Competition: Hindi to English Machine Translation System

Shruti Wasnik
20111062
gdshruti20@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

In this competition, various Seq2Seq models with encoder-decoder architecture have been experimented with to perform Hindi to English Translation using Neural Networks to get the best-performing model. An LSTM model got me 27th rank with a BLEU score of 0.0019 and a METEOR score of 0.159. In the 2nd phase, I tried a GRU model and got 9th rank with a BLEU score of 0.0347 and METEOR score of 0.200. I experimented with different hyper-parameters and modified pre-processing techniques in the 3rd phase and got 6th rank with a 0.0400 BLEU score and 0.231 METEOR score. I tried more models like with Bi-LSTM encoder, Bi-LSTM encoder and decoder, Bi-GRU encoder, Bi-LSTM encoder with Attention in the last phase, and the model with two-layered GRU gave the best performance on the test set with a BLEU score of 0.0609 and a METEOR score of 0.298 which got me 24th rank.

1 Competition Result

Codalab Username: S_20111062
Final leaderboard rank on the test set: 24
METEOR Score wrt to the best rank: 0.298
BLEU Score wrt to the best rank: 0.0609
Link to the CoLab/Kaggle notebook: `https://colab.research.google.com/drive/1id6sAlnutRX3iut9YSQjMqW-S`
`usp=sharing`

2 Problem Description

The competition required the implementation of Neural Machine Translation models to perform Hindi to English translation. We were needed to experiment with different models with various hyper-parameters. The motive behind this is to get the best-performing model, which can give a comparable leaderboard rank with a good BLEU and METEOR score in the final phase.

3 Data Analysis

The training data had a total of 102,322 Hindi-English sentence pairs which were taken from the following publicly available sources: `https://opus.nlpl.eu/` and `http://www.opensubtitles.org/`. Following are some points noted after the analysis of the given train and test data:

1. Many of the Hindi sentences contained some (or all) English words or unnecessary symbols like musical notes, Unicode characters, etc. There were approximately 5848 such Hindi sentences. Around 1538 Hindi sentences had Devanagiri numerals for representing numbers. There were some incomplete Hindi sentences as well.
2. Some English sentences had contracted verb forms like *I'll* instead of *I will*. Approximately 27,293 sentences used such short form representations. Some words were written as *b-a-b-e-e*, while some contained characters from a different language. Some sentences also had slang words like *uh ,umm, etc.*

3. There were some Hindi-English sentence pairs with incorrect English translations, while some sentences with opening braces (or quotes) but not closing braces (or quotes).
4. Most of the Hindi and English sentences had ≤ 50 words. There were only a few sentences with more words. Only one or two sentences had more than 400 words, so I kept the $max\_seq\_length = 400$ for the decoder, such that the decoder will predict at most 400 words.
5. The test set had less noise than the train set. There were total 24,102 Hindi sentences in test set with 45 sentences (approx) containing invalid characters like musical symbols, and 27 sentences (approx) having Devanagiri numerals. Unlike train set, there were no English words in any of the sentences in test set. Even the test set included some incomplete sentences.

Hindi sentence with English words, or invalid characters, and use of contracted verb forms in English sentence	**Hindi :** - यह Vika, एक फूल है. **English :** It's a flower, Vika.
Sentences with 0 words	**Hindi :** (? **English :** (?
Sentence pair with incorrect translation	**Hindi :** पी **English :** Like he says, you're lucky you're still in.
Use of slang words in English sentences	**Hindi :** - मैं पीता नहीं हूं. **English :** Do you have any wine or vodka? Umm...

Figure 1: So examples of inconsistencies in train set

4 Model Description

I tried variants of Seq2Seq encoder-decoder architecture in this competition.

4.1 Phase-1

Model-1: In an encoder-decoder architecture, the encoder uses one RNN for mapping the given input sequence into a fixed dimensionality vector known as context vector, and the other RNN in the decoder, conditioned on the input sequence, will take this context vector as input to predict the output sequence. In this phase, I first tried a basic two-layered Seq2Seq encoder-decoder architecture with LSTM. The model used `RMSProp` optimizer for the encoder and `Adam` for the decoder with `CrossEntropyLoss` function. Weights were randomly initialized before, but the predictions were not that good, so I initialized the model parameters with uniform distribution in the range `[-0.08,0.08]` as specified in [1]. The batch size 64 gave out *CUDA out of memory error*, so I used batch size of 32. The model with learning rate 0.0001 took much time to run, so a learning rate of `0.001` was used later. The predictions were still not good, so then I tried `Torchtext` package to use pre-defined classes like `Field, BucketIterator,` etc., which can ease and automate some tasks. The `Field` takes the tokenizer function as a parameter and creates a vocabulary from the tokens. `BucketIterator` can divide the data into batches according to the number of tokens in the source sentence, with each batch having sentences of similar length to reduce the padding. I also experimented with optimizers like `Adagrad, AdamW and Adamax,` for encoder, but the best performance was given by `RMSProp`.

Model-2: I then implemented the same model with the approach suggested in [1] of reversing the words in the input sequence, which, according to their experiment, improved the model's performance, but my model's performance did not improve even after 25 epochs. Thus, the LSTM model-1 got me a better rank and score in this phase.

4.2 Phase-2

Model-1: In this phase, I modified certain pre-processing techniques and replaced LSTM with GRU in both the phase-1 models. This model produced much better results than the phase-1 models, even though the hyper-parameters were same. But even with GRU, the model with reversed input gave worse performance than the normal input, which contradicts with the experiments stated in [1]. I also capitalized the first letter of the predicted sentence and added punctuations at the end (which I removed in further phases.).

4.3 Phase-3

Model-1: In this phase, the pre-defined classes of `Torchtext` package like `Field` and `BucketIterator` were removed. So I implemented these classes from scratch. The approach was first to create a vocabulary based on the tokens of the tokenized text to implement the functionality of the Field class. I created a class `Lang` that will create the vocabulary for each language object. Then the next task was to divide the training data into batches. I first tokenized the sentences, added `<sos>` and `<eos>` tokens at the beginning and end of the sentence respectively, and maintained a dictionary to store the token lists of both source and target sentences along with the number of tokens in each of them. This dictionary was then sorted based on the number of tokens in Hindi sentences to minimize the required padding. This sorted data was then divided into batches. If the batch contains sentences less than the specified batch size, some dummy sentences with `<unk>` tokens were appended in that batch. Each sentence of the batch was then padded according to the sentence with most tokens and converted into tensors. These batches were then shuffled to get better results.

Apart from all these, I tried `SGD` optimizer for encoder and `NLL Loss` function, but that didn't improve the performance, so I used `Adam` optimizer for both encoder and decoder with `CrossEntropyLoss` function. I tried changing the hidden size to 1024, but that gave me a `CUDA out of memory error`, so I tried hidden size 256 and embedding size 128, but that degraded the performance. So I kept the hidden size as 512 and the embedding size as 256. Then I tried increasing the number of layers to 3, but the model took a long time to train, and the convergence was significantly less, so I then tried four layers, but that again gave me `CUDA out of memory error`. So, I kept only two layers. The model got me 6th rank with a BLEU score of 0.400 and METEOR score of 0.231.

4.4 Final Phase

Model-1: In a bidirectional RNN, be it a GRU or LSTM, the input sequences are fed into the RNN in both forward and backward directions. For each layer, there is an RNN layer for each direction. The Bi-RNN can capture the context in both directions, considering the previous and the following words. Thus such models perform generally better than a unidirectional RNN. In this phase, I first tried a single-layered Bi-GRU encoder with a unidirectional decoder. Since only the encoder is bidirectional, the hidden layer and the output will be in both directions, i.e., we will have the context vector of both the directions, but the decoder needs only one. So I concatenated the forward and the backward context vectors and passed it through a linear layer to convert it into a form required by the unidirectional decoder. The same thing can be done with the output as well. Though Bi-GRU must have performed better, it did not produce good results.

Model-2: Next, I replaced GRU with LSTM in model-1 to test Seq2Seq architecture with Bi-LSTM encoder with a unidirectional decoder. Since LSTM returns three values: output, cell, and hidden, we need to concatenate cell state as well. So I concatenated the forward and backward cell state and passed it through a linear layer, which can then be given as an input to the unidirectional decoder. This Bi-LSTM encoder architecture performed much better than the one with a unidirectional LSTM encoder and the model with Bi-GRU encoder. However, it could not outperform the model with unidirectional GRU.This can be because two layers were used in GRU, while a single layer in Bi-LSTM.

Model-3: I then tried a model with both Bi-LSTM in both encoder and decoder. Since the decoder is also bidirectional, we did not need to concatenate the results of the encoder, I just adjusted the shapes of some parameters in the decoder. The performance of this model was almost similar to that of Bi-LSTM encoder with a unidirectional decoder.

Model-4: Next, I tried a Bi-LSTM encoder model with an attention mechanism. As stated in [2],

attention mechanism is used for both align and translate. Given an input sequence, the attention mechanism helps in focusing on relevant words or features, ignoring the irrelevant ones. The relevant information is then used for predicting the appropriate translated output sequence. This improves the model's performance. The model first calculates the attention scores (described in [2]), and then use them to distinguish relevant and irrelevant information.

With the same hyper-parameters as previous models, it gave me `CUDA out of memory error`, so I tried reducing the batch size to 16 and then 8, but I still got the same error. So then I tried reducing the hidden size to 256 and embedding size to 128. The model took longer, but got trained successfully, but the predictions were not as good as the GRU model. This can be because of significantly less hidden and embedding size and also only a single layer.

In all the models, model parameters were initialized with uniform distribution in the range `[-0.08,0.08]` as specified in [1].

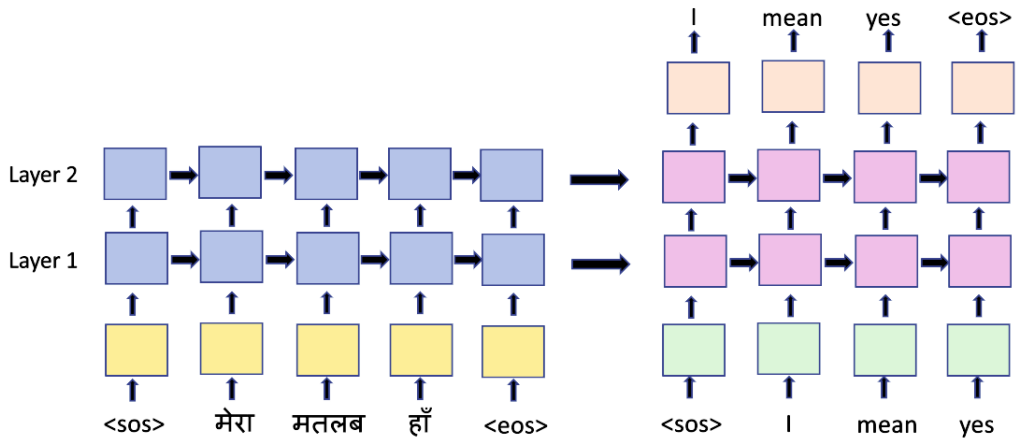

Figure 2: The figure represents the Seq2Seq architecture with two layers and unidirectional RNN. The same architecture has been used by the best performing model which uses GRU in encoder and decoder.

Out of all the models experimented in different phases, the two-layered model with uni-directional GRU in both encoder and decoder gave the best results on the test set. The model had two layers, and was trained with a batch size of 32, hidden size of 512 and embedding size of 256. It was trained for 20 epochs with a learning rate of 0.001. On trying a learning rate of 0.0001, it was taking longer to converge, but gave a comparable score on validation set. `Adam` optimizer in for both encoder and decoder with `CrossEntropyLoss` function was used, and took only 2-3 hrs training time.

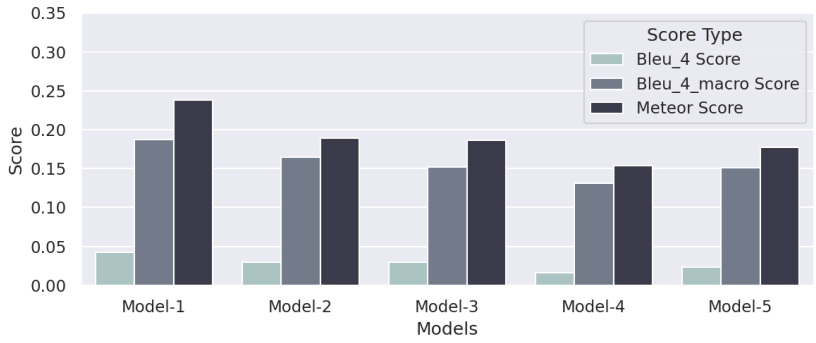

Figure 3: The figure depicts the scores of best five models tested on the dev set of phase-4.

`CrossEntropyLoss` function was used in all the models. Other loss functions like `NLL` (Negative Log Likelihood) function did not work properly with the used architectures. The models used in this competition used greedy decoding strategy.

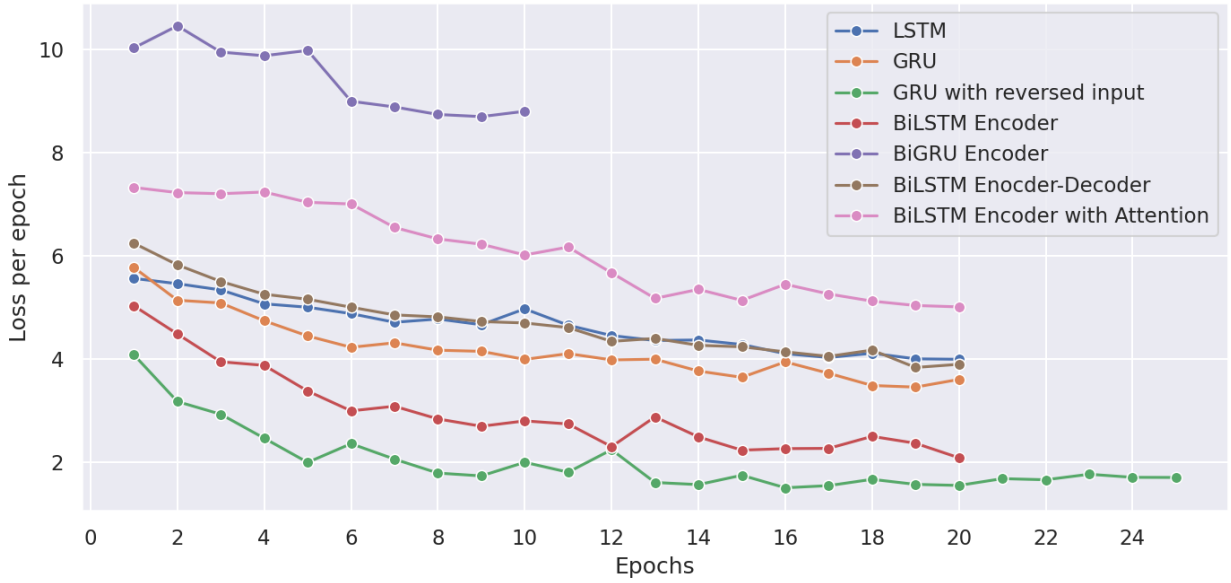

Figure 4: This figure shows how different models converged after each epoch.

5 Experiments

5.1 Data Pre-processing

For Hindi sentences: The Hindi sentences are first normalized using the `Devanagiri Normalizer` of `IndicNLP` with `remove_nuktas` set to `True`. Then the punctuations are removed from the normalized sentences. Since Hindi sentences include `Devanagiri numerals`, they are replaced by their `Wester Arabic numeral` forms. The sentence is then tokenized into tokens using IndicNLP tokenizer. If any of the tokens are invalid Hindi tokens, i.e., if any of them contain irrelevant characters like some symbols, Unicode representations, or English alphabets, they are replaced by the `<unk>` token. There are some English sentences as well in the Hindi sentences section. For such sentences, all the tokens would be treated as invalid Hindi tokens, so they would be replaced by `<unk>`. Such sentences (along with their translation) containing all `<unk>` tokens are being removed during the pre-processing because the model will not learn anything from such sentences.

For English sentences: The sentence is first converted into lower case. A dictionary is maintained which stores the contracted verb forms with their corresponding verb forms. So such verb forms are being replaced by their corresponding full forms according to that dictionary. Then after removing the punctuations, the sentence is tokenized using `IndicNLP` tokenizer. Like Hindi sentences, the English sentences with 0 tokens are also being removed.

5.2 Hyper-parameters and Training time

The following table states all the hyper-parameters which have been tried for each model architecture. The hyper-parameter which got me the best score were then used in the final models.

Hyper-parameters	LSTM	LSTM with reversed input	GRU	GRU with reversed input	Bi-GRU encoder	Bi-LSTM encoder	Bi-LSTM encoder-decoder	Bi-LSTM encoder with attention
Batch size	64, 32	32	32	32	32	32	32	32, 16, 8
Hidden size	512	512	1024, 512, 256	512	512	512	512	512, 256
Embedding size	256	256	256, 128	256	256	256	256	256, 128
Layers	2	2	2, 3	2	1	1	1	1
Learning rate	0.0001, 0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Encoder Optimizer	RMSProp, Adagrad, AdamW, Adamax	RMSProp	RMSProp, Adam , SGD	Adam	Adam	Adam	Adam	Adam
Decoder Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Loss Function	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss , NLL Loss	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
Epochs	20	25	20	20	10	20	20	20
Training Time	2-3hrs	2-3hrs	2-3hrs	2-3hrs	2-3hrs	2-3hrs	2-3hrs	3-5hrs

6 Results

Phase	Models	BLEU_4 Score	BLEU_4_macro Score	METEOR Score	Leaderboard Rank
Phase-1	LSTM	0.0019		0.159	27
Phase-1	LSTM with reversed input	0.0010		0.107	
Phase-2	GRU	0.0347	0.1557	0.200	9
Phase-3	GRU	0.0400	0.1862	0.231	6
Final phase	Bi-LSTM encoder	0.0462	0.2153	0.262	
Final phase	GRU	0.0608	0.2397	0.298	24

The model with two-layered unidirectional GRU outperformed all the other models and gave the best score on the test set with 24th rank and a BLEU score of 0.0608 and a METEOR score of 0.298 in the final phase. It was expected that the Bi-LSTM encoder model with attention should have produced the best results, but that did not happen. This might be because I had to reduce the values of hyper-parameters due to the resource constraints. The table describes the ranks and scores of models submitted in each phase.

Generally, bidirectional encoder models perform better than a simple unidirectional model. This was true with LSTM models since the single-layered model with Bi-LSTM performed better than the one with unidirectional. Attention models perform better in general, but here Bi-LSTM encoder with attention gave a comparable performance, but the score was still lesser. This can be because the hidden size and embedding size were less in the attention model and could not be increased due to the resource constraints. However, in the case of GRU, unidirectional two-layered GRU outperformed all the models. The reason for this can be the use of more layers and larger hidden and embedding sizes than other experimented models.

7 Error Analysis

1. The best score was given by two-layered unidirectional GRU, the attention model got me the next best score. The models with Bi-LSTM encoder with unidirectional decoder and Bi-LSTM encoder-decoder gave me the score similar to the attention model.
2. Since the full-forms replaced their corresponding contracted verb forms during the pre-processing, some predictions included incorrect helping verbs. This can be because some clitics are used for multiple helping verbs, like *'d* is used as a contracted verb form for both *would* and *had*, so it depends on the sentence which helping verb should replace it. However, during pre-processing, *'d* was replaced by *would*.
3. The predictions made by the models are not grammatically correct. Sometimes the sentences include repetition of words or incorrect use of braces, quotes, and other punctations, which were not removed during pre-processing. Removing symbols(or punctuations) may produce some meaningful phrases but has the disadvantage that the structure of actual translations will not match the predictions. Even after replacing the contracted verb forms, *'s* used for possessive nouns was not removed. So some predictions included incorrect usage of *'s*, like *you's*, *his's*, etc. However, these problems, including capitalization, were taken care of by the evaluation script at the test phase.
4. There can be multiple words representing the same thing, like *father*, *dad*, *daddy*, all mean the same. So the model may predict *father*, but the actual translation contains *dad*.

5. It was observed that the performance of the models degraded for longer sentences. After predicting certain words, the models kept repeating some words again and again. This might be because the train set had most of the sentences with ≤ 50 words and very few with more words, so the models were trained mostly for shorter sentences. This problem can be solved by including more longer sentences in the train set.
6. The Hindi sentences included numbers, say 9, so there is a possibility that the model may predict the numeral representation 9, but the actual translation may have *nine*.
7. Even in the true translations, there were some translations which do not correspond to its source Hindi sentence. The model does the same mistakes as it did on the dev set. Also, the true translation and contracted verb forms, but the model is not trained on that, so the predictions will contain only the full forms.

8 Conclusion

Various Seq2Seq models performing Neural Machine Translations from Hindi to English have been experimented with in this competition to get the best performing model. This competition helped us learn the basic structure of the NMT models (especially Seq2seq) and get in-depth knowledge of its working, hyper-parameters to be used, and the effect of hyper-parameters on the model's performance. I tried various models and evaluated them on dev sets each week to improve the performance. The leaderboard ranking helped us know where our model stands when compared to other students' models to try new models and improve the existing ones. The model with a unidirectional two-layered GRU gave the best performance on both the dev sets and the test set. The models may not be perfect because it might require some better pre-processing techniques and better resources. The future work includes improving the attention models, trying pre-trained embeddings, and trying Transformer-based models along with pre-trained transformers.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[2] K. C. . Y. B. Dzmitry Bahdanau, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473v7 [cs.CL]*, May 2016.

Tutorials and Blogs

1. PyTorch tutorials: `https://pytorch.org/tutorials/`
2. Jay Alammar's Blog : `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of`
3. https://github.com/lkulowski/LSTM_encoder_decoder
4. Test pre-processing tutorial : `https://colab.research.google.com/drive/1p3oGPcNdORw5_MDcufTDYWJhJt3XVPuC?usp=sharing`
5. Colah's Blog `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`