CS779 Competition: Machine Translation System for India

Shivam Pandey
200938
shivamp20@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

Neural Machine Translation (NMT) has emerged as a critical component in bridging linguistic gaps and facilitating global communication. This research project endeavors to tackle the complex task of translating seven diverse Indian languages into English, employing NMT techniques. The project extends beyond the basic transformer-based architecture, incorporating several modifications aimed at enhancing both the efficiency and the quality of translations. These modifications include the strategic reduction of the training dataset using predefined heuristics, the development of a rule-based tokenizer from scratch, optimization of the training loop, and the implementation of strategies to improve the speed and accuracy of the inference process. The results of these enhancements are evaluated and discussed, shedding light on the potential of NMT for multilingual translation tasks, particularly in the context of the Indian subcontinent.

1 Competition Result

Codalab Username: S_200938
Final leaderboard rank on the test set: 6
charF++ Score wrt to the final rank: 0.355
ROGUE Score wrt to the final rank: 0.330
BLEU Score wrt to the final rank: 0.129

2 Problem Description

The challenge was to translate seven distinct Indian languages into English, each endowed with its linguistic heritage. There was strict requirement to abstain from utilizing pre-trained models, representing a deviation from standard NMT practices. Model had to rely solely on the provided training data, necessitating the creation of translation models from the ground up. External GPU usage was strictly prohibited, with computational resources confined to Google Colab and Kaggle's GPU infrastructure, ensuring a uniform and fair competition. Additionally, the prohibition of pre-trained tokenizers mandated the development of tokenization processes tailored to the intricacies of Indian languages.

3 Data Analysis

The dataset was in JSON format, and a script in the "dataset_generation.ipynb" notebook was employed to split it into separate training and validation files for each language. The extracted files were then placed in a new folder called "new_dataset." In Table 1, the corpus statistics were presented. Notably, the training data for Gujarati, Kannada, and Telugu each contained one sentence with null values, necessitating addressing in the preprocessing pipeline.

Source Language	No of Sentences	Average Length of Sentence		Total Words in Vocabulary	
		Source	Target	Source	Target
Hindi	80796	18.98	18.72	96444	67782
Bengali	68848	14.37	18.39	117733	62500
Gujarati	47481	15.99	18.95	92670	49459
Kannada	46794	11.59	16.44	107436	44133
Tamil	58361	12. 53	17.89	145792	54626
Telugu	44904	11.77	16.32	98732	46213
Malayalam	54056	10.46	16.14	146604	50363

Table 1: Corpus Statistics for Training Data

4 Model Description

1. Seq-to-Seq using GRU and attention

   Seq2seq with attention was implemented first because the addition of attention would never hurt the accuracy using a plain vanilla RNN/GRU seq-to-seq model. The architecture mainly used 3 components: an encoding, an attention mechanism, and a decoder.

   Encoder: The input sequence was first fed in an embedding layer. This layer converted each input token to word embeddings. This embedded sequence is then fed into a Gated Recurrent Unit (GRU) to capture the sequential information. Another type of RNN based model : LSTM, was also used during experimentation but it did not perform equally well; hence, it was replaced with a GRU. The Encoder finally returns both: the outputs from GRU layer (which is later used for attention calculation), and the final hidden state.

   Attention mechanism: This implementation uses a dot product attention mechanism which calculates a similarity score between the current hidden state of Decoder and each hidden state in Encoder's output sequence. These similarity scores are then used to compute attention weights. Masking is used to so that attention is applied only to valid tokens and not padding tokens. A mask is applied to attention scores to prevent model from attending to padding tokens.

   Decoder: The Decoder's hidden state is initialized with the final hidden state of the Encoder. This serves as the initial context for decoding the target sequence. For each time step in the output sequence:
   Previously generated token is passed(<eos> for first step) through the embedding layer to get word embedding. This embedded representation is then passed a GRU layer, which updates its hidden state based on it. This hidden state from the Decoder GRU is then used as a query to the attention mechanism, which computes attention weights that signify how much attention should be given to each position in the source sequence. These attention weights are used to compute a weighted sum (context vector) of encoder outputs, highlighting the source sequence's relevant information for generating the current output token. The concatenation of context vector with the output of Decoder GRU is passed through a linear layer to produce the output logits for the current token. The token with the highest probability in the logits is selected as the predicted token for the current time step. This process is repeated for each time step in the output sequence until either an end-of-sequence token (<eos>) is generated or a maximum sequence length is reached.

   During training phase, the cross-entropy loss is optimized by comparing its predicted logits with the actual target sequence. The Adam optimizer updates the model's parameters to capture complex relationships.

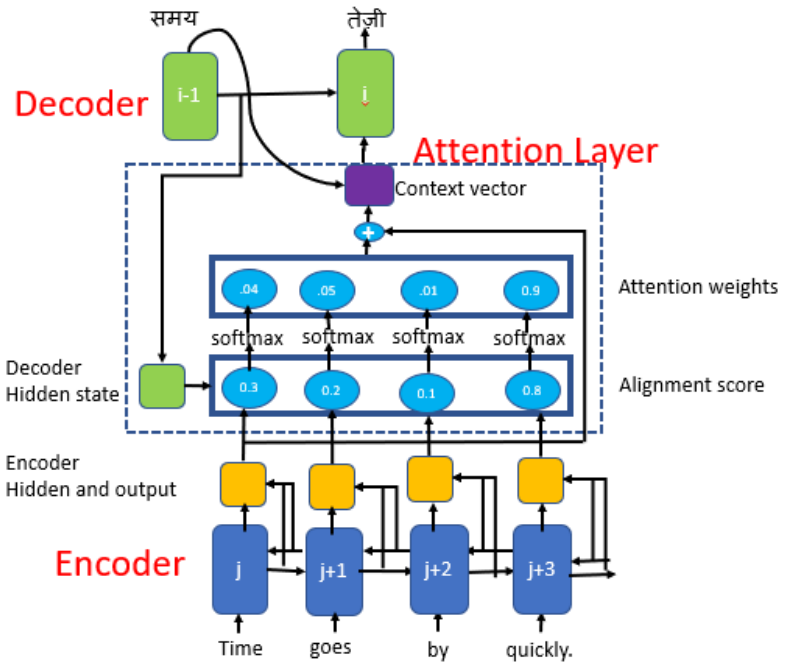


Figure 1: Seq-2-Seq GRU + Attention architecture used

During the inference, the model is employed to generate translations. This is done by using "translate_sentence" function, which takes an input sentence, applies tokenization, converts tokens into embeddings, and decodes the output sequence. This is accomplished one token at a time in a greedy manner. Using the learned probabilities, the model selects the most probable tokens at each time step.

This model is inspired by official pytorch tutorial[1], paper by Ezeani et al. (2020)[2] and its implementation.

2. Seq-to-Seq using Transformers

This model uses a standard encoder-decoder architecture for sequence-to-sequence tasks where encoder processes input sequence, and decoder generates output sequence. It consists of multiple layers of Transformer encoder and decoder blocks, to capture complex linguistic relationships. Token embeddings for both source and target sequences are created using TokenEmbedding layers. Positional Encoding incorporates information about the position of tokens in the sequence. This encoding uses a combination of sine and cosine functions to encode token positions so there is no need for learned position embeddings.

Attention mechanisms are used in both encoder and decoder to focus only on relevant parts of the input sequence when generating the output. In source and target sequences, masking helps prevent the model from attending to padding tokens. The final output is then passed through a linear layer to produce logits, which are used to predict the target sequence.
The model uses the Adam optimizer to minimize the cross-entropy loss(with padding tokens ignored) between predicted logits and ground truth target sequences

Categorical Cross-Entropy Loss $= -\sum_{i=1}^{V} y_i \log(p_i)$
where V is the vocabulary size. y_i is the true probability (1 or 0) for the actual token. p_i is the predicted probability for the same token.

During inference, the model generates translations by selecting the most probable tokens at each

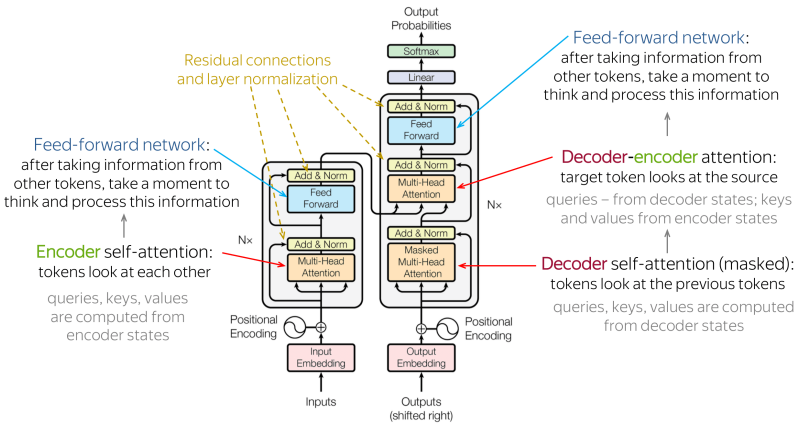


Figure 2: Seq-2-Seq Transformers architecture used

step. Model parameters are initialized using Xavier initialization. Batch Processing is used to improve efficiency during training.

Beam search was implemented but it introduced another hyperparameter: beam width, which needed to be tuned. Greedy decoding was preferred to save computational resources while experimenting.
This model is inspired by official Pytorch Tutorial implementation.

5 Experiments

Here are the experiments and modifications tried with different parts of the architecture:

1. Tokenization Approach and Experimentation
   (a) The initial tokenization strategy employed Spacy's "en_core_web_sm" model with specific components, such as the parser, tagger, and named entity recognizer (NER), disabled. This selective deactivation aimed to streamline the tokenization process and prevent any unintended interference with the linguistic structure of the English text. For tokenizing the Indian languages, the Indic NLP tokenizer was employed.
   (b) However, subsequent to receiving new instructions from our academic advisor, the directive was to abstain from the use of any pre-trained tokenizers. This necessitated the development of a rule-based tokenizer tailored specifically for the English language. The rule-based tokenizer addressed various aspects of tokenization. Firstly, it focused on the separation of punctuation and whitespace, distinguishing between white spaces (whitespace characters, commas, semicolons, full stops, exclamation marks, question marks, and hyphens) and words.
   (c) Notably, non-capturing groups, such as contractions (e.g., "don't") and phrases enclosed within quotation marks (e.g., "He said 'I'm here'.") were treated as distinct entities, ensuring accurate tokenization. This process involved the utilization of regular expressions from the Python "re" library.
   (d) It is worth noting that the implementation of this rule-based tokenizer was relatively complex for the seven Indian languages, thus relying solely on white space separators for tokenization was deemed the most feasible approach. This approach maintained consistency and fairness across all languages, even in the absence of pre-trained tokenizers.
2. Dataset Reduction Strategy
   (a) The necessity for dataset reduction emerged due to the excessive duration of training for a single language during one epoch. Attempts were made to augment the batch size for accelerated training, although this proved ineffective for languages with substantial vocabularies such as Tamil, mainly due to GPU constraints.

(b) Experimentation with minimum word frequencies (1, 2, and 3) for vocabulary creation yielded a drastic reduction in vocabulary size, adversely impacting model performance. As a solution, a heuristic was implemented, which involved filtering out the least frequently occurring words and replacing them with a generic token ("<unk>"). However, this approach introduced undesirable consequences as it led to the model acquiring incorrect word representations and subsequently generating inaccurate translations.

(c) To address these issues, sentences containing any of the least frequently occurring words in the vocabulary were systematically removed from the training set, with a primary focus on the source language. After extensive experimentation, it was determined that filtering out the lowest 10% of the least frequently occurring words in the vocabulary provided an optimal solution. This approach resulted in a training set size reduction of approximately 15-17%, depending on the source language. This reduction did not significantly compromise training quality while enabling an increase in batch size, thus reducing the time required for each training epoch.

3. Training loop optimizations and improvements:

(a) The training loop represented a substantial time-consuming component within the overall pipeline, comprising multiple sub-parts. C profiling was employed to discern the time consumption at both sub-part and line levels, revealing an excessive computational delay in calculating operations. This was attributed to GPU constraints, wherein calculations were deferred in the GPU stack, leading to wait times even for rudimentary calculations such as loss computation. Consequently, this served as a substantial bottleneck within the training loop.

(b) To address this bottleneck, an approach was adopted in which variables pertaining to less computationally intensive operations were detached from the GPU and relocated to the CPU using the ".detach()" function. This shift allowed simpler tensor calculations to be executed on the CPU, mitigating the time spent within the GPU stack. However, the optimization delivered only marginal improvements in training time due to the computational overhead associated with the detachment process.

(c) C profiling also unveiled that gradient calculations were a major bottleneck. To rectify this issue, gradient accumulation was implemented. Instead of updating model weights after each batch, this technique involved accumulating gradients across several mini-batches before executing a singular weight update. Gradients from each mini-batch were aggregated to form accumulated gradients. While this approach enabled training with larger effective batch sizes without necessitating additional GPU memory, it introduced a trade-off: model weight updates occurred less frequently, potentially extending the convergence time. The optimal number of loops before model updates with accumulated gradients was determined to be 4-5, concurrently reducing training time and moderately extending the total loops required for convergence.

(d) Furthermore, within the training loop, measures were instituted to mitigate overfitting. Early stopping was applied based on the validation set, a 20% subset of the initial training data. If an upward trend in validation loss was observed, even as training loss continued to decrease, the training loop was terminated, thereby addressing concerns related to overfitting.

4. Inference Experiments:

(a) In the initial stages of the inference process, a straightforward greedy decoding strategy was employed. Two conditions determined when to halt the decoding: encountering the "¡eos¿" token or reaching a maximum length for the target language translation. The determination of the maximum length was derived from an examination of the average length of source and target sentence pairs within the training set for all seven language pairs. The chosen maximum length was set at 10 tokens more than the number of words in the source language. This value of 10 was preferred over 5 because it allowed for a more comprehensive capture of the context within the translation. The larger value of 10 facilitated more coherent and contextually accurate translations.

Let: y_t be the next token to be generated. $y_1, y_2, \ldots, y_{t-1}$ be the previously generated tokens.

Hyperparameter	Value
Embedding size	200
Number of attention heads	10
Feed-Forward Network Hidden Dimension	500
Batch Size	30
Number of Encoder Layers	4
Number of Decoder Layers	4
Number of Training Epochs	60
Learning Rate	10^{-4}
Beta1	0.9
Beta2	0.98
Epsilon	10^{-9}
Optimizer	Adam

Table 2: Hyperparameters of the final model

For Greedy Decoding: $P(y_t|y_1, y_2, \ldots, y_{t-1})$ is determined by selecting the token with the highest probability.

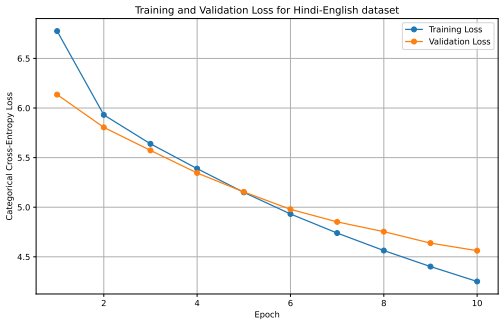
(b) Subsequently, an exploration of beam search was conducted with varying hyperparameters, including different beam widths (2-4) and diverse values for the top k tokens (k values from 5-8). Despite the potential for improved translation quality, beam search incurred significantly longer processing times compared to greedy decoding. This extended processing time resulted from the exhaustive exploration of various translation paths. The selected range of hyperparameters was chosen because it strikes a balance between exploring multiple translation options and maintaining computational efficiency.
For Beam Search: $P(y_t|y_1, y_2, \ldots, y_{t-1}) = P(y_t|h)$, where h represents the hypothesis or partial sequence.

(c) Top-p sampling, which prioritizes tokens based on cumulative probability thresholds (ranging from 0.6 to 0.9), was also investigated. However, top-p sampling did not surpass the performance of greedy decoding in this NMT task. The reason for this lies in the inherent complexities of NMT and the limitations of sampling-based decoding strategies. The range of 0.6 to 0.9 was chosen for the probability threshold experimentation because it captures a sweet spot in balancing exploration and exploitation during decoding, even though the final performance did not surpass that of greedy decoding.
For Top-p Sampling: $P(y_t|y_1, y_2, \ldots, y_{t-1})$ is calculated based on the chosen sampling strategy.

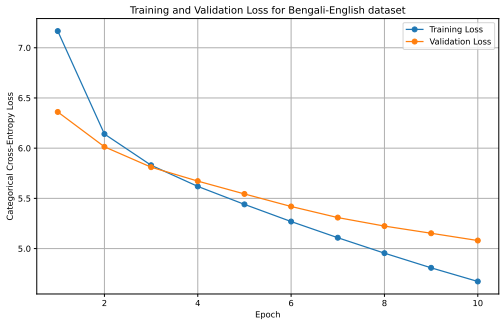
(d) Furthermore, after identifying a performance bottleneck through C profiling, efforts were made to optimize the process of finding corresponding target language words after obtaining logits. Default vocabulary structures in torchtext employ linear searches for token-to-number and number-to-token conversions, which can be computationally intensive with large vocabularies. To address this, a two-way dictionary was implemented, with one dictionary facilitating number-to-token conversions and another handling token-to-number conversions. This modification significantly reduced inference time, as the query time for finding corresponding tokens was reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$.

6 Results

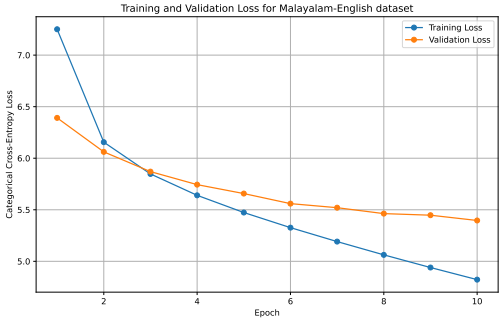
Hyperparameters of the final model:



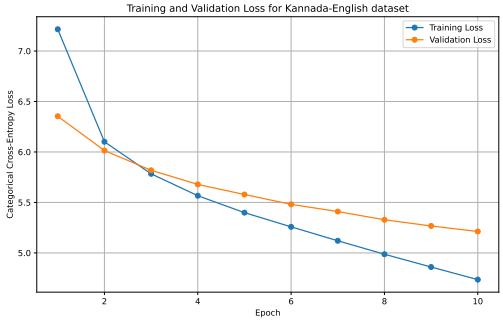
(a) Hindi-English



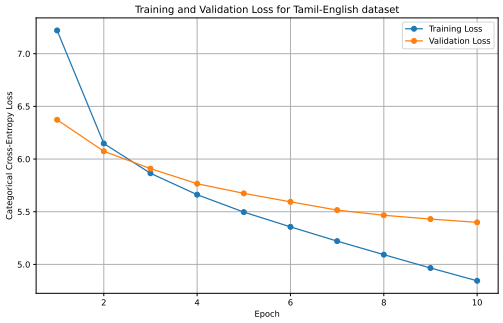
(b) Bengali-English



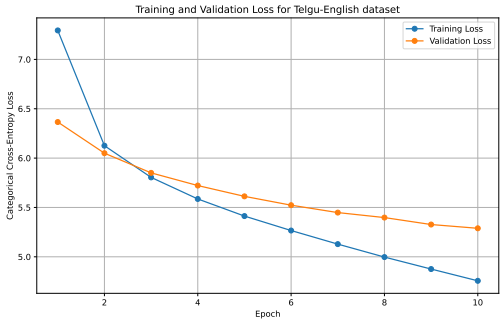
(c) Malayalam-English



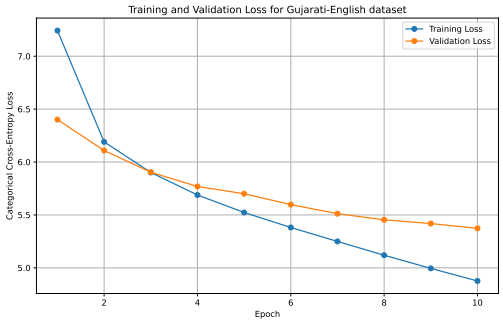
(d) Kannada-English



(e) Tamil-English



(f) Telgu-English



(g) Gujarati-English

Figure 3: Categorical Cross-Entropy Loss Plot for all 7 language pairs

	Phase 1		Phase 2	
Metric	Validation Set	Test Set	Validation Set	Test Set
CHRF Score	0.299	0.315	0.325	0.355
ROUGE Score	0.258	0.255	0.327	0.330
BLEU Score	0.089	0.103	0.109	0.129

Table 3: Evaluation Scores of final model for Phase 1 and Phase 2

7 Error Analysis

In this phase of NMT competitions, observations were made during the error analysis of the Transformer model. It was noted that the model exhibited remarkable adaptability to various Indian languages, effectively deciphering complex linguistic nuances and context, rendering it a reliable choice for translation tasks. Nevertheless, the Transformer model was not without its limitations; struggles were occasionally observed with less common words, leading to the generation of translations that were grammatically sound but contextually incorrect.

To address these limitations, various solutions were explored. One approach involved the incorporation of additional domain-specific datasets or in-domain training data, aimed at enhancing the model's vocabulary and comprehension of specialized terminology. Insights from attention heatmaps revealed areas where the model's focus faltered, guiding improvements in attention mechanisms for future enhancements. Ultimately, the analysis shed light on the strengths and weaknesses of the Transformer model in NMT, providing valuable insights for future research and advancements in the field.

8 Conclusion

The problem of translating seven distinct Indian languages into English, posed a captivating challenge in the domain of Natural Language Processing. Although baseline models demonstrated commendable performance, a series of precise optimizations were required to achieve an optimal balance between translation quality and efficiency. Notably, a customized rule-based tokenizer was crafted for the English language, and thorough performance profiling unveiled the time-consuming bottlenecks within the training loop. The exploration of various decoding techniques, including top-p sampling, beam search, and greedy decoding, each tuned with differing hyperparameters, played a pivotal role in finding the right trade-off between translation speed and accuracy.

While the model constitutes significant improvement, it is important to recognize that it is not exempt from limitations. It occasionally grapples with less frequent words and context, leaving room for further enhancements. Prospective research may entail the utilization of supplementary in-domain data, the integration of domain-specific training, and the application of advanced data augmentation techniques to bolster vocabulary and domain knowledge.

References

[1] S. Robertson, "Nlp from scratch: Translation with a sequence to sequence network and attention," in *Pytorch Tutorials*, 2019.

[2] I. Ezeani, P. Rayson, I. Onyenwe, C. Uchechukwu, and M. Hepple, "Igbo-english machine translation: An evaluation benchmark," in *ICLR*, 2020.