Competition: Hindi to English Machine Translation System

Saksham Gupta
170613
gupsak@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

The report gives an in-depth analysis of models used in the NMT Competition: Hindi to English translation as part of the CS779A course at IIT Kanpur. The paper presents two model architectures with in-depth analysis and tries to justify their shortcomings. It also illustrates some data pre-processing techniques to improve the prediction accuracy. Towards the end, we will see some recommended paths for greater accuracy.

1 Competition Result

Codalab Username: S_170613
Final leaderboard rank on the test set: 36
METEOR Score wrt to the best rank: 0.215
BLEU Score wrt to the best rank: 0.0334
Link to the CoLab/Kaggle notebook: https://colab.research.google.com/NMTCompNotebook

2 Problem Description

Machine translation (MT) is an end-to-end learning task for automatically converting input text into another language. Given a sequence of text in the source language, there is no unique translation to another language because of the human language's natural ambiguity and flexibility. Therefore, the challenge of automatic machine translation is perhaps one of the most difficult in artificial intelligence. The approach of using deep learning models for MT is called Neural Machine Translation (NMT). This paper presents two deep learning models to convert the text from Hindi to the English language. The train data consists of 102,322 sentence pairs. The paper uses only the PyTorch, SpaCy and Indicnlp library (no other modules or pre-trained models[1]) to build the Hindi to English NMT systems.

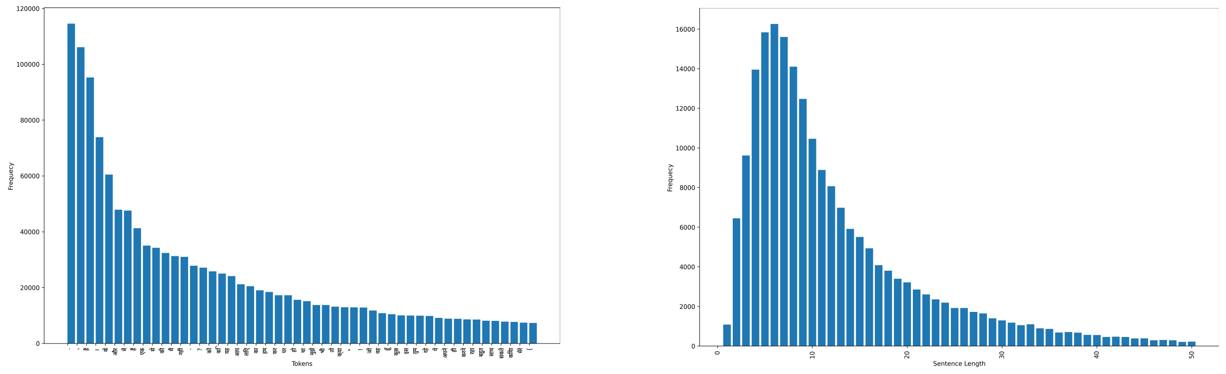
3 Data Analysis

The train data consists of 102,322 Hindi-English sentence pairs curated from publicly available sources [1]. The tokenization of Hindi sentences, using Indicnlp, shows there are around 45,000 tokens in the data, out of which approximately 52% tokens are present only once - highlighting the fact that handling rare words is a massive challenge for NMT systems. The most frequent token, excluding punctuation marks, is है. Sequence length is an essential characteristic of train data as many deep learning methods have difficulties in dealing with long term dependencies. In the given dataset, sequence length varies from 1 to 400: a significant variation. Seven is the most frequent sequence length. The number of sentences for different lengths follows a bell-shaped distribution, skewed to the right, in (Figure 1b). The tokenization of English sentences using SpaCy yields results similar to the Hindi text data. The following are some more critical insights from the data:

- Around 9000 Hindi sentences have English words. Example: यह Vika, एक फूल है.

[1]Only pre-trained Hindi word embeddings are used during Phase - 1 and 2.

- There are different special characters in some sentences like: "..." and "♪". Example:
  1. ♪I'M ON THE RUN WITH MY ELLIE ♪
  2. ♪The one behind my eyes... ♪



(a) Frequency of top 50 tokens

(b) Frequency of sentence lengths

Figure 1: Data analysis of Hindi sentences

4 Model Description

The paper presents two model architectures for Hindi to English translation. The base architecture for both models is a **sequence to sequence model** [2, 3, 4]. The Table 1 summarises the models for different phases of the competition, followed by a detailed description of the models.

Competition Phase	**Model Name**
Phase - 1	Seq2Seq
Phase - 2, 3	Seq2Seq with Attention
Final Phase	Seq2Seq with Attention

Table 1: Summary of models for the different phases of the competition

Seq2Seq Model: The input and output is a sequence of Hindi and English text tokens, respectively. The model has an Encoder and a Decoder, as shown in Figure 2. The Encoder consists of an Embedding and a GRU layer. The last hidden state acts as the **context vector** to the Decoder. The Decoder has a very similar structure - it starts with an Embedding and GRU layer. The GRU in the Decoder takes the context vector as the initial hidden state and a special token, <*sos*> (start of the sentence), as the initial input. A Dense layer follows the GRU, which has output units equal to English vocabulary size. The softmax function is applied to get the final output. During training, the model uses **Teaching Force**[2] . Teaching Force is a method where the model uses the ground truth, rather than the model's predicted output, as input at each timestamp. During inference, the model uses a **greedy search algorithm**, i.e. at each timestamp, the most probable word is the output, which is also the hidden state for the next time step. The objective function to train the model is the Negative Log-Likelihood loss (**NLL loss**) function. Assume the NLL loss function to train models for the remainder of the paper.

Seq2Seq with Attention Model: Figure 3 shown the complete architecture of the model. It uses the **Attention mechanism** [5, 6, 7, 8]. The Encoder architecture is the same as in the Seq2Seq model. The only difference is that it passes all the output states to the Decoder. The Decoder performs a critical extra step, in the beginning, before following the same path as the Seq2Seq model (a GRU

[2]Click here to read more about Teacher Forcing for RNNs.

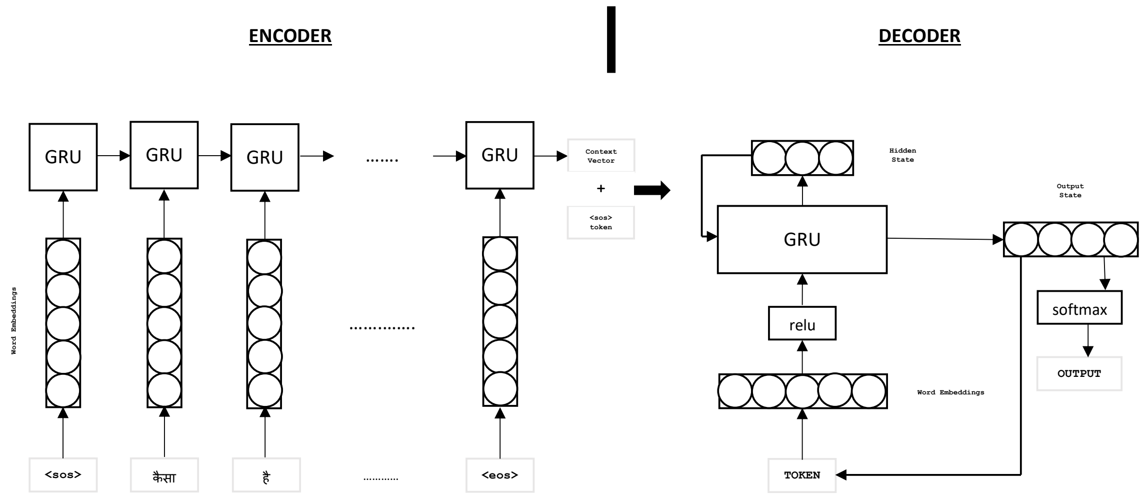


Figure 2: Seq2Seq Model architecture

layer, followed by a dense layer and softmax function) to calculate the output. The extra step is to calculate the *Atten_Vector* is as follows: The Decoder calculates the attention score for all Encoder output state using the previous hidden and input state as input for a Dense layer. Next, the Decoder multiplies the corresponding attention scores, after softmax, to create a weighted combination, called *Atten_Vector*, to help the Decoder focus on specific parts of the input sentence. Finally, the Decoder feeds the GRU layer with the previous hidden state and the output of another Dense layer (input is *Atten_Vector* concatenated with input state).

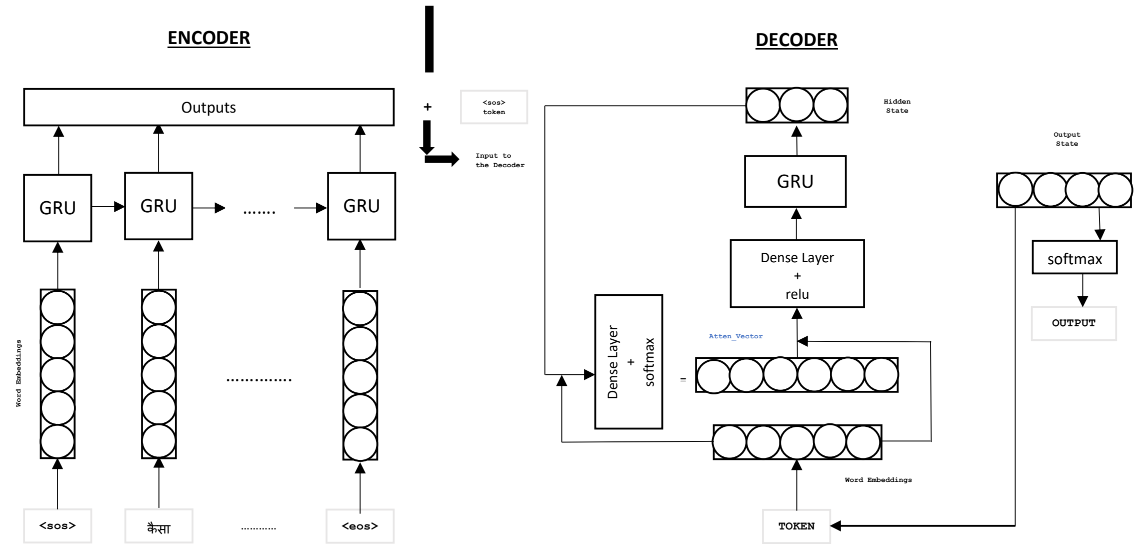


Figure 3: Seq2Seq with Attention Model architecture

5 Experiments

Throughout the competition, an in-built function of the Indicnlp library normalizes Hindi sentences, and it also eliminates *nuktas*, thus simplifying the data. All models use **Stochastic Gradient Descent** ((SGD) for training parameters throughout the competition. SGD is a gradient descent method to update model parameters using only a single data point per iteration. The Word Embeddings for English vocabulary is trainable and initialized randomly for all phases of the competition. All Hindi sentences start of *<sos>* token and end with another special token, *<eos>*, to signify end of sentence both in English and English text data.

During Phase - 1 and 2, the Encoder uses ”non-trainable” Word Embeddings from the **iNLTK library** [9]. The learning rate is equal to 0.01. The number of epochs is equal to five, which takes around six hours to train the Seq2Seq with Attention model.

During Phase - 3, the author applies some Pre-Processing methods on train data. As seen in section 3, there are many corrupted sentences pairs in the train data. The final train data, after data cleaning, has 81,382 training pairs after applying the following criteria on training data:

- Include a pair only if the number of tokens in the Hindi sentence is less than 75. It will eliminate around 4000 sentences but decreases memory and computational requirements significantly.
- Exclude a pair if the Hindi sentence has any English character, or there is some unique characters, which are common in slang language like 𝅘𝅥𝅮, “–” or “...”.

The vocabulary of Hindi and English consists of words whose frequency is greater than 1. Based on this criterion, the size of the vocabulary of Hindi and English is 12309 and 13215, respectively. The models match the rare words to a special token, *<ukn>*. The author also uses hyperparameter tuning to improve the performance of the Seq2Seq with Attention model. Table 2 summarizes the results of different experiments with hyperparameters. The final train data is divided into 3 part: training, validation and test with perchance of sentence pairs 85%, 7.5% and 7.5%, respectively. Train loss is the final loss on the training dataset after completing all epochs. Validation (Val) loss is the minimum loss on the Val dataset throughout the training process.

Epochs	**Embedding Size**	**Learning rate**	**Dataset loss**		
			Train	**Val**	**Test**
5	100	0.01	4.90	4.58	16.04
5	200	0.01	3.98	4.05	11.10
5	300	0.01	3.63	2.48	8.04
5	400	0.01	2.70	1.95	5.12
5	300	0.10	4.97	3.96	8.45
5	**300**	**0.05**	**2.89**	**1.65**	**4.70**
10	300	0.05	6.41	1.68	8.34

Table 2: Hyperparameter tuning for Seq2Seq with Attention model

6 Results

The results for different phases of the competition are listed in Table 3. Using the Attention mechanism improves model accuracy. It allows the Decoder to focus on specific parts of the input sentence and is especially useful for extracting long and short term dependencies. Data Pre-Processing helps as it provides the models with cleaner and refined data, allowing them to learn the language structure effectively.

Phase #	**Model Name**	**BLEU Score**	**METEOR Score**	**Rank**
1	Seq2Seq	0.0021	0.134	25
2	Seq2Seq	0.0096	0.111	-
2	Seq2Seq with Attention	0.0145	0.132	23
3	**Seq2Seq with Attention** [3]	**0.0191**	**0.157**	**21**
Final	Seq2Seq with Attention	0.0334	0.215	36

Table 3: Competition results

7 Error Analysis

- **Rare words**: Both the proposed models have difficulties dealing with rare words as all of them are replaced by a common token. Input sentences with *<ukn>* tokens are more prone to the erroneous output sequence. Figure 4 shows how the model prediction deviates from the target (using Attention matrix) when it encounters the *<ukn>* token, substitute for कैपोन, in the current input sentence. Using a lot more training data will definitely to reduce this issue.

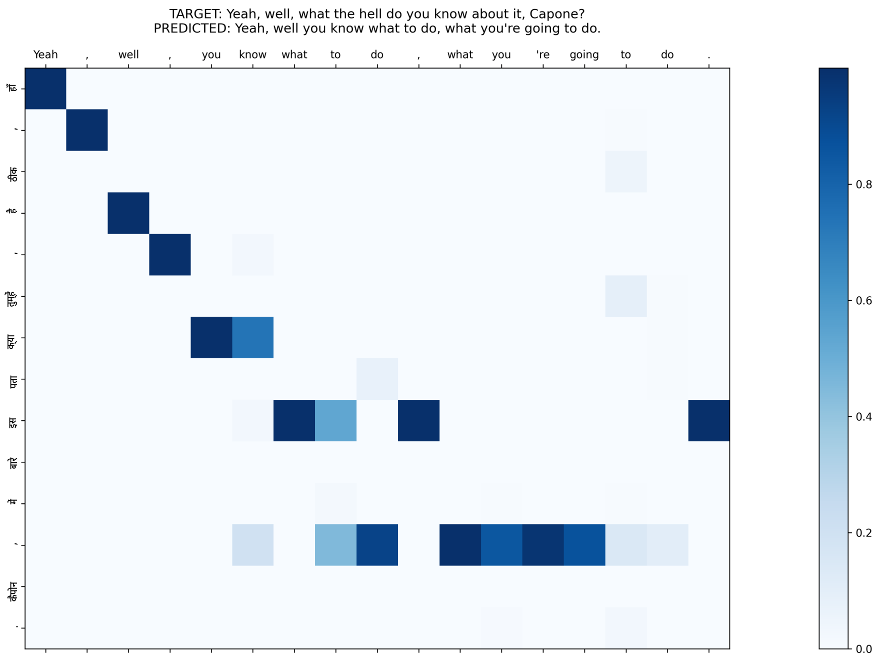


Figure 4: Attention matrix where the input sentence is:
हाँ, ठीक है, तुम्हें क्या पता इस बारे में, कैपोन.

- **Incorrect data annotation**: Example of wrong target sequence: Seq2Seq with Attention model correctly translates the sentence "ओह, मेरे भगवान." to "Oh, my God" but it will be penalized during training, as the target output is "Oh, my goodness." in the training dataset. Secondly, translation has no one correct answer, which may result in ambiguous train pairs like for the sentence "शर्म की बात है!" , the target is "Shame!". A more suitable translation would be "It is a shameful thing!". Capturing such variations is extremely difficult for deep learning models, in general.
- **Shallow models**: The models that the paper presents use only a single layer of GRU. Today, most of the state-of-the-art models use multiple layers of GRU/LSTMs or even multiple layers of Encoders-Decoders (Transformers). Therefore, using deeper models will improve the accuracy.

8 Conclusion

This work shows that shallow GRU-based Encoder-Decoder models with a limited vocabulary can perform Hindi to English translation with an acceptable level of accuracy. One shall be able to improve the accuracy using multiple layers of GRU/LSTM but at the expense of computational resources. It warns us that NMT is computationally expensive both in training and inference, and most NMT systems have difficulty with rare words. These issues have hindered the use of NMT in practical deployments and services, where both accuracy and speed are essential. Yet, the recent development of Transformers may provide Neural Machine Translations systems that are faster and accurate.

[3] Plus data Pre-Processing and hyperparameter tuning

References

[1] “Competition: Hindi to English Neural Machine Translation System,” Available at `https://docs.google.com/document/d/1p74wG-bECCgbpyq5x_x2QJrf5RSf9FnMLGSAiyUkHLo/edit`.

[2] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *CoRR*, vol. abs/1406.1078, 2014.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” *CoRR*, vol. abs/1409.3215, 2014.

[4] “Building a LSTM Encoder-Decoder using PyTorch to make Sequence-to-Sequence Predictions,”

[5] J. Alammar, “Visualizing a Neural Machine Translation Model (Mechanics of Seq2seq Models with Attention),” *Blog Post*.

[6] M. Luong, H. Pham, and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” *CoRR*, vol. abs/1508.04025, 2015.

[7] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *ICLR*, 2015.

[8] “NLP from Scratch: Translation with a Sequence to Sequence Network and Attention,” Available at `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`.

[9] G. Arora, “iNLTK: Natural language toolkit for indic languages,” in *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*, (Online), pp. 66–71, Association for Computational Linguistics, Nov. 2020.