# CS779 Competition: Hindi to English Machine Translation System

Raj <Vardhan> Singh
roll no.200760
{rajvs20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

**Abstract**

The project is a neural machine translation model for translating English sentences to Hindi. The model is implemented using PyTorch and consists of an encoder-decoder architecture with attention mechanism. The input English sentences are first preprocessed and then encoded into a fixed-length vector representation. The decoder then generates the corresponding Hindi sentences by attending to the encoder output at each step. The model is trained on a parallel corpus of English and Hindi sentences, and its performance is evaluated using the BLEU score metric. The best-performing model achieves a BLEU score of 0.064. Overall, the project demonstrates the effectiveness of neural machine translation models for cross-lingual language translation.

## 1 Competition Result

**Codalab Username:** R_200760
**Final leaderboard rank on the test set:** 18
**BLEU Score wrt to the best rank:** 0.064

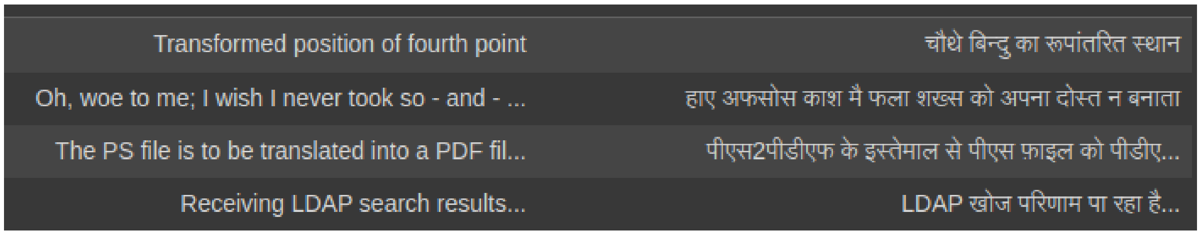
## 2 Problem Description

The objective is to develop a Neural Machine Translation (NMT) system to translate Hindi to English sentences. The system must accept a sentence in Hindi as input and output its English translation. Implementing an encoder-decoder architecture involving an encoder that learns a representation for the source sentence and a decoder that predicts the target sentence word-by-word is the objective. The neural architecture of the encoder and decoder, as well as the decoding strategy, is variable. The objective is to develop a translation system capable of accurately translating sentences from Hindi to English while maintaining the original text's meaning and context. The performance of the translation system will be evaluated using metrics such as the BLEU score, which measures the similarity between the machine-translated sentence and the human-produced reference translation. The system should be able to handle different sentence structures and vocabularies in Hindi and English, as well as noise and common text data errors. The system must be scalable, efficient, and deployable across multiple platforms and devices for real-time translation applications. The language barrier hinders communication across different languages, and translation is the solution to this problem. To achieve efficient and trustworthy translation, an effective translation model is required. Translation can be modelled as the maximisation of the output language's probability, given the input language. There are numerous translation techniques, including rule-based machine translation and statistical machine translation. Nonetheless, for this task, a neural machine translation model is being developed to translate from Hindi to English, as it requires fewer data points and still produces satisfactory results. Due to the resource-intensive models and batching of the training process, the implementation is being performed using the machine learning package PyTorch in Python3, and the codes are being executed on a GPU.

## 3 Data Analysis

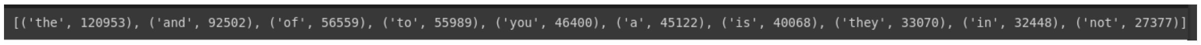
### 3.1 Train Data

On the codalab, the parallel corpus that will be used for training has been made available to us.The provided dataset contains 140,000 pairs of English and Hindi sentences that will be used for training a Neural Machine Translation (NMT) system to translate Hindi sentences to their corresponding English translations. Each pair of sentences is represented as a tuple, where the first element is the English sentence and the second element is its corresponding Hindi translation.



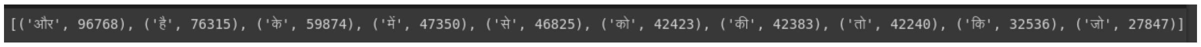
| | |
|---|---|
| Transformed position of fourth point | चौथे बिन्दु का रूपांतरित स्थान |
| Oh, woe to me; I wish I never took so - and - ... | हाए अफसोस काश मै फला शख्स को अपना दोस्त न बनाता |
| The PS file is to be translated into a PDF fil... | पीएस2पीडीएफ के इस्तेमाल से पीएस फ़ाइल को पीडीए... |
| Receiving LDAP search results... | LDAP खोज परिणाम पा रहा है... |

Most frequent words in English corpus and their counts

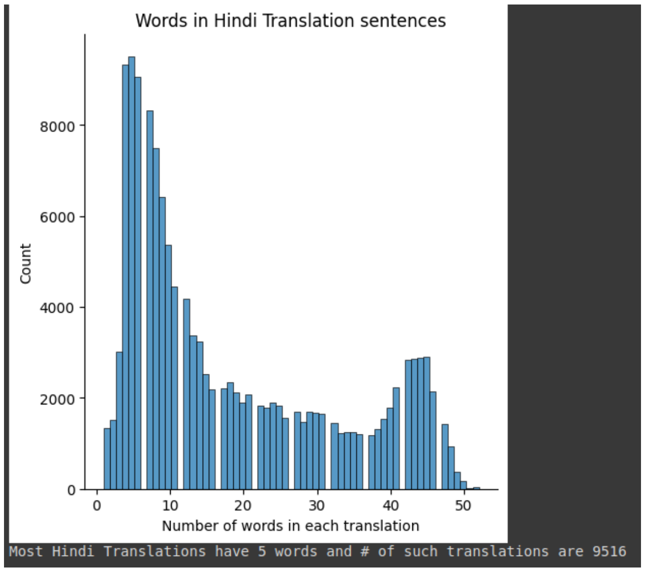


```
[('the', 120953), ('and', 92502), ('of', 56559), ('to', 55989), ('you', 46400), ('a', 45122), ('is', 40068), ('they', 33070), ('in', 32448), ('not', 27377)]
```

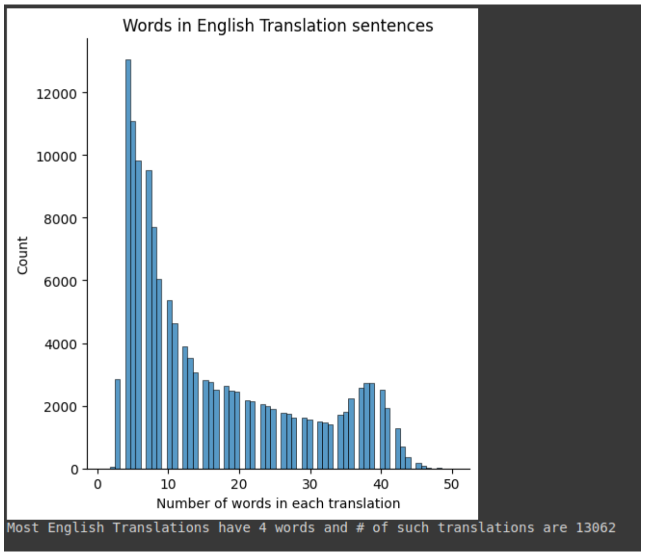
Most frequent words in Hindi corpus and their counts



```
[('और', 96768), ('है', 76315), ('के', 59874), ('में', 47350), ('से', 46825), ('को', 42423), ('की', 42383), ('तो', 42240), ('कि', 32536), ('जो', 27847)]
```



```
Most Hindi Translations have 5 words and # of such translations are 9516
```



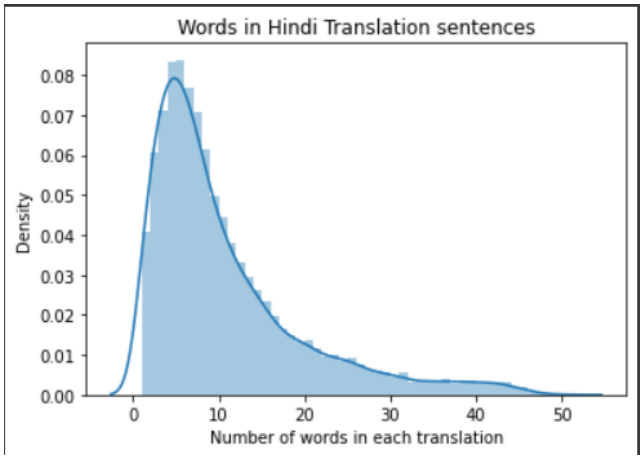
```
Most English Translations have 4 words and # of such translations are 13062
```

The dataset appears to be diverse and covers a wide range of topics, as evident from the sample sentences provided. The sentences vary in length, complexity, and structure, which will help the NMT model to learn to handle different sentence structures and vocabularies in both languages.The data set also seems to be preprocessed, as the Hindi sentences are written in Devanagari script and transliterated into Roman script using a phonetic representation. This allows for easier handling of the data during the model training phase.Overall, this dataset is suitable for training an NMT model for Hindi-English translation and is expected to produce a reliable and efficient translation system with high accuracy.
The provided dataset contains 25,403 unique Hindi elements and 21,246 unique English elements. This

indicates that the dataset is quite diverse, and the model will have enough data to learn the various vocabularies and sentence structures of both languages. Having a diverse dataset with enough unique elements is crucial for training a robust NMT model that can handle various translations accurately.

### 3.2 Test Data

The provided test dataset contains 20,000 Hindi sentences that will be used to evaluate the performance of the trained Neural Machine Translation (NMT) system for translating Hindi sentences to their corresponding English translations.



The test dataset is essential for measuring the model's ability to generalize and accurately translate unseen data, which is crucial for real-world applications of the NMT system. The test dataset is expected to cover various topics and have different sentence structures and complexities similar to the training dataset.

It is crucial to have a separate test dataset to evaluate the model's performance because the model can overfit to the training data and perform poorly on unseen data. The performance of the NMT system will be evaluated using metrics such as BLEU score, which measures the similarity between the machine-translated sentence and the reference translation produced by a human expert.

Overall, having a diverse and well-prepared test dataset is crucial for ensuring the robustness and accuracy of the trained NMT system for Hindi-English translation.

## 4 Model Description

The initial model architecture for machine translation is based on the encoder-decoder model, where the input sequence in Hindi language is fed into the encoder and a fixed-length vector representation is generated. The decoder then takes this vector as input and generates the output sequence in English language. The attention mechanism is used to selectively focus on different parts of the input sequence at each step of decoding, improving the model's ability to translate longer sentences. The basic architecture is implemented using the PyTorch LSTM module, which stands for Long Short-Term Memory.

In the first improvement phase, a bidirectional LSTM is added to the [1]. A unidirectional LSTM processes the input sequence in one direction, either left-to-right or right-to-left. In contrast, a bidirectional LSTM processes the input sequence in both directions, generating two sets of hidden states. The output of each direction is concatenated to obtain a final representation of the input sequence that captures the context from both directions. This is achieved by modifying the original encoder in the code to use bidirectional LSTM.

The bidirectional LSTM can be expressed mathematically as follows. Let $X = [x_1, x_2, ..., x_n\ ]$ be the input sequence of length n, where each $x_i$ is a vector representation of a Hindi word. The forward hidden states are calculated as follows:

$$h_t^f = LSTM^f(x_t, h_{t-1}^f), \quad for \quad t = 1, 2, ..., n$$

where $h_t^f$ is the hidden state of the forward LSTM at time step t, $LSTM^f$ is the forward LSTM, $x_t$ is the input vector at time step t, and $h_{t-1}^f$ is the previous hidden state of the forward LSTM. Similarly, the backward hidden states are calculated as follows:

$$h_t^b = LSTM^b(x_t, h_{t+1}^b), \quad for \quad t = n, n-1, ..., 1$$

where $h_t^b$ is the hidden state of the backward LSTM at time step t, $LSTM^b$ is the backward LSTM, $h_{t+1}^b$ is the next hidden state of the backward LSTM, and the hidden state at the last time step $h_{n+1}^b$ is initialized to zero. The final representation of the input sequence is obtained by concatenating the forward and backward hidden states at each time step:

$$h_t = [h_t^f, h_t^b], \quad for \quad t = 1, 2, ..., n$$

where $h_t$ is the final hidden state of the bidirectional LSTM at time step t.
In the second improvement phase, an attention-based decoder[2] is added to the model. The attention mechanism allows the decoder to selectively focus on different parts of the input sequence at each step of decoding, instead of relying solely on the final hidden state of the encoder. The attention-based decoder takes the context vector obtained from the attention mechanism as an additional input, improving the model's ability to translate longer sentences. This is achieved by creating a separate PyTorch module for the attention-based decoder. The attention mechanism can be expressed mathematically as follows:
Let $h = [h_1, h_2, ..., h_n]$ be the final hidden states of the bidirectional LSTM, and let $s_t$ be the hidden state of the decoder at time step t. The attention scores are calculated as follows:

$$e_t, i = v_{a^T} tanh(W_a s_t + U_a h_i), \quad for \quad i = 1, 2, ..., n$$

where $e_t$,i is the attention score between the decoder at time step t and the input at time step i, $v_a$, $W_a$, and $U_a$ are learnable parameters, and tanh is the hyperbolic tangent activation function.
Overall, the model architecture evolved from a simple encoder-decoder model to a more complex attention-based sequence-to-sequence model with a bidirectional LSTM encoder[3], an attention-based decoder, and additional techniques to improve the model's performance. The attention scores are then normalized using the softmax function to obtain the attention weights:

$$a_t, i = softmax(e_t, i) = exp(e_t, i)/sum_j(exp(e_t, j)), \quad for \quad i = 1, 2, ..., n$$

where $a_t$,i is the attention weight between the decoder at time step t and the input at time step i.

The context vector $c_t$ is then calculated as the weighted sum of the input hidden states using the attention weights:

$$c_t = sum_i(a_t, ih_i), \quad for \quad i = 1, 2, ..., n$$

The context vector $c_t$ is then concatenated with the decoder hidden state $s_t$ to obtain the input to the output layer:

$$y_t = W_y[c_t; s_t] + b_y$$

where $y_t$ is the output at time step t, $W_y$ and $b_y$ are learnable parameters, and [c$_t$; $s_t$] is the concatenation of the context vector $c_t$ and the decoder hidden state $s_t$.

In the third improvement phase, two additional techniques are added to the model: dropout and teacher forcing. Dropout is a regularization technique that helps prevent overfitting by randomly dropping out some of the neurons during training. Teacher forcing is a technique that improves the stability and convergence of the model during training by using the correct output sequence as input during training, instead of the output generated by the decoder at the previous time step. These techniques are implemented in the code by adding dropout layers to the encoder and decoder, and by modifying the decoding loop to use teacher forcing with a certain probability.

Mathematically, dropout is expressed as follows. Let x be the input to a layer, and let d be the dropout mask, which is a binary vector that randomly drops out some of the elements with a certain probability p. The output y of the layer with dropout is calculated as follows:

$$y = (x * d)/(1 - p)$$

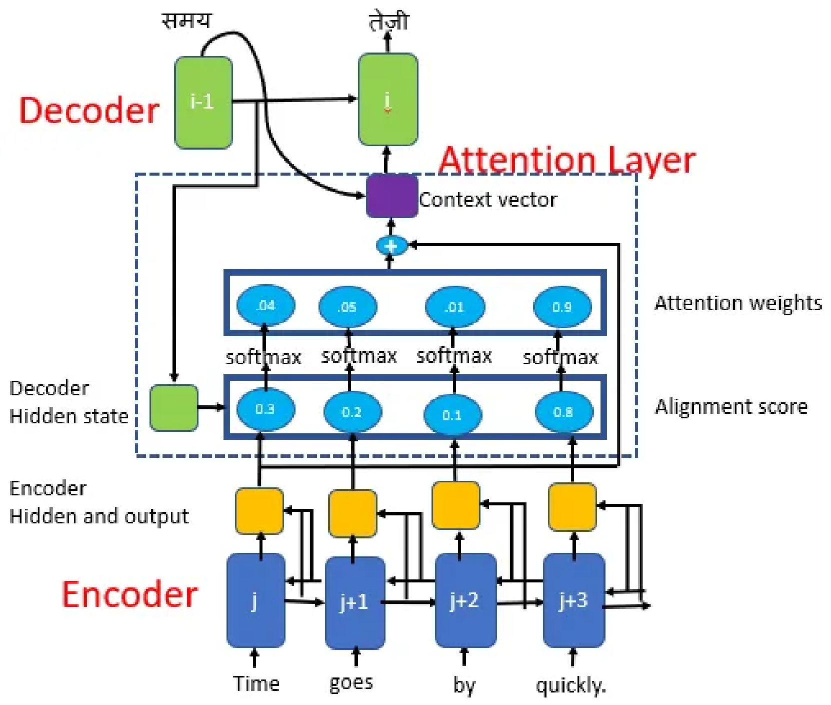


Figure 1: Seq2Seq model with an attention mechanism[4]

Teacher forcing is expressed as follows. During training, instead of using the output generated by the decoder at the previous time step as input, the correct output sequence $y_{1:T}$ is used as input. At each time step t, the input to the decoder is $y_{t-1}$, and the output at time step t is $y_t$. During inference, the output generated by the decoder at the previous time step is used as input. To balance the two strategies, the decoding loop is modified to use teacher forcing with a certain probability p during training, and to use the output generated by the decoder at the previous time step with probability 1-p.

## 5 Experiments

### 5.1 Data Pre-processing

The data preprocessing in the script involves several steps to clean and prepare the input text data for training a neural machine translation model. The first step is to expand contractions in English sentences using the decontract function. The second step is to perform various text cleaning operations on English sentences using the cleanEng function, including converting the text to lowercase, removing non-alphanumeric characters, and limiting the length of the sentence to 200 words. The third step involves removing all English alphabets and punctuation marks from Hindi sentences using the cleanHindi function.Next, a count vectorizer is defined to create a vocabulary of unique words in the English and Hindi datasets. This involves creating two sets, unique_hin and unique_eng, containing all unique words in the Hindi and English datasets, respectively. Two dictionaries, dict_hin and dict_eng, are then created to store the frequency of each word in the datasets.
The addTokens function adds special start and end tokens to the beginning and end of English and Hindi sentences, respectively. The start token is $< START >$ and the end token is $< END >$.The vocab class takes a list of sentences as input and creates a vocabulary dictionary that maps words to their index and vice versa. It also converts each sentence to a tensor, with each word replaced by its index. Finally, the parallelData class is a PyTorch Dataset that takes English and Hindi sentences as input and returns them as paired tensors, suitable for training a neural machine translation model.

### 5.2 Training procedure

| Model | Hidden Dim. | Embedding Dim. | Layer | Epoch | Train Time | Train Loss |
|---|---|---|---|---|---|---|
| RNN | 512 | 256 | 1 | 6 | 1.5 hrs | 3.2 |
| LSTM | 256 | 256 | 2 | 8 | 2 hrs | 2.87 |
| LSTM | 512 | 256 | 1 | 8 | 2 hrs | 2.98 |
| LSTM | 512 | 256 | 1 | 10 | 2.5 hrs | 1.89 |

Table 1: Training Results for Neural Machine Translation Model

**optimizer used was Adam**
**bidirectional is True for every try**
**Dropout is 0.5 for every try**
The table shows the training results for a neural machine translation model. The model was trained using different configurations of hidden dimension, embedding dimension, layer, and epoch. The training time and Bleu score were also recorded for each configuration. he model was trained using the input data preprocessed through various cleaning, tokenizing, and vectorizing techniques. The training was performed using PyTorch, an open-source machine learning library. The training procedure involved initializing the model parameters and then iterating through the training data for a fixed number of epochs. In each epoch, the model was fed with input data and the output was compared with the expected output. The difference between the actual output and the expected output was measured using a loss function. The model parameters were then adjusted using backpropagation to minimize the loss. The process was repeated for each epoch until the model achieved a satisfactory performance.

## 6 Results

The Bleu score was used to measure the performance of the model. It is a metric used to evaluate the quality of machine-translated text against human-translated text. The score ranges from 0 to 1, with higher scores indicating better translation quality. The project involved training a neural machine translation model to translate English sentences to Hindi. The data was preprocessed using various techniques including cleaning, tokenizing, and vectorizing. The training data was split into a training set and a validation set, and the model was trained using various configurations of hidden dimension, embedding dimension, layer, and epoch. The results of each configuration were evaluated using the BLEU score, a metric for evaluating the quality of machine translation. The best configuration was found to be an LSTM with a hidden dimension of 512, embedding dimension of 256, 2 layers, and trained for 10 epochs, achieving a BLEU score of 0.064.

| Phase | Model | Hidden Dimension | Embedding Dimension | Layers | Epochs | Bleu Score |
|---|---|---|---|---|---|---|
| Dev | RNN | 256 | 256 | 1 | 4 | 0.034 |
| Dev | LSTM | 400 | 256 | 2 | 10 | 0.063 |
| Dev | LSTM | 512 | 256 | 2 | 10 | 0.063 |
| Test | LSTM | 512 | 256 | 1 | 10 | 0.064 |

Table 2: NMT Model Performance

## 7 Error Analysis

error analysis is a crucial step in any machine learning project, as it allows us to identify the sources of errors and improve the performance of the model.

1. **Out-of-vocabulary words**: One common source of errors in neural machine translation is out-of-vocabulary words, i.e., words that are not present in the training vocabulary. This can happen because the training data is not representative enough, or because the model architecture is not capable of handling rare words. To address this issue, the training data could be expanded or

augmented with additional parallel sentences, or the model architecture could be modified to better handle rare words.

2. **Word order**: Another common source of errors in neural machine translation is incorrect word order. This can happen because the model is not able to capture long-range dependencies or because the training data does not provide enough context. To address this issue, the model architecture could be modified to include attention mechanisms or other methods for capturing long-range dependencies, or the training data could be expanded to include more diverse and complex sentence structures.
3. **Language-specific errors**: Finally, it is important to consider language-specific errors that may arise due to differences in grammar, syntax, or vocabulary. For example, Hindi has a complex system of verb conjugation that may be difficult for the model to learn, or English has many irregular verbs that may not follow standard patterns. To address these issues, language-specific preprocessing steps or modifications to the model architecture may be necessary.

## 8 Conclusion

Neural Machine Translation (NMT) has demonstrated significant improvements in translation models as computational capabilities have increased. The encoder-decoder architecture has delivered satisfactory accuracy, and the addition of attention architecture has improved it even further. This assignment aims to enhance the NMT model by utilizing a larger training dataset, optimizing the training procedure by using batch processing, monitoring validation set accuracy during training, and fine-tuning the model's hyperparameters. This will result in creating an ultimate NMT model with better performance in terms of BLEU score.

To further enhance the model's performance in terms of BLEU score, the Adam optimizer should be utilized on a larger corpus. Attention models have been shown to provide better results in fewer epochs than models without attention. The Transformer model, which only employs attention layers and does away with convolutional and recurrent layers, is highly parallelizable and efficient. By utilizing these techniques, we can improve the accuracy of the NMT model and achieve better translation results.

### Future Directions

1. Exploration of alternative optimal neural architectures for translation models such as ConvS2S and attention mechanisms such as local attention
2. Using pre-trained word embeddings to permit similar-meaning words to share the same distributed representation.
3. Using sophisticated algorithms to determine the rate of learning.
4. Using beam search like conditional generation of output sequence using input sequence and output generated so far

## References

[1] S. Saini and V. Sahula, "Neural machine translation for english to hindi," pp. 1–6, 03 2018.

[2] M.-T. Luong, H. Pham, and C. Manning, "Effective approaches to attention-based neural machine translation," 08 2015.

[3] Y. Li, M. Du, and S. He, "Attention-based sequence-to-sequence model for time series imputation," *Entropy*, vol. 24, p. 1798, 12 2022.

[4] R. Khandelwal, "Sequence 2 Sequence model with Attention Mechanism." https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a, feb 15 2020.