CS779 Competition: 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, linguisitcs-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, semantic-nuances and use of fewer tunable parameters. This challenge presents NMT from english to indian languages, presenting difficulty 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 facililate translation process.

1 Competition Result

Codalab Username: R_200762
Final leaderboard rank on the test set: NA
charF++ Score wrt to the final rank: NA
ROGUE Score wrt to the final rank: NA
BLEU Score wrt to the final rank: NA

2 Problem Description

The entire challenge consists of two phases. The **Phase 1** includes attempting machine translation models on translation from english to 7 different indian languages whereas the **Phase 2** consists of the implementaion of reverse translations. The model is to be trained on train dataset and validation is done on **dev** dataset for each of the languages. The final evaluation of the model is carried out on a test set consisting of english languages and a key representing the language it is to be translated to.

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. No special data analysis was done on while training on individual languages. Only normal preprocessing and removal of **stop words**, **special characters**, punctuations were carried out as a basic data cleaning step.
3. The next step was the vocabulary creation for each of the languages

4 Model Description

1. **LSTM based Sequence to Sequence model**
   The first approach([**?**])included utilizing a basic **LSTM (Long Short Term Memory)** based Sequence to Sequence model. The LSTM module helps to mitigate the **vanishing gradients**,

exploding gradients, inability to capture long term dependencies issue in case of the RNN module 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 first 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.

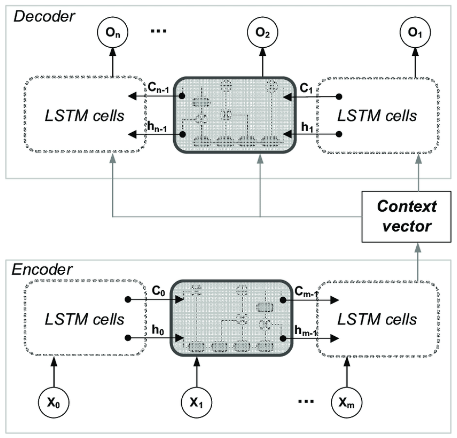


Figure 1: LSTM Seq2Seq architecture

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

   My second approach included trying out with the attention mechanism to cope up with the loss values and improving translations since I was obtaining a pretty low **BLEU** score with the **LSTM seq2seq** model. **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 translation. In this model, I used a **Bidirectional LSTM**([**?**]) and implemented **Bahdanau's Attention** mechanism which utilizes a **bidirectional** lstm to capture both preceding and succeeding words. Here, the attention score 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. This implementation seemed to improve the translations by a significant amount.

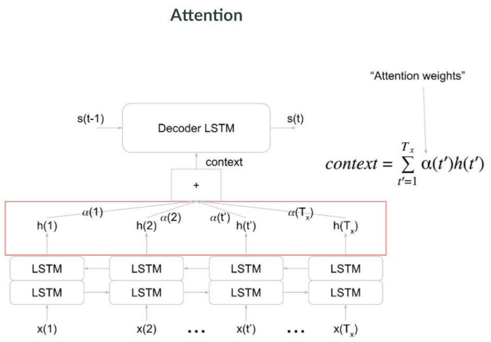


Figure 2: LSTM with attention diagram

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

   This implementation is the same as the previous with the only change done to the **LSTM** block replacing it 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, **input** and **reset** gates and it seemed to improve both training speed and translations due to its efficient, lightweight architecture.

4. **Transformer model**
   This is essentially regarded as the **state-of-the-art** in neural machine translation([**?**]). 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**([**?**]). This model includes three representations like **Query**, **Key**, **Value**. Initially the **self-attention** mechanism indicates multiplying the 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 takes the input embeddings as a whole which improves efficiency.

5. **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.

6. **Decoding Strategies**
   The decoding strategies used includes 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**, **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([**?**]) 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([**?**])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.

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

   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 three approaches in case of the **transformer** model but **beam search decoding** led to extremely repetitive translations and **top k sampling** led to various meaningless translations hence I resorted to the normal **greedy decoding** after much 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 parameters and choice of theses parameters are listed in the table below

Hyperparameters	Value
Optimizer	**Adam**
Learning Rate	**0.002**
Epochs	**30**
Hidden Size	**50**
Embedding Size	**512**
Teacher Force Ratio	**0.1**

Table 1: LSTM Seq2Seq model

Hyperparameters	Value
Optimizer	**Adam**
Learning Rate	**0.001**
Dropout Rate	**0.1**
Epochs	**40**
Hidden Size	**600**
Embedding Size	**300**
Teacher Force Ratio	**0.1**
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	**0.001**
Dropout Rate	**0.1**
Epochs	**40**
Hidden Size	**600**
Embedding Size	**300**
Teacher Force Ratio	**0.1**
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	**0.0001**
Epochs	**30**
Attention Heads	**8**
Embedding Size	**512**
Feedforward Dimension	**256**
Encoder Layers	**3**
Decoder Layers	**3**
Dropout Rate	**0.1**

Table 4: Transformer model

These hyperparameters were essentially selected via mostly intuitions and **hit and trial** based methods. I even though of using **weights and biases (wandb)** which is a good library for selecting models with best hyperparameters keeping **validation loss** as the selection criteria but since it relies on a grid search on sample space, my computational resource didn't permit it due to heavy **GPU RAM usage**. This resource constraint immediately crossed out the scope of applying **randomized bayesian** sampling of the hyperparameters as well.

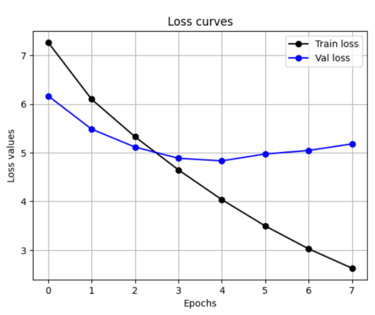
6 Results

1. I was facing some issues with the transformer implementation. I tried out various decoding strategies like greedy, **beam search**, **top k sampling**, **nucleus sampling**, also made some tweaks in the transformer architecture but was unable complete the evaluation on the test phase. Hence, I was unable to test my best model/transformer (which produced good translations on almost all of the Indian languages) on the test dataset released after the training phase got ended so I didn't get a proper leaderboard position. The leaderboard score for the last submission on te dev set is:
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. It can be seen from the results on the test the performance of the transformer is not that good especially on the **tamil** dataset mostly because of the tokenization issues present in it so the decoder is not quite able to decode it given a smaller number of epochs of training.

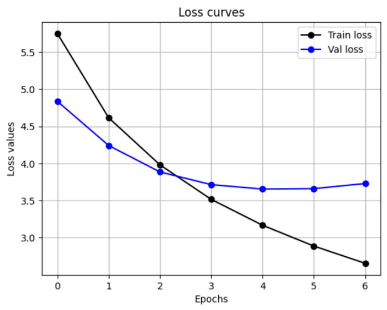
Model	chrf_score	rouge_score	bleu_score
GRUSeq2SeqAttn	**0.179**	**0.239**	**0.023**
GRUSeq2SeqAttn	**0.161**	**0.205**	**0.013**
GRUSeq2Seq	**0.176**	**0.188**	**0.009**
LSTMSeq2SeqAttn	**0.500**	**0.595**	**0.009**
LSTMSeq2Seq	**0.188**	**0.119**	**0.008**

Table 5: Metrics on dev phase

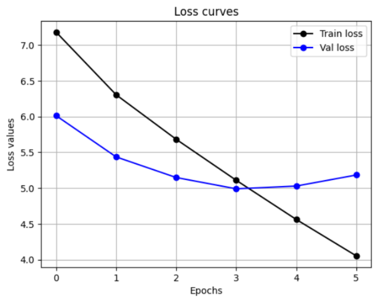
4. The loss curves I obtained for different languages for the transformer models on the **test dataset** are given below:



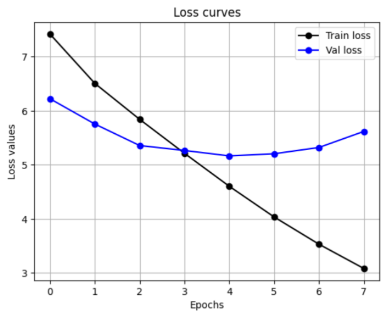
(a) Loss curves for Bengali



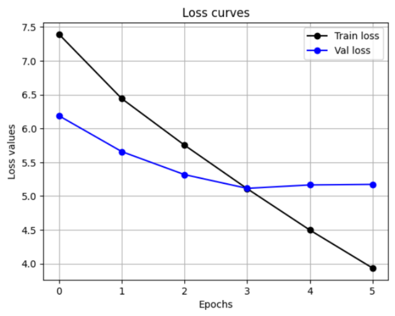
(b) Loss curves for Hindi



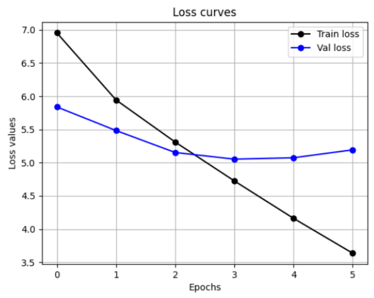
(c) Loss curves for Malayalam



(d) Loss curves for Telegu

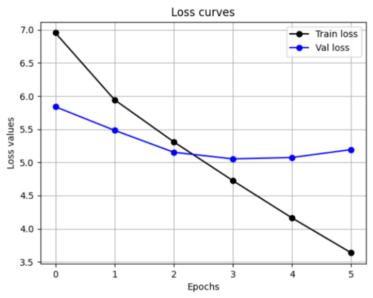


(e) Loss curves for Kannada



(f) Loss curves for Gujarati

Figure 3: Six plots arranged in two rows



(a) Loss curves for Tamil

5. The following models were trained on the training dataset and evaluated on the validation data on the **dev phase**. The metrics chosen for evaluation were **rouge score**, **BLEU score**, **chrf score**

7 Error Analysis

1. Models can act differently depending on training data. It may not be able to generalize in the absence of a diverse training dataset. An extremely deep and complex architecture can sometimes lead to overfitting in the training dataset. The traditional **rnn** based methods essentially suffers from problems like vanishing gradients which decreases the performance of the model. In some cases, models don't perform upto the mark because it's quite difficult to find the most accurate set of hyperparameters required for its improvement in performance. In languages based tasks, sometimes cases like excess biases in the data, ambiguities in data etc. can sometimes hamper the model as well. Catering to these use cases all at once can seemingly be difficult.

   In this task mostly the best 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 led to the following conclusion that transformers perform way better than normal sequence to sequence models due to the **multi-head attention** block and it also enables parallelization. The results also indicated that translation into native Indian languages are quite difficult because the tokenizers are not so well developed for these languages and model internally needs a heavy amount of hyperparameter tuning for each of the languages. I also fund out that translation to **tamil** surprisingly requires a very small **batch size** and the validation loss seems to increase progressively. Overall still I would recommend using transformer based models because they provide much improved 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.