CS779 Competition: Machine Translation System for India

Divyaksh Shukla
231110603
{divyakshs23}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

October 2, 2023

Abstract

Machine Translation using Gated-Recurrent Unit in a Sequence to Sequence architecture. I tested out 6 different configurations of GRU in Seq2Seq architecture, utilising Bidirectional GRU [1] and context vector to capture details of the source words. I scored an overall rank 18 in the competition with CHRF score 0.109 and rank 17, Rouge score 0.064 and rank 17 and BLEU score 0.001 and rank 16.

1 Competition Result

Codalab Username: D231110603
Final leaderboard rank on the test set: 18
charF++ Score wrt to the final rank: 0.109 (17)
ROGUE Score wrt to the final rank: 0.064 (17)
BLEU Score wrt to the final rank: 0.001 (16)

2 Problem Description

The problem statement was to create a machine translation system that can read sentences in English and predict output sentence from a set of Indian languages (Hindi, Bengali, Gujarati, Tamil, Telugu, Kannada, Malayalam). The dataset also contains sentence-pairs for English to Indian languages. The task is to create a translation system from scratch without using any pre-trained models and compete with the students of CS-779 Fall 2023 on a leaderboard hosted on Codalab. The problem organisers used CHRF, Rouge and BLEU scores to asses the outputs of trained models.

3 Data Analysis

Table 1 presents statistics about the lengths of sentences in each language. We shall keep English out of some measures as it is the source language and so it a part of each language-pair.

Figure 1 shows the distribution of sentences for each target language. We can see that number of sentences in Hindi is double that that present in Telugu.

Figure 2 shows that the length of sentences in each language is right-skewed and there is not much information available in sentences beyond the length of 150. We shall see more about this in 3 and 5.

Figure 3 Gives us an idea about the maximum length of sentences that we need to take. We can see that till about 175 we have 90% of the sentences.

Figure 4 gives an idea about the approximate size of the vocabulary for each language. Here the unique words were identified by splitting sentences based on "space" character. We can use the same strategy while modelling as well but then having such large vocab sizes can lead to slow training times. Additionally, creating a vocabulary by just taking words as token limits us during evaluation,

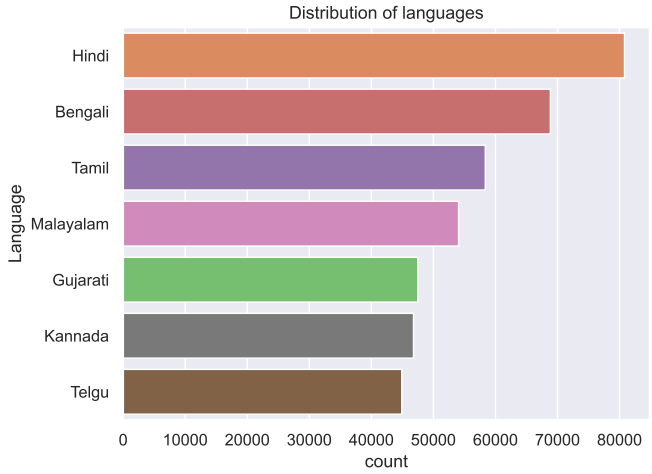

Figure 1: Distribution of languages in training data

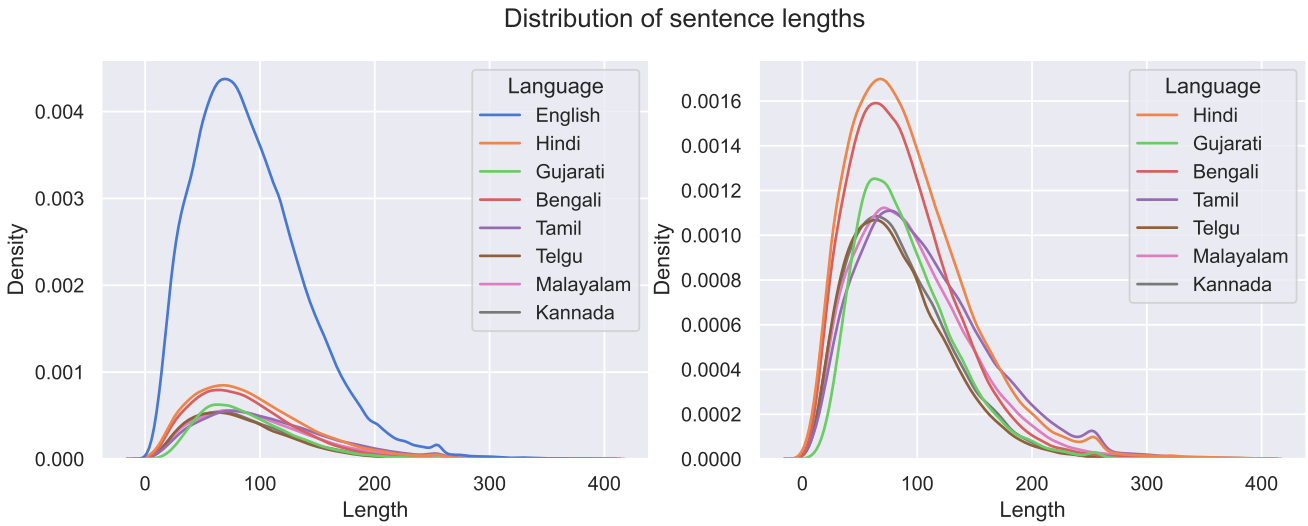

Figure 2: Distribution of sentence lengths in training data

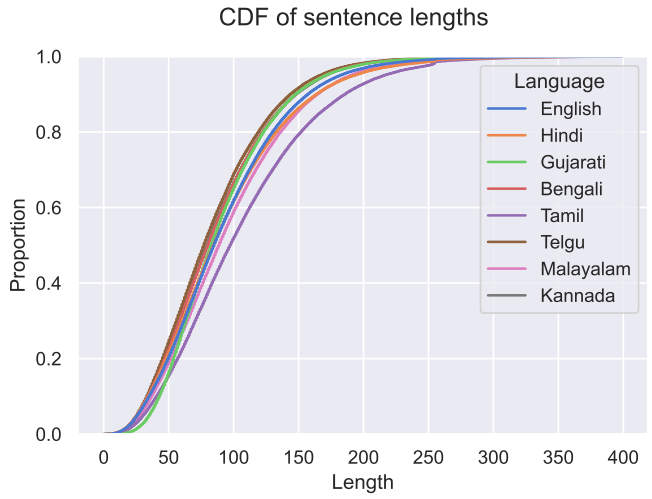

Figure 3: Cumulative Distribution of sentence lengths in training data

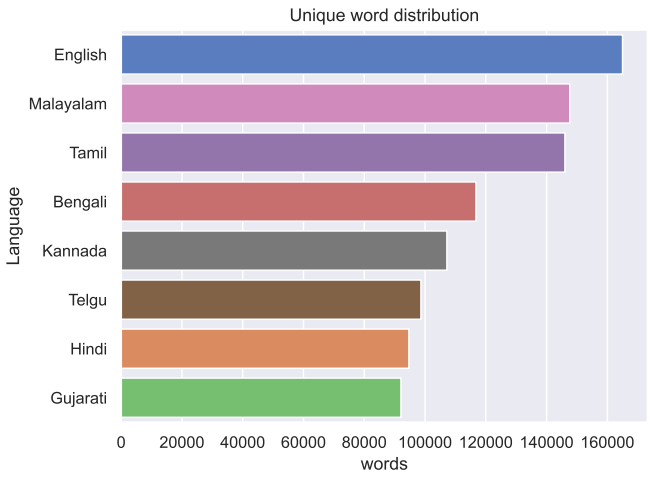

Figure 4: Distribution of words in the training data

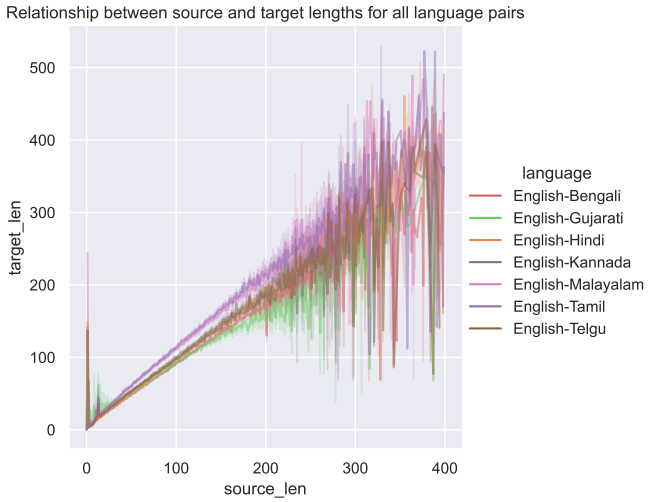

Figure 5: Relational plot between source and target sentence lengths for all language pairs

	English	Hindi	Gujarati	Bengali	Tamil	Telgu	Malayalam	Kannada
count	401246	80797	47482	68849	58361	44905	54057	46795
mean	92	94	90	86	107	84	96	86
std	53	55	43	46	58	45	52	46
min	0	1	0	2	3	0	2	0
25%	56	54	59	53	64	51	58	52
50%	85	85	83	80	97	77	89	79
75%	120	123	115	113	140	110	126	113
max	12732	1203	519	628	662	524	1133	662

Table 1: Statistics of sentence lengths in all languages

during which the source sentence may have out-of-vocabulary (OOV) words. We shall see a strategy to overcome the same below.

Figure 5 shows a line-plot comparing the lengths of source and target sentences in each language-pair. We can see an outlier at source length 0, the targets have a long length, with Malayalam having more that 200 words for the corresponding 0 length source. It is observed that the source lengths between 20 and 150 we are able to get a linear relationship between source and target lengths. For source lengths beyond 150 we can see large discrepancies between source length and target length. This could be due to lexical complexity in Indian languages, or can be simply attributed to incorrect or incoherent translations.

Thus, setting the length between 20 and 150 we are able to cover maximum number of sentences in the source and target languages while reducing possible noisy relationships. From Figure 5 we also notice and approximate 1:1 relationship between English sentences and Indian language sentences, this could be helpful during training and evaluation as we can safely set the number of words to decode by the decoder to be equal to the source length.

4 Model Description

I created 6 different models using Gated-Recurrent Units in Sequence to Sequence architecture. Throughout model creation and experiments there were some learnings and insights which led me to try out more complex model architectures. I started out simple so as to get quick results and get a feel for the dataset.

4.1 Model 1: Forward single-layered GRU

This is the first model I created to get a feel for the data and understand pytorch modelling with Recurrent neural networks. This is a simple Gated-Recurrent Unit with Embedding layers for both input to encoder and decoder. I used embedding dimensions of 256 units, 512 hidden dimensions for the encoder and decoder and limiting the vocab size to 30,000 in English and Indian languages. Additionally, I use a trivial space-separated tokenizer for all the languages.

I trained the same encoder for all language-pairs but kept on switching decoders for each language. The idea behind being that the encoder learns about the semantic structure of sentences in English, while the decoder learns about the structure of sentences in Indian languages. This way I just trained the encoder only on English-Hindi language pair and used the trained encoder to learn decoder weights for rest of the languages. This methodology is used for the following models as well.

4.2 Model 2: Forward single-layered GRU with spacy tokenizer

This model uses the same architecture as the above along with the use of spacy tokenizer. It is the default (non-pretrained) tokenizer from spacy for English and all Indian languages.

All the model including Model 1 use teacher forcing method to train the decoder. Although I did not parallelize the same during training as I wanted to add ways to remove teacher forcing during later

development. But, this led to slow training times.

4.3 Model 3: Bidirectional single-layered GRU with spacy tokenizer

Before jumping completely into the model architecture mentioned in [**?**] I wanted to test out Bidirectional GRUs independently. This is exactly what I did in Model 3. The architecture of model 3 is same as model 2 and uses spacy tokenizer.

4.4 Model 4: Bidirectional single-layered GRU with Byte-Pair-Encoding and context vector

Here I introduce the system of context vector to the decoder network. The context is taken as the last-output of the encoder and fed to the decoder GRU along with the embeddings of the target words. This model also has the same information bottleneck that was identified while creating attention mechanisms, but I wanted to see the same for myself. Additionally, I made use of byte-pair encoding to tokenize words and get their vocab ids. Here, I use 'tokenizer' package from huggingface but trained the model from scratch using the corpus from the data that is available to me.

4.5 Model 5: Bidirectional multilayered-layered GRU with Byte-Pair-Encoding and context vector

Here, I add more than 1 layer to the encoder and decoder blocks. Currently my code only supports having same number of encoder and decoder layers. Adding layers helps in capturing non-linear relationship between words and is a step towards deep architectures.

This is the submitted model for leaderboard ranking and zip file for code-submission.

4.6 Model 6: Bidirectional multilayered-layered GRU with space tokenizer and context vector

This also uses the same architecture as Model 5, but I reverted back to using trivial word tokenizer to test out model performance.

All the models above use greedy-search as a decoding strategy. Greedy search seems to give acceptable performance for decoding and is the fastest decoding strategy available. In future designs I plan to use top-k decoding along with beam search to further boost model performance on test set. The model submitted for leaderboard score is Model 5 which has 256 embedding dimensions for source and target tokens, 512 hidden dimensions in GRU, 2 layers of GRU in encoder and decoder with dropout=0.1. The model was trained on sentence-pairs having source length less than 35, while using byte-pair tokenisation strategy having less than 30,000 tokens in the source and target vocabulary. I used a batch-size of 32 while training and 256 while testing. Each language-pair encoder and decoder used around 20-25 minutes to train and 3 minutes to evaluate on the submission data.

5 Experiments

5.1 Data-Preprocessing

There is not much data-preprocessing required as Devnagri languages do not have any capital and small letters and do not have many character variations. I did convert English letters to lowercase but it did not affect the translation quality much. There is also instances of double brackets and double punctuations. This had affected earlier model architectures but did not affect later modelling architectures.

5.2 Tokenization

Model described above in Section 4 used 3 types of tokenizing strategies. First, I used a trivial tokenizer using whitespace. This led to a very large vocabulary not just in English but in Indian languages as well. This also had the disadvantage of not being able to include out-of-vocabulary words

for translation. (Although in 'Model 6' I switched back to trivial tokenization as I wanted to see the results of the same on deep architectures as well).

To include OOV tokens and reduce vocabulary size I switched to Byte-pair-encoding. The used huggingface tokenizer library to tokenize words in English and Indian languages, with the trivial tokenizer as pre-tokenization strategy for BPE. The model was trained from scratch using all the training data. The tokens were then saved into 'tokenizer' directory in the project folder.

5.3 Model training

All the models described in Section 4 used a Negative-Log-Likelihood loss function as the model were coded to output log-softmax probability scores. This is similar to using cross-entropy loss with logit outputs. Further, all the models used an Adam optimizer with 10^{-5} as learning rate and were trained for 10 epochs over the training set.

The training set was divided into batches of size 128 and 32 (only for Tamil as it somehow managed to consume entire GPU memory). The models took more than 2 hours to train for a language, so I trained the encoder only on 'English-Hindi' language pair and for the rest I froze training of encoder and trained new decoders for each target language. This improved complete modelling training time, but still I needed about 8 hours to train all models, save their weights and run it on test-set for submission.

A strategy I would like to apply in the future to speed up training time while using lower batch sizes (to not consume large amounts of GPU memory) would be to accumulate gradients for 4 or 8 batches and then run the optimizer to backpropogate gradients.

6 Results

'Model 5' (Section 4.5) led to the best modeling score in all of my submissions. It used Byte-Pair-Encoding along with a multi-layered stacked GRU. This achieved a BLEU score of 0.007, rouge score of 0.199 and chrf score of 0.137 with a total score of 0.343 on Codalab. The scores for the remaining models are mentioned in Table 2.

I believe this model achieved the best score is because it was using the context vector from the encoder, while decoding each translated word, BPE for capturing OOV words during evaluation stacked bidirectional GRUs to capture token semantics while reading tokens from left-to-right and right-to-left.

Model	BLEU	ROUGE	CHRF	Total
1	0.001	0.174	0.073	0.248
2	0.005	0.207	0.120	0.332
3	0.004	0.200	0.090	0.294
4	0.002	0.08	0.117	0.199
5	0.007	0.199	0.137	0.343
6	0.001	0.048	0.079	0.128

Table 2: Model Scores

7 Error Analysis

Models with vocabulary sizes between 30,000 and 50,000 seemed to perform better with byte-pair-encoding as the tokenization strategy. Using stacked Bidirectional GRUs gave a boost to performance. The lack of attention mechanism is evident from looking at the translated values of the model.

7.1 Improvements

The model can be further improved by using attention mechanism for getting better translations. Additionally, use of larger dimensions in the hidden layers and more GRU stacks can improve model

performance. Teacher forcing could be used more effectively during decoder stage to parallelize training and speed up training time. Also, teacher forcing can be reduced in later epochs to tune the model based on its own past outputs thereby making it more robust.

During data-analysis we see that there is a lot of noise in data for source length 0-10. These sentences can be avoided in future modelling implementations, along with sentences greater than length of 150 words.

8 Conclusion

Machine Translation is an interesting an challenging task at the same time. It led to many innovations in natural language processing and language modelling, like innovation of sequence to sequence modelling, attention mechanism and transformer architecture. Applying the same in an Indian setting led to a new set of challenges, absence of sophisticated tokenisation systems, lack of popular word-ontology, entity recognition and word-embeddings. The reported model used Bidirectional GRUs in a sequence to sequence modelling architecture with trainable embedding matrices for source and target language. Further improvements are possible by using Beam-search while decoding the translations and using transformer architecture to not just speed-up training but also make use of attention mechanisms more effectively.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.