Competition: Hindi to English Machine Translation System

Akshaj Bansal
180060
{akshaj}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

In this competition we were supposed to build a Neural Machine Translation (NMT) model for translating Hindi sentences to English sentences. I used Seq2Seq model to build the Hindi to English translator. A prominent architecture in NMT makes use of encoder-decoder architecture. The encoder and decoder which I implemented used GRU (Gated Recurrent Unit) and SGD optimizer was used for training. Since I am new to this field, I tried to implement quite a simple model. The model hyperparameters were tested by me manually and I used the best parameters I could find for the final test phase. The model was trained using the above and the final test phase evaluated using testing phase of the model resulted in 0.142 METEOR score and 0.0130 BLEU score.

1 Competition Result

Codalab Username: A_180060
Final leaderboard rank on the test set: 57
METEOR Score wrt to the best rank: 0.142
BLEU Score wrt to the best rank: 0.0130
Link to the CoLab/Kaggle notebook: Google Colab Notebook

2 Problem Description

In this competition we need to implement Neural Machine Translation (NMT) model to translate Hindi sentences into English sentences. The problem described some approaches in which we could implement such a model like Seq2Seq architecture, transformers etc. The main objective of this competition was to gain hands-on experience with Machine Learning models.
The problem was set in such a way so that we could learn by trying. We were supposed to build everything from scratch using only Pytorch. Any other external library was not allowed to be used. This was done so that we could understand the working of the model and get insight on as to how NMT models work since many students are new to this field.
Also, we were supposed to submit the results on CODALAB where we got to know how well our model performed and we could also see how other batchmates are doing. This resulted in a healthy competition where we learn from our mistakes and also tend to improve by getting inspired from the performance of other batchmates by looking at leaderboard rankings. These leaderboard rankings were based on METEOR and BLEU scores computed for our translated sentences.
The most important part is that the entire competition was partitioned into different phases along with a final evaluation phase which allowed us to work uniformly throughout the duration of competition (avoiding last day burdens) and we were also guided by TA's throughout this period in case we encountered hassles of any kind during the competition.

3 Data Analysis

We have been provided with 1,02,322 Hindi - English Sentence pairs for the train data set. This is given in csv format with 3 columns: Index, Hindi sentence, corresponding English sentence. From this data we could choose how to train the model. The problem statement intended us to randomly split data into train set and validation set in the ratio of 80:20. Using this trained model we were then supposed to translate the Hindi sentences in the test set into corresponding English sentences.
I tried to manually analyse some part of train and test data. In my opinion, according to length of sentences, train data and test data appeared somewhat different in the fact that the test data set contained larger sentences compared to train data set. Also, there is a possibility of encountering words in test data (like names of places, persons or some unknowns) which were never seen before in train data. The model may give non-satisfactory results at such points. Only if we could incorporate some worldly knowledge into the model regarding language structure, the model may perform better. This is most likely to happen in my case since I trained using only randomly sampled 12800 data-pairs (due to large training time if more data pairs were selected).

4 Model Description

I used a Seq2Seq architecture for all the phases implemented only using Pytorch. I took help from here [1, 2, 3, 4]. My final model constituted of encoder-decoder architecture using GRU (Gated Recurrent Unit). The encoder takes as input a sentence from Hindi language (source) and tries to map it into a representation like trying to extract relevant information and features from the sentence. This representation is then fed into the decoder which tries to decode it into the corresponding English sentence (target). This encoder-decoder architecture is mediated by SGD optimizer in the train phase.
In my final model, I first preprocess training data by tokenizing it using spacy module for English sentences and INDICNLP [5] for Hindi sentences. Then I create a vocabulary of English and Hindi sentences using class Language with default tokens SOS, EOS, UKN in each language. The preprocessing of entire train set resulted in 38349 distinct words of English and 46383 distinct words of Hindi. This is done for sentences whose maximum length (in terms of number of words) is 25 (for my model). The preprocessing phase required approximately 50-55 minutes of processing time.
Next part is the actual training phase of the model. This includes encoder-decoder architecture [1, 2]. The encoder uses GRU architecture with 1 layer. The number of layers can be changed which would result in stacked-RNN-GRU architecture as discussed in one of the lectures. The encoder only used uni-directional RNN structure. The decoder also uses GRU architecture with 1 layer. A linear mapping provided by Pytorch [6] has been used to get output from hidden vectors initially but afterwards a non-linear mapping ReLu [7] has been used. A greedy decoding strategy has been used to arrive at final output using LogSoftmax [8] function from nn.Module in Pytorch.
The train function first initialises the hidden state to all zeroes. The input is encoded using encoder. The encoded input is decoded using scheduled sampling with a teacher forcing ratio of 0.6. This usage enables to reduce cascading effect in case a wrong symbol is generated at some point. SGD [9] is used during training over randomly sampled 12800 data pairs for 10 epochs with a learning rate of 0.006. However, an early stopping after 4 epochs is used since loss started to increase afterwards. This part required approximately 2 hours of training time and we also plot the loss function for each epoch on a single plot after complete training of the model. The objective loss function for this model is NLL loss (negative log liklihood) in Pytorch library [10].
The final part constituted of testing phase of the model which is implemented using evaluate function. This part is more or less same as train function in implementation. It takes in a Hindi sentence and returns the corresponding translated English sentence. Initially, I store all the translated sentences in a string "translation" with each sentence separated by "\n" and finally all these statements are written into answer.txt file. This required 7-8 minutes.
The detailed description of each phase is as follows:

1. In phase 1, I used a fairly simple model which trains on data provided and generates the translated sentences for test data. This was a real challenge as I am new to this field and building a model for the first time gave me a hard time. I first tried to use LSTM but it failed. So I resorted to using GRU and the model ran successfully.

2. In phase 2, I improved the model by adding more epochs which was absent in phase 1 and also tried to restructure my code for better readability. I also tried to add validation phase to my model but I failed to implement it properly. I got some errors which I could not fix so I dropped this idea and submitted the result without validation phase.
3. In phase 3, we were supposed to get rid of pandas library. So first I tried to use bash script for reading the csv file but my effort failed because I used ',' as delimiter for sentences (csv is comma separated value format). The cause of failure was that some sentences have comma in them so the script breaks the sentence at that point. This resulted in wrong data for training. So then I decided to use in-built module of python: csv module [11] which made the job a lot easier.
4. In the final phase, I tried using different data size, epoch values and model parameters and tried to figure out which was giving a better performance. I chose the best parameters which I could find for the final test data. This was done manually by me. I also fixed a major error which was present in phase 3 due to which my model performance was quite poor in phase 3. I was reading second column of test data instead of third which is why I did not get the intended output. Moreover, I was restricted by colab environment which stores the logs only for a limited time. I tried to use large data set for training but after some point all the logs were deleted and before the train phase could complete, all information processed till that point was gone. So I had to submit the result using limited train set.

5 Experiments

1. For data pre-processing, I used INDICNLP [5] for Hindi sentences and spacy module for English sentences. I first tokenized the sentences. Then I created language for both Hindi and English and processed sentences to learn vocabulary from the train data. This seemed a nice approach since many tutorials used spacy for tokenizing and INDICNLP seemed good for Hindi sentences. The idea for using these came becuase they were mentioned in the competition document and after reading about them they appeared to be easily implementable.
2. During the training phase, I used SGD optimizer, a learning rate of 0.006, and 10 epochs with a break statement at 4 epochs for early stopping criteria, teacher forcing ratio of 0.6 for scheduled sampling during training. This is elaborately described in Model Description.
3. I failed at my attempt to implement validation phase in my model. So I resorted to manually try various hyperparameters and chose the best out of them. The various hyperparameters chosen by me included learning rate and epoch value for different size of train data set. I initially tried to model these for 12800 train data pairs and later tried to see these variations for larger size of train data sets. I concluded that as size of train data set increased, we need lower epoch value to arrive at optimal solution otherwise the loss increased and we get faulty results.

6 Results

In my model I did not use dev/validation set because I encountered some errors which I was unable to fix. So I removed that functionality altogether. Instead what I did was manually analyzed results for different hyperparameters and chose the one which in my opinion gave the best performance.
My leaderboard rankings along with METEOR and BLEU scores is shown in the following table:

Table 1: Test Phase Results

Phase	**BLEU Score**	**METEOR Score**	**Leaderboard Rank**
1	0.0001	0.079	48
2	0.0061	0.100	40
3	0.0000	0.007	66
4	0.0074	0.107	38
5(Final)	0.0130	0.142	57

For the above table, note that phase 4 had been cancelled but I mentioned it in the above table. Also note that the reason for such a low score in phase 3 was not the bad model but a foolish mistake made by me which I later corrected in phase 4. The mistake I made was that I read 2nd column(indices) from test data instead of 3rd column(Hindi statements) by mistake. Changing this mistake resulted in better performance of my model as is evident from phase 4 results.
I did not implement many models. Since I am new to this field, I decided to make a model and try to optimize it as much as I could. I think this worked quite well (except for phase 3 due to a mistake) since the model performance increased in each subsequent phase.

7 Error Analysis

The performance of the model is largely impacted by the set of hyperparameter values, training data size, similarity between training and test data, and effectiveness of machine (in terms of performance). We can also see that we have been evaluated using METEOR and BLEU scores but they tend to work differently as is evident from the leaderboard rankings corresponding to each of them. Since I have implemented quite a basic model, I cannot expect it to work with high performance. Moreover, there are some restrictions we face in terms of total time for training due to limited machine capabilities. As I analysed the performance of my model, I noticed that the model performed better on small sentences compared to large sentences. This may be due to the fact that I trained using maximum length of 25 and using only forward direction RNN. If bi-RNN was used, the model could look into future words and we might get a better performance.

8 Conclusion

The key findings during this competition include:

1. A small error in the code can result in complete failure in the model as happened with me during phase 3 due to wrong writing of column number.
2. As number of epochs increase, the loss begins to increase so an early stopping would result in better model performance. (may not always be true)
3. A larger learning rate gives errors due to inability in reaching optima while a small learning rate leads to very slow convergence. Thus we need to choose it wisely.
4. The number of train data pairs used during training phase have a huge impact on the model performance. Initially I used limited training pairs so my score was low. As I increased the number of training data pairs, my score increased.
5. I found an inverse relation between number of training data pairs and number of epochs. As we increase train data size, we need to decrease number of epochs to reach optima.
6. The use of teacher forcing ratio increased the performance of the model since it tries to do away with the wrongly predicted output by choosing between target output and predicted word with some probability for each (in my case 0.6 probability for the former).

The above key findings are in general true for any machine learning model. Please note that these findings may differ in case of different models. I used these findings to carefully structure my code and decide on the values of various hyperparameters to be used in my model.
The future course of action for my project would be to use validation set for learning model hyperparameters instead of doing so manually. Also, we can use transformer based models as they prove to be more efficient. We could also try out different mappings and functionalities in the encoder-decoder architecture. We could also implement attention mechanism in our model as they increase the performance of the model. A bi-directional RNN could be used instead of single forward direction RNN along with a multi layered structure. During training, we were restricted with limited data pairs for training and our machine capacity. So, given more train data and better machine performance, the model may prove to be more efficient and might give better results.

References

[1] S. Robertson, “NLP FROM SCRATCH: TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION.” https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html, 2017.

[2] A. Mishra, “Pytorch English to Hindi Translation using Attention.” https://ash1998.github.io/tech/2019/04/28/EnglishtoHindi-with-attention-network.html, 2019.

[3] “PYTORCH DOCUMENTATION.” https://pytorch.org/docs/stable/index.html, 2019.

[4] J. Johnson, “LEARNING PYTORCH WITH EXAMPLES.” https://pytorch.org/tutorials/beginner/pytorch_with_examples.html, 2017.

[5] A. Kunchukuttan, “The IndicNLP Library.” https://github.com/anoopkunchukuttan/indic_nlp_library/blob/master/docs/indicnlp.pdf, 2020.

[6] “Linear.” https://pytorch.org/docs/stable/generated/torch.nn.Linear.html, 2019.

[7] “Relu.” https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html, 2019.

[8] “LogSoftmax.” https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html, 2019.

[9] “SGD.” https://pytorch.org/docs/master/generated/torch.optim.SGD.html, 2019.

[10] “NLLLoss.” https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html, 2019.

[11] “csv — CSV File Reading and Writing.” https://docs.python.org/3/library/csv.html, 2021.