CS779 Competition: Hindi to English Machine Translation System

Kaushikraj <Vaikuntaraman> Nadar
200499
{nkaushik20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

The objective of this competition is to develop a natural language processing system that can automatically translate sentences from Hindi to English in order to address the issue of language barrier between Hindi and English speakers. The system is expected to generate grammatically correct and semantically meaningful sentences in English from Hindi input sentences. The task involves numerous challenges, such as dealing with the morphological complexities of Hindi words, recognizing different sentence structures, idiomatic expressions, and variations in the meaning of the same word based on context. The study has implemented an Attention based LSTM seq2seq2 translation model, which achieves a BLEU score of 0.067, and the system's accuracy and fluency have been evaluated using various metrics.

1 Competition Result

Codalab Username: K_200499
Final leaderboard rank on the test set: 15
BLEU Score wrt to the best rank: 0.067

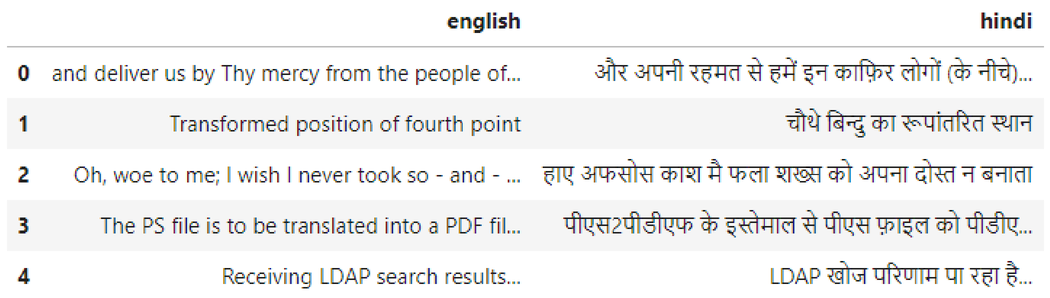
2 Problem Description

The task at hand is to create a natural language processing system that can automatically translate sentences from Hindi to English. This problem statement aims to solve the issue of language barrier between Hindi and English speakers by providing a machine-based translation system. The system should be able to take a sentence in Hindi as an input and provide a grammatically correct and semantically meaningful sentence in English as an output. The translation should be accurate, concise, and convey the intended meaning of the original sentence. The problem statement involves various challenges, such as dealing with the morphological complexities of Hindi words and understanding the grammatical structures of both languages. Additionally, the system should be able to recognize and handle different sentence structures, idiomatic expressions, and variations in the same word's meaning based on the context. The objective of this task is to develop a high-quality machine translation system that can aid in communication between Hindi and English speakers. The system's accuracy and fluency will be evaluated using various metrics, such as BLEU score, which measures the similarity between the machine-generated translation and the human reference translation.

3 Data Analysis

3.1 Train Dataset

The provided train dataset for Neural Machine Translation (NMT) contains 140,000 rows and two columns: 'english' and 'hindi'. The 'english' column contains sentences in the English language while the 'hindi' column contains their corresponding translations in the Hindi language. The number of unique elements in the Hindi data is 25,403, while the number of unique elements in the English data



	english	hindi
0	and deliver us by Thy mercy from the people of...	और अपनी रहमत से हमें इन काफ़िर लोगों (के नीचे)...
1	Transformed position of fourth point	चौथे बिन्दु का रूपांतरित स्थान
2	Oh, woe to me; I wish I never took so - and - ...	हाए अफसोस काश मै फला शख्स को अपना दोस्त न बनाता
3	The PS file is to be translated into a PDF fil...	पीएस2पीडीएफ के इस्तेमाल से पीएस फ़ाइल को पीडीए...
4	Receiving LDAP search results...	LDAP खोज परिणाम पा रहा है...

Figure 1: First 5 rows from the train dataset

is 21,246. This indicates that the Hindi data has more diverse vocabulary compared to the English data.

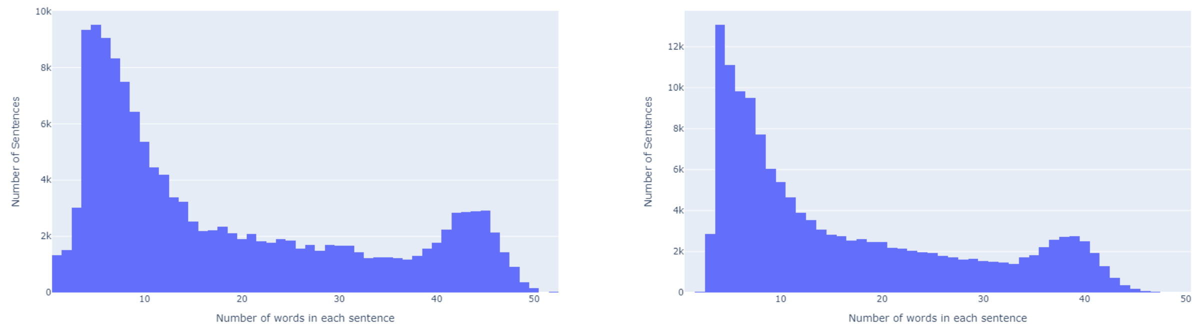
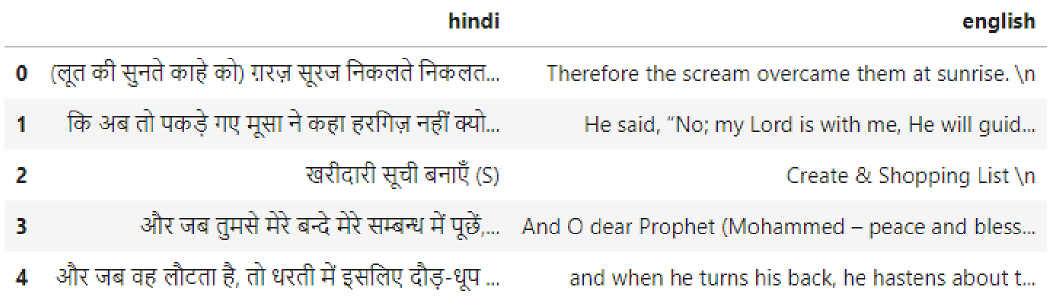


Figure 2: Plot displaying the distribution of number of words in hindi(left) and english(right) sentences

The top 10 frequent Hindi elements are: 'और', 'है', 'के', 'में', 'से', 'को', 'की', 'तो', 'कि', and 'जो'. These words are commonly used in the Hindi language, and their high frequency in the dataset reflects their importance in the language. Similarly, the top 10 frequent English elements are: 'the', 'and', 'of', 'to', 'you', 'a', 'is', 'they', 'in', and 'not'. These words are among the most commonly used words in the English language, and their high frequency in the dataset is expected. The most common Hindi translations have five words, and the number of such translations is 9,516. This suggests that sentences in the Hindi language used in the dataset are relatively short. Similarly, the most common English translations have four words, and the number of such translations is 13,062. The dataset provides a good representation of the Hindi and English languages and can be used for training an NMT model. However, it is important to note that the dataset may not cover all the variations and complexities of the languages. Moreover, the dataset may have biases or errors, which can affect the performance of the NMT model. Therefore, it is important to perform data cleaning, preprocessing, and validation to ensure the quality of the dataset before using it for training an NMT model.

3.2 Test Dataset

The provided test dataset for NMT consists of 20,000 sentence pairs, where each pair contains a Hindi sentence and its corresponding English translation. The main objective of this dataset is to evaluate the performance of a trained NMT model on previously unseen data. One important aspect of the dataset is the number of unique words in the test data that are not present in the training data. This is a crucial factor in evaluating the model's ability to generalize to new and unseen vocabulary. The dataset contains 680 such words that were not present in the training set. The presence of these new words in the test set can help in identifying the model's limitations in handling out-of-vocabulary words and its ability to generate meaningful translations for them.



	hindi	english
0	(लूत की सुनते काहे को) ग़रज़ सूरज निकलते निकलत...	Therefore the scream overcame them at sunrise. \n
1	कि अब तो पकड़े गए मूसा ने कहा हरगिज़ नहीं क्यो...	He said, "No; my Lord is with me, He will guid...
2	खरीदारी सूची बनाएँ (S)	Create & Shopping List \n
3	और जब तुमसे मेरे बन्दे मेरे सम्बन्ध में पूछें,...	And O dear Prophet (Mohammed – peace and bless...
4	और जब वह लौटता है, तो धरती में इसलिए दौड़-धूप ...	and when he turns his back, he hastens about t...

Figure 3: First 5 rows from the test dataset

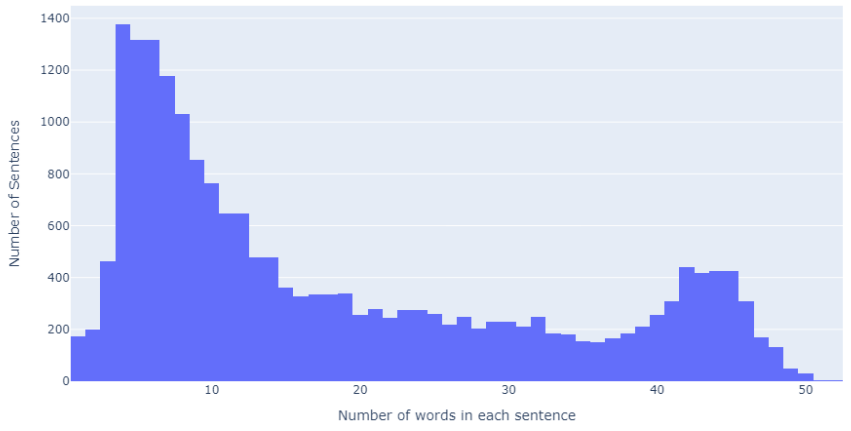


Figure 4: Plot displaying the distribution of number of words in hindi test sentences

The frequency distribution of Hindi elements in the test dataset also provides valuable insights into the language patterns in the data. The top 10 frequent Hindi elements in the test set are 'और' (and), 'है' (is), 'के' (of), 'में' (in), 'से' (from), 'को' (to), 'तो' (then), 'की' (of), 'कि' (that), and 'जो' (who/which). This information can help in identifying the important keywords and phrases in the dataset and their corresponding translations in English. Furthermore, analyzing the quality of the translations produced by the NMT model on the test dataset can provide insights into its performance. The dataset can be used to calculate various evaluation metrics, such as BLEU score, which can help in quantifying the quality of the translations generated by the model. The dataset can also be used to identify specific areas where the model may be struggling and require improvement, such as handling negation, idiomatic expressions, and context-dependent meanings.

4 Model Description

4.1 Simple RNN based Seq2Seq model

The Simple RNN-based Encoder-Decoder model consists of two parts, an encoder and a decoder, both of which are based on Recurrent Neural Networks (RNNs). The encoder takes a sequence of inputs and produces a fixed-length context vector, which is then fed into the decoder along with a start symbol to generate a sequence of outputs. The architecture can be illustrated as follows:

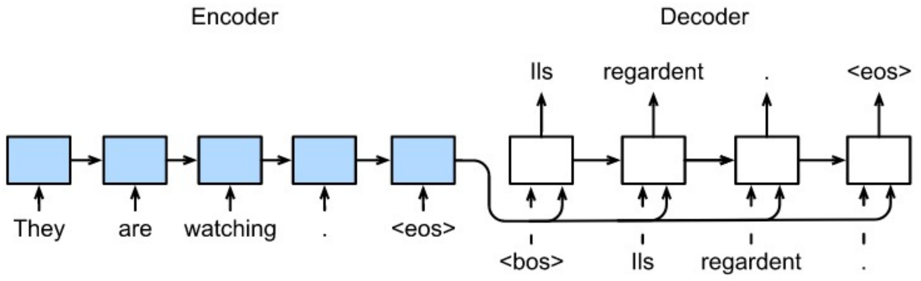


Figure 5: Architecture of basic Seq2Seq model[1]

Encoder:

$$
\begin{aligned}
h_t &= \text{RNN}(x_t, h_{t-1}) \\
c &= f(h_T)
\end{aligned}
$$

where x_t is the input at time step t, h_t is the hidden state of the encoder RNN at time step t, h_{t-1} is the hidden state of the encoder RNN at the previous time step, f is a function that maps the final hidden state h_T to the context vector c.

Decoder:

$$
\begin{aligned}
s_0 &= g(c) \\
s_t &= \text{RNN}(y_{t-1}, s_{t-1}) \\
\hat{y}t &= \text{softmax}(Wouts_t)
\end{aligned}
$$

where s_t is the hidden state of the decoder RNN at time step t, y_{t-1} is the output at time step $t-1$, s_{t-1} is the hidden state of the decoder RNN at the previous time step, g is a function that maps the context vector c to the initial hidden state s_0 of the decoder RNN, W_{out} is a weight matrix that projects the hidden state s_t to the output $\hat{y}_t$ at time step t. The output $\hat{y}_t$ is a probability distribution over the possible values of y_t.

During training, the model is optimized to minimize the cross-entropy loss between the predicted output and the ground truth output.

4.2 LSTM based Seq2Seq Model

The sequence-to-sequence (seq2seq) LSTM-based encoder-decoder model is a type of deep learning model that is often used in natural language processing (NLP) tasks such as machine translation, text summarization, and question-answering systems. It is composed of two main components: an encoder network and a decoder network[2].

Encoder Network

The encoder network processes the input sequence $X = (x_1, x_2, ..., x_T)$, where T is the length of the input sequence, and generates a fixed-length context vector c that summarizes the input sequence. This is done using a bidirectional LSTM network, where the forward and backward LSTM layers process the input sequence in opposite directions, concatenating their final hidden states to create the context vector c:

$$\begin{aligned}\overrightarrow{h}t &= \overrightarrow{\mathrm{LSTM}}(x_t, \overrightarrow{h}t-1)\\ \overleftarrow{h}t &= \overleftarrow{\mathrm{LSTM}}(x_t, \overleftarrow{h}t+1)\\ h_t &= [\overrightarrow{h}_t; \overleftarrow{h}_t]\\ c &= q(h_1, h_2, ..., h_T)\end{aligned}$$

where $[\cdot;\cdot]$ represents concatenation, $q(\cdot)$ is a function that summarizes the hidden states into a fixed-length vector (such as taking the average or using an attention mechanism), and $\overrightarrow{\mathrm{LSTM}}$ and $\overleftarrow{\mathrm{LSTM}}$ are the forward and backward LSTM layers, respectively.

Decoder Network

The decoder network takes the context vector c and generates the output sequence $Y = (y_1, y_2, ..., y_{T'})$, where T' is the length of the output sequence. This is done using another LSTM network, which uses the context vector c as input to generate the hidden states of the decoder:

$$\begin{aligned}s_t &= \mathrm{LSTM}(y_{t-1}, s_{t-1}, c)\\ \hat{y}_t &= \mathrm{softmax}(W_s s_t + b_s)\end{aligned}$$

where s_t is the hidden state of the decoder at time step t, y_{t-1} is the previously generated output token, W_s and b_s are learnable parameters, and $\hat{y}_t$ is the predicted probability distribution over the vocabulary for the next output token.

During training, the model is trained to maximize the log-likelihood of the target sequence given the input sequence:

where $\mathcal{D}$ is the training dataset and θ are the model parameters.

During inference, the model generates the output sequence word-by-word by feeding the previously generated output token as input to the decoder until an end-of-sequence token is generated.

4.3 Attention-based LSTM Seq2Seq Model

The attention-based sequence-to-sequence (seq2seq) LSTM model is an extension of the standard seq2seq model that addresses the problem of information loss that can occur in the encoder-decoder architecture. The attention mechanism allows the decoder to selectively focus on different parts of the input sequence during decoding[3]. The attention-based seq2seq LSTM model consists of three main components: an encoder, an attention mechanism, and a decoder. The encoder is responsible for encoding the input sequence into a fixed-length vector representation. The attention mechanism dynamically selects which parts of the encoder output to focus on during decoding, and the decoder generates the output sequence based on the attended encoder output and previous generated outputs.

The encoder is typically a multi-layer LSTM network that takes the input sequence X = x_1, x_2, ..., x_n as input and produces a sequence of hidden states H = h_1, h_2, ..., h_n. Each hidden state h_i is computed as:

$$h_i = \mathrm{LSTM}(h_{i-1}, x_i)$$

The decoder is also a multi-layer LSTM network that takes the attended encoder output and the previous generated output as input and generates the next output. The decoder state at time t is denoted as s_t, and the attended encoder output at time t is denoted as c_t. The decoder state s_t is computed as:

$$s_t = \mathrm{LSTM}(s_{t-1}, y_{t-1}, c_t)$$

where y_{t-1} is the previously generated output.

The attention mechanism calculates a set of attention weights α_t that indicate how much attention to pay to each encoder output h_i at time t. The attention weights are calculated as follows:

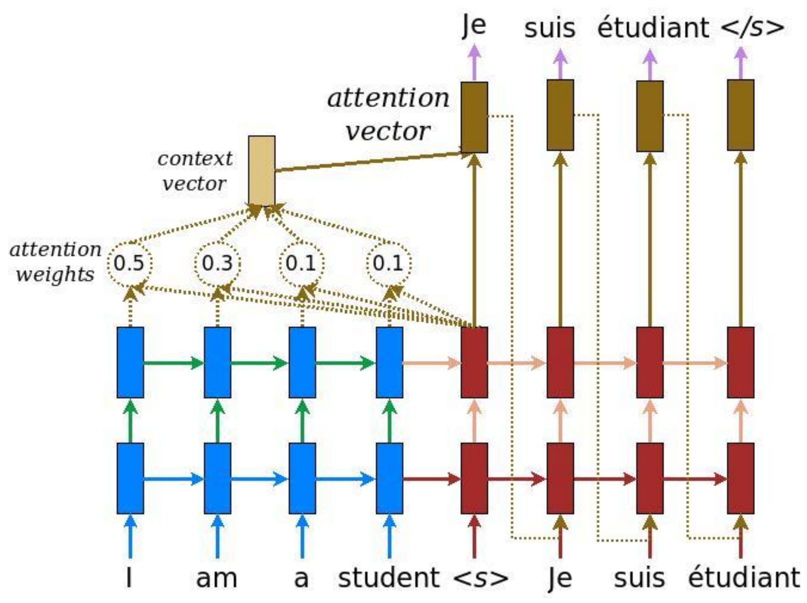


Figure 6: Architecture of Attention based Seq2Seq[4]

$$e_{ti} = \text{score}(s_{t-1}, h_i) \tag{1}$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{j=1}^{n} \exp(e_{tj})} \tag{2}$$

where score is a function that scores how well the decoder state s_{t-1} and encoder output h_i match. One common scoring function is the dot product:

$$\text{score}(s_{t-1}, h_i) = s_{t-1} \cdot h_i \tag{3}$$

Once the attention weights α_t‘ are calculated, the attended encoder output c_t is computed as a weighted sum of the encoder outputs hi:

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i \tag{4}$$

The attended encoder output c_t is then used as input to the decoder LSTM to generate the next output. The overall loss function for training the attention-based seq2seq LSTM model is typically the cross-entropy loss between the predicted output sequence and the true output sequence.

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t | y_{<t}, x) \tag{5}$$

where T is the length of the output sequence and $P(yt|y < t, x)$ is the probability of generating output y_t given the previous outputs $y < t$ and the input sequence x.

5 Experiments

5.1 Data Pre-processing

The pre-processing involves cleaning the text data and converting it into a numerical format.

The first step is to define two functions to clean the English and Hindi data. The ’cleanEng’ function converts the input text to lowercase, removes all non-alphanumeric characters, removes extra spaces, and limits the sentence length to 200 words. The ’decontract’ function replaces the contracted forms of words with their expanded form in English. The ’cleanHindi’ function removes all English characters, removes punctuation, removes extra spaces, and limits the sentence length to 200 words. Finally, the ’BaseNormalizer’ class from the ’indicnlp’ library is used to normalize the Hindi text. The cleaned English and Hindi sentences are then stored in two lists, ’clean_data_english’ and ’clean_data_hindi’. A vocabulary is created for each language using the ’vocab’ class. The ’vocab’ class first creates a dictionary of word to index and index to word for the language. Then, it converts each sentence in the language to a sequence of indices, with each index representing a unique word in the sentence. If the sentence is shorter than the maximum sentence length, padding is added at the end to make all sentences of the same length. The final output is a list of tensors, where each tensor represents a sentence in the language, and each element in the tensor is the index representation of a word. The pre-processing is done to convert the text data into numerical data that can be used as input to the machine learning model. The cleaning of data is done to remove unwanted characters, spaces, and punctuation. The use of the ’BaseNormalizer’ class in Hindi is to convert the text into a standardized form, which can improve the performance of the model. The use of padding is to make all the sentences of the same length, which is necessary for efficient computation in the machine learning model.

5.2 Training Procedure

The table provides a comparison of four different models trained on a certain dataset. The models differ in their architecture, optimizer used, learning rate, number of epochs, hidden dimensions, embedding size, number of layers, and the resulting training loss.

Model	Optimizer	LR	Epochs	T Time	Hidden Dims	Embedding	Layers	Train Loss
RNN	Adam	0.001	4	1 hr	512	400	1	4.67
LSTM	Adam	0.001	10	2.67 hrs	512	512	2	1.945
LSTM	SGD	0.001	10	2.67 hrs	200	300	1	2.45
LSTM with attention	Adam	0.001	12	3 hrs	512	512	1	1.682

Table 1: Training results for different models

Some common Parameters:

Dropout = 0 for RNN and 0.5 for others, Input Dimensions for encoder = Vocabolary size of Hindi Data, Input Dimensions for decoder = Vocabolary size of English Data, Bidirectional = Yes for LSTM and No for RNN

From the table, we can infer that the LSTM model trained with the Adam optimizer and an initial learning rate of 0.001 achieved the lowest training loss of 1.945, and required 10 epochs and 2.67 hours to train. The other LSTM model trained with SGD optimizer and initial learning rate of 0.001 had a higher training loss of 2.45 and required the same amount of time and epochs to train. The RNN model had the highest training loss of 4.67 and was the simplest model with only one layer and lower hidden dimensions and embedding size compared to the LSTM models. Finally, the LSTM model with attention had the lowest training loss of 1.682, but required a longer training time of 3 hours despite having similar settings to the first LSTM model.

6 Results

Based on the information in the table, we can see that the second model with 12 epochs performed better than the first model with 10 epochs. The second model had a lower training loss and a higher Bleu score, indicating that it generated more human-like text. However, the second model had only one layer in the LSTM, while the first model had two layers, which may suggest that the number of layers is not as important as the number of epochs in determining the quality of the generated text.

Model	Epochs	T Time	Hidden Dims	Embedding	Layers	Train Loss	Bleu Score
LSTM	10	2.67 hrs	512	512	1	1.945	0.049
LSTM with attention	10	4.3 hrs	512	512	2	2.45	0.033
LSTM with attention	10	4hrs	200	300	2	3.2	0.004
LSTM with attention	12	3hrs	512	512	1	1.682	0.065

Table 2: Performance comparison of LSTM models during dev phase

Model	Epochs	T Time	Hidden Dims	Embedding	Layers	Train Loss	Bleu Score
LSTM with attention	10	4hrs	512	512	2	1.97	0.059
LSTM with attention	12	3hrs	512	512	1	1.682	0.067

Table 3: Performance comparison of LSTM models during test phase

7 Error Analysis

Seq2Seq models have some limitations that can prevent them from achieving higher accuracy in some scenarios:

1. Limited context: Seq2Seq models have a limited context window, which means they may not be able to consider all the relevant information from the input sequence.
2. Training data quality: Seq2Seq models require large amounts of high-quality training data to learn effectively. If the training data is not representative of the target domain, the model may not perform well on unseen data.
3. Overfitting: Seq2Seq models are prone to overfitting, which occurs when the model performs well on the training data but poorly on the test data.
4. Handling rare words: Seq2Seq models often have difficulty handling rare words or out-of-vocabulary words, which can negatively impact the quality of the generated output.

The Bleu score of 0.067 in the provided table may be due to several factors such as the complexity of the dataset, the size and quality of the training data, the model architecture, and the hyperparameters used during training. To improve the performance of the Seq2Seq model, one can try using larger models, better quality training data, and more advanced techniques like attention mechanisms, beam search decoding, and ensemble methods.

8 Conclusion

Translation from Hindi to English is a crucial issue that arises in our daily lives, especially for those who are not well-versed in both languages. To address this problem, I attempted to implement an attention-based LSTM based Seq2Seq model. This model is designed to take in Hindi language input and output an equivalent English translation.

To improve the performance of the model, I experimented with different hyperparameters such as the number of hidden dimensions, embedding size, and the number of layers. Through these experiments, I was able to identify the best hyperparameters for the neural machine translation model. This allowed me to obtain accurate translations from Hindi to English with minimal errors.

Despite the success of the model, there is still room for future improvements. One possible area of future work could be to explore the use of attention based transformer to further improve the model's accuracy. Another possible area of improvement could be to incorporate more training data to improve the model's performance on a wider range of Hindi text inputs.

References

[1] d2l contributors, "Encoder-decoder seq2seq for machine translation." `https://d2l.ai/chapter_recurrent-modern/seq2seq.html`, 2023. [Online; accessed 16-April-2023].

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

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

[4] Tensorflow contributors, "Neural machine translation with attention." `https://www.tensorflow.org/text/tutorials/nmt_with_attention`, 2023. [Online; accessed 16-April-2023].