Competition: Hindi to English Machine Translation System

Akash Halayyanavar
20111006
{akashh20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

The task given in the competition was to implement a Neural Machine Translation (NMT) model to translate from Hindi to English. In the first phase, a simple Sequence to Sequence model is used. In the second phase, a Sequence to Sequence model with attention is used. In the third phase, the model used in the second phase with different parameters is used. For the final phase, the Sequence to Sequence model with attention with the best hyperparameters obtained is used. The Evaluation Metrics and Rank for the final phase are as follows, BLEU Score: 0.08877206970104597, METEOR Score: 0.3461518177511583, and Rank:10.

1 Competition Result

Codalab Username: A_20111006
Final leaderboard rank on the test set: 10
METEOR Score wrt to the best rank: 0.3461518177511583
BLEU Score wrt to the best rank: 0.08877206970104597
Link to the CoLab/Kaggle notebook:
`https://colab.research.google.com/drive/1BAaQ6Fn6hn2wSv3Cmn2wDt-zzXnKuyBq?usp=sharing`

2 Problem Description

The task of this competition is to implement a Neural Machine Translation (NMT) model to translate from Hindi to English. A Training set consisting of Hindi and English sentences was provided for the same. The competition consisted of 3 phases and a final phase. For each of the phases, we were supposed to implement a neural model. The NMT used is encoder-decoder architecture. The encoder part encodes the source sentence in the form of some representation which is then passed on to the decoder to predict the target sentence using the encoded representation. The encoder and Decoder can be any one of neural architectures such as LSTM, GRU, Transformer, etc. Evaluation metrics were provided such as BLEU score, METEOR score, etc so that we can evaluate our model on the validation set. Dev sets were released for each of the phases, which were supposed to be translated using our model, and then submit the results on Codalab.

3 Data Analysis

The dataset provided consists of 102,322 Hindi-English sentence pairs. The dataset has been curated from the following publicly available sources: `https://opus.nlpl.eu/` and `http://www.opensubtitles.org/`. The data consisted of some noise such as currency symbols, musical symbols, numbers, English words in Hindi sentences etc. And there were also many words that occured only once in the training set. Hence, such noise is removed, and then the dictionary is formed. The resulting dictionary size for Hindi sentences and English sentences for the entire training set are 21,672 and 19,009 words respectively. Also, there were many sentences which were too lengthy, therefore I did a frequency analysis of the sentences with respect to length. And then chose the maximum length for the sentences. The

sentences whose lengths were greater than that of the maximum length are filtered out.
The test data consisted of 24,102 Hindi sentences. The dictionary size of the test set after removing the noise form the sentences is 19,560 words. The test set dictionary consists of 12,617 words that are present in the training set dictionary and it consists of 6,943 new words. Since majority of the words present in the train set are also present in the test set, the trained model predicts reasonably well. An interesting observation was that for few sentences in the training set, the target sentence (English) was not the rightly translated version of the hindi sentences. Few such examples are shown in the figure 1.

```
32172,मेरे surprise से मज़ा आया?,Simple is good.
32178,एक करने के लिए.,Thank you.
25258,जैक!,"Oh, I'm so sorry."
102313,"बस, बस।","There, there."
102315,- आप क्या कह रहे हैं,"- Yes, I saw a ghost"
102320,तुमनेमेरी पूरी ज़िंदगी गया .,Since I was a boy.
```

Figure 1: Noise in the training data

4 Model Description

4.1 Phase 1

In the first phase, I implemented a simple Sequence to Sequence model [1]. This Sequence to Sequence model is an encoder-decoder architecture. The encoder and decoder used in this phase is unidirectional LSTM (Long Short Term Memory). Simple RNN has a hidden state that stores the information of the data it has seen till then. Simple RNN's suffer from vanishing gradient problems for long sequences, and hence is not suitable to preserve the long term dependencies. LSTM preserves the long term dependency using the gated mechanism. Each LSTM module consists of two states, a hidden state and a cell state. The cell state acts like a short term memory and keeps the relevant information that is needed. At each module, the information is added or removed through the gates [2]. During this phase, I trained this model multiple number of times by varying the number of hyperparameters such as, number of layers, hidden dimension, embedding size, dropout, teacher forcing ratio etc, and then noted down the hyperparameter values for the best model. The best model obtained consisted of two layers and is as shown in the Figure 2.

4.1.1 Key Observations

1. Increasing the number of layers to 2 from 1 worked well. But further increasing the number of layers overfits the model.
2. A very high number of hidden units overfits the model and very less number of hidden units underfits the model.

4.2 Phase 2

In the second phase, I implemented Sequence to Sequence model with Attention mechanism [3]. First, I implemented this architecture, where in both the encoder and decoder are built using unidirectional LSTM. The results were good when compared to the model in phase 1. Then, I implemented the architecture using bidirectional GRU for the encoder and unidirectional GRU for the decoder. GRU works similar to LSTM, but has simple model than LSTM. The GRU just uses the hidden state and unlike LSTM, it does not use an extra cell state to keep the track of relevant information. It directly uses the hidden state to keep the relevant information. GRU just uses two gates to add or remove the relevant information [2]. Using bidirectional GRU encoder with attention mechanism gave very good results as compared to unidirectional encoder model with attention mechanism. The attention mechanism helps the decoder to pay attention to those input parts that are relevant to decode the

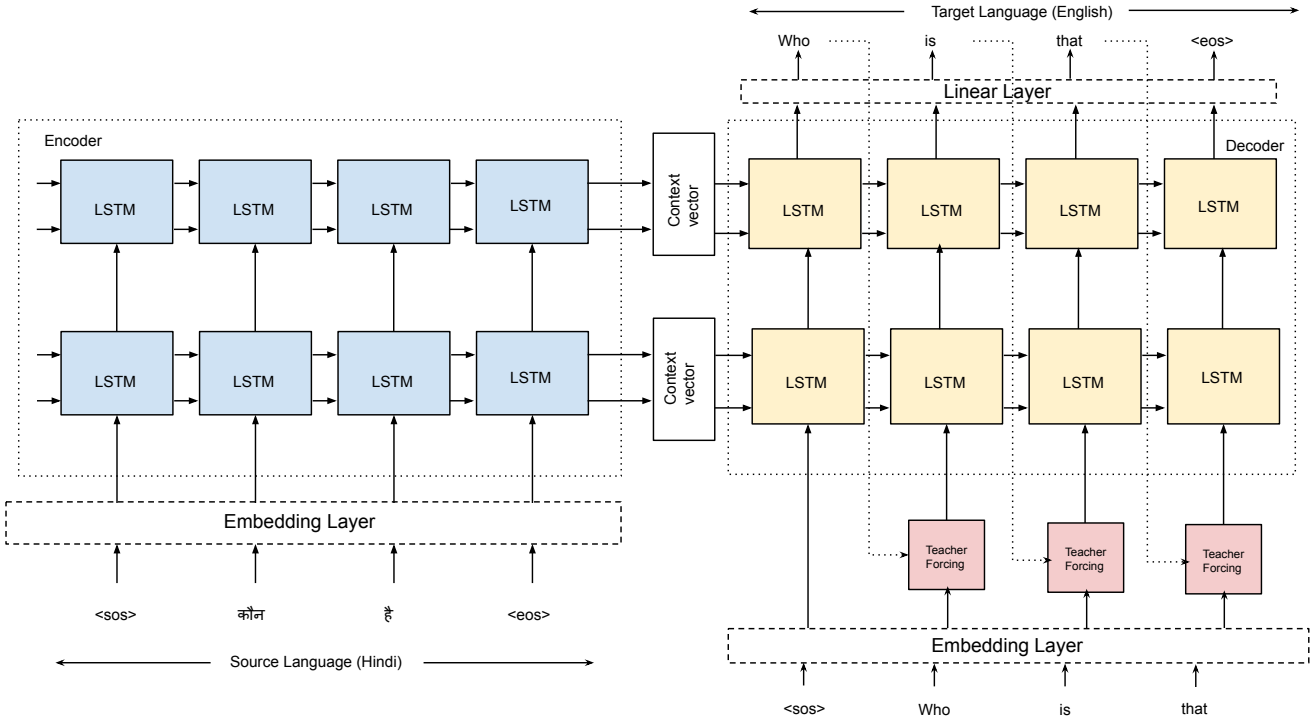

Figure 2: Sequence to Sequence LSTM Architecture

particular word. The attention network takes in all the hidden states of the encoder at each step along with the previous hidden state of the decoder and outputs an attention vector [3]. Then, the weight of each source sentence is calculated using the attention vector which helps the decoder to concentrate on the those parts of the input sequence that helps in the prediction of output at that time step.
In this phase the best model was the Seq2Seq model with attention mechanism, where the encoder is bidirectional GRU and the decoder is unidirectional GRU. The model used in this phase is shown in Figure 3.

4.2.1 Key Observations

1. The Seq2Seq model with attention mechanism gave good results when compared to simple sequence to sequence neural network.
2. Usage of bidirectional encoder Seq2Seq model with attention mechanism gave very good results when compared to unidirectional encoder Seq2Seq model with attention mechanism. This is because, the word appearing earlier may depend on the words appearing later in the sentence. The usage of BiGRU encoder outputs two context vectors, one is the context obtained by moving in the forward direction, and the other by moving in the backward direction. The context obtained by moving in reverse direction i.e., from future to past will also help to predict the next word based on the future context.

4.3 Phase 3

In the third phase, I implemented the Seq2Seq model with attention mechanism using Bidirectional LSTM as encoder and unidirectional LSTM as decoder. The performance of the model was good but was quite lower when compared with the same architecture using GRU module. Then, I trained the Bidirectional GRU model with attention mechanism by increasing the number of layers. The model's performance decreased with increase in the number of layers. Hence, the best model I found was again the same model that was used in the Phase 2 i.e., Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq model with attention mechanism.

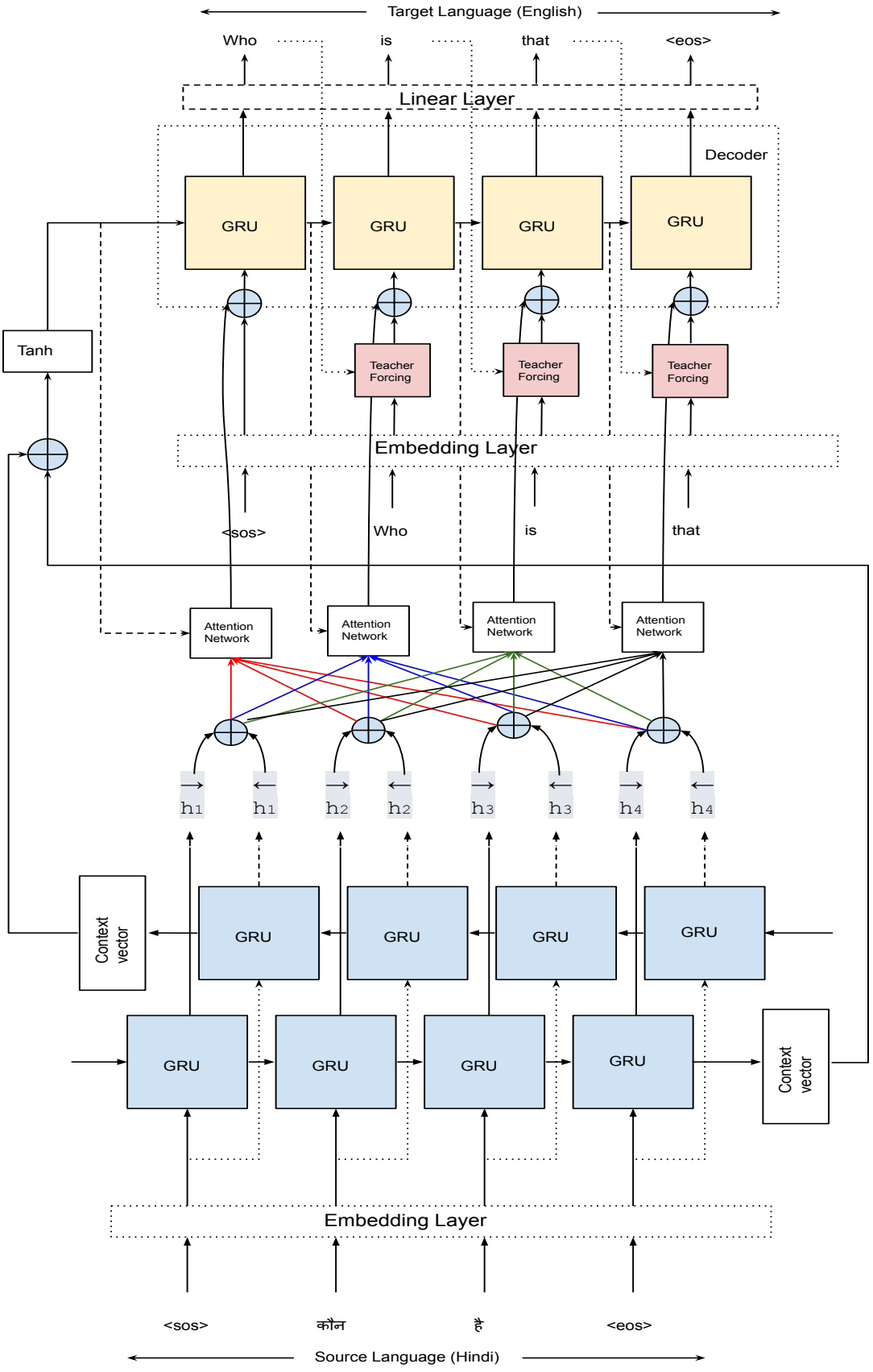

Figure 3: Sequence to Sequence GRU Architecture with Attention Mechanism

4.3.1 Key Observations

1. Seq2Seq attention model with BiGRU encoder trains faster when compared with Seq2Seq attention model with BiGRU encoder. This is because of the simple model of GRU that just uses only one state and two gates. On otherhand the LSTM is quite complex model, it uses two states and four gates.
2. For same set of hyperparameters the Seq2Seq attention model with BiGRU encoder gives better results than that with BiLSTM encoder model. For small dataset the performance of GRU is slightly better than LSTM [4].

4.4 Final Phase

In the final phase, I used the best model obtained till now i.e., the Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq model with attention mechanism, and tuned the model for various hyperparameters.

4.4.1 Key Observations

1. The model performs better for single layer with 512 hidden dimension, when compared to that with two layers and 256 hidden dimension.
2. The model works better embedding size 128. As the dictionary size is smaller, increasing the embedding size gives sparse representation of word embeddings. Very small embedding size results in dense representation. Hence the embedding size 128 worked better in comparison to others.

4.5 Model objective (loss) functions

I have used Cross Entropy loss function during the entire competition. The Cross Entropy Loss function is given as,

$$\text{CE}_{\text{Loss}} = -\sum_{l=1}^{L} \delta_{y=l} \log(p(y = l \mid x; \theta))$$

The cross entropy loss function uses softmax function, which gives the probability distribution over all the words in the dictionary. The loss is then calculated using this probability distribution and the true output. The Cross Entropy loss function uses the ignore index as the index of <eos>, which ignores the loss for the words after the end of sentence.

4.6 Inference

For the first three phase of the competition I used greedy search decoding strategy. The greedy search strategy is very simple strategy which just picks the word with highest probability. Though choosing the most probable word may work good for the current time step, but for the full sentence prediction it is the sub-optimal choice [5]. Beam Search is another decoding strategy that considers multiple alternatives for an input sequence at each timestep based on conditional probability [5]. In the final phase I tried to implement the Beam search strategy with Beam width=5, but the model's accuracy didn't improve and also it was taking very long time for the prediction. Hence, I switched back to the Greedy decoding strategy.

5 Experiments

5.1 Data Preprocessing

The data is read from the provided training dataset and is then split into 80% train data and 20% test data.

Hindi Sentences: For Hindi sentence I first detokenized the sentences, so as to remove any extra

spaces from the sentences if present. For tokenization, I used "INDIC NLP" tokenizer along with "regex". The tokens obtained from the INDIC NLP tokenizer are then selected using the regular expression. The tokens are added to the dictionary if their frequency ≥ 2. At first, I used the regex to select only the hindi words. Then, later after the analysis of the dataset I found even many english words in the hindi sentences, also the punctuations at the end of sentences would make sense, hence I changed the regex to consider the english words and punctuation marks. Two dictionaries word to index and index to word are formed.

English Sentences: For English sentences tokenization I earlier used Spacy tokenizer along with "regex". The tokens obtained from the Spacy tokenizer are then selected using the regular expression. At first, I used the regex to select only the english words. The tokens are added to the dictionary if their frequency ≥ 2. The inclusion of punctuations makes more sense, and hence I changed the regex to even consider the punctuation marks. Two dictionaries word to index and index to word are formed. Later there was some problem with spacy, hence I shifted to NLTK tokenizer.

Max Sentence Length: To find the maximum length for each of the languages, I made frequency analysis based on their sentence length. Then sorted the frequency counts in the decreasing order and then chose the lengths for which the number of sentences are ≥ 300. Then found the maximum length among these lengths. This gives the maximum length for both Hindi and English respectively. Then the Max Sentence Length is found by taking the maximum of these maximum lengths.

Sentence Filtering: The sentences with length greater than the Max sentence length are removed.

Vector Representation of Sentences: The filtered sentences are then tokenized and each token is assigned an index according to the word to index dictionary. Each sentence begins with <sos> token and ends with <eos> token and hence the corresponding indices are assigned to them. The words which do not appear in the dictionary are assigned with the index corresponding to <unk> token. The sentences whose lengths are less than the maximum length are padded with the <pad> token and hence the corresponding index is assigned to it after the <eos>.

5.2 Training procedure

For optimization I used Adam Optimizer. I used 0.001 as initial learning rate. I also tried with other learning rates such as 0.01, 0.005 etc, but I found 0.001 to work better than others. The number of epochs and training time for different model used in each of the phase is shown in table 1.

Phase	**NMT model**	**Epochs**	**Training Time**
1	LSTM Seq2Seq model	30	4 hrs
2	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	10	2 hrs 30 min
3	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	20	4 hrs
Final	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	15	3 hrs

Table 1: Number of Epochs and Training Time for different models

5.3 Hyperparameters

For all the model, the dropout of 0.5 is used. I tried the models with different teacher forcing ratio such as 0.5, 0.75, 1 etc, and found that the model worked best for teacher forcing ratio 1. The remaining hyperparameters for different models are shown in the table 2

Phase	NMT model	Layers	Embedding size	Hidden dimension	Batch Size
1	LSTM Seq2Seq model	2	256	512	32
2	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	1	128	512	32
3	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	1	128	512	32
Final	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	1	128	512	32

Table 2: Hyperparameters for different models

Phase	NMT model	BLEU Score	METEOR Score	Rank on leaderboard
1	LSTM Seq2Seq model	0.000521	0.109010	39
2	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	0.032387	0.230141	11
3	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	0.031479	0.222168	11

Table 3: Results for different models

6 Results

The best performing model is found to be the Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model. In the first phase the simple unidirectional LSTM encoder-decoder SeqSeq model was implemented. Then, in the second phase the usage of Attention mechanism gave good results as it allows the decoder to pay attention to the relevant parts of the input sequence. Further using the Bidirectional GRU encoder, wherein the context vector is obtained in both, the forward and the reverse direction, helps the decoder to the predict the word using both the past context and future context. Hence the Bidirectional GRU model with attention mechanism works well. The embedding size of 128 was found to be the best. Increasing the number of embedding size will result in sparse word embedding representation and decreasing the embedding size will result in more dense representation. Hence the embedding size of 128 works better. The model performed the best for 1 layer and 512 hidden dimension. Increasing the number of layers or hidden dimension overfits the model. The best performing model was trained for 15 epochs. The result of different models on dev set is shown in Table 3. The result of different models on test set is shown in Table 4.

7 Error Analysis

The BLEU (Bilingual Evaluation Understudy) metric uses modified n-gram precision to compare the candidate translation with multiple reference translation [6]. The BLEU compares the n-gram of the candidate translation with n-gram of the reference translation to count the number of matches [7]. The BLEU score penalizes if the translations ate too short. For the earlier phases, I implemented the model such that, during the prediction if there appears any word that is not present in the dictionary then, the model would just skip that word instead of placing <unk> token. This resulted in shorter translations for few sequences and hence there was some reduction in the BLEU score. So later I included the <unk> tokens for the predicted words which do not occur in the dictionary. So, the BLEU score

Model Number	NMT model	BLEU Score	METEOR Score	Epochs	Sequence Length	Rank
1	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	0.088772	0.346151	15	38	10
2	Bidirectional GRU encoder - Unidirectional GRU decoder Seq2Seq Attention model	0.087828	0.350421	10	24	-

Table 4: Results of different models on the test data

takes the translated sequence length into consideration. METEOR score is another evaluation metric for machine translation. METEOR score is based on the the harmonic mean of unigram precision and recall [8]. BLEU score doesn't take meaning into consideration. On other hand, METEOR uses stemming and synonymy matching along with the standard exact word matching and hence is better than BLEU evaluation metric [8].

Few reasons for as to why the models are not perfect:

1. The provided dataset is small and also contains some noise in it. After the removal of noise the dictionary size becomes very small and hence the trained model is not perfect to translate every sentence in the test set reasonably well.
2. The anlaysis made earlier shows that, there are few new words present in the test set which were never seen during the training phase. So the model won't be able to predict well for such words.
3. Though I included the punctuation marks in the dictionary to make the model predict more accurately, the dataset consists of many useless punctuations in between the sentences that hinders the model's performance.

8 Conclusion

The key findings from the models implemented during this competition are:

1. Usage of bidirectional encoder instead of unidirectional encoder provides better results. Since, the bidirectional encoder provides the context vectors in both the directions, one obtained by moving in the forward direction and other obtained by moving in the backward direction. This helps the decoder to predict the word more accurately by looking at both context vectors.
2. Seq2Seq models with attention mechanism performs better. This is because the attention mechanism helps the decoder to predict the word better by using the relevant information of the source sentence rather than using all the information.
3. Usage of bidirectional encoder Seq2Seq model with attention mechanism provided the best results, as it has the advantage of both, bidirectionality and attention mechanism.

Some of the possible Future directions are:

1. Using Beam search, Sampling, Top-K Sampling, Top-p Neuclues Sampling decoding strategies instead of Greedy search.
2. Using the pretrained word embeddings such as word2vec embedding.
3. Using Scheduled Sampling method instead of Teacher Forcing.
4. Using Transformer models or a hybrid of Seq2Seq RNN and Transformer model instead of Seq2Seq RNN models.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[2] M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation."

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[4] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pp. 98–101, IEEE, 2020.

[5] R. Khandelwal, "An intuitive explanation of beam search."

[6] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[7] R. Khandelwal, "Bleu — bilingual evaluation understudy."

[8] Wikipedia, "Meteor."