# CS779 Competition: Machine Translation System for India

Shubham Devang Patel
210709
devang21@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

**Abstract**

The second phase of the Machine Translation Competition of CS779 required participants to develop a single model to translate text from seven Indian Languages to English. I experimented with GRUs, LSTMs, Attention mechanisms and Transformers. Post-processing of generated sentences by capitalisation and joining of space-separated numbers gave a significant improvement of 0.047 BLEU. My best-performing model was a Transformer followed by post-processing of the generated sentences. It secured second place and the following scores on the test set: charF++ 0.439, ROUGE 0.453 and BLEU 0.170.

## 1 Competition Result

**Codalab Username:** S_210709
**Final leaderboard rank on the test set:** 2
**charF++ Score wrt to the final rank:** 0.439
**ROUGE Score wrt to the final rank:** 0.453
**BLEU Score wrt to the final rank:** 0.170

## 2 Problem Description

The Machine Translation Competition of CS779 was about developing models to translate text between English and seven Indian Languages. The seven Indian Languages were Bengali, Gujarati, Hindi, Kannada, Malayalam, Tamil and Telugu, which are among the most popular languages spoken across India. The first phase was on translating from English to Indian Languages, while the second phase required the reverse.
The second phase consisted of an additional challenge apart from the constraints on using pre-trained components and Pytorch for implementation. In the first phase, participants were allowed to use separate models for each language pair, but for the second phase, the participants were supposed to develop a single model for translating from all seven Indian Languages to English. This was because the target language was the same (English) for each language pair. This made the task even more challenging.

## 3 Data Analysis

The dataset provided for training consists of several sentences in Indian Languages, along with their translations in English. The Data set is organised into seven language pairs. Each language pair has several source-target sentence pairs, consisting of the text in the Indian Language and its translation in English.
**Sample**
Source: मधुमेह के मरीजों में आँख के पर्दे की खराबी होने की संभावना बनी रहती है ।
Target: The chance of defect of retina of eye remains in patients of diabetes .

For both the training and the testing data, the distribution of sentence pairs across languages is depicted in Figure 1. The data provided is significantly higher for Hindi and Bengali than others. This similarity between the train and test data could be one of the reasons for the same model giving similar scores on both.
The additional data provided in the Hindi-English corpus is that in a few sentences, they have provided more than one translation for terminology terms. One is the exact translation, and the other is the transliteration of the given English word. For example:
और **((परीक्षण केस | टेस्ट केस | test cases))** को इनपुट फ़ाइलों में संग्रहित किया जाना चाहिए जिसे इनपुट पुनर्निर्देशन द्वारा पढ़ा जा सकता है ।

For certain sentences in the corpus, the language has wrongly been tagged, while for some others, the source and target sentences have been inter-changed. The corpus also consists of specific pairs which have correct language labels but have wrongly been translated. Table 1 depicts the extent of mis-labelling across various languages.

| Labels | Bengali | Gujarati | Hindi | Kannada | Malayalam | Tamil | Telgu |
|---|---|---|---|---|---|---|---|
| **Correct** | 68719 | 47467 | 80448 | 46741 | 53223 | 58313 | 44800 |
| **Swapped** | 0 | 0 | 0 | 0 | 779 | 0 | 0 |
| **Wrong** | 129 | 15 | 349 | 53 | 55 | 48 | 104 |
| **Total** | 68848 | 47482 | 80797 | 46794 | 54057 | 58361 | 44904 |

Table 1: Labelling of Train Data

As studied in class, corpus for languages tends to follow Zipf's law as depicted in Figure 2 for the training corpus of the English Language. Zipf's law states that the frequency of the word in a corpus is inversely proportional to its rank in the frequency table.

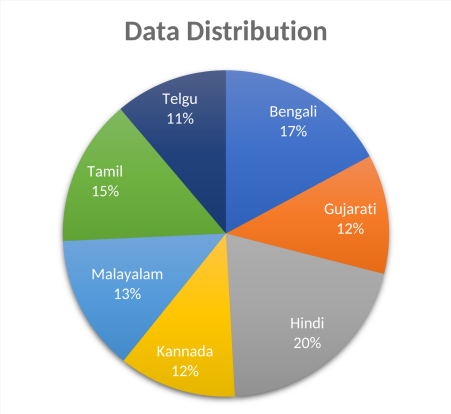


Figure 1: Corpus Across Languages

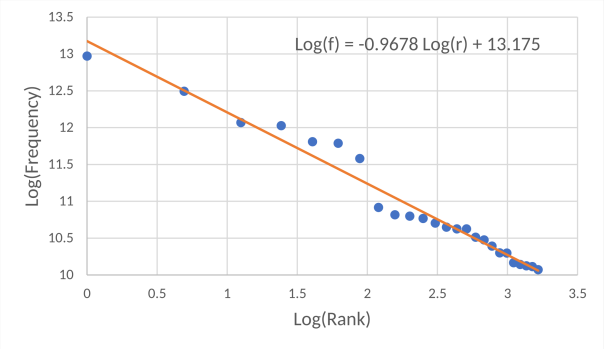


Figure 2: Zipf's Law for English

# 4 Model Description

## 4.1 Baseline

In the first phase, I experimented with techniques inclined towards the Statistical Machine Translation domain. The method I developed was based on learning common word embeddings for both languages and then performing a word-by-word translation. To set a baseline for the second phase, I utilised the same architecture as first-phase and used AI4Bharat word embeddings[1].

[1] https://github.com/AI4Bharat/indicnlp_corpus

## 4.2 Seq2Seq Architecture

The Sequence-to-Sequence Architecture is one of the popular architectures for machine translation. It consists of two components: the encoder and the decoder. The encoder takes the source sentence as input and outputs a representation vector of this sentence. This representation vector is passed to the decoder, which, using this, outputs the translated sentence in the target language. Various models can be used for encoders and decoders. Since Recurrent Neural Networks (RNNs) did not work well for me in the first phase, in the second phase, I have experimented with more advanced models.

## 4.3 GRUs and LSTMs

To improve the memory capabilities of vanilla RNN, Long Short Term Memory (LSTMs) use gating mechanisms and two hidden states: short-term and long-term memory. The gating mechanism helps in having dynamic control over the extent to which a word updates the hidden state. Gated Recurrent Units (GRUs) are based on a similar concept but simpler architecture. To analyse which performs better under which circumstances, for different parameter and data settings, I trained both models. They did not face the issue of predicting the same word, but it seemed difficult to achieve better performance beyond a limit.

## 4.4 Attention Mechanism

The following paper [1] introduces the method of attention mechanism for improving machine translation. Along with the previous word, the decoder receives a weighted average of the encoder's steps' outputs. The weight is calculated using an attention function measuring the importance of each encoder output with the previous hidden state. I have used the attention function described in the paper, which is a linear layer followed by an activation function. For LSTMs and GRUs, the predictions made by the model improved. Also, the training time for both speeded up by nearly 10%. The reason for this was something I was not able to understand precisely. After experimenting with attention, I decided to move on to one of the most powerful architectures.

## 4.5 Transformers

The following paper [2] introduces the transformer architecture built on multi-headed self-attention. Self-attention also focuses on words within the same sequence, and multi-headed refers to having multiple such units for capturing diverse representation. The encoder and decoder layers consist of attention layers, fully connected layers and residual connections. Positional encodings provide a positional representation of words. In the decoder, masking prevents attention over the following words. These make the transformer much more parallelisable and powerful. For transformers, the training time was nearly 10-15% faster per epoch as compared to GRUs and LSTMs.

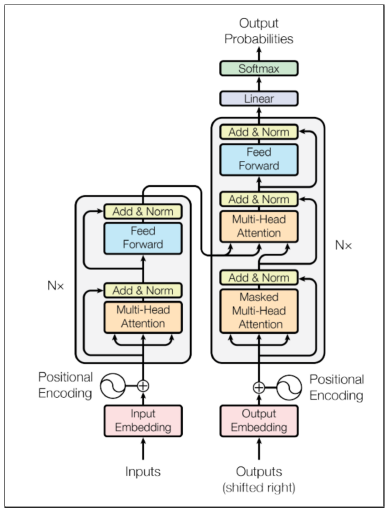


Figure 3: Transformer Architecture [2]

## 4.6 Post-Processing Module

Since I had employed lower casing to reduce Vocabulary size, the model outputted words in lowercase. The following paper [3] shows that an improvement of nearly 0.100 BLEU can be achieved by properly casing outputs by machine translation models. Since, in English, the first word in a sentence is always capitalised, I did this in post-processing. Various other methods of case restoration have been discussed

in [4], out of which I employed an approach similar to [5]. The words that occur in capitalised form in more than THRESH% of places in the training corpus have also been capitalised during the post-processing. Since the model had learnt individual digits, space-separated numbers were outputted in the predictions. These were also joined together during post-processing. All of these methods helped to achieve significant improvements in scores. Figure 4 depicts a sample application of these methods.

**Original:** the diameter of the stupa is 3 6 . 6 0 metres and height 1 6 . 4 6 metres .

**Numerical:** the diameter of the stupa is 36.60 metres and height 16.46 metres .

**First-Word:** The diameter of the stupa is 3 6 . 6 0 metres and height 1 6 . 4 6 metres .

**Corpus-Based:** the diameter of the Stupa is 3 6 . 6 0 metres and height 1 6 . 4 6 metres .

Figure 4: Post-Processing Methods

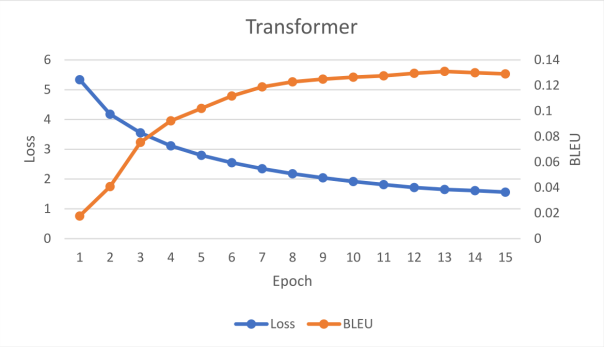


Figure 5: Train Loss and Val BLEU vs Epochs

### 4.7 Final Model

My final model was a transformer followed by post-processing. The source vocabulary comprised the most frequent 25,000 * 7 = 175,000 words in the source corpus. The target vocabulary consisted of all the words (75,000 words). The model parameters were used as initialised in the following tutorial[2]. Three layers for the encoder and three for the decoder were used with eight heads. The embedding size and feed-forward network dimension were both kept at 512. Adam Optimiser, with a learning rate of 1e-4 and batch size of 100, along with Cross Entropy Loss, was used. The model was trained for 15 epochs with a test-train split of 99-5 on the data to ensure that not too much time (nearly five minutes) was spent calculating the BLEU score. Possibly due to a large vocabulary size, the inference time for test data using the greedy strategy was three and a half hours, so I could not experiment with beam search. Figure 5 depicts how rapidly it learned. In only two epochs, it crossed my phase-1 best score of 0.047 BLEU.

## 5 Experiments

### 5.1 Data Pre-Processing

The dataset consisted of sentences with wrong language labels. To eliminate such sentences, I developed a rule-based language classifier that predicts the language based on the language of most sentence characters. Using this, I first filtered out wrongly labelled sentences. For the sentence-pairs that were reverse-labelled, I rectified and used them. I also filtered out pairs consisting of zero-length sentences. To remove wrongly translated sentences of excessive length, I dropped the top one percentile of sentences by length. For a reduction in vocabulary size, I employed lower casing, which caused a reduction in target vocabulary by nearly 30,000. To ensure correct learning of numbers, I employed tokenisation on numerical and punctuation in not only English but also all the Indian Languages.

### 5.2 Model Related

#### 5.2.1 Original Implementation

For the first phase, I implemented all the components, including RNNs, from scratch. Utilising the implementation, I experimented with various model parameters but got output as a model with exploding parameters or the model predicting only one word. To simplify the model's task, I also experimented

[2] `https://pytorch.org/tutorials/beginner/translation_transformer.html`

with English-to-English translation, which also didn't work out well. In the end, I realised that there was some issue with my implementation, after which I referred to the following[3].

| Parameter | Values |
|---|---|
| Model | RNN, GRU, LSTM |
| Word Embedding | One-Hot, 100, 300, 100 Glove (Pre-Trained)[4] |
| Output Layer | Linear, Neural-Network (ReLU, Sigmoid, Tanh) (Number of Layers) |
| Other | Hidden Size, Learning Rate |

Table 2: Original Implementation Experimentation

### 5.2.2 GRUs and LSTMs

Using shorter sentences for training the model resulted in faster training and loss reduction, but the model only predicted short sentences, which caused the model to be penalised by brevity. Due to this, for training the final models, I utilised the entire data. To reduce the training time, I tried using the Sparse-Adam optimiser developed for sparse updates, but I did not achieve significant speed up in training with it. So, I used Adam Optimiser, with a learning rate of 1e-3. On profiling of my code, **loss.backward()** was the step that was taking the most amount of time - about 0.5s per call. I suspected that this was because even for sparse updates, the whole embedding matrix would be being updated. To improve that, I built my embedding class, which stored embeddings as a dictionary of vectors instead of a matrix, but contrary to expectations, this resulted in a doubling of the training time.

| Data %ile | Time/Epoch |
|---|---|
| 10 | 10 - 15 sec |
| 50 | 4 - 5 min |
| 99 | 35 - 45 min |

| Parameter | Experimented |
|---|---|
| Optimiser | Adam, Sparse-Adam |
| Vocabulary | Torch-Embedding, Self |

Table 3: GRU & LSTM Experimentation

On smaller batch sizes, the training loss would not decrease after a few epochs and saturate, so I used the maximum possible batch size. Increasing the hidden size gave the model more memory to store words, which assisted it in predicting longer sentences, so I used the maximum possible hidden size. To further improve this, I also decreased the word embedding dimension to 100 to increase the hidden_size:embedding_size ratio. Table 4 depicts the effect of varying the hidden size.

| Model | Hidden Size | BELU |
|---|---|---|
| GRU | 128 | 0.014 |
| GRU | 1024 | 0.020 |
| LSTM | 128 | 0.011 |
| LSTM | 1024 | 0.020 |

Table 4: Effect of Hidden Size

For all the cases, I also ran experiments for both GRU and LSTM to compare how each of them performed in different circumstances. I also experimented with an attention mechanism, which not only improved the predictions but also speeded up the training by 10%. Changing the vocabulary size

[3] `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`
[4] `https://nlp.stanford.edu/projects/glove/`

did not have any significant impact on the training time of the model, so for both source and target languages, I used the entire vocabulary. This resulted in a source vocabulary of nearly 650,000 and a target vocabulary 75,000.

### 5.2.3 Transformers

Due to GPU constraints, it was not possible to accommodate the entire vocabulary for both the source and target languages, so I employed a vocabulary of 25,000 x 7 = 175,000 most frequent words for the source and the whole vocabulary of 75,000 for the target. As an initial run, I first ran a transformer model using 50% of the data and observed great results, so I proceeded with training it on the entire data. After training the transformer, I realised that while calculating the scores, the predictions of numbers were wrongly being penalised. This was because the model outputted space-separated digits, which were treated as different tokens by the scoring script. To address this issue and this issue of being penalised due to lowercase words, I explored various post-processing methods.

## 6 Results

For the training phase, out of all the models I experimented with, transformers not only performed the best but also produced far better results as compared to any other model. My best-performing model secured 5$^{th}$ place on the train-phase leaderboard.

| Model | Data% | Hidden Size | chrF++ | ROUGE | BLEU |
|---|---|---|---|---|---|
| Word-by-Word | 100 | - | 0.334 | 0.301 | 0.032 |
| GRU | 10 | 128 | 0.173 | 0.169 | 0.020 |
| GRU | 99 | 128 | 0.167 | 0.209 | 0.008 |
| GRU | 50 | 128 | 0.190 | 0.203 | 0.014 |
| LSTM | 50 | 128 | 0.189 | 0.198 | 0.011 |
| GRU | 50 | 1024 | 0.199 | 0.214 | 0.020 |
| LSTM | 50 | 1024 | 0.199 | 0.214 | 0.020 |
| GRU + Attn | 50 | 512 | 0.212 | 0.234 | 0.026 |
| LSTM + Attn | 50 | 512 | 0.203 | 0.230 | 0.018 |
| LSTM + Attn | 99 | 512 | 0.231 | 0.259 | 0.031 |
| GRU + Attn | 99 | 512 | 0.222 | 0.247 | 0.024 |
| Transformer | 99 | - | 0.382 | 0.349 | 0.101 |
| Transformer | 99 | - | **0.408** | **0.363** | **0.120** |

Table 5: Results - Training Phase

For the testing phase, post-processing of the outputs produced by the transformer greatly improved the results. Seeing such great improvements, I tried to analyse the impact of different post-processing methods on the score. The best-performing model was a transformer, followed by all the post-processing. It secured 2$^{nd}$ place on the test-phase leaderboard.

| Model Details | chrF++ | ROUGE | BLEU |
|---|---|---|---|
| Transformer | 0.412 | 0.365 | 0.123 |
| Transformer + Numerical Joining | 0.431 | 0.437 | 0.144 |
| Transformer + First Letter Capitalisation | 0.430 | 0.430 | 0.145 |
| Transformer + Corpus Based Capitalisation(80%) | **0.441** | 0.445 | 0.151 |
| Transformer + Full Post Processing(80%) | 0.439 | 0.453 | **0.170** |
| **Transformer + Full Post Processing(50%)** | 0.439 | **0.454** | **0.170** |

Table 6: Results - Testing Phase

# 7 Error Analysis

## 7.1 GRUs, LSTMs and Attention

I found out that for similar training times, GRUs outperformed LSTMs on smaller-length sentences. However, as the length of sentences increased, GRUs were unable to capture the long-term dependencies despite training for a sufficiently long time and performed poorer as compared to LSTMs. Using shorter sentences, although speeding up the training time, caused the model to predict shorter sentences. An issue with using the entire dataset was that it took too much time to reach the loss minimum. For both GRUs and LSTMS, adding attention improved their performance.

| Data Percentile | Attention | GRU | LSTM |
|---|---|---|---|
| 50 | x | 0.014 | 0.011 |
| 50 | ✓ | 0.026 | 0.018 |
| 99 | ✓ | 0.024 | 0.031 |

Table 7: GRUs vs LSTMs - BLEU Score

## 7.2 Post Processing

In Table 8, we can observe the effect of various post-processing methods on the BLEU score for identical translations. The method of numerical joining improved the model's performance because the model had correctly learnt to predict numbers, but the scorer was wrongly scoring them since each digit was treated as different tokens. Corpus-based capitalisation has caused more significant improvement as compared to first-letter capitalisation. A possible reason for this is that not always the first word predicted by the model is the correct word, and sentences can consist of more than one proper noun, which has to be capitalised. Extending the threshold to 50% has not caused any observable improvement because, during the 80% threshold, there were around 36,000 words, which went to 40,000 at 50%. Also, these newly added words were not always capitalised in the original corpus.

| Post-Processing | BLEU |
|---|---|
| Numerical Joining | + 0.021 |
| First Letter Capitalisation | + 0.022 |
| Corpus-Based Capitalisation(80%) | + 0.028 |
| All (80%) | + 0.047 |
| All (50%) | + 0.047 |

Table 8: Post-Processing Analysis

## 7.3 Final Model

**Good Translations**
Source: स्तूप का व्यास 36.60 मीटर और ऊँचाई 16.46 मीटर है ।
Prediction: The diameter of the Stupa is 36.60 metres and height 16.46 metres .
**Words out of Source Vocabulary**
Source: रथयात्रा के समय मौसीबाड़ी में विशाल मेला लगता है ।
Prediction: Huge fair is held in the month of July .
**Long Sentences**
Source: जब एक प्रकोप का पता चलता है, तो संक्रमण की रोकथाम और नियंत्रण समिति को सूचित किया जाना चाहिए और एक प्रकोप टीम का गठन किया जाना चाहिए।
Prediction: When a outbreak finds out , prevention and control agency should be informed to the prevention and control agency should be formed .

For normal-length sentences consisting of commonly used words, the model has performed quite well.

In cases when the source sentence consists of words out of vocabulary, the model is only able to retain the overall context of the statement and is not able to translate it properly. This shortcoming can be addressed by having a larger source vocabulary and transferring tensors between GPU and CPU to address GPU memory constraints. Also, for longer sentences, the quality of translations was found to be a bit poorer as compared to shorter ones. To address this issue, tweaking the existing architecture could help by increasing the number of encoder-decoder layers.

# 8 Conclusion

Although GRUs, LSTMs and Attention Mechanisms perform well at the translation task, Transformers outperformed all of them by a huge margin. Transformer is a really powerful model. Better tuning of various hyper-parameters of the transformer model can be explored to better its performance. It was also noticed that the post-processing of the outputs by the model had a significant impact on the scores achieved by the model, and the impact of various post-processing methods was analysed. To further improve the quality of translation, cased models can be experimented with, and more sophisticated methods of case restoration can be employed.

# References


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

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[3] T. Etchegoyhen and H. Gete, "To case or not to case: Evaluating casing methods for neural machine translation," in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, (Marseille, France), pp. 3752–3760, European Language Resources Association, May 2020.

[4] V. Păiş and D. Tufiş, "Capitalization and punctuation restoration: a survey," *Artificial Intelligence Review*, vol. 55, pp. 1681–1722, jul 2021.

[5] A. Mikheev, "A knowledge-free method for capitalized word disambiguation," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, (College Park, Maryland, USA), pp. 159–166, Association for Computational Linguistics, June 1999.