# CS779 Competition: Machine Translation System for India

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

**Abstract**

The first phase of the CS779 Machine Translation Competition required models to translate English text into seven different Indian languages. The models I developed were based on bringing the word embeddings of both languages to the same vector space for word-to-word translation followed by a word-order correction mechanism. I also experimented with RNNs but was unsuccessful in achieving good results. On the test set, my best-performing model has a rank of 10. The scores it has achieved are charF++: 0.318, ROGUE: 0.297 and BLEU: 0.047.

## 1 Competition Result

**Codalab Username:** S_210709
**Final leaderboard rank on the test set:** 10
**charF++ Score wrt to the final rank:** 0.318
**ROGUE Score wrt to the final rank:** 0.297
**BLEU Score wrt to the final rank:** 0.047

## 2 Problem Description

The competition aims to provide a hands-on experience in natural language processing. The objective has been fulfilled by developing models to translate text between English and seven Indian languages. The first phase requires models for translation from English to Indian Languages. The seven Indian languages for which the task has to be done are Bengali, Gujarati, Hindi, Kannada, Malayalam, Tamil and Telugu.
There were certain restrictions on the models and their development. The models had to be coded using PyTorch without utilising any higher-level libraries. The usage of pre-trained models or components was not permitted. Only pre-trained static word embeddings were allowed to be used. Apart from this, we were free to use anything.
There was a leaderboard on Codalab where the various predictions by students were scored based on chrF++ score, ROUGE score and BLEU score. chrF++ measures the character n-grams overlaps between the source and the target while ROUGE and BLEU do similar for the word-n-grams.

## 3 Data Analysis

The dataset provided for training consists of several sentences in English along with their translations in Indian Languages. The Data set is organised into seven language pairs. Each language pair has several source-target pairs, each with a unique ID, consisting of the text in English and its translation in the Indian Language. An example for the English-Hindi pair:
**source**: cancel everything on my calendar
**target**: मेरे कैलेंडर पर सब कुछ रद्द करें

The additional data provided in the English-Hindi corpus is that in a few sentences, they have provided

more than one translation for terminology terms. There are nearly 4300 such terms. One is the exact translation, and the other is the transliteration of the given English word. For example:
**source**: The air conditioning filters should be cleaned periodically, and fans that can spread airborne pathogens should be avoided in high-risk areas.
**target**: एयर कंडीशनिंग फ़िल्टर को और पंखों को समय-समय पर साफ किया जाना चाहिए जिससे उच्च-जोखिम वाले क्षेत्रों में **((वायुजनित | एयरबोर्न |Airborne))** रोगजनकों न फैल पाएँ|

The dataset also consists of a few incorrect translations. In the following case, the translation mentions an underwater castle, while nothing is mentioned in the source. Since I was unable to come up with any idea to identify these in the dataset, they were kept as it is. Example for English-Gujarati:
**source**: You can also participate in water - Sport competitions organised by the West Zone Cultural Centre and Daman administration.
**target**:પાણીની અંદર પહેલા ચાર માળ અને તેનો છેલ્લો માળ પાણીની બહાર ધરાવતો ભવ્ય મહેલ જોઇ શકાય છે.

Following are the statistics of the dataset. For the number of words, only space-separated sequences of characters have been considered as tokens. Src_Ch (Characters in the source) and Src_W (Words in the source) and similarly Tgt_Ch and Tgt_W represent the target.

| Language | Total Sentences | Src_Ch | Tgt_Ch | Src_W | Tgt_W |
|---|---|---|---|---|---|
| **Bengali** | 68849 | 95.48 | 86.95 | 16.82 | 14.27 |
| **Gujarati** | 47482 | 100.09 | 90.45 | 17.74 | 15.94 |
| **Hindi** | 80797 | 96.31 | 94.07 | 17.07 | 18.93 |
| **Kannada** | 46795 | 85.6 | 86.6 | 15.6 | 11.5 |
| **Malayalam** | 54057 | 85.23 | 96.46 | 15.30 | 10.51 |
| **Tamil** | 58361 | 94.10 | 107.23 | 16.68 | 12.53 |
| **Telgu** | 44905 | 85.41 | 84.20 | 15.53 | 11.75 |

Table 1: Average Statistics of Sentences across Language Pairs

The amount of data provided is significantly higher for Hindi and Bengali as compared to others. The number of characters in a sentence is significantly higher than their English translations for Malayalam and Tamil. In contrast, it is either less or nearly the same for the others. For all the languages except Hindi, the number of words is much less than the English counterpart. Table 2 shows that the fraction of sentences across languages is the same for all the datasets.

| Language | Train | Validation | Testing |
|---|---|---|---|
| **Bengali** | 17.15 | 17.15 | 17.15 |
| **Gujarati** | 11.83 | 11.83 | 11.83 |
| **Hindi** | 20.13 | 20.13 | 20.13 |
| **Kannada** | 11.66 | 11.66 | 11.66 |
| **Malayalam** | 13.47 | 13.47 | 13.47 |
| **Tamil** | 14.54 | 14.54 | 14.54 |
| **Telgu** | 11.19 | 11.19 | 11.19 |

Table 2: Percentage of Sentences across languages in various Datasets

## 4 Model Description

### 4.1 Overview

I have majorly explored two types of models. The first one was developed based on a theory regarding word embeddings covered in class. After working with these models, I experimented with RNNs, which didn't yield promising results. Falling back to the original series of models and adding a word order scorer led to the development of my best-performing model.

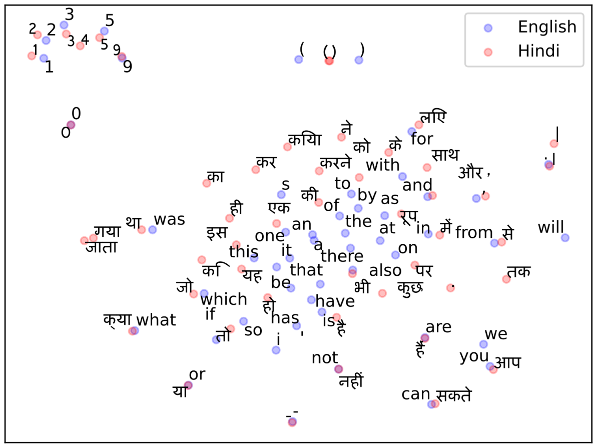

Figure 1: Top 50 frequent word's representation learnt by Best Model

### 4.2 Word Embedding Based Models

#### 4.2.1 Inspiration

"Similar meaning words occur in similar sentences so their word-embeddings receive similar updates, leading them to have similar word-embeddings" was a topic covered in class. Also, averaging the word embeddings of all the words in the sentence practically works well in representing the meaning of a sentence. As similar words with similar meanings lie closer in the word embedding space for a given language, if the same could be achieved for a pair of languages, then a word-by-word translation could be easily performed.

#### 4.2.2 Model 1

Given such word embedding for the pair of languages, the sentence representation of two sentences with the same meanings in both languages would be the same. To make the model learn such word embeddings, MSE loss has been used between the vector representations of the pair of sentences with the same meaning, which has been found by averaging the word embeddings. The tokeniser I used was splitting the sentence at punctuation and spaces.
I also speeded the training by representing the forward process as matrix products. It was impossible to store all sentences as binary vectors in the GPU, Sub-epoch denoting the number of times iterating over a set of vectorised sentences. For prediction, all I did was tokenise the input sentence; if the word is in english vocab, get its word em and find the nearest word in the Indian language. If not in vocabulary, then don't predict anything. Output is a space-separated concatenation of all the above words.
There were several shortcomings of this model, which I tried to address in the upcoming models, but the best part was I got rough translations, punctuations and some words, which was sufficient motivation!

### 4.2.3 Model 2

Analysing the predictions of the first model, it was observed that it was successful at predicting some numbers, but it wouldn't be possible to predict all numbers since it wouldn't be in its vocabulary. To address this issue, I added splitting on english numbers in the tokenisers. Word Normalisation helped reduce the English Vocabulary for the total sentences in the corpus by 10,000 size. In the Indian languages, non-English numbers and other punctuation were used. I manually added the punctuation and numbers of all seven languages from Wikipedia to my tokeniser. Also, there was an issue in the previous model of having too many spaces in the translations, so I replaced the word in the sentence as it was in the original sentence, keeping the space the same.

### 4.2.4 Model 3

The good part of Model 2 was that it learned the translation for frequently occurring words well but performed poorly on less frequently occurring words. So, to accommodate this, inspired by the TF-IDF method, I added a weightage of IDF in updation. This was to ensure that the model learned the rarer words faster than the more frequent ones. The initial results of this turned out to be negative since the model started not learning anything. After experimenting with parameters, it was able to learn a much more diverse vocabulary of translations than before. But it got a lesser score than before, making me drop this technique in later ones. I suspect the reason for it is that it didn't perform as well on frequent words, so it had a poorer score even though it had a diverse vocabulary.

### 4.2.5 Model 4

I utilised the pre-trained Glove word embeddings for english available on the Stanford NLP Website. Since all the words present in the vocabulary were not present in the pre-trained word embedding, I learnt these words similarly to before. After experimenting, I decided to move on with 100 dim. The positive point for pre-trained word embeddings was that it had a lesser chance of coming to a situation in which the original model could have come to make the loss zero. This time, the model was even better than the TF-IDF at learning a very diverse vocabulary but could not match previous models' performance in terms of score.

### 4.2.6 Model 5

Now, one unexploited information was the extra translations available in Hindi. If somehow similar to how the base model is learning a one-to-one mapping between words in both languages, if we could learn a mapping between the letters of both languages, this could be used to handle unknown tokens in English. There were around 4000 such words which were present in the Hindi corpus. Using regex, I extracted these words and tried to build a similar model on it, but it didn't work out, so I was unable to incorporate it.

## 4.3 RNN Based Models

### 4.3.1 RNN

Referring to the tutorials, I implemented an RNN. I had kept one encoder, which gave no output and only gave output as a hidden state. The input to the encoder was initially 0 and one-hot vectors of the words in the vocabulary. The output of the encoder was fed into the decoder as the representation of the sentence. Which then produced the output sentence. The input to the decoder was NULL, and the output was a probability softmax vector. I experimented with many things like losses, different hidden dimensions, and filtering of input data, but still, it predicted the same word.

## 4.4 RNN with teacher-forcing

Also, this time, in the decoder, I added the previous word as input. While training, the model was a little more diverse during initial iterations of training than normal RNN, but in prediction it also did a similar thing as RNN.

## 4.5 Adding a Word Order Scorer and Replacing Word Embeddings

### 4.5.1 Inspiration

The following paper [1] explains that the ordering of grammar in Indian languages of the form Subject-Object-Verb while for English, it is Subject-Verb-Object. It also mentions that better machine translations can be obtained by performing re-ordering of subject, object and verb in the source language. In the following approach for machine translation [2], they have described the idea of improving word-to-word translations by having a language model that can score grammatically better sentences higher than those that are incorrect. Combining these two ideas, I developed a word order checker to score the order of adjacent pairs of words. The Indic-NLP repository by AI4Bharat, IIT Madras[1] contains pre-trained static FastText word embeddings of 300 dimensions which I utilised for Indian languages. Since I hoped for some dimensions of the language's word embeddings to capture the language's grammar, the following was the architecture I decided to go for.

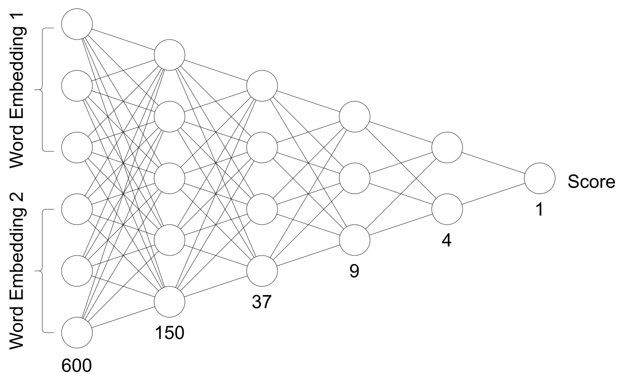

Figure 2: Word Order Scorer Architecture

### 4.5.2 Model 8 and 9

The Word-Order-Scorer inputs the concatenated vector representation of two words and outputs the score for that order. It is a fully connected neural network with the architecture as depicted in Figure 2, with 600-dimensional output and the score as the output and sigmoid activations have been used. A higher score indicates the arrangement to be more probable. To train the model, I have taken all possible pairs of adjacent words in the corpus and given them as input along with a label. The label is a binary value denoting if word one should occur before word two or no. To avoid any memorisation in training, I have created both a true and false example for every pair of adjacent words.

### 4.5.3 Model 10

I carried out the parallel training of two models similar in architecture to the fourth model on English-Gujarati, with the difference being the language that had pre-trained word embeddings used. The first one utilised 100-dimensional Glove English Glove word embeddings while the second utilised Indic_NLP 300-dimensional word embeddings. The second one learned the word mappings much faster and more accurately than the first one, so I decided to proceed with training all language pairs with the Indian word embeddings. This increased the BLEU score from 0.039 to 0.047.

## 4.6 Final Model

The final model is Model 10, followed by Word-Order-Sentence Corrector.

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

## 5 Experiments

### 5.1 Data Preprocessing

### 5.2 Hyper Parameters for Models

#### 5.2.1 Word Embedding Based Models

To monitor how models are performing, after a fixed amount of time I used to run the model on a given example and print its output to measure visually the generated out and also print the loss function.
Adding sub-epochs helped speed up the training due to the speed of vectorised calculations. The maximum possible batch size I could use basenearly 7000 sentences of 100 dimensional representation. Using such larger batch sizes from the start led to slow learning, while very small batch sizes led to overfitting of the sentence. So using a batch size of 128 at the start seemed the best, but soon after some time, that also started to overfit. So I used a batch size of 1000 to improve its performance. Also, having the number of sub-epochs comparable to or more than the batch size led to overfitting and learning wrong representations for words. So, I kept it nearly the square root of the batch size.
Initially, I tried varying the word embedding dimension between 10, 100 and 100. At 10, it was not able to learn much. While for 1000, even after some time, it had not learnt anything. One hundred was optimal out of these three. Weighing the terms by Inverse Document frequency significantly slowed down the learning process. To accommodate the decrease I tried to increase the learning rate, but this leaded to explosion of word embeddings. So to compensate for it, I trained the model for longer time and change the weight to sqrt(IDF) but still got poorer results so dropped out it.
In the side-by-side training process of Glove Embeddings and AI4B embeddings, I observed that the AI4B embeddings learnt significantly faster as compared to Glove. One possible reason for this could have been that Glove embeddings were 100 dimensional while AI4B ones were 300 dimensional, but due to time constrains I proceeded with the AI4B word embeddings for the final model.
For the models 1 to 5 and 10, each epoch took around 5 minutes. While for the word order scorers, one epoch took around 45 to 60 minutes. For model 10, I trained it first on a smaller batch size so that it could first learn decent things quickly and then followed by a few epochs on a larger batch size for better generalisation. In the details mentioned in Table 3, Adam optimiser was used with a learning rate of 1e-3, and the loss function used was mean squared error.

| Model | Epoch | Sub-Epoch | Batch Size | Embedding-Dim | Word-Embedding |
|---|---|---|---|---|---|
| 1 | 5 | 5 | 1000 | 100 | Self |
| 2 | 5 | 5 | 1000 | 100 | Self |
| 3 | 15 | 15 | 1000 | 100 | Self |
| 4 | 1 | 100 | 1000 | 100 | Glove |
| 5 | 30 | 30 | 1000 | 100 | Glove |
| 8 | 1 | - | 1 | - | - |
| 9 | 2 | - | 1 | - | - |
| 10 | 30<br>10 | 10<br>30 | 128<br>1024 | 300 | AI4B |

Table 3: Word Embedding Based Model Hyper-Parameters

#### 5.2.2 RNNs

Table 4 summarises the parameters I tried experimenting with for RNN models. While some of these techniques helped produce a little more variety of words during the initial iterations of training, I was unable to overcome the error of producing the same word during testing. I used the loss functions to calculate the error between each output word produced and the word's representation in the target. I had forced the model to generate as many number of words as in the target sentence. For Average Cross Entropy loss and Average MSE loss across words, I was unable to see any significant change in the loss across, while for NLL Loss which is similar to cross-entropy loss, I was able to see some changes in the loss function, although not much. For the sum of MSE across sentences, if I used a very small learning rate from 1e-5 to 1e-8, then there was a significant change in the loss. But in all of these cases, the model ended up predicting the same word. I had also tried using EOS and SOS tokens to denote the end and start of the sentence, but the issue that I faced was that the model ended up always predicting EOS since this caused it to get one word always correct, so I removed these tokens. Adding Teacher forcing helped it predict more diverse words for initial iterations as compared to without it, but soon it again went back to square one. Ordering the data such that the sentences appeared in decreasing order of length helped the model to learn something quickly, but it was overfitting on sentences and again predicting the same word. I also tried varying the word representation and the hidden representation but got similar results.

| **Parameter** | **Values** |
|---|---|
| Loss Functions | MSE, NLL, Cross-Entropy |
| Training | Teacher-Forcing, SOS & EOS Tokens, Data-Ordering |
| Hidden Representation Dimension | 100, 1000 |
| Word Representation | One-Hot Vector, Word Embeddings |
| Sub-Epochs | 1, 10 |

Table 4: RNN Based Model Experimentations

| Model | Examples | Input |
|---|---|---|
| 6 | 10,000 | One-Hot Vetor |
| 7 | 10,000 | Word-Embedding |

Table 5: RNN Based Model Hyper-Parameters

## 6 Results

The model developed by utilising pre-trained Indian Language Word Embeddings performed the best out of all the models I had developed during the training phase. This was because it had a longer training time as compared to its predecessors, and also it had higher dimensional word embeddings capturing more information. It had achieved 8th rank on the leaderboard.

| **Model** | **chrF++** | **ROUGE** | **BLEU** |
|---|---|---|---|
| 1 | N/A[2] | N/A | 0.034 |
| 2 | 0.278 | 0.263 | 0.039 |
| 3 | 0.270 | 0.244 | 0.036 |
| 4 | 0.268 | 0.236 | 0.032 |
| 5 | 0.266 | 0.249 | 0.034 |
| 6 | 0.029 | 0.145 | 0.000 |
| 7 | 0.034 | 0.139 | 0.000 |
| 8 | 0.278 | 0.263 | 0.039 |
| 9 | 0.278 | 0.266 | 0.039 |
| 10 | 0.318 | 0.292 | 0.047 |

Table 6: Results - Training Phase

The model developed by attaching the word order scorer to model 10 performed the best in the testing phase. This model retained the good qualities of model 10 while adding on a minor module to improve its predictions. It had achieved 10th rank on the leaderboard. This combination had not been tested during the training phase.

[2]The testing script on Codalab initially had some error due to which ROUGE and chrF++ scores were wrongly shown

| Model | chrF++ | ROUGE | BLEU |
|---|---|---|---|
| 10 | 0.318 | 0.292 | 0.047 |
| Final | 0.318 | 0.297 | 0.047 |

Table 7: Results - Testing Phase

## 7 Error Analysis

Figure 1 [3] depicts the distribution of the word embeddings learnt by the model for the top 50 frequent words from the source and target language for the English-Hindi pair. The model has successfully brought the words having exact translations between the source and the target language as we can see for numbers, punctuations and few other words. Example for English-Hindi pair:
Source: And then we need to assure students that a computer can perform all the steps that are used in manual logarithms, and so whatever manual algorithm you have we can somehow feed to a computer as well.
Translation by Model 10:
और फिर हम जरूरत भी वाकया छात्रों भी एक कंप्यूटर सकता करने सभी भी कदम भी हैं किया में मैनुअल , और तो जो मैनुअल एल्गोरिदम आप भी हम सकता तरह भी भी एक कंप्यूटर और भी।
Red is the model predicting common words
Translation by Final Model:
और फिर हम जरूरत भी वाकया भी एक कंप्यूटर सकता छात्रों सभी भी कदम भी करने किया में मैनुअल हैं , और तो जो मैनुअल आप एल्गोरिदम भी हम तरह भी भी सकता एक कंप्यूटर और भी।
Blue have been re-ordered by the Scorer
The Model 10 has successfully learnt the translations for punctuations and several other words correctly while not for some words. But one major issue it suffered from was that for words in English which didn't have exact independent translations in Indian languages it predicted few common words like भीin the above case. For example words like is, are, to do not have exact translations in Hindi so these types of words suffer. Also, one more issue with this model is that the sentence length is always same as the length of the source sentence which is not true in general and this causes the degradation of translation quality.
The additional part of word order correction which we had attached to improve our predictions has successfully improved the grammatical structure of the sentence locally at some places while worsened it at few places. For example किया में मैनुअल हैंappears to be grammatically more sound as compared to the original ordering of हैं किया में मैनुअल. But if we look at the other changes made by this unit they appear to still remain grammatically incorrect. The reason for this appears to be the lack of context the model is being provided. All the model has is the order of two words, due to which it is successful at only a local scale while had it been given more context in the form of few previous words and few later words, it could have performed better.

## 8 Conclusion

Comparing the amount of time required for training and the number of parameters of the model created with explicit Encoder-Decoder models, this model gives a much better output for the same training time with fewer parameters. Also, there was no requirement to prune or be careful while selecting our training dataset since the model could handle high-load data while training at much faster speeds. But even though it has a better output-to-train-time ratio, it seems that it can't give better predictions beyond the limit with the current architecture. Another major flaw in the outputs of the current model is the length of sentences generated is nearly the same as the source language, which is not true in general. In the second phase, I look forward to experimenting with better standard Encoder-Decoder Architectures like GRUs, LSTMs etc.

[3]For the creation of plot used t-SNE algorithm in Pytorch by `https://github.com/mxl1990/tsne-pytorch` and for proper placing of labels in plot used `https://github.com/Phlya/adjustText`

## References

[1] A. Kunchukuttan and P. Bhattacharyya, “Utilizing language relatedness to improve machine translation: A case study on languages of the indian subcontinent,” 2020.

[2] M. Ott, “Unsupervised machine translation: A novel approach to provide fast, accurate translations for more languages,” Aug 2018.