Competition: Hindi to English Machine Translation System

Abhas Kumar
20111001.
{ababhas20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

In the NLP competition, I started with the LSTM network in phase 1 with ADAM optimizer and obtained METEOR and BLUE score 0.245 and 0.0066 respectively with rank 13. In phase 2, I used Bidirectional GRU with ADAM optimizer and obtained METEOR score=0.356 and BLEU Score=0.0687 with rank 4, then in the next phase, I implemented uni-directional GRU with SGD optimizer and Negative log-likelihood Loss function and obtained METEOR score=0.200, BLEU Score=0.0247 and rank=13. In the final phase, the Transformer model was implemented by me to obtain BLEU and METEOR Scores of 0.1030 and 0.389 respectively, ranked at 7.

1 Competition Result

Codalab Username: A_20111001
METEOR Score wrt to the best rank: 0.389
BLEU Score wrt to the best rank: 0.1030
Link to the CoLab notebook: Final Phase Notebook

2 Problem Description

I have been employed in a tech team of a mass-media conglomerate that publishes news and magazines in Hindi. The company wants me to develop a neural machine translation(NMT) system that could automatically translate the articles/editorials written in Hindi to English. A competition has been organized for developing a Hindi-English MT system. From the course CS799, I remember different types of neural models, e.g., RNN based models, sequence-to-sequence models, transformer models. So I plan to use these to implement an NMT system for translation from Hindi to English [1].
****Note** I have intentionally used the same problem statement as provided in the competition doc to present this report as a complete story.

3 Data Analysis

1. The training data had **102322** Hindi-English sentence pairs. Test data for phase 1 to 3 had **5000** Hindi sentences while the final phase test data had **24102** Hindi sentences. Few of the length and size based analysis on training and final phase test data are shown here in Table 1 and Figure 1
2. Although we see a huge difference in (Table 1) maximum length of Hindi and English sentence between train and test sets, but the average sentence length for both Hindi and English is almost the same for both data sets.
3. Number of Unique tokens(Vocab Size) for Hindi Sentence is **45835** and for English sentences is **31373** for the training set. For the Final Test set unique number of tokens for Hindi and English sentences are **20019** and **16898** respectively.
4. In training set, maximum length of unique Hindi token is 43, English token is 30. In test set 1, maximum length of unique token is 27, in test set 2 length is 29, and in final test set maximum length is 43.
5. In training dataset, I have found lot of **"sanskrit words"** e.g. **"Hitkarak"** which in hindi means **"Hitkari"**

[1] **NLP Challenge Document**

Sentence Property	Train set		Final Test set	
	Hindi	English	Hindi	English
Average Length	51.51	52.50	51.36	52.43
Maximum Length	2088	1880	597	662
Minimum Length	1	1	2	1
Maximum tokens	444	318	597	458

Table 1: Some of the length based properties for both Hindi and English Sentences in Training and Final Test datasets, where "Maximum tokens" field gives maximum number of tokens present in any Hindi or English sentence.

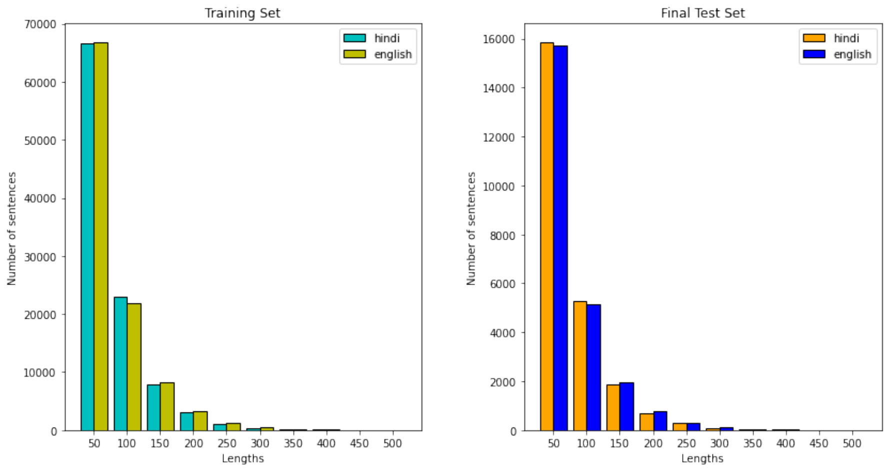
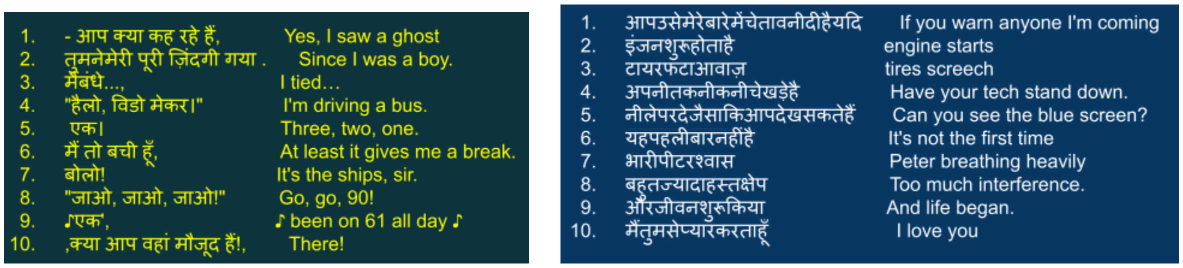


Figure 1: **Left**: Total Number of sentences having length less than equal to 50, 100, 150,..., in Training set. **Right**: Total Number of sentences having length less than equal to 50, 100, 150,..., in Final Test set.

3.1 Noise:

1. **5697** out of 102322 Hindi sentences in training dataset[2] had English words.
2. **643** sentence pairs where there in which Hindi-Sentence = English-Sentence, i.e both source and target was in English language.
3. Absolutely Wrong/incomplete translation of Hindi sentence were present in many sentence pairs. Examples are shown in Figure 2(Left)



1.	- आप क्या कह रहे हैं,	Yes, I saw a ghost
2.	तुमनेमेरी पूरी ज़िंदगी गया .	Since I was a boy.
3.	मैंबंधे...,	I tied...
4.	"हैलो, विडो मेकर।"	I'm driving a bus.
5.	एक।	Three, two, one.
6.	मैं तो बची हूँ,	At least it gives me a break.
7.	बोलो!	It's the ships, sir.
8.	"जाओ, जाओ, जाओ!"	Go, go, 90!
9.	♪एक',	♪ been on 61 all day ♪
10.	,क्या आप वहां मौजूद हैं!,	There!

1.	आपउसेमेरेबारेमेंचेतावनीदीहैयदि	If you warn anyone I'm coming
2.	इंजनशुरूहोताहै	engine starts
3.	टायरफटाआवाज़	tires screech
4.	अपनीतकनीकनीचेखड़ेहै	Have your tech stand down.
5.	नीलेपरदेजैसाकिआपदेखसकतेहैं	Can you see the blue screen?
6.	यहपहलीबारनहींहै	It's not the first time
7.	भारीपीटरश्वास	Peter breathing heavily
8.	बहुतज्यादाहस्तक्षेप	Too much interference.
9.	औरजीवनशुरूकिया	And life began.
10.	मैंतुमसेप्यारकरताहूँ	I love you

Figure 2: **Left**: 10 examples wrong English translation of a Hindi sentence. **Right**: 10 examples where words in a Hindi sentence were not properly separated.

4. Many Hindi sentences where not properly space separated(approx **300** sentences), some of them are shown in Figure 2 (Right).
5. Although the Hindi sentences were quite noisy, English sentences were atleast free from adulteration by Hindi words.

[2]Dataset has been curated from the this website

4 Model Description

4.1 Phase 1:

4.1.1 Overview

Model's Architecture: 2 Layer Uni-LSTM encoder, 2 Layer Uni-LSTM decoder, ADAM optimizer and Cross Entropy Loss function. Model's METEOR Score=**0.245**, BLEU Score=**0.0066**.
In the first phase, I started understanding the general concepts of NMT by implementing the model from **Sequence to Sequence Learning with Neural Networks** paper[1]

4.1.2 LSTM Encoder and Decoder

Here my encoder is a 2 layer LSTM(although [1] uses a 4-layer LSTM). 2 layer equations to LSTMs for the encoder can be given as:

$$(h_t^1, c_t^1) = \textbf{EncoderLSTM}^1(e(x_t), (h_{t-1}^1, c_{t-1}^1))$$
$$(h_t^2, c_t^2) = \textbf{EncoderLSTM}^2(h_t^1, (h_{t-1}^2, c_{t-1}^2))$$

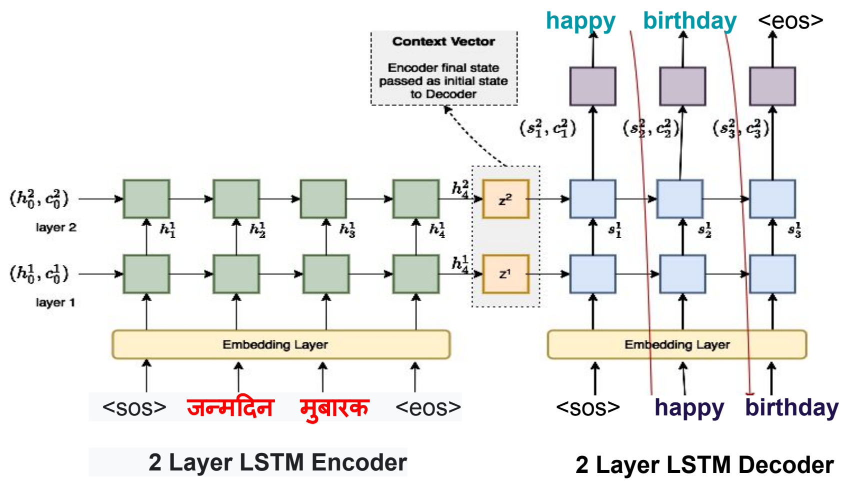


Figure 3: An example translation[3], where a Hindi sentence, is passed to a 2 Layer LSTM Encoder to produce 2 context vectors, which are used by the decoder to English translation "Happy Birthday".

The Decoder outputs single token per time-step[4], first layer receives a hidden and cell state from the previous time-step, (s_{t-1}^1, c_{t-1}^1), and passes it through the LSTM(Figure 3) with the current embedded token, y_t, to produce a new hidden and cell state, (s_t^1, c_t^1). Next layer will use the hidden state from the layer below, s_t^{l-1}, and the previous hidden and cell states from their layer, (s_{t-1}^l, c_{t-1}^l).

$$(s_t^1, c_t^1) = \textbf{DecoderLSTM}^1(d(y_t), (s_{t-1}^1, c_{t-1}^1))$$
$$(s_t^2, c_t^2) = \textbf{DecoderLSTM}^2(s_t^1, (s_{t-1}^2, c_{t-1}^2))$$

4.2 Phase 2:

***Note:** One negative of the Phase 1 model was that the decoder was trying to put lots of information into the hidden states. While decoding, the hidden state require to contain information about the whole of the source sequence, as well as all of the tokens that have been decoded so far. I tried to work on this issue with the help of papers[2, 3] to create a better model shown in Figure 4(Left) in phase 2.

[3]Image source
[4]Totorial on Sequence to Sequence Learning with Neural Networks paper

4.2.1 Overview:

Model's Architecture: Bidirectional-GRU Encoder, GRU Decoder with Attention mechanism, ADAM optimizer and Cross Entropy Loss function.Model's METEOR score=**0.356**, BLEU Score=**0.0687**.

4.2.2 Encoder and Attention:

I used single layer GRU in the Encoder, which was a bidirectional GRU. With a bidirectional GRU, we have two GRUs in each layer. A forward GRU going over the embedded sentence from left to right and a backward GRU from right to left . We now have[5]:

$$h_t^{\rightarrow} = \mathbf{EncoderGRU}^{\rightarrow}(e(x_t^{\rightarrow}), h_{t-1}^{\rightarrow})$$
$$h_t^{\leftarrow} = \mathbf{EncoderGRU}^{\leftarrow}(e(x_t^{\leftarrow}), h_{t-1}^{\leftarrow})$$

I only passed an input (embedded) to the GRU, which tells PyTorch to initialize both the forward and backward initial hidden states ($h_0^{\rightarrow}$ and $h_0^{\leftarrow}$, respectively) to a tensor of all zeros. I also got two context vectors, one from the forward GRU after it has seen the final word in the sentence, $z^{\rightarrow} = h_T^{\rightarrow}$, and one from the backward GRU after it has seen the first word in the sentence, $z^{\leftarrow} = h_T^{\leftarrow}$

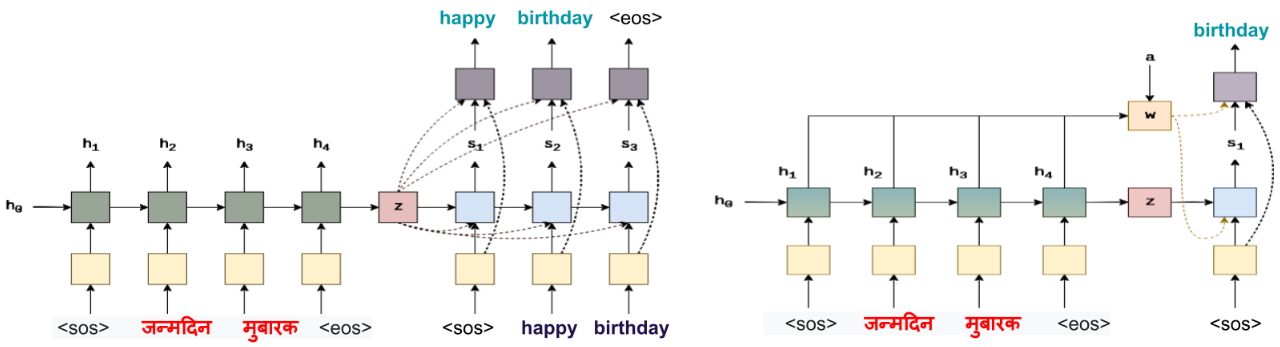


Figure 4: **Left:** Modified encoder and decoder[6] to alleviate some of the information compression in context vector z. **Right:** Decoding word birthday using context vector z and attention vector a

The attention layer[7], will take in the previous hidden state of the decoder, s_{t-1}, and all of the stacked forward and backward hidden states from the encoder, H. The attention will layer will output an attention vector, a_t, that is the length of the source sentence, where $a_t \in \{0, 1\}$ and $\sum_t a_t = 1$. This layer takes what has been decoded so far, s_{t-1}, and all of what has been encoded, H, to produce a_t, that points which words in the source sentence we should pay the most attention to in order to correctly predict the next word to decode, $\hat{y}_{t+1}$.

4.2.3 Attention Decoder:

The decoder contains the attention layer, s_{t-1}, H, and returns the attention vector, a_t, which is used to create a weighted source vector, w_t, which is a weighted sum of the encoder hidden states, H, using a_t as the weights. Embedded input word, $d(y_t)$, w_t, and s_{t-1}, are then passed into the decoder GRU, with $d(y_t)$ and w_t being concatenated together as shown in Figure 4(Right).

$$s_t = \mathbf{DecoderGRU}(d(y_t), w_t = a_t H, s_{t-1})$$

4.3 Phase 3

4.3.1 Overview:

Model's Architecture: uni-GRU encoder, Uni-GRU Decoder with Attention mechanism Stochastic gradient descent (SGD) Optimizer, Negative log-likelihood(NLL) Loss function, Dropout technique with probability= 0.1. Model's METEOR score= **0.200**, BLEU Score= **0.0247**.
In previous two phases I worked with ADAM optimizer and Cross Entropy loss function but in this phase I tried SGD optimizer and NLL loss function, following the PyTorch official tutorial[8] for NMT.

[5]Medium Tutorial on paper[2]
[6]Image source
[7]Medium Tutorial on paper[3]
[8]PyTorch official Tutorial for NMT

4.3.2 Encoder and Attention Decoder:

The encoder of this seq2seq network is a GRU network. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word. Attention[4] allowed the decoder to focus on a different part of the encoder's outputs for every step of the decoder's own outputs. First I calculated set of attention weights, multiplied it by the encoder output vectors to create a weighted combination which contain information about that specific part of the input sequence, so it helped decoder to choose the right output words.

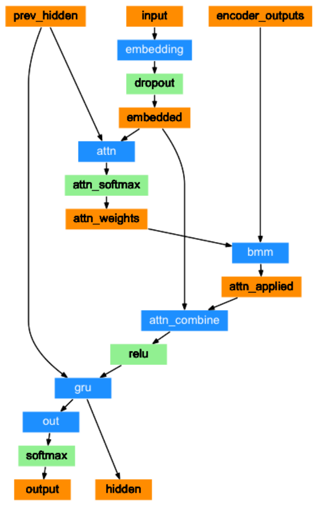


Figure 5: Flow chart showing Model Architecture Flow using attention decoder

4.4 Final Phase:

Best Model's METEOR score = 0.389, BLEU Score= 0.1030. Model's Architecture: Transformer with 8-head Multi-Attention Layer, Encoder and Decoder with Position wise Feed Forward Layer. ADAM Optimizer, Cross Entropy Loss fuction, Dropout technique with probability= 0.1, Learning rate=0.0005 and Xavier Weight Initialisation

In the final phase I tried implementing a (slightly modified version) of the Transformer model from the paper **Attention is All You Need paper**[5].

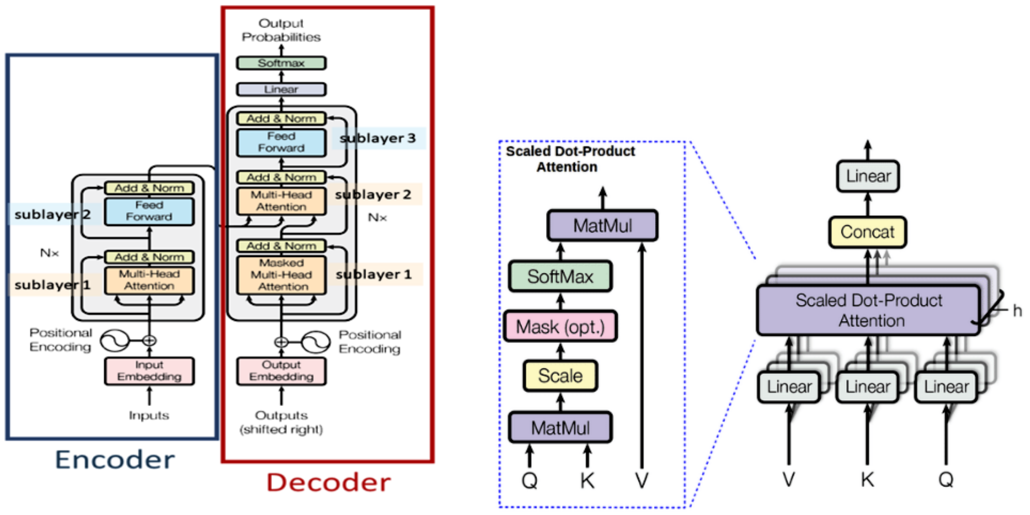


Figure 6: **Left:** Various Layers of Transformer's[5] Encoder and Decoder Architecture. **Right:** Mutli Head Attention Layers

4.4.1 Encoder:

The Transformer's encoder doesn't compresses the whole source sentence, $X = (x_1, ..., x_n)$, into one context vector, z. Instead it produces multiple context vectors, $Z = (z_1, ..., z_n)$. So, if input sequence is 4 tokens long we would have $Z = (z_1, z_2, z_3, z_4)$. First, the tokens are passed through embedding layer. Since the model has no recurrent it doesn't know ordering of the tokens within a sequence. This is solved by a **positional embedding layer**, where the input is the position of the token within the sequence. Then the token and positional embeddings are element wise summed together to produce a vector having information about both the token and its position. Beforehand the token embeddings are multiplied by $\sqrt{d_{model}}$, where d_{model} is the hidden dimension size[9].

I first passed the Hindi sentence into the multi-head attention layer, applied a residual connection, and passed it through a Normalization layer. Followed by passing it to **position-wise feed forward** layer and then applied dropout, a residual connection and then **layer normalization** to get the output of this layer which was fed into the next layer. The multi-head attention layer is used by the encoder layer to attend the Hindi sentence, it applies attention over itself. What the normalization Layer does is that it normalizes the values of the features, so that each feature has a mean of 0 and a standard deviation of 1. This allows Transformer, to be trained easier.

4.4.2 Mutli Head Attention Layer:

The Transformer uses scaled dot-product attention[10], where the query and key are combined by taking the dot product between them, then applying the softmax operation and scaling by d_k before, finally then multiplying by the value. d_k is the head dimension

$$\textbf{Attention}(Q, K, V) = \textbf{Softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V$$

instead of paying attention to one concept per attention application, we pay attention to h heads(8 in case of my implementation), which are re-combined into their hidden dimension shape, thus each hidden dimension is potentially paying attention to h many different concepts.

$$\textbf{MultiHead}(Q, K, V) = \textbf{Concat}(\textbf{head}_1, ..., \textbf{head}_h)W^O$$
$$\textbf{head}_i = \textbf{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where, W^O is the linear layer applied at the end of the multi-head attention layer.

4.4.3 Decoder:

The decoder is similar to encoder, however it now has **two multi-head attention layers**. A masked multi-head attention layer over the target sequence, and a multi-head attention layer which uses the decoder representation as the query and the encoder representation as the key and value. The decoder uses positional embeddings and combines them using element wise sum with the scaled embedded target tokens, followed by dropout. The combined embeddings are then passed through the N decoder layers, along with the encoded source. The decoder representation after the N^{th} layer is then passed through a linear layer.

5 Experiments

5.1 Data Pre-processing:

- I removed **643** sentence pairs in which both Hindi and English sentences were written in English language, as they won't be useful in learning Hindi to English translation.
- Training example having wrong translations were removed(as much as I found). Hindi sentences where space were not properly provided I manually placed space at proper places for approx **60** Hindi sentences(Figure 2)
- Duplicate training examples were removed. English sentences were lower cased, to have uniformity.
- Removed multiple occurrence of more than **15** symbols like ['.', ',', ' ', '-', '(', ')', ':', '"', '!', '?', '/', ' ', '[', ']', '—'] from both Hindi and English sentences, as these were frequently repeating in a sentences, decoder with attention mechanism were not able to pay attention to actual words in the text.
- Removed few of the Hindi stop words. Also removed Stop words in English like "a", "the", "is", "are".
- Followed by spliting the training data provided into two sets having ratio **80:20** by number of examples, namely the training set and validation set.
- Combined tokens(generated using **Spacy**) like (" gon " and " na " = " ganna"), (" we' ", " ll " = " we'll "), (" have "," n't " = " haven't "), etc. Because the combined form makes more sense as token.

[9]The Illustrated Transformer
[10]The Annotated Transformer

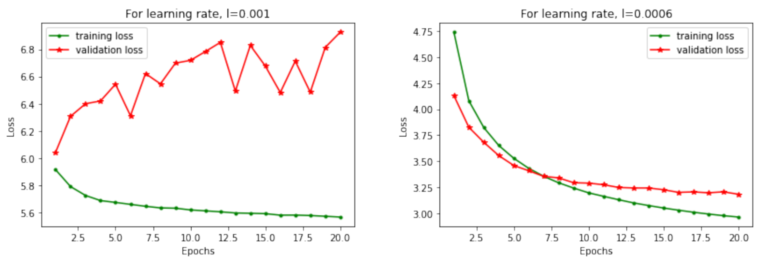
5.2 Training Procedure:

Phase	Model	Optimizer	Loss function	Learning rate	Epochs	Training time
1	**LSTM**	ADAM	Cross Entropy	0.001	10	2 hours
2	**Bi-GRU**	ADAM	Cross Entropy	0.001	5	3 hours
3	**GRU**	SGD	NLL	0.01	20	12 hours
Final	**Transformer**	ADAM	Cross Entropy	0.0005	30	3 hours

Table 2: Loss function, Learning rate, number of epochs, and training time details for each of model.

5.3 Selection of Hyper-parameters:

Values chosen for various hyper-parameters like **Encoder and decoder hidden dimensions, dropout probability, number of layers and batch size** are mostly influenced by **a**. Hyper-parameter values used in the papers I tried to implement[1, 2, 3, 5, 6], **b** in this competition. Computation power provided by Google Colab(without Colab pro) and **c**. monitoring my validation set loss on different values of hyper-parameters. For phase 1 and 2, I used default **Learning rate** of **0.001** of ADAM optimizer later Learning rate was decided empirically after observing the plots of training and validation losses vs number of epochs, if there was a huge difference in these losses like Figure 7(Left) I refused to use that learning rate. **Numbers of epochs** were decided mainly after seeing if the validation loss didn't decrease in **3** consecutive epochs.



6 Results

Phase	Model	Validation set		Test set		Leaderboard Rank
		BLUE	METEOR	BLEU	METEOR	
1	LSTM	0.0082	0.258	0.0066	0.245	13
2	Bi-GRU	0.0693	0.331	0.0687	0.356	**4**
3	GRU	0.0380	0.260	0.0247	0.200	13
Final	Transformer	0.1070	0.401	**0.1030**	**0.389**	**7**

Table 3: BLEU and METEOR scores of all models used in 4 phases of the competition

Transformer was the best performing model, both in terms of METEOR and BLEU score when evaluated on validation and test set. Since in transformer sentences are processed as a whole rather than word by word, that's why its doesn't suffer from long dependency issues like LSTMs and GRUs. Transformer also doesn't relies on past hidden states to capture dependencies[5] with previous words, they process a sentence as a whole, reason why there is no risk to forget past information. Moreover, multi-head attention and positional embeddings both provide information about the relationship between different words.

7 Error Analysis

After comparing translations generated by 4 models that I have implemented with actual translation provided at the end, I observed that even my best performing Transformer model is making repetitions in the translation of some example while the provided answer doesn't suffer from this issue. My model also outputs $< unk >$ tokens in some examples whereas the actual translation outputs correct word.

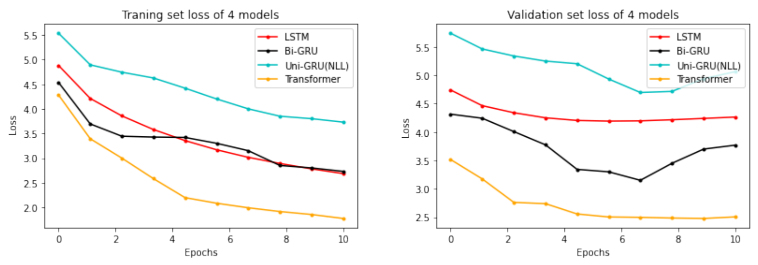
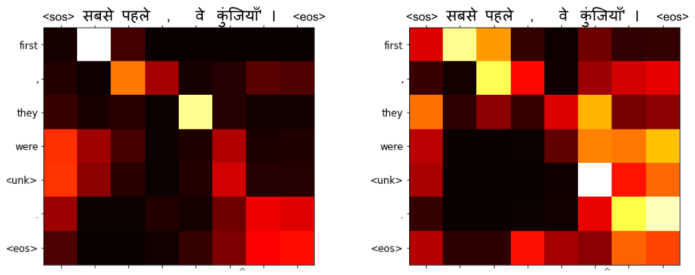
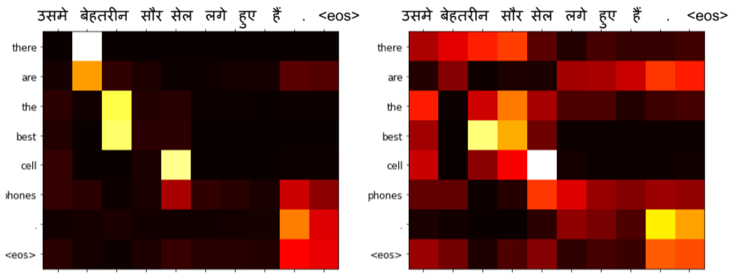


Figure 7: Plot showing variation in Training loss(left) and Validation set loss(right) as the number of number of epochs increases for all for models



(a) Attention heat map for head 1 and head 2 of an example from validation set



(b) Attention heat map for head 1 and head 2 of an example from Test set

Figure 8: Attention heat maps of Multi-head Attention Layer

From the attention heat maps of 2 heads out of 8 heads of multi-head attention in the above figure, we can see that each is certainly different, but it's difficult to reason about what head has actually learned to pay attention to. For one of the examples from validation (Figure 8a) head 1 pay full attention to the 1^{st} word of Hindi sentence while the head 2 pays more attention to the 5^{th} word. After observing all 8 heads attention heat map for various examples from validation set and test set, they all seem to follow somewhat similar "downward staircase" pattern and the attention when outputting the last two tokens is equally spread over the final two tokens in the input sentence. Again for an example from test set Again, head 1 pays full attention to 2^{nd} word of Hindi Sentence while head 2 pays no attention to it rather it pays full attention 4^{th}. Again, most of the heads seem to spread their attention over both the **"."** and $< eos >$ tokens in the Hindi sentence , though some seem to pay attention to tokens from near the start of the sentence.

8 Conclusion

In this competition I tried Basic RNN, LSTM, Bi-GRU, CNN and Transformer models for Hindi to English Translation. Simple RNN model didn't give good results while its Variants like LSTMs, GRUs produced significantly better translations. Highest metric scores were achieved with the Transformer model that too with least number of parameters and the fastest training time. So I would definitely recommend Transformer model for the task of Hindi to English machine translation. Better pre-processing(especially for Hindi sentences) and data cleaning techniques together with some principled approaches for finding optimal hyper-parameters like weight-initialisation , learning rate, number of layers in encoder and decoder and number of heads in multi head attention model can improve the Translation model for this NMT task.

References

[1] I. Sutskever, O. Vinyals, and V. L. Quoc, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, (Montreal, Canada), 2014.

[2] D. Bahdanau, K. Hyun Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations*, (San Diego, United States), 2015.

[3] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, (Doha, Qatar), 2014.

[4] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), 2015.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, (Red Hook, NY, USA), 2017.

[6] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," 2017.