INTRODUCTION TO NLP

Project - Code Mix Generation

SRIHARSHITHA BONDUGULA (2018111013)
SAMARTHA S M (2018101094)

Project 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.

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 to 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. We have worked on the Hindi - English code mixed (Hinglish) dataset for this project.

Neural Language model (LSTM based)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNNs) capable of learning order dependence in sequence prediction problems.
The link referred: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Preprocessing

The link referred:
https://towardsdatascience.com/nlp-building-text-cleanup-and-preprocessing-pipeline-eba4095245a0

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.

1) The dataset given contained <unk> tags which were removed as the first step.
2) Empty lines were removed.
3) Special characters, as you know, are non-alphanumeric characters. These characters are most often found in comments, references, currency numbers, etc. These characters add no value to text understanding and induce noise into algorithms. We used regular expressions (regex) to get rid of these characters and numbers.
4) Extra whitespaces and tabs do not add any information to text processing. These were removed as well.
5) Each line is then converted into lower case.

The code snippet used for the same is given below,

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

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, I have collected all the 5-grams. Following is the code snippet used for the same.

```
for i in range(0,len(new_line.split())):
    temp1 = new_line.split()[i]
    tokens.append(temp1)
    if (i < len(new_line.split())-4):
      temp4 = [new_line.split()[i+j] for j in range(0,5)]
      five_grams.append(temp4)
```

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(). We have considered all these words as 'unk' tokens. This is added as an improvisation to the baseline model. We have not done this 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 function get_int_seq which returns the value of token2int[w] for each word 'w' in the sequence passed. 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']
```

Model 1 (Baseline)

The following is the baseline model that we have built and used; It is a 5-gram LSTM model. We haven't considered common words in this model.

```
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)
)
```

Model 2 (with unk tokens)

This is an improvisation of the previous baseline model. In this model, we have considered the most common words as vocabulary and the other words as unknown.

Model 3 (with language information)

The following model architecture is for using the language information also with the data.

```
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)
)
```

Model 4 (with language information and unk tokens)

The following model architecture is for using language information also with the data along with the improvisation that is done in model 2.

```
WordLSTM(
  (emb_layer): Embedding(21196, 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=21196, bias=True)
)
```

Training

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

Parameters used are;

Number of epochs: 15/20

Batch size: 32

Learning rate: 0.001

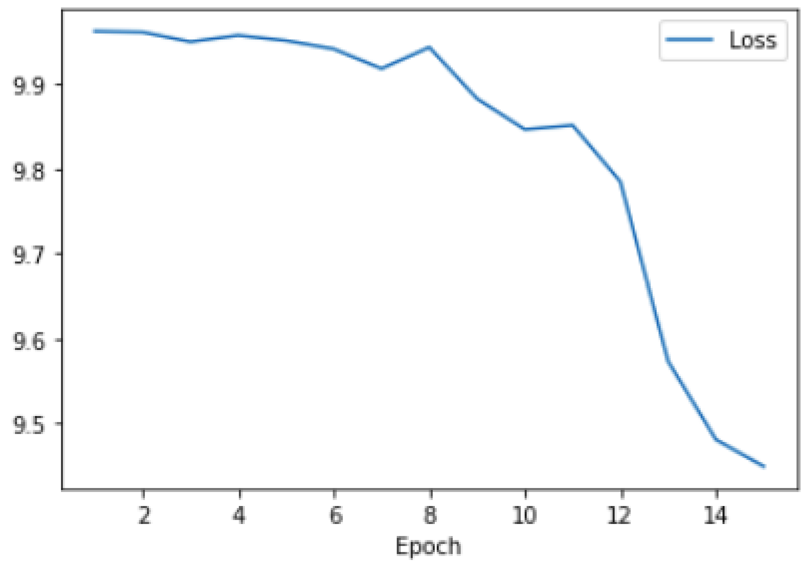
Loss function: CrossEntropyLoss()

Optimiser: Adam

The losses are calculated per each epoch and the graphs for the same for each of the 5 models described before are shown below

Model 1

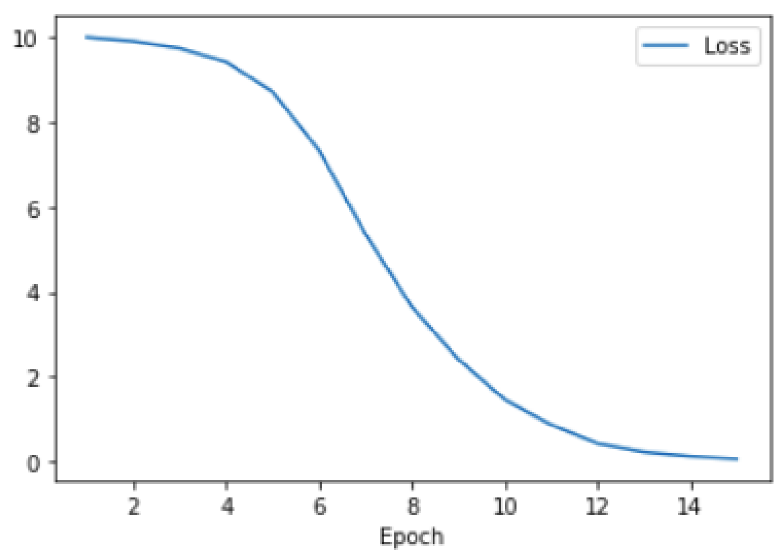
This model was converging very slowly. This is the baseline model.



	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

Model 2

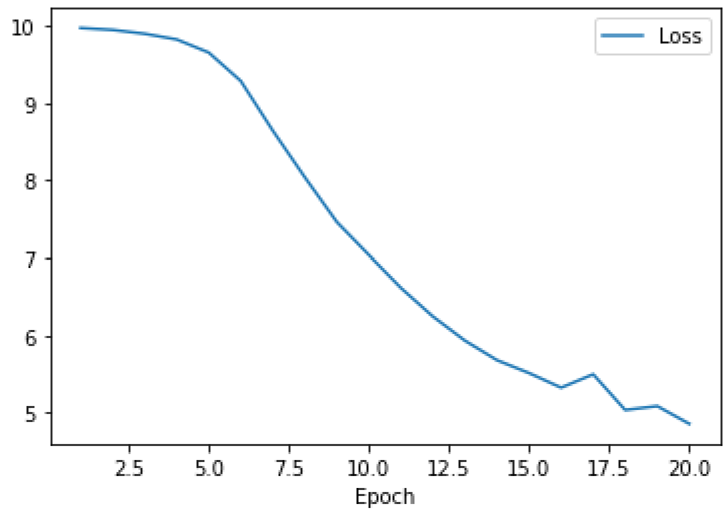
This model converged considerably well with the introduction of the 'unk' tokens. And the graph clearly shows that the model is learning from the data.



	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

Model 3

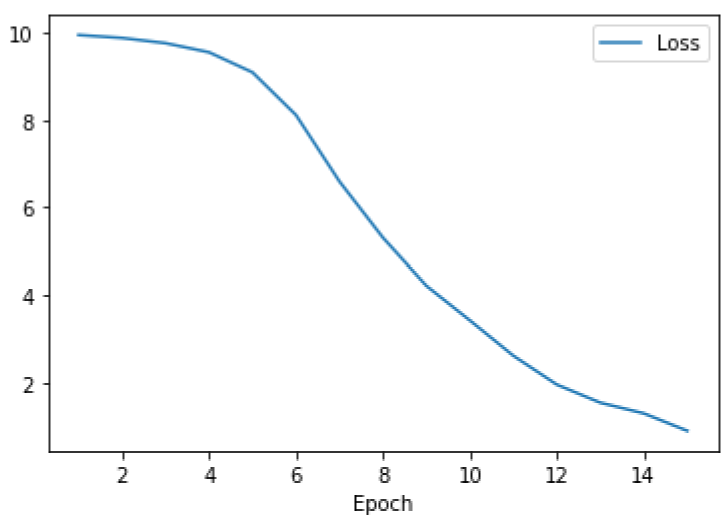
This model converges pretty well. But it took more time to converge than the above model. It worked better than model1 with the introduction of language ids. This model is run for 20 epochs.



	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

Model 4

This model converges and it is quick as well compared to the previous model. The introduction of language ids and unk tokens makes it a better model.



	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

Loss per epoch is decreasing as we can see from the above graphs. However, we see that not all of the graphs are similar in terms of rate of decrease and loss values.

We can clearly say from the loss values and graph that model 4 is better than model 3 because it is learning the code mixed dataset faster and better than model 3.

Train-Test division

We have considered the first 30K sentences as the train set and the following 10K sentences as the test set.
We have not used any validation set in our model. We have used the following code snippet to generate these train and test sets.

```
for x in file:
    ss = x.isspace()
    if(not ss and counter_train<30000):
        f_train.write(x)
        counter_train+=1
    elif(not ss and counter_test<10000):
        f_test.write(x)
        counter_test+=1
```

Predicting probabilities of sentences

The link referred: https://towardsdatascience.com/perplexity-in-language-models-87a196019a94

We have used the chain rule to predict the probability of a sequence. The softmax layer of the model and the concept of hidden state in LSTMs play a major role in this.

These predicted probabilities are further used to predict the perplexities of sentences. The formula used to calculate perplexity is as shown in the figure. w1w2...wN is the sequences. (N=5 in my LSTM model). P(w1w2...wN) is expanded using the chain rule.

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

But at times, the product was going to inf and at times it was going to 0. So, as a part of normalization, we did the following,
Get_prob function returns the sum of the log of probabilities calculated using softmax. This is then passed to get the perplexity of a sentence. The code snippet for the same is as follows,

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

```
def get_perp(prob,n):
  p = math.exp(prob*(1/n))
  return 1/p
```

CodeMix Index (CMI)

The link referred:
https://www.researchgate.net/publication/340806414_PHINC_A_Parallel_Hinglish_Social_Media_Code-Mixed_Corpus_for_Machine_Translation

It is the measure of the degree of code-mixing in a corpus.

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

- w_i is the number of words of the language
- max{wi} represents the number of words of the most prominent language
- n is the total number of tokens,
- u represents the number of language-independent tokens (such as named entities, abbreviations, mentions, hashtags, etc.)

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.

Results:

- Trained all the language models on 30K sentences (trained on the same dataset)
- Tested them on 10,000 train sentences and test sentences. And the average is printed below.

	Train Perplexity	Test Perplexity	Least train loss	CMI (1000 sentences)
Model 1	2.440906438163917e+25	2.50884831429142e+25	9.449256	37.11875
Model 2	1.511138787017082	1.513985439644699	0.072029	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	42.33749999999999

We predicted the most probable 3 words and picked a random one from the 3.

Generated sentences:

Model 1:

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

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

```
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
```

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

Model 2:

```
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
```

Model 3:

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

```
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
```

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

Model 4:

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

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

```
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
```

```
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.

Git Repo Link

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

Colab Link

https://colab.research.google.com/drive/12nLPZayc-WquCJbxFaU7vh9wkxbZl7X7?usp=sharing
https://colab.research.google.com/drive/1h7t1Fi8KN3rQ00CLG9cWCZGwX_-RBLkq?usp=sharing