Competition: Hindi to English Machine Translation System

Shivam Aggarwal
20111058
shivama20@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

Neural Machine Translation (NMT) is a very popular approach to translating sentences from one language to another. In this competition, a translation model was developed for translating Hindi sentences into English. Different models were tried, including Sequence to sequence architectures and their variants, Transformers, etc. The transformer was the best performing model on the test dataset as it generated the Meteor Score of 0.158 and the BLEU score of 0.0158 on the test data. The model achieved rank 54 on the final leaderboard.

1 Competition Result

Codalab Username: S_20111058
Final leaderboard rank on the test set: 54
METEOR Score wrt to the best rank: 0.158
BLEU Score wrt to the best rank: 0.0158
Link to the CoLab/Kaggle notebook: https://colab.research.google.com/drive/1PREosf5udyGib11nJey_awnH_Vpq8bjS?usp=sharing

2 Problem Description

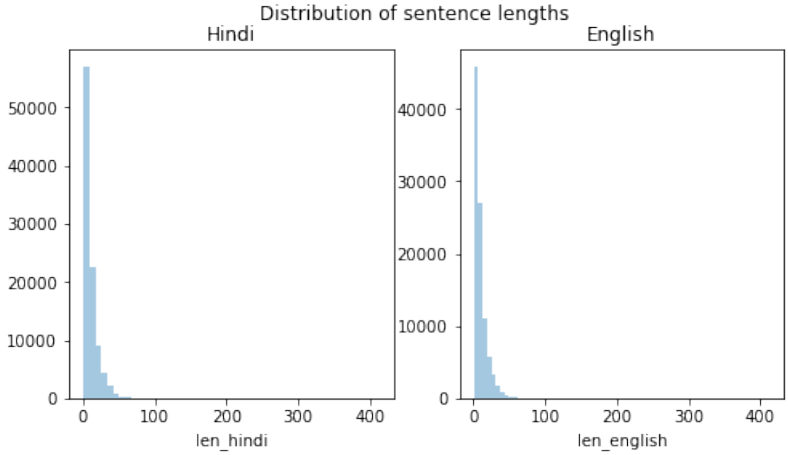
This competition aims to develop a neural machine translation model to translate Hindi sentences into English effectively. For machine translation, there are three approaches available in general. These are Rule-based translation, Statistical Machine translation, and Neural Machine Translation (NMT). NMT makes use of Deep Learning techniques, and it is the most famous approach for translation today. The dataset is provided to us to train the model, and we have to develop an efficient translation system.

3 Data Analysis

3.1 Train data

The train data provided us is a CSV file that contains Hindi sentences and their corresponding English translations. The dataset includes 102,322 Hindi-English pairs. On observing the given dataset, we can see some Hindi sentences with some English words. We found out that these kinds of sentences are almost 5% of the given corpus on analysis. The next significant property of the sentence is sentence length. The distribution of the sentence length for Hindi and English sentences is as follow:

Figure 1: Distribution of Sentence lengths for Train dataset

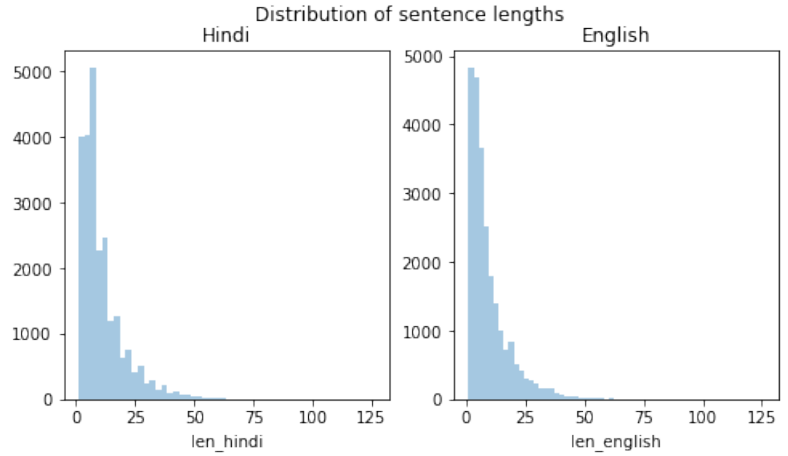


We can observe that almost all sentences have a length of less than 50. This matches general assumption also as we don't generally speak sentences that have so many words.

3.2 Test data

We are also provided the final test data with actual translation. It contains 24,101 Hindi sentences. On analysis, it is found that the test data does not have any null pair. Also, there is no noise in the test data (i.e., mixed sentences). The distribution of sentence lengths is almost similar to the training dataset, i.e., most of the sentences are having a length less than 50.

Figure 2: Distribution of Sentence lengths for Test data



4 Model Description

For neural machine translation, fundamental models used are sequence to sequence models [1] (also called seq2seq). So initially, during the first phase, I experimented with seq2seq models. These models contain two separate modules, i.e., Encoder and Decoder. The Encoder module encodes the information of a sentence into a context vector. The decoder module uses this information stored in the context vector for translation. Encoders and Decoders can contain different architectures. There are two very famous RNN's available named GRU's [2] and LSTM's [3]. I tried both of these in Encoder and Decoder architectures, and on the comparison, it was found out that GRU is faster to train than LSTM, but LSTM gives better results in general. After this, I tried to use Bi-directional LSTM's in the next phase, but they didn't perform as well as expected. Generally, it is observed that as the

sentence length increases, the performance of simple seq2seq models decreases because these models don't utilize the original context vector at each decoding step. For this purpose, there is a special technique named the Attention mechanism [4]. I tried to implement this in one of the models but could not do it.

Best Model: Transformer

In the final test phase of the competition, I used a basic Transformer model for machine translation. For the implementation, PyTorch's nn.transformer [1] was used. This implementation is based on the paper [5]. For transformer, we need to generate the embeddings for both source and target language. There are two kinds of embeddings required: word embeddings and the other is positional embeddings. Word embedding size is (vocabulary size, embedding size). Word embedding is a dense representation of the vocabulary. Positional embedding is used to encode the information of a word's position into the transformer as, generally, transformers are positional independent, and word order can affect the translation a lot. Positional embedding's shape is (max length, embedding size). To implement the embeddings, nn.Embedding [2] was used, which generates a kind of lookup table. The final embedding which the transformer will use will be the sum of both word embeddings and positional embeddings. For transformer, we also need to generate masks for both source and target languages. For source (Hindi), the mask will be of shape (batch size, max length), and it will contain '1' where there is pad token and '0' otherwise. For the target language (English), the mask will be in the form of a lower triangular matrix. The idea is that we want the model to look at only one word at each timestamp. After generating these embeddings and masks, we will pass this to the transformer model, and its output will be given to a fully connected layer to predict one of the words in the target language.

Now, to train this transformer model, 100 epochs were used. The training of the model was very fast as compared to seq2seq architecture. A batch size of 64 was used, and the Adam optimizer was used to optimize the model. To calculate the loss during the training, Cross-Entropy Loss was used. There can be some issues related to the Gradient Explosion problem, so for this, the PyTorch function 'nn.utils.clip_grad_norm_()' was used.

Finally, to predict the translation of a sentence, greedy decoding was used, which selects the most probable word from the target language. To predict the translation of a sentence, we first remove all the punctuations present in it and append the "start," "end," and "pad" tokens.

5 Experiments

1. **Data Pre-processing**: As we found out in the data analysis, given data contains some noise (mixed sentences, etc.), so we need to clean the data and also apply some basic pre-processing on the data to make it ready for further processing. First of all, we have filtered the data on the sentence length. We select a specific value of Max sentence length and only processed the sentences whose length was less than the Max length. Then we leave any pair whose 'Hindi' or 'English' sentence is having a null value. After getting the sentences that satisfy both of the above filters, we converted all 'English' sentences into lowercase characters. Then all punctuations were removed from both source and target sentences. Removing punctuations and lowercase conversion are commonly used practices to pre-process the data before passing it to the model, and so these approaches were used. After this, we need to create vocabularies for both source and target languages to map each unique word to a unique index. Vocabulary initially contains some special tokens like "start," "end," "pad" and "ukn" (for unknown words). Then we process each of the sentences and add new words into the corresponding vocabulary. While creating the vocabulary, also added the "start" and "end" tokens to the sentence and also added the required number of "pad" tokens to make the length of the sentence equal to the Max length as decided before. Then to pass the sentences into the model, it is needed to convert each sentence into a tensor using its index in the corresponding vocabulary.
2. **Training Procedure:** During different phases, different models were used, and in these models, some different training strategies were used. During the phase with the seq2seq model using GRU, I used Stochastic Gradient Descent Algorithm (SGD). One sentence was selected at random in each epoch, and calculated loss was backpropagated to update the model parameters. This

[1] `https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html`
[2] `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html`

resulted in very fast training, and 1 lakh epochs were completed in around 30-35 mins. Still, as expected, this was not converging, and even after these many iterations, train loss was about 4.5. Then in the first phase only, LSTM based model was used in place of GRU based model. Using SGD in this model, training becomes somewhat slower, but the loss was still hovering around 4-5. Then instead of SGD, I used the batch gradient descent algorithm with Adam optimizer with an initial learning rate of 0.01. This was trained for 50 epochs, and the loss became around 3.2. In the second phase, I experimented with different values of hyperparameters to converge the model. The architecture was almost similar to the first phase. The best model during phase 2 contained single-layer LSTM's in both Encoder and Decoder modules. Due to single-layer LSTM, training was faster as compared to the first phase. Adam optimizer was used with an initial learning rate of 0.005. For 70 epochs, it took around 5-6 hrs. In the final phase, I used the basic Transformer model. For optimization, Adam optimizer was used with an initial learning rate of 0.001. This model converged in about 100 epochs, and its training was also very fast. It completed 100 epochs in around 1 hr.

3. **Hyper-parameters for different models:** For the first, second, and third phase of the competition, seq2seq models were used. The hyper-parameters used in these models are shown in Table 1.

Hyper-parameter	Phase 1	Phase 2	Phase 3
Embedding size	300	400	400
Hidden size	1024	512	512
Number of layers	2	1	1
Dropout	0.5	0.5	0.5
Batch size	64	64	64

Table 1: Hyper-parameters for seq2seq models

To decide on these parameters, some primary value was chosen. For the embedding size, initially, 1024 was selected, then by some experiments, it was observed that 512 is giving better convergence. Two-three different values were tried out before finalizing a specific value. In the same way, other hyper-parameters were chosen. It was observed that a single-layer LSTM is working better than a multi-layer LSTM's. For batch-size value, when 128 was used, it was training faster, but convergence was not there. So for batch size, 64 was fixed. For dropout, generally, 0.5 is used as the default value, so the same was used.

In the final test phase of the competition, the Transformer model was used. For this, the hyper-parameters used are shown in Table 2.

Hyperparameter	Value
Embedding size	512
Encoder Layers	1
Decoder Layers	1
Num of heads	8
Dropout	0.10
Forward expansion	4

Table 2: Hyper-parameters for Transformer model

These values of Hyper-parameters are default as given in the official documentation of Pytorch Transformers. Different values were tried, but the model was not converging for the number of layers in Encoder and Decoder. For single layers in both Encoder and Decoder, the model converged, so these values were chosen.

6 Results

In each phase of the competition, a new dev test was released to evaluate the models. Results of different models on dev test are as following:

Phase	Bleu macro score	Bleu Score	Meteor Score	Rank on the Leaderboard
Phase 1	–	0.000	0.063	64
Phase 2	0.04128	0.00139	0.0607	48
Phase 3	0.0400	0.00169	0.0643	56

Table 3: Results of different models on dev sets

In the above table, we can observe that the best model on the dev test was during phase3 in both Meteor and Bleu. In phase 3, the basic seq2seq model was used with optimized hyper-parameters.

In the final test phase, two different variants of the transformer were used. These differ in terms of the max length of the sentence. For the first version, the max length of 10 was used, and for the second version max length of 15 was used.

Transformer version	Bleu macro score	Bleu Score	Meteor Score
Max-length 10 Transformer	0.1479	0.0158	0.158
Max-length 15 Transformer	0.152	0.0153	0.163

Table 4: Results of models on final test set

Scores of these models are shown in Table 4. From the table, we can observe that there is not much difference in the scores of both models but Max-length 10 Transformer performs slightly better. During the training also, for similar number of epochs, Max-length 10 transformer converged faster.

7 Error Analysis

I tried two categories of models in the competition. The first is the sequence to sequence models, and the second is the Transformers model. Out of these two, transformers outperformed seq2seq models. In seq2seq models, LSTM based models worked better than the GRU-based models. There is no specific reason for this behavior as both LSTM and GRU are widely used architectures for RNN. Also, it was observed that single-layer architecture worked better than the multi-layer architectures in both Encoders and Decoders. One more reason for not very good performance of the model can be manual batches creation. Since I didn't use any library for the batch creation, all sentences were given a fixed length. In some libraries, sentences are grouped according to various sentence lengths so that padding can be as minimum as possible. Out of the three kinds of optimizers, I tried (Adam, Adagrad, and SGD), Adam performs well in almost all the cases.

In the transformer model, we need to fix the sentence length because positional embeddings are generated according to this only. So when I tried different values of sentence length, model convergence got affected very much. For sentence length 10, model convergence was the best. Also, in the results generated by the transformer model, it can be observed that the model is making some mistakes, like it is repeating very high-frequency words in some of the output (like is, the, etc.). This can be because of the decoding strategy used. Using Beam search decoding may improve it further.

8 Conclusion

In the competition, different models were tried to perform the translation. These include seq2seq models with some variations, transformers, etc. In general, it was observed that seq2seq models do not perform very well. Using even a very basic transformer produces a far better result than the seq2seq models. Using some complex transformer architecture can further increase the performance.

As future work, some pre-trained word embeddings and pre-trained transformers can be used, as these will already be having knowledge of translation tasks and so can perform better.

References

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

[2] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600, IEEE, 2017.

[3] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[4] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

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

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[7] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.