# Codemix generation with RNN language models

Introduction to NLP (CSE401) Course Project

Akshay Bankar (2019201011)

Gajanan Modi (2019201049)

April 2021

# Contents

# List of Figures

# List of Tables

# Abstract

Code-mixing (CM) refers to the mixing of various linguistic units (morphemes, words, modifiers,phrases, clauses and sentences) primarily from two or more participating grammatical systems within a sentence. The development of CM NLP systems has significantly gained importance in recent times due to an upsurge in the usage of CM data by multilingual speakers. However, this proves to be a challenging task due to the complexities created by the presence of multiple languages together. The complexities get further compounded by the inconsistencies present in the raw data on social media and other platforms.

We explore two methods: One, a baseline neural stack-based RNN language model for CM data of Hindi-English by utilizing pre-existing resources for closely related Hindi-English CM. Second, an RNNLM model with POS tag features added to the input layer. Along with this, we also explore various ways of training the neural networks by defining training curriculum with different ordering of data points.

We used code-mix datasets obtained from various sources and are mentioned in reference section. We also resorted to using CRF-based pre-trained model for creating POS-tagged dataset for training our baseline++ model.

To measure the code-mix generated quality, we used Code-mix index (CMI) and Multilingual Index (M-index). To measure the quality of the model trained, we used perplexity scores.

We observed that the baseline++ model, with POS-tag features used at the, input gives better index values and perplexity scores than the baseline model.

# 1 Introduction

## 1.1 What is Code-mix

Code-Mixing refers to the embedding of linguistic units such as phrases, words, and morphemes of one language into an utterance of another language. Consider a sentence and its corresponding translation:

1. English: Every red insect is 0.75 inches long.
2. Hindi: प्रत्येक लाल पतंग पौन इं च लम्बा होता है ।

The corresponding Code-Mixed Sentences could be:

1. Every red insect 0.75 inches लम्बा होता है .
2. प्रत्येक लाल पतंग is पौन इं च long.

In following sections, we introduce LSTM-based RNNLM. Then we define and explain our baseline and baseline++ models. Then in section 4, we detail methods of training the RNN language models. In section 5 we define quantitative measures for modeling Code-Mixing across corpora. In section 6, we tabulate the performance measures of CMI, M-Index and perplexity scores of the models and compare them based on the values. We also list the challanges faced in terms of obtaining quality datasets for our models and issues faced while training RNN models, in the following section.

## 1.2 RNN Language Models

The language model is modeling the probability of generating natural language sentences or documents. You can use the language model to estimate how natural a sentence or a document is. Also, with the language model, you can generate new sentences or documents.
Let's start with modeling the probability of generating sentences. We represent a sentence as $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_T)$, in which $\mathbf{x}_t$ is a one-hot vector. Generally, $\mathbf{x}_0$ is the one-hot vector of BOS (beginning of sentence), and $\mathbf{x}_T$ is that of EOS (end of sentence).
A language model models the probability of a word occurrence under the condition of its previous words in a sentence. Let $\mathbf{X}_{[i,j]}$ be $(\mathbf{x}_i, \mathbf{x}_{i+1}, ..., \mathbf{x}_j)$, the occurrence probability of sentence $\mathbf{X}$ can be represented as follows:

$$P(\mathbf{X}) = P(\mathbf{x}_0) \prod_{t=1}^{T} P(\mathbf{x}_t | \mathbf{X}_{[0,t-1]})$$

So, the language model $P(\mathbf{X})$ can be decomposed into word probabilities conditioned with its previous words. We model $P(\mathbf{x}_t | \mathbf{X}_{[0,t-1]})$ with a recurrent neural network to obtain a language model $P(\mathbf{X})$.

# 2 RNN language model

Saved-Models for each curricula

## 2.1 Layers in Model

### 2.1.1 Word embedding layer

This is a deep neural network method for representing data with a huge number of classes more efficiently. Embeddings greatly improve the ability of networks to learn from data of this sort by representing the data with lower dimensional vectors.

1. Our corpus has vocabulary size 50K. Representing them categorically (one-hot en-code) is very inefficient, and the matrix created is very sparse
2. Embeddings are just a fully connected layer. We call this layer the embedding layer and the weights are embedding weights. This layer skips multiplication into the embedding layer by instead directly grabbing the hidden layer values from the weight matrix.
3. The embedding lookup table is just a weight matrix. The embedding layer is just a hidden layer. The lookup is just a shortcut for the matrix multiplication and is trained just like any weight matrix as well

### 2.1.2 LSTM layer(s)

This takes the embeddings as the input. An LSTM layer is defined by the number of LSTM cells that it contains. This number of cells corresponds to the time steps over which the layer will perform learning. This, number of cells, is an hyperparameter.

1. The LSTM overcomes the vanishing gradient problem of RNN. It does this by storing(latching) the input without forgetting for longer period of time.
2. This happens by use of four gates inside an LSTM cell. Every cell takes as input a long term memory(LTM), a short-term memory(STM) and outputs a new LTM, STM and prediction.

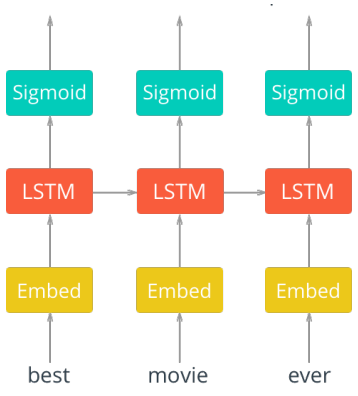


Figure 1: RNNLM Network diagram

#### 2.1.3 Fully connected layers

Finally, the output from LSTM layer is given to the fully connected layer which produces logits as outputs. These, when applied with softmax activation, produces probabilities of each word in the vocabulary.

#### 2.1.4 Loss function: Sequence loss

Since our input and outputs are a sequence of words, we use cross-entropy loss for a sequence of logits outputed by the final fully connected layer. The $sequence\_loss$ functionality in Tensorflow computes weighted cross-entropy loss. It takes logits of dimension $[seq\_length, vocab\_size]$ as input and targets of dimension $[seq\_length]$ which represents the true class at each time step.

#### 2.1.5 Optimizer: Adam optimizer

1. The Adam optimization algorithm is an extension to stochastic gradient descent.
2. Whereas Stochastic gradient descent maintains a single learning rate for all weight updates,in Adam optimization a learning rate is maintained for each network weight (parameter)and separately adapted as learning unfolds.
3. Adam combines the advantages of two other extensions of stochastic gradient descent, Ada-grad and RMSProp. It adapts the parameter learning rates based on the average first moment (the mean) as in RMSProp, and also makes use of the average of the second moments of the gradients.
4. Adam is a popular algorithm in the field of deep learning because it achieves good results fast.

# 3 Improvement: RNN language model with textual features

Saved-Models for each curricula

## 3.1 POS-tag features

Textual features such as words and Part-of-Speech tags might predict code-switching points. To model this in the structure of our network, we add POS tags to the input layer.
Our model consists of three classes: One class for all English words, one for all Hindi words, one for other languages. We extend the input layer by concatenating word vector $w(t)$ with vector $f(t)$ which provides features corresponding to the current word. These features are POS tags.

#### 3.1.1 CRF-model

Conditional random fields (CRFs) are a class of statistical modeling method often applied in pattern recognition and machine learning and used for structured prediction.
we use pre-trained CRF model to predict tags on our dataset. we also consider HMM model but as precision and f1 score of CRF model is better we are using CRF model for further predictions of POS tags on our dataset.

### 3.2 Model

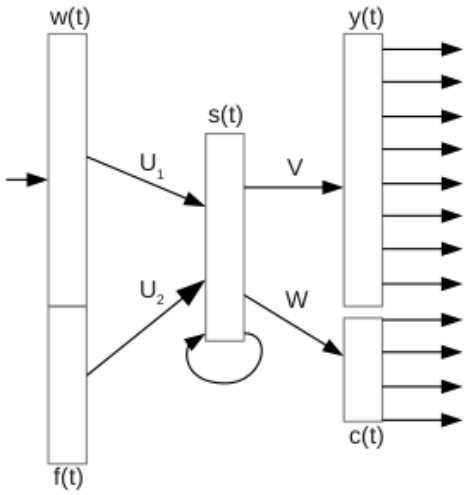


Figure 2: RNNLM with POS-tag features

The improved model is thus defined by the use of POS-tag features at the input layer to the baseline RNNLM model. The model follows the same layers as the baseline model. The difference is, during the training phase, not only the current word is activated but also its feature. Because the POS-tags are integrated into the input layer, they are also propagated into the hidden layer and back-propagated into its history. Thus, not only the previous feature is stored in the history but also all features several time steps in the past.

## 4 Training Curricula for code-mix

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.
For complex non-convex optimization problems, training with simpler examples first and introducing the complex examples at later stage has distinctive benefits.
We explore various curricula for training with monolingual and CS data. Let $T_1$ , $T_2$ and $T_{12}$ be respectively the set of training examples in $l_1$ , $l_2$ and intra-sentential code-mix between $l_1$ and $l_2$. We 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 curriculum 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 $T_j$.
Based on the ordering of the training instances, we define perform training with following different curricula,

1. $T_{12}$: Training with only the code-mix data.

2. $T_{12}; T_1$: Training with entire $T_{12}$ data first, followed by $T_1$
3. $\{T_1; T_{12}\}$: Training with instances from $T_1$ and $T_{12}$ simultaneously

# 5 Performance Measures

When comparing code-mixed corpora generated by the language model, it is desirable to have a measurement of the level of mixing between languages. We use following quantitative measures for modeling code-mixing across the generated corpora:

1. Perplexity: The **perplexity** $PP_p(T)$ of a model p is the reciprocal of the (geometric) average probability assigned by the model to each word in the test set T,

$$PP_p(T) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i/w_{i-n+1}^{i-1})}}$$

   - The higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model.
   - Perplexity can be also thought as the **weighted average branching factor of a language**. The branching factor of a language is the number of possible next words that can follow any word.
   - In neural LMs, the perplexity for the input sentence is calculated by taking product of probabilities of the words that the input sentence is actually composed of. Once this joint probability is computed, the perplexity is computed as above.

2. Code-mix index: At the utterance level, this amounts to finding the most frequent language in the utterance and then counting the frequency of the words belonging to all other languages present. If an utterance $x$ only contains language independent tokens, its code-mixing is zero; for other utterances, the level of mixing depends on the fraction of language dependent tokens that belong to the matrix language (the most frequent language in the utterance) and on $N$, the number of tokens in $x$ except the language independent ones (i.e., all tokens that belong to any language $L_i$):

$$C_u(x) = \frac{100}{U}[\sum_{i=1}^{U}(1 - \frac{native(x) - P(x)}{N(x)} + \delta(x))]$$

3. Multilingual Index (M-index): It is developed from the Gini coefficient, is a word-count-based measure that quantifies the inequality of the distribution of language tags in a corpus of at least two languages. The M-index is calculated as follows:

$$M - index = \frac{1 - \sum p_j^2}{(k - -1) \cdot \sum p_j^2}$$

   where k ($> 1$) is the total number of languages represented in the corpus, $p_j$ is the total number of words in the language $j$ over the total number of words in the corpus, and $j$ ranges over the languages present in the corpus.

# 6 Results

## 6.1 Dataset

### 6.1.1 For baseline model

To create the curricula defined in 4, we use English-Hindi code-mix data in which Hindi is written in Devanagari script.

### 6.1.2 For baseline++ model

Since training this model required POS-tags, we use code-mix corpora mentioned at [6]. We use CRF model explained in 3 for finding language ID along with POS-tags for each word. For English-only dataset, we use movie-review dataset obtained from [5]. We use the pre-trained CRF based model to find the POS-tags for this dataset.

## 6.2 Results

| Model | Curricula | Index values | |
|---|---|---|---|
| | | CMI | M-index |
| Baseline model | $T_{12}$ | 11.12 | 0.39 |
| | $T_{12}; T_1$ | 5.802 | 0.02 |
| | $\{T_1; T_{12}\}$ | 15.751 | 0.44 |
| Baseline++ model | $T_{12}$ | 68.49 | 0.64 |
| | $T_{12}; T_1$ | 14.841 | 0.81 |
| | $\{T_1; T_{12}\}$ | 90.41 | 0.58 |

Table 1: Performance of models

### 6.2.1 Observation on Index values

For code-mix index (CMI), higher value indicates more language switching in the utterance. We see that for both models, the curricula $T_{12}; T_1$ gives almost no code-mixing. This is because, the network gets trained on a mono-lingual language and thus both networks tend to learn the mono-lingual corpora.
The curricula $\{T_1; T_{12}\}$ performs well in case of both the models. It tells that training the model with batches of mix of code-mix and mono-lingual language makes the network learn weights better for code-mixing. In this, the baseline++ seems to outperform the baseline model due to the fact that it uses POS-tags as added feature which helps it to learn the code switching points much better.

| Model | Curricula | Perplexity Scores | |
|---|---|---|---|
| | | Train set | Test set |
| Baseline model | $T_{12}$ | 25485.67 | 33952.89 |
| | $T_{12}; T_1$ | 3391.85 | 5197.73 |
| | $\{T_1; T_{12}\}$ | 17837.49 | 21493.69 |
| Baseline++ model | $T_{12}$ | 13098.83 | 17565.31 |
| | $T_{12}; T_1$ | 4723.89 | 5238.77 |
| | $\{T_1; T_{12}\}$ | 9573.28 | 14614.79 |

Table 2: Perplexity Scores on train and test set

### 6.2.2 Observations on Perplexity values

Perplexity gives how well a language is able to predict the sequence in the utterance. For this purpose, we used 70:30 split of the data. The values show that models that are initially trained with monolingual data and then with code-mix data perform well in both the cases. The models that is trained with only monolingual data performs badly as compared to model that is trained with batches of mix of code-mix and monolingual data. In all curricula, the baseline model is able give better perplexity than the baseline model.

## 6.3 Conclusion

From the results we can say that the baseline++ model with POS-tag performed well in all the measures. The baseline++ model generated better code-mix data than its baseline counter part as can also be observed with CMI and M-Index. It gave higher CMI values indicating that the baseline++ is able to generate better code-mixed data. Also, M-Index with either side of 0 or 1 indicates higher percentage of a certain language which indicates poor code-mixing. We can observe that the M-Index values indicate balanced percentage of code-mixing in case of baseline++ model.
We also explored various training curricula for training a language generation model. The curricula $T_{12}; T_1$ seemed to work very well as shown by both CMI, M-Index and Perplexity values. The baseline model with this curricula gave the best perplexity values indicating that it is able to predict the sequences better than the baseline model. This can be attributed to less data availability of POS-tag data to train the baseline++ model.

# 7 Challenges

1. Dataset availability: With respect to the baseline++ model, we required Hindi-English code-mix dataset with POS-tags. Such a dataset was not readily available and also lacked in quality. To overcome this, we used pre-trained CRF model for to generate our own POS-tagged data.

2. Training with categorical sequences: When training an RNN model, we initially worked with categorical sequences where the output sequence was one-hot encoded. This caused the output vector to be very large and surpassed the machine capability to handle such huge sparse matrix and caused RAM overflow. We corrected this by first adding an Embedding layer before the input layer which reduced the input dimension to the RNN. Also, we used numerical sequences as outputs and sequence loss available in Tensorflow, to back-propagate the loss and reduce RAM usage.

3. POS-tag generation: To create POS-tagged dataset, we looked for pre-trained model which can compute POS tags on Hindi-English code-mix. The available implementations were on old libraries and required efforts to build those on deprecated libraries.

# References

[1] Ashutosh Baheti, Sunayana Sitaram, Monojit Choudhury, Kalika Bali *Curriculum Design for Code-switching: Experiments with Language Identification and Language Modeling with Deep Neural Networks*. Microsoft Research Lab, India.

[2] Heike Adel , Ngoc Thang Vu et al. *Recurrent Neural Network Language Modeling for Code Switching Conversational Speech*. In Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on · May 2013.

[3] Simran Khanuja, Sandipan Dandapat, Anirudh Srinivasan, Sunayana Sitaram, and Monojit Choudhury. *GLUECoS : An Evaluation Benchmark for Code-Switched NLP*. Microsoft Research, Bangalore, India.

[4] Anupam Jamatia, Björn Gambäck, Amitava Das. *Part-of-Speech Tagging forCode-Mixed English-Hindi Twitter and Facebook Chat Messages*. Proceedings of the International Conference Recent Advances in Natural Language Processing, September 2015

[5] Movie review data made available by Cornell University
`www.cs.cornell.edu/people/pabo/movie-review-data/`
`http://pylmnn.readthedocs.io/en/latest/`

[6] Scrapped twitter data.
`https://iiitaphyd-my.sharepoint.com/:f:/g/personal/prashant`$_k odali_r esearch_i iit_a c_i n/EsSRXSd$
$2oWxcB9t7NHA9oB5_u gYwp_H ftYA?e = 2iaTE1$