CS779 Competition: Machine Translation System for India

Edula Vinay Kumar Reddy
241110024
vinaykr24@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

This competition involves creating a machine translation system for English-Hindi and English-Bengali. The best model was achieved with a transformer-based encoder-decoder architecture, giving scores of CHRF: 0.30, ROUGE: 0.36, and BLEU: 0.11. Techniques like reversing input sequences during training (but not target sentences), usage of pre-trained GloVe embeddings of 300 dimensions for embedding layer initialization, and language-specific preprocessing were used. For training efficiency the model is trained with mixed precision mode.

1 Competition Result

Codalab Username: E_241110024
Final leaderboard rank on the test set: 1
charF++ Score wrt to the final rank: 0.30
ROGUE Score wrt to the final rank: 0.36
BLEU Score wrt to the final rank: 0.11

2 Problem Description

The problem involves developing a Machine Translation system for translating sentences from English to two Indian languages (Hindi or Bengali). The solution of this problem is essential for the growth of many non-English-speaking markets in India.

3 Data Analysis

1. **Training Data Description:** The training data is given as a JSON file with sentence pairs for example: English-Bengali, sentence pairs are organized by language pairs. Each entry within the dataset includes a unique identifier, a source sentence in English, and a corresponding target sentence in the target language (e.g., Bengali).
2. **Analysis and Insights About the Data:**
   (a) **Missing Values:** It is a complete dataset without any missing values.
   (b) **Sentence length distribution:** During model training we have to choose maximum sentence length, This plot helps us in understanding distribution of sentence lengths in the datasets. I have marked the 90th percentile as a reference. Based on the visualizations i have choosed max sequence length as 30.

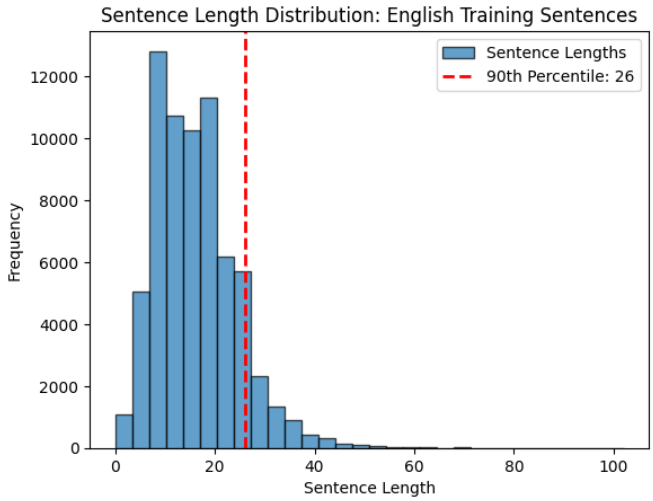

Figure 1: Sentence length distribution

(c) **Dataset statistics:** During the process of batching the data, we can able to only batch sequences of same length so we need to pad or truncate all the sequences to same length.

- Hindi Training: Sentences range from 1 to 214 characters, with an average length of 18.70 and a median of 17.
- English Training: Sentences vary from 0 to 102 characters, averaging 15.98, and the median length is 15.
- English Test: Sentence lengths span 0 to 99 characters, with a mean of 16.10 and a median of 15.
- Bengali Training: Sentences are between 1 and 75 characters long, averaging 14.19, and the median is 13.

(d) **Test Data:** The test data consists of two language pairs: English-Hindi with 23,085 test sentences and English-Bengali with 19,672 test sentences. These sentences are categorized under the Test data type in JSON file, predictions needs to be generated on this file.

4 Model Description

(a) **Model Evolution:** The model evolved in the following stages:

- **Initial Trials:** Started with simple sequential-to-sequential model, specifically an RNN encoder and decoder. Then shifted to a bidirectional GRU-based encoder-decoder architecture with attention, followed by experiments with language-specific tokenization and hyper-parameter tuning. Tried to implement byte pair encoding for tokenization so that we can effectively tokenize out of vocabulary words but due to compute constraints in colab couldnt integrate it in further models.
- **Transformer Architecture:** During the test phase, transitioned to a transformer-based encoder-decoder architecture[2]. Improvements included:
  - Reversal of input sequences and keeping target sentences constant so that many inter dependencies will be formed between source and target language which makes optimization problem easier[Inspired from the [1]]
  - Used Indic NLP library for indian language specific normalization and tokenization before starting actual task.
  - Pre-trained GloVe embeddings (300-dimensional) for initializing English word embeddings. Next made it learnable so that model can adjust embeddings
  - Mixed precision training, which reduced training time by 5 times while maintaining accuracy and stability.

- Used GradScaler which scales the gradients dynamically to prevent numerical underflow during backpropagation with lower-precision floating-point numbers.
- Used autocast context which enables computations to be performed in FP16 (half-precision) where possible, improving training speed and reducing memory usage
.

3. **Model Objective:** The model uses cross-entropy loss with padding masked to exclude PAD tokens from contributing to the loss.
4. **Inference:**Greedy decoding is used during inference, where the most probable token is selected at each step using `argmax`.

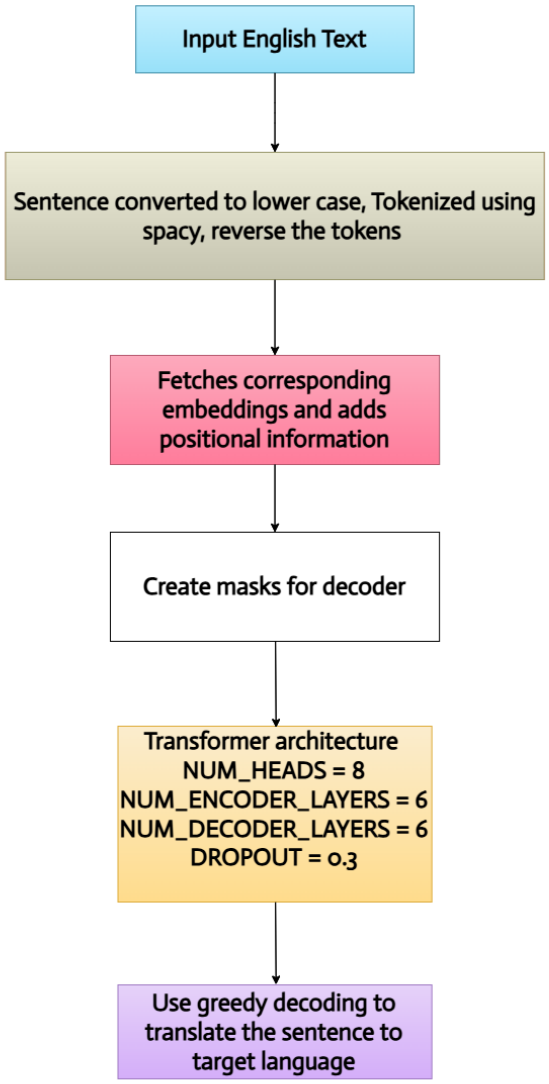

Figure 2: High Level Model architecture during Inference

5 Experiments

1. Data Pre-processing

- **English Preprocessing:**

  - Sentences were converted to lowercase to reduce vocabulary size and complexity of task.
  - Each sentence was tokenized using spaCy, retaining only alphabetical tokens (removing punctuation and numbers).
  - The order of tokens was reversed in each sentence to introduce shorter dependencies between source and target sequences,as suggested in the referenced paper[1].
  - Choosed max sequence length as 30 after performing 90 percentile sentence length analysis. Based on that sentences were truncated or padded using PAD tokens.

- **Hindi Preprocessing:**
  - Sentences were normalized using the `IndicNormalizer` to handle inconsistencies like Unicode normalization common in Hindi text.
  - Numbers and punctuation were removed to simplify the vocabulary and focus on meaningful content.
  - Sentences were tokenized using a tokenizer from the Indic NLP library, which is specifically suited for Indic languages.
  - Vocabularies for source (English) and target (Hindi) languages were created by collecting all unique tokens in the datasets. Special tokens (`<PAD>`, `<SOS>`, `<EOS>`) were added to handle padding, sentence start, and sentence end during training.
  - Word-to-index and index-to-word mappings were generated to convert sentences into numerical format for model training and decoding predictions back into text.

- **Reason for Chosen Pre-processing:** Converting to lowercase and normalizing text ensures consistency across the dataset, reducing noise and vocabulary size simplifying the model's learning task.

2. Training Procedure [Final Model]

- **Optimizer:** Adam optimizer was used with a learning rate of 0.001, and weight decay of 1×10^{-5}.
- **Learning Rate Scheduler:** ReduceLROnPlateau was employed to dynamically reduce the learning rate with patience set to 5 epochs.
- **Loss Function:** CrossEntropyLoss was used for the sequence-to-sequence transformer model.
- **Batch Size:** 64 for both training and validation phases.
- **Epochs:** Training was conducted for up to 50 epochs, with early stopping applied if validation loss did not improve over 5 epochs.
- **Mixed Precision Training:** Enabled using PyTorch's `torch.cuda.amp` module to accelerate training and reduce memory usage on the GPU.
- **Embedding Initialization:** Pre-trained GloVe embeddings were used for the source language, while embeddings for the target language were learned during training.
- **Training Time:** Approximately 3 minutes per epoch with mixed precision enabled on a GPU.

3. Hyper-parameters for Different Models

Selection Process: Hyper-parameters were fine-tuned via manual experimentation and validation loss tracking. Dropout rates, embedding dimensions, and learning rates were adjusted iteratively to balance performance and computational efficiency.

Model Component	Hyper-parameter	Value	Reason for Selection
Embedding Layer	Embedding Dimension	300	Matches GloVe pre-trained embeddings.
	Dropout	0.1	Prevents overfitting.
	Positional Encoding	Sinusoidal	Adds positional information to tokens.
Transformer	Encoder Layers	6	Followed original architecture[2],
	Decoder Layers	6	Followed original architecture[2],
	Attention Heads	8	Followed original architecture[2],
	Feedforward Size	512	Ensures adequate capacity for learning.
Training	Batch Size	64	Due to GPU memory constraints.
	Learning Rate	0.001	Starting point for optimization later adjusts based on validation.

Table 1: Hyper-parameters for the Transformer Model

6 Results

1. **Results of Different Models on Dev Data:**

Model	**CHRF Score**	**ROUGE Score**	**BLEU Score**	**Rank**
Bidirectional GRU-based encoder and GRU-based decoder with attention	0.124%	0.25%	0.02%	6
Language specific tokenization and Bidirectional GRU-based encoder and GRU-based decoder with attention	0.2%	0.30%	0.07%	3
Hyper parameter tuning + Language specific tokenization and Bidirectional GRU-based encoder and GRU-based decoder with attention	0.245%	0.30%	0.07%	2

Table 2: Performance of Created Models on Dev Data

2. **Results of Different Models on Test Data:**

Model	CHRF Score	ROUGE Score	BLEU Score	Rank
Transformer Architecture	0.296%	0.39%	0.10%	2
Transformer Architecture + Reverse Input + Initializing with glove embedding	0.304 %	0.36%	0.11%	1

Table 3: Performance of Created Models on Test Data

3. **Explanation of Results:** Based on my experimentation i have observed the increase in scores as i try to do architectural improvements and preprocessing. The initially proposed bidirectional GRU-based encoder-decoder with attention had very limited capability, with a BLEU score of 0.02%, mainly because even thought it has gates to control the memory due to limited dimensionality of hidden vectors it cannot capture long-range dependencies effectively.Use of language specific tokenization and some hyperparameter tuning resulted in an improved performance, which yielded a BLEU score of 0.07%. Transformer-based architecture has attention mechanism which captures global context much better, due to this fact it has achieved better scores comparatively. Further refinements, such as reversing input sequences and initializing embeddings with pre-trained GloVe vectors, further improved alignment and semantic understanding, yielding the best BLEU score of 0.11% and a CHRF score of 0.304%.

7 Error Analysis

1. **Error analysis of different models on the dev set and the test set:**
   - The initial RNN-based encoder-decoder performed poorly, Later added attention mechanism which helped the model to concentrate on key features this led to moderate gains in BLEU and ROUGE scores.
   - Introducing language-specific tokenization improved results by focusing on relevant linguistic features and reducing noise.
   - The Transformer-based encoder-decoder architecture achieved the best results on test set due to its ability to model global dependencies through multi-head attention.
   - The use of reverse input and GloVe embeddings further enhanced performance(from 0.296 to 0.304), optimizing the alignment of source and target sentences.
2. **Reasons for imperfections:**
   - The errors included mistranslations of rare words due to lack of training data for such cases.
   - In my approach, inference was done using greedy decoding mechanism, which limited the model's ability to generate diverse translations.
3. **Interesting insights:**
   - Reversing the input sequence proved effective(Score increased from 0.296 to 0.304) for improving alignment between source and target sentences, as it simplified dependencies.
   - Mixed precision training keeps the computational cost low allowing faster experimentation..
   - Pre-trained embeddings such as GloVe adds semantic richness to the model, even model can improve the embeddings during the course of training.

8 Conclusion

In this competition i have developed a machine translation system for English-to-Hindi and English-to-Bengali, achieving the best performance with a transformer-based encoder-decoder model. Some

of the reasons for this performance is reversing input sequences, initializing embeddings with pre-trained GloVe vectors, and applying language-specific preprocessing techniques like tokenization and normalization. The model scored CHRF 0.304, ROUGE 0.36, and BLEU 0.11 on the test set, significantly outperforming earlier RNN-based models. Improvements like mixed precision training reduced computational costs, while attention mechanisms effectively captured global dependencies.

Key Findings:

- The Transformer-based architecture with reverse input and GloVe embeddings outperformed RNN-based models on this dataset.
- Language-specific preprocessing, including tokenization and normalization was critical for improving translation accuracy.

Recommendations and Future Directions:

- Incorporate beam search decoding instead of greedy decoding to enhance translation diversity and accuracy.
- Use larger and more diverse datasets to improve the handling of rare words and complex grammatical structures.
- Explore fine-tuning on pre-trained transformer models like mBERT or IndicBERT for domain-specific enhancements.
- Experiment with multilingual pre-trained models to leverage shared linguistic features across multiple languages.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv preprint*, arXiv:1409.3215, 2014. [Online]. Available: `https://arxiv.org/abs/1409.3215`.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv preprint*, arXiv:1706.03762, 2017. [Online]. Available: `http://arxiv.org/abs/1706.03762`.