Language Modelling for Code-Switched Text

Bachelors Thesis Project

Bachelors of Technology
in
Computer Science and Engineering

by

Tanmay Parekh
(140100011)

in co-ordination with

Saurabh Garg
(140070003)

under the guidance of

Prof. Preethi Jyothi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

May, 2018

Abstract

Code-switching is a routine phenomenon in multilingual communities, wherein speakers use switch between two or more distinct languages within a single utterance. Due to this mixing, there are significant changes in the structure of the language. Additionally, due to lack of code-switched textual data, standard models aren't robust enough and don't generalize well.

In this report, we present a simple and elegant approach to language modeling for bilingual code-switched text. Since code-switching is a blend of two or more different languages, a standard bilingual language model can be improved upon by using structures of the monolingual language models. We propose a novel technique called Dual Language Models, which involves building two complementary monolingual n-gram language models and combining them using a probabilistic model for switching between the two.

In the latter part, we also extend this idea to deep learning language models. We devise a new architecture called Dual RNNLM, which modifies the recurrent unit of the standard RNNLM architecture to better learn the structures of the two languages and also models the switch better. We show variants and modifications of the architecture and their respective analysis.

We evaluate the efficacy of our approach using a conversational Mandarin-English speech corpus. We prove the robustness of our model by showing significant improvements in perplexity measures over the standard bilingual language model without the use of any external information. We also observe similar consistent improvements in automatic speech recognition error rates for n-gram Dual Language Models. We also explore the use of various external information like external data, embeddings and semantic features to improve the existing language model. Later, we also use synthetic data from generative models to pre-train and improve the language models.

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 6th June, 2018

Place: IIT Bombay, Mumbai

Tanmay Parekh

Roll No: 140100011

Acknowledgements

This report is an outcome of my work with the coordination of Saurabh Garg under the guidance of Prof. Preethi Jyothi on Language Modelling for Code-Switched Text. I am thankful to these people and others who have been instrumental in helping me out throughout this project. More importantly, I would like to express my sincere gratitude towards my supervisor Prof. Preethi Jyothi for her guidance and encouragement. My research in language modelling and building speech recognition systems is being driven by her vision and support. It was immensely helpful in gaining the understanding required for writing this report.

Secondly, I would also like to take this opportunity to thank Saurabh Garg for his continuous support throughout the project. His ideation has been really key towards the research output of this project.

A special note: A good portion of the content of the report comes from two papers, both authored by Saurabh Garg, Tanmay Parekh and Preethi Jyothi. One has been accepted at Interspeech '18 [14] and the other has been submitted to a top-tier NLP conference.

Contents

Chapter 1

Introduction

Code-switching is a commonly occurring phenomenon in multilingual communities, wherein a speaker switches between languages within the span of a single utterance [4]. Building speech and language technologies to handle code-switching has become a fairly active area of research and presents a number of interesting technical challenges [9]. Language models for code-switched text is an important problem with implications to downstream applications such as speech recognition and machine translation of code-switched data.

Our approach towards building better language models can be distinctively split into two major regimes - (1) n-gram LMs (Statistical approach) and (2) RNNLMs (Deep Learning approach). We build simple baselines and explore mechanisms to improve upon them in both these regimes.

1.1 N-Gram LMs

A very naïve solution towards building such a language model would be to directly use a n-gram based bilingual language model. However, this can be significantly improved upon by using knowledge of the individual base languages, especially in such a limited data setting. Many sophisticated approaches relying on translation models have been proposed to overcome this challenge and are discussed in Chapter 2, but they use external resources to build the translation model in the first place.

We introduce an alternative – and simpler – approach to address the challenge of limited data in the context of code-switched text without use of any external resources. At the heart of our solution is a *n-gram dual language model* (DLM) that has roughly the complexity of two monolingual language models combined. A DLM combines two such models and uses a probabilistic model to switch between them. Its simplicity makes it amenable for generalization in a low-data context.

We further discuss the framework of DLMs and their benefits over the standard bilingual model in Chapter 3.

1.2 RNNLMs

A more natural choice for building robust language models would be to use recurrent neural networks (RNNs) [22], which yield state-of-the-art language models in the case of monolingual text. We explore a few major mechanisms to improve upon such a baseline model and are briefly described below.

- At first, we alter the structure of the basic RNN unit to include separate components that focus on each language in code-switched text separately while coordinating with each other to retain contextual information across code-switch boundaries. Our new model is called a Dual RNN Language Model (D-RNNLM), discussed in detail in Chapter 4.
- Secondly, we also propose using *same-source pretraining* – i.e., pretraining the model using data sampled from a generative model which is itself trained on the given training data – before training the model on the same training data (discussed further in Chapter 5). We find this to be surprisingly effective.
- We also explore various ways to use external knowledge about the individual languages, which includes additional monolingual data, pre-trained embeddings, translation tables, various semantic features like POS tags, brown clusters, etc. The various settings are further described in Chapter 6.

We study the improvements due to these techniques under various settings. We use perplexity as a proxy to measure the quality of the language model, evaluated on code-switched text in English and Mandarin from the SEAME corpus. We describe the dataset and the experiments in detail in Chapter 7. Further discussion on the results obtained are delineated in Chapter 8. We discuss other ideas which have been tried and can be further explored in Future Work (Chapter 9).

Chapter 2

Related Work

There have been many past efforts towards enhancing the capability of language models for code-switched text. Prior work on building such language models for code-switched speech can be broadly categorized into three sets of approaches:

(1) Without the use of external data. This work includes the use of normal n-gram and RNN based language models. Adel et al. [1] presented an approach to convert RNN based LMs into backoff n-gram LMs for efficient decoding within ASR systems.

(2) With the help of external information source. Adel et al. [2, 3] explored the use of POS taggers, syntactic and semantic features extracted from code-switched data for factored language models to enhance the capability of language models. Yeh et al. [30] employed class-based *n*-gram models that cluster words from both languages into classes based on POS and perplexity-based features. Vu et al. [27] used an SMT system to enhance the language models. Another common approach is to separately train language models on monolingual texts and then interpolate them to build code-switched language model [7, 16, 20]. With extra monolingual data, Baheti et al. [5] explored various curriculum learning strategies for training code-mixed language models.

(3) By incorporating linguistic constraints. Linguists [23, 25] discovered specific constraints know as the “equivalence constraint", which are satisfied when people switch between languages. Chan et al. [10] used grammar to model code-switching languages. Li and Fung [18, 19] incorporated syntactic constraints by a code-switch boundary prediction model and Functional Head constraints into language modeling.

Chapter 3

N-gram Dual Language Model

Amongst the various statistical approaches, n-gram LMs have been one of the most basic and simplest to build and use. They can also be encoded as a *finite state machine (FST)* and can be further used in various applications like speech recognition systems, etc. There are various extensions and variants of the traditional n-gram LMs which have enhanced it even more (class-based n-gram LMs, sequence n-gram LMs, etc). Here we consider the traditional n-gram LM with backoff and interpolation and various variants for smoothing like Kneyser-Ney and Good Turing. This forms the strong baseline language model.

Since the code-switched text comprises of two monolingual phrases mixed together, we believe that the individual phrases can be separately modelled in a better fashion and later can be combined. This forms the basis of the motivation behind *n-gram dual language models*. Below we describe n-gram DLM in more detail via a 2-player game.

3.1 The 2-player Game

We define a *n-gram dual language model* (DLM) to have the following 2-player game structure. A sentence (or more generally, a sequence of tokens) is generated via a co-operative game between the two players who take turns. During its turn a player generates one or more words (or tokens), and either terminates the sentence or transfers control to the other player. Optionally, while transferring control, a player may send additional information to the other player (e.g., the last word it produced), and also may retain some state information (e.g., cached words) for its next turn. At the beginning of the game one of the two players is chosen probabilistically.

In the context of code-switched text involving two languages, we consider a DLM wherein the two players are each in charge of generating tokens in one of the two languages. Suppose the two

languages have (typically disjoint) vocabularies V_1 and V_2. Then the alphabet of the output tokens produced by the first player in a single turn is $V_1 \cup \{\langle \texttt{sw} \rangle, \langle \texttt{/s} \rangle\}$, $\langle \texttt{sw} \rangle$ denotes the *switching* – i.e., transferring control to the other player – and $\langle \texttt{/s} \rangle$ denotes the end of sentence, terminating the game. We shall require that a player produces at least one token before switching or terminating, so that when $V_1 \cap V_2 = \emptyset$, any non-empty sentence in $(V_1 \cup V_2)^*$ uniquely determines the sequence of corresponding outputs from the two players when the DLM produces that sentence. (Without this restriction, the players can switch control between each other arbitrarily many times, or have either player terminate a given sentence.)

To illustrate how this game would be played, we can assume the sentence to be generated to be:

Sentence: *Kya aap* credit card accept *karte ho?*

We can assume Player 1 to have Hindi vocabulary and Player 2 to have English vocabulary. The game played to generate the above sentence would be:

Player 1: Kya aap <sw>
Player 2: credit card accept <sw>
Player 1: karte ho? </s>

3.2 Framework

Here, we explore a particularly simple DLM that is constructed from two given LMs for the two languages. More precisely, we shall consider an LM $\mathcal{L}_1$ which produces $\langle \texttt{/s} \rangle$-terminated strings in $(V_1 \cup \{\langle \texttt{sw} \rangle\})^*$ where $\langle \texttt{sw} \rangle$ indicates a span of tokens in the *other* language (so multiple $\langle \texttt{sw} \rangle$ tokens cannot appear adjacent to each other), and symmetrically an LM $\mathcal{L}_2$ which produces strings in $(V_2 \cup \{\langle \texttt{sw} \rangle\})^*$.

In Section 3.4, we will describe how such monolingual LMs can be constructed from code-switched data. Given $\mathcal{L}_1$ and $\mathcal{L}_2$, we shall splice them together into a simple DLM (in which players do not retain any state between turns, or transmit state information to the other player at the end of a turn). Below we explain this process in a more formal sense (for bi-gram language models).

Given two language models $\mathcal{L}_1$ and $\mathcal{L}_2$ with conditional probabilities P_1 and P_2 that satisfy the following conditions:

$$P_1[\langle \texttt{/s} \rangle \mid \langle \texttt{s} \rangle] = P_2[\langle \texttt{/s} \rangle \mid \langle \texttt{s} \rangle] = 0 \tag{3.1}$$

$$P_1[\langle \texttt{sw} \rangle \mid \langle \texttt{s} \rangle] + P_2[\langle \texttt{sw} \rangle \mid \langle \texttt{s} \rangle] = 1 \tag{3.2}$$

$$P_1[\langle \texttt{sw} \rangle \mid \langle \texttt{sw} \rangle] = P_2[\langle \texttt{sw} \rangle \mid \langle \texttt{sw} \rangle] = 0 \tag{3.3}$$

$$P_1[\langle \texttt{/s} \rangle \mid \langle \texttt{sw} \rangle] = P_2[\langle \texttt{/s} \rangle \mid \langle \texttt{sw} \rangle] = 0 \tag{3.4}$$

We impose conditions (3.1)-(3.4) on the given LMs. Condition (3.1) which disallows empty sentences in the given LMs (and the resulting LM) is natural, and merely for convenience. Condition (3.2) states the requirement that $\mathcal{L}_1$ and $\mathcal{L}_2$ agree on the probabilities with which each of them gets the first turn. Conditions (3.3) and (3.4) require that after switching at least one token should be output before switching again or terminating. If the two LMs are trained on the same data as described in Section 3.4, all these conditions would hold.

We define a combined language model $\mathcal{L}$, with conditional probabilities P, as follows:

$$P[w' \mid w] = \begin{cases} P_1[w' \mid \langle \mathtt{s} \rangle] & \text{if } w' \in \mathrm{V}_1 \\ P_2[w' \mid \langle \mathtt{s} \rangle] & \text{if } w' \in \mathrm{V}_2 \\ 0 & \text{if } w' = \langle /\mathtt{s} \rangle \end{cases} \qquad \text{for } w = \langle \mathtt{s} \rangle$$

$$P[w' \mid w] = \begin{cases} P_1[w' \mid w] & \text{if } w' \in \mathrm{V}_1 \cup \{\langle /\mathtt{s} \rangle\} \\ P_1[\langle \mathtt{sw} \rangle \mid w] \cdot P_2[w' \mid \langle \mathtt{sw} \rangle] & \text{if } w' \in \mathrm{V}_2 \end{cases} \qquad \text{for } w \in \mathrm{V}_1$$

$$P[w' \mid w] = \begin{cases} P_2[w' \mid w] & \text{if } w' \in \mathrm{V}_2 \cup \{\langle /\mathtt{s} \rangle\} \\ P_2[\langle \mathtt{sw} \rangle \mid w] \cdot P_1[w' \mid \langle \mathtt{sw} \rangle] & \text{if } w' \in \mathrm{V}_1 \end{cases} \qquad \text{for } w \in \mathrm{V}_2$$

To see that $P[w' \mid w]$ as described above is a well-defined probability distribution, we check that $\sum_{w'} P[w'|w] = 1$ for all three cases of w, where the summation is over $w' \in \mathrm{V}_1 \cup \mathrm{V}_2 \cup \{\langle /\mathtt{s} \rangle\}$. When $w = \langle \mathtt{s} \rangle$, $\sum_{w'} P[w'|w]$ equals

$$\begin{aligned} &\sum_{w' \in \mathrm{V}_1} P_1[w'|\langle \mathtt{s} \rangle] + \sum_{w' \in \mathrm{V}_2} P_2[w'|\langle \mathtt{s} \rangle] \\ &\quad = (1 - P_1[\langle \mathtt{sw} \rangle \mid \langle \mathtt{s} \rangle]) + (1 - P_2[\langle \mathtt{sw} \rangle \mid \langle \mathtt{s} \rangle]) = 1 \end{aligned}$$

where the first equality is from (3.1) and the second equality is from (3.2).

When $w \in \mathrm{V}_1$, $\sum_{w'} P[w'|w]$ is

$$\begin{aligned} &\sum_{w' \in \mathrm{V}_1 \cup \langle /\mathtt{s} \rangle} P_1[w'|w] + P_1[\langle \mathtt{sw} \rangle \mid w] \sum_{w' \in \mathrm{V}_2} P_2[w'|\langle \mathtt{sw} \rangle] \\ &\quad = \sum_{w' \in \mathrm{V}_1 \cup \langle /\mathtt{s} \rangle} P_1[w'|w] + P_1[\langle \mathtt{sw} \rangle \mid w] = 1. \end{aligned}$$

The case of $w \in \mathrm{V}_2$ follows symmetrically.

3.3 Implementation as FST

Figure 3.1 illustrates how to implement a DLM as a finite-state machine using finite-state machines for the monolingual bigram LMs, $\mathcal{L}_1$ and $\mathcal{L}_2$. The start states in both LMs, along with all the

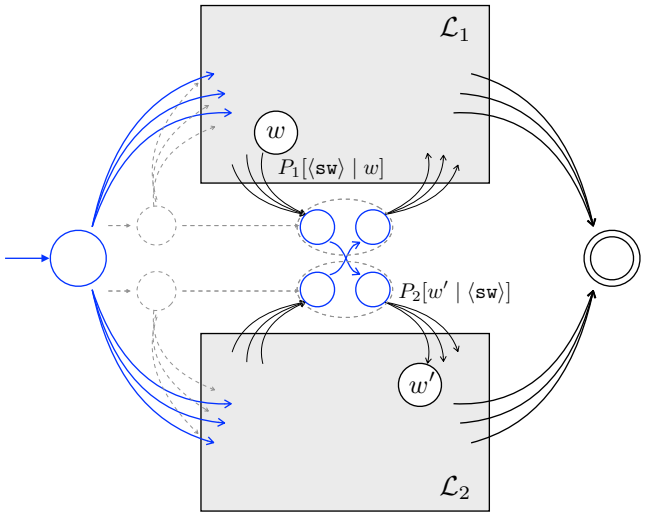

Figure 3.1: *DLM using two monolingual LMs, $\mathcal{L}_1$ and $\mathcal{L}_2$, implemented as a finite-state machine.*

arcs leaving these states, are deleted; a new start state and end state is created for the DLM with accompanying arcs as shown in Figure 3.1. The two states maintaining information about the $\langle \texttt{sw} \rangle$ token can be split and connected, as shown in Figure 3.1, to create paths between $\mathcal{L}_1$ and $\mathcal{L}_2$.

As illustrated in Figure 3.1, we can observe that DLM can be implemented as a FST by addition of a few states and corresponding arcs. The complexity of this FST is slightly greater than the individual monolingual FSTs and will be much lesser than the bilingual n-gram FST in the worst case scenario.

3.4 Monolingual LMs for the DLM construction

Given a code-switched text corpus D, we will derive two complementary corpora, D_1 and D_2, from which we construct bigram models $\mathcal{L}_1$ and $\mathcal{L}_2$ as required by the DLM construction described in Section 3.2. In D_1, spans of tokens in the second language are replaced by a single token $\langle \texttt{sw} \rangle$. D_2 is constructed symmetrically. Standard bigram model construction on D_1 and D_2 ensures conditions (3.1) and (3.2). The remaining two conditions may not naturally hold: Even though the data in D_1 and D_2 will not have consecutive $\langle \texttt{sw} \rangle$ tokens, smoothing operations may assign a non-zero probability for this; also, both LMs may assign non-zero probability for a sentence to end right after a $\langle \texttt{sw} \rangle$ token, corresponding to the sentence having ended with a non-empty span of tokens in the other language. These two conditions are therefore enforced by reweighting the LMs.

Chapter 4

Dual RNNLM

Statistical language models are simple in design and usage, but with the advent of deep learning, they were consistently outperformed. Today, recurrent neural networks based language models (RNNLMs) yield the state-of-the-art perplexity numbers in the case of monolingual text [22]. In the past few years, many more variants have been introduced in terms of the basic rnn unit (LSTMs, GRUs) as well as the overall architecture (FLMs, class-based LMs).

Apart from baseline performance, it will be much easier to integrate abundant monolingual data to improve the bilingual language model for RNNLMs as compared to n-gram DLMs. The improvements shown as well will be more robust in comparison. As described in Chapter 3, the vanilla version of DLMs don't share any context when switching from one language to another. We aim to overcome this drawback by introducing context-sharing across switching points and capturing long-term cross-lingual dependencies.

All of these factors contributed towards shifting from the statistical models to deep-learning based models for language modelling. The baseline model we consider here is the standard RNNLM with LSTM unit as the recurrent unit. We fine-tune across various parameters (discussed further in Chapter 7) to build a strong and robust baseline.

4.1 The Idea

Towards improving the modeling of code-switched text, we introduce *Dual RNN Language Models (D-RNNLMs)*. The philosophy behind D-RNNLMs, an extension of the idea for DLMs, is that two different sets of neurons will be trained to (primarily) handle the two individual languages. For the same, we explicitly model the individual stretches of both the languages via separate LSTM units. We facilitate sharing of context across the LSTM units to capture cross-lingual long-term

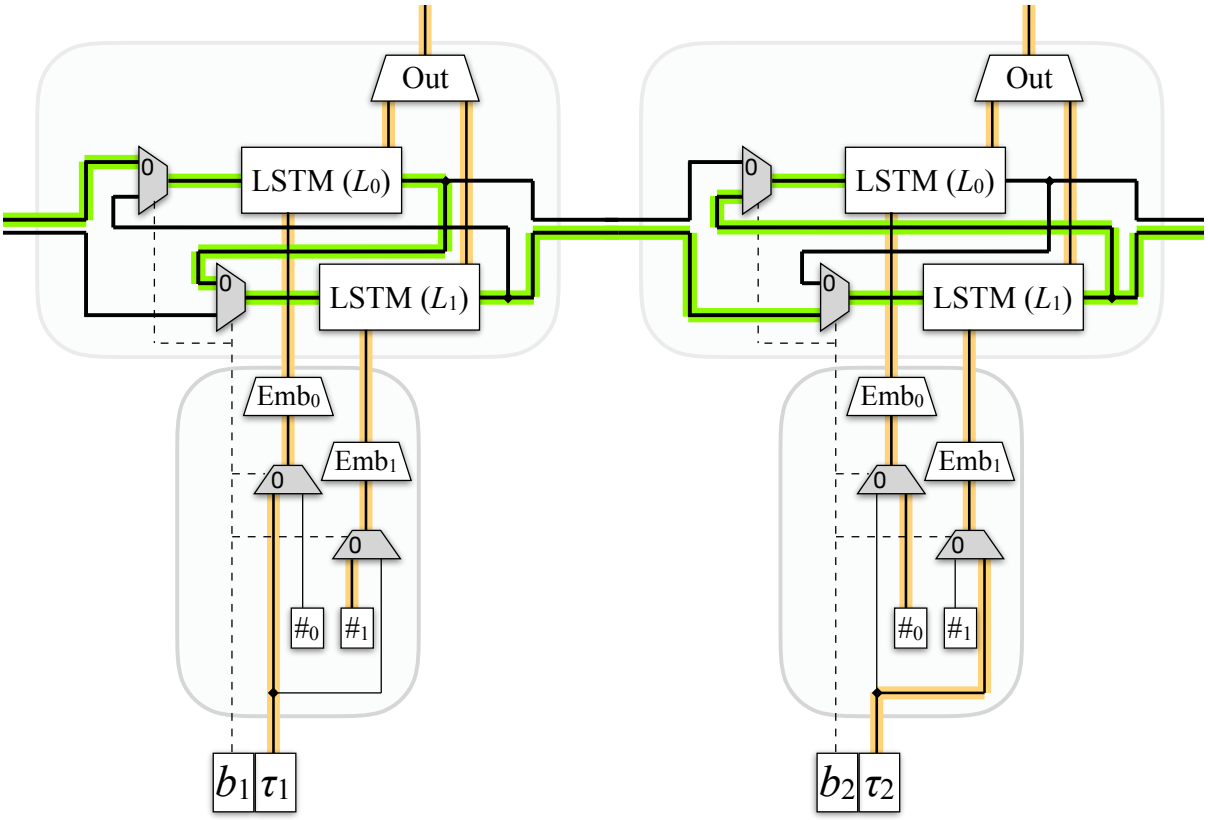

Figure 4.1: *Illustration of the dual RNNLM (see the text for a detailed description). The highlighted left-to-right path (in green) indicates the flow of state information, when $\lambda_1 = 0$ and $\lambda_2 = 1$ (corresponding to token c_1 belonging to language L_0 and c_2 belonging to L_1). The highlighted bottom-to-top path (in orange) indicates the inputs and outputs.*

dependencies.

There are various components to building the D-RNNLM, namely the LSTM unit, the input encoding layer and the output combination layer. We have tried various variants for each of the components and are explained in more detail. The details of the best working architecture and the corresponding implementation are explained in the Section 4.2 below.

4.2 Dual LSTM cell

As shown in Figure 4.1, the D-RNNLM consists of a "Dual LSTM cell" and an input encoding layer. The Dual LSTM cell, as the name indicates, has two long short-term memory (LSTM) cells within it. The two LSTM cells are designated to accept input tokens from the two languages L_0 and L_1 respectively, and produce an (unnormalized) output distribution over the tokens in the

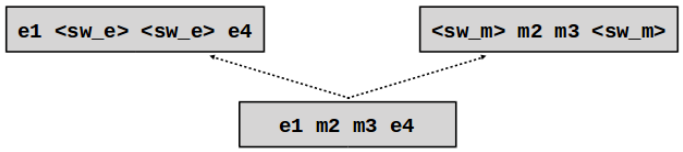

Figure 4.2: *Illustration of input encoding layer*

same language. When a Dual LSTM cell is invoked with an input token τ, the two cells will be invoked sequentially. The first (upstream) LSTM cell corresponds to the language that τ belongs to, and gets τ as its input. It passes on the resulting state to the downstream LSTM cell (which takes a dummy token as input). The unnormalized outputs from the two cells are combined and passed through a soft-max operation to obtain a distribution over the union of the tokens in the two languages. Figure 4.1 shows a circuit representation of this configuration, using multiplexers (shaded units) controlled by a selection bit b_i such that the i^{th} token τ_i belongs to L_{b_i}.

4.3 Input encoding layer

The input encoding layer also uses multiplexers to direct the input token to the upstream LSTM cell. Two dummy tokens $\#_0$ and $\#_1$ are added to L_0 and L_1 respectively, to use as inputs to the downstream LSTM cell. The input tokens are encoded using an embedding layer of the network (one for each language), which is trained along with the rest of the network to minimize a cross-entropy loss function.

An illustration of how a generic stream of input tokens will be broken into two separate input streams is shown in Figure 4.2. Here, the dummy tokens are labelled as $< sw\_e >$ and $< sw\_m >$ respectively.

4.4 Output combination layer

As indicated in Figure 4.1, we obtain two sets of outputs (each from the corresponding LSTM unit) in different vocabularies. We have tried various variants to combine the outputs and pass the gradients in a meaningful manner, of which some are described below.

4.4.1 Common Softmax

This is the output layer setting shown in Figure 4.3 and as implemented in Figure 4.1. We directly combine the outputs of the two cells and pass it through a common softmax layer. The obtained

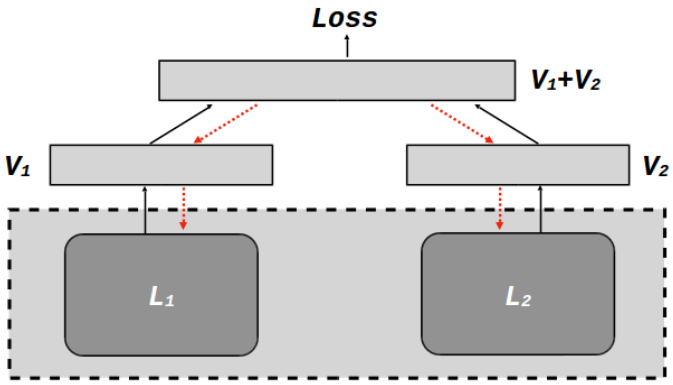

Figure 4.3: *Common Softmax Output Combination Layer*

distribution is now used to compute the cross-entropy loss. The gradients (as indicated in the figure by red arrows) are passed through the common softmax into both the LSTM units per input token and both are trained simultaneously.

4.4.2 Trainable switch parameter

Figure 4.4 illustrates the trainable switch parameter setting of output combination layer. Here the outputs from each LSTM unit are directly passed on to the individual softmax layers, whose distributions are further used to compute the individual losses. The final loss is computed as an interpolation between the individual losses which is parametrized be a *switch parameter*, which is trained simultaneously as well. In such a setting, gradients (as indicated in the figure by red arrows) are passed only to a single LSTM per input token and training happens in turns for the LSTM units.

The perplexity numbers as observed by experimentation with this setting were not at par with the perplexities obtained by the common softmax output layer combination.

4.5 Context Sharing

We introduce the notion of context sharing between the LSTMs to facilitate cross-lingual context sharing for better modelling of language at switching points. There are various variants for the same which have been tried out, some of which are briefly explained below.

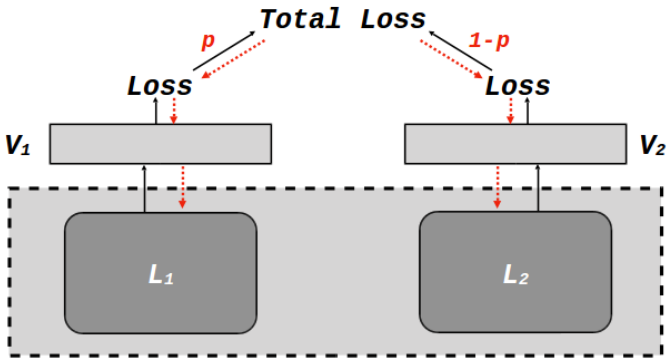

Figure 4.4: *Trainable switch parameter Output Combination Layer*

4.5.1 No sharing

This setting is inspired from the idea of no-context sharing in n-gram DLMs. Here, both the LSTM units are trained independently of the other without sharing any common information.

4.5.2 Direct sharing

This setting is explained in the implementation details in Section 4.2, where the context between the LSTM units are shared by upstream-downstream LSTM architecture. Here the LSTM unit (upstream) corresponding to the language of the current input token τ is run and the corresponding updated context is then passed to the other LSTM unit (downstream). Due to this direct sharing of context, the training results in updating weights for both the LSTMs.

4.5.3 Damped sharing

This is a slight variant of the direct sharing explained above, but instead of directly passing the entire context, we passed a damped version of the context. The damped context was the original context multiplied with a damping factor of α (where $0.0 < \alpha < 1.0$) which was used in the downstream LSTM unit. The parameter α had been finely tuned.

4.5.4 Masked sharing

This variant can be thought of as an interpolation between no sharing and direct sharing. Here, we don't share the entire context but only a part of the context. The entire context is split into language-specific (or LSTM specific) contexts. During sharing of context, the part of the context

specific to the upstream LSTM unit will be masked out and only the remaining context will be shared to the downstream LSTM unit. This motivates the LSTM units to learn language-specific and cross-lingual contexts separately and better.

Amongst all the above four settings of context sharing, the best one which gave the highest perplexity instruments was the one of direct sharing.

Chapter 5

Same-Source Pre-Training

Building robust LMs for code-switched text is challenging due to the lack of availability of large amounts of training data. One solution is to artificially generate code-switched to augment the training data. We propose a variant of this approach – called same-source pretraining – in which the actual training data itself is used to train a generative model, and the data sampled from this model is used to pretrain the language model.

Same-source pretraining can leverage powerful training techniques for generative models to train a language model. We note that the generative models by themselves are typically trained to minimize a different objective function (e.g., a discrimination loss) and need not perform well as language models.[1] There are various techniques and models to generate this synthetic data and we explore a few of them, as described below.

5.1 Naïve Sampling

We train the base RNNLM using the maximum likelihood training paradigm (based on the cross-entropy loss) on the training data. During inference in order to generate artificial data, at every time step, we sample from the output layer of the RNNLM given the history. Here we sample directly from the RNNLM, hence the name *"Naïve Sampling"*.

[1] In our experiments, we found the perplexity measures for the generative models to be an *order of magnitude* larger than that of the LMs we construct.

5.2 Scheduled Sampling

The above model based on Naïve Sampling tends to suffer from the problem of *exposure bias* during inference when the model generates a text sequence by conditioning on previous tokens that may have never appeared during training. Bengio et al. [6] proposed to bridge this gap between the training and inference stages by using model predictions to synthesize prefixes of text that are used during training, rather than using the actual text tokens.

5.3 SeqGAN

A more promising alternative to generate text sequences was recently proposed by Yu et al. [31] where sequence generation is modelled in a generative adversarial network (GAN) based framework. This model – referred to as “SeqGAN" – consists of a generator RNN and a discriminator network trained as a binary classifier to distinguish between real and generated sequences and both are trained using the adversarial training paradigm. A typical model trained by the standard GAN paradigm doesn’t yield promising results as text generation is discrete token generation task wherein discrete outputs make it difficult to pass the gradients from the discriminator to the generator. The main innovation of SeqGAN is to train the generative model using policy gradients (inspired by reinforcement learning) which helps bypass the generator differentiation problem. It also uses the discriminator to determine the reward function.

5.4 D-RNNLM SeqGAN

As the name suggests, this is just a small variant of the SeqGAN model wherein we use D-RNNLM (described in Chapter 4) architecture in the generator instead of the RNN architecture as in the traditional SeqGAN model. The remaining components and the training paradigm of the model remain the same.

The perplexity numbers with pre-training using samples generated from Naive and Scheduled Sampling (Dev - 84.63 and Test - 70.66) were nearly similar and poor compared to that using SeqGAN generated samples (Dev - 79.16 and Test - 65.96).

Chapter 6

Using External Knowledge

Till now we explored various mechanisms and techniques to best use the available limited code-switched data to model the language. Though the amount of code-switched data is quite limited, but the amount of monolingual data for the individual languages is far more in abundance relatively. Hence, it would be prudent to use external knowledge derived from this abundant data to improve upon our baseline language models.

At first, we could directly use the monolingual data for the individual languages as a supplementary source of data to train the language models. Other derived sources of external knowledge include using pre-trained word embeddings as a better initialization. Apart from the data itself, we could as well use various semantic features derived for the data like POS tags, language ids, brown word clusters, etc [3] to better model code-switched text. We can also use translation tables to artificially generate code-switched text from the monolingual data. Each of these are further described in detail below.

6.1 Additional Monolingual Data

The most simple way of using derived external knowledge is to use it as additional data to pre-train the network. Code-switched text for most pairs of languages is limited, while monolingual data for them is abundant and thus, this additional data can be used to pre-train the network to provide a good initialisation. The order of training follows a simple routine of (1) training on the additional monolingual data (Pre-training) and (2) training the pre-trained network on the code-switched text. We tried various mixes of the additional monolingual data to best pre-train the model and found that a random mix works the best.

6.2 Pre-trained Embeddings

A substitute to pre-train the network apart from using additional data is to directly use pre-trained embeddings for initialisation. There are various sources for the same, one of which we used was *FastText*, which are word embeddings for several languages made available by Facebook Research [17].

For multilingual word embeddings, Facebook Research also recently released a technique called *Multilingual Unsupervised and Supervised Embeddings*, which can be used to align the monolingual embeddings to a common space. The technique uses three different phases - (1) Domain-Adversarial learning (2) Refinement procedure by applying Procrustes solution (3) Translation using CSLS (Cross-domain Similarity Local Scaling) distance metric [11].

We tried using these sources of pre-trained embeddings, but they didn't improve over the baseline numbers. This can be attributed to a poor overlap of words between the data corpus and embedding dictionary. Another reason could be the poor alignment between the English and the Mandarin embedding space.

6.3 Syntactic and Semantic Features

Another way of using external knowledge is to externally derive semantic and syntactic features. These features can be readily used along with the text for better modelling of text. Even though the structure and syntax of the code-switched text is different, these features will definitely aid in stretches of monolingual text. Inspired by use of such features used in Factored Language Models (FLMs) [2, 3], we use POS tags, brown word clusters and language ids along with the raw text for training.

6.4 Artificially Derived Data

One of the simplest ways to artificially generate code-switched data is to use translation tables for the language pairs and artificially introduce code-switching in the monolingual text by simple heuristics. We use the abundant monolingual data as the base data and *Google Translate API* to generate the word-level translation table. We describe both the heuristics below.

- **Single Parameter:** Wick et al. [29] suggested a heuristic to replace a word by it's translated counterpart with a fixed probability and obtained better bilingual embeddings using pre-training on such additional artificial code-switched data. Such a heuristic is parametrized by a single

parameter i.e. fixed probability p which had been finely tuned.

$$sw(w_t) = Bernoulli(p)$$

- **Dual Parameter:** To overcome the challenges posed by the single parameter setting, we introduce another degree of freedom in the form of a parameter. The parameters (p_1 and p_2) signify the probability to switch from L_1 and L_2 respectively. Instead of fine-tuning the parameters, we use the MLE (Maximum Likelihood Estimate) for these parameters based on the original code-switched data.

$$sw(w_t|w_{t-1} = L_1) = Bernoulli(p_1)$$
$$sw(w_t|w_{t-1} = L_2) = Bernoulli(p_2)$$

On analysis of the heuristics, we find that since the first heuristic has only one parameter, it can't model the code-switching of the two languages separately. The generated samples had very few long stretches of monolingual text, which is quite different from the true code-switched text. On the other hand, the second heuristic doesn't have this drawback.

We tried both the heuristics to generate samples and pre-train the LM. To compare, we generate a fair baseline where we use equivalent amount of monolingual data as that of derived data to pre-train the model. We expect that pre-training using code-switched data should provide lesser perplexity as compared to pre-training with monolingual data. The perplexity numbers are summarized in Table 6.1.

	Dev	Test
Single Parameter	76.63	63.54
Dual Parameter	76.13	63.3
Fair Baseline	75.11	62.53

Table 6.1: *Summary of perplexities of pre-training using artificially derived data.*

Firstly, the dual parameter derived data performs better than the single parameter derived data. Secondly, we observe that the this pre-training doesn't beat our baseline and performs worse. This can be attributed to a loss of structure by blindly replacing words between the languages and depicted in the example below.

Original Sentence: I love watching movies.
Artificially generated sentence: *Mein mohabbat* watching movies.

Hence, we conclude that mere replacing words via translation tables isn't the best way to generate code-switched data. But, it can be improved by replacing at phrase level using a larger translation table.

Chapter 7

Experiments and Results

As our approaches to better model code-switched text were split into two major regimes, our experiments have also been organised in a similar fashion - (1) N-gram LMs and (2) RNNLMs

7.1 Data description

For experiments in both the approaches, we make use of the SEAME corpus [21] which is a conversational Mandarin-English code-switching speech corpus. It comprises of speech segments of conversations and interviews of speakers who are natively speak Mandarin and use English for code-switching. Hence, the base language here is Mandarin.

Preprocessing of data. Apart from the code-switched speech, the SEAME corpus comprises of a) words of foreign origin (other than Mandarin and English) b) incomplete words c) unknown words labeled as $\langle unk \rangle$, and d) mixed words such as bleach跟, cause就是, etc.. Since it was difficult to obtain pronunciations for these words, we removed utterances that contained any of these words. A few utterances contained markers for non-speech sounds like laughing, breathing, etc. Since our focus in this work is to investigate language models for code-switching, ideally without the interference of these non-speech sounds, we excluded these utterances from our task. After adopting our preprocessing steps, we saw an overall reduction of $\approx 15\%$ compared to the original corpus.

7.2 N-gram LM

7.2.1 Data distribution

We construct training, development and test sets from the preprocessed SEAME corpus data using a 60-20-20 split. Table 7.1 shows detailed statistics of each split. The development and evaluation sets were chosen to have 37 and 30 random speakers each, disjoint from the speakers in the training data.[1] The out-of-vocabulary (OOV) rates on the development and test sets are 3.3% and 3.7%, respectively.

	Train	Dev	Test
# Speakers	90	37	30
Duration (hrs)	56.6	18.5	18.7
# Utterances	54,020	19,976	19,784
# Tokens	539,185	195,551	196,462

Table 7.1: *Statistics of the dataset*

7.2.2 Perplexity experiments

We used the SRILM toolkit [26] to build all our LMs. The baseline LM is a smoothed bigram LM estimated using the code-switched text which will henceforth be referred to as *mixed LM*. Our DLM was built using two monolingual bigram LMs. (The choice of bigram LMs instead of trigram LMs will be justified later in Section 8.1.4). Table 7.2 shows the perplexities on the validation and test sets using both Good Turing and Kneser-Ney smoothing techniques. DLMs clearly outperform mixed LMs on both the datasets. All subsequent experiments use Kneser-Ney smoothed bigram LMs as they perform better than the Good Turing smoothed bigram LMs.

Smoothing Technique	Dev		Test	
	mixed LM	DLM	mixed LM	DLM
Good Turing	338.2978	**329.1822**	384.5164	**371.1112**
Kneser-Ney	329.6725	**324.9268**	376.0968	**369.9355**

Table 7.2: *Perplexities on the dev/test sets using mixed LMs and DLMs with different smoothing techniques.*

We also evaluate perplexities by reducing the amount of training data to $\frac{1}{2}$ or $\frac{1}{3}$ of the original training data (shown in Table 7.3). As we reduce the training data, the improvements in perplexity of DLM over mixed LM further increase, which validates our hypothesis that DLMs are capable of generalizing better. Section 5 elaborates this point further.

[1] We note that choosing fewer speakers in the development and test sets led to high variance in the observed results.

Training data	Dev		Test	
	mixed LM	DLM	mixed LM	DLM
Full	329.6725	**324.9268**	376.0968	**369.9355**
1/2	362.0966	**350.5860**	400.5831	**389.7618**
1/3	368.6205	**356.012**	408.562	**394.2131**

Table 7.3: *Kneser-Ney smoothed bigram dev/test set perplexities using varying amounts of training data*

7.2.3 ASR experiments

All the ASR systems were built using the Kaldi toolkit [24]. We used standard MFCC+delta+double-delta features with fMLLR transforms to build speaker-adapted triphone models with 4200 tied-state triphones, henceforth referred to as "SAT" models. We also build time delay neural network (TDNN)-based acoustic models using i-vector based features (referred to as "TDNN+SAT"). Finally, we also re-scored lattices generated by the "TDNN+SAT" model with an RNNLM [15] (referred to as "RNNLM Rescoring"), trained using Tensorflow. [2] We trained a single-layer RNN with 200 hidden units in the LSTM cell.

The pronunciation lexicon was constructed from CMUdict [28] and THCHS30 dictionary [12] for English and Mandarin pronunciations, respectively. Mandarin words that didn't appear in THCHS30 were mapped into Pinyin using a freely available Chinese to Pinyin converter.[3] We manually merged the phone sets of Mandarin and English (by mapping all the phones to IPA) resulting in a phone inventory of size 105.

To evaluate the ASR systems, we treat English words and Mandarin characters as separate tokens and compute token error rates (TERs) as discussed in [27]. Table 7.4 shows TERs on the dev/test sets using both mixed LMs and DLMs. DLM performs better or at par with mixed LM and at the same time, captures a significant amount of complementary information which we leverage by combining lattices from both systems. The improvements in TER after combining the lattices are statistically significant (at $p < 0.001$) for all three systems, which justifies our claim of capturing complementary information. Trigram mixed LM performance was worse than bigram mixed LM; hence we adopted the latter in all our models (further discussed in Section 8.1.4). This demonstrates that obtaining significant performance improvements via LMs on this task is very challenging.

Table 7.5 shows all the TER numbers by utilizing only $\frac{1}{2}$ of the total training data. The combined models continue to give significant improvements over the individual models. Moreover, DLMs consistently show improvements on TERs compared to mixed LMs in the $\frac{1}{2}$ training data setting.

[2] This rescoring was implemented using the `tfrnnlm` binary provided by Kaldi [24] developers.

[3] `https://www.chineseconverter.com/en/convert/chinese-to-pinyin`

[1] statistically significant improvement (at $p < 0.001$)

ASR system	Data	mixed LM	DLM	combined
SAT	Dev	45.59	45.59	**44.93***
	Test	47.43	47.48	**46.96***
TDNN+SAT	Dev	35.20	35.26	**34.91***
	Test	37.42	37.35	**37.17***
RNNLM Rescoring	Dev	34.21	34.11	**33.85***
	Test	36.64	36.52	**36.37***

Table 7.4: *TERs using mixed LMs and DLMs*

ASR system	Data	mixed LM	DLM	combined
SAT	Dev	48.48	48.17	**47.67**[1]
	Test	49.07	49.04	**48.52***
TDNN+SAT	Dev	40.59	40.48	**40.12***
	Test	41.34	41.32	**41.13***
RNNLM Rescoring	Dev	40.20	40.09	**39.84***
	Test	40.98	40.90	**40.72***

Table 7.5: *TERs with $\frac{1}{2}$ training data*

	Train	Dev	Test
# Utterances	74,927	9,301	9,552
# Tokens	977,751	131,230	114,546
# English Tokens	316,726	30,154	50,537
# Mandarin Tokens	661,025	101,076	64,009

Table 7.6: *Statistics of data splits derived from SEAME.*

7.3 RNNLMs

7.3.1 Data Distribution & Tokenisation

The training, development and test sets split that we use here is a random 80-10-10 split (Speakers were kept disjoint across these datasets). We increase the training data here to build a strong and robust LM and maximise performance gains. Table 7.6 shows more details about our datasets.

Also, another major change in the datasets is the tokenisation of the Mandarin text. Unlike previous dataset, here we break Mandarin words into characters. Though the SEAME corpus provided word boundaries for Mandarin text, we used Mandarin characters as individual tokens since a large proportion of Mandarin words appeared very sparsely in the data. Using Mandarin characters as tokens helped alleviate this issue of data sparsity; also, applications using Mandarin

	w/o syntactic features				with syntactic features			
	w/o mono. data		with mono. data		w/o mono. data		with mono. data	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test
Baseline	89.60	74.87	74.06	61.66	81.87	68.23	71.04	59.00
D-RNNLM	88.68	72.29	72.41	60.73	81.01	66.26	70.83	59.04
With RNNLM SeqGAN	79.16	65.96	72.51	60.56	77.30	63.75	68.43	55.71
With D-RNNLM SeqGAN	**78.63**	**65.41**	**72.33**	**60.30**	**77.19**	**63.63**	**67.79**	**55.60**

Table 7.7: *Development set and test set perplexities using RNNLMs and D-RNNLMs with various pretraining strategies.*

text are typically evaluated at the character level and do not rely on having word boundary markers [27].

7.3.2 Improvements over Baseline

This section will explore the benefits of both our proposed techniques – (1) using D-RNNLMs and (2) using text generated from SeqGAN for pretraining – in isolation and in combination. This section focuses only on the numbers listed in the first two columns of Table 7.7.

Baseline: The Baseline model is a 1-layer LSTM LM with 512 hidden nodes, input and output embedding dimensionality of 512, trained using SGD with an initial learning rate of 1.0 (decayed exponentially after 80 epochs at a rate of 0.98 till 100 epochs) The development and test set perplexities using the baseline are 89.60 and 74.87, respectively.

Using D-RNNLMs: The D-RNNLM is a 1-layer language model with each LSTM unit having 512 hidden nodes. The training paradigm is similar to the above-mentioned setting for the baseline model.[4] We see consistent improvements in test perplexity when comparing a D-RNNLM with an RNNLM (i.e. 74.87 drops to 72.29).[5]

Using SeqGAN: Next, we use text generated from a SeqGAN model to pretrain the RNNLM.[6] We use our best trained RNNLM baseline as the generator within SeqGAN. We sample 157,440 sentences (with a fixed sentence length of 20) from the SeqGAN model; this is thrice the amount of code-switched training data. We first pretrain the baseline RNNLM with this sampled text, before

[4]D-RNNLMs have a few additional parameters. However, increasing the capacity of an RNNLM to exactly match this number makes its test perplexity worse; RNNLM with 720 hidden units gives a development set perplexity of 91.44 and 1024 hidden units makes it 91.46.

[5]Since D-RNNLMs use language ID information, we also trained a baseline RNNLM with language ID features; this did not help reduce the baseline test perplexities.

[6]To implement SeqGAN, we use code from `https://github.com/LantaoYu/SeqGAN`.

training it again on the code-switched text. This gives significant reductions in test perplexity, bringing it down to 65.96 (from 74.87).

Combination: Finally, we combine both our proposed techniques by replacing the generator with our best-trained D-RNNLM within SeqGAN. Although there are other ways of combining both our proposed techniques, e.g. pretraining a D-RNNLM using data sampled from an RNNLM SeqGAN, we found this method of combination to be most effective. We see modest but consistent improvements with D-RNNLM SeqGAN over RNNLM SeqGAN in Table 7.7, further validating the utility of D-RNNLMs.

7.3.3 Using External Knowledge

This section will explain the experiments conducted using two additional resources (1) monolingual text for pretraining and (2) a set of syntactic features used as additional input to the RNNLMs that further improve baseline perplexities. We show that our proposed techniques continue to outperform the baselines albeit with a smaller margin.

Monolingual Data: We used additional monolingual text in the candidate languages (i.e. English and Mandarin) to pretrain the RNNLM and D-RNNLM models. We used transcripts from the Switchboard corpus[7] for English; AIShell[8] and THCHS30[9] corpora for Mandarin monolingual text data. This resulted in a total of ≈3.1 million English tokens and ≈2.5 million Mandarin tokens.

The summary of improvements in perplexity by using monolingual data is summarized in the third and fourth columns of Table 7.7. We observe a great drop in perplexity for baseline and D-RNNLM numbers and significant drop for other settings as well. The best development and test perplexities under this setting are 72.33 and 60.30 respectively obtained by training by D-RNNLM SeqGAN pre-training.

Semantic and Syntactic features: We used an additional set of input features to the RNNLMs and D-RNNLMs that were found to be useful for code-switching in prior work [2]. The feature set included part-of-speech (POS) tag features and Brown word clusters [8], along with a language ID feature. We extracted POS tags using the Stanford POS-tagger[10] and we clustered the words into 70 classes using the unsupervised clustering algorithm by [8] to get Brown cluster features.

[7] http://www.openslr.org/5/
[8] http://www.openslr.org/33/
[9] http://www.openslr.org/18/
[10] `https://nlp.stanford.edu/software/tagger.shtml`

Improvements by using syntactic features only under various settings are summarized in the fifth and sixth columns of Table 7.7. We see relatively significant perplexity drops as compared to the simple baseline. The drops are lesser than those compared to using additional monolingual data though.

Combination: We combine both the variants of using additional monolingual data and using semantic and syntactic features whose results are summarized in the last two columns. We use the similar taggers to extract these features for the additional data. We observe that the best improvements are recorded for this setting with the best perplexity numbers of 67.79 and 55.60 for development and test respectively using D-RNNLM SeqGAN pre-training.

Overall, the last six columns in Table 7.7 show the utility of using either one of these resources or both of them together (shown in the last two columns). The perplexity reductions are largest (compared to the numbers in the first two columns) when combining both these resources together. Interestingly, all the trends we observed in Section 7.3.2 still hold. D-RNNLMs still consistently perform better than their RNNLM counterparts and we obtain the best overall results using D-RNNLM SeqGAN.

Chapter 8

Discussion

Code-switched data corpora tend to exhibit very different linguistic characteristics compared to standard monolingual corpora, possibly because of the informal contexts in which code-switched data often occurs, and also possibly because of the difficulty in collecting such data. It is possible that the gains made by our language model are in part due to such characteristics of the corpus we use, SEAME. (We note that this corpus is by far the most predominant one used to benchmark speech recognition techniques for code-switched speech.)

8.1 N-gram LMs

In this section we analyze the SEAME corpus and try to further understand our results in the light of its characteristics in context of n-gram LMs.

8.1.1 Code-switching boundaries

Code-switched bigrams with counts of ≤ 10 occupy 87.5% of the total number of code-switched bigrams in the training data. Of these, 55% of the bigrams have a count of 1. This suggests that context across code-switching boundaries cannot significantly help a language model built from this data. Indeed, the DLM construction in this work discards such context, in favor of a simpler model.

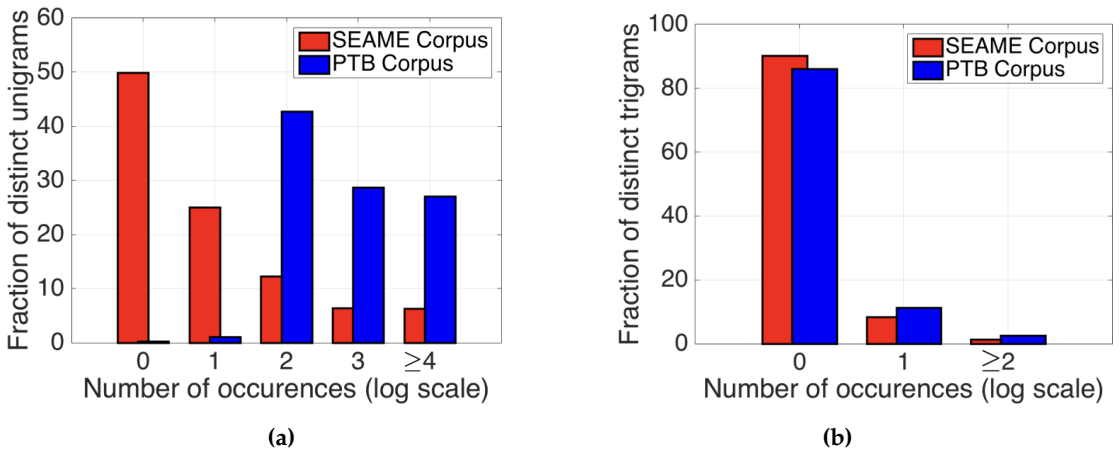

Figure 8.1: *Comparison of fraction of data vs frequency of n-grams in code-mixed text and monolingual text.*

8.1.2 *n*-gram token distribution

We compare the unigram distribution of a code-switched corpus (SEAME corpus) with a standard monolingual corpus (PTB corpus). A glaring difference is observed in their distributions (Figure 8.1-a) with significantly high occurrence of less-frequent unigrams in the code-switched corpus, which makes them rather difficult to capture using standard *n*-gram models (which often fall back to a unigram model). The DLM partially compensates for this by emulating a "class-based language model", using the only class information readily available in the data (namely, the language of each word).

8.1.3 Illustrative examples

Below, we analyze perplexities of the mixed LM and the DLM on some representative sentences from the SEAME corpus, to illustrate how the performances of the two models compare.

Sentence	*Mixed LM* perplexity	*DLM* perplexity
我们 的 total 是 五十七	920.8	720.4
哦 我 没有 meeting 了	92.2	75.9
okay kay 让 我 拿出 我 的 calculator	1260.3	1284.6
the roomie lives in serangoon right	2302.1	1629.3
oh 他 拿 third class 他 差 一点点 他的 f. y. p. screwed up 他 拿 到 b minus c plus	299.7	257.1

We observe that when less frequent words appear at switching points (like total, meeting, etc.),

the DLM outperforms the mixed LM by a significant margin as illustrated in the first two sentences above. In cases of highly frequent words occurring at switching points, the DLM performs on par with or slightly worse than the mixed LM, as seen in the case of the third sentence. The DLM also performs slightly better within long stretches of monolingual text as seen in the fourth sentence. On the final sentence, which has multiple switches and long stretches of monolingual text, again the DLM performs better. As these examples illustrate, DLMs tend to show improved performance at less frequent switching points and within long stretches of monolingual text.

8.1.4 Effect of Trigrams

In standard monolingual datasets, trigram models consistently outperform bilingual models. However, in our code-switched corpus we do not find a pronounced difference between a bigram and a trigram model. This could be attributed to the fact that the number of highly frequent trigrams in our corpus is lesser in comparison to that in the monolingual dataset (PTB) (Figure 8.1-b). Further, in our ASR experiments we observe that using the trigram model adversely affects performance (e.g., the error rate for trigram LM (TDNN+SAT) on test set is 37.61!) As such, we have focused on bigram LMs in this work.

8.2 RNNLMs

8.2.1 Perplexity Analysis

Table 8.1 shows how the perplexities on the development set from six of our prominent models decompose into the perplexities contributed by English tokens preceded by English tokens (Eng-Eng), Eng-Man, Man-Eng and Man-Man tokens. This analysis reveals a number of interesting observations. 1) The D-RNNLM mainly improves over the baseline on the "switching tokens", Eng-Man and Man-Eng. 2) The RNNLM with monolingual data improves most over the baseline on "the monolingual tokens", Eng-Eng and Man-Man, but suffers on the Eng-Man tokens. The D-RNNLM with monolingual data does as well as the baseline on the Eng-Man tokens and performs better than "Mono RNNLM" on all other tokens. 3) RNNLM SeqGAN suffers on the Man-Eng tokens, but helps on the rest; in contrast, D-RNNLM SeqGAN helps on *all tokens* when compared with the baseline.

8.2.2 SeqGAN Text Quality

As an additional measure of the quality of text generated by RNNLM SeqGAN and D-RNNLM SeqGAN, in Table 8.2, we measure the diversity in the generated text by looking at the increase in

	Eng-Eng	Eng-Man	Man-Eng	Man-Man
RNNLM	133.18	157.18	2617.28	34.98
D-RNNLM	140.37	151.38	2452.16	32.89
Mono RNNLM	101.61	181.28	2510.48	30.00
Mono D-RNNLM	101.66	156.44	2442.81	29.64
RNNLM SeqGAN	120.28	154.44	2739.85	30.40
D-RNNLM SeqGAN	120.26	149.68	2450.85	30.60

Table 8.1: *Decomposed perplexities on the development set on all four types of tokens from various models.*

the number of unique n-grams with respect to the SEAME training text. D-RNNLM SeqGAN is clearly better at generating text with larger diversity, which could be positively correlated with the perplexity improvements shown in Table 7.7.

	SeqGAN-RNNLM	SeqGAN-DLM
Bigram	25.57	31.33
Trigram	75.88	83.86
Quadgram	137.98	145.71

Table 8.2: *Percentage of new n-grams generated.*

Chapter 9

Other Explorations and Future Work

We introduced DLMs and showed robust improvements over mixed LMs in perplexity for code-switched speech. While the performance improvements for the ASR error rates are modest, they are achieved without the aid of any external language resources and without any computational overhead. We observe significant ASR improvements via lattice combination of DLMs and the standard mixed LMs. A future direction would be to investigate properties of code-switched text which can be incorporated in our DLM setting.

D-RNNLMs and same-source pretraining provide significant perplexity reductions for code-switched LMs. These techniques may be of more general interest. Leveraging generative models to train LMs is potentially applicable beyond code-switching; D-RNNLMs could be generalized beyond LMs, e.g. speaker diarization. We leave these for future work to explore.

There are some other ideas which seem relevant and can be explored. Using lower-levels of representations like bytes seem to improve performance in various applications like speech recognition in low-resource setting. We used byte-stream inputs for RNNLMs to explore this idea in our domain but the results were poor as compared to the baseline RNNLM.

Other ideas which can be explored include improving heuristics to generate better artificially code-switched data, which can act as a secondary source of independent code-switched data. Another way to generate code-switched data is to use MaskGAN [13], another variant of generative models which uses a actor-critic variant of REINFORCE.

Bibliography

[1] Heike Adel, Katrin Kirchhoff, Ngoc Thang Vu, Dominic Telaar, and Tanja Schultz. Comparing approaches to convert recurrent neural networks into backoff language models for efficient decoding. In *Proceedings of Interspeech*, 2014.

[2] Heike Adel, Dominic Telaar, Ngoc Thang Vu, Katrin Kirchhoff, and Tanja Schultz. Combining recurrent neural networks and factored language models during decoding of code-switching speech. In *Proceedings of Interspeech*, 2014.

[3] Heike Adel, Ngoc Thang Vu, Katrin Kirchhoff, Dominic Telaar, and Tanja Schultz. Syntactic and semantic features for code-switching factored language models. *Proceedings of IEEE Transactions on Audio, Speech, and Language Processing*, 23(3):431–440, 2015.

[4] Peter Auer. *Code-switching in conversation: Language, interaction and identity*. Routledge, 2013.

[5] Ashutosh Baheti, Sunayana Sitaram, Monojit Choudhury, and Kalika Bali. Curriculum design for code-switching: Experiments with language identification and language modeling with deep neural networks. In *Proceedings of ICON*, pages 65–74, 2017.

[6] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of NIPS*, pages 1171–1179, 2015.

[7] K Bhuvanagiri and Sunil Kopparapu. An approach to mixed language automatic speech recognition. *Proceedings of Oriental COCOSDA, Kathmandu, Nepal*, 2010.

[8] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[9] Özlem Çetinoglu, Sarah Schulz, and Ngoc Thang Vu. Challenges of computational processing of code-switching. *EMNLP 2016*, page 1, 2016.

[10] Joyce YC Chan, PC Ching, Tan Lee, and Helen M Meng. Detection of language boundary in code-switching utterances by bi-phone probabilities. In *Proceedings of International Symposium on Chinese Spoken Language Processing*, pages 293–296. IEEE, 2004.

[11] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

[12] Zhiyong Zhang Dong Wang, Xuewei Zhang. THCHS-30 : A free Chinese speech corpus, 2015.

[13] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the _. *arXiv preprint arXiv:1801.07736*, 2018.

[14] Saurabh Garg, Tanmay Parekh, and Preethi Jyothi. Dual language models for code switched speech recognition. In *Proceedings of Interspeech*, 2018.

[15] Dongji Gao Yiming Wang Ke Li Nagendra Goel Yishay Carmie Daniel Povey Sanjeev Khudanpur Hainan Xu, Tongfei Chen. A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In *Proceedings of ICASSP*. IEEE, 2017.

[16] David Imseng, Hervé Bourlard, Mathew Magimai Doss, and John Dines. Language dependent universal phoneme posterior estimation for mixed language speech recognition. In *Proceedings of ICASSP*, pages 5012–5015, 2011.

[17] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[18] Ying Li and Pascale Fung. Improved mixed language speech recognition using asymmetric acoustic model and language model with code-switch inversion constraints. In *Proceedings of ICASSP*, pages 7368–7372. IEEE, 2013.

[19] Ying Li and Pascale Fung. Language modeling with functional head constraint for code switching speech recognition. In *Proceedings of EMNLP*, 2014.

[20] Ying Li, Pascale Fung, Ping Xu, and Yi Liu. Asymmetric acoustic modeling of mixed language speech. In *Proceedings of ICASSP*, pages 5004–5007, 2011.

[21] Dau-Cheng Lyu, Tien-Ping Tan, Eng-Siong Chng, and Haizhou Li. An analysis of a Mandarin-English code-switching speech corpus: SEAME. *Proceedings of Age*, 21:25–8, 2010.

[22] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[23] Shana Poplack. Sometimes ill start a sentence in spanish y termino en espaol. In *Proceedings of Linguistics*, pages 581–618, 1990.

[24] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *Proceedings of ASRU*, 2011.

[25] David Sankoff. A formal production-based explanation of the facts of code-switching. In *Proceedings of Bilingualism: language and cognition*, pages 39–50, 1998.

[26] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of Interspeech*, 2002.

[27] Ngoc Thang Vu, Dau-Cheng Lyu, Jochen Weiner, Dominic Telaar, Tim Schlippe, Fabian Blaicher, Eng-Siong Chng, Tanja Schultz, and Haizhou Li. A first speech recognition system for Mandarin-English code-switch conversational speech. In *Proceedings of ICASSP*, pages 4889–4892. IEEE, 2012.

[28] Robert L Weide. The CMU pronouncing dictionary. *URL: http://www. speech. cs. cmu. edu/cgibin/cmudict*, 1998.

[29] Michael Wick, Pallika Kanani, and Adam Craig Pocock. Minimally-constrained multilingual embeddings via artificial code-switching.

[30] Ching Feng Yeh, Chao Yu Huang, Liang Che Sun, and Lin Shan Lee. An integrated framework for transcribing Mandarin-English code-mixed lectures with improved acoustic and language modeling. In *Proceedings of Chinese Spoken Language Processing (ISCSLP), 2010 7th International Symposium on*, pages 214–219. IEEE, 2010.

[31] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of AAAI*, pages 2852–2858, 2017.