സ്വാഭാവിക மொழி प्रक्रियाकरण इंडिक ಭಾಷೆಗಳು
Natural Language Processing for Indic Languages

Team members:

Tirumala Kaggundi
Dhruv Sinha

Link to Github code: https://github.com/Tiru-Kaggundi/Adv_ML_project

1. Language identification: https://github.com/Tiru-Kaggundi/Adv_ML_project/blob/main/LM_detection.ipynb
2. Text classification: https://github.com/Tiru-Kaggundi/Adv_ML_project/blob/main/Classification.ipynb
3. POS tagging: https://github.com/Tiru-Kaggundi/Adv_ML_project/blob/main/PoS_Tagger_final.ipynb
4. RNN based sentence generation: https://github.com/Tiru-Kaggundi/Adv_ML_project/blob/main/RNN_based_sentence_generation.ipynb

Abstract

This paper aims to replicate some common NLP tasks on 5 Indic languages- Hindi, Bengali, Tamil, Malayalam, and Kannada. We will be performing language detection, text classification, part of speech tagging, sentence generation on these languages. We have trained several datasets using BoW, LSTMs, and GRUs to perform the mentioned tasks. Our model was able to perform exceptionally well in language detection. The model performed well on classification of Kannada and Malayalam texts but failed for Hindi, plausibly due to data constraints. For Part of Speech tagging in Hindi, the model was performing well but suffers from severe overfitting. Finally, for sentence generation, we saw that the model was able to learn some very interesting nuances of Hindi which took us by surprise!

Introduction

Over the last few decades, there has been a lot of research done in the field Natural Language Processing and the methods and technology has significantly advanced in the last ten years. There are various packages and modules that are able to identify languages, tag each word with their corresponding parts of speech, summarize a given paragraph, and also generate content based on a given prompt. While a lot of work has been done in ‘English’, we realized that application of NLP functionalities in Indian languages is very limited. Through this project, we will leverage NLP to perform 3 main tasks, but on texts from different Indian languages.

There are copious texts generated daily in different parts of India in the form of newspapers, magazines, movies, or social media content. Moreover, the coverage of this content is significantly large as India is the second largest country in terms of population in the world. In India, almost everyone is consuming content in Indian languages in some way or the other. Therefore, it would be interesting to test some basic functionalities of NLP on these vernacular languages. This would allow us to understand if we need to modify the way we use traditional NLP algorithms when we work on Indian languages. However, this comes with a significant challenge as the content is not available in an aggregated format that can be used for training. In this project, we have collated and cleaned several datasets for different tasks. Each task required modification, cleaning, and preparation of a dataset for training and testing. This proved to be a major challenge as we had to spend a lot of time, first collating the datasets and then cleaning it so that it is compatible to be used on pytorch. We will discuss this in the data section of this paper.

The other implication of this project is that we have experimented with Pytorch and pandas functionalities on Indic languages. This experiment can serve as a structural and base for further research in NLP on Indic Languages. Researchers can avoid making the same mistakes we did (of course we were not aware) and can save a lot of time that goes into cleaning the dataset and making it compatible with PyTorch functionalities.

Scope of this project: Parts to implement

In this paper we have implemented 4 main NLP tasks on 5 indic languages.

1. **Language detection** task is implemented on Hindi, Kannada (a language spoken in the State of Karnataka), Malayalam (spoken in Kerala), Tamil (spoken in Tamil Nadu), and Bengali (spoken in West Bengal).

   Result: Our model was reliably able to differentiate and classify each of the 5 languages

2. **Text Classification** task is done on Hindi, Kannada, and Bengali. This task involves classifying sentence into its major theme- business, international, sports, entertainment, etc.

Result: Text Classification worked best on Kannada and performed relatively well for Malayalam and Hindi

3. **PoS tagger:** Part of speech tagging task is done only on Hindi. We didn't find reliable data sources to perform part of speech tagging on other languages.

   Result: PoS Tagger performed decently well on the limited sized data we had

4. **Sentence Generation:** We train a LSTM and GRU model on corpus of sentences in Hindi. We use these models to generate sentences based on random words.

   Result : From our knowledge of Hindi, we can conclude that the model trained generates sentences that make partial grammatical or intuitive sense.

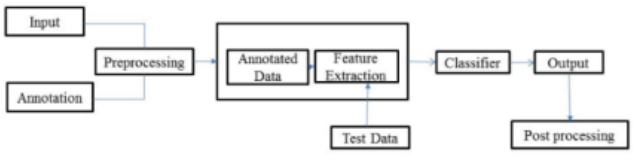
Literature Research

Most of the data cleaning and manipulation process is dataset and task specific. The models that we have used are mostly inspired from the Homework Problems we solved in class. For this project, we only referred to a few research papers. We will discuss the insights we derived from these papers in this section-

Kumar et al (2015) differentiate between English and a few Indic languages using Tokenizers. They look at each sentence, and use tokenizer to build Vocab. However, the script used for other languages in their dataset is Latin. They are not actually differentiating between Indic languages written in their original script. However, the methodology that they follow is very similar to ours-

Input Query	Output
And ibruna meet maadid kushinu aythu !!!	And\en ibruna\kn meet\en madid\kn kushinu\kn aythu\kn !\X
Dhoni risk edutha gumbala risk edukanam !	Dhoni\NE Risk\en edutha\ta gumbala\ta risk\en edukanam\ta !\x

Input and Output in Kumar(2015)

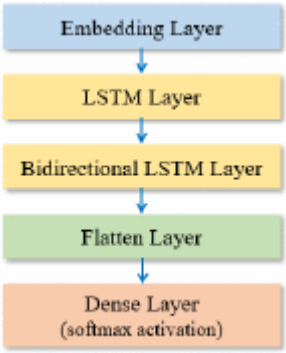


Proposed Design Flow of Language Detection in Kumar(2015)

In the output above, The algorithm is able to assign each word its respective language. The process that Kumar(2015) follow is very similar to ours- we also pre-process our data and use BoW for feature extraction and language detection.

For sentence generation task, we referred to Das et al (2020). They implemented a sentence generation system based on Long Short-Term Memory (LSTM) architecture. Their system generally followed the basics of word embedding where words from the dataset get tokenized

and turned into vector forms. These vectors were then processed and passed through a Long Short-Term Memory layer. Successive words get generated from the system after each iteration. This process when repeated leads to formation of a sentence or a passage of the specified length. The authors propose the following model-



The embedding layer maps a set of words to a vector form for improving the ability of neural networks. The LSTM layer, their model has an input gate, a forget gate, and an output gate. The output from LSTM layer is passed onto a Bidirectional LSTM layer which passes the output to get flattened and converted into probabilities via softmax activation.

For Part of Speech Tagging in Hindi, we refer to Srivastava et al. They use a simple Hidden Markov Models (HMM) for PoS Tagging. The core idea of this approach is to “explode” the input in order to increase the length of the input and to reduce the number of unique types encountered during learning. This in turn increases the probability score of the correct choice while simultaneously decreasing the ambiguity of the choices at each stage. This also decreases data sparsity brought on by new morphological forms for known base words.

Data used

For the tasks, we adapted different datasets as indicated below:

a) **For language detection task:** We prepared the data from the news articles with labels downloaded from the link:
https://storage.googleapis.com/ai4bharat-public-indic-nlp-corpora/evaluations/classification/indicnlp-news-articles.tgz
The dataset contains news articles with labels of type of news in various Indian languages. We identified the languages and replaced the labels with the label of languages. For example, hindi language articles were replaced with label ‘hi’.
In the end, we had around 52827 rows for 5 languages as shown below:

Language	# Train sentences
Hindi	3467
Kannada	24000
Malayalam	4800
Bangla	11200
Tamil	9360

	lang	text
0	hi	मेट्रो की इस लाइन के चलने से दक्षिणी दिल्ली से...
1	hi	नेटिजन यानि इंटरनेट पर सक्रिय नागरिक अब ट्विटर...
2	hi	इसमें एक फ़्लाइट एटेनडेंट की मदद की गुहार है औ...
3	hi	प्रतीक खुलेपन का, आज़ाद ख्याली का और भीड़ से अ...
4	hi	ख़ासकर पिछले 10 साल तक प्रधानमंत्री रहे मनमोहन...
...	...	...
52822	ta	எம் . ஜி . ஆர் காலத்திலிருந்து விமல் காலம் வரை...
52823	ta	கப்பல்கள் மோதிய விவகாரத்தில் கடந்த 8 நாட்களாக ...
52824	ta	பிக்பாஸ் நிகழ்ச்சியில் தற்போது தலைவியாக இருக்க...
52825	ta	பரதன் இயக்கத்தில் விஜய் நடித்துள்ள அவருடைய 60வ...
52826	ta	இந்திய அணியின் கேப்டன் விராட் கோலி , ஒவ்வொரு ப...

52827 rows × 2 columns

b) **For news item classification task:** We used the same dataset as above. However, we divided the datasets into different languages and used the news item classification labels for training.

c) **For POS tagging task:** We used the dataset available from IIT Bombay's website, where they had attempted a POS tagger task earlier on Hindi language. The dataset was cleaned and read into pandas for further work. The IIT Bombay team had also developed a POS tagger. However, the interface to the tagger was complex and we developed our own POS tagger. The link to the dataset is: http://www.cfilt.iitb.ac.in/tools/POS_tagger.zip

d) **For sentence generation task:** We downloaded the wikipedia articles dataset from kaggle (https://www.kaggle.com/datasets/disisbig/hindi-wikipedia-articles-55k) in Hindi.

The dataset contained 55000 articles in hindi as txt files. Out of these, we clubbed longer length articles and adjusted the lengths to get the following datasets:

lm-dev.txt	14.4 Mb
lm-test.txt	13.8 Mb
lm-train.txt	77.1 Mb
lm-dev-small.txt	883 Kb
lm-test-small.txt	878 Kb
lm-train-small.txt	4.2 Mb

Challenges Faced

1. **Dataset availability**:

Availability of datasets was a challenge. The Indian languages has attracted research activities from big tech firms such as google and amazon. However, the availability of labeled and tagged data to small researchers is limited. The Indian language news classification dataset shown above was created by joint collaboration of Google and IIT Madras. Similarly, IIT Bombay had done the work on PoS tagger. However, the amount of effort required dissuades small time or independent researchers to carry out dataset creation task. The closest we get in the open access category are the wikipedia articles. These articles require clubbing and creation of customized sized datasets for our purposes. Once clubbed, the data needs to be brought to the form where it can be fed into a dataloader in pytorch. This is described below.

2. **Translation of data to pytorch iterator format of language training -**

Pytorch requires that the dataloader is fed with a dataiterpipe format. Therefore, we need to convert the text data into an iterpipe before the dataloader can accept the same.

This is done in following steps:

Step 1: Read in the data in pandas.

Step 2: Create a Dataset class by leveraging troch.utils.data.Dataset class as shown below:

```
class CustomTextDataset(Dataset):
    def __init__(self, df):
        self.labels = df.iloc[:,0]
        self.text = df.iloc[:,1]

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        if idx >= self.__len__():
          raise StopIteration
        label = self.labels[idx]
        text = self.text[idx]
        return label, text
```

In the above instance, the pandas dataframe has two columns - labels and text. The text column contains sentences and the labels column contains the labels for the text.
torch.utils.data.Dataset is an abstract class representing a dataset. The custom dataset should inherit Dataset and override the following methods:

a) __len__ so that len(dataset) returns the size of the dataset.
b) __getitem__ to support the indexing such that dataset[i] can be used to get ith sample.

Refer: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

Step 3: Write a relevant collate_batch function:

```
def collate_batch(batch):
    label_list, text_list, offsets = [], [], [0]
    for (_label, _text) in batch:
      label_list.append(label_pipeline(_label))
      processed_text = torch.tensor(text_pipeline(_text), dtype=torch.int64)
      text_list.append(processed_text)
      offsets.append(processed_text.size(0))
    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)
    return label_list.to(device), text_list.to(device), offsets.to(device)

train_iter = CustomTextDataset(train_lang)
dataloader = DataLoader(train_iter, batch_size=32, shuffle=False, collate_fn=collate_batch)
```

The above collate batch function is customized for the classification requirement. The text and label pipeline code used above are simple mapper functions as shown below:

```
language_dict = {'hi':0, 'kn':1, 'ta':2, 'ml':3, 'bn':4 }
text_pipeline = lambda x: vocab(tokenizer(x))
label_pipeline = lambda x: int(language_dict[x])
```

Refer: https://pytorch.org/tutorials/beginner/text_sentiment_ngrams_tutorial.html

Train_iter which is a customTextDataset is an iterpipe that can now be fed into dataloader for batch processing.

Language Identification

In this part, we are training a simple neural network to detect 5 languages- Hindi, Kannada, Malayalam, Tamil, and Bengali.

Dataset Preparation

We have training, testing, and validation dataset of 5 languages stored separately. We append them together and also introduce a new label column indicating the language. Thus, our dataset looks like this. Here, *hi* is label for Hindi and *ta* is the label for Tamil.

	lang	text
0	hi	मेट्रो की इस लाइन के चलने से दक्षिणी दिल्ली से...
1	hi	नेटिजन यानि इंटरनेट पर सक्रिय नागरिक अब ट्विटर...
2	hi	इसमें एक फ़्लाइट एटेनडेंट की मदद की गुहार है औ...
3	hi	प्रतीक खुलेपन का, आज़ाद ख्याली का और भीड़ से अ...
4	hi	ख़ासकर पिछले 10 साल तक प्रधानमंत्री रहे मनमोहन...
...	...	...
52822	ta	எம் . ஜி . ஆர் காலத்திலிருந்து விமல் காலம் வரை...
52823	ta	கப்பல்கள் மோதிய விவகாரத்தில் கடந்த 8 நாட்களாக ...
52824	ta	பிக்பாஸ் நிகழ்ச்சியில் தற்போது தலைவியாக இருக்க...
52825	ta	பரதன் இயக்கத்தில் விஜய் நடித்துள்ள அவருடைய 60வ...
52826	ta	இந்திய அணியின் கேப்டன் விராட் கோலி , ஒவ்வொரு ப...

52827 rows × 2 columns

Methodology

We first create a bag of words from all the sentences in the table. We will only consider those words that have minimum frequency of 10. This Bag of Words will have all the words that occur in our training sample. Here is the BoW that we created from the table above-

```
vocab.get_itos()[10:22]
```

```
['और', 'से', 'को', 'ಹಾಗೂ', 'ಎಇ', 'IST', 'ಎಂದು', 'कि', "'", 'का', 'ने', 'ಒಂದು']
```

We will train our model on these Bag Of Words. For training, we have used embedding dimension of 128 and have lineary modify each layer. So the first linear operation is on input vocab_size*embed_size. The second liner operation coverts it into a vector of 5 dimensions which is the number of classes that we have in our dataset.

For training we have used SGD optimizer and cross entropy loss. We are also dynamically decaying the learning rate at some epochs. Initially when we ran the model, our model was performing exceptionally well on training data. To avoid overfitting, we have used a scheduler.

Results

Results for Language detection are very encouraging. We saw that our simple model was correctly able to classify cases with the maximum accuracy. This is because the words/script used for different languages are different and therefore there should not be any confusion as the model becomes more learned. This revelation also adds to how we learn as humans- once we know th script of Bengali and Hindi, differentiating between the two languages is not a hassle even though we don't know the meaning of the words. Here are the results from our training-

```
| epoch   1 |   500/  785 batches | accuracy    0.861
-----------------------------------------------------------
| end of epoch   1 | time:  9.22s | valid accuracy    0.983
-----------------------------------------------------------
| epoch   2 |   500/  785 batches | accuracy    0.989
-----------------------------------------------------------
| end of epoch   2 | time:  7.74s | valid accuracy    0.994
-----------------------------------------------------------
| epoch   3 |   500/  785 batches | accuracy    0.997
-----------------------------------------------------------
| end of epoch   3 | time:  7.81s | valid accuracy    0.998
-----------------------------------------------------------
| epoch   4 |   500/  785 batches | accuracy    0.999
-----------------------------------------------------------
| end of epoch   4 | time:  7.76s | valid accuracy    1.000
-----------------------------------------------------------
| epoch   5 |   500/  785 batches | accuracy    0.999
-----------------------------------------------------------
| end of epoch   5 | time:  8.12s | valid accuracy    1.000
-----------------------------------------------------------
| epoch   6 |   500/  785 batches | accuracy    0.999
-----------------------------------------------------------
| end of epoch   6 | time:  7.86s | valid accuracy    1.000
```

Future Work

It would be interesting to see how our Language Model would perform for languages in same Indic Script. For example, there are several languages like Hindi, Bhojpuri, Magadhi, Maithili that are written in the Devnagri script.

Text Classification

In this part, we classify text clippings into some broad categories- Sports, Business, Entertainment, etc.

Dataset Preparation

In this exercise, we classify texts for Hindi, Kannada and Malayalam. However, we carry the classification text separately for each language. The snippet given below is of Hindi, but that of Kannada and Malayalam looks the same.

	label	text
0	india	मेट्रो की इस लाइन के चलने से दक्षिणी दिल्ली से...
1	pakistan	नेटिजन यानि इंटरनेट पर सक्रिय नागरिक अब ट्विटर...
2	news	इसमें एक फ़्लाइट एटेनडेंट की मदद की गुहार है औ...
3	india	प्रतीक खुलेपन का, आज़ाद ख्याली का और भीड़ से अ...
4	india	ख़ासकर पिछले 10 साल तक प्रधानमंत्री रहे मनमोहन...
...	...	...
3462	india	जैसे ही उन्हें पता चलता है कि कोई व्यक्ति परेश...
3463	india	जैसे ही सदन की कार्यवाही शुरू हुई तमिलनाडु की ...
3464	news	चीन ने पिछले हफ़्ते अप्रत्यक्ष रूप से भारत को ...
3465	entertainment	मुक्ता आर्ट्स की 'कांची' कहानी है एक ख़ूबसूरत ...
3466	entertainment	फॉक्स स्टार स्टूडियोज़ और विशेष फ़िल्म्स की सि...

3467 rows × 2 columns

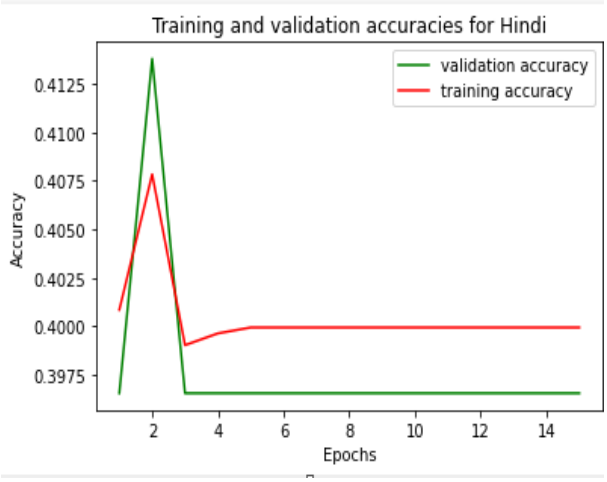
Methodology

The methodology for this exercise is same as that of the language detection task. We have linear operations on embedding layer and in the end we get probabilities of each of the classes.

Results

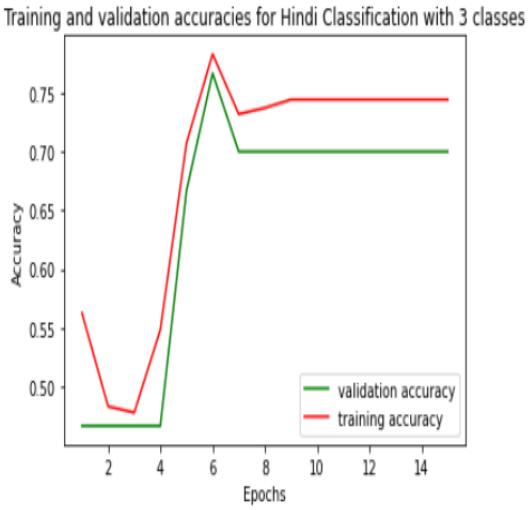
We first ran the data on Hindi dataset. This dataset is small (~2500) and the number of classes are also large- 14. As expected, we didn't get very encouraging results.

```
-----------------------------------------------------------
| end of epoch   1 | time:  5.17s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   2 | time:  4.21s | valid accuracy    0.414
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   3 | time:  4.87s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   4 | time:  4.29s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   5 | time:  4.20s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   6 | time:  4.15s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   7 | time:  4.16s | valid accuracy    0.397
-----------------------------------------------------------
-----------------------------------------------------------
```



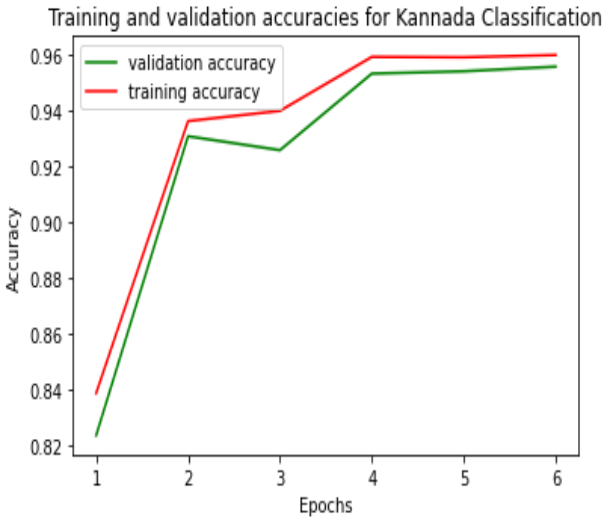
We can see that model performs poorly on classification of Hindi text. This is because the number of observations in are very few and number of classes are large. Moreover, it is very difficult to differentiate between some of the classes. For example, sentence belonging to class ‘India’ and sentence belonging to class ‘Sports’ might have intersection of words. Therefore, to avoid this problem, we only include observations in our dataset that belong to either entertainment, business, or sports. While this reduced the size of our dataset significantly, it increased the probability of classification. One reason is that the classes in this case are very distinct. THerefore, our model through Bag of Words is able to recognize the distinct features of these classes.

```
-----------------------------------------------------------
| end of epoch   7 | time:  0.59s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   8 | time:  0.58s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   9 | time:  0.56s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  10 | time:  0.58s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  11 | time:  0.59s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  12 | time:  0.60s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  13 | time:  0.60s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  14 | time:  0.61s | valid accuracy    0.700
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  15 | time:  0.57s | valid accuracy    0.700
-----------------------------------------------------------
```

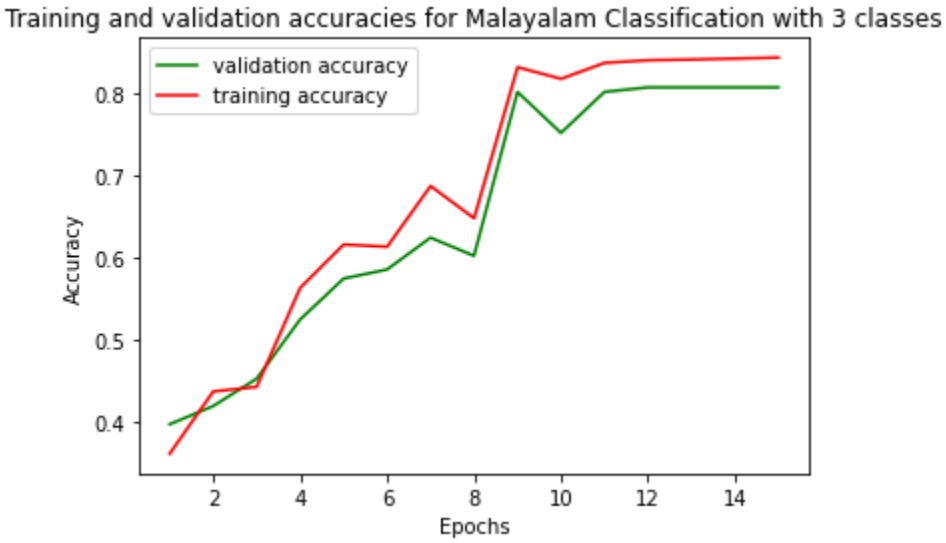


For classification of Kannada, the dataset is significantly large. The results for Kannada are very encouraging.

```
-------------------------------------------------------------
| end of epoch   1 | time: 43.11s | valid accuracy    0.823
-------------------------------------------------------------
-------------------------------------------------------------
| end of epoch   2 | time: 41.33s | valid accuracy    0.931
-------------------------------------------------------------
-------------------------------------------------------------
| end of epoch   3 | time: 41.24s | valid accuracy    0.926
-------------------------------------------------------------
-------------------------------------------------------------
| end of epoch   4 | time: 41.37s | valid accuracy    0.953
-------------------------------------------------------------
-------------------------------------------------------------
| end of epoch   5 | time: 41.59s | valid accuracy    0.954
-------------------------------------------------------------
-------------------------------------------------------------
| end of epoch   6 | time: 49.01s | valid accuracy    0.956
-------------------------------------------------------------
```



Malayalam also follows the similar trend.



Future Work

Through above experiments, we found that text classificatio task can be implemented on Indic languages. However, to be able to classify larger number of classes (with intersections among classes), we need a very large training dataset.

PoS Tagger

In this section we tag each hindi work with their respective part of speech.

Dataset Preparation

Indian Institute of Technology, Bombay have done a lot of work in Natural Language Processing for Indic Languages. They have been working on their own part of speech tagger. However, their tagger is not very each to use and needs a lot of functionalities to work with. Therefore, we only use their training data and train our own PoS Tagger. Their training data looks like this-

```
परिचय_[ NN ] का_[ PSP ] यह_[ DEM ] भ्रम_[ NN ] काफ़ी_[ INTF ] स्वाभाविक_[ JJ ] है_[ VM ] ._[ SYM ]
पुरानी_[ JJ ] पीढ़ी_[ NN ] के_[ PSP ] भारतीय_[ NN ] --_[ SYM ] हमारे_[ PRP ] पिता-पितामह_[ NN ] --_[ SYM ]
```

In the example above, PoS is in square brackets and word is mentioned before the '_'. We import the text file into Pandas and convert it into a more usable format.

	pos	clean_sentence
0	[JJ, NN, PSP, NN, SYM, PRP, NN, SYM, NN, PSP, ...	[पुरानी, पीढ़ी, के, भारतीय, --, हमारे, पिता-पित...
1	[JJ, SYM, JJ, SYM, NST, VM, PSP, JJ, NN, PSP, ...	[बदमिज़ाज, ,, चिड़चिड़े, ,, दूर-दूर, रहने, वाले, ...
2	[PRP, NN, PSP, INTF, JJ, NN, PRP, NN, VM, VAUX...	[अपनी, ऐंठ, में, थोड़ा-बहुत, भद्र, पीढ़ी-दर-पीढ़ी...
3	[PRP, NN, VM, CC, CC, PRP, SYM, NN, SYM, CC, S...	[मुझे, विश्वास, है, कि, अगर, उनकी, `, खुशमिज़ाज...
4	[CC, PRP, PRP, PRP, NN, PSP, NN, VM, VAUX, VAU...	[और, जब, मैं, अपने, पिता, की, बात, कर, रहा, ह...

To make it compatible with PyTorch, we create one list for each observation above- the first element (which is also a list) of the list consists of all the sentences and the second element consists of their respective Parts of Speech.

Methodology

Before training the model, we first do some pre-processing on our dataset. The neural network that we are going to design can only accept fixed-sized inputs. Each input is a subsequence of one of the examples in the above dataset. We also a define a window, W. Each subsequence would consist of 2*W+1 words with each W words before and after the center word. Along with this, the input will also have the PoS tag for the center word. Let's say we have the following sentence- **"**आज चिकागो में बोहोत ठंड है ।**"**। Preprocessing converts this sentence into -

"आज चिकागो में**"- NN**
"चिकागो में बोहोत**"- PRO**
"में बोहोत ठंड**"- ADV**

We also padded the sentence with <s> and </s> in the beginning and the end.

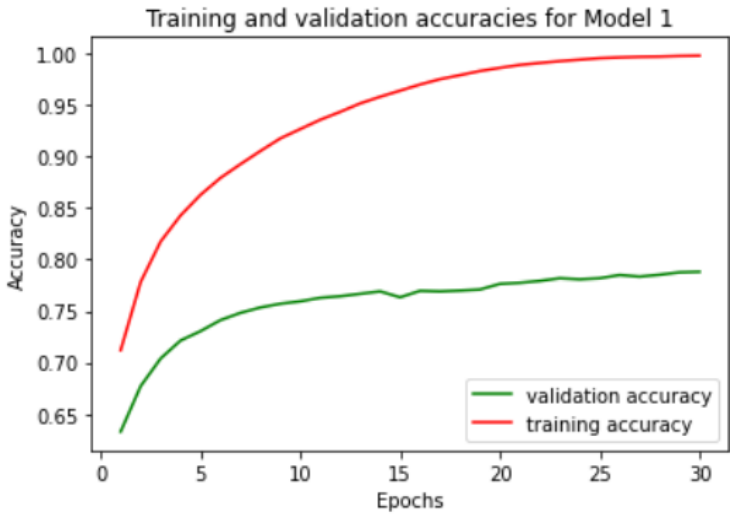
We will implement a two layered neural network-

$$
\begin{aligned}
x &= [E_{[w_1]}, E_{[w_2]}, E_{[w_3]}] \\
h &= \tanh(xW^1 + b^1) \\
\tilde{y} &= hW^2 + b^2 \\
\hat{y} &= \text{softmax}(y)
\end{aligned}
$$

So each input would be a string of three words. Each word is represented as a vector of length 300. We concatenate these three words into a single vector *x* of length 900. We pass this input through a hidden layer with 128 dimensions (or nodes) and apply tanh. The output of this hidden layer is passed through a final layer when we apply linear function and then softmax to convert it into probabilities. The final output will be probabilities assigned to each PoS class that we had. The class that has maximum probability is the PoS for the center word in our input. We use cross entropy and adam optimizer for training.

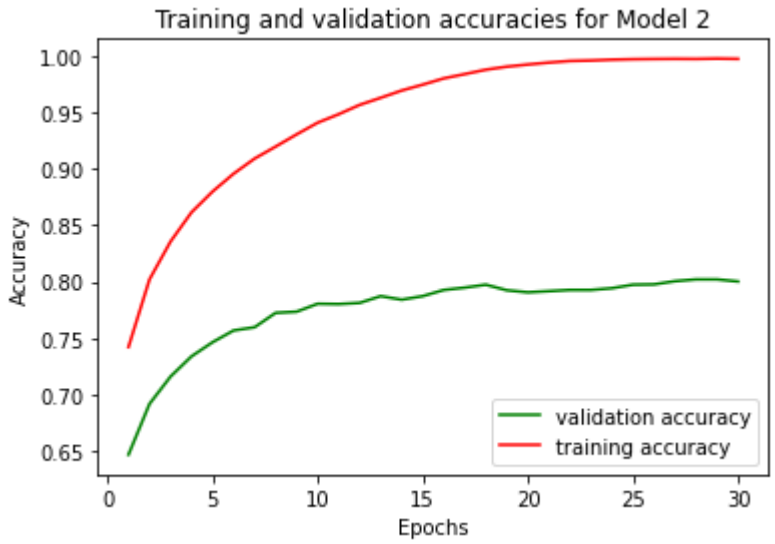
Results

We start by running the above model with the specifications described above. We get the following results -

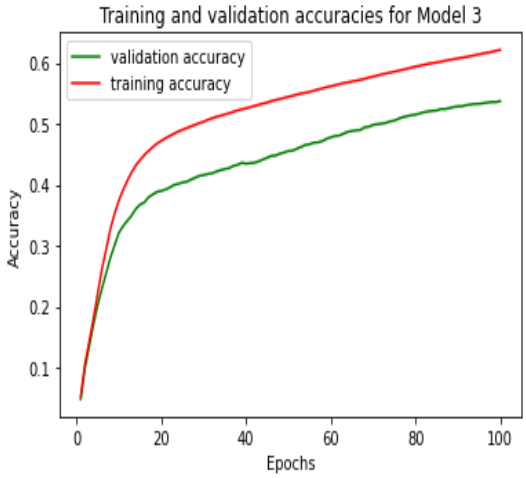


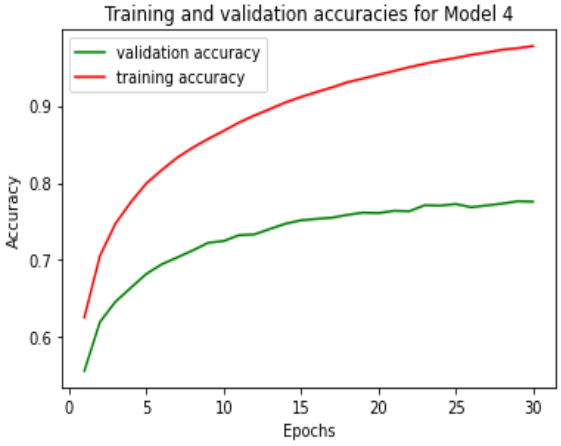
As we can see, the validation accuracy for the model hovers around 75%. Moreover, our model is severely overfitting as the difference between training and validation accuracy is significantly large.

In the above model, we first try by increasing the dimension of the hidden layer from 128 to 256. The result that we get is similar-



We try two other models- In model 3, we reduce the size of embedding dimension to 200 and size of hidden dimension to 64. We also change our optimizer to SGD from Adam. In Model 4, we stick with Adam but change the embedding dimension to 200 and hidden dimension to 64 to prevent overfitting. The results of these models remain the same-





Future Work

It would be interesting to work with more observations to see if we can overcome the problem of overfitting. Our current training dataset is of size ~2500 and validation dataset is of size ~400.

Sentence Generation task in Hindi language:

In this section, we have generated sequences/sub-sequence in hindi language based on a given prompt. This was a multi-layer long short-term memory (LSTM) and (GRU) RNN application to an input sequence similar to the HW4 attempted in the class.

Dataset Preparation:
As described above, the data was downloaded from Kaggle (Hindi Wikipedia 55k articles). The data was cleaned and downsized for the requirements to suit the sentence generation purposes.

Methodology:
Standard LSTM and GRU:

The LSTM model may be represented as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Where h_t is the hidden state at time t, C_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at the time t-1 or the initial hidden state at time o, and it, ft, gt, ot are the input, forget, cell, and output gates, respectively. Normal sigmoid and hadamard product is used.
(Ref: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html)

For GRU model, we use:

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

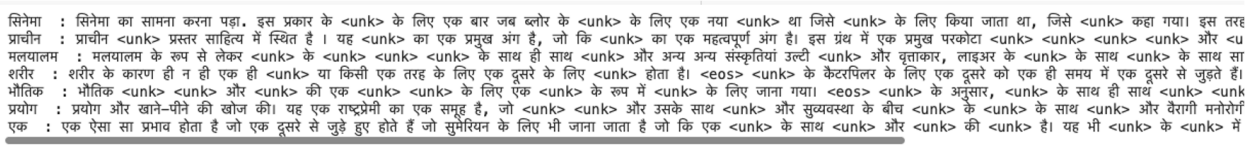
Where h_t is the hidden state at time t, X_t is the input at time t, h_{t-1} is the hidden state of the layer at the time t-1 or the initial hidden state at time o, and r_t, z_t and n_t are the reset, update and new gates respectively. Normal sigmoid and hadamard product is used.

(Ref: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html?highlight=gru#torch.nn.GRU)

As the code is available on Github at:
https://github.com/Tiru-Kaggundi/Adv_ML_project/blob/main/RNN_based_sentence_generation.ipynb
The same is not repeated here. The code follows HW4 of the class very closely.

Results:

a) LSTM produced very impressive results:



The above sentences were generated based on the prompts given. The sentence sub-sequences make a lot of sense and show tremendous promise.

b) GRU also produced extremely good results:

```
words_list = ["सिनेमा", "प्राचीन", "मलयालम", "शरीर", "भौतिक", "प्रयोग", "एक"]
for i in words_list:
  print(i," :", sample(best_model_GRU, 50, prime=i))
```

सिनेमा : सिनेमा के लिए एक नया और हिस्सा था। इस प्रकार, एक बार एक नया आश्चर्य है की इ
प्राचीन : प्राचीन और अरुणाचल में <unk> के रूप में, यह <unk> की तरह के लिए जाना जा रह
मलयालम : मलयालम और अधिक <unk> के साथ ही <unk> <unk> <unk> और <unk> के
शरीर : शरीर ही <unk> <unk> और अन्य संस्कृतियों के लिए भी एक विशेष <unk> के लिए एव
भौतिक : भौतिक ही <unk> है। <eos> <unk> के अनुसार, एक ही समय में, <unk> के लिए
प्रयोग : प्रयोग की और यह <unk> की एक श्रृंखला के लिए एक <unk> है। <unk> <unk> के
एक : एक विशेष महत्व था और इस तरह यह भी पता चला था क्योंकि वह अपने कानून-व्यवस्था के

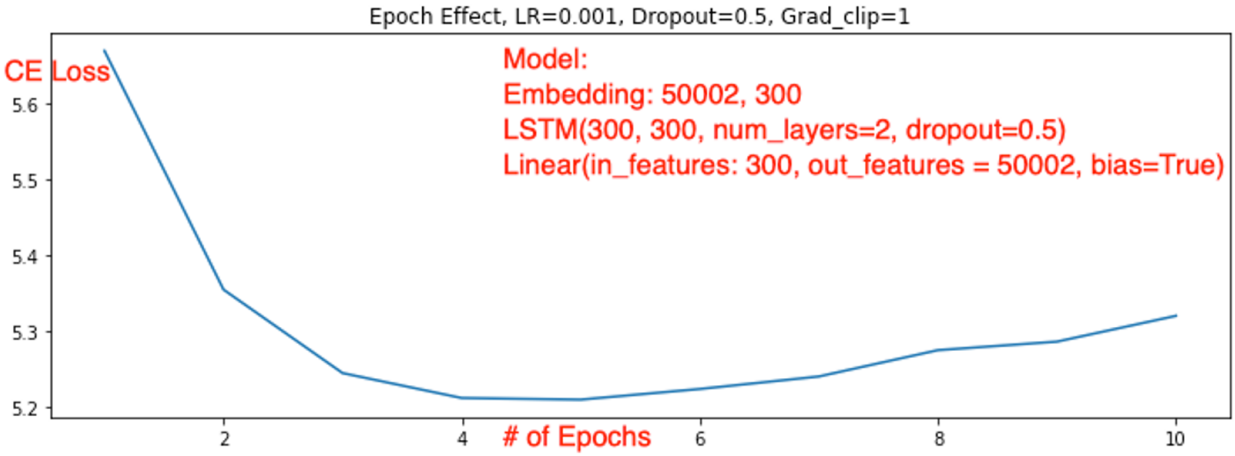
It may be seen (if one knows Hindi), that finer nuances of the language are also picked up quite well by the language models. For example, the gender identification of a word in Hindi is unique. This aspect is displayed in some sentences. The singular/plural words are also used in almost perfect ways.

Analysis on training and some observations:

Usually such analysis should be done with a grid search or other ways of iterating over the values of various hyperparameters. However, as our model was taking a long time to train (over

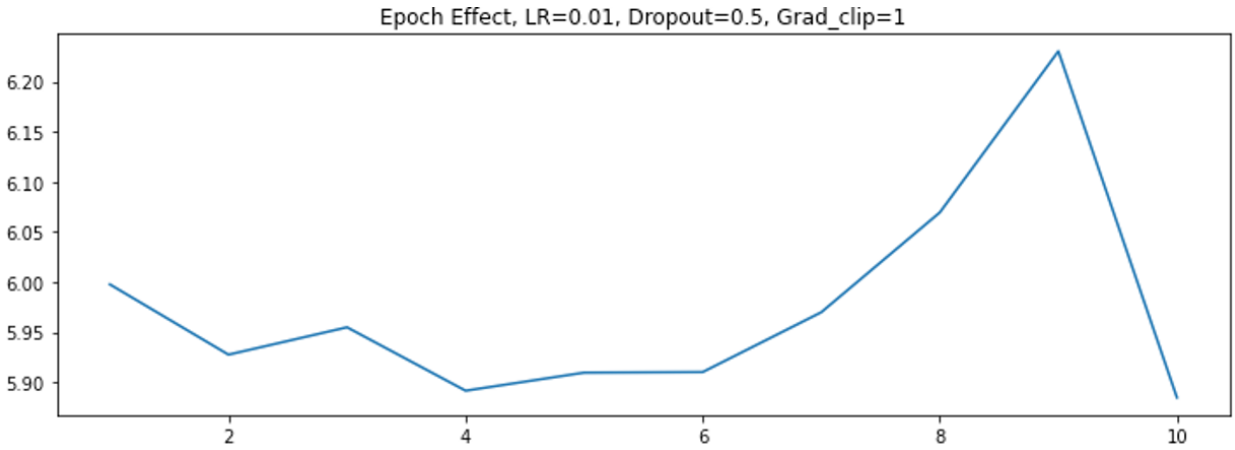
1.5 hours for around 10 epochs), we could do only limited experiments in the given amount of time.

The base LSTM model, with dropout of 0.5, grac_clip of 1, 2 layers with 300 embeddings trained well within 5 epochs as shown below.



Perplexity of best model on validation set: 181.8
Perplexity of best model on test set: 167.5

1. <u>Effect of changing learning rate from 0.001 to 0.01:</u>

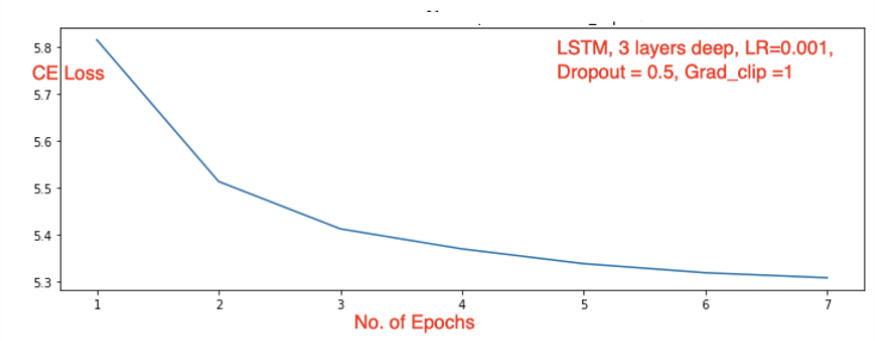


We expected a reduction in training time. However, the tim taken to train didn't reduce significantly. AT the same time, we noticed a deterioration in performance as measured by perplexity score.
Perplexity of best model on validation set: 765.5
Perplexity of best model on test set: 791.9

2. <u>Effect of changing stacks/layers of LSTM:</u>
   We also tried out with 3 and 4 layer stacks in place of 2. The results indicated that the best sentence sequences are generated for 4 layer stacks. However, the perplexity

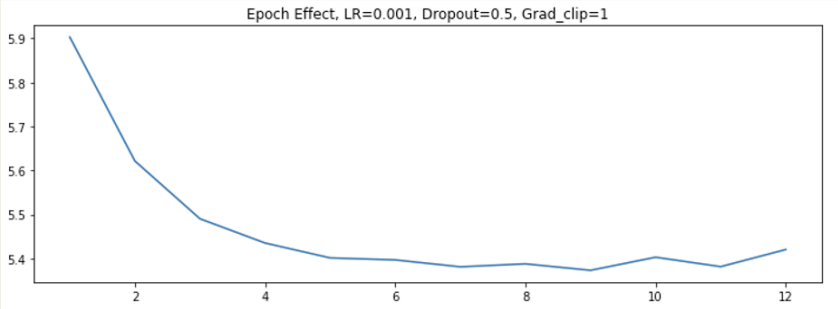
scores also go up with we train it with more layers.

3 Layers model



Perplexity of best model on validation set: 202.6
Perplexity of best model on test set: 196.1

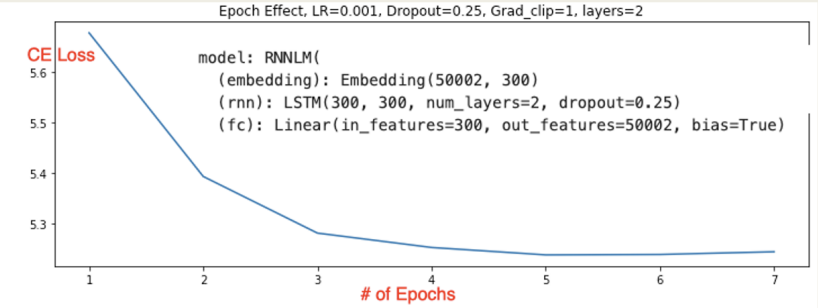
4 Layers model



Perplexity of best model on validation set: 212.4
Perplexity of best model on test set: 203.4

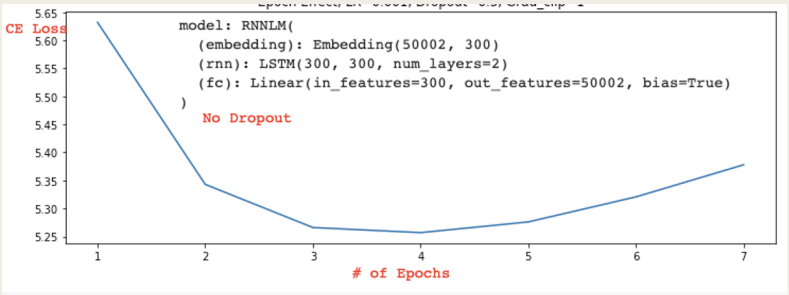
3. <u>Effect of changing dropouts (no dropout, 0.25, 0.5, 0.75):</u>
   The best output was noticed for a dropout value of 0.5 (base model initial). The altered models are shown below:

Effect of Dropout (from 0.5 to 0.25)



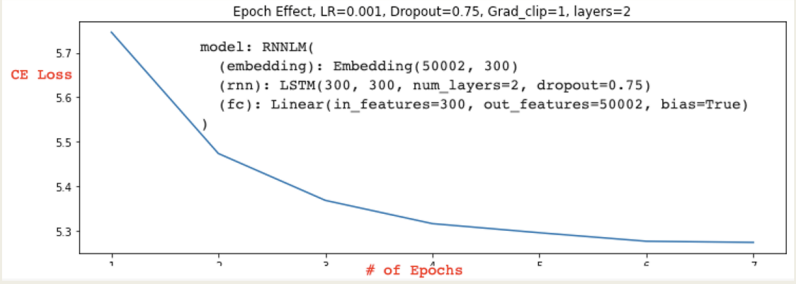
Perplexity of best model on validation set: 185.37 (vs. 181 in base model with dropout = 0.5)
Perplexity of best model on test set: 176 (vs 167 in base model with dropout = 0.5)
Conclusion: 0.5 was better than 0.25

Effect of Dropout (from 0.5 to 0) – no dropout



Perplexity of best model on validation set: 192.4 (vs. 181 in base model with dropout = 0.5)
Perplexity of best model on test set: 188.4 (vs 167 in base model with dropout = 0.5)
Conclusion: 0.5 was better than 0

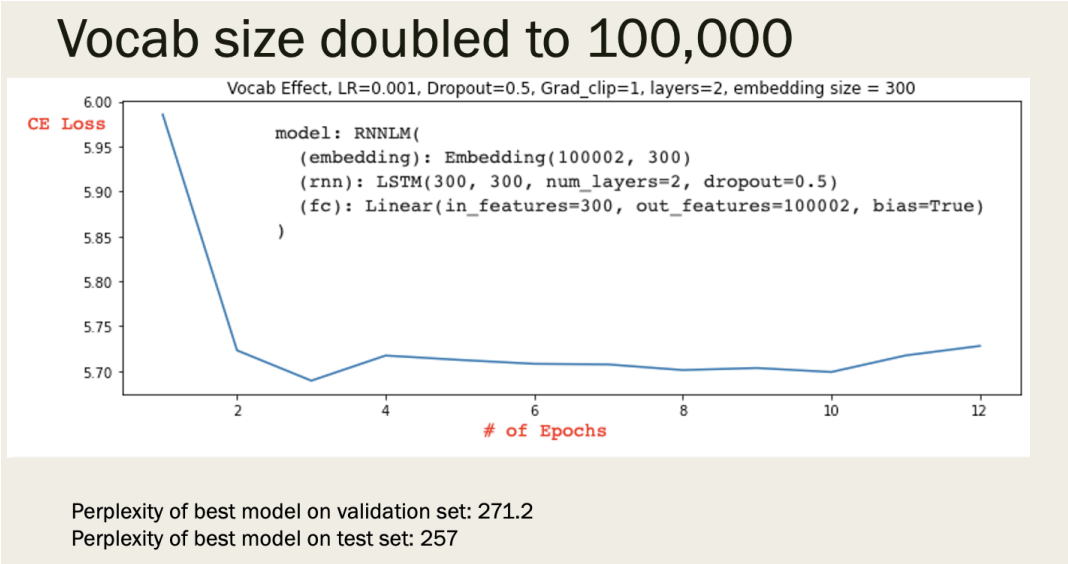
Effect of dropout = 0.75



Perplexity of best model on validation set: 197.9 (vs. 181 in base model with dropout = 0.5)
Perplexity of best model on test set: 192.3 (vs 167 in base model with dropout = 0.5)
Conclusion: 0.5 was better than 0.75

4. Effect of increasing vocabulary size from 50000 to 100000:

Contrary to expectations, we didn't find significant improvements when we increased the vocabulary size on our standard LSTM. This is probably because we couldn't increase the training size.



Due to paucity of time and resources, we couldn't carry out experiments on the GRU model. We are however sure that given enough time and resources, we can tweak the model and the training data to produce much better results and conclusively arrive at optimized parameters for the model.

Impact and road ahead

The results that we have gotten from this paper are really encouraging. However, we are glad that we were able to successfully run various NLP models on wide range of datasets that comprised of different Indian languages. The data and model constraints that we faced and overcame can be used a template for future research in this area.

The first challenge was to collate datasets from multiple sources and make it compatible with PyTorch. We spent several hours converting the pandas dataframe and to PyTorch custom dataset and the Class that we formed for this task can be used for other related work.

We performed four tasks- Language Detection, Sentence Classification, Parts of Speech Tagging, and Sentence Generation. The results that we got in all of these tasks are very encouraging. With diligent data collation and little modifications to models, we can run several NLP algorithms on Indic languages.

As NLP in Indic Languages is still an upcoming field, there is not enough documentation online. **We are planning on writing a blog of all the challenges that we faced and how we can seamlessly collate data, convert it into PyTorch compatible dataloader, and run standard**

models like GRU, LSTM, and Neural Networks. Hopefully, this blog will guide researchers working on topics like Fake Language Detection, Toxic Comments Classification, or Hate Speech Blocking to be more focussed on the task rather than the cleaning and collation process.

References

Rahul Venkatesh RM Kumar, Anand M Kumar, and KP Soman. AmritaCEN NLP @ FIRE 2015 Language Identification for Indian Languages in Social Media Text. In Working Notes of the Forum for Information Retrieval Evaluation (FIRE 2015), pages 28–30, Gandhinagar, India, 2015.

S. Das, S. B. Partha and K. N. Imtiaz Hasan, "Sentence Generation using LSTM Based Deep Learning," *2020 IEEE Region 10 Symposium (TENSYMP)*, 2020, pp. 1070-1073, doi: 10.1109/TENSYMP50017.2020.9230979.

M. Srivastava, P. Bhattacharyya. “Hindi POS Tagger Using Naive Stemming : Harnessing Morphological Information Without Extensive Linguistic Knowledge”. Department of Computer Science and Engineering. Indian Institute of Bombay.