CS779 Competition: Machine Translation System for India

Aryan Srivastava
210204
{aryans21}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

This report is about the Machine Translation Challenge on Codalab. The competition is all about translating Indian languages (Bengali, Gujarati, Hindi, Kannada, Malayalam, Tamil and Telgu) to English using a machine learning model that is trained from scratch on training data and validated on validation data and finally tested on test data. We trained our Sequence to Sequence transformer-based model which is based on same transformer model explain in [1]

1 Competition Result

Codalab Username: A_210204
Final leaderboard rank on the test set: 22
charF++ Score wrt to the final rank: 0.131
ROGUE Score wrt to the final rank: 0.072
BLEU Score wrt to the final rank: 0.001

2 Problem Description

The problem may seem simple, but it is not. We need to translate sentences from Indian languages such as Bengali, Gujarati, Hindi, Kannada, Malayalam, Tamil, and Telgu to English language. The major challenge is that we do not understand all these languages, making it challenging to analyze whether our output is correct or not. Secondly, we cannot use any pre-trained model, so we have to train our model from scratch and set hyper-parameters for our specific task.

3 Data Analysis

Our model is trained on the provided dataset, comprising a cumulative total of 4,01,243 data points for all seven languages and a validation dataset of 3,15,966 for the same. At the time of training, we have a random validation split of 80-20 i.e. we use 80% for training and 20% for validation and Validation data is used as test data in the training phase.

Languages	**Training data**	**Validation data**	**Test data**
English-Bengali	68848	32919	19671
English-Gujarati	47482	39702	13567
English-Hindi	80797	11542	23085
English-Kannada	46794	46387	13370
English-Malayalam	54057	54109	15445
English-Tamil	58361	62446	16675
English-Telgu	44904	68861	12830

Table 1: Dataset distribution of all different languages

Our training data is in JSON format with a label of source for sentences of Indian language and target for English language, and we need to extract and convert it into CSV format for our task.

In the CSV file we use two labels for our dataset are source sentence and target sentence so it is easy to process rather than a JSON file.

With the help of Pytorch library, we tokenize our dataset and create a tuple of English words and Indian language words so that by data loader we can easily generate batches.

At the time of processing data, we have an error when we create tokens for vocabulary the word nan(not-a-number) in the dataset is treated as a float variable and we can't be iter so I removed those tokens.

4 Model Description

4.1 Model I tried but not Successfully implemented

4.1.1 Seq2Seq model

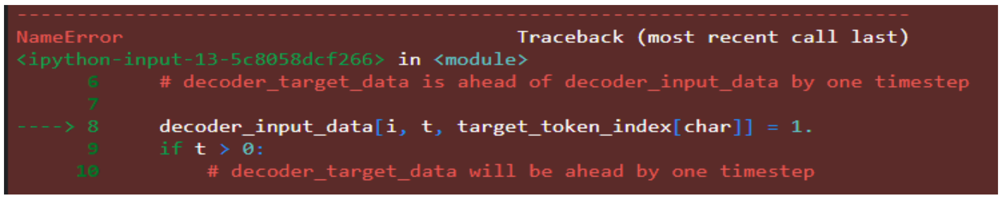
Firstly, I tried a seq2seq model explained in [2], but after training the model, I received output containing unknown tokens. I attempted to resolve this issue, but I was unsuccessful and I attached the Kaggle link for accessing that you can click here.

4.1.2 Transformers

Secondly, I tried transformers explained in [1] but unfortunately not able to match the batch size of source and target tensor and I attached the Kaggle link for accessing that you can click here.

4.1.3 LSTM model

Thirdly, I tried to use the LSTM model [3] but was not able to match the timestep of the decoded target data and decoded input data and the following image of the terminal also explaining the issue. I also attached the Kaggle link for accessing that you can click here.



```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-5c8058dcf266> in <module>
      6     # decoder_target_data is ahead of decoder_input_data by one timestep
      7
----> 8     decoder_input_data[i, t, target_token_index[char]] = 1.
      9     if t > 0:
     10         # decoder_target_data will be ahead by one timestep
```

Figure 1: Transformer based Seq2Seq model architecture from [4]

4.2 Successful Model

Finally, I tried the transformer-based seq2seq model explained in [5] and got the best results.

4.3 Why do we use transformers?

We use transformer because according to [1] :

1. Transformers incorporate a self-attention mechanism that enables the model to weigh the importance of each input token when generating an output token.
2. Transformers can use multi-head attention mechanisms to focus on different parts of the input sequence simultaneously so this allows the model to capture different types of information and relationships within the data that enhance its overall understanding.
3. Transformers incorporate positional encoding to provide information about the position of words in a sequence and this helps the model to understand the word order and sentence structure.

4.3.1 Architecture of Transformer based seq2seq model

Now, we talk about the architecture of the model as shown in Figure 2:

Hyperparametersfor our model

1. The size of the souce vocabulary is according to source data set.
2. The size of the target vocabulary is according to target dataset.
3. The size of token embeddings is 64.
4. The number of attention heads in the multi-head attention mechanism is 8.
5. The dimension of the feedforward neural network hidden layer in the Transformer is 64.
6. The batch size used during training is 128.
7. The number of layers in the encoder stack is 6.
8. The number of layers in the decoder stack is 6.
9. Number of EPOCHS is 16.
10. DEVICE is GPU.

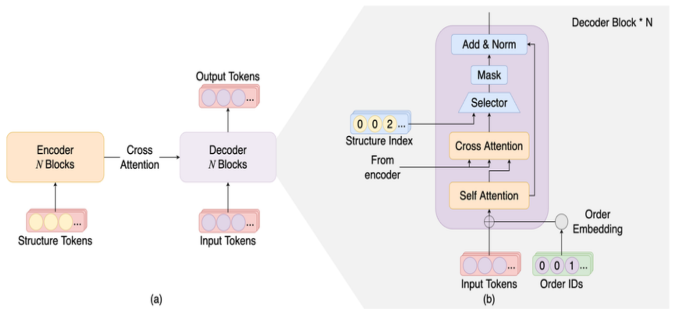


Figure 2: Transformer based Seq2Seq model architecture from [4]

Seq2SeqTransformer model is initialized on the above hyperparameters and this model is based on the transformer architecture, which is known for its effectiveness in handling sequence-to-sequence tasks.

We initialize the parameters using Xavier uniform initialization for each parameter that has more than one dimension. This choice is made to avoid issues such as vanishing and exploding gradients, as it helps in preserving variance and improving convergence.

We use the cross-entropy loss function while ignoring the padding index to ensure that padding tokens do not negatively influence the loss calculation.

Adam optimizer with specific hyperparameters, such as a learning rate(0.0001), beta values (0.9 and 0.98), and an epsilon value(eps = 1e-9), is essential for effective and stable training of our model.

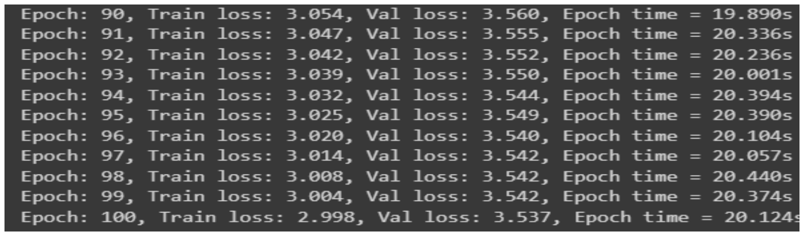
In our code, the motivation for using a greedy algorithm in sequence generation lies in its simplicity, speed, and low computational demands and our test data for any of seven Indian languages is not more than 23.1k.

5 Experiments

First of all, I made the dataset iterable and we tokenized source and target language sentences and build vocabularies for both languages based on the training data and handled special tokens and default indices in the vocabulary such as for padding and unknown special characters.

The hyperparameter for my final model is discussed earlier but we will talk about the epochs For example, we are talking about translating from Telgu to English in Figure 3.

We are using the same optimizer, which is Adam, and setting the learning rate to 0.0001 for all of my different models. However, I trained each model for a varying number of epochs: 1, 2, 5, 10, 20, 50



```
Epoch: 90, Train loss: 3.054, Val loss: 3.560, Epoch time = 19.890s
Epoch: 91, Train loss: 3.047, Val loss: 3.555, Epoch time = 20.336s
Epoch: 92, Train loss: 3.042, Val loss: 3.552, Epoch time = 20.236s
Epoch: 93, Train loss: 3.039, Val loss: 3.550, Epoch time = 20.001s
Epoch: 94, Train loss: 3.032, Val loss: 3.544, Epoch time = 20.394s
Epoch: 95, Train loss: 3.025, Val loss: 3.549, Epoch time = 20.390s
Epoch: 96, Train loss: 3.020, Val loss: 3.540, Epoch time = 20.104s
Epoch: 97, Train loss: 3.014, Val loss: 3.542, Epoch time = 20.057s
Epoch: 98, Train loss: 3.008, Val loss: 3.542, Epoch time = 20.440s
Epoch: 99, Train loss: 3.004, Val loss: 3.542, Epoch time = 20.374s
Epoch: 100, Train loss: 2.998, Val loss: 3.537, Epoch time = 20.124s
```

Figure 3: Transformer based Seq2Seq model architecture

Models	**decoder layer**	**encoder layer**	number of head
transformer	6	6	8
transformer seq2seq	6	6	8

Table 2: different hyperparameter in different models

and 100. The training time for each epoch depends on the language we are training on and typically ranges from seconds 15 to 35 seconds, with an average of 25 seconds in each epoch.

Our hyperparameters, such as the device, epochs, and batch size, are consistent across all models. The variations in hyperparameters are detailed in Table 2.

6 Results

Transformer-based Seq2Seq is my best performing model because in other models i am not able to resolve the error and my output have major unknown tokens rather than translation.
Table 3 and 4 is describing my rank and score on codalab.

Note: I trained the same model for all seven Indian Languages having some changes in tokenizations to convert float tokens in other Indian languages to string and the rest of the code is the same. I uploaded code for Telgu to English translation but we changed it at the time of extracting data from the JSON file to other languages too.

7 Error Analysis

In the Seq2Seq model, there is an error during training where the output file returns start tokens instead of translations and in the Transformer and LSTM model, I am unable to resolve a runtime error, so I am unable to obtain any output.

In the Transformer-based Seq2Seq model, the output is a translation, but it is not grammatically correct for every language. Additionally, there is a runtime error in the output when a float or integer value is encountered. To address this issue, we convert the token sentence to a string, but this can reduce the accuracy because the model treats the entire sentence as a single token, which is unknown to it. Consequently, the output contains unknown tokens..

For language like Hindi collab crashes for 8 heads and 6 layer of transformer encoding

Data set	**Based on chrf score**	**Based on rouge score**	**Based on bleu score**	**Leaderboard**
dev	20	19	14	14
test	21	23	21	22

Table 3: Ranks of Transformer-based seq2seq model

Data set	**chrf score**	**rouge score**	**bleu score**
dev	0.067	0.153	0.021
test	0.131	0.072	0.001

Table 4: Score of Transformer-based seq2seq model

8 Conclusion

This report focuses on a machine translation competition, where the goal is to develop a translation model from scratch using provided training data and evaluate its performance on test data. The evaluation is based on the Blue, CHRF, and Rouge scores.

Our findings indicate that the Transformer-based Seq2Seq model achieved the best results on the test data. However, we acknowledge the presence of an issue in our model where in output special tokens are coming. We are actively working to resolve this issue and improve the accuracy of the model's translations.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[3] G. Tiwari, A. Sharma, A. Sahotra, and R. Kapoor, "English-hindi neural machine translation-lstm seq2seq and convs2s," in *2020 International Conference on Communication and Signal Processing (ICCSP)*, pp. 871–875, 2020.

[4] C.-P. Tan, A. Su, and Y.-H. Yang, "Melody infilling with user-provided structural context," 10 2022.

[5] M. I. Azad, R. Rajabi, and A. Estebsari, "Sequence-to-sequence model with transformer-based attention mechanism and temporal pooling for non-intrusive load monitoring," 2023.