CS779 Competition Phase 2: Machine Translation System for India

Rajarshi Dutta
200762
rajarshi20@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

Neural Machine Translation represents a revolutionary shift from the traditional statistics-based, linguistics-based approach to modern deep-learning-based approaches. This method employs a **encoder**, **decoder** architecture which takes in source languages input and produces the output in the desired language. The main advantage of NMT compared to previous probabilistic, rule-based methods lies in its handling long-term dependencies, enhanced abstraction (the entire intricacies is dealt by the network itself which leaves behind little scope for manual tuning), semantic-nuances and use of fewer tunable parameters. The phase 2 of this challenge presents NMT from Indian languages to English languages which brings in certain difficulties due to morphological variations, syntactic diverseness in this languages which essentially leads to the poor performance of the traditional approaches, thereby leveraging neural networks to facilitate translation process. This challenge also paves the way towards development of a multilingual approach to machine translation between diverse low resource languages.

1 Competition Result

Codalab Username: R_200762
Final leaderboard rank on the test set: 20
charF++ Score wrt to the final rank: 0.215
ROGUE Score wrt to the final rank: 0.241
BLEU Score wrt to the final rank: 0.017

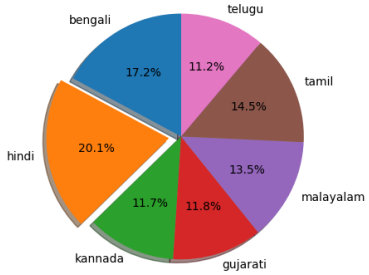
2 Problem Description

The entire challenge consists of two phases. The **Phase 1** includes designing/implementing neural machine translation models for translation from English to 7 different Indian languages. The**Phase 2** consists of implementing NN based models for creating translations back from those 7 Indian languages to English. The models are to be trained on train dataset and validation is done on **validation** dataset for each of the 7 Indian languages. The final evaluation of the best performing model is carried out on a separately curated test set comprising of many instances of all 7 seven Indian languages. The model is then evaluated based on several NMT based metrics like **BLEU**, **charF++** and **ROUGE** scores and ranked correspondingly. This implementation can be further extended to handle other morphologically diverse, low resource languages to devise a robust multilingual NMT system.

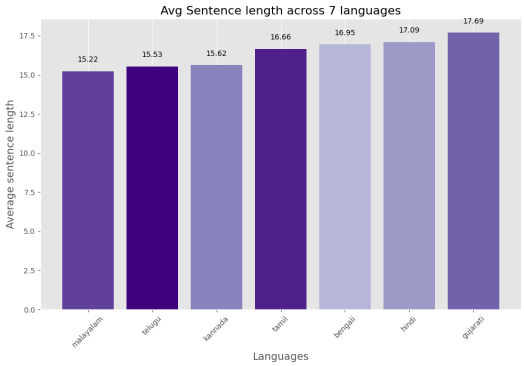
3 Data Analysis

1. The training dataset consisted of sentences in english and 7 different indian languages like **bengali, hindi, malayalam, telugu, kannada, gujarati**.
2. Data Analysis involved visualizing and interpreting the distributions of sequence lengths for the case of 7 languages via a **KDE plot**. The total contribution of all the languages in terms of data size is almost equal with **Hindi** having the largest number of training samples. Other steps included calculating the average sentence length in each language, eliminating missing/null

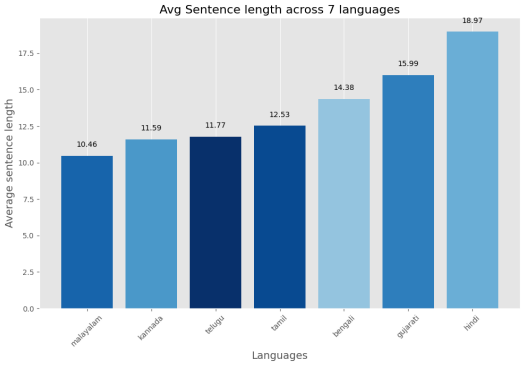
values, some common **statistical analysis** like finding out **word/phrase frequencies** common to train and val sets, **out-of-vocabulary analysis** The data preprocessing steps involved removal of **stop words**, **special characters**, punctuations and eliminating the common phrases.



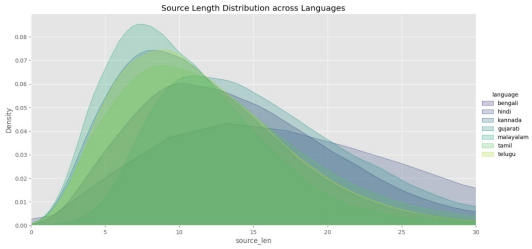
(a) Volume Analysis of 7 languages



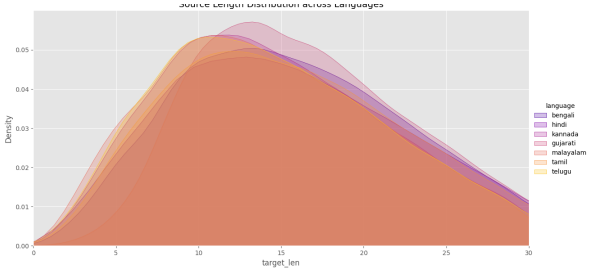
(b) Average Source Length Distribution



(c) Average Target Length Distribution



(d) Source Length Distribution



(e) Target Length Distribution

3. The next step was the vocabulary creation for each of the languages using the Pytorch Field class. The average size of the vocabulary for each of the languages was around **50000**.

4 Model Description

1. **LSTM Sequence to Sequence model**
   The first approach([1]) included implementing a basic **LSTM (Long Short Term Memory)** based Sequence to Sequence model. The LSTM module helps to mitigate some common RNN based issues like **vanishing gradients**, **exploding gradients**, inability to capture long term

dependencies through the flow of information through the three gates (**input**, **forget** and **output**). The input sequence was compressed down to a fixed size context sequence which lead to information loss for long sequences. The model essentially creates this context sequence through the **encoder** and for the translation part takes in the context sequence from **encoder**, thereby predicting target word by word. For each step, it takes the previous word's output and the hidden state as input and produces the next word in the sequence. The decoding strategy may vary according to the implementations like greedy decoding, beam search etc. There are nevertheless certain drawbacks to this approach like **LSTMs** are extremely prone to overfitting because of their complex structure and large number of parameters which makes them bad candidates for online learning tasks and they also require a large amount of training and inference time which poses a major challenge in their application in soft computing devices with resource constraints. Given below is a diagram depicting the architecture of an LSTMSeq2Seq model:

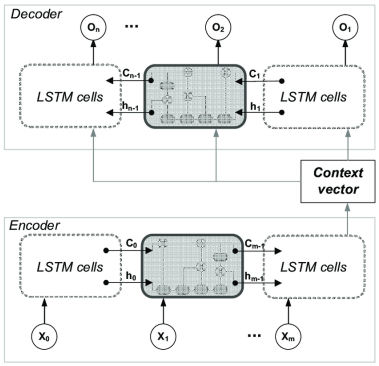


Figure 2: LSTM Seq2Seq architecture

2. **LSTM Sequence to Sequence model with Bahdanau Attention**

The second approach included trying out with the attention mechanism to cope up with worse loss values and bad translations since the prior model produced a pretty low **BLEU** score. **Attention** mechanism really helps in creation of a dynamic context representation which helps the decoder to focus on specific parts of the input to improve its translations. The main advantages of the **attention** mechanism is to reduce complexity, memory usage, information loss and focus on important parts of the input which further enhances interpretability and also helps in visualizations. In this model, I used a **Bidirectional LSTM**([2]) and implemented the **Bahdanau's Attention** mechanism which utilizes a **Bidirectional LSTM** to capture dependencies of both preceding and succeeding words. Here, the **attention weights** between each decoder hidden state and updated hidden state from the encoder obtained by the concatenation of both the forward and backward hidden state **(hidden_size * 2 + embedding_size)** is calculated through a trainable **dense** layer. The final output of the decoder originates by passing the updated context vector through a dense layer which outputs the token with maximum probability from the set of target tokens.

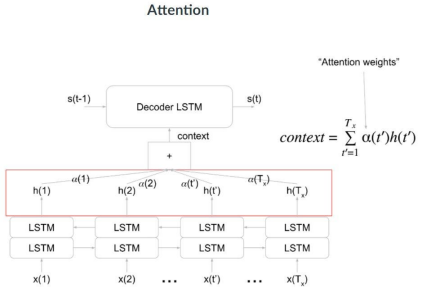


Figure 3: LSTM with attention diagram

3. **GRU Sequence to Sequence model with Bahdanau Attention**

   The third implementation is the same as the previous with the only replacement of **LSTM** block with a **GRU (Gated Recurrent Unit)** since the previous architecture essentially took a long training time due to the complex architecture of **LSTM** block and presence of a larger number of gates. **GRU** has mainly two gates ,the **input** and **reset** gates and it seemed to improve both training speed and translations due to its efficient, lightweight architecture.

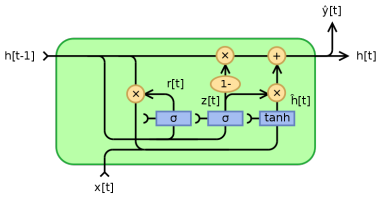


Figure 4: GRU architecture

4. **GRU Sequence to Sequence with Luong Attention**

   The **Luong attention** mechanism calculates alignment scores using source and target hidden states. The encoder yields hidden states for each input element, while the decoder produces a new state from the previous one. Attention weights, derived from the encoder and decoder states, form a probability distribution after applying softmax. This probability distribution multiplies with the encoder states to yield the context vector. Finally, the context vector and decoder states concatenate and pass through a neural network to produce translation tokens.

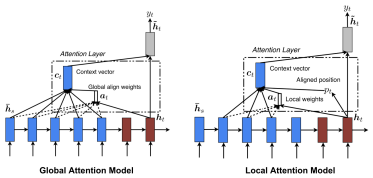


Figure 5: Luong Local and Global Attention mechanism

5. **Transformer model**

   This is essentially regarded as the **state-of-the-art** in neural machine translation([3]). The **transformer** model has a novel architecture comprising of **encoder** and **decoder**. I followed the **sinusoidal embeddings** based approach for **positional embeddings** and used a trainable dense layer for **token embeddings**([4]). This model includes three trainable tensors named as **Query**, **Key**, **Value**. Initially the **self-attention** mechanism involves calculation of dot product of current word indicated by **query** with the set of words related to it **keys** resulting in values. The **context vectors** are calculated by the weighted sum of attention weights and values which creates a final vector for each word in the sequence for prediction. I have used both 3 stacked encoders and decoders to construct the transformer model. Also unlike seq2seq models, this approach takes the input embeddings in a parallel fashion which improves efficiency.

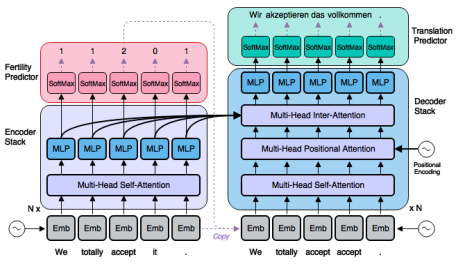


Figure 6: NMT Transformer diagram

6. **Loss function**

   The **loss function** used in all of the above models includes a **Cross Entropy** loss for training. The **Cross Entropy** loss had an additional ignore_index parameter to ignore the **padding** token index.

7. **Decoding Strategies**

   The decoding strategies used include the following:

   (a) **Greedy Decoding**: This decoding step essentially is the most basic of all. The **decoder** at each time step outputs the **softmax** probabilities of collection of tokens of size equal to the target vocabulary size. This decoding strategy selects the token with the maximum probability as the predicted token in the target language during training and inference. This approach suffers from various issues like not penalizing longer sequences having smaller **joint probabilities**, **localization**, producing **suboptimal predictions** etc.

   $$w_t = \arg\max_w P(w|w_1, w_2, \ldots, w_{t-1}) \tag{1}$$

   (b) **Beam Search Decoding**: This decoding strategy([5]) is essentially heuristic since it selects a set of k tokens (k being the **beam width**) which is a tunable parameter. The decoding strategy essentially expands the sequence for each of the selected tokens and leads to both probable translations other than the greedy based one. This decoding helps to mitigate the low probabilites in longer sequences. It has certain disadvantages like less efficient more memory, repetitive translations, prune errors etc.

   $$S_{t+1}^{(i)} = \arg \top_k P(w|S^{(i)}) \times P(S^{(i)}) \tag{2}$$

   (c) **Top K Sampling**: This decoding method([6])essentially selects top k tokens out of all the probabilities over the target vocabulary size. It then randomly samples a token from this set which serves as the word for the next prediction in the time step. This strategy leads to diversity by introducing randomness in the decoding and also avoids highly improbable tokens but still can lead to meaningless tokens.

   $$w_t \sim P(w|w_1, w_2, \ldots, w_{t-1}) \quad where \quad w \in top{-}k{-}tokens \tag{3}$$

   (d) **Nucleus Sampling**: The fourth approach selecting w from the subset of tokens whose cumulative probability is less than or equal to a threshold p. This subset is not determined by a fixed number k, but dynamically based on the cumulative probability exceeding the threshold p.

   $$w_t \sim P(w|w_1, w_2, \ldots, w_{t-1}), \quad w \in nucleus \quad \& \quad nucleus = \{w' : \sum_{w'' \in S(w')} P(w'') \leq p\} \tag{4}$$

   The greedy decoding led to wrong translations especially in case of the **LSTM** based sequence to sequence model. I tried using the **Beam search decoding** which led to a slight improvement in translations but some of them turned out to be repetitive. I have also tried using all the four mentioned approaches for the case of the **transformer** model but **beam search decoding** led to extremely repetitive translations and **top k sampling** led to various randomized translations which ultimately led to worse **BLEU** score which checks on token similarity irrespective of their ordering. Hence I resorted to the normal **greedy decoding** after these explorations.

5 Experiments

1. The analysis included basic preprocessing of the data like removal of **stop words**, like converting all the text into **lower letters**, removing the words between brackets (), special characters and **extra white spaces**. Then the vocabulary for the source and target were created using **Pytorch's Field** variables using only **unigrams**. The **train iterators** and **val iterators** were further created from **Pytorch's TabularDataset** object.

2. Training procedure: The training procedure included both the training and validation loops. The loss values were calculated after every 100 iterations and a sample translation for a random sentence of the corresponding language was monitored for every epoch. The **training loop** was integrated with a early stopping mechanism to stop the training process if the validation loss started increasing after a mentioned **patience** value of **3**. The hyperparameters and their optimal values obtained for each of the models are mentioned as follows:

Hyperparameters	Value
Optimizer	**Adam**
Learning Rate	**0.005**
Epochs	**25**
Hidden Size	**256**
Embedding Size	**512**
Teacher Force Ratio	**0.3**

Table 1: LSTM Seq2Seq model

Hyperparameters	Value
Optimizer	**Adam**
Learning Rate	**0.009**
Dropout Rate	**0.03**
Epochs	**40**
Hidden Size	**200**
Embedding Size	**100**
Teacher Force Ratio	**0.4**
Encoder Input Size	**source vocab size**
Decoder Input Size	**target vocab size**

Table 2: LSTM Seq2Seq with Attn model

Hyperparameters	Value
Optimizer	**Adam**
Learning Rate	**0.003**
Dropout Rate	**0.2**
Epochs	**25**
Hidden Size	**400**
Embedding Size	**200**
Teacher Force Ratio	**0.4**
Encoder Input Size	**source vocab size**
Decoder Input Size	**target vocab size**

Table 3: GRU Seq2Seq with Attn model

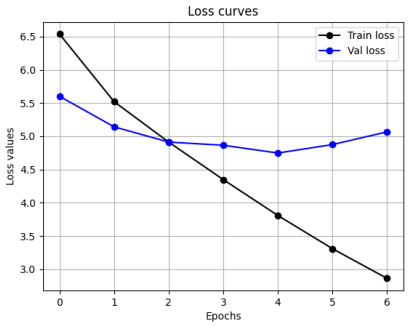
Hyperparameters	Value
Optimizer	**Adam**
Learning rate	**0.0003**
Epochs	**25**
Attention Heads	**8**
Embedding Size	**512**
Feed forward Dimension	**256**
Encoder Layers	**2**
Decoder Layers	**2**
Dropout Rate	**0.01**

Table 4: Transformer model

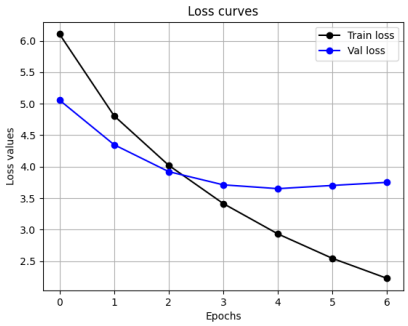
These hyperparameters were essentially selected mostly via intuitions, **hit and trial** based methods. I even thought of using **weights and biases (wandb)** which is decent library for selecting models with best hyperparameters keeping **validation loss** as the selection criteria but since it relies on a grid search on required sample space, it was not an intelligent choice due to hard constraints on the available computation resource thanks to its heavy **GPU RAM usage**. These resource constraints immediately tempted me to apply **randomized Bayesian** sampling of the hyperparameters since random sampling is far less memory intensive when compared to searching the whole hyperparameter space for optimal values though random sampling may rule out good hyperparameter sets as well.

6 Results

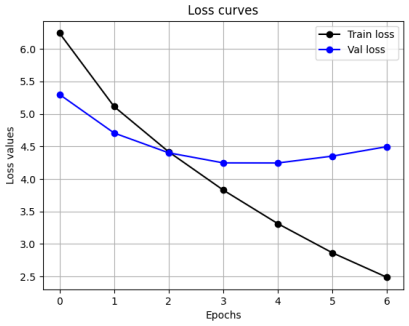
1. The evaluation scores for my last submissions made during the phase 2 is summarized in this table:
2. The best performing model is the **transformer** because it captures the context around a given word in the source language efficiently due to the **self-attention** mechanism build in it.
3. The loss curves I obtained for different languages for the transformer models on the **test dataset** are given below:
4. The following transformer models were trained on the training dataset and evaluated on the validation data on the **dev phase** provided in CodaLab. The metrics chosen for evaluation were **rouge score**, **BLEU score**, **charF++ score**. The main score for leaderboard ranking was the



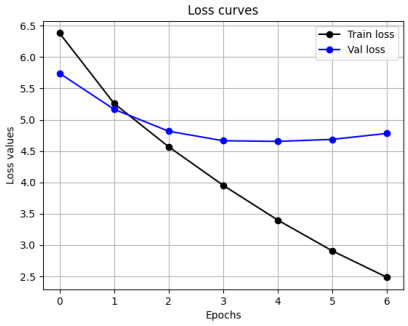
(a) Loss curves for Gujarati



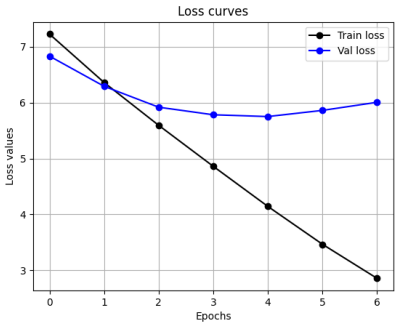
(b) Loss curves for Hindi



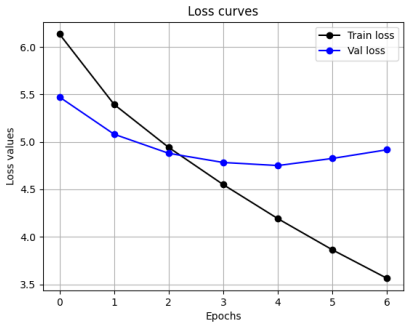
(c) Loss curves for Bengali



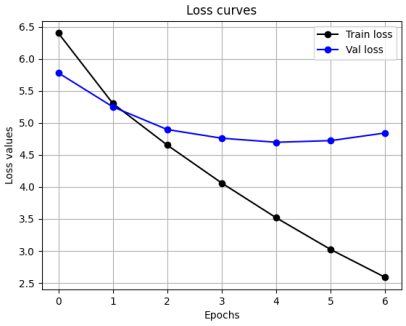
(d) Loss curves for Kannada



(e) Loss curves for Malayalam



(f) Loss curves for Tamil



(g) Loss curves for Telugu

Figure 7: Loss curves for languages - English training

Model	charF++_score	rouge_score	BLEU_score
LSTMSeq2SeqAttn	**0.130**	**0.102**	**0.001**
GRUSeq2SeqAttn	**0.142**	**0.234**	**0.011**
GRUSeq2SeqAttn	**0.145**	**0.219**	**0.014**
Transformer	**0.215**	**0.241**	**0.017**

Table 5: Stats for training phase

BLEU score which indirectly measures the unigram similarity between the ground truth and the translated sentence irrespective of ordering.

7 Error Analysis

1. The behaviour of a model can vary based on the training set. Without a varied training dataset, they might not be able to generalise. In rare cases, overfitting in the training dataset might result from an incredibly intricate and deep architecture. The model's performance is substantially decreased by the classic **RNN** based approaches' issues with vanishing gradients. Occasionally, models underperform because it might be challenging to determine the optimal combination of hyperparameters needed to maximise performance. In language-based tasks, the model may also be hampered by situations like as excessive biases in the data, ambiguities in the data, etc. Providing for all of these use cases at once may seem challenging.

   In this task mostly the best performing models were **GRU Sequence to Sequence with Attention** due to is efficiency and the state-of-the art **transformer** performed quite well in predicting good translations.

8 Conclusion

The machine translation challenge results showed that because of the **multi-head attention** block and its ability to facilitate parallelization, transformers outperform conventional sequence to sequence models in terms of performance. The findings also showed that tokenizers for native Indian languages are not as well established, and that each language requires a significant amount of internal model hyperparameter tuning. These factors contribute to the difficulty of translation into local Indian languages. Overall, I would still advise utilising transformer-based models because they yield far better translations.

References

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

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[4] PyTorch, "Transformers tutorial," Year of access.

[5] R. Leblond, J.-B. Alayrac, L. Sifre, M. Pislar, J.-B. Lespiau, I. Antonoglou, K. Simonyan, and O. Vinyals, "Machine translation decoding beyond beam search," 2021.

[6] D. Li, R. Jin, J. Gao, and Z. Liu, "On sampling top-k recommendation evaluation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*, ACM, aug 2020.