Bridging Generations: Comparative Performance of LSTM and Transformer in Language Translation

M240699CS - Vishwaksanar NM
M240602CS - Tatigunta Bhaviteja Reddy

Introduction :

Language translation systems have undergone significant advancements in recent years, thanks to the evolution of deep learning architectures. From the advent of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models to the emergence of Transformers, the landscape of machine translation has transformed dramatically. These models not only enhance the accuracy of translations but also expand the scope of multilingual applications, ranging from real-time communication tools to document localization and beyond.

LSTMs have been widely used in sequence-to-sequence tasks, offering mechanisms to handle long-term dependencies in sequential data. Despite their success, these architectures often struggle with scalability and computational efficiency when dealing with extensive datasets. On the other hand, the Transformer architecture, introduced in Vaswani et al.'s seminal paper Attention Is All You Need, has revolutionized natural language processing (NLP). Leveraging self-attention mechanisms, Transformers excel in parallelizing computations and capturing contextual relationships over long sequences.

This project, "Bridging Generations: Comparative Performance of LSTM and Transformer in Language Translation," explores the performance gap between these two architectures in the domain of English-to-Hindi translation. By training and evaluating both models on a shared dataset, we aim to quantify their effectiveness using BLEU scores, computational efficiency, and translation quality.

Dataset:

Dataset: The IIT Bombay English-Hindi Parallel Corpus (cfilt/iitb-english-hindi) is loaded. This dataset is widely used for training translation models between English and Hindi, providing high-quality parallel sentences.
Source: The dataset is fetched from the Hugging Face datasets library.
Structure: Each entry in the dataset contains translations of sentences in English ("en") and Hindi ("hi") under the "translation" key.

- Training Set: Used to train the models on input-output pairs.
- Validation Set: Used for performance evaluation during training.
- Test Set: Reserved for final model evaluation.

METHODOLOGY

2.1 RECURRENT NEURAL NETWORK(RNN):

Recurrent Neural Network is a feedback back neural network where the output of the network is fed again to the input at each timestamp, this kind of neural network is useful in applications where sequencing or ordering is importatnt and needed to be preserved.

2.1.1 Feedforward Network:

In feedforward network, at timestamp t=0 we input the network with the word that appears first in the sentence which is already converted into vectors using word2vec or any other data preprocessing functions. After that a weight is assigned along with the vectored input to fed it to the network.And also note that RNN loops back again, there is another weight which is assigned to the output that is generated for next time step.For t=0 , this weight is 0 as there is no input form previous time step.

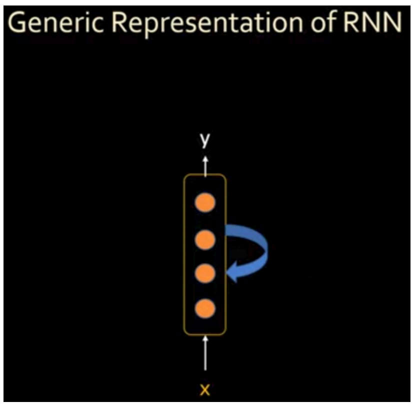


Figure 2.0

2.1.2 Backpropagation :

In Backpropagation, the weights are adjusted according to the loss function by using optimisers like Gradient decent using chaining rule.

This RNN can be used in Language Transaltion,

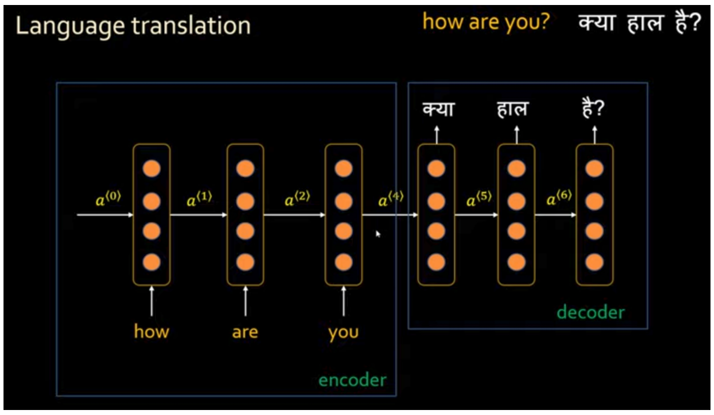


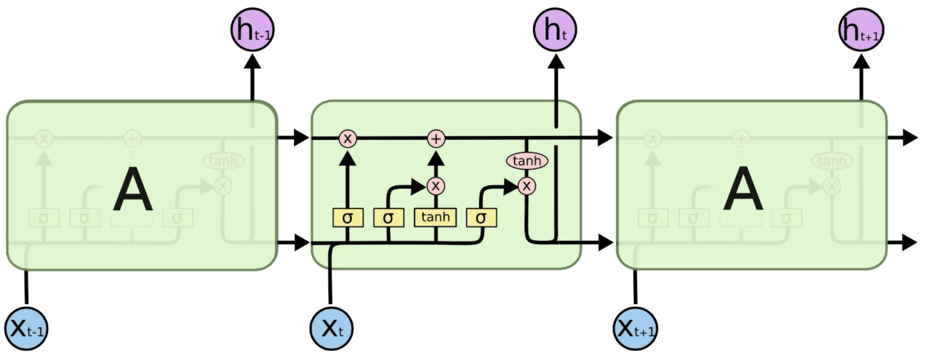
Figure 2.1

2.2 LSTM(Long Short Term Memory) :

It is a variant of RNN,where it uses the concept of long term and short term memories which helps to preservs the context of the sentence in sequence to sequence translation as traditional RNN would fail in this case since it laks in maintaining long term memory.

2.2.1 Structure of LSTM :

LSTM consists of three gates namely input gate,forget gate and ouput gate to regulate the cell state.



The repeating module in an LSTM contains four interacting layers.

Figure 2.3

Cell State :

The Horozontal Line is the cell state where it interacts with other three gates to decide whether to keep particular information or not with respect to the current input and previous output that has been extarcted as a whole from the previous timestamp.

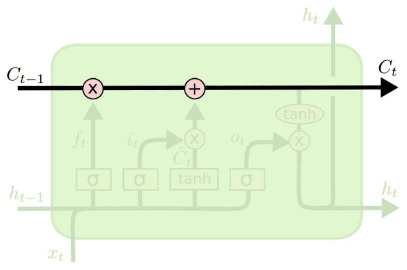
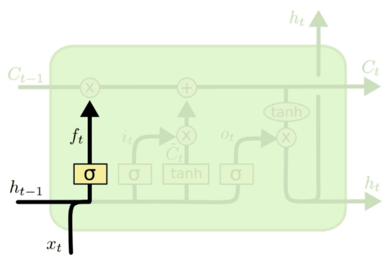


Figure 2.4

Forget Gate :

This gate tells which information need to be forgetton as it is nothing to do with the context.



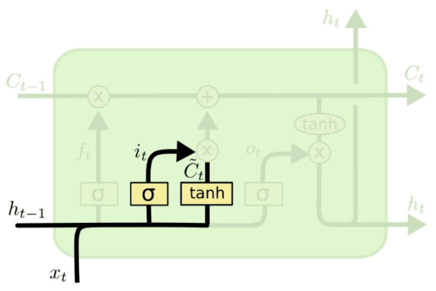
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 2.5

The gate contains a sigmoid function where it outputs 0 or 1 to tell whether to preserve the information or not.It looks h(t-1) and x(t) to decide the information can be forgotten or not.

Input Gate :

This gate decides on what new information can be added into context.

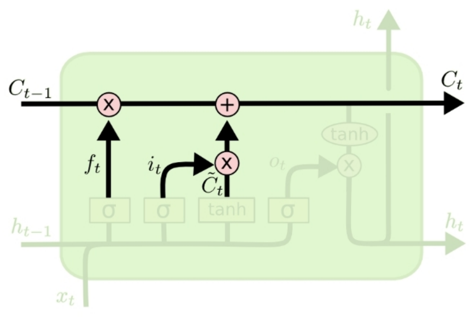


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 2.6

It contains two parts, in the first part the sigmoid layer decides which values need to be updated and in the second part the tanh layer decides what values will be updated by creating a candiate vector.

After deciding what to forgot and what to add, we actuallty perform the operation to happen by,



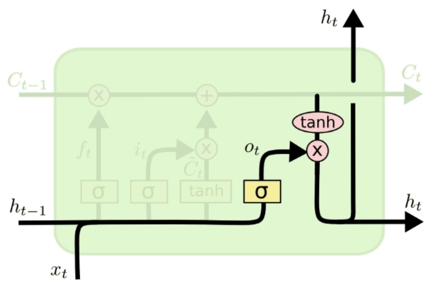
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.7

Multiplying f(t) and c(t-1) for forgetting the values and then we add i(t)*c(t) for adding new information.

Output Gate :

This gate decides what to output from the cell, it performs a sigmoid operation to decide on which parts of the cell state will be shown in output and then it is multiplied with result of tanh operation where the cell state is fed as the input.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.8

2.3 Encoder Decoder using LSTM :

For sequence to sequence translation , theoratically we can go with RNN but in practical use RNN fails in keeping the Long term memory so we use LSTM for the transaltion problem.

2.3.1 Encoder Decoder Architecture :

Encoder :

In Encoder section , the RNN(LSTM) reads the input sequentially at every time step and creates a hidden vector which will be fed into the decoder as input.

At every timestamp the hidden vector is updated with current input, the output of the RNN is ignored.

For example, consider input sequence “Iam a student” is fed into encoder.There will be four timestamps as there are four tokens,at each time step the hidden vector will get updated with respect to current input and produce the final output.

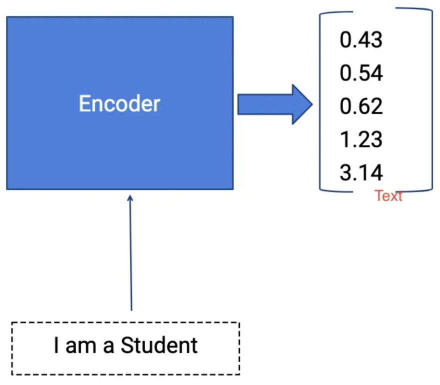


Figure 2.9

Decoder :

In Decoder section, it gets the hidden vector as the input from the encoder and outputs a probability distribution of each word at each timestamp which is the prediction to tell what are all the possible words that can fit into the sentence translation.This probability distribution is used to search the best fit word using Beam search algorithm.

2.3.2 Beam Search:

After getting output from encoders, we feed it into the decoder. We won't be ignoring the output of the decoder as decoder gives us the probability distribution of the word as the output.

With the use of this decoder's output, we use an search algorithm called Beam Search to find the exact word that matches in the context with the translating language.

Working of Beam search Algorithm:

Let's take an example to understand the algorithm, consider a beam width of two. This beam width is considered to give us the best possible candidates as the width increases possible candidates increases.

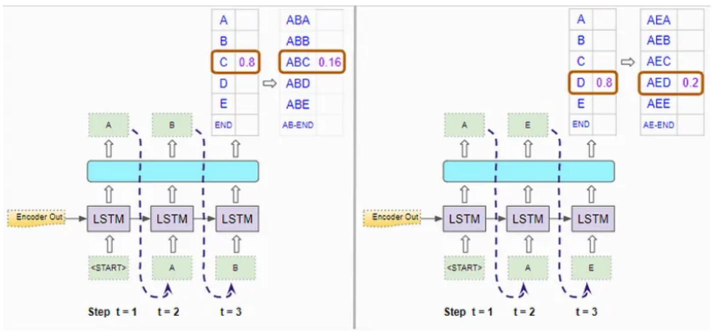


Figure 2.10

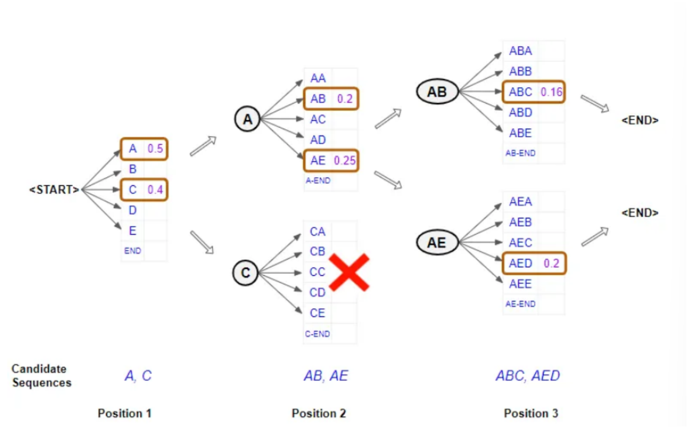


Figure 2.11

From the above figure we can infer that the algorithm first outputs possible candidate words for the initial word, after the first timestamp we got ‘A’ and ‘C’ as candidates, the algorithm now iterates two times first considering A as first word it computes next possible candidates getting ‘AB’ and ‘AE’. Likewise, the algorithm iterates again for ‘C’ to get possible candidates and now algorithm looks for highest combined probabilities for consideration for next evaluation. This process repeats until the last word.

2.4 Encoder and Decoder using Transformers:

Transformers are very powerful in sequence-to-sequence translation compared to LSTM(s) as they lack in translating very long sentences. The idea behind using transformers is that it extracts the context of sentence and calculates a score with every other word in correspondence to particular processing word.

2.4.1 Working of Transformers:

Here we normally have encoder and decoder part like in RNN, but the input is fed all at once each embedded word travels in separate path to get processed and finally a matrix is resulted which is fed into the decoder input.

Given below is the simplified version of Encoder decoder architecture using transformer,

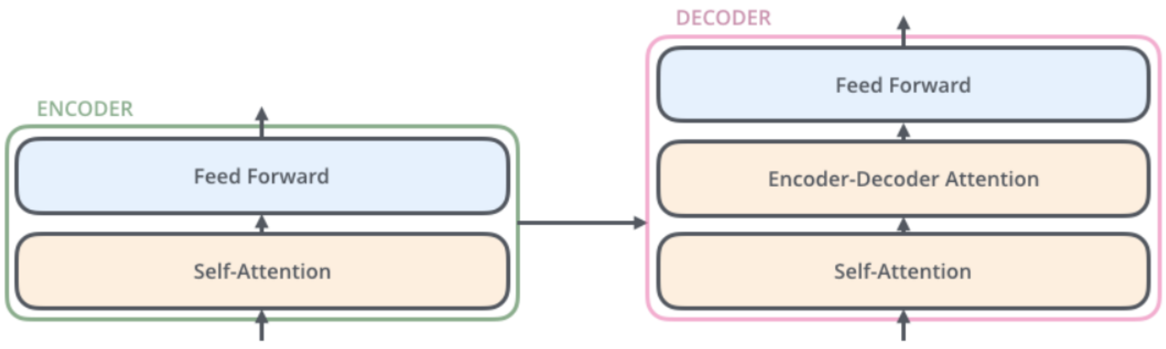


Figure 2.12

Each encoder contains a self-attention layer and a feed forward layer. In decoder it contains both layer what encoder has in addition to that it also contains a Attention layer.

Now we can take deep dive into the working of this encoder decoder model, first input is fed into the encoder model after converting each word into vectors of size 512, these vectors first encounter self-attention model where three vectors are created namely a query vector, a key vector and a value vector for each word. These three vectors may or may not be smaller in dimension than the embedded vectors. Let’s take a simple example of one sentence “THINKING MACHINES” after vectors creation now next step is create attention scores, this score decides how much attention has to be given to other words in comparison with the word at the position that is encoded. So the first score is calculated by taking dot product with q1 and k1 , second score is calculated by taking dot product between q1 and k2.

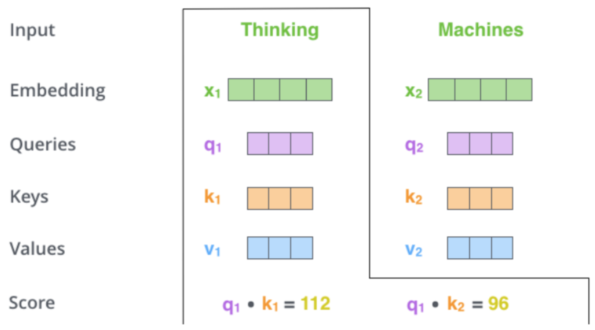


Figure 2.13

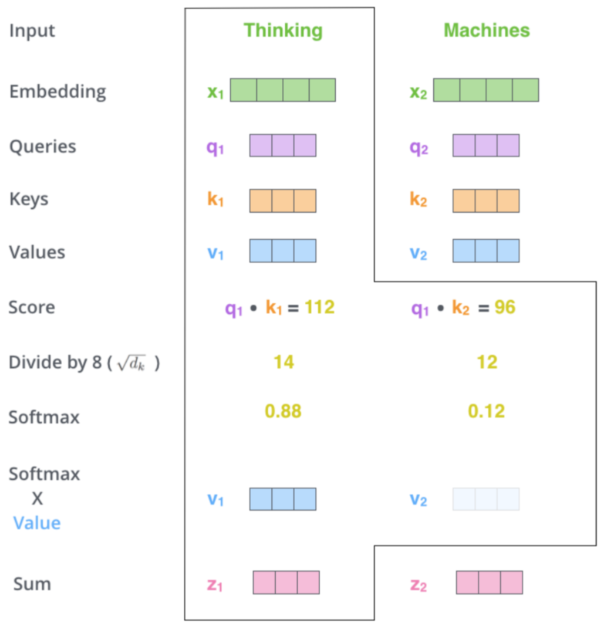


Figure 2.14

After the calculating the scores, it is divided by 8 which is the square root of the dimension of key vector. Then it is passed into the SoftMax function after that it is calculated with value vector to get Z vector.

Instead of vectors we will using matrix for faster calculation and also multiheaded self attention layer where instead of computing single z matrix eight different matrices are calculated,

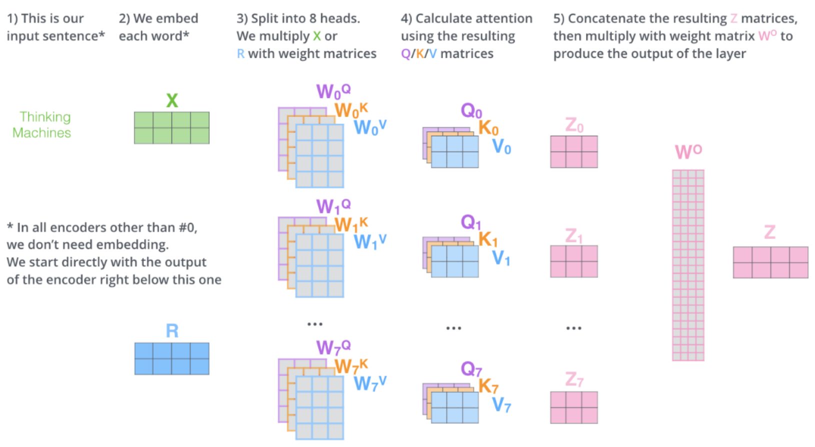

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
R
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z

Figure 2.14

Decoder:

In decoder part, same process carried out but with slight difference that is in self-Attention layer is configured to compute earlier positions in the sequence by masking the future sequence positions. The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

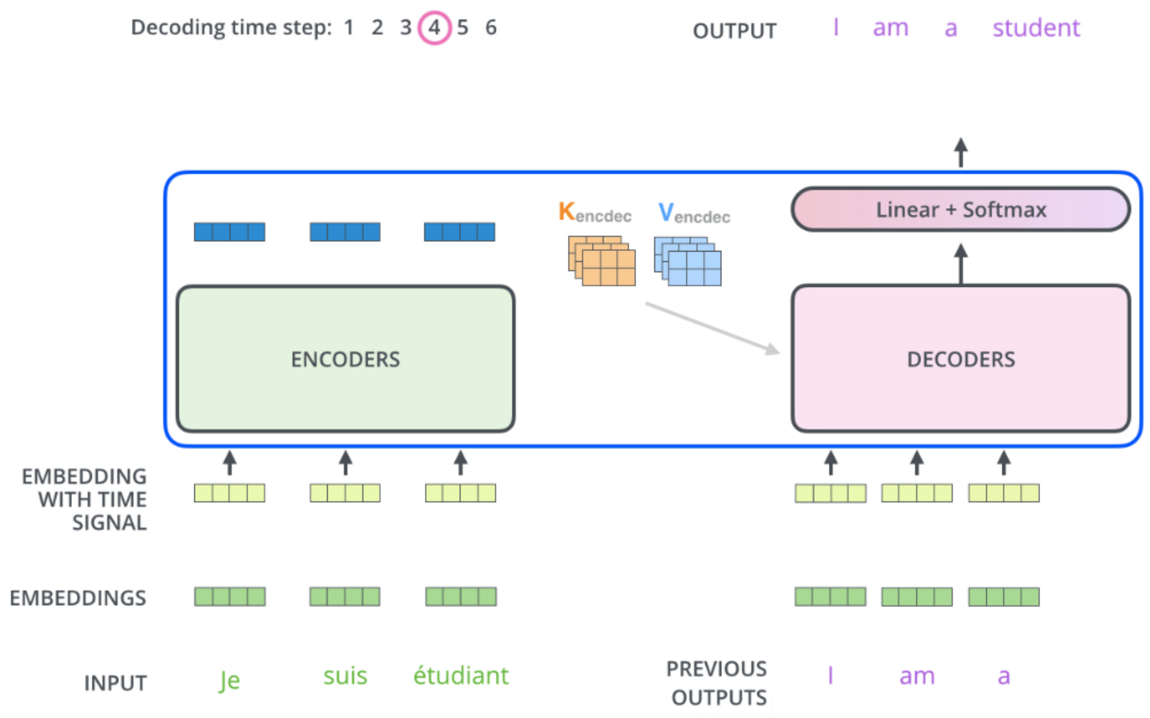


Figure 2.15

3.0 Metrics to Compare Algorithms :

LSTM

1.Optimizer:

- Adam Optimizer
- Adaptive learning rate optimizer designed to work well with sparse gradients and noisy datasets.
- Key parameters:
- Learning rate : 0.001 (default in many implementations).
- beta_1 = 0.9, beta_2 = 0.999: Exponential decay rates for the first and second moment estimates.

2.Loss Function:

- Categorical Cross-Entropy Loss:
- Suitable for multi-class classification problems, as each output token is predicted from a vocabulary of possible tokens.

3.Evaluation Metrics:

- BLEU Score
- Measures the overlap between predicted and reference translations, focusing on n-gram precision.
- Higher BLEU scores indicate closer alignment with human translations.
- Training Loss:
- Observed during training to ensure convergence.

Transformer

1. Optimizer:
- Adam Optimizer with Weight Decay (AdamW):
- Enhances Adam by incorporating L2 regularization (weight decay) to prevent overfitting.
- Key parameters:
- Learning rate (alpha): 2 * 10^-5.
- Weight decay: 0.01.
- beta_1 = 0.9, beta_2 = 0.98: Slightly adjusted decay rates to stabilize training for Transformers.

2.Loss Function:
- Label Smoothed Cross-Entropy Loss:
- A variant of cross-entropy loss where a small probability is assigned to all incorrect classes to prevent overconfidence in predictions.

Key Differences Between LSTM and Transformer Approaches

Aspect	LSTM	Transformer
Optimizer	Adam	AdamW (with weight decay)
Loss Function	Cross-Entropy	Label Smoothed Cross-Entropy
Evaluation	BLEU	BLEU
Efficiency	Sequential updates (slower)	Parallel computation (faster)
Context Handling	Focused on short-term memory with attention	Captures long-range dependencies effectively
BLEU	6.3	6.9

Conclusion:

This study highlights the superiority of the Transformer architecture over LSTM for English-to-Hindi translation. With a higher BLEU score and faster training times, Transformers proved to be more suitable for real-world machine translation tasks. While LSTMs are still effective for smaller datasets and simpler tasks, the Transformer's ability to handle long-term dependencies and parallelize computations makes it the architecture of choice for modern NLP applications.

Future work could explore domain-specific fine-tuning, alternative evaluation metrics like ROUGE, and optimization techniques to improve the computational efficiency of Transformers further. This comparative analysis underscores the transformative impact of modern deep learning architectures on natural language processing.

References:

1. Google.co.in. (2018). *The IIT Bombay English-Hindi Parallel Corpus*. [online] Available at: https://scholar.google.co.in/citations?view_op=view_citation&hl=en&user=jnoUuGcAAAAJ&citation_for_view=jnoUuGcAAAAJ:bnK-pcrLprsC [Accessed 28 Nov. 2024].

2.Laskar, S.R., Dutta, A., Pakray, P. and Bandyopadhyay, S. (2019). Neural Machine Translation: English to Hindi. *2019 IEEE Conference on Information and Communication Technology*, [online] pp.1–6. doi:https://doi.org/10.1109/cict48419.2019.9066238.

3.Singh, A., Bhase, N., Jain, M. and Ghorpade, T. (2022). Machine Translation Systems for English Captions to Hindi Language Using Deep Learning. *ITM Web of Conferences*, 44, p.03004. doi:https://doi.org/10.1051/itmconf/20224403004.

4.Joshi, N., Hemant Darbari and Mathur, I. (2012). Human and Automatic Evaluation of English to Hindi Machine Translation Systems. *Advances in intelligent and soft computing*, [online] pp.423–432. doi:https://doi.org/10.1007/978-3-642-30157-5_42.