

Codemix generation and text fluency

INLP project report

Akshit Kumar, Aryan Chandramania, Lakshmipathi Balaji

May 6, 2023

Contents

1	Introduction	3
2	LSTM Model	3
2.1	Description	3
2.2	Architecture	3
2.3	Architecture of Encoder-Decoder	3
2.3.1	Understanding the Encoder part of the model:	4
2.3.2	Understanding the Decoder part of the model in the Training Phase:	4
2.4	Hyperparameter tuning	4
2.5	Results	7
3	IndicBART	10
4	mT5	11
4.1	Details	11
4.2	Implementation	11
5	Appendix	11
5.1	Challenges in the dataset	11
5.2	Challenges while training	12
5.2.1	LSTM	12
5.2.2	mT5	12
5.2.3	IndicBART	12
5.3	Challenges post training	12
5.3.1	LSTM	12
5.3.2	mT5/IndicBART	12
5.4	BLEU Scores	12
5.5	References	13
5.6	Acknowledgements	13
5.7	Comparisons of various models	14

1 Introduction

Our project aims to generate a Hindi-English (Hinglish) code-mixed sentence given a sentence in English. To this end, we used 3 different models – an LSTM encoder-decoder, the mT5-small model, and the indicBART model. The first one was fully trained by us, and the latter two were finetuned.

The dataset we used can be found [here \(LinCE\)](#) or [here \(Google Drive\)](#). It contains tab-separated English and Hinglish sentences on each line.

For evaluation, we have used BLEU scores till the 8-gram level. We have also created visualizations for easier viewing. Please refer the wandb project for more interesting observations.

2 LSTM Model

2.1 Description

This section aims to develop a neural machine translation model using an LSTM-based encoder-decoder architecture for English-to-Hinglish translation. The model is trained using a teacher-forcing approach, which involves feeding the correct output sequence from the previous time step as input to the decoder during training. Using teacher forcing can lead to faster convergence and better model accuracy. This section explores the effectiveness of this approach in improving the performance of the English-to-Hinglish translation model.

2.2 Architecture

The architecture of the seq2seq model consists of an encoder that processes the input English sentence and encodes it into a fixed-length vector representation, which is then fed into a decoder that generates the corresponding Hinglish translation word by word. The encoder and decoder are LSTM-based neural networks trained using a combination of teacher forcing to optimise the translation quality.

2.3 Architecture of Encoder-Decoder

The overall structure of sequence to sequence model(encoder-decoder) which is commonly used is as shown below-It consists of 3 parts: encoder, intermediate vector and decoder.

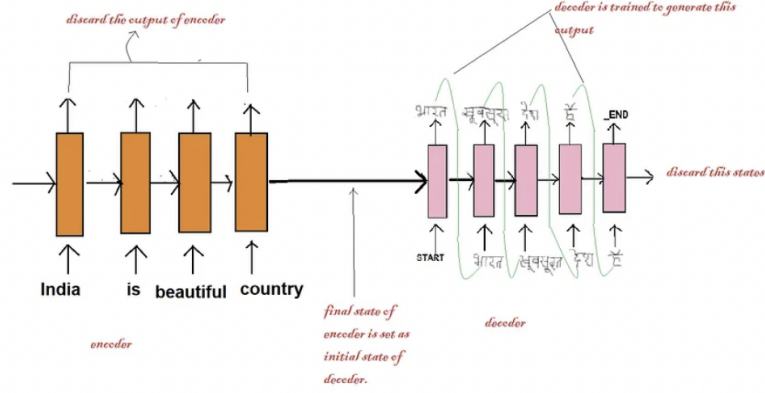


Figure 1: Architecture of LSTM based encode-decoder model

2.3.1 Understanding the Encoder part of the model:

The encoder is basically LSTM/GRU cell. An encoder takes the input sequence and encapsulates the information as the internal state vectors. Outputs of the encoder are rejected and only internal states are used.

2.3.2 Understanding the Decoder part of the model in the Training Phase:

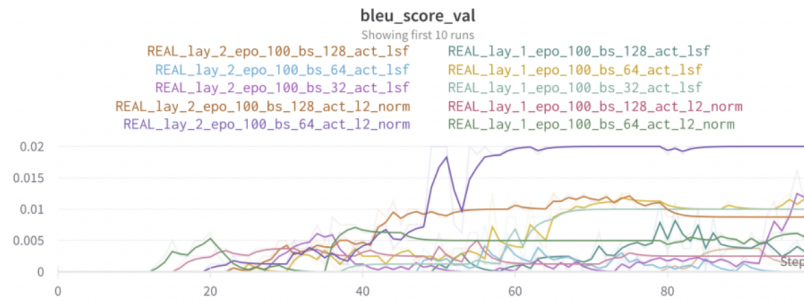
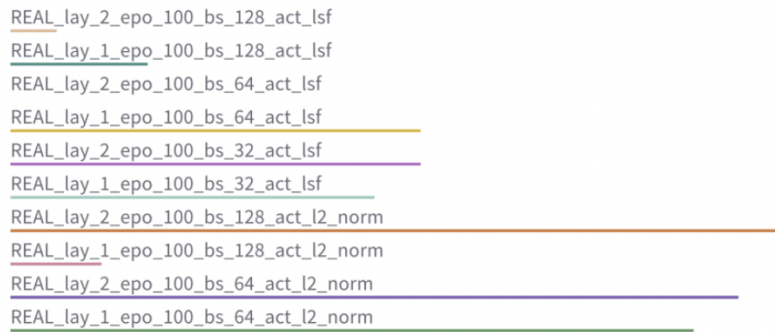
The working of the decoder is different during the training and testing phase. The initial states (h_0 , c_0) of the decoder is set to the final states of the encoder. Then the output sequence is trained to the decoder model, as shown in the FIGURE1, using the teacher-forcing mechanism.

2.4 Hyperparameter tuning

- batch-size = [32,64,128,256,512],
- embedding-dim = [300],
- hidden-dim = [512],
- num-epochs = [100],
- num-layers = [1,2, 3],
- activation = ["lsf" , "l2_{norm}"]

These parameters have been trained across multiple sweeps and wandb and few of the useful results for analysis to decide hyperparameters are kept below and discussed.

bleu_score_val



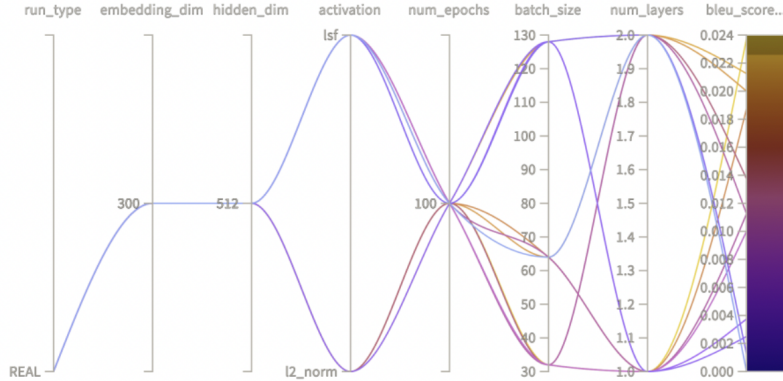
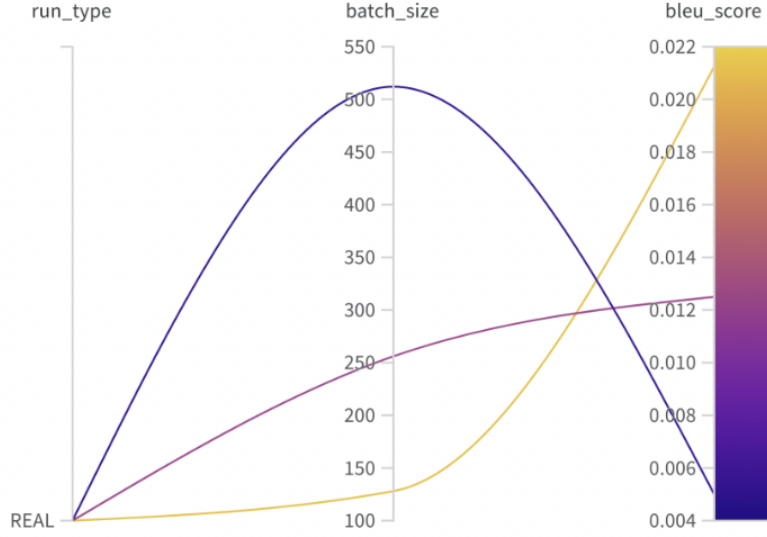


Figure 2: Parallel plots for various hyperparameters on multiple runs and their respective bleu-scores

In this study, we trained a sequence-to-sequence (Seq2Seq) model for English to Hinglish translation using PyTorch and analyzed the hyperparameters using the Weights and Biases (W&B) tool. We experimented with three different batch sizes (32, 64, 128), two activation functions (log-softmax and l2-normalization), and two LSTM layer configurations (1,2) for 100 epochs each. Our analysis of the W&B graphs showed that the top-3 models with the best validation performance had the same activation function (l2_{norm}) and used one layer each for the encoder and decoder LSTMs. Additionally, we observed that the batch size of 128 performed the best on the validation data. Although we were not able to experiment with different hidden dimensions due to the high computational requirements, we believe that our findings provide important insights for optimizing the Seq2Seq model for English to Hinglish translation.

Note that we have done this hyperparameter tuning for only 20% of the dataset due to compute limitations.



2.5 Results

After figuring out the hyperparameters model has been trained with the parameters -

- batch-size = [128],
- embedding-dim = [300],
- hidden-dim = [512],
- num-epochs = [100],
- num-layers = [2],
- activation = ["lsf" , "l2_{norm}"]

And the results are explained through graphs below.

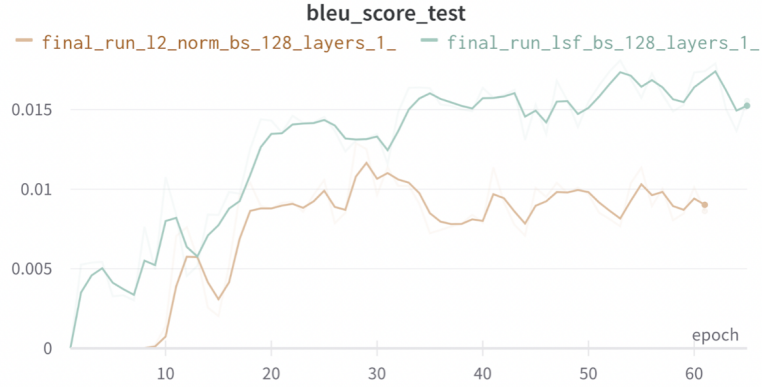


Figure 3: bleu-score-test on test_{set} comparison between 2 different runs - observe that the l_2 normalization out performs the $\log_{\text{softmax}}(\text{lsf})$ which was observed in hyperparameter tuning

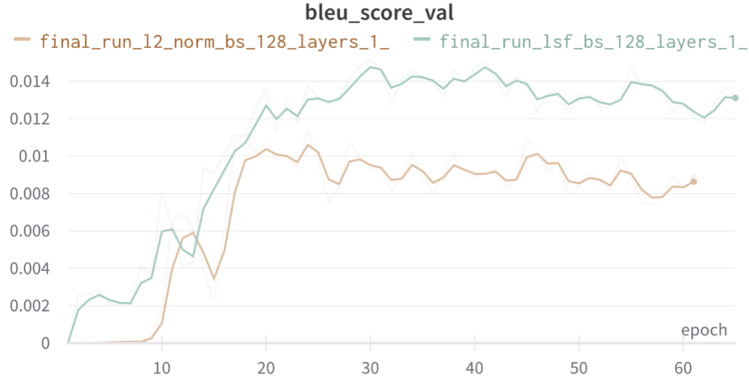


Figure 4: $\text{bleu}_{\text{score test}}$ on test_{set} comparison between 2 different runs - observe that the l_2 normalization out performs the $\log_{\text{softmax}}(\text{lsf})$ which was observed in hyperparameter tuning



Figure 5: It is observed that in the case of \log_{softmax} activation the $\text{train}_{\text{loss}}$ reduces gradually and in the case of $\text{l2}_{\text{normalization}}$ there is a sudden drop in the $\text{train}_{\text{loss}}$ at initial epochs but then the model improved its performance over epochs though the $\text{train}_{\text{loss}}$ wasn't reducing much.

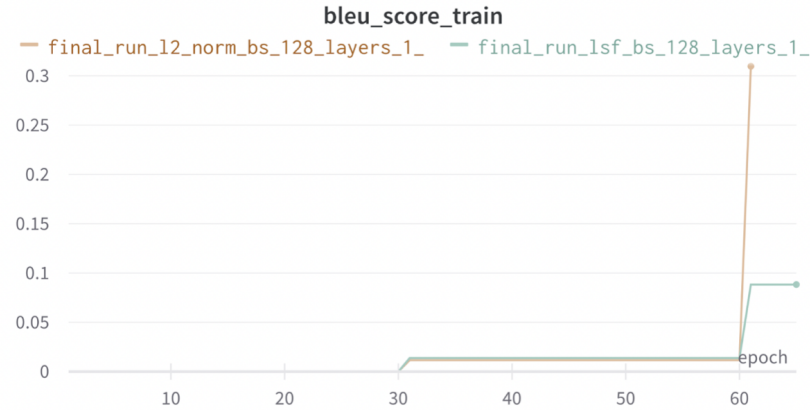
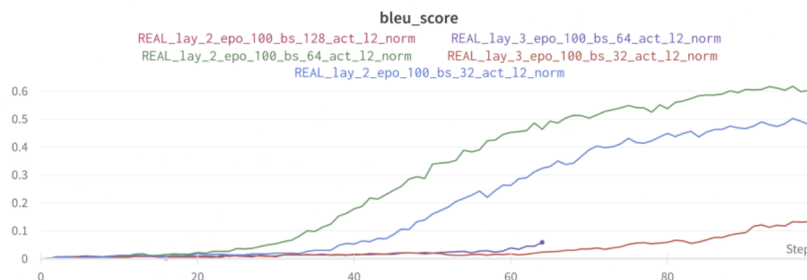


Figure 6: It can be seen that the model with \log_{softmax} activation learns and overfits the given training set well much quicker than the $\text{l2}_{\text{normalisation}}$ as it is gradually learning from the training set



Observe that the bleu-score at the bigram level is more fluctuating than the unigram since it is much tougher to predict the exact bigrams than unigrams as a translated sentence.

**Are the scores of the model really low, is the model inefficient?
Or is it just compute limitations?**



When trained on a part of the dataset (20% of the $\text{train}_{\text{set}}$) the model had out-performed the actual scores for 20% of the test_{set} . So it can be understood that it's the compute that the model requires to improve itself.

3 IndicBART

IndicBART is a multilingual, sequence-to-sequence pre-trained model focusing on Indic languages and English. It currently supports 11 Indian languages and is based on the mBART architecture. The model is much smaller than the mBART and mT5(-base) models, so less computationally expensive for finetuning and decoding. This was experimentally observed when we saw that the IndicBART model was able to load and train on the entire dataset with ease, while the mT5 model was only able to train on ~ 1500 sentences without exceeding Colab's RAM limits.

It used the IndicCorp data spanning 12 languages with 452 million sentences (9 billion tokens). The model was trained using the text-infilling objective used in mBART.

In order to finetune the model, we used HuggingFace's APIs.

4 mT5

4.1 Details

mT5 is the multilingual variant of the “Text-to-Text Transfer Transformer” (T5)” which leveraged a unified text-to-text format and scale to attain state-of-the-art result on a wide variety of English language NLP tasks. mT5 was pre trained on a new common crawl based dataset covering 101 languages – we utilized the hindi-english capabilities of this model to work on code mixed datasets.

4.2 Implementation

Initialized the tokenizer and the model from huggingface,

```
tokenizer = MT5Tokenizer.from_pretrained("google/mt5-small")  
model = MT5ForConditionalGeneration.from_pretrained("google/mt5-small")
```

After pre-processing the data, passed it into the model trainer:

```
trainer = Trainer()
```

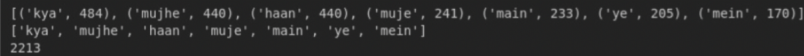
And trained the model:

```
trainer.train()
```

5 Appendix

5.1 Challenges in the dataset

The dataset was heavily skewed towards some words in the beginning of sentences, which would have made the model learn to only start with those words.



```
[('kya', 484), ('mujhe', 440), ('haan', 440), ('muje', 241), ('main', 233), ('ye', 205), ('mein', 170)]  
['kya', 'mujhe', 'haan', 'muje', 'main', 'ye', 'mein']  
2213
```

Figure 7: Seven words made up 2213/8000 sentences

In order to deal with this, we decided to keep a 5% threshold for such recurring beginning words so as to make the dataset meaningful.

This left us with ~5800 sentences in the training set.

5.2 Challenges while training

5.2.1 LSTM

To implement a teacher-forcing mechanism, we had to go through a lot of different architectures, make observations and try different things. The training time was longer than the other 2, roughly 2 and a half hours.

5.2.2 mT5

We were unable to finetune with the entire dataset, because it required too much RAM and exceeded Google Colab’s memory limits. Hence, we had to cut down on the amount of training data we used. Changing the number of epochs had no impact on how much memory was required.

5.2.3 IndicBART

Being a smaller model, it was able to load the entire training dataset. There were some issues training with HuggingFace’s AutoModel, so we had to look for alternatives. Training times for mT5 and IndicBART were both smaller than the LSTM.

5.3 Challenges post training

5.3.1 LSTM

We had to run for more than 60 epochs to get any kind of results on the model, and even then they were not very good. Words often repeated themselves.

5.3.2 mT5/IndicBART

Bad/Deprecated API and functions. The model would not work without those deprecated elements, so even though we would like to avoid it, we were forced to use them. The guides and tutorials on HuggingFace were only partially helpful in terms of how to load data and finetune the models using it.

5.4 BLEU Scores

BLEU (Bilingual Evaluation Understudy) is a metric used to evaluate the quality of machine translation output by comparing it to one or more reference translations. The basic idea behind BLEU is to compare the n-gram

overlap between the machine translation output and the reference translations. The metric works by counting the number of times each n-gram (a sequence of n words) appears in both the machine translation output and the reference translations. The resulting counts are then used to calculate a score that ranges from 0 to 1, with 1 being a perfect match between the machine translation output and the reference translations. The metric can be used to compare different machine translation systems, as well as to track the progress of a single system over time – as we have done in LSTM.

We decided to choose BLEU because it is language-independent, and so did not require us to do anything special in order to be used.

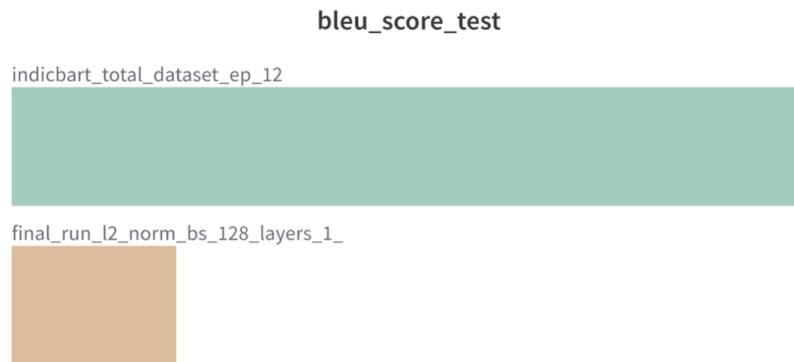
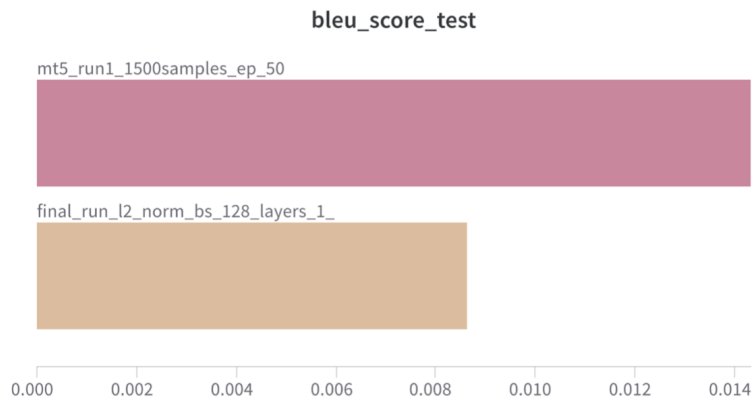
5.5 References

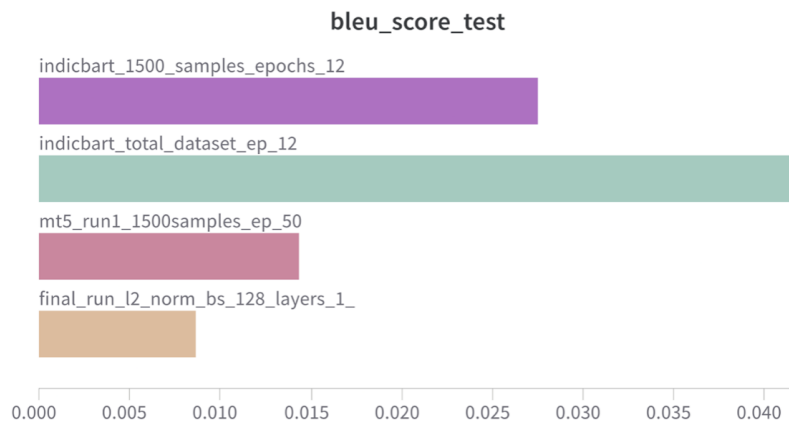
- Sentence Bleu-Score in NLTK
- A guide for seq2seq encoder-decoder LSTM based model for machine translation
- mT5
- IndicBART

5.6 Acknowledgements

Prof. Manish Shrivastava, for giving us the opportunity to work on this project. Teaching Assistants Ekansh Chauhan and Ankita Maity, for guiding us. Prashant Kodali (LTRC), for providing guidance with the project.

5.7 Comparisons of various models





We also tried a seq2seq transformer based encoder-decoder model which had unreliable results thus was not included in the report.