# Competition: Hindi to English Machine Translation System

Deeksha Arora
20111017
{deeksha20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

**Abstract**

In this competition I have implemented Sequence to Sequence based models with LSTM encoder and decoder, Attention model with Bi-LSTM encoder and Attention model with Bi-GRU encoder. Different optimizers like Adam, RMSprop and AdamW are used and AdamW worked best for NMT from Hindi to English. My rank based on Bleu-4 score is 11, 17, 17, 7 and 21 in phase 1, 2, 3, 4 and final phase respectively. Based on Meteor score my rank in different phases is 18, 15, 22, 7 and 13 respectively. Attention based model with Bi-GRU encoder and GRU decoder along with AdamW optimizer gave best results: Bleu-4 macro score: 0.2250, Bleu-4 score: 0.0673 and Meteor score: 0.3451

## 1 Competition Result

**Codalab Username:** D_20111017
**Final leaderboard rank on the test set:** 21
**METEOR Score wrt to the best rank:** 0.345
**BLEU Score wrt to the best rank:** 0.0673
**Link to the CoLab/Kaggle notebook:**
https://colab.research.google.com/drive/12n9aWI69h4ouDeIld0FlEAMaqcpdHCnQ?usp=sharing

## 2 Problem Description

In the competition, we were required to develop a Neural machine Translation model for translation from Hindi to English language. We were free to experiment with different neural models like RNN based models, Sequence-to-Sequence models, Transformer models, etc. It was necessary that the implementation must be carried out using *pytorch*. A train set with 102323 Hindi and English sentence pairs was provided to train the models. The dataset has been curated from the following publicly available sources: https://opus.nlpl.eu/ and https://www.opensubtitles.org/en/search. The models are evaluated on the basis of Bleu-4 score, Bleu-4 macro score and Meteor score on a dev set provided in each phase. The final evaluation is carried out on a separate test set.

## 3 Data Analysis

### 3.1 Analysis of Train Data

The provided dataset consists of 102323 Hindi-English sentence pairs. After careful analysis of the dataset it is found that in 6,102 Hindi sentences contain English words. Also, in many sentence pairs both the sentences are in English. It is clear that these sentence pairs are noisy and need to be removed from the train set. The English translation for some Hindi sentences is quite poor and completely unrelated to the context. The dataset is quite noisy with alot of special characters like ♪, =, ¶, ∼. The dataset lacks consistency since in some Hindi sentences, a full stop (.) is used as end delimiter whereas in some Hindi sentences a purn viram ('।') is used to mark the end of sentence. Also, the hindi sentences have a mix of Hindi(Devanagari) numerals and English numerals. The train dataset contains a lot of movie slangs, a possible reason for this could be that the some sentences pairs are extracted from movie subtitles. Figure 1 gives insight about the length of sentences in the train set.

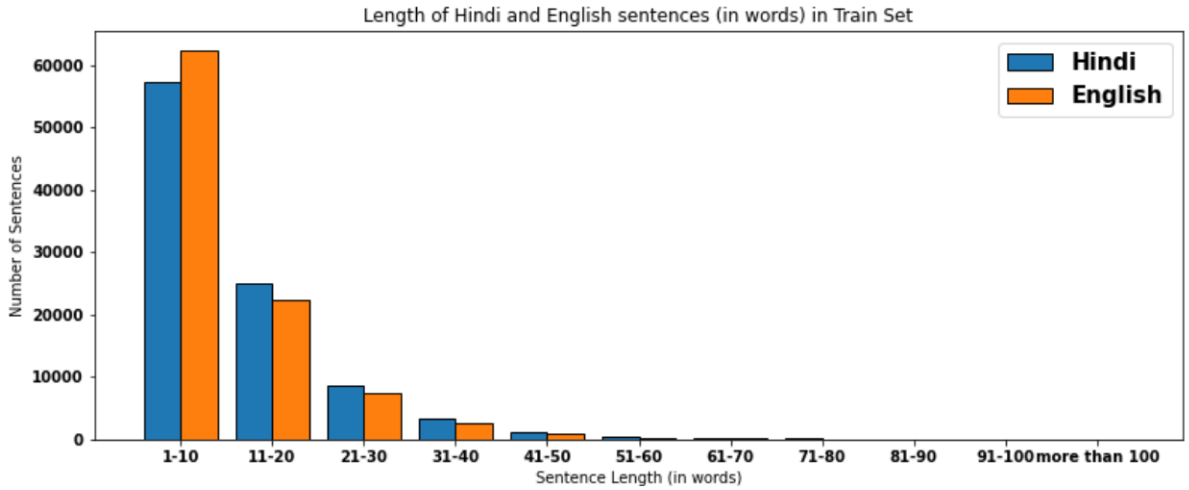


Figure 1: Length of Hindi and English sentences in Train Set

From the above graph it is clear that around 80% sentences have length less than 20 whereas around 98% sentences have length less than 70.
To know about how frequently a word occurs in the train set, a bar graph for Word Frequency Analysis is plotted(figure 2). From figure 2 it is clear that most of the words have frequency 1 i.e. these words occurred only once in the training corpus and there are around only 5000 Hindi words and 8000 English words which have frequency greater than 10.

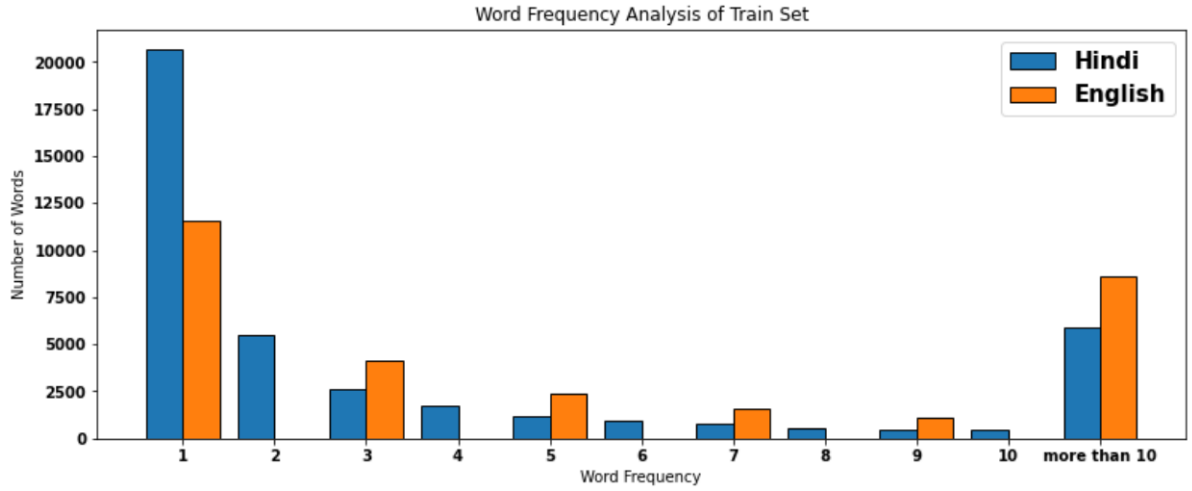


Figure 2: Word Frequency Analysis of Train Set

### 3.2 Analysis of Test Data

1. The test dataset is very similar to the train set in terms of punctuation and other characters.
2. The test dataset doesn't contain any English words.
3. Similar to the train set, the test set has both Hindi(Devanagari) numerals and English numerals.

## 4 Model Description

The models used in different phases of the competition are described in this section.

## 4.1 Phase 1,2 and 3

### 4.1.1 Model Architecture: Sequence to Sequence Model With Single Context Vector Given To The Decoder At Initial Time Step

In the first phase of the competition I implemented a simple Seq2Seq based NMT using the concepts described in the paper "Sequence to Sequence Learning with Neural Networks" by I. Sutskever et al [1]. This model uses an encoder-decoder architecture for NMT from Hindi to English as shown in fig 3 [1]. The idea is to encode the information present in a source sentence into a single vector, called context vector with the help of an encoder. This context vector is then passed to the decoder which decodes it to generate target sentence by generating one word at a time.

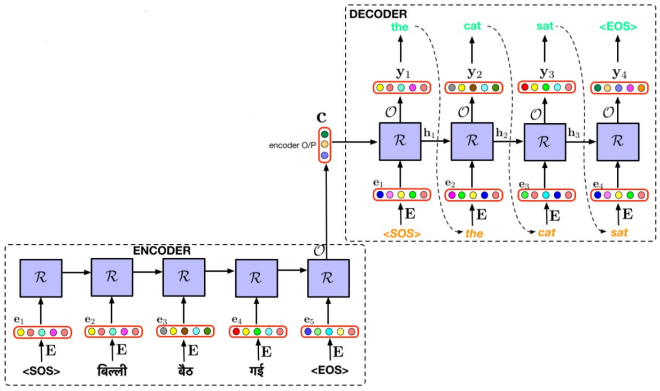


Figure 3: Seq2Seq Model With Single Context Vector Given To The Decoder At Initial Time Step

In phase 1, I implemented the above described Seq2Seq model with bi-layer Uni-directional LSTM for encoder and decoder. I used Adam Optimizer and CrossEntropy Loss function for updation of model's parameters. For tokenization, building vocabulary and creating dataset iterators, Torchtext library was used. *trivial_tokenize() of IndicNLP* and *tokenizer() of spacy* were used for tokenization of Hindi and English sentences. I experimented with different hyper-parameters of this model which are described in section 5.
In phase 2, I experimented with the optimizer used to update the parameters of the model. Taking the best hyper-parameter values from results obtained in phase 1, I trained the models using different optimizers like Adam, AdamW and RMSprop and evaluated their performance on the validation set. Using RMSprop, the Bleu score and Meteor score dropped. However, the model trained using AdamW optimizer showed significant improvement in Bleu score (0.0249) and Meteor score(0.185) as compared to the scores of phase 1. This shows that AdamW works better than Adam and RMSprop Optimizer for NMT from Hindi to English.
In phase 3, I further improved the performance of best model obtained in phase 2 by tuning it's hyperparameters. Also, all the functionalities of Torchtext library were replaced by basic functions provided by *Pytorch*.

### 4.1.2 Key Learning:

In this model the context information is passed to the decoder only in the first time step, thus the context information tends to get lost in later time steps. As the length of the sentences increases, this problem becomes more prominent resulting in decrease in accuracy of translation.

## 4.2 Phase 4 and Final Phase

### 4.2.1 Model Architecture: Sequence to Sequence Model with Attention Mechanism

The performance of seq2seq model described in section 4.1 decreased with increase in sentence length. Therefore, I switched to seq2seq model with attention mechanism where the decoding process is guided by an attention vector which represents which words in the source we should pay the most attention to

[1]Image: Module 3.6, CS779, Dr. A.Modi, IIT Kanpur

in order to correctly decode the next word. To implement this architecture the concepts described by D. Bahdanau et.al.(2016) [2] are followed. I implemented two variations of this model, one using a Bi-LSTM encoder (in phase 4) and the other using a Bi-GRU encoder (in Final phase). Figure 4[2] describes the

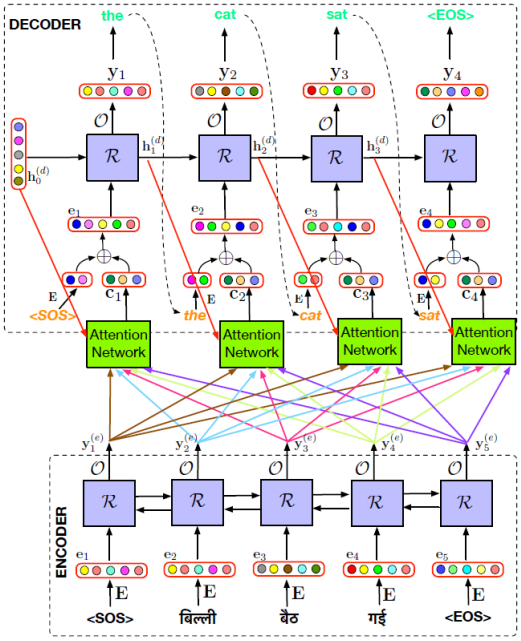


Figure 4: Sequence to Sequence Model with Attention Mechanism

architecture of this model, the encoder is a bi-directional RNN and decoder is a uni-directional RNN. The forward RNN goes over the embedded input from left to right and the backward RNN traverses in opposite direction. The context vectors of forward RNN and backward RNN are concatenated together through a linear layer and then the *tanh* activation function is applied. All the encoder hidden states along with the previous decoder hidden state are passed through an attention network. This attention network provides a context vector $c_i$ for each time-step $i$, given by

$$c_i = \sum_{k=1}^{n} \alpha_{ik} h_k$$

where $\alpha_{ik}$ is the attention score representing the weight of $k^{th}$ word of input sentence at time step $i$, calculated using the formula:

$$\alpha_{ik} = Softmax(e_{ik})$$

where $e_{ik}$ represents the energy state for the $i^{th}$ time step. These energy states are computed by concatenating all the encoder hidden states and the previous decoder hidden state and passing them through a linear layer and then a tanh activation function.

In phase 4, I implemented this architecture using Bi-LSTM encoder(1 neural layer) and a uni-directional LSTM decoder. This model gave better Meteor score as compared to the model used in phase 3, however any significant improvement wasn't observed in Bleu score. Therefore, in final phase I implemented the Attention model using Bi-GRU encoder(1 layer) and uni-directional GRU decoder(1 layer). This model showed significant improvement in both Meteor score and Bleu score as evident from table 1 section 6.

In all the phases of the competition I have used Cross-Entropy Loss function. The Cross Entropy loss function is preferred since it penalizes the probability based on how far the prediction is from the actual expected value, thus yielding less loss for differences tending to 0. The objective function of Cross-Entropy

[2]Image: Module 3.6, CS779, Dr. A.Modi, IIT Kanpur

Loss is given by,

$$\mathcal{L}_\theta = \sum_{i=1}^{|\mathcal{Y}|} -k_i \log(P(\hat{\mathbf{Y}} \mid \mathbf{X}; \theta))$$

where $k_i = 1$, if the predicted label $\hat{\mathbf{Y}}$ is same as the true label, else 0. $\mathbf{X}$ is the input and $\theta$ represents the model parameters.
In all the phases, *Greedy* search is used as the Decoding strategy.

# 5 Experiments

## 5.1 Data Pre-processing

On the basis of data analysis done in section 3.1 the following pre-processing steps were carried out on source and target language:

1. Hindi sentences containing any English word were removed from the train set.
2. Hindi sentences have a mixed use of Devanagari numerals and Western Arabic Numerals. For consistency, all the Devanagari numerals were replaced by Western Arabic Numerals, for example, '१' was replaced with '1'.
3. Punctuation marks and symbols like (, ), -, , , [, ], :, ,", #, /, \, ♪, %, ¶,=, ,∼ were removed from both Hindi and English sentences. However, apostrophe(') and punctuation at end of sentence like ?, !, . were retained. This was done to maintain difference between different type of sentences like affirmative, exclamatory and introductory sentence.
4. Multiple occurrences of period(.) were replaced with a single occurrence.
5. If at the end of a Hindi sentence, a "purn viram ('।')" was not present then it was added. Also, if the end of Hindi sentence is marked using a period(.) then the period was replaced with "purn viram ('।')". This was done to maintain consistency in the training corpus.
6. Figure 1 shows that only 2% sentences have length greater than 70. Therefore to decide the maximum length of sentences I trained the model with maximum length constraint as 50, 70 and 100. Best results were obtained with 70 as the maximum length and therefore in the final phase, sentences with more than 70 words were removed from the train set. This trimming of the train set reduced the amount of padding required to great extent and resulted in faster training.
7. Sentence pairs where either a Hindi or English sentence was missing were removed from the corpus. Example of one such pair in the train set is ' ',' ' where both the sentences in the pair are missing.

## 5.2 Training Procedure and Hyper-parameters:

Section 4 describes about the different models used in each of the phase. The best hyper-parameters for these models were selected by evaluating the Blue score and Meteor score on the validation set. In each phase, the top 2 or 3 models were used to generate predictions for dev set *hindistatements.csv* for submission on *Codalab* platform.
The encoder-decoder architecture, optimizer, loss function and hyper-parameters of the best model of each phase are described in table 1

### 5.2.1 Experiments with Hyper-Parameters

1. **Learning Rate:** I experimented with the learning rates- 0.05, 0.001 and 0.0001. It was observed that models trained using 0.001 learning rate converged faster and gave better performance.
2. **Dropout:** Dropout was used to prevent the model from overfitting by randomly removing units from the neural network for each training batch. A dropout value of 0.5 seemed to work well for all my models.
3. **Optimizer:** I evaluated the performance of Adam, RMSprop and AdamW optimizer by keeping the model architecture and other hyper-parameters same. RMSprop gave lowest scores and AdamW gave best scores, therefore I used AdamW optimizer for all the models from phase 2.

| Model Description | Phase 1 | Phase 2 | Phase 3 | Phase 4 | Final Phase |
|---|---|---|---|---|---|
| Encoder | LSTM | LSTM | LSTM | Bi-LSTM | Bi-GRU |
| Decoder | LSTM | LSTM | LSTM | LSTM | GRU |
| Attention | No | No | No | Yes | Yes |
| #Layers | 2 | 2 | 2 | 1 | 1 |
| Loss | CrossEntropy | CrossEntropy | CrossEntropy | CrossEntropy | CrossEntropy |
| Optimizer | Adam | AdamW | AdamW | AdamW | AdamW |
| LR | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Dropout | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Emb. Dim. | 250 | 250 | 250 | 250 | 256 |
| Hidden Dim. | 1024 | 1024 | 512 | 512 | 512 |
| HVS | 20004 | 25000 | 41006 | 41006 | 38361 |
| EVS | 20004 | 25000 | 28207 | 28207 | 28071 |
| Batch Size | 32 | 32 | 32 | 32 | 50 |
| Decoding | Greedy | Greedy | Greedy | Greedy | Greedy |
| Epochs | 30 | 30 | 30 | 40 | 25 |
| Train Time | 4 | 5 | 5 | 7 | 5 |

Table 1: Description of Best Model of each Phase

4. **Number of Neural Layers:** For the same hyper-parameters, I trained seq2seq based LSTM model for 1, 2 and 4 neural layers. With 1 neural layer the model didn't learn well. With 4 neural layers, the training time increased significantly, also, the model's performance decreased maybe because the model started suffering from vanishing gradient problem. Thus, I preferred to used encoder/decoder with 2 neural layers.
5. **Batch size:** Based on the embedding size and hidden dimensions, batch size varied across models. In general, I preferred batch size of 32 because a batch size of more than 32 mostly resulted in memory overflow and a batch size of less than 32 slowed down the training process.
6. **Number of Epochs:** While training the models, a check was maintained on training loss and validation loss. If the model's performance on validation loss started decreasing then the training was stopped. The models were saved after every 5 epochs and performance of these models was evaluated on validation set to decide the best value of "number of epochs".
7. **Embedding Size and Hidden Size:** Larger values for embedding and hidden dimensions gave better performance, however this also required more memory and training time. After experimenting with the following combinations for embedding and hidden size: (128,256), (128, 512), (250, 512), (256, 1024) I observed that (250, 1024) worked well for seq2seq with 2-layer LSTM model and (250, 1024) worked well for seq2seq with Bi-LSTM/GRU Attention model.
8. **Vocabulary size:** Based on minimum frequency of words(k), vocabularies were created for k= 1 and 2. Both the models i.e. Simple unidirectional LSTM and Bi-LSTM with attention mechanism gave good results for k=1, probably because for k=2 the model encountered too many <unk> token which resulted in poor learning.
9. **Tokenizers:** trivial_tokenize() of IndicNLP worked well for tokenization of Hindi sentences. For English sentences, tokenization was carried out such that the clitics are not separated from the stem word, for example, if the word is "they'll" then its token should be "they'll" and not "they" and "'ll" or "theyll". Regexp tokenizer of NLTK handled the clitics correctly and therefore I preferred it for tokenization of English text.

# 6 Results

**Abbreviations for Model Architecture**
**LSTM-Adam:** Seq2Seq-LSTM model with 2-layer unidirectional encoder and decoder where the context

vector is given as input to the decoder only in the initial time step. Adam Optimizer is used.
**LSTM-AdamW:** Architecture same as LSTM-Adam but this time AdamW optimizer is used.
**Attention-GRU:** Seq2Seq based Attention model with Bi-GRU 1-layer encoder and uni-directional 1-layer GRU decoder.

## 6.1 Performance on dev set

| Model | Phase | Bleu-4 macro score | Bleu-4 score | Meteor score | Rank based on Bleu-4 score | Rank based on Meteor score |
|---|---|---|---|---|---|---|
| LSTM-Adam | Phase 1 | - | 0.008 | 0.173 | 11 | 18 |
| LSTM-AdamW | Phase 2 | 0.1470 | 0.0248 | 0.1848 | 17 | 15 |
| LSTM-AdamW | Phase 3 | 0.1400 | 0.0227 | 0.1641 | 17 | 22 |

Table 6.1 : Results on dev set

## 6.2 Performance on test set

In the final phase, I submitted predictions generated using the Seq2Seq-Attention model with Bi-GRU 1-layer encoder and a uni-directional 1-layer GRU decoder. The architecture of this model is described in table 1 under the column named "Final Phase".

| Model | Epochs | Bleu-4 macro score | Bleu-4 score | Meteor score | Rank based on Bleu-4 score | Rank based on Meteor score |
|---|---|---|---|---|---|---|
| Attention-GRU | 20 | 0.2187 | 0.0641 | 0.3413 | - | - |
| **Attention-GRU** | **25** | **0.2250** | **0.0672** | **0.3451** | **21** | **13** |

Table 6.2 : Results on test set

## 6.3 Analysis of Best Model of Final Phase

From table 6.1 and table 6.2, it is clear that **"Seq2seq-Attention model with Bi-GRU 1-layer encoder and Uni-directional 1-layer GRU decoder"** gave best results. The hyper-parameters of this model are specified in table 1 under the column "Final Phase". The training loss for the model at each time step is shown in figure 5. From the plot it can be observed that initially the training loss decreased with speed and later the loss decreased at a steady rate. The consistent performance of Attention models

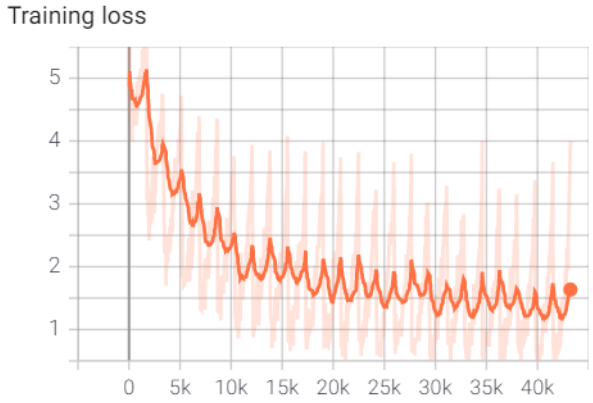


Figure 5: Plot for Training Loss at each time step plotted using Tensorboard

with increasing sentence length motivated me to implement models based on attention mechanism. The model by itself learns the alignment between input and output words. Attention weights for a sample test sentence are shown in figure 6. The nice alignment between input and output words is visible from this heatmap.

# 7 Error Analysis

1. Bi-GRU with Attention mechanism gave best results on both the dev set and test set. This qas expected since both these datasets are quite similar.
2. The translation obtained using simple seq2seq model with uni-directional LSTM contained a lot of repetition of a particular word. I observed that such translations were obtained mostly for longer sentences. A possible way to rectify this problem is to use scheduled sampling instead of teacher forcing. Using Beam search may also rectify this problem.
3. When models were trained with restricted dictionary size, it was observed that <unk> token occurred quite frequently. Removing these <unk> token from the translation resulted in improved Bleu score, however the Meteor score remained unaffected.
4. The translation accuracy of Attention based model decreased when the sentence length exceeded 30-35. In such situation the model repeatedly predicted same words in the latter part of the sentence, however the initial predictions seemed good. Beam search and scheduled sampling may help in improving model's accuracy for longer sentences.
5. Models trained using Attention mechanism tend to have better Meteor score as compared to Simple seq2seq models. In phase 4, for Bi-LSTM with Attention model I observed lower Bleu score as compared to bi-layer LSTM without Attention, however the Meteor score forn model was better.
6. Figure 6 shows the attention weights for translation of a Hindi sentence from test set. Prediction using Attention with GRU model is "it's just a dream dream." and the correct translation for this sentence is "it's just a stupid dream." The model failed to predict the token "stupid". From the heatmap it can be observed that at time step 4 the model's focus should be on $4^{th}$ word, however the model paid more attention to $5^{th}$ word resulting in predicting the token "dream" twice.

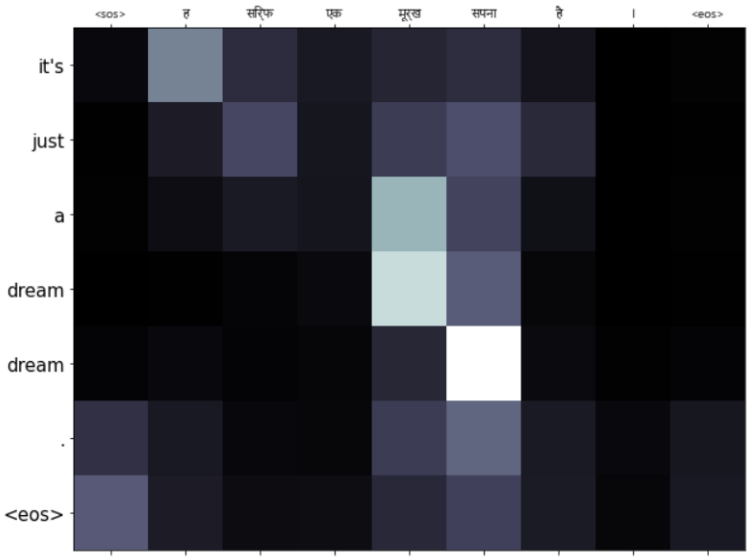


Figure 6: Heatmap showing attention weights for a Hindi sentence from Test Set

# 8 Conclusion

The performance of Simple Seq2Seq models increases to a great extent when implemented using Attention mechanism. As compared to Simple Seq2Seq models, Attention models can handle long sentences quite efficiently, however when the sentence length increases beyond 35 even the attention model results in inaccurate translations. The final model can be further improved by using scheduled sampling instead of teacher forcing and by using Beam search as the decoding strategy instead of Greedy search. To obtain the best results, Transformer based architectures can be used instead of Sequence-to-Sequence based architectures.

# References

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

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