Introduction to NLP

Sriharshitha Bondugula Samartha S M

Prashant Kodali

CODE MIX GENERATION

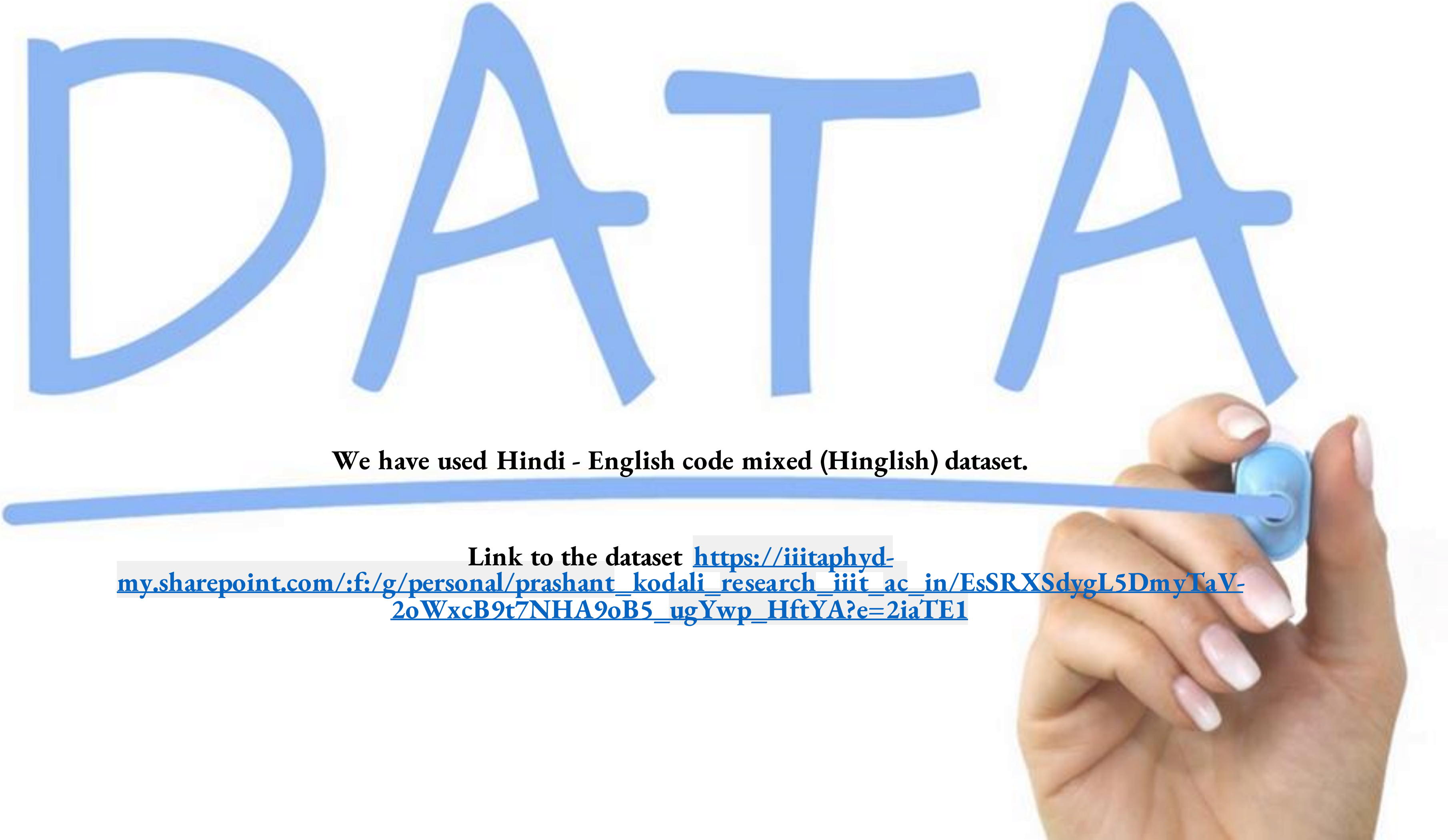
Developing a CodeMix Language Model

What is Code mixing?

Code Mixing is a phenomenon where a speaker mixes two or more languages in a single sentence, typically occurring in bilingual/multilingual societies. It can also be referred as intra-sentential code switching. Hinglish is an example of code mixing where Hindi and English languages are mixed. This is more frequently seen in user generated text on social media, comments on websites etc.

Problem Statement

The aim of the project is to generate code-mixed data. This is done by improvising the baseline LSTM LM to work efficiently on codemixed data and using it further to generate code mixed sentences. This project was aimed at developing different possible derivatives of existing baseline RNN and evaluate and analyze their performances.



We have used Hindi - English code mixed (Hinglish) dataset.

Link to the dataset https://iiitaphyd-my.sharepoint.com/:f:/g/personal/prashant_kodali_research_iiit_ac_in/EsSRXSdygL5DmyTaV-2oWxcB9t7NHA9oB5_ugYwp_HftYA?e=2iaTE1

Approach

- **Reference Paper: Language Informed Modeling of Code-Switched Text**

- **Summary: It is known that training language models for Code-Mixed (CM) language is very difficult due to lack of enough data because Code-Mixed (CM) language is very rare and informal which makes it hard to find enough reliable data for training language models. The task of language modeling is very important to several downstream applications in NLP including speech recognition, machine translation, etc. This is particularly important in domains that lack annotated data, such as code-switching, where the need to leverage unsupervised techniques is a must. This paper discusses different Language Models that can be derived from multilayer LSTM architecture which can be used on CM data. The paper also shows that encoding language information into the model helps language models perform better by learning the switching points in the Code-Mixed (CM) sentences. The perplexity of this new language model is also improved over baseline language model.**

IMPLEMENTATION

```python
ss = lines.isspace()
if (not ss):
  lines = lines.replace("<unk>","")
  # Remove punctuations
  new_line = re.sub("[^a-zA-Z0-9 ]", "", lines).lower().strip()
```

PREPROCESS THE TEXT CORPUS

The first part of the language modeling is to clean the text. As a part of cleaning the text, the following operations are done on the sentences using regex.

Noise: <unk> tags

Empty lines were removed.

We used regular expressions (regex) to get rid of special characters and numbers.

Extra whitespaces and tabs.

Each line is then converted into lower case.

```
for i in range(0,len(new_line.split())):
    templ = new_line.split()[i]
    tokens.append(templ)
    if (i < len(new_line.split())-4):
      temp4 = [new_line.split()[i+j] for j in range(0,5)]
      five_grams.append(temp4)
```

TOKENISATION AND OBTAINING n-grams

The preprocessed line obtained is then split and all the tokens are collected. The LSTM baseline model that we are going to build is a 5-gram model. So, we have collected all the 5-grams.

Int2token and Token2int dictionaries

We cannot feed sentences to neural networks in the form of text. To solve this problem each word/token is assigned an id (integer) and it is stored in a dictionary called token2int. The network gives an integer as an output which will be the id of some token. This should hence be converted into the token for which we need an int2token dictionary. The following are the 2 dictionaries:

```
{'kajrival': 0, 'paltu': 1, 'bmw': 2, 'huyi': 3, 'opportunities': 4, 's
{0: 'kajrival', 1: 'paltu', 2: 'bmw', 3: 'huyi', 4: 'opportunities', 5:
```

Considering most_common words as vocabulary and others as 'unk'

We have calculated the number of tokens with a frequency of 1 by using Collections.counter() and have considered all these words as 'unk' tokens. This is added as an improvisation to the baseline model.

```
common_words = word_counter.most_common(vocab_size-ind)
```

Obtaining input and target sequences and feeding them to the neural network

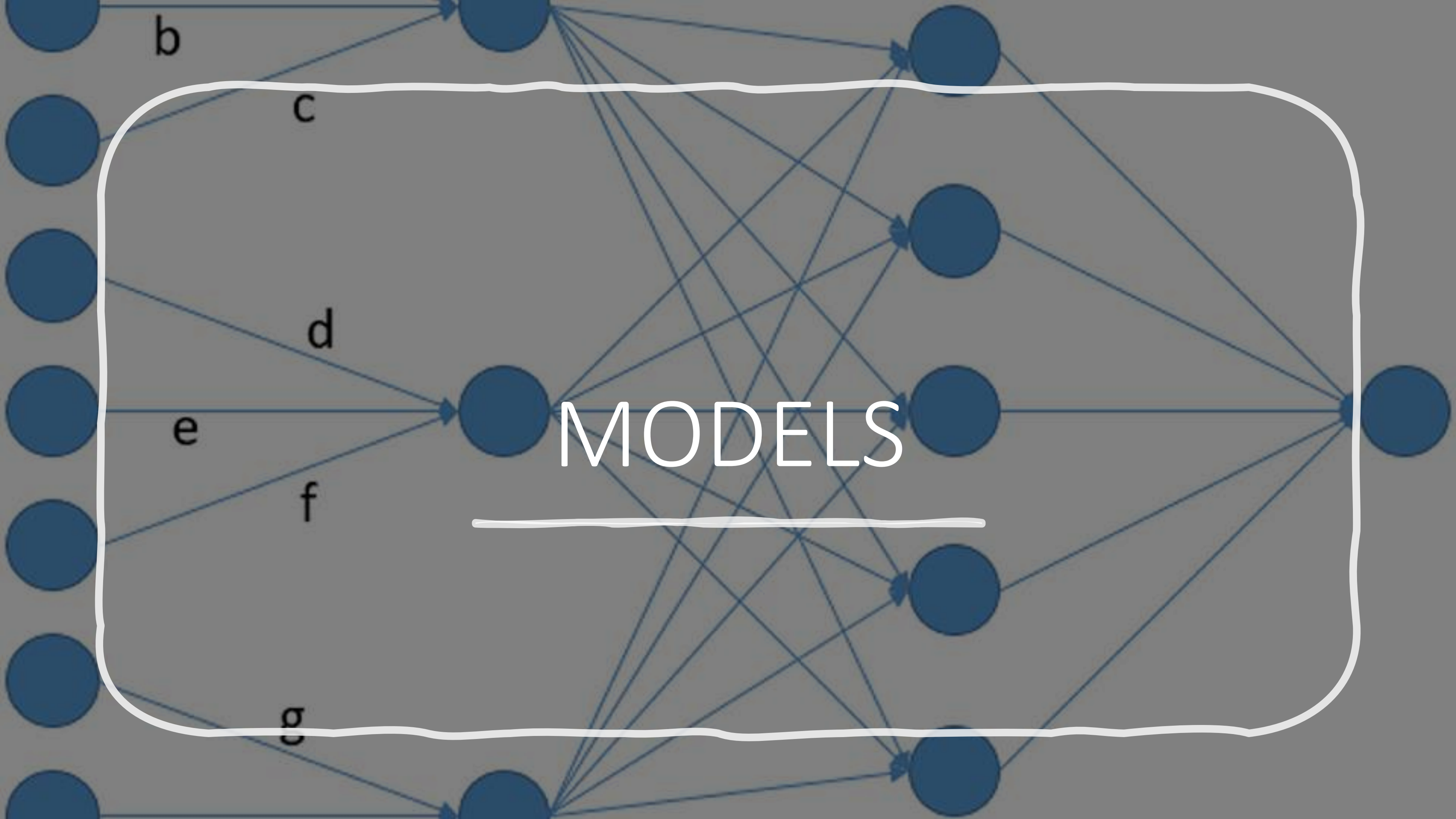
Input sequences and target sequences are obtained for the five grams. These are then converted into integer sequences using the token2int[w] for each word ‘w’ in the sequence. The words that are common are passed as it is, whereas the words say w, which are not common, are passed as ‘w’ as well as ‘unk’.

```
for s in five_grams:
    x.append(s[:-1])
    y.append(s[1:])

print(x[:10])
print(y[:10])
```

```
[['ye', 'to', 'hona', 'hi'], ['to', 'hona', 'hi', 'tha'], ['
[['to', 'hona', 'hi', 'tha'], ['hona', 'hi', 'tha', 'kabhi']
```

MODELS



Baseline Model

For the baseline, we have trained a basic LSTM RNN model on the given code mixed dataset. Below is the baseline model architecture.

```
WordLSTM(
  (emb_layer): Embedding(26744, 200)
  (lstm): LSTM(200, 256, num_layers=4, batch_first=True, dropout=0.3)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=26744, bias=True)
)
```

Improved Model 1 (with UNK tokens)

We further experimented with the baseline LSTM RNN model by introducing UNK tokens, which gave better results than the baseline model. The model architecture is the same as that of the baseline model.

Improved Model 2 (with language IDs)

As the code mixed dataset contains sentences with words in two different languages, using the language information of the tokens become important to get better results. Hence, in this improved model, we use language IDs to train the model on the given dataset. We train the language information also parallely and integrate it with the regular LSTM model in loss function to improve the learning process. Below is the model architecture.

```
WordLSTM(
  (emb_layer): Embedding(21194, 200)
  (lstm1): LSTM(200, 256, num_layers=4, batch_first=True, dropout=0.3)
  (lstm2): LSTM(200, 256, num_layers=4, batch_first=True, dropout=0.3)
  (dropout1): Dropout(p=0.3, inplace=False)
  (dropout2): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=21194, bias=True)
)
```

Improved Model 3 (with language information and UNK tokens)

In this model, we experimented by combining the improved model 1 (using unk tokens) and 2 (using language information) to analyse the performance. The model architecture for this improved mode, is same as that of the improved model 2 with parallel training for language information with the regular training on dataset.

Training

We have trained the models with the input and target sequences with language information sequences (for the models 3 and 4) constructed, for 15/20 epochs.

Parameters used are,

Number of epochs: 15/20

Batch size: 32

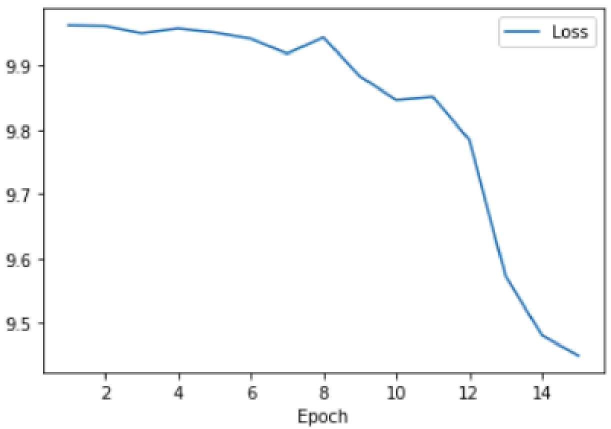
Learning rate: 0.001

Loss function: CrossEntropyLoss()

Optimiser: Adam

TRAIN RESULTS

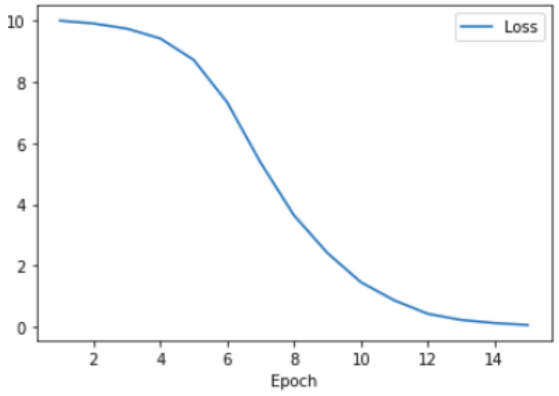
Training (Baseline model)



	Epoch	Loss
0	1	9.961883
1	2	9.960908
2	3	9.949471
3	4	9.957108
4	5	9.950838
5	6	9.941049
6	7	9.917805
7	8	9.943020
8	9	9.882340
9	10	9.846100
10	11	9.851092
11	12	9.784495
12	13	9.573050
13	14	9.480783
14	15	9.449256

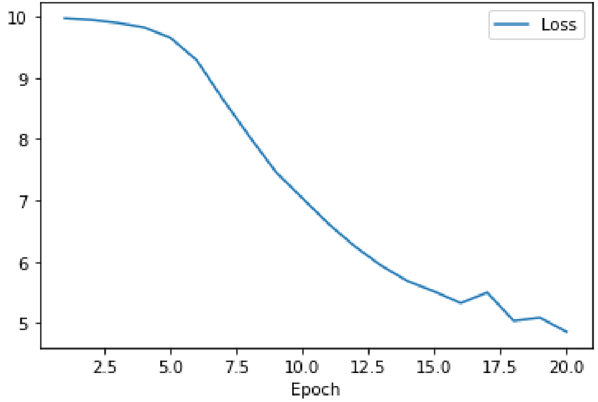
Training (Improved model 1)

	Epoch	Loss
0	1	9.985563
1	2	9.893213
2	3	9.724904
3	4	9.405950
4	5	8.706365
5	6	7.329415
6	7	5.363165
7	8	3.646013
8	9	2.419678
9	10	1.468599
10	11	0.877111
11	12	0.439783
12	13	0.237585
13	14	0.135531
14	15	0.072093



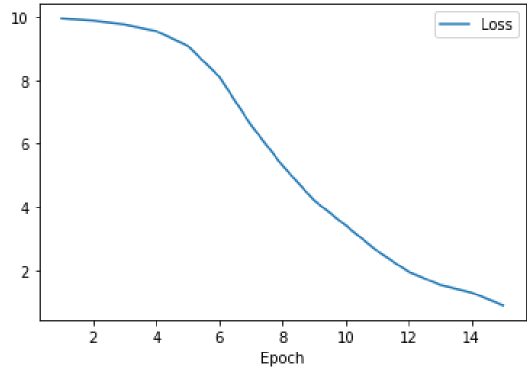
Training (Improved model 2)

	Epoch	Loss
0	1	9.970361
1	2	9.945708
2	3	9.895752
3	4	9.822148
4	5	9.652771
5	6	9.284848
6	7	8.642557
7	8	8.039134
8	9	7.460744
9	10	7.035605
10	11	6.612917
11	12	6.243889
12	13	5.931624
13	14	5.679304
14	15	5.512897
15	16	5.324346
16	17	5.497158
17	18	5.035444
18	19	5.085797
19	20	4.856457



Training (Improved Model 3)

	Epoch	Loss
0	1	9.941549
1	2	9.873579
2	3	9.754515
3	4	9.546169
4	5	9.086511
5	6	8.110243
6	7	6.590604
7	8	5.305894
8	9	4.211862
9	10	3.422567
10	11	2.613928
11	12	1.952968
12	13	1.539642
13	14	1.295286
14	15	0.896017



Predicting probabilities of sentences

We have used the chain rule to predict the probability of a sequence. These predicted probabilities are further used to predict the perplexities of sentences. To handle the values that are extremely high and extremely low, we used a normalised formula.

$$\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}
\end{aligned}$$

$$\begin{aligned}
e^{\frac{\ln(P(W))}{N}} &= e^{\frac{\sum_{i=1}^{N} \ln P(w_i)}{N}} \\
(e^{\ln(P(W))})^{\frac{1}{N}} &= (e^{\sum_{i=1}^{N} \ln P(w_i)})^{\frac{1}{N}} \\
P(W)^{\frac{1}{N}} &= \left(\prod_{i=1}^{N} P(w_i)\right)^{\frac{1}{N}}
\end{aligned}$$

CMI

- It is the measure of the degree of code-mixing in a corpus.
- CMI values range from 0 to 100. A value close to 0 suggests multilingualism in the corpus, whereas high CMI values indicate a high degree of code-mixing. To calculate the value of CMI, we generated 100 sentences of length 15 for every seed (10 in total) for each model and annotated them at the token level with the language tags.

$$CMI = \begin{cases} 100 * [1 - \frac{max(w_i)}{n-u}] & n > u \\ 0 & n = u \end{cases}$$

TEST RESULTS

Results (Loss and perplexities)

- Trained all the language models on 10K sentences (trained on the same dataset)
- Tested them on 10000 train sentences and test sentences. And the average is printed below.

	TRAIN PERPLEXITY	TEST PERPLEXITY	LEAST TRAIN LOSS	CMI
Model 1	2.440906438163917e+25	2.50884831429142e+25	9.421479	37.11875
Model 2	1.511138787017082	1.513985439644699	0.147454	40.5
Model 3	3.08264925867907e+24	3.08882214009567e+24	4.856457(for 20 epochs)	40.18125
Model 4	1.5112563914224086	1.52046411998412	0.896017	42.3374999999

Results (Generated Sentences)

Baseline Model

```
teacher ko bhi to koi bhi
teacher ko to koi hi baat
teacher ko bhi to fir koi
teacher ka naam nhe hai to
```

```
comments ki image ko bhi kam
comments me hai jo to ye
comments ki tarah kharab karne ke
comments ki baat kar diya to
```

```
modiji ki tarah bhi nhi hota to kya baat h jo
modiji ki kami se hi malum hai or aap party ki
```

```
india ki baat nahi to ye
india ka baat hai aur ye
india ki bhi bhi bhi hi
india ka bhi news me hai
```

```
life ki tarah nahi hoga ye
life ko koi bada jarurat h
life ko bhi bhi bhi bhi
life ki bhi bhi kam nhi
```

Results (Generated Sentences)

Improved Model 1

```
doctor ki baat hai aur to
doctor ka naam hai aur ye
doctor ki jarurat h ki koi
```

```
teacher ki tarah hai to ye
teacher ki jarurat h ki kya
```

```
india ka naam nhi h ye to koi problem
india me bhi to koi problem nhi hai aur
india ka name hai jo bhi bhi to ye
india ka name hai jo to ye bhi nahi
```

```
bjp ko koi bhi news hai
bjp me hi hai to kya
bjp ka naam nhi h ye
```

Improved Model 2

Results (Generated Sentences)

```
India crore crore news me hai
India rs me bik rahe ho
India crore ko kam nhi h
India id me hai ye sab
```

```
doctor ko bhi hai to to
doctor ka sath liya to ye
doctor ko bhi hai ki ye
doctor ki tarah khabar de raha
```

```
respect to to fir bhi to
respect ki baat nahi h to
respect to ye bhi nahi hai
respect to bhi bhi hai jo
```

```
teacher ki baat nahi hota to
teacher ko to to koi problem
teacher ko to koi news nahi
teacher ki baat kar rahe h
```

Improved Model 3

```
india me to koi kam ho
india ke sath hai ye bhi
india ke baad kuch nhi hai
india ke liye to aaj kal
```

```
doctor ki baat nhi hai ye
doctor ki jarurat ho to tum
doctor ko bhi koi kam nhi
doctor ka kya halat ho raha
```

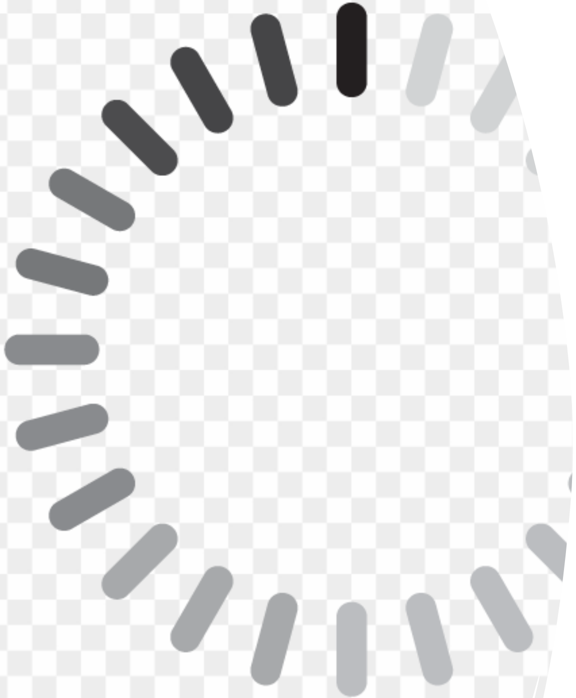
```
life me koi badi bat hai to
life me to fir koi problem hai
life me koi nahi hai ki ye
life me bhi koi nahi hai ye
```

```
teacher ki baat nahi ho raha
teacher ko hi malum hai to
teacher ka bhi koi kam nhi
teacher ka naam nhe to ye
```

```
bjp ko hi koi badi baat
bjp ko hi malum hai ki
bjp ko bhi hi milna chahiye
bjp ki baat nahi ho raha
```

CONCLUSION

From train and test losses it was clear that 2,3,4 models were better than the first one. Though it was not evident from the generated sentences, the CMI index made it clear that model 4 did a better job in generating codemix sentences. Hence, the improvisation done by considering common words and by giving token-level language details helped the model perform better.



FUTURE WORK

- Experiment with other evaluation metrics.
- Experiment with different model architecture
- Experiment with the cross lingual word embeddings

Timeline

March 15

Complete understanding of the paper. Collecting all the required data and preprocessing (Step1)

March 25

Build the baseline model

April 15

Improvise the baseline model for better performance

April 20

Testing and evaluating the models and analysing the results

May 1

Final Presentation


THANK YOU

Git Repo Link

https://github.com/samarthamahesh/NLP-Project---Code-Mix-Generation