Curriculum Design for Code-switching: Experiments with Language Identification and Language Modeling with Deep Neural Networks

Ashutosh Baheti, Sunayana Sitaram, Monojit Choudhury, Kalika Bali
Microsoft Research Lab, India
ashutosh.baheti95@gmail.com
{t-susita, monojitc, kalikab}@microsoft.com

Abstract

Curriculum learning strategies are known to improve the accuracy, robustness and convergence rate for various language learning tasks using deep architectures (Bengio et al., 2009). In this work, we design and experiment with several training curricula for two tasks – word-level language detection and language modeling – for *code-switched* text data. Our study shows that irrespective of the task or the underlying DNN architecture, the best curriculum for training the code-switched models is to first train a network with monolingual training instances, where each mini-batch has instances from both languages, and then train the resulting network on code-switched data.

1 Introduction

Code-switching (CS) refers to the linguistic phenomenon of fluid alternation between two or more languages during a single conversation or even an utterance (Myers-Scotton, 1993). It is observed in all stable multilingual societies (Auer, 1995) and recent studies have shown that social media posts from such societies almost always contain small to moderate amount of CS (Bali et al., 2014; Dorleijn, 2016; Molina et al., 2016; Rudra et al., 2016; Rijhwani et al., 2017). For instance, Rijhwani et al. (2017) shows that 2-12% (3.5% on average) of the tweets from the cities around the world are code-switched. It is therefore imperative to build speech and text processing technologies that can handle CS. Indeed, quite some amount of effort is being invested towards technology for CS (see Diab et al. (2014; 2016), Sharma et al. (2015), and references therein).

It is of theoretical and practical interest to ponder on the question: whether for a particular NLP task (say ASR, MT or POS Tagging), it is possible to build CS models only from pretrained monolingual models or monolingual training data? Indeed, several studies in the past (Solorio and Liu, 2008; Vyas et al., 2014; Gadre et al., 2016; Gonzalez-Dominguez et al., 2015) have proposed techniques for combining monolingual models or training data coupled with a little amount of CS data to build models of CS text or speech. These techniques have reported promising results. However, all these studies, except (Johnson et al., 2016; Rijhwani et al., 2017; Chan et al., 2009), have tried to combine the outputs of pre-trained monolingual models in intelligent ways. On the other hand, one might ask whether a single system trained on monolingual data from both the languages would be able to handle CS between these languages? And, if we also had a little amount of CS data, how best to use it during the training process?

In this paper, we explore various training strategies, also known as *Curriculum* (Bengio et al., 2009) for DNN-based architectures for code-switching. In particular, we design a set of strategies or curricula involving various ordering of the monolingual and CS data. We experiment with these curricula for *Language Identification* (LID) and *Language Modeling* (LM) tasks. Our study shows that the best curriculum across the tasks as well as DNN architecture is the same one: first train a network with monolingual instances alternating between the languages, and then train the resultant network with CS data, if available. The models trained solely with monolingual data also achieve reasonably high accuracies.

As far as we know, this is the first study on curriculum design for CS. Our study has two important implications: first, it shows that it is possible to train models for CS using primarily monolin-

gual data; this obviates the need for creation of large amount of CS datasets. Second, it also brings out the fact that training curriculum is extremely important while building CS models from monolingual data, and there seems to be an ideal way of ordering the training examples that works best across tasks and network structures.

2 Background and Motivation

In this section, we present a typology of the monolingual model combination strategies for CS, through which we will motivate the central idea of this work.

2.1 A Note on Terms

It is important to differentiate between inter-sentential and intra-sentential CS. The former refers to a situation where each sentence (or sometimes clause) is in a single language, but the language might change across the sentences. On the other hand, intra-sentential CS, which is also sometimes called *Code-mixing*, refers to a situation where words in the same sentence/clause can be drawn from multiple languages.

Tasks that operate on sentence level context (like POS tagging, ASR and MT) do not require any special technique for handling inter-sentential CS, except LID and sentence boundary detection. However, intra-sentential CS is more challenging to handle, and will be our primary focus. In this paper, the terms *monolingual model* and *monolingual data* will be used for cases where the data was collected and the model was built assuming that the input will be only in a single language. Such datasets might also contain some borrowed words and text in other language(s). On the other hand, we will use the term *CS data* to imply datasets where all instances contain intra-sentential CS, even though most of the datasets released in the past for training CS models, e.g., (Molina et al., 2016; Das, 2016; Sequiera et al., 2015b), do contain fair amounts of monolingual and inter-sentential CS. The term *CS model* will be used for systems that can handle monolingual, inter-sentential as well as intra-sentential CS.

2.2 A Taxonomy of CS Models

In order to succinctly represent the various types of CS models proposed in the literature, we will use the following notation. Let l_1 and l_2 be two languages. Let x denote the input string, usually a sentence, i.e., string of tokens, in l_1, l_2 or l_{12}, i.e., $l_1 \leftrightarrow l_2$ code-switched. Let y be the output string of tokens in a target language (as in MT, ASR or POS tagging). Let g_i and f_i denote models trained on data from l_i. Further, we describe a special function $lid(x)$ which returns the string of language labels for each word; $lid_1(x)$ and $lid_2(x)$ are projection functions which returns only those tokens of x that are in l_1 and l_2 respectively.

CS models described in the literature can be broadly categorized into the following four classes (in descending order of amount of CS data required for training).

Purely Supervised Models: When a large amount of annotated CS data is available, a supervised model can be learnt simply from the monolingual and CS data. Thus,

$$y = g_{1\cup2\cup12}(x) \tag{1}$$

These models often use features or extra information specific to CS, but do not particularly modify the training process or system architecture for handling CS. This approach has been applied to language identification, e.g., most submissions in the LID shared task in the Computational Approaches to Code-Switching Workshops (Solorio et al., 2014; Molina et al., 2016); to POS tagging, e.g., most submissions in the ICON 2016 shared task on CS POS tagging (Das, 2016) and also (Jamatia and Das, 2014; Jamatia et al., 2015); and to ASR (Gebhardt, 2011).

Combining Monolingual Models: In this approach, the output of two monolingual systems on x is used as features for a third model (f_{12} in Eq. 2). This third model f is trained on a small amount of CS data, and can use other features which often includes LID output.

$$y = f_{12}(g_1(x), g_2(x), lid(x)) \tag{2}$$

Solorio and Liu (2008) proposed this architecture for POS tagging of English-Spanish CS data, and Lyu et al. (2006) proposed a similar model for ASR. Both reported significant gain over the monolingual models by using very little CS data. Later works, such as (Sequiera et al., 2015a), along this line also reported promising results.

Divide and Conquer: In this approach, the input is first passed through a LID system and split into parts according to the language of the tokens. The token strings are then passed on to the respective monolingual systems and the outputs are combined (shown as the operator $\oplus$ in Eq. 3)

$$y = g_1(lid_1(x)) \oplus g_2(x)(lid_2(x)) \quad (3)$$

This approach does not require any CS training data, but it does not work well for intra-sentential CS because splitting by language can lead to loss of context especially at the code-switch points. However, some benefits of this approach have been shown for POS tagging (Vyas et al., 2014), MT (Gadre et al., 2016) and ASR (Lyudovyk and Pylypenko, 2014) respectively.

Zero Shot Learning: This is an extreme case, where only monolingual data from two or more languages is used to train a single system with the hope that it will work for CS data as well.

$$y = g_{1\cup 2}(x) \quad (4)$$

A recent work (Rijhwani et al., 2017) uses this technique very effectively for developing an LID system for 7 languages. While no annotated CS data is used for training, the system uses unlabeled data that is expected to contain CS data, for unsupervised training. Johnson et al. (2016) trains a neural MT system with data from two pairs of languages, $l_1 \Leftrightarrow l_2$ and $l_1 \Leftrightarrow l_3$ and show that the resultant model not only works for $l_2 \Leftrightarrow l_3$ (the so called "zero shot learning" but also for CS input in these languages, albeit to a limited extent.

Factors such as lack of large-scale CS datasets, possibility of CS between any pair (or even triplet) of languages (which in turn implies the need for nearly a quadratic number of such datasets) and the difficulty in creation of CS datasets owing to the requirement of skilled multilingual annotators make Zero Shot Learning a very lucrative approach CS. However, we do not know of any work that systematically explores the various training strategies and effective use of CS data in the context of Zero Shot Learning.

3 Training Curricula for CS

Originally proposed by (Elman, 1993), *Curriculum learning* refers to a sequence of weight distributions over the training example, such that during the training process certain examples are used with higher weight at the initial stages of the training and other examples are used later (Bengio et al., 2009). In general, it is believed that for complex non-convex optimization problems, training with *simpler* examples first and introducing the complex examples at later stage has distinctive benefits. Empirically, it has shown promising results for several NLP tasks like parsing (Spitkovsky et al., 2009) and language modeling (Bengio et al., 2009; Graves et al., 2017). Shi et al. (2015) describe curricula for domain adaptation of Language Models, in which they order the data such that general data is presented to the RNN first, followed by in-domain data.

In principle, the purely supervised (Eq. 1) and the zero shot learning approaches (Eq. 4) should benefit from curriculum based training. It is well known that proficient bilingual children learn to code-switch without any exposure to CS data (Cantone, 2007). This leads us to explore various curricula for training with monolingual and CS data. While complexity of training instances can be defined across various dimensions, in this study we will restrict ourselves to only one aspect of the curriculum design - the language(s).

Let T_1, T_2 and T_{12} be respectively the set of training examples in l_1, l_2 and intra-sentential CS between l_1 and l_2. We will use the notation $T_i; T_j$ to indicate a basic curriculum where the system is trained with all instances from T_i first, and then with instances from T_j. Similarly, $\{T_i, T_j\}$ will be used to indicate the curiculum where the system is trained with instances from T_i and T_j simultaneously; in the context of deep learning, this means each *mini-batch* contains samples from T_i and T_j (ideally, but not necessarily, in a ratio $|T_i| : |T_j|$).

Based on the ordering of the training instances, we define 7 different curricula: (C1) $T_1; T_2$ (C2) $T_{12}; T_1; T_2$ (C3) $T_1; T_2; T_{12}$ (C4) $\{T_1, T_2\}$ (C5) $T_{12}; \{T_1, T_2\}$ (C6) $\{T_1, T_2\}; T_{12}$ (C7) $\{T_1, T_2, T_{12}\}$. We consider the curricula (C0) T_{12} (i.e., training with only CS data) and C7 (i.e., all instances randomized[1]) as the two baselines.

In the next two sections, we will describe experiments with these curricula for deep learning models applied to two tasks – Language Identification (LID) and Language Modeling (LM), both for English (En) and Spanish (Es) CS.

4 Language Identification

Along the lines of (Rijhwani et al., 2017), we define word-level LID as a sequence labeling problem, where each token in the input sentence is labeled with one of the three tags: l_1, l_2 and X (meaning "none of the languges", such as numbers, punctuations, urls and hashtags). In the

[1]Strictly speaking, in C7 we ensure that in each mini-batch there are training instances from T_1, T_2 and T_{12} in certain fixed ratio.

Language	Train	Dev	Test
En	1240k	1006	13542
Es	1240k	1119	4874
En-Es	17.5k	750	8678

Table 1: Datasets for LID (in number of words).

following subsections, we describe the datasets, DNN architecture, and experimental results.

4.1 Datasets

All our experiments are done on En and Es tweets, which are primarily drawn from two existing datasets: (Rijhwani et al., 2017) for monolingual training data, and (Solorio et al., 2014) for CS training data, and all dev and test datasets. Since we define LID as a classification problem, we consider each word with its context as an instance (instead of each tweet as an instance). Further, we differentiate between CS and monolingual instances as those where the context (a window of $2k$ words around the target word) has or does not have a code-switch point, respectively. Table 1 summarizes the size of the datasets. There are total 1328 switching points in the CS test data.

4.2 DNN Architecture

Fig. 1 shows the architecture of the DNN for LID. The model takes $2k + 1$ word window input with target word at the center, and predicts the language of the target word. In this word-context block, all the input words are projected into a d_w dimensional word-embedding space. The embedding vectors of the k words in the left and right contexts are averaged separately. These left-average, right-average and the current word embedding are merged into one $3d_w$ dimensional vector. This vector is passed to an intermediate Dense layer (with ReLu activation) of d_{im} dimension. To speed up the convergence of the network, we add a batch normalization layer. Also, a dropout layer with 0.3 probability is introduced to prevent overfitting. Finally, a softmax layer with two nodes, one for each language, l_1 and l_2, is used for predicting the output.

We define another architecture - the *Char-LID model* where this basic LID model is augmented with a character-context block (as shown in the dashed box in the Fig. 1). We embed character tri-grams into a d_c dimensional space; an average of all the character tri-grams of the target word is then concatenated with the embeddings of

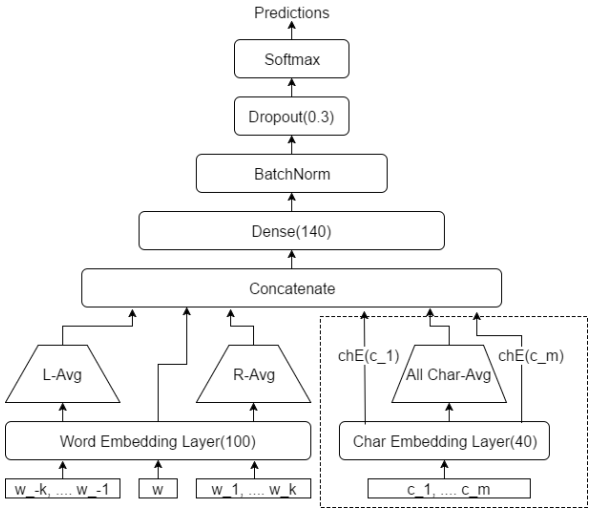

Figure 1: DNN achitecture for LID

the first and last trigrams to generate a $3d_c$ dimensional character-representation vector of the target word. In the Char-LID model, this vector is concatenated to the aforementioned $3d_w$ dimensional word vector, and is fed as the input to the intermediate dense layer. Rest of the network is identical to the LID model. We expect the Char-LID model to work better for out-of-vocabulary words.

4.3 Experiments

The networks were implemented, trained and tested using the Microsoft's Cognitive Toolkit (CNTK)(Yu et al., 2014). For all our experiments, we set[2] $k = 3$, $d_w = 100$, $d_c = 40$, $d_{im} = 140$, and dropout rate = 0.3. The networks are trained using mean square error loss and Adadelta SGD under default parameter settings of CNTK. The word and character vectors were initialized uniformly randomly.

Note that we do not use the DNN to predict the X labels, as these are identified through regular expressions during the pre-processing of the data and are never used as target words during training or testing. Details of how we handle the special and boundary cases (e.g., out-of-vocabulary words and contexts for target words in the beginning and end of a sentence) are discussed in the supplementary material.

For each curriculum, we train the models for 20 epochs and choose the model that has maximum overall accuracy on the dev set. For curricula involving interleaving of instances of different types (e.g., C4 to C7), we presented the training data

[2]Experiments with different values of these parameters led us to these numbers which seem to work well.

Curriculum	All	En-Es	En	Es
LID Model				
C0: T_{12}	89.1	86.6	89.1	93.4
C1: $T_1; T_2$	37.0	48.3	17.8	70.3
C2: $T_{12}; T_1; T_2$	35.4	54.2	0.1	100
C3: $T_1; T_2; T_{12}$	84.1	78.0	95.5	63.6
C4: $\{T_1, T_2\}$	94.7	87.0	99.6	96.3
C5: $T_{12}; \{T_1, T_2\}$	94.8	86.7	99.3	96.7
C6: $\{T_1, T_2\}; T_{12}$	**95.1**	**89.1**	98.8	95.8
C7: $\{T_1, T_2, T_{12}\}$	94.4	87.7	97.5	97.4
Char-LID Model				
C4: $\{T_1, T_2\}$	95.5	87.6	99.7	97.7
C5: $T_{12}; \{T_1, T_2\}$	95.5	87.3	99.7	98.4
C6: $\{T_1, T_2\}; T_{12}$	**97.1**	**93.7**	98.7	98.3
C7: $\{T_1, T_2, T_{12}\}$	96.2	89.6	99.7	98.2

Table 2: Curriculum training accuracies (in %) for LID and Char-LID models. The maximum accuracy for the models are in bold and are statistically significantly higher ($p < 0.001$) than all other values in the column for that model.

to the network in randomized order. Thus, every minibatch is expected to contain the instances in ratio $|T_1| : |T_2|$. Since the $|T_{12}|$ training data is significantly low compared to $|T_1|$ and $|T_2|$, we oversample T_{12} by replicating the data 10 times for curriculum C7. For curricula that involves ordering of inputs by blocks (all except C4 and C7), we first train on the first block of instances for 20 epochs and choose the best model which is trained on the next block of instances for 20 epochs.

4.4 Results

In Table 2 we report the l_1 and l_2 labeling accuracy on the En, Es and En-Es CS test sets, as well as the combined accuracy (column 1) on all the test instances. Due to the significantly poorer performance of C1, C2 and C3 for the LID model, these experiments were not conducted for the char-LID model. For C0, overall accuracy for the Char-LID model is 94.5% (Table 3).

The key observations from Table 2 are: (a) curriculum C6 is most effective across the models; the performance on CS set increases significantly with only a marginal drop in accuracy on the monolingual data; (b) C6 achieves an 11% (55%) and 23% (47%) error reduction over the baseline curriculum C7 (C0) for the LID and Char-LID models respectively; (c) The improvements are highly significant at $p < 0.001$ for a paired t-test, which implies that C6 is able to correct labeling errors made by C7 and other curricula, and hardly makes new labeling errors; (d) Providing the CS training data as the last block is much more effective than providing it in the beginning or distributing it over the entire training curriculum; (e) On the other hand, mixing of monolingual data is more effective than providing them in block; (f) Finally, it is also interesting to note that curriculum C4, where only monolingual training data is used, achieves reasonably high accuracies.

We manually inspected around 100 erroneously labeled words by the best C6 model. There are three noticeable error patterns: (a) errors around an X tag (approx. 15%), (b) errors at or near the interjections such as "haha", "jaja", "lmao" etc. (approx. 15%), and (c) errors near the code-switch point and sentence boundaries (approx. 25%). Rest of the errors didn't have any noticeable pattern, though we also discovered that some of the system labels classified as errors were actually correct and rather the gold standard label was incorrect (approx. 5%).

We also conduct two auxiliary sets of experiments to understand the effect of the ratio of monolingual training instances ($|T_1| : |T_2|$) and that of the CS data to monolingual data ($|T_{12}| : |T_1 \cup T_2|$). In the first experiment, we train the LID model where we vary the percentage of training instances used from the En and Es monolingual datasets. The results are shown in Fig. 2. In the x-axis, we plot the percentage of training instances used from the En and Es training sets during each experiment, where En fractions (the first value in the tuple) increases from right to left, Es fractions (the second value) from left to right. It is evident from the plot that the system performance is not strongly sensitive to the ratio of $|T_1| : |T_2|$. Rather, it is more sensitive to the absolute amount of data available for training; when the data for either T_1 or T_2 drops significantly (less than 1% of the training set here, as in the extremes of the plot), the accuracy is affected significantly.

In the second set of experiments, we train the Char-LID model with curriculum C7. We vary the monolingual and CS training data independently and report the accuracies on the entire test set for each setup in Table 3. The trends, as expected, shows diminishing marginal utility for both monolingual and CS datasets. Nevertheless, we note that the marginal utility of monolingual data is

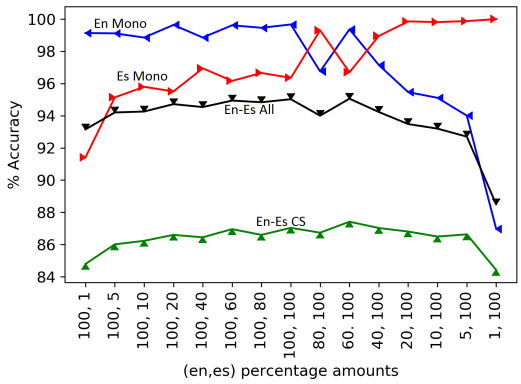

Figure 2: en-es Skew experiment using Curriculum C4 in LID model

% of $\lvert T_{12} \rvert$	% of $\lvert T_1 \rvert$ and $\lvert T_2 \rvert$			
	0	**0.1**	**1**	**10**
20	89.5	93.3	94.4	95.8
50	92.5	93.6	95.4	95.9
100	94.5	93.3	95.1	96.1

Table 3: Overall accuracy of the Char-LID model for C7 with varying amounts of training data.

much more pronounced (i.e., systematic increase of accuracy in each row from left to right) than that of CS data. This could be because of much higher sizes of the monolingual training sets as compared to the CS dataset.

We have also conducted some of these experiments with English-French CS data, which shows similar trends. Reader may refer to the supplementary material for more details.

4.5 Related Work on LID

There have been many works on LID for CS. See (Solorio et al., 2014; Molina et al., 2016; Rijhwani et al., 2017) and references therein. Two systems (Samih et al., 2016; Jaech et al., 2016) submitted in the shared task on language detection in EMNLP 2016 use deep learning based techniques. Samih et al. (2016) uses LSTMs on top of word and character context with a CRF classifier and achieves an accuracy of 96.3% on the same En-Es test set. Jaech et al. (2016) uses only the character sequence data with stacked CNNs to create a word embedding. Then it creates a global context by adding a bi-directional LSTM on top of it. Their system achieves 94.6 average F1 score for En-Es. Chang and Lin (2014 describes an RNN based system which is trained and tested on the EMNLP 2014 shared task dataset. This system outperforms all the submitted systems in that shared task.

The EMNLP shared task dataset had 6 labels including named entities and mixed language words which we did not consider in this work. Therefore, even though we evaluate on the same dataset, the accuracies are not directly comparable. However, since majority of the tokens are labeled En and Es, our results are certainly comparable to these state-of-the-art systems. Rijhwani et al. (2017) uses HMM model initialized by monolingual data and retrained it on unlabeled data using Baum-Welch algorithm. It achieves an average F-1 score of 97.8% for En and Es labels, which is only slightly better than our best performing system. However, the models described here do not use the unlabeled data, incorporating which could be an interesting future direction for this research.

5 Language Modeling

Statistical Language Models estimate the probability of a word sequence given a large training corpus. Language Modeling has applications in various NLP and Speech processing tasks, most notably in MT and ASR. In this section, we describe experiments on building En-Es code-switched LM.

5.1 Datasets

Similar to our LID experiment datasets described in Table 1, we use the En and Es monolingual tweets from (Rijhwani et al., 2017) (described as the *weakly-labeled data* in the paper), and the language-labeled CS data from (Solorio et al., 2014) for training. However, unlike the LID experiments, here a tweet, rather than a word, is considered as an instance; any tweet with CS is a part of the CS training instance. Since we do not need the language labels for the LM training experiments, the tweets were stripped off those labels. We used 212k En tweets, 92k Es tweets and 798 En-Es CS tweets as training data. The amount of CS data used is a very small fraction (0.2%) of the size of the monolingual data and the amount of En data is more than double the Es data.

We also created a held-out test set for the LM experiments, which consists of 2200 En-Es tweets that were automatically tagged as code-switched by our best En-Es LID system described in the previous section. We use the standard evaluation metric – *perplexity* (PPL) on the held-out test set

– to compare the LMs (lower the better).

5.2 Models

We train both RNNLMs and ngram language models on the same data to compare their performances on various training data curricula.

We use the RNNLM toolkit (Mikolov et al., 2011) to train and test all our RNNLM models. The ngram models are also built with the same toolkit, which invokes the SRILM toolkit (Stolcke et al., 2002). During our initial experiments on finding the appropriate curriculum for training CS LMs, we use the single iteration recipe provided in the RNNLM toolkit. In this setting, the learning rate is manually set to decrease after 4 epochs and no validation data is used.

We then use the regular LM training recipe in the RNNLM toolkit that makes use of a validation data set. We adjust the values of various hyperparameters in our experiments. One crucial parameter we adjust is the number of classes (700), according to a rule of thumb saying that the number of classes should be approximately the square root of the vocabulary size. By doing this, we sacrifice accuracy for speed of training our models; since the aim here is to analyze the trends rather than look at the absolute values, we believe this is a reasonable policy. Our models have a very large vocabulary size owing to the presence of two languages and also due to large amount of spelling variations found in tweets. For the experiments with a single iteration, our models have 150 hidden layers. Other hyper parameters are kept at their default values in the corresponding recipes in the RNNLM toolkit. In our final experiments with the full iteration recipe, we used CS data as validation and 100 hidden layers.

5.3 Experiments

We experiment with all the training curricula described in Sec. 3, except C5 because from our experiments with C2 and C3, we find that adding CS data at the beginning produces worse results. For experiments involving $\{T_1, T_2\}$ (i.e., curricula C4 and C6), we provide the input by interleaving the tweets from T_1 and T_2. Since En has almost twice as many tweets as Es, the extra En tweets that remain after interleaving all the Es tweets, are simply appended to the training set. Finally, since we have very little CS data as compared to the monolingual data, for curriculum C7 we replicate the entire T_{12} dataset after every 10k instances from $\{T_1, T_2\}$. Thus, we use 30 replicas of $\{T_{12}\}$ in this curriculum. In addition to these curricula, we also build a baseline using only CS data (C0).

We build all the six models using the RNNLM single iteration setting first. Then, we build models for the best performing curriculum using the multiple iteration recipe. In addition, we build 5-gram models for all the 6 curricula and the C0 baseline. However, since ngram models do not depend on the ordering, there are only four unique corpora for training them: (1) with only monolingual data: $T_1 \cup T_2$, which is comparable to the models for curricula C1 and C4; (2) with monolingual and CS data without replication: $T_1 \cup T_2 \cup T_{12}$, which is comparable to RNNLMs built using curricula C2, C3 and C6; and (3) monolingual data and with replicated CS data[3]: $T_1 \cup T_2 \cup 30 \cdot T_{12}$, which is comparable to the RNNLM trained using curriculum C7 (4) only code-switched data T_{12}, which is comparable to the RNNLM trained using only CS data (comparable to C0).

5.4 Results

Table 4 shows the results of the LM experiments in terms of perplexity on the held-out test set. As we see from the numbers in first column, the best performing RNNLMs are those that are trained initially with monolingual data and then trained with CS data (C3 and C6). The model that is trained initially with CS data and then with monolingual data (C2) performs as badly as the model that was trained with only monolingual data in blocks (C1). In addition, training models with alternate monolingual sentences gives better performance (C4, C5) than training it with large chunks of monolingual text (C1, C2). Training with monolingual and CS data using curricula C3, C6 and C7 outperforms the CS-data only baseline (C0). Also, C4, that uses alternate sentences of monolingual data outperforms this baseline, probably due to the large difference in data size between the monolingual and CS data.

Although the PPL values for the RNNLM are very high, one must note that the test set consists entirely of CS sentences, whereas the amount of "in-domain" (i.e., CS) data used in training is very low (0.2%). To improve our models, we build the best performing model, the one corresponding to C6, with multiple iterations; the PPL value for it is

[3]Since ngrams models strongly depend on frequency of occurrence, we represent this using a slight abuse of notation to indicate 30 replications of the T_{12} set

Curriculum	RNNLM	Intpl.
C0: T_{12}	27443	477
C1: $T_1; T_2$	46628	1120
C2: $T_{12}; T_1; T_2$	44516	609
C3: $T_1; T_2; T_{12}$	7987	358
C4: $\{T_1, T_2\}$	9749	771
C6: $\{T_1, T_2\}; T_{12}$	**6533** (4544)	**320** (298)
C7: $\{T_1, T_2, T_{12}\}$	23384	673

Table 4: Perplexity results for RNNLM and interpolated RNNLM+ngram LM. Values in parenthesis are for the multiple iteration model.

shown within parenthesis.

Table 4 (column 2) also shows the PPL for models obtained by interpolating the probabilities given by RNNLM and the 5-gram LM. The interpolation coefficient is kept fixed at 0.5. In all cases, the PPL of the ngram model is significantly lower than the RNNLM. For C1/C4, the 5-gram PPL is 915, for C2/C3/C6 it is 778 and for C7 it is 574. However, in all cases, interpolating the ngram model with the RNNLM gives the best results. Experimenting with the interpolation coefficient could potentially give better results.

5.5 Related Work on LM

Language Models for CS have been studied in the context of three main approaches: (a) Bilingual models that combine data from both languages, (b) Factored models that take into account strong indicators of CS like POS and LID, and (c) Models that incorporate linguistic constraints for CS. Bilingual language models are typically trained using pooled text data (Weng et al., 1997). Gebhardt (2011) describes a framework to use Factored Language Models for rescoring n-best lists during decoding. The factors used include POS tags, CS point probability and LID.

In Adel et al.(2014b; 2014a; 2013) recurrent language models built on the same corpus are combined with n-gram based models, or converted to backoff models, giving improvements in perplexity and mixed error rate. Li and Fung (2014) integrates Functional Head constraints for code-switching into the Language Model for decoding a Mandarin-English corpus. Li and Fung (2013) use inversion constraints to predict CS points and integrates this prediction into the decoding process.

Our work is similar to the bilingual model approach in that we pool data from both languages. However, we also add a very small amount of CS data to our models in some of the experiments. In addition, we also focus on the ordering of the monolingual and CS data, which to our knowledge, none of the previous approaches do.

6 Discussion and Conclusion

The experiments presented here on the two tasks and three different DNN architectures show that across all these cases, C6: $\{T_1, T_2\}; T_{12}$ is the most effective curriculum for training CS models. C7: $\{T_1, T_2, T_{12}\}$ also performs quite well, and in absence of CS data, C4: $\{T_1, T_2\}$ seems to be the most effective curriculum. Presenting the monolingual training instances in blocks produce the worst models, and neither is training with the CS data in the beginning any more effective than not using CS data at all.

This is similar to the findings reported in (Shi et al., 2015) in the context of RNNLM adaptation to specific domains. In general, empirical results on transfer learning of DNNs show that training with in-domain data at the end leads to better models. This explains why training first with one language and then another is not ideal; in such cases the model adapts to the second language, as observed for C1, C2 and C3 in Table 2. For similar reasons, training with T_{12} at the beginning provides no benefit. One could possibly argue that during training with $\{T_1, T_2\}$ the upper layers (closer to input) of the networks learns the low level features of the languages. Then during the last phase of training with T_{12}, the lower layers of the network learns when to switch from one language to another, i.e., the CS specific features.

An interesting cognitive metaphor (albeit not an explanation) is as follows: an ideal bilingual is exposed to both the languages almost simultaneously; such a user is able to code-switch even if never exposed to CS (as in C4); however, with exposure to CS, the frequency and perceived-naturalness of CS increases in his/her language use (as in C6). Of course, in multilingual communities children grow up with both languages as well as CS between them (similar to C7). Thus, it is tempting to predict that with large amount of CS training data, i.e., when $|T_{12}| \approx |T_1| \approx |T_2|$, C6 and C7 should perform equally well.

Here, we have explored the ordering of the training instances based on the language. There are other dimensions of complexity, for example the number of code-switch points, syntactic

structure, etc., which could as well be harnessed for more effective curricula. Going forward, we would also like to explore techniques such as Self-paced learning (Kumar et al., 2010; Jiang et al., 2015; Graves et al., 2017).

References

Heike Adel, K. Kirchhoff, N. T. Vu, D. Telaar, and T. Schultz. 2014a. Combining recurrent neural networks and factored language models during decoding of code-switching speech. In *INTERSPEECH*. pages 1415–1419.

Heike Adel, K Kirchhoff, N T Vu, D Telaar, and T Schultz. 2014b. Comparing approaches to convert recurrent neural networks into backoff language models for efficient decoding. In *INTERSPEECH*. pages 651–655.

Heike Adel, N T Vu, and T Schultz. 2013. Combination of recurrent neural networks and factored language models for code-switching language modeling. In *ACL (2)*. pages 206–211.

Peter Auer. 1995. The pragmatics of code-switching: a sequential approach. In L. Milroy and P. Muysken, editors, *One speaker, two languages*, Cambridge Univ Press, pages 115–135.

Kalika Bali, Y. Vyas, J. Sharma, and M. Choudhury. 2014. "I am borrowing ya mixing?" An analysis of English-Hindi code mixing in Facebook. In *Proc. First Workshop on Computational Approaches to Code Switching, EMNLP*.

Yoshua Bengio, J. Louradour, R. Collobert, and J. Weston. 2009. Curriculum learning. In *The 26th annual international conference on machine learning*. ACM, pages 41–48.

Katja F Cantone. 2007. *Code-switching in bilingual children*, volume 296. Springer.

Joyce YC Chan, H. Cao, PC Ching, and T. Lee. 2009. Automatic recognition of cantonese-english code-mixing speech. *Computational Linguistics and Chinese Language Processing* 14(3):281–304.

Joseph Chee Chang and Chu-Cheng Lin. 2014. Recurrent-neural-network for language detection on twitter code-switching corpus. *CoRR* abs/1412.4314.

Amitava Das. 2016. Tool contest on POS tagging for Code-mixed Indian Social Media Text. In *ICON*.

Mona Diab, P. Fung, M. Ghoneim, J. Hirschberg, and T. Solorio, editors. 2016. *Proc.of the 2nd Workshop on Computational Approaches to Code Switching*.

Mona Diab, J. Hirschberg, P. Fung, and T. Solorio, editors. 2014. *Proc. of the 1st Workshop on Computational Approaches to Code Switching*. ACL.

Margreet Dorleijn. 2016. Can internet data help to uncover developing preferred multilingual usage patterns? an exploration of data from turkish-dutch bilingual internet fora. *Journal of Language Contact* 9(1):130–162.

Jeffrey L Elman. 1993. Learning and development in neural networks: The importance of starting small. *Cognition* 48(1):71–99.

Akshay Gadre, R. Begum, K. Bali, and M. Choudhury. 2016. Machine translating code mixed text: Pain points and sweet spots. In *WILDRE*.

Jan Gebhardt. 2011. Speech recognition on english-mandarin code-switching data using factored language models .

Javier Gonzalez-Dominguez, D Eustis, I Lopez-Moreno, A Senior, F Beaufays, and Pedro J Moreno. 2015. A real-time end-to-end multilingual speech recognition architecture. *IEEE Journal of Selected Topics in Signal Processing* 9(4):749–759.

Alex Graves, M G Bellemare, J Menick, R Munos, and K Kavukcuoglu. 2017. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003* .

Aaron Jaech, G Mulcaire, S Hathi, M Ostendorf, and N A Smith. 2016. A neural model for language identification in code-switched tweets. *EMNLP 2016* page 60.

Anupam Jamatia and A Das. 2014. Part-of-speech tagging system for Hindi social media text on twitter. In *The First Workshop on Language Technologies for Indian Social Media, ICON*.

Anupam Jamatia, B Gambck, and A Das. 2015. Part-of-speech tagging for code-mixed english-hindi twitter and facebook chat messages. In *RANLP*.

Lu Jiang, D Meng, Q Zhao, S Shan, and A G Hauptmann. 2015. Self-paced curriculum learning. In *AAAI*. volume 2, page 6.

Melvin Johnson, M Schuster, Q V Le, M Krikun, Y Wu, Z Chen, N Thorat, F Viégas, M Wattenberg, G Corrado, et al. 2016. Google's multilingual neural machine translation system: Enabling zero-shot translation. *arXiv preprint arXiv:1611.04558* .

M Pawan Kumar, B Packer, and D Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*. pages 1189–1197.

Ying Li and P Fung. 2013. Improved mixed language speech recognition using asymmetric acoustic model and language model with code-switch inversion constraints. In *ICASSP*. pages 7368–7372.

Ying Li and P Fung. 2014. Language modeling with functional head constraint for code switching speech recognition. In *EMNLP*.

Dau-Cheng Lyu, R-Y Lyu, Y-C Chiang, and C-N Hsu. 2006. Speech recognition on code-switching among the chinese dialects. In *IEEE ICASSP 2006*. volume 1, pages I–I.

Tetyana Lyudovyk and Valeriy Pylypenko. 2014. Code-switching speech recognition for closely related languages. In *SLTU*. pages 188–193.

Tomas Mikolov, S Kombrink, A Deoras, L Burget, and J Cernocky. 2011. RNNLM-recurrent neural network language modeling toolkit. In *ASRU Workshop 2011*. pages 196–201.

Giovanni Molina, N Rey-Villamizar, T Solorio, F AlGhamdi, M Ghoneim, A Hawwari, and M Diab. 2016. Overview for the second shared task on language identification in code-switched data. *EMNLP 2016* page 40.

Carol Myers-Scotton. 1993. *Dueling Languages: Grammatical Structure in Code-Switching*. Claredon, Oxford.

Shruti Rijhwani, R Sequiera, M Choudhury, K Bali, and C S Maddila. 2017. Estimating code-switching on Twitter with a novel generalized word-level language identification technique. In *ACL*.

Koustav Rudra, S Rijhwani, R Begum, K Bali, M Choudhury, and N Ganguly. 2016. Understanding language preference for expression of opinion and sentiment: What do Hindi-English speakers do on Twitter? In *EMNLP*. pages 1131–1141.

Younes Samih, S Maharjan, M Attia, L Kallmeyer, and T Solorio. 2016. Multilingual code-switching identification via lstm recurrent neural networks. In *2nd Workshop on Computational Approaches to Code Switching,*. pages 50–59.

Royal Sequiera, M Choudhury, and K Bali. 2015a. Pos tagging of Hindi-English code mixed text from social media: Some machine learning experiments. In *Proceedings of ICON*.

Royal Sequiera et al. 2015b. Overview of fire-2015 shared task on mixed script information retrieval. In *FIRE*. pages 21–27.

Shashank Sharma, P Srinivas, and R C Balabantaray. 2015. Text normalization of code mix and sentiment analysis. In *ICACCI, 2015 International Conference on*. IEEE, pages 1468–1473.

Yangyang Shi, M Larson, and C M Jonker. 2015. Recurrent neural network language model adaptation with curriculum learning. *Computer Speech & Language* 33(1):136–154.

Thamar Solorio and Yang Liu. 2008. Part-of-speech tagging for english-spanish code-switched text. In *Proc. of EMNLP*.

Thamar Solorio et al. 2014. Overview for the first shared task on language identification in code-switched data. *In 1st Workshop on Computational Approaches to Code Switching, EMNLP* pages 62–72.

Valentin I Spitkovsky, H Alshawi, and D Jurafsky. 2009. Baby steps: How less is more in unsupervised dependency parsing. *NIPS: Grammar Induction, Representation of Language and Language Learning* pages 1–10.

Andreas Stolcke et al. 2002. SRILM-an extensible language modeling toolkit. In *Interspeech*. volume 2002, page 2002.

Yogarshi Vyas, S Gella, J Sharma, K Bali, and M Choudhury. 2014. POS Tagging of English-Hindi Code-Mixed Social Media Content. In *Proc. EMNLP*. pages 974–979.

Fuliang Weng, H Bratt, L Neumeyer, and A Stolcke. 1997. A study of multilingual speech recognition. In *EUROSPEECH*. Citeseer, volume 1997, pages 359–362.

Dong Yu, A Eversole, M Seltzer, K Yao, Z Huang, B Guenter, O Kuchaiev, Y Zhang, F Seide, H Wang, et al. 2014. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112* .