CS779 Competition: Machine Translation System for India

Prakriti Prasad
230770
prakritip23@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

This report focuses on solving a sentence translation problem from English to Hindi and Bengali using LSTM models with Attention Mechanism. Sentences were tokenized and converted into sequences, followed by the application of a Seq2Seq model architecture, implemented from scratch using PyTorch. The final model performed well, achieving a CHRF score of 0.22, Rouge score of 0.24 and Bleu score of 0.05. It ranked 5th among the 19 submitted models on the leaderboard.

We use concepts introduced in [1] for encoder, decoder Seq2Seq Architecture, and attention mechanisms. Additionally, PyTorch tutorials provide implementation guidance [2].

1 Competition Result

Codalab Username: P_230770
Final leaderboard rank on the test set: 5
charF++ Score wrt to the final rank: 0.22
ROGUE Score wrt to the final rank: 0.24
BLEU Score wrt to the final rank: 0.05
Best Bleu Score was 0.11

2 Problem Description

The goal of this problem is to solve a sentence translation problem. We are given sentences in English and we need to predict their Hindi and Bengali translations.

In today's world, there are thousands of different languages, and we frequently have to converse with people who may not know our language. Sentence translation becomes a fundamental problem to solve to enable cross language communication. Further, a lot of resources need to be translated to different languages to ensure that it can reach everyone, and language is not a barrier to information and learning.

3 Data Analysis

3.1 Hindi Dataset

3.1.1 Training Dataset

There are a total of 80797 sentences with a English vocabulary size of 57168 and Hindi Vocabulary size of 74785.

The average size of the English sentences is 16.73 while the Hindi sentences average size is 18.75.

Rank	English Word	Frequency	Hindi Word (in English)	Frequency
1	the	86654	danda (full stop)	64915
2	of	51262	ke	62729
3	and	36310	hai	57719
4	in	32979	mein	47111
5	(space)	32312	ki	31644
6	to	29558	se	29152
7	is	29507	aur	29008
8	a	23583	hain	23450
9	for	11911	ko	22451
10	this	11240	ka	22444

Table 1: Most Common Words in English and Hindi with Their Frequencies

The space and the fullstop character remaining in the dataset shows small errors in the tokenization. However this is the overview of the most common words.

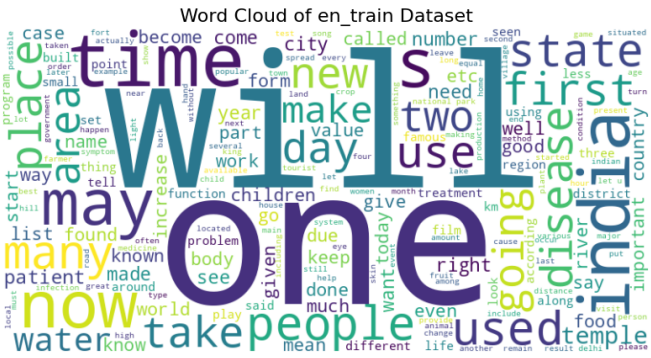

Figure 1: English Word Cloud

The word cloud showcases the most frequent data points in a visual format. More important words are depicted larger in the word cloud.

3.1.2 Validation and Test Datasets

The validation dataset contained 11543 more sentences and the new English vocabulary size became 61101, meaning there were about 4000 words which were not present at all in the English training dataset. The other distributions were similiar to the training dataset.

The test dataset contained another 22000 sentences to translate, and there were another 5000 new elements added to our vocabulary.

3.2 Bengali Dataset

3.2.1 Training Dataset

There are a total of 68849 sentences with a English vocabulary size of 53817 and a Bengali vocabulary size of 99392, nearly double!

The average size of the English sentences is 16.43 while the average size of the Bengali sentences is 14.23. Curiously, even though Bengali has a much larger vocabulary, it's average size of sentences is

shorter. This means that Bengali has more unique words as compared to English.

Rank	English Word	Frequency	Bengali Word (in English)	Frequency
1	the	77165	danda (full stop)	37299
2	of	46498	danda (full stop)	17593
3	and	30790	ebang	16592
4	in	30630	kore	10113
5	(space)	28732	ei	9125
6	to	23074	theke	8769
7	is	22963	jonno	8206
8	a	19962	kora	7559
9	for	9457	ekta	7313
10	are	8713	o	7171

Table 2: Most Common Words in English and Bengali with Their Frequencies

Figure 2: English Word Cloud

Here, the english dataset is significantly different from the hindi translation one, as we can see through the word cloud. This might be because the sources of the datasets are different. We can notice how India, time, state are few of the most important words in the word cloud.

3.2.2 Validation and Test Datasets

The validation dataset contained 9836 more sentences and the new English vocabulary size became 57487, meaning there were about 4000 words which were not present at all in the English training dataset. The other distributions were similar to the training dataset.

The test dataset contained another 19672 sentences to translate, and there were another 4000 new elements added to our vocabulary.

4 Model Description

4.1 Model Evolution

The method used to solve the problem was implementing Seq2Seq Architecture. The first iteration of the model was using GRU as the RNN model to implement the encoder and decoder. However, the results from this model were not satisfactory. The model failed to get the construction of valid

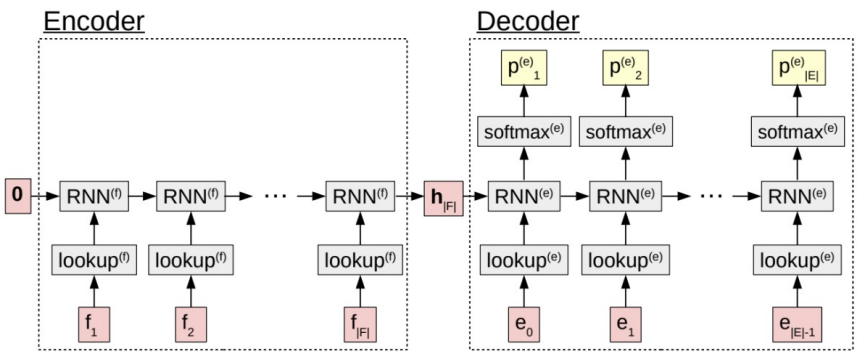

Figure 3: Encoder Decoder Model

sentences, and the output often involved a single word repeated multiple times. This indicated an underlying issue in how the probability of one word was being too high regardless of the previous context.

Next, LSTM was used as the underlying RNN architecture for the encoder and the decoder. LSTM stands for Long Short-Term memory, and as in the name, they are designed for problems that require long range memory. They are able to handle both long term and short term dependencies. Thus, this was a perfect fit for our task. This model performed better, producing reasonable results, without the previous issues. However, there was considerable room for improvement as the model often produced weakly related sentences as the output, and while mostly grammatically correct, they were not correct translations.

Attention Mechanism was incorporated into the Seq2Seq architecture. Attention Mechanism is a method that allows us to dynamically assign weights to the various components in our sequence depending on it's relative importance. The model showed significant improvements and it produced both grammatically correct sentences, while being likely to keep the core meaning of the sentence intact.

Finally, Glove was used instead of the default embedding matrix from Pytorch. This led to only a marginal improvement in results.

Attempts were made to incorporate transformers into the model architecture, however it was not a success.

4.2 LSTM Model Working

4.2.1 Encoder

The encoder takes as input a sequence of the size of the English dictionary, representing the words. Then, it has an embedding_matrix (either pytorch default, nn.Embedding, or pretrained Glove model), which converts these input sequences into dense vector spaces of smaller dimension, which are easier to work with.

Then, it has a LSTM RNN model from this embedding to produce the hidden context vectors and hidden cell. Finally, the output is of size hidden_size, which represents the context vector passed to the Decoder.

4.2.2 Decoder

The decoder has an internal embedding matrix, this time of the size of the Hindi/Bengali Dictionary. It also has a LSTM RNN model, and 2 linear functions : the Attention Mechanism function, and

another linear function fun which maps the Decoder outputs to the output vocabulary space.

The decoder accepts 3 things as input : Partially translated sentence (x) till now, and the hidden, cell from the encoder outputs. It then embeds x using its embedding matrix, and produces the final context vector using the Attention linear function. Then, the context, hidden and cell are passed to the LSTM to get the output. The output is passed through fun to produce the prediction.

4.2.3 Overall Working

We first pass the English sentence to the encoder. It produces hidden, cell as its outputs. The hidden is used for capturing short term information about the current time step, while the cell is used for long term persistent information storing, which are at the core of a LSTM model.

Then, we start with x = SOS, where x represents the partially translated sentence. We pass x, hidden, cell to the decoder, which gives us the prediction. This is of the size of the Bengali/Hindi vocabulary, and we take an argmax over it's dimension to get the most likely next word. Then, we append this to x, and repeat this process till we get EOS (or exceed our maximum length).

In the training loop, we sometimes use Teacher Forcing and instead of directly passing x, we pass the correct partially translated sentence till now, to optimize the training.

4.3 Loss Function

The loss function was chosen to be the popular choice nn.CrossEntropyLoss, nn.NLLLoss was another possible alternative, however that would require to first take the data to the log domain.

The data needed to be transformed into the correct dimensions before passing to the loss function, because the loss function does not directly take the output data in it's original dimensions.

5 Experiments

5.1 Data Pre-Processing

First of all, all punctuation and numbers were removed from the sentences. Then, the sentences were tokenized into their words using NLTK tokenizer. However, midway through the competition, there was an issue with NLTK tokenizer, so spacy was used for English, spacy-udpipe for Hindi, and Indic Tokenizer for Bengali.

The words were then assigned numbers to convert the tokenized sentences into sequences. Further, there were 3 extra tokens introduced, SOS, EOS and PAD, indicating the start of sentence, end of sentence, and padding token.

This was done to enable easy processing for our models, since they process sequences and not sentences directly. All the sentences were padded to be of uniform length, again for easy processing through model (27 for Hindi in the final model, and 23 for Bengali). SOS and EOS token are useful for telling the model when to start and stop.

5.2 Training

The models were all trained for $10 - 15$ epochs with total running time 1 - 2 hrs each. optim.Adam module was used as the optimizer with learning rate at 0.001. Higher learning rates led to results being inconsistent, while lower learning rates increased the time complexity.

Further, the model training followed a **Teacher forcing** paradigm, where even if the decoder produces an incorrect output, it is often fed the correct output, so that it has a better chance of predicting the next word correctly, rather than the output it produced. When done correctly, Teacher forcing helps optimize the learning process for the model as otherwise the first few epochs would essentially be useless for the latter half of the sentence, since the model has gone off track.

5.3 HyperParameters

Parameter	**Value**
Length of Sentences (Hindi)	27
Length of Sentences (Bengali)	23
Embedding Size	256
GloVe Embedding Size	200
Number of Epochs	13
Batch Size	50
Hidden Size	512
Number of Layers	2
Learning Rate	0.001

Table 3: Model Hyper parameters

These hyper parameters were gotten through a mix of data analysis and experimentation. For example, the length of the sequences was gotten through looking at the average length of the corpus (14 for Bengali and 18 for Hindi). Experimenting with different hidden and batch sizes, these proved to be more efficient in faster training while not compromising accuracy.

6 Results

6.1 Validation Data

Model Name	**CHRF Score**	**ROUGE Score**	**BLEU Score**
GRU Model	-	-	< 0.01
LSTM Model	0.15	$\sim$0.20	0.02
LSTM Model + Attention	0.20	$\sim$0.25	0.05
LSTM Model + Attention + GloVe	0.21	$\sim$0.25	0.05

Table 4: Performance of Different Models on Validation Data

6.2 Test Data

Model Name	**CHRF Score**	**ROUGE Score**	**BLEU Score**
GRU Model	-	-	-
LSTM Model	-	-	-
LSTM Model + Attention	0.20	$\sim$0.25	0.05
LSTM Model + Attention + GloVe	0.22	0.24	0.05

Table 5: Performance of Different Models on Test Data

6.2.1 Explanation

The LSTM model was found to be superior to the GRU model for this problem of Machine Translation. This is due to the how LSTM model can retain both long and short term memory efficiently.

Attention Mechanism greatly improved the scores as well, because it allowed the models to focus on the important portions in the sentence. Finally, Glove embedding marginally optimized the entire process because it allowed a more efficient embedding of the data into the embedding vector space.

7 Error Analysis

Overall, the scores were not on the high side, however this is expected for Sentence Translation problems, because it is quite hard to produce a model which can predict sentences very correctly. The limited size of the corpus further limited the model.

The model still has some errors where it produces similar words which would be close to each other semantically, but doesn't get the actual meaning. This issue arises due to insufficient context. Additionally, some small errors in one word cascade makes the model go off track for the whole sentence, where it starts producing convoluted and logically invalid sentences.

Improved Attention Mechanisms, better RNN models, and training the model on a larger, more diverse datasets, are some of the ways we could improve the model.

There was one interesting observation, about the GRU model output, where it produced the same word repeated multiple times instead of any resemblance of a sentence structure.

8 Conclusion

Sentence Translation is a very critical problem in today's world. There already exist widely used google translate models for example, which are efficient and trained on huge sets of data. However, even such models always have scope for improvement.

My key finding was about how LSTM models are powerful at solving such Translation problems. Their long term and short term memory retention enables them to remember the context of the sentences and encode that correctly to produce meaningful output.

Attention Mechanism is really useful tool and improves performance by an significant amount. This improvement is due to the importance of selectively attending to critical information, especially for longer and more complex sentences.

However, there are still lots of possible improvements that could be done in the models, as listed above in the Error Analysis. Transformers could be introduced into the model architecture since they are very powerful in such problems, due to their Self Attention Mechanism, Parallelization and Long term Dependency handling.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[2] PyTorch Developers, "Pytorch tutorials." `https://pytorch.org/tutorials/`, 2023.