselines that use BPE. Rather than deterministically splitting every word into subword units, subword regularization involves learning a probabilistic model over subword units and sampling a new segmentation for every word each time it appears. This technique is conceptually similar to regularization techniques like dropout and data augmentation, which we discussed in Chapter 2. While part of the motivation of this work is simply to extend this technique to a new

domain, we were particularly interested to explore how it would work in low-resource scenarios.

In this chapter, we describe our work applying subword regularization to ASR, where it had not previously been tested. As in machine translation, this simple technique yields large ASR improvements across a variety of datasets. While we confirm earlier results that standard subword units are most effective when the vocabulary size is small, we find that subword regularization supports much larger vocabulary sizes, and that larger vocabulary sizes work best, as long as enough regularization is applied. This finding extends to low-resource scenarios, specifically the 20-hour Librispeech subset and the 5- and 10-hour WSJ subsets, where we demonstrate that subword regularization is more impactful when less training data is available. We additionally present analysis demonstrating that the mechanism behind these improvements is a regularizing effect similar to other techniques that penalize very confident output distributions.

6.2 Prior Work

The idea of finding and using a vocabulary of subword units is a longstanding one within ASR. These units were originally concieved as a way to avoid the out-of-vocabulary (OOV) problem with traditional HMM-based models (Bazzi and Glass, 2000; Bulyko et al., 2012; Kneissler and Klakow, 2001), whose output units are typically words. More recent character-based end-to-end models (Chan, Jaitly, et al., 2016) do not have an OOV problem, but researchers have still found advantages to using a larger vocabulary of subword units as opposed to characters.

The subword units used with end-to-end ASR models are typically discovered using byte pair encoding (BPE) (Sennrich et al., 2016), which learns a subword vocabulary from a text corpus that can be used to deterministially segment any word. BPE units have been shown to improve ASR performance when used with attention-based systems (Chiu et al., 2018; Zeyer et al., 2018), CTC-based systems (Zenkel et al., 2017), and hybrid attention-CTC models (Xiao et al., 2018). This prior work

shows a consistent trend of small improvements due to subword units, with the best performance coming with relatively small vocabularies of 300 or 500 units. Chiu et al. (2018) also argues for subword units with attention-based models on efficiency grounds: larger units require fewer decoding steps, and also limit the length of the dependencies the decoder must learn.

6.3 Subword Regularization

Subword regularization (Kudo, 2018b) is based on a simple unigram language model (LM). Given a vocabulary of subword units, such a model is easy to train, and Viterbi search can efficiently find the best segmentation for any word according to that model. Alternatively, segmentations can be sampled from the language model. Kudo (Kudo, 2018b) introduced a technique for joint learning of the unigram LM and a vocabulary of a desired size. The vocabulary always includes all of the single characters in the alphabet, so that there will never be out-of-vocabulary words. This technique starts with the set of the most frequent strings in the corpus, a set much larger than the desired vocabulary size. Given this seed vocabulary, the LM is computed. Kudo then computes the "loss" associated with each unit - the reduction in the overall likelihood of the corpus if that unit were to be left out. The LM is then re-estimated using the 80% of subword units with the highest associated loss. This process is repeated until the appropriate vocabulary size is reached.

Once the LM and vocabulary are fixed, we can sample a segmentation from the following multinomial distribution:

$$P(x_i|X) \cong \frac{p(x_i)^\alpha}{\sum_{j=1}^{n} p(x_j)^\alpha}$$

where n is the number of n-best segmentations used to approximate the true distribution and α is a smoothing parameter. $\alpha = 0$ creates a uniform distribution, while larger settings of α move closer to the Viterbi segmentation. Kudo uses the forward-filtering and backward-sampling algorithm (Scott, 2002) for exact sampling from all

possible segmentations. All experimental results in this paper use this setting.

We make some minor modifications to the technique described in Kudo (2018b): in addition to specifying a desired vocabulary size, we specify the maximum subword length in characters, which is easily implemented by only considering subwords up to that length in the seed vocabulary. In Kudo (2018b), subwords do not cross word boundaries, but each space is included as part of the following word. We treat all spaces as stand-alone characters, not included in any subword units.

6.4 Data and Methods

6.4.1 Data

This chapter contains experiments run on both Librispeech and WSJ. All Librispeech results reported here are on the clean test set. All subword unigram models and vocabularies were learned on the text of the full training set for each corpus, regardless of the amount of data used to train the ASR model. Word-level n-gram language models for both datasets were trained on the accompanying text corpora; we use a 3-gram LM for librispeech and a 4-gram LM for WSJ.

We used the SentencePiece library (Kudo, 2018a) to train unigram models and sample subword segmentations of text. The large vocabularies we use here - 7775 units with $k = 4$ and 2025 units with $k = 3$ for Librispeech, and 7229 units with $k = 4$ for WSJ - were the largest vocabularies that could be recovered by that library using its most permissive settings.

Table 6.1 shows some example segmentations for the utterance "A LITTLE ATTACK OF NERVES POSSIBLY", which comes from the Librispeech corpus. We segment this utterance with two unigram models; one with 500 units and one with 7775 units. The first line of each section of this table shows the Viterbi best segmentation according to the unigram model for that section. The smaller model gives a sequence of 15 units, while the larger model gives a sequence of 11 units, not including spaces. Both are much shorter than the original sequence of 29 characters.

|V|	k	α	A LITTLE ATTACK OF NERVES POSSIBLY
500	4	∞	A LITT_LE AT_T_AC_K OF N_ER_VES POSS_I_B_LY
		1	A LITT_LE AT_T_A_C_K OF N_ER_VES POSS_I_B_LY
		1	A LITT_LE A_T_T_AC_K OF N_ER_VES POSS_I_BL_Y
		0.5	A LITT_L_E AT_T_A_C_K OF NE_R_VES POSS_I_B_LY
		0.5	A L_IT_T_LE AT_T_AC_K OF N_ER_V_ES POSS_I_B_L_Y
7775	4	∞	A LITT_LE AT_TACK OF NERV_ES POSS_I_BLY
		1	A LITT_LE AT_TACK OF NERV_E_S POSS_I_BLY
		1	A LI_TTLE AT_TACK OF NERV_ES POSS_I_BLY
		0.5	A LITT_L_E AT_TACK OF NER_VES PO_S_SI_BLY
		0.5	A L_I_TTLE A_T_TACK OF NERV_ES POSS_I_BLY

Table 6.1: Example segmentations of the utterance "A LITTLE ATTACK OF NERVES POSSIBLY", from the Librispeech corpus, using the unigram segmentation model. Units that differ from the best segmentation according to the unigram model ($\alpha = \infty$) are highlighted in red.

The additional lines in each section show two sampled segmentations each for two different settings of the α parameter, $\alpha = 1$ and $\alpha = 0.5$. Higher settings of α mean less regularization; the segmentations sampled with $\alpha = 1$ differ from the Viterbi best segmentation in only one or two places. The segmentations sampled with $\alpha = 0.5$ differ more often from both the Viterbi best segmentation and from each other.

6.4.2 Model Details

We use the same attention-based model architecture from previous chapters. The embedding layers in all models have 64 units, regardless of the size of the subword vocabulary. The other layer sizes were dependent on the training corpus. Models trained with the 960- and 460- hour Librispeech corpora had recurrent layers with 512 units each; those trained with the smaller Librispeech corpora had 256 unit recurrent layers. We also used 256 unit recurrent layers for the model trained with the 10 hour WSJ subset, and we used 128 unit layers for the model trained with only 5 hours of WSJ.

6.5 Results

6.5.1 Librispeech

Train Corpus (hours)	Characters	Subword ($\|V\| = 500, k = 4$)			
		$\alpha = \infty$	$\alpha = 1$		
960	14.3	10.2	10.1		-0.1
460	19.0	12.7	11.9		-0.8
100	28.6	22.7	21.3		-1.4
20	53.3	48.0	43.8		-4.2

Table 6.2: Subword regularization results on subsets of the Librispeech corpus. All results in this table are word errror rates. $|V|$ is the size of the subword vocabulary, k is the maximum length of a subword unit in characters. α is the regularization parameter; $\alpha = \infty$ is the condition where we always use the Viterbi best subword segmentation (no regularization), while $\alpha = 1$ is with segmentations sampled from the unigram model. The last column shows the performance difference between the two previous columns.

Table 6.2 shows WER results on subsets of the Librispeech corpus using characters, deterministic subword units ($\alpha = \infty$), and subword regularization ($\alpha = 1$). The same training corpus and model size were used for all results within each row. The first column indicates the amount of training data used; the next three columns represent different output units used for training. For both subword conditions we used a vocabulary of 500 subword units, with a maximum subword unit length in characters of 4. These settings match what previous research has shown to be the best settings for the use of deterministic subword units with ASR.

In the second column, where single characters were used as output targets, we see the impact of reducing the amount of training data on ASR performance; once again, training with limited data severely increases WER. The models in the third column were trained using a fixed subword segmentation for each word. This usage of subword units has become standard for end-to-end ASR models, and for good reason: in all four cases, subwords produce better results than characters. Using subwords reduces the word error rate (WER) relative to using characters by roughly the same amount in all cases (between 4 and 6 percent absolute WER).

The fourth column, with $\alpha = 1$, is the subword regularization case; the last

column shows the additional absolute gains from subword regularization beyond the use of deterministic subwords. Subword regularization is more effective than using deterministic subword units in all cases. As hypothesized, subword regularization becomes more effective as the size of the training corpus is reduced.

$\|V\|$, k	$\alpha = \infty$	$\alpha = 2$	$\alpha = 1$	$\alpha = 0.5$
500, 3	51.1	46.5	45.2	44.8
500, 4	48.0	44.9	43.8	43.6
500, 5	51.4	47.0	44.8	43.2
2025, 3	58.5	45.2	42.6	41.8
7775, 4	58.0	50.2	41.5	40.9

Table 6.3: Subword regularization results on the 20 hour Librispeech subset with different vocabulary sizes, maximum subword lengths, and regularization amounts.

Table 6.3 shows results for a range of subword regularization settings on the 20 hour subset. For each subword vocabulary used here, we look at three different settings of the subword regularization parameter. In the first section of this table, we explore the effect of k, the maximum subword unit length parameter. The $k = 4$ subword vocabulary performs best in most cases, although the $k = 5$ vocabulary performs slightly better at $\alpha = 0.5$.

In the next section of Table 6.3, we explore larger vocabulary sizes. For both $k = 3$ and $k = 4$, we use the maximum subword vocabulary size that could be discovered on the text of the Librispeech corpus. The second column of this table confirms the results of previous research: using large vocabularies of subword units hurts performance when no regularization is applied. However, in the remaining columns, we see that subword regularization allows us to achieve better performance with larger vocabularies than with smaller ones.

6.5.2 Wall Street Journal

For the WSJ corpus, we focused on low-resource settings, training with either 5 or 10 hours of transcribed speech. These results are in Table 6.4. For each subset, we experimented with both a 500 unit vocabulary and a 7229 unit vocabulary. While the large vocabulary worked well on the 20-hour Librispeech corpus, we were unsure

Train Corpus	($\|V\|$, k)	α	WER	CER	WER (+LM)
5 hours	(31, 1)		65.1	35.3	63.7
	(500, 4)	∞	64.5	34.4	60.9
		1	54.5	25.8	47.7
		0.5	54.3	24.1	46.7
		0.25	55.9	25.0	46.9
	(7229, 4)	∞	85.7	62.7	84.3
		1	58.0	28.9	53.4
		0.5	**53.6**	**23.3**	**44.2**
		0.25	55.4	23.6	45.0
10 hours	(31, 1)		42.6	19.9	35.1
	(500, 4)	∞	39.3	19.0	34.6
		1	37.1	14.8	28.9
		0.5	37.8	14.5	30.0
		0.25	38.1	14.4	29.9
	(7229, 4)	∞	43.3	22.1	39.6
		1	**33.7**	**13.2**	**27.1**
		0.5	36.3	13.8	27.9
		0.25	38.4	14.1	29.1

Table 6.4: Subword regularization results on the 5 and 10 hour subsets of the WSJ corpus. $|V| = 31$ is the character baseline.

whether it would be possible to train with such a large vocabulary on the 5 hour subset. As in the previous section, we find that the large vocabulary with $\alpha = -1$ hurts performance compared to the character baseline. With one exception, the large vocabulary produced better results than the small vocabulary across both corpora and all choices of α less than ∞. The one exception is the 5-hour subset with $alpha = 1$; with this setting, the small vocabulary is better than the large. This result demonstrates the importance of doing enough regularization with the large vocabulary to expose the model adequately to all of the subword units.

Overall, the best subword regularization setting was better than the best deterministic subwords by 11.5% absolute WER on the 5 hour set; this difference is only 5.6% on the 10 hour training set. These results again confirm the hypothesis that subword regularization is particularly useful in low-resource cases. We also see that the optimal α setting is lower - meaning more regularization used - for the 5 hour set than the 10 hour set.

Subword regularization can also be thought of as a semi-supervised learning technique in that we are using information gleaned from a large text corpus to help our ASR training. For this reason, it was important to demonstrate that the gains from subword regularization are additive with the inclusion of a language model in decoding. This can be seen in the last column of 6.4.

6.5.3 Babel Languages

$\|V\|$	α	WER	CER
chars		58.9	30.4
200	∞	57.8	30.0
	2	55.7	29.4
	1	54.7	27.7
	0.5	55.5	28.3
1000	∞	57.3	31.0
	5	55.5	30.1
	2	54.1	29.4
	1	54.3	28.4
	0.5	55.5	27.9
4760	∞	58.2	32.6
	2	55.1	29.1
	1	**52.9**	26.6
	0.5	53.4	**26.4**

Table 6.5: Subword regularization results on the Babel Dholuo corpus.

Subword regularization results on the Dholuo corpus are in Table 6.5. There is not much benefit from just using subword units (rows with $\alpha = \infty$, compared to the character baseline), but we can get big improvements with regularization. We see again that a small vocabulary (in this case 1000 units) is best with $\alpha = \infty$, but a larger vocabulary works better with subword regularization.

Subword regularization results on the Swahili corpus are in Table 6.6. These results differ slightly from our previous results. While our main findings that subword regularization works better than deterministic subwords and that subword regularization works well with large vocabularies hold, we see here that deterministic subwords also work well with the large vocabulary in this case.

$\|V\|$	α	WER	CER
chars		74.6	42.4
1000	∞	69.8	41.6
	1	66.8	38.1
	0.5	65.7	36.1
	0.2	69.5	37.5
5910	∞	68.6	40.9
	2	66.0	38.9
	1	**65.2**	36.9
	0.5	66.1	**35.9**
	0.2	68.0	36.0

Table 6.6: Subword regularization results on the Babel Swahili corpus.

With both of these low-resource languages, across the different vocabulary sizes tested, we see that the best WER and best CER are achieved with different choices of α, with more regularization giving a better CER and less giving a better WER.

6.6 Analysis

In this section, we would like to understand more about the mechanism through which subword regularization works so well. We have so far presented subword regularization first and foremost as a regularization technique with the main goal of penalizing confident output distributions. However, it is possible that subword regularization is actually changing the way the model uses subwords, relative to the setting with $\alpha = \infty$. There is no reason to think that the best subword segmentation according to a unigram language model trained on text is the best subword segmentation for ASR. Subword regularization exposes the model to other possible segmentations - does the ASR model ever learn to use those?

For this analysis, we focus on the models trained on the Librispeech 20 hour subset, using the 7775 unit vocabulary. To understand how the model is using subword units, we look at the decoded validation data, counting how many different ways each word is segmented and whether the most common segmentations line up with the best segmentation according to the unigram model.

In counting how many ways each word is segmented, we find a clear difference:

without subword regularization, 2.0% of the words that appear more than once are segmented more than one way; with subword regularization, that same figure is 33.9%. Likewise, while 94.6% of words are most often segmented with the Viterbi best segmentation when no subword regularization is used, only 63.4% of words are most often segmented that way with subword regularization.

Looking more closely at the words that are not segmented according to the Viterbi segmentation when subword regularization is used, we can see two patterns as to when this happens. The first is when the Viterbi segmentation does not separate off a common prefix or suffix. For example, according to the unigram model, the best segmentation for the word ITS is the unit ITS, and the best segmentation for the word THINGS is the sequence TH_INGS. In both cases, the ASR model prefers to have the 'S' character as its own unit, most often outputting IT_S and TH_ING_S.

The second pattern are words where the Viterbi segmentation places a break in the middle of a sound. For example, while the unigram model segments the word ANYTHING as ANYT_HING, the ASR model trained with subword regularization prefers ANY_TH_ING. Without subword regularization, the ASR model uses the Viterbi segmentation. This particular example seems like it could be an artifact of the choice to set $k = 4$, but there are many other words where the unigram model separates 'T' and 'H', including EITHER, THINK, THOSE, and NOTHING. These other words are sometimes segmented with the 'T' and 'H' together by the regularized ASR model, but more often they use the Viterbi segmentation. This finding motivates the work in the next chapter, where we explore whether it is possible to let the ASR model learn its own subword segmentations, without biasing it towards a model learned from text.

6.7 Chapter Summary

In this chapter, we have demonstrated that the subword regularization technique, originally developed for machine translation, transfers well to attention-based end-to-end ASR. As hypothesized, subword regularization is particularly useful in low-

resource scenarios. Our best model achieves a 17.7% relative WER reduction on the 5 hour WSJ subset. Our results suggest that, while smaller subword vocabularies (500-1000 units) work best when deterministically sampling subword units, subword regularization is most effective with larger vocabularies. Finally, we have shown that subword regularization combines well with the use of an external language model.

These findings also extend to two true low-resource languages, Dholuo and Swahili. In both cases, subword regularization improves the baseline performance by more than 10% relative WER. In the case of Swahili, the best model trained with subword regularization performs better than a character-based model training with data from multiple languages (Cho et al., n.d.). For Dholuo, we recover more than half of the difference between our baseline and a multilingual end-to-end model that also makes use of a pronunciation dictionary (Rosenberg et al., 2017).

Our analysis demonstrates that subword regularization sometimes allows the ASR model to choose segmentations that are more aligned with phonetics than the segmentations chosen by the unigram model learned from text. Still, the regularized ASR model more often uses the text-based segmentation. In the next chapter, we will introduce a loss function that allows the ASR model to explicitly choose the subword segmentations that are most useful for the ASR task.

Chapter 7

Discovering an ASR-Specific Subword Inventory

7.1 Motivation

In the previous chapter, we discussed the use of subword units as the output vocabulary for end-to-end ASR. We showed that the subword regularization technique, in which the subword sequence for a given utterance is sampled from a unigram language model, can significantly improve ASR performance especially in low-resource scenarios. Our analysis showed that the subword segmentations produced by the unigram language model do not always correspond well with phonetic or syllable boundaries, and that subword regularization allows the ASR model to sometimes choose more natural segmentations.

In this chapter, we argue that the discovery and usage of subword units in end-to-end ASR should be dependent on the ASR task itself. Rather than learn a vocabulary of subword units and a model to sample from a text corpus, we will learn those things as part of the ASR training process. Based on the analysis from the previous chapter, as well as the past success of using phonetic output targets for ASR training, we hypothesize that these learned subword units will capture predominantly phonetic information, compared to the primarily semantic information that can be captured from text only. Part of the work of this chapter is to analyze which units the ASR model

chooses to use and to understand to what extent capturing phonetic information in the output targets is useful for ASR performance.

This work builds on the latent sequence decomposition (LSD) framework (Chan, Zhang, et al., 2016), in which the use of subword units for ASR is dependent on both the speech input and text output, because the subword sequences used for training are sampled from the output distributions of the ASR model itself. We follow their method for using subword units but introduce an updated loss function that improves the ASR model's ability to perform unit discovery. With this loss function, called n-gram maximum likelihood loss, we simply maximize the combined probabilities of all valid n-grams at each time-step. We show that this loss function outperforms standard maximum likelihood loss within the LSD framework, particularly when we allow the model to choose from a larger starting vocabulary of subword units, and that this finding extends to low-resource scenarios.

We also experiment with two alternatives to LSD that enable us to train our models more efficiently: uniform sampling and statistical model sampling. While uniform sampling is the fastest technique tested, it also degrades performance. Statistical model sampling, by contrast, is faster than LSD but performs equally well when combined with the n-gram loss.

End-to-end ASR models trained with n-gram maximum likelihood loss combine much better with an external language model than the same models trained with either character outputs or subword regularization. This finding suggests that this loss function is enabling the model to learn primarily about how subword units correspond to sounds, with the additional language model providing the semantic information not learned during ASR training.

Along with our quantitative results, we present analysis of the subword units used by these models. We show that models trained with the n-gram loss learn to use only a subset of the available subword units. As a contrast to the subword regularization technique discussed in Chapter 6, we refer to the process of training a model with the n-gram loss as performing "subword discovery." We demonstrate a clear correlation between the chosen subword units and English phonetics; the learned

subword inventory is strongly influenced by the properties of speech in addition to text.

7.2 Background

7.2.1 Graphones

In this work, we allow an end-to-end ASR model to learn the best subword inventory for the ASR task. We hypothesize that this model will learn acoustically-inspired units that capture the correspondence between phonology and orthography. There is a related line of research that has explored the learning of joint phoneme-letter units for HMM-based models (Deligne and Bimbot, 1997; Deligne, Yvon, et al., 1995). These units are called 'multigrams' or 'graphones,' and can enable recognition of words that do not have an entry in the pronunciation dictionary (Bisani and Ney, 2003; Bisani and Ney, 2008). That work differs from ours in that learns these correspondences from a pronunciation dictionary which contains paired sequences of phonemes and letters; we instead learn directly from the acoustic signal.

Graphones are learned through the EM algorithm, with the goal of finding the set of units and associated probabilities that maximizes the likelihood of the training data. This model can then be used to find the best segmentation for the training data; this segmented data can be used to train an n-gram graphone language model. Using this model, it is now possible to decode an unknown word into a graphone sequence that defines its most likely pronunciation. This can then be used to augment a pronunciation dictionary with this new word.

7.2.2 Gram-CTC

Gram-CTC (Liu et al., 2017) is a technique for CTC-based models that is similar in spirit to our work. In that paper, the authors modify the CTC loss function to marginalize over all valid subword sequences. For example, the word "OF" could be produced either by individually outputting the characters "O" and "F" or by

outputting the unit “OF". This is closely related to our decoding method, which similarly considers all possible segmentations of the same word sequence. Liu et al. (2017) also use Gram CTC as a subword unit discovery method, starting with all possible subwords up to a given length and then iteratively selecting the units most often used by their trained model during greedy decoding. Unfortunately, while this technique does improve performance, the authors note that it is difficult to train with a large vocabulary and thus limit their experiments to a vocabulary of only 100 bigrams along with the original character set.

7.2.3 Latent Sequence Decomposition

This work builds on the latent sequence decomposition (LSD) framework, in which subword decompositions are sampled from the ASR model itself and thus rely on both the input speech and output text.

When the output vocabulary is made up only of characters, there is exactly one possible decomposition of any particular sentence. Once subword units of length 2 or more are introduced, there are several possible valid decompositions of most sentences. The “best" decomposition to use for training is unknown; latent sequence decomposition (LSD) seeks to learn these decompositions as part of the ASR training process, so that they depend on both speech and text (Chan, Zhang, et al., 2016).

The LSD problem is formulated as follows. Z is the set of valid decompositions of text y. The goal is to maximize the likelihood of y by marginalizing over all z in Z:

$$log(p(y|x)) = log \sum_{z} p(y|z)p(z|x)$$

Computing the gradient of this version of maximum likelihood loss is intractable, but can be approximated by taking an expectation of $log(p(z|x))$ over $p(z|x, y)$. In practice, this means sampling a subword decomposition z from the ASR model and performing standard maximum likelihood training using z as the label sequence. Unfortunately, exact sampling from $p(z|x, y)$ is also very difficult. LSD instead samples z greedily, left-to-right, one unit at a time. This is a heuristic method that is not,

initially, a good approximation of $p(z|x, y)$.

To correct for this, LSD follows an ϵ-greedy exploration strategy (Sutton and Barto, 1998). Instead of always sampling from the output distribution of the model, LSD samples from a mixture of that output distribution and a uniform distribution. ϵ is the mixture weight of the output distribution - it starts at zero and ramps up over the course of training. The specifics of this ramp up are not discussed in detail in Chan, Zhang, et al. (2016) - part of this chapter is exploring a range of potential search strategies for LSD.

7.3 N-gram Maximum Likelihood Objective

7.3.1 Method

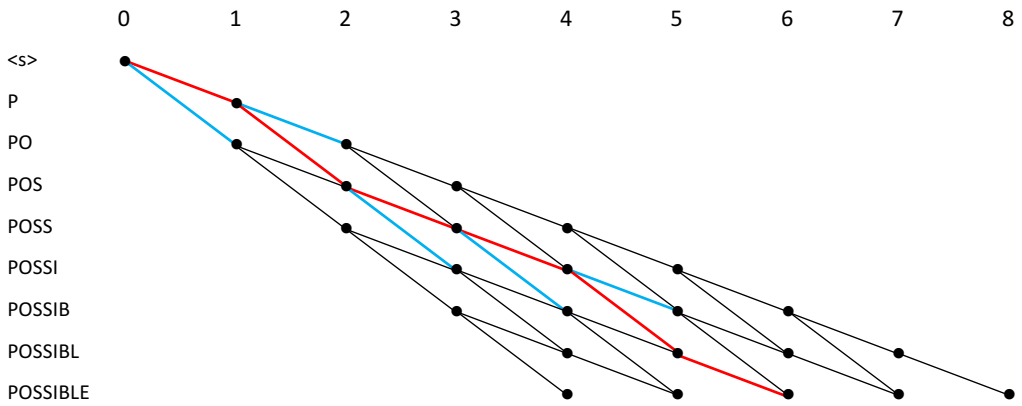


Figure 7-1: Lattice depiction of all of the valid subword segmentations of the word POSSIBLE into single characters and character bigrams. Each column is one time-step; each row indicates the character sequence that has already been decoded. The red path represents the sequence P_OS_S_I_BL_E.

The process of decoding a sequence of characters using subword units can be thought of as a lattice. An example is shown in Figure 7-1; in this case, we are decoding the word POSSIBLE with a vocabulary that includes individual characters and 2-character units. Each column represents one decoding time-step, each row represents the character sequence that the decoder has already seen. Any left-to-

right path through this lattice is a valid sequence of decoder outputs. The red path, for example, indicates the output 'P OS S I BL E'.

One way for the ASR model to be agnostic as to the subword segmentation chosen by the model is to marginalize over the entire lattice of valid subword sequences. However, because the attention-based model is autoregressive, two decoders that reach the same point in the lattice by following different paths will be in different states. This makes it intractable to marginalize over the entire lattice. LSD approximates this marginalization by sampling a single path on each iteration and performing maximum likelihood training using that sampled sequence as the labels.

Our method in this chapter starts the same way as LSD; we sample a single path through the lattice to use as the decoder input. However, instead of using that path as the fixed target labels for training, we introduce a new loss function that marginalizes over the valid subword units at each time-step. For example, if we sampled the red path in Figure 7-1, our method would allow the model to consider both the red transitions and the blue transitions.

The standard maximum likelihood loss for end-to-end ASR is:

$$L(x, y) = -\sum_{t=1}^{T} log(p_t(y_t|x, y_{1:t-1}))$$

where x is the sequence of speech feature inputs and y is the sequence of target labels.

For this section, we define the set of valid output labels at time t as $n_t = \{n_t^1, ..., n_t^k\}$ where k is the maximum length of subword units in the vocabulary. The sampled subword unit sequence fed into the decoder is $z_{1:t-1}$. The n-gram maximum likelihood loss is, then:

$$L(x, z) = -\sum_{t=1}^{T} log(\sum_{j=1}^{k} p_t(n_t^j|x, z_{1:t-1}))$$

with the added condition that $p(n|x, z) = 0$ if n is not in the vocabulary. Thus, the n-gram maximum likelihood objective decouples the decoder inputs and the training labels: the sampled sequence is still used as decoder input but the objective function is agnostic as to which of the valid subword units the model outputs at each timestep.

In Chan, Zhang, et al. (2016), the authors found that sampling from the output distribution from the beginning of training produced a model that used only character outputs; we find that the same is true when the n-gram loss is used. For this reason, we use ϵ-greedy search with the n-gram loss. When testing both standard LSD and LSD with the n-gram loss, we also experiment with ϵ-greedy search in which the final ϵ is less than one.

7.3.2 Experimental Details

All English experiments in this chapter use the Wall Street Journal corpus. We experiment with the full 81 hour training set as well as the 5- and 10-hour subsets.

Following Chan, Zhang, et al. (2016), we create subword vocabularies by simply taking the most common n-grams in the training corpus. We experiment with two vocabulary sizes, 512 and 5111, using $k = 4$ with both. The smaller vocabulary size was chosen to match the experiments in Chan, Zhang, et al. (2016); the larger was reached by selecting all n-grams that appear 100 times or more in the training corpus. As in Chan, Zhang, et al. (2016) and Drexler and Glass (2019b), we always use <SPACE> as a stand-alone unit, not included in any subwords.

Chan, Zhang, et al., 2016 does not include details of the ϵ-greedy search parameters - how often ϵ is updated or at what rate it ramps up from zero to one. For all LSD results reported in this section, we used a simple linear ramp over 100 epochs, approximately the number of epochs that were required to fully train the baseline character-based model. We experimented with three different stopping points for our ramp: $\epsilon = 1.0$, $\epsilon = 0.8$, $\epsilon = 0.5$.

7.3.3 Results

Table 7.1 shows the results of our implementation of LSD, experimenting with two different vocabulary sizes and three choices for the final ϵ used in the ϵ-greedy sampling. The first line is the character baseline, which outperforms the very similar model in Bahdanau et al. (2016) (WER 18.0) but underperforms the baseline from Chan,

	final ϵ		
$\|V\|$	1.0	0.8	0.5
31	16.7		
512	15.5	15.6	16.2
5111	16.5	15.7	15.8

Table 7.1: Word error rate for models trained on the full 81 hour WSJ corpus using latent sequence decomposition.

Zhang, et al. (2016) (WER 14.8). There are two main differences between our model and the baseline in Chan, Zhang, et al. (2016): they use much higher-dimensional input features (including deltas) and their encoder downsamples the input by a factor of 4 rather than 8. Nonetheless, we are able to demonstrate similar improvements from LSD as in Chan, Zhang, et al. (2016). We see that with the smaller vocabulary it is better to ramp all the way up to $\epsilon = 1.0$, while the larger vocabulary works better with $\epsilon < 1.0$. It is likely that we have not found the optimal parameters for this sampling - perhaps the larger vocabulary would work well with $\epsilon = 1.0$ if we lowered the slope of the ramp. We also compared our models to the models described in Chan, Zhang, et al. (2016), and found that our LSD models use single characters more often. This also suggests that we increased ϵ too quickly. For the remaining results in this chapter, all experiments use a final ϵ of 0.8.

Loss	$\|V\|$	WER	CER
Standard	31	16.7	6.0
Standard	512	15.6	5.6
	5111	15.7	5.3
N-gram	512	15.2	5.1
	5111	14.5	5.0

Table 7.2: Comparison of standard maximum likelihood loss and our n-gram maximum loss when used with latent sequence decomposition. All models were trained on the 81 hour WSJ corpus.

Table 7.2 compares LSD with standard maximum likelihood loss and n-gram loss. All of our subword models outperform the character baseline. N-gram loss significantly improves LSD performance over the standard loss. Our results do corroborate the assertion in Chan, Zhang, et al. (2016) that LSD is not much impacted by the

size of the subword vocabulary. However, the n-gram loss is able to effectively use the extra available subwords in the 5111 unit case compared to the 512 unit case, resulting in our best overall score of 14.5% WER.

7.3.4 Analysis

We perform a similar analysis to that in the previous chapter, looking at the segmentations chosen by the models trained with the n-gram loss. We do this by looking at the output when we decode the WSJ validation set. Whereas the model trained with subword regularization segmented 33.9% of words that appeared more than once in multiple ways, the subword discovery models do this for 64% (512 units) or 65.9% (5111 units) of words. Looking closely at these segmentations, we can calculate what percentage of the time particular units are used, in this case focusing on the model with 512 output units. For example, there are 895 instances of the letter sequence TH, and they are only split apart 44 times, which is 4.9%. By contrast, the letter sequence TS occurs 146 times and they are split apart 123, or 84.2%, of those times. Unexpectedly, the letters SH are split apart 62.5% of the time. This highlights the importance of letting the ASR model choose the subword units that are most useful in practice, rather than imposing phonetic categories on the model in a top-down fashion.

7.4 Alternative Segmentation Sampling Methods

LSD has one key downside: it is quite slow to train, because the sequence of decoder inputs is sampled within the decoder loop. We experiment here with two methods of speeding up this decomposition. First, we simply sample from a uniform distribution. While this method does not work well with the standard maximum likelihood loss, it is comparable to LSD when the n-gram maximum likelihood loss is used. This is because the n-gram loss essentially decouples the sequence of units used as input to the decoder from the sequence of units used as target labels for computing the objective function. If the n-gram loss enables us to train with any decomposition

strategy, we should instead be able to use something that runs more quickly.

The second decomposition strategy we test here is to approximate the ASR model outputs with a statistical model. Then, instead of sampling from the ASR model inside of the decoder loop, we can quickly sample a segmentation from the statistical model prior to training. The statistical model is frequently re-estimated, so that it keeps up with the training of the ASR model.

7.4.1 Statistical Model Sampling

Unlike uniform sampling, sampling from the output distribution enables us to focus over time on the most important parts of the space - the decompositions that are likely to be seen when decoding new audio. While uniform sampling is simple and fast, we unnecessarily cover the entire space which means that the model requires many more training iterations to converge.

In this section, we introduce a third option that combines the best features of LSD and uniform sampling. For this technique, we train a statistical model to approximate the output distributions of the ASR model. This will enable us to sample decompositions of text into speech-inspired subword units outside of the decoder loop. We re-estimate this statistical model frequently so that it stays up-to-date as ASR training progresses.

The statistical model that we train is an n-gram model. It is slightly different than a typical n-gram language model because our task is segmentation rather than prediction - we are not trying to predict the next subword token but to choose one subword token out of a small set that is defined by the text to be segmented. For this reason, this statistical segmentation model is conditioned on both previous and future characters. The number of characters in each direction that we condition on is a parameter of the model; we have found that it works well to set that parameter equal to k, the maximim length of subword units in the vocabulary. This can be modified, but the number of future characters that we condition on must be at least k to ensure that we can select subword units of length k.

This statistical model is estimated on a subset of the training data. The first step

in this estimation is to force-decode this data; this means feeding the outputs of the decoder back in as input like we do during inference, but only allowing the model to output the correct character sequence. To encourage variety in these force-decoded outputs, we sample each unit from the output distribution, rather than selecting the most likely unit. As an example, imagine we are decoding speech of the phrase THE MANAGEMENT, and our force decoding output is TH_E M_A_N_AG_E_M_ENT. At each step in this decoding process, we save both the current context and the output distribution that our model sampled from. In this example, at the beginning of this decoding, the prior context is empty (we are at the start of a word), and the future context is THE. The ASR model produces a distribution over all possible output units at each timestep, but we save a normalized version of the distribution over the valid output units only. At this first timestep, the distribution we are saving is over the units T, TH, and THE. Example output distributions for the force-decoding of this phrase are in Table 7.3.

t	prev	next	$P(n_t^1)$	$P(n_t^2)$	$P(n_t^3)$	$P(n_t^4)$	samp
1		THE	0.03	0.84	0.13		TH
2	TH	E	1.00				E
3		MANA	0.86	0.14	0.00	0.00	M
4	M	ANAG	0.98	0.02	0.00	0.00	A
5	MA	NAGE	1.00	0.00	0.00	0.00	N
6	MAN	AGEM	0.04	0.93	0.03	0.00	AG
7	ANAG	EMEN	1.00	0.00	0.00	0.00	E
8	NAGE	MENT	0.41	0.01	0.37	0.21	M
9	AGEM	ENT	0.00	0.01	0.99		ENT

Table 7.3: Force-decoding the phrase "THE MANAGEMENT". t is the decoding timestep, $P(n_t^k)$ is the output probability at time t of the valid output unit that is k characters long.

Once we have done this for all of the data we are force-decoding, the statistical model that we save is the average over all of the times we saw the same past and future context. For example, imagine we saw the word THE one other time during this decoding, and the second time the output of the ASR model had $P(T) = 0.01$, $P(TH) = 0.98$, and $P(THE) = 0.01$. Averaging this with the distribution in the first row of Table 7.3, we would save: $P(T| < s >, THE) = 0.02$, $P(TH| < s >, THE) =$

0.91, and $P(THE| < s >, THE) = 0.07$. In practice, we set a minimum count, so we only save these probabilities for contexts that appear at least M times in the force-decoding. It is also important to note that these probabilities are conditioned on being at the start of a word and having the future context THE. These probabilities would not be used if we were at the start of a longer word beginning with THE like THEN or THERE, or in the middle of a word that ended THE.

prev	next	$P(n_t^1)$	$P(n_t^2)$	$P(n_t^3)$	$P(n_t^4)$	C	samp
	COUP	0.98	0.02	0.0	0.0	32	C
C	OUPL	1.0	0.0	0.0	0.0	15	O
CO	UPLE	0.27	0.73	0.0	0.0	15	UP
SAMP	LED	0.38	0.62	0.0		5	
TRIP	LED	0.06	0.94	0.0		6	
*P	LED	0.21	0.79	0.0		11	LE

Table 7.4: Segmenting the word COUPLED into C_O_UP_LE_D. $P(n_t^k)$ is the output probability of the valid output unit that is k characters long.

When segmenting new text, we sample greedily from left to right. For contexts that we did not see during force-decoding, we use a simple backoff scheme. An example of the process for segmenting the word COUPLED is in Table 7.4. The first three segmentation steps are straightforward: we have probabilities saved for these contexts in our statistical model, and we can sample from them. On the fourth step, however, the past context is COUP and the future context is LED. We did not see this context when training our statistical model - the training data had the word COUPLE but not COUPLED. To get a probability distribution to sample from, we remove the first character of the past context and look for any saved probabilities for contexts of the form (*OUP, LED). We do not find any of these, so we repeat the process with the first two characters of the past context removed. None of these contexts exist in our model, so we remove the third character from the past context. Here, we find two examples of the form (*P, LED): (SAMP, LED) and (TRIP, LED). We take a weighted average of these two distributions to get the backoff distribution for (*P, LED) that we can sample from. If we had been unable to find any examples of the future context LED, we would remove the last character and repeat the backoff

process, looking for contexts of the form (COUP, LE*), then (*OUP, LE*) and so on.

7.4.2 Experimental Details

We re-estimate the statistical model every 10 epochs using a randomly selected subset of 5000 utterances from the training set (or the full training set, if it has fewer than 5000 utterances, as in the case of the 5 and 10 hour WSJ subsets). We set the minimum count to 5 for all estimated distributions in the statistical model. We use ϵ-greedy sampling when segmenting text with the statistical model, with the same ramp as we used for LSD.

7.4.3 Results

$\|V\|$	LSD	Uniform	Stat
512	15.2	15.2	14.9
5111	14.5	15.1	14.5

Table 7.5: Word error rates for models trained with n-gram loss on the 81 hour WSJ corpus, compared across decomposition strategies.

The effectiveness of uniform sampling when combined with the n-gram loss seems to depend on the size of the vocabulary. While uniform sampling performs just as well as LSD with the smaller vocabulary, it significantly degrades performance on the larger vocabulary. It appears that uniform sampling does not allow the model to effectively learn to use rare subword units.

The statistical model, however, slightly outperforms the original LSD model, in addition to training more efficiently. This difference could be due to two potential factors: the stability of the statistical model, which is re-estimated less often than the neural network parameters are updated, and the fact that all of the probabilities in the statistical model are averaged over several instances of the same word. Both of these should prevent the model from overfitting to the pronunciations of specific speakers in the training set.

7.4.4 Analysis

In this section, we conduct our analysis of the models' subword usage by greedily sampling a valid decomposition for every training utterance. Instead of analyzing the decompositions themselves, we save the output distributions that we sample from at every time step. Once this is complete, we can analyze the set of output distributions over each set of valid subwords. For example, we can look at every time the model was asked to choose between 'THER', 'THE', 'TH', and 'T' and produce an average distribution over those units.

When the next four characters of the text are 'THER', as in the above example, the unit 'TH' has an average likelihood of 80.2% and the unit 'THER' has an average likelihood of 19.5%. We will say that the model is 'using' a particular unit if, in any context, that unit has an average likelihood greater than 10%. We set this threshold intentionally low so that all units that are not being 'used' could be removed from the model with minimal expected impact.

$\|V\|$	Model	2-gram	3-gram	4-gram	Total
512	LSD	175	122	23	320
	Uniform	170	99	5	274
	Stat.	130	80	3	213
5111	LSD	319	886	433	1638
	Uniform	324	744	193	1261
	Stat	237	530	188	955

Table 7.6: Number of subword units "used" by models trained with the n-gram loss. The model is 'using' a particular unit if, in any context, that unit has an average likelihood greater than 10%.

First, in Table 7.6, we look at how many units the different models are using. All four models are using many fewer subword units than are available. With both vocabularies, the models trained with uniform sampling use fewer units than those trained with LSD. As noted above, uniform sampling does not degrade the performance of the model with smaller vocabulary, suggesting that, in this case, LSD has selected some units that are unnecessary. However, uniform sampling significantly degrades the performance of the model with bigger vocabulary, suggesting that there

may be some useful units that the uniform model is not using.

Interestingly, the models trained with statistical model sampling use the fewest number of units. This is likely due to the minimum count we use to estimate our statistical model. We do not save probabilities for contexts that appear fewer than five times in the training set; instead, we use back-off probabilities for segmentation in this case. The models trained with statistical sampling perform best out of the models tested, suggesting that these rare contexts are not necessary for generalization performance.

Given the black-box nature of end-to-end neural network models, an important research question is to what extent these models' choice of units corresponds to our understanding of how language works. Are these models using units that are phonetically meaningful?

We will focus on the letter 'C' as an instructive example - it is a letter that, in English, can denote many different sounds depending on context. When force-decoding with the large-vocabulary LSD model, we look at all of the places where C is the next letter in the sequence to be sampled. There are 228 different contexts in which C is the next letter; in most (171) of these, the unigram letter C has an average likelihood greater than 0.5. In 35 contexts, the model prefers a bigram unit, but these contexts include just six bigrams: CE, CH, CK, CT, and CY. These clearly demonstrate the different ways in which the letter C can be used - the letter C and the bigrams CK and CT indicate the phoneme /k/, CE and CY indicate /s/, and CH indicates /t/. There are 8 other bigrams that start with C in the large vocabulary, but none are ever used.

CH is a further interesting example because it can, in fact, be used to denote either a /t/ or /k/. Our model seems to have learned this: there are contexts in which the model is most likely to output the letter C even though the next two characters are CH. These contexts include CHRY (as in Chrysler), CHNO (as in technology), CHRI (as in Christian or Christopher), and CHEM (as in chemistry) - all contexts in which the letters CH indicate the phoneme /k/.

7.5 Low-Resource Experiments

Before embarking on these experiments, we hypothesized that the n-gram loss would help in some low-resource settings (say, 20 hours of training data) but would hurt performance in extremely low-resource settings where there would not be enough data to effectively learn a vocabulary of subword units. This hypothesis did not hold: we found that subword discovery with the n-gram loss performs better than subword regularization even with 5 hours of training data, and leads to bigger improvements on the 5 hour corpus than the 10 hour corpus.The results in this section all use the statistical model decomposition technique.

On the 5 hour WSJ subset, the baseline WER was 65.1% and the best subword regularization model had a WER of 54.3%. Using subword discovery (with an initial vocabulary of 512 subword units), the WER was reduced to 50.2%. The baseline and subword regularization models had CERs of 35.3% and 24.1%, respectively. Here, we get a CER of 20.3%.

On the 10 hour WSJ subset, the baseline WER was 39.5% and the best subword regularization model had a WER of 36.0%. Using subword discovery, we were able to get a WER of 35.4%. The improvement in CER from subword regularization to subword discovery is larger: 14.9% versus 13.2%. The improvement in character error rate seems to translate to much better results when combined with an external n-gram language model. With the addition of this language model, the best subword regularization methods achieved 29.9% WER and 15.8% CER. The subword discovery model performs much better with the language model: 25.0% WER and 11.5% CER.

This result supports the original motivation for learning acoustically-inspired subword units. Subword units learned from text capture mainly semantic information, and cannot capture phonetic information. We expected it would be easier to train a low-resource ASR model with a set of subword units that would instead represent predominantly phonetic information. The large improvement from the addition of a language model to the subword discovery model suggests that the learned subwords are not capturing as much semantic information as the text-based subword units are.

In the next chapter, we will explore whether these units make it easier to construct a shared embedding space for speech and text.

7.6 Babel Language Experiments

We use statistical model sampling for all experiments with the Babel corpora. We present results for two languages here, Dholuo and Swahili. In both cases, subword discovery improves performance over the character baseline but does not perform as well as subword regularization. On the Dholuo corpus, subword discovery with 5638 units (all units that occur at least ten times in the training set) gives a WER of 56.3% and CER of 28.1%. This is worse than all subword regularization settings in terms of WER and most in terms of CER. Similarly, subword discovery with 7655 units on the Swahili corpus gives a WER of 67.2% and CER of 37.1%, worse than most subword regularization settings on that corpus.

We hypothesize two reasons for this discrepancy. First, we have chosen to use Babel languages that use the Latin alphabet - the orthography of these languages is likely largely phonetic. Given this, characters likely correspond well to the acoustic input, making speech-inspired subword units less necessary.

It is important to note that, while subword discovery lets the ASR model choose which output units to use, it is not doing any active regularization on the output distributions; the model can learn extremely confident output distributions. Subword regularization, by contrast, is necessarily doing regularization. Thus, if subword units are not particularly helpful for ASR in these languages, subword regularization should perform better than subword discovery.

7.7 Chapter Summary

In this chapter, we have introduced a new loss function, the n-gram maximum likelihood loss, that yields significant improvements over a character-based model. We build on the LSD framework, demonstrating that models trained with the n-gram

loss outperform other LSD models with two different subword vocabularies. Additionally, we propose a modification of LSD in which we train a statistical segmentation model to approximate the ASR model outputs. This statistical model both speeds up training and further improves performance.

Analysis of our English models suggests that the n-gram loss enables the ASR model to effectively learn which subword units to use, and that these units correspond to phonetic categories. In our low-resource experiments, we demonstrate this subword discovery method is more effective than subword regularization with text-based subword units. We see especially stark improvements on the low-resource training sets when we decode with an external language model. This finding provides further evidence for the importance of phonetic output units when limited training data are available.

This technique also outperforms our character baselines on the Babel languages, but does not work as well as subword regularization in those cases. We hypothesize that this is because their orthography is largely phonetic.

Chapter 8

Using Subword Units to Construct a Shared Embedding Space

In Chapters 4 and 5, we outlined two different methods for creating a shared embedding space for text and speech. While both methods improved performance in both fully- and semi-supervised scenarios, analysis demonstrated that we were not able to fully align the embedding spaces with either technique. Our hypothesis is that this is because individual characters do not represent the same information contained in the speech signal. A set of speech feature frames captures a particular sound, an individual character can represent a number of sounds, depending on context. In Chapters 6 and 7, we showed that we can improve ASR performance using subword units, particularly ones learned as part of the ASR training process.

This chapter will explore the construction of a shared text/speech embedding space using subword units instead of characters. We expect subword regularization and subword discovery to both work well with adversarial training, because the text space will represent a huge number of subword units and some of these will capture specific sounds and line up well with the speech embeddings. With direct alignment, we do not expect subword regularization to work as well, because we are asking the speech embeddings to match up with embeddings of text units that may or may not capture acoustic/phonetic information.

This chapter is divided into two sections. The first focuses on the adversarial

alignment method introduced in Chapter 4 and the second section looks at the combination of subword units with the direct embedding alignment method from Chapter 5. In both cases, we compare the results from those earlier chapters with new results using the same alignment methods but segmenting the text into subword units. In both cases, we explore several different settings related to the use of subword units to create a text embedding space. Ultimately, we find that subword units improve the performance of both methods, with speech-inspired subword units performing best overall.

8.1 Subword Adversarial Alignment

8.1.1 Methods

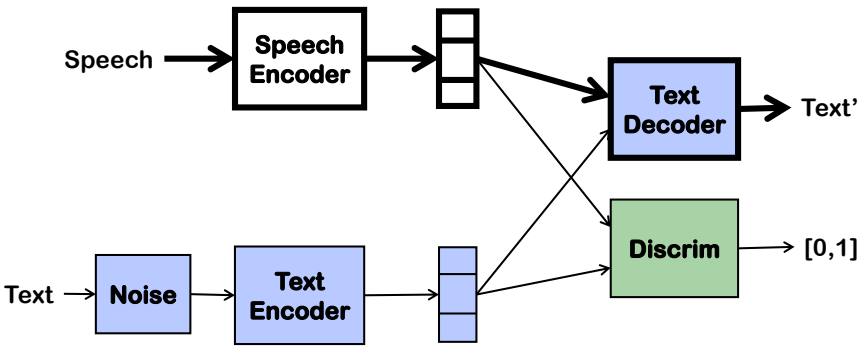


Figure 8-1: Simplified semi-supervised adversarial ASR model architecture used in this chapter. Speech-to-text model is outlined in bold; text autoencoder is shaded blue; speech autoencoder is shaded red; discriminator for adversarial training is shaded green.

For this chapter, we use a modified version of the architecture presented in Chapter 4. This architecture is shown in Figure 8-1; it does not have the speech autoencoder component. In Chapter 4, our ablation studies showed that the speech autoencoder did not contribute much to the overall performance, and our goal in this chapter is to focus on the use of text data.

In this adversarial alignment model there are a number of different places where we segment text into subword units. Each time we see a text utterance, we sample a new segmentation for it. This includes both the input and output of the text autoencoder. We experiment with removing the noise component of the text autoencoder (setting the probability of deletion to 0.0) for this model, because we are already making the text autoencoder task much harder by having different input and output sequences.

For subword discovery, we always used the n-gram loss for ASR training, and tried both the standard maximum likelihood loss and the n-gram loss for the text autoencoder training. With each of these, we tested two methods for subword discovery. The first condition, referred to in the results tables as $p_0 = 0.0$, is very similar to the statistical model sampling in Chapter 7. For this condition, we used greedy uniform random sampling to segment all text at the beginning of training; every ten epochs, we used the current ASR model parameters to train a new statistical text segmentation model. we sampled from this model with probability p and from a uniform distribution with probability $1 - p$, where p started at zero at the first epoch and was increased by 0.05 every ten epochs. For the other set of experiments, we used the statistical segmentation model from the final baseline ASR model at the beginning of training. This condition is indicated with $p_0 = 0.8$, meaning that the segmentation is sampled from the pre-trained statistical model with $p = 0.8$ at the beginning of training. After 160 Epochs, when p would have reached 0.8 in the $p_0 = 0.0$ condition, we begin re-estimating the statistical model every 10 epochs.

8.1.2 WSJ Results

Table 8.1 contains results for the combination of subword regularization and adversarial embedding space alignment. These results use the 5 hour WSJ subset and no other data. We vary the both α parameter for subword regularization and the probability of deleting each text input unit in the noise model.

The first line of results in this table come from Chapter 4. The second line of results uses deterministic subword segmentation; these results are worse than using characters. This could be because the text-based model is choosing the wrong sub-

$\|V\|$	p_{del}	α	WER	CER	+ LM WER
31	0.1	N/A	58.6	27.8	51.1
500	0.1	∞	61.5	35.3	59.8
		1	54.1	25.0	46.5
		0.5	53.3	23.6	44.7
	0.0	1	56.1	25.3	45.9
		0.5	54.8	24.0	44.1

Table 8.1: Results of adversarial alignment with subword regularization on the 5 hour subset of the WSJ corpus

words and thus making alignment even harder. It could also be because the noise model makes the text autoencoder too hard in combination with subwords: deleting a 3- or 4-character unit might force the encoder to represent a lot of (implied) context, which could actually make constructing the shared embedding space more difficult.

In the next two lines of Table 8.1, we see that we are able to improve upon the performance of the character-based model with subword regularization. These results are also slightly better than the baseline model with subword regularization; the WER with the vocabulary and $\alpha = 0.5$ was 54.3% without a language model and 47.8% with one. Interestingly, the performance improvement is even larger with the language model than without.

Finally, in the last two lines, we have the performance of this model with subword regularization but without input text units deleted. While these results are worse than those in the previous section when a language model is not used, these models are slightly better when combined with an external language model. It makes sense that the model would do better with the noise: the text autoencoder is forced to learn more about semantics and do more language modeling in that case. It also makes sense, then, that an ASR model that does better at language modeling would have less to gain from the addition of an external language model. What is interesting is that the second set of models outperforms the first when the external language model is added. This echoes our results from Chapter 7, in which models that learned more bout the phonetic content of speech combined very well with external language models.

$\|V\|$	Text Auto. Obj.	(p_0, p_max)	WER	CER	+ LM WER
31	N/A	N/A	58.6	27.8	51.1
512	N-gram	(0.8, 0.8)	51.0	21.5	40.0
		(0.0, 0.8)	52.2	20.9	40.7
		(0.0, 0.0)	52.3	21.5	41.4
	ML	(0.8, 0.8)	50.9	21.6	40.0
		(0.0, 0.8)	52.2	21.0	39.8

Table 8.2: Results of adversarial alignment with subword discovery on the 5 hour subset of the WSJ corpus

Table 8.2 has results on the 5 hour WSJ set for adversarial training with subword discovery. We again show the character-based results in the first line of this table. The next section uses subword discovery, with the n-gram loss used for both ASR training and text autoencoder training. In the last section of the table, we use the n-gram loss for ASR training but the standard maximum likelihood loss for the text autoencoder training. Somewhat surprisingly, we find that this change has minimal impact on the results.

	Parallel Only			Semi-Supervised		
Seg.	WER	CER	+ LM	WER	CER	+ LM
Characters	58.6	27.8	51.1	55.6	24.7	45.1
Subword Reg.	53.3	23.6	44.7	51.9	21.9	41.6
Subword Disc.	51.0	21.5	40.0	48.6	22.0	40.1

Table 8.3: Summary of subword adversarial training results on the 5 hour subset of the WSJ corpus. All results use the same model architecture and training procedure. 'Parallel Only' results use the speech and text data from the 5 hour training set for all parts of model training. 'Semi-supervised' results use all 81 hours of WSJ speech data for adversarial training, and additional text from the LM training data for adversarial training and text autoencoder training.

Finally, in Table 8.3, we take the best performing model settings from the previous two tables and also train semi-supervised models with these settings. These models use additional text and speech for the adversarial training as well as addiitonal text for autoencoder training. Subword discovery performs best overall, although it does not gain much from semi-supervised learning when an external language model is used.

8.2 Subword Direct Alignment

8.2.1 Methods

The direct encoding alignment procedure outlined in Chapter 5 has four steps (see 5.1). To get the best performance out of the combination of this technique with the use of subword units, we found that it was necessary to use different parameters for each of these steps. For both types of subword units, the key modification was to use a more stable subword segmentation setting when performing the alignment steps than when doing ASR training.

For subword regularization, this means using $\alpha = \infty$ during the alignment steps (steps 2 and 3), and a smaller α setting during ASR training. Specifically, we use $\alpha = 0.5$ with the 5 hour subset of the WSJ corpus, since that was the best performing setting with the baseline ASR model. We use that baseline ASR training that we already did for Chapter 6 as our step 1 training for this section.

For subword discovery, we also used a previous model training as step 1 for this section. In this case, we used the model trained with our statistical segmentation procedure. While training that model, we set the maximum language model sampling probability at 0.8, meaning that at the end of training we were still sampling from a uniform distribution 20% of the time. For steps 2 and 3, we set that sampling probability to 1.0. Note that we are still sampling from the language model, so this is not a deterministic segmentation like subword regularization with $\alpha = \infty$. For step 4, we set that probability back to 0.8, also returning to the statistical model estimation procedure used in step 1, in which we re-estimated that model every 10 epochs.

For both subword segmentation methods, we experimented again with text autoencoder training during step 4, using either the text side of the training corpus or additional text data used for LM training. In the case of subword discovery, we tested using both the n-gram loss and standard maximum likelihood loss for this autoencoder training.

8.2.2 WSJ Results

Text	α_1	α_2	α_3	α_4	WER	CER	+ LM
	0.5	0.5	0.5	0.5	55.7	24.5	47.2
	0.5	∞	∞	∞	56.0	24.8	48.4
	0.5	∞	∞	0.5	55.2	24.0	45.8
WSJ5	0.5	∞	∞	0.5	53.2	22.7	43.9
LM	0.5	∞	∞	0.5	52.7	22.2	42.8

Table 8.4: Results from the combination of direct embedding alignment and subword regularization on the 5 hour WSJ subset. The first column indicates the text used for text autoencoder training in Step 4. α_1 is the value of α used for text segmentation during Step 1.

Results for direct alignment and subword regularization are in Table 8.4. All of settings shown in this table work better than using characters, where the best WER (with semi-supervised learning) was 60.5% without an external language model and 59.8$ with one. Using $\alpha = 0.5$ for all steps or using $\alpha = \infty$ for steps 2-4 both work okay, but as noted above it works best to used $\alpha = \infty$ for steps 2 and 3 and $\alpha = 0.5$ for steps 1 and 4. The difference between these three settings is more pronounced when an external language model is used during decoding. As in Chapter 5, we get some improvement from text autoencoder training with the text portion of the 5 hour WSJ subset, and an additional improvement from the use of extra standalone text.

Text	Text Obj.	WER	CER	+ LM
		51.6	20.3	33.7
WSJ 5	n-gram	49.8	19.8	34.5
WSJ 5	ML	49.5	19.9	34.6
LM	n-gram	50.2	19.8	34.4
LM	ML	50.0	19.8	33.8

Table 8.5: Results from the combination of direct embedding alignment and subword discovery on the 5 hour WSJ subset. The first and second columns indicates the text and objective function used for text autoencoder training, respectively.

Results for direct alignment and subword discovery are in Table 8.5. Subword discovery works even better in combination with this alignment technique than subword regularization. Of particular note is the performance when an external language model is added. We saw in Chapter 7 that subword discovery models combine well

with an external language model; we get even more from that combination when we use direct alignment as well. Interestingly, while we get some improvement from the use of text autoencoder training when we do not use an external language model, the best overall result does not incorporate the text autoencoder.

8.3 Chapter Summary

In this chapter, we have explored combining our techniques for creating a shared embedding space for speech and text with our techniques for using subword units in ASR. We have shown that the use of subword units improves the performance of both embedding alignment techniques presented here. The n-gram loss combines particularly well with the direct alignment method, achieving a 33.7% WER on the 5 hour WSJ subset with an external language model, compared to 61.9% WER for the baseline model.

Chapter 9

Conclusion

9.1 Summary

This thesis has explored the problem of training end-to-end ASR models when limited training data are available. End-to-end models are an attractive architecture for low-resource languages because they do not require expert resources for training. However, neural network models are prone to overfitting when trained on small corpora, making them ill-suited to low-resource ASR in their current form.

We have explored two different categories of techniques for improving the performance of end-to-end ASR models in low-resource settings. The first set of techniques were motivated by a desire to perform semi-supervised learning to train some model parameters with non-parallel speech and text. We introduced a general model architecture for this task, adding a text autoencoder that shared decoder parameters with our attention-based ASR model. In order to effectively train these decoder parameters with only text, it is important that the original ASR encoder outputs and the text encoder outputs use the same embedding space. We presented two methods for this task that differ in how they approach the construction of this shared space.

In Chapter 4, we presented a technique for using adversarial training to construct this shared space. This was a novel modification of the adversarial training paradigm: rather than matching a generated distribution to a data distribution, we used adversarial training to match two generated distributions to each other. This adversarial

training method can be performed with non-parallel speech and text, making it ideal for scenarios with limited parallel ASR training data but available non-parallel data. We also added a speech autoencoder to this model, which can be trained on speech only and thus allow the model to take full advantage of all available data.

In Chapter 5, we developed a supervised method of embedding space alignment. We introduced an objective function, the encoder loss, that we used sequentially to train the text encoder and then to fine-tune the speech encoder. The training process included a final step of ASR training to adapt the decoder to the fine-tuned encoder outputs. We experimented with incorporating text autoencoder training into this final step, making this model architecture semi-supervised as well.

The next two chapters of this thesis explored ways to make better use of text data within the standard ASR training paradigm. Both of the techniques tested here make use of subword output units, rather than characters, and both segment the text used for training into subword units in a probabilistic way. This probabilistic segmentation is conceptually similar to regularization techniques like dropout (Srivastava et al., 2014) and data augmentation (Park et al., 2019) that force the model to be adaptable and thus reduce overfitting.

Subword regularization, discussed in Chapter 6, probabilistically segments text into subword units based on a unigram language model learned from text. In Chapter 7, we introduced a novel loss function designed to allow the ASR model to learn these subword segmentations for itself as part of the ASR training process. We used this loss function in conjunction with the LSD framework (Chan, Zhang, et al., 2016), and demonstrated that it improved model performance. Subsequently, we presented a method for training a statistical segmentation model to mimic the outputs of the ASR model. Using this statistical model, we were able to speed up LSD considerably while also further improving performance. This statistical segmentation model can also segment text that is not associated with speech, allowing us to use this subword discovery framework in combination with the semi-supervised learning methods described above.

In Chapter 8, we explored combinations of the methods introduced in the previous

four chapters, investigating the creation of a shared embedding space for speech and text using subword units. We experimented with each possible combination of one embedding space alignment technique and one subword segmentation technique in both fully- and semi-supervised scenarios.

	Fully Supervised			Semi-Supervised		
	WER	CER	+ LM WER	WER	CER	+ LM WER
Character Baseline	65.1	35.3	61.7	-	-	-
Ch. 4: Adv. Alignment	58.6	26.3	51.1	55.3	24.7	45.1
Ch. 5: Direct Alignment	59.8	32.8	53.4	57.6	30.9	50.0
Ch. 6: Subword Reg.	53.6	23.3	45.2	-	-	-
Ch. 7: Subword Disc.	50.2	21.0	37.9	-	-	-
Ch. 8: Adv. + Subword Reg.	53.3	23.6	44.7	51.9	21.9	41.6
Ch. 8: Adv. + Subword Disc.	51.0	21.5	40.0	48.6	22.0	40.1
Ch. 8: Direct + Subword Reg.	53.2	22.7	43.9	52.7	22.2	42.8
Ch. 8: Direct + Subword Disc.	49.5	19.9	34.6	50.0	19.8	33.8

Table 9.1: Summary of results on the 5 hour subset of the WSJ corpus. All results presented here use the best parameter settings found for that particular model on this corpus. Fully supervised results use the text and speech from the 5 hour corpus for all training of neural network parameters. Semi-supervised results use a selection of the text used to train the language model and, for the adversarial training models, the speech data from the full 81 hour corpus.

Table 9.1 contains results from all of the techniques described in this thesis on the 5 hour subset of the WSJ corpus. The baseline model that we introduced in Chapter 2 achieved a 61.9% WER when trained on this subset and decoded with the inclusion of an external language model. Our best performing model, shown in the last line of Table 9.1, is able to reduce this WER to 34.6% with no additional training data. This is a 44.1% relative improvement in WER. The best semi-supervised model that we trained achieved 33.8% WER, for a relative improvement of 45.4%.

There are several other notable results in Table 9.1. The subword segmentation techniques introduced in Chapters 6 and 7 are able to achieve most of our overall performance gain with very simple changes to how we handle the text data and to how we compute the training loss. These models do not require any modifications to the model architecture or any additional training data. While both semi-supervised

architectures that we introduced were able to effectively use non-parallel speech and text data to train parameters of our end-to-end model, the gains from this extra data were modest once this semi-supervised learning was combined with the use of subword units.

9.2 Future Work

In future work, there are a number of potential improvements to be made to the techniques developed here.

Our results in Chapter 4 make clear that we are not achieving the full potential of the additional speech data we are using for training. One direction this work could take is to incorporate recent developments in unsupervised representation learning from speech (Baevski et al., 2020) into our semi-supervised models. This could apply to models trained with either the adversarial alignment technique or the direct alignment technique, as both model architectures could easily incorporate this training. Another possible improvement to these models is to combine adversarial alignment and direct alignment, using the available non-parallel speech and text for the former and the parallel speech and text for the latter. This would most likely require development of a modified version of the direct alignment technique in which the various objective functions are used jointly rather than in sequence.

Subword regularization is a simple technique to use and implement, but is hampered by the need for hyperparameter search to find the best α setting for a particular corpus. An interesting direction for future work would be to explore a schedule for updating α during training to hone in on the ideal value. This could take a similar form to one of the many optimization techniques that begins with a high learning rate and gradually lower it over the course of training.

The n-gram maximum likelihood loss introduced in Chapter 6 allowed our ASR model to learn which subword units to use for ASR out of a large initial subword vocabulary. However, our sampling procedure, in which we sample uniformly from this vocabulary 20% of the time even at the end of training, will still expose the model

to units it has chosen not to use. This technique could likely benefit from a pruning procedure that would allow the model to focus on the acoustically-inspired subword units.

The simplicity of subword regularization and subword discovery should make them easy to combine with almost any end-to-end ASR architecture. We have already published a modified CTC beam search decoding algorithm for use with subword regularization (Drexler and Glass, 2019b). Another possible architecture to investigate is transformer models (Vaswani et al., 2017). We saw that subword regularization and subword discovery both combine effectively with our semi-supervised ASR models. It could potentially be fruitful to combine either technique with existing multi-task or multilingual models for low-resource ASR.

An alternative direction to take the n-gram maximum likelihood loss is to explore its use in HMM-based ASR models. The subword inventory and statistical models that we have learned here could be used as a substitute for a lexicon and pronunciation dictionary in an HMM-based model. This could potentially make that architecture more effective for languages in which a lexicon is not available.

Bibliography

Artetxe, M., Labaka, G., Agirre, E., & Cho, K. (2017). Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*.

Baevski, A., Zhou, H., Mohamed, A., & Auli, M. (2020). Wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., & Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Baskar, M. K., Watanabe, S., Astudillo, R., Hori, T., Burget, L., & Černock, J. (2019). Semi-supervised sequence-to-sequence asr using unpaired speech and text. *arXiv preprint arXiv:1905.01152*.

Bazzi, I., & Glass, J. (2000). Heterogeneous lexical units for automatic speech recognition: Preliminary investigations, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Bell, P., & Renals, S. (2015). Regularization of context-dependent deep neural networks with context-independent multi-task training, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Besacier, L., Barnard, E., Karpov, A., & Schultz, T. (2014). Automatic speech recognition for under-resourced languages: A survey. *Speech Communication*, *56*, 85–100.

Bisani, M., & Ney, H. (2003). Multigram-based grapheme-to-phoneme conversion for lvcsr, In *Eighth European Conference on Speech Communication and Technology*.

Bisani, M., & Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication*, *50*(5), 434–451.

Bulyko, I., Herrero, J., Mihelich, C., & Kimball, O. (2012). Subword speech recognition for detection of unseen words, In *Thirteenth Annual Conference of the International Speech Communication Association*.

Chan, W., Jaitly, N., Le, Q., & Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Chan, W., Zhang, Y., Le, Q., & Jaitly, N. (2016). Latent sequence decompositions. *arXiv preprint arXiv:1610.03035*.

Chen, D., & Mak, B. K.-W. (2015). Multitask learning of deep neural networks for low-resource speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(7), 1172–1183.

Chen, D., Mak, B., Leung, C.-C., & Sivadas, S. (2014). Joint acoustic modeling of triphones and trigraphemes by multi-task learning deep neural networks for low-resource speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Chiu, C.-C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, E. Et al. (2018). State-of-the-art speech recognition with sequence-to-sequence models, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Cho, J., Baskar, M. K., Li, R., Wiesner, M., Mallidi, S. H., Yalta, N., Karafiat, M., Watanabe, S., & Hori, T. (n.d.). Multilingual sequence-to-sequence speech recognition: Architecture, transfer learning, and language modeling, In *Proceedings of the Spoken Language Technology Workshop (SLT)*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-based models for speech recognition, In *Advances in neural information processing systems*.

Conneau, A., Lample, G., Ranzato, M., Denoyer, L., & Jégou, H. (2017). Word translation without parallel data. *arXiv preprint arXiv:1710.04087*.

Cui, X., Goel, V., & Kingsbury, B. (2015). Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(9), 1469–1477.

Dalmia, S., Sanabria, R., Metze, F., & Black, A. W. (2018). Sequence-based multilingual low resource speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Deligne, S., & Bimbot, F. (1997). Inference of variable-length linguistic and acoustic units by multigrams. *Speech Communication*, *23*(3), 223–241.

Deligne, S., Yvon, F., & Bimbot, F. (1995). Variable-length sequence matching for phonetic transcription using joint multigrams, In *Fourth European Conference on Speech Communication and Technology*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dieng, A. B., Ranganath, R., Altosaar, J., & Blei, D. M. (2018). Noisin: Unbiased regularization for recurrent neural networks. *arXiv preprint arXiv:1805.01500*.

Drexler, J., & Glass, J. (2018). Combining end-to-end and adversarial training for low-resource speech recognition, In *Proceedings of the Spoken Language Technology Workshop (SLT)*. IEEE.

Drexler, J., & Glass, J. (2019a). Explicit alignment of text and speech encodings for attention-based end-to-end speech recognition, In *Proceedings of the Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE.

Drexler, J., & Glass, J. (2019b). Subword regularization and beam search decoding for end-to-end automatic speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Drexler, J., & Glass, J. (2020). Learning a subword inventory jointly with end-to-end automatic speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning?, In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

Gal, Y., & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks, In *Advances in neural information processing systems*.

Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets, In *Advances in Neural Information Processing Systems*.

Gowda, D., Garg, A., Kim, K., Kumar, M., & Kim, C. (2019). Multi-task multi-resolution char-to-bpe cross-attention decoder for end-to-end speech recognition., In *Proceedings of Interspeech*.

Graves, A., Fernandez, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks, In *Proceedings of the 23rd international conference on Machine learning*. ACM.

Graves, A., & Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks, In *International Conference on Machine Learning*.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A. Et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Harper, M. (2014). *Iarpa babel program*. Retrieved February 22, 2018, from https://www.iarpa.gov/index.php/research-programs/babel

Hayashi, T., Watanabe, S., Zhang, Y., Toda, T., Hori, T., Astudillo, R., & Takeda, K. (2018). Back-translation-style data augmentation for end-to-end asr, In *Proceedings of the Spoken Language Technology Workshop (SLT)*. IEEE.

Heigold, G., Vanhoucke, V., Senior, A., Nguyen, P., Ranzato, M., Devin, M., & Dean, J. (2013). Multilingual acoustic models using distributed deep neural networks, In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hori, T., Astudillo, R., Hayashi, T., Zhang, Y., Watanabe, S., & Le Roux, J. (2019). Cycle-consistency training for end-to-end speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Hori, T., Watanabe, S., Zhang, Y., & Chan, W. (2017). Advances in joint CTC-attention based end-to-end speech recognition with a deep cnn encoder and rnn-lm. *arXiv preprint arXiv:1706.02737*.

Hsu, W.-N., Lee, A., Synnaeve, G., & Hannun, A. (2020). Semi-supervised speech recognition via local prior matching. *arXiv preprint arXiv:2002.10336*.

Hsu, W.-N., Zhang, Y., & Glass, J. (2017). Unsupervised learning of disentangled and interpretable representations from sequential data, In *Advances in neural information processing systems*.

Jaitly, N., & Hinton, G. E. (2013). Vocal tract length perturbation (vtlp) improves speech recognition, In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech and Language*.

Kahn, J., Lee, A., & Hannun, A. (2020). Self-training for end-to-end speech recognition, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Kannan, A., Datta, A., Sainath, T. N., Weinstein, E., Ramabhadran, B., Wu, Y., Bapna, A., Chen, Z., & Lee, S. (2019). Large-scale multilingual speech recognition with a streaming end-to-end model. *arXiv preprint arXiv:1909.05330*.

Karafiát, M., Baskar, M. K., Matejka, P., Vesel, K., Grezl, F., Burget, L., & Cernock, J. (2017). 2016 but babel system: Multilingual blstm acoustic model with i-vector based adaptation., In *Proceedings of Interspeech*.

Karita, S., Watanabe, S., Iwata, T., Ogawa, A., & Delcroix, M. (2018). Semi-supervised end-to-end speech recognition., In *Proceedings of Interspeech*.

Kim, S., Hori, T., & Watanabe, S. (2017). Joint CTC-attention based end-to-end speech recognition using multi-task learning, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Kneissler, J., & Klakow, D. (2001). Speech recognition for huge vocabularies by using optimized sub-word units, In *Seventh European Conference on Speech Communication and Technology*.

Ko, T., Peddinti, V., Povey, D., & Khudanpur, S. (2015). Audio augmentation for speech recognition, In *Sixteenth Annual Conference of the International Speech Communication Association*.

Kudo, T. (2018a). *Sentencepiece*. https://github.com/google/sentencepiece

Kudo, T. (2018b). Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*.

Lample, G., Denoyer, L., & Ranzato, M. (2017). Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*.

LDC. (1994). *Wall Street Journal Corpus*. Retrieved February 22, 2018, from https://catalog.ldc.upenn.edu/ldc94s13a

LeCun, Y., & Bengio, Y. (1998). Convolutional networks for images, speech, and time series, In *The handbook of brain theory and neural networks*. MIT Press.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks, In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Challenges in Representation Learning*.

Lin, H., Deng, L., Yu, D., Gong, Y.-f., Acero, A., & Lee, C.-H. (2009). A study on multilingual acoustic modeling for large vocabulary asr, In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE.

Liu, A. H., Lee, H.-y., & Lee, L.-s. (2019). Adversarial training of end-to-end speech recognition using a criticizing language model, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Liu, H. [Hairong], Zhu, Z., Li, X., & Satheesh, S. (2017). Gram-CTC: Automatic unit selection and target decomposition for sequence labelling, In *International Conference on Machine Learning*.

Liu, H. [Hu], Jin, S., & Zhang, C. (2018). Connectionist temporal classification with maximum entropy regularization, In *Advances in Neural Information Processing Systems*.

Meng, Z., Li, J., Chen, Z., Zhao, Y., Mazalov, V., Gang, Y., & Juang, B.-H. (2018). Speaker-invariant training via adversarial learning, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, *16*(1), 69–88.

Munteanu, C., Baecker, R., Penn, G., Toms, E., & James, D. (2006). The effect of speech recognition accuracy rates on the usefulness and usability of webcast archives, In *Proceedings of the SIGCHI conference on Human Factors in computing systems*.

Ochiai, T., Matsuda, S., Watanabe, H., & Katagiri, S. (2017). Automatic node selection for deep neural networks using group lasso regularization, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*.

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., & Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P. Et al. (2011). The kaldi speech recognition toolkit, In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.

Pratap, V., Sriram, A., Tomasello, P., Hannun, A., Liptchinsky, V., Synnaeve, G., & Collobert, R. (2020). Massively multilingual asr: 50 languages, 1 model, 1 billion parameters. *arXiv preprint arXiv:2007.03001*.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Rosenberg, A., Audhkhasi, K., Sethy, A., Ramabhadran, B., & Picheny, M. (2017). End-to-end speech recognition and keyword search on low-resource languages, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Rossenbach, N., Zeyer, A., Schlüter, R., & Ney, H. (2020). Generating synthetic audio data for attention-based speech recognition systems, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Scott, S. L. (2002). Bayesian methods for hidden Markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, *97*(457), 337–351.

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units, In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Serdyuk, D., Audhkhasi, K., Brakel, P., Ramabhadran, B., Thomas, S., Bengio, Y., & MILA, C. F. (2016). Invariant representations for noisy speech recognition. *arXiv preprint arXiv:1612.01928*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, *8*(3), 185–190.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning, In *International conference on machine learning*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks, In *Advances in neural information processing systems*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision, In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Tjandra, A., Sakti, S., & Nakamura, S. (2017). Listening while speaking: Speech chain by deep learning, In *Proceedings of the Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE.

Tjandra, A., Sakti, S., & Nakamura, S. (2018). Machine speech chain with one-shot speaker adaptation. *arXiv preprint arXiv:1803.10525*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need, In *Advances in neural information processing systems*.

Wang, Y., Chen, X., Gales, M. J., Ragni, A., & Wong, J. H. M. (2018). Phonetic and graphemic systems for multi-genre broadcast transcription, In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE.

Wu, C., Gales, M. J., Ragni, A., Karanasou, P., & Sim, K. C. (2017). Improving interpretability and regularization in deep learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *26*(2), 256–265.

Xiao, Z., Ou, Z., Chu, W., & Lin, H. (2018). Hybrid CTC-attention based end-to-end speech recognition using subword units. *arXiv preprint arXiv:1807.04978*.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zenkel, T., Sanabria, R., Metze, F., & Waibel, A. (2017). Subword and crossword units for CTC acoustic models. *arXiv preprint arXiv:1712.06855*.

Zeyer, A., Irie, K., Schlüter, R., & Ney, H. (2018). Improved training of end-to-end attention models for speech recognition. *arXiv preprint arXiv:1805.03294*.

Zhou, Y., Xiong, C., & Socher, R. (2017). Improved regularization techniques for end-to-end speech recognition. *arXiv preprint arXiv:1712.07108*.