COL870 PROJECT REPORT

BY

KURISETI RAVI SRI TEJA - 2019CS10369

ABHISHEK AGGARWAL - 2019PH10609

Contents

1 Introduction

Google language translator is widely used for translation from one language to another. It supports back-and-forth conversion between more than 100 languages like Bengali, Bhojpuri, French, and more. In fact, google translate powers much more sophisticated google applications like google lens. Owing to the technological prowess of google and the data it collects from the customer, Google Translate is state of the art in the field of machine translation. The architectural and training details of the translator have been given in this **paper**.

In this paper, we talk about our implementation of the above-mentioned Google Translator. Since the architecture is detailed in the original paper, we will mostly focus on our implementation i.e. the datasets we used, how we worked around the hardware constraints, and the results we achieved. We will be covering translation from English to Hindi.

2 Architecture

Google Translator comprises of three main components-

1. Encoder
2. Attention
3. Decoder

Encoder-Encoder is used to convert the input sentence into an embedding vector. It is made up of multiple LSTMs layers. Each layer is implemented parallelly on different GPUs. The original implementation used 8 layers, one on each GPU.
The initial two layers are initialization layers. They take words directly

from the input sentence and output a vector. The first layer takes input from left to right in the forward direction. The second layer is reversed; it takes input from right to left.
All the layers afterward takes input in the forward direction from left to right. There are skip connections in the later layers, as it is usually observed that adding skip connections in deep networks improves the accuracy. The final layer of LSTM outputs an embedding vector encoding the input sentence.

Attention- Attention is used to teach the model to focus more on the specific region of the sentence. It does that using a neural network that outputs attention weights α. Google Translator uses Bahadanau Attention detailed in the paper **Bahadanau Attention**. The attention neural network inputs the embedding by the encoder along with the output of the initial layer of the decoder to output the attention vector.

Decoder- Decoder network is similar to the encoder network; it is made up of 8 LSTM layers, each on a different GPU. There is a slight difference from the encoder network. In the end, there is a softmax layer; hence, instead of word embedding, the decoder outputs the probability for each word.
The output of the decoder at time t is used as an input to the LSTM at time $t+1$. The initial input is a special start character. Apart from this, the attention vector from the Bahadanau network is also given to the decoder as input.
All three components can be trained end-to-end, with the encoder taking the input sentence and the decoder outputting the translated sentence. Loss can be Cross Entropy Loss. Due to parallelization on multiple GPUs, google implemented the downpour SGD algorithm for optimization.

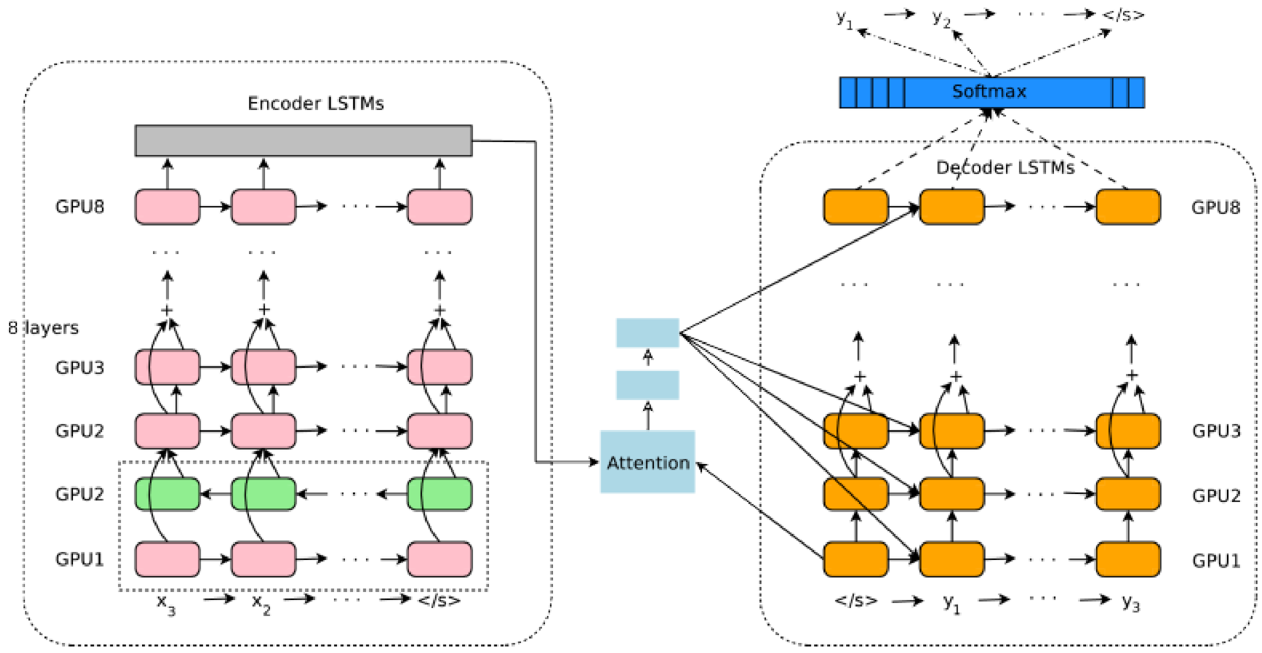


Figure 1: GNMT Architecture

3 Design Choices Made

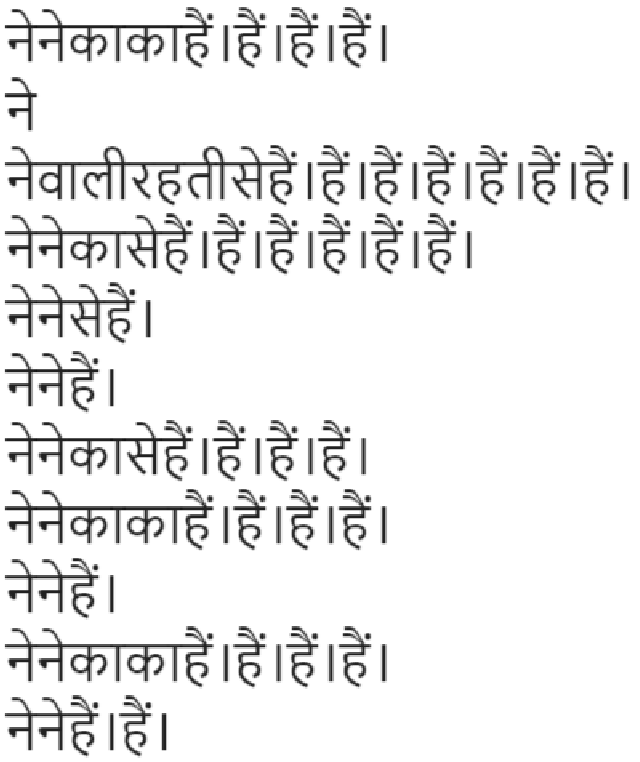
Due to limited GPU access and the unavailability of large amounts of data, we had to make some modifications to the original implementation. First, we reduced the dimension in each layer from 1024 to 128 and made the number of layers from 8 to 4. GNMT has around 598M parameters, but we only compressed the model to train with 21M parameters. Also, multiple GPUs were unavailable, so we had to use a single GPU for training. You can find the code **here**.

4 Dataset Used

We used a publicly available dataset on Kaggle. The dataset contained both English and Hindi sentences. The corpus was compiled from sources like TED talks, news articles, Wikipedia articles, etc. More details can be viewed from here **HindiEnglishCorpora**. Due to limited GPU memory and a large number of parameters, the whole dataset couldn't be used for training; hence we used a small, randomly shuffled subset from the corpus.

5 Observations

We used CE Loss for evaluation.As we suspected,being a very large model with large number of parameters and with scarcity of data and large compute infrastructure,our model is overfitting on the most frequently occuring words as described in below figures.



```
नेनेकाकाहैं।हैं।हैं।हैं।
ने
नेवालीरहतीसेहैं।हैं।हैं।हैं।हैं।हैं।हैं।
नेनेकासेहैं।हैं।हैं।हैं।हैं।हैं।
नेनेसेहैं।
नेनेहैं।
नेनेकासेहैं।हैं।हैं।हैं।
नेनेकाकाहैं।हैं।हैं।हैं।
नेनेहैं।
नेनेकाकाहैं।हैं।हैं।हैं।
नेनेहैं।हैं।
```

Figure 2: A sample output from the code

Plots of Losses with epochs

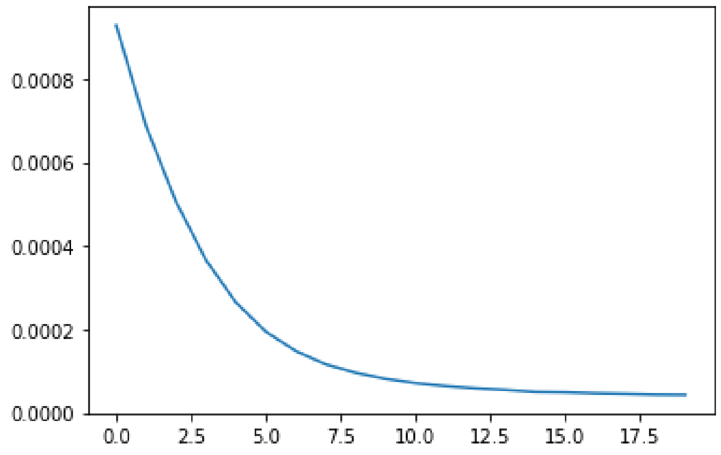


Figure 3: Training Loss

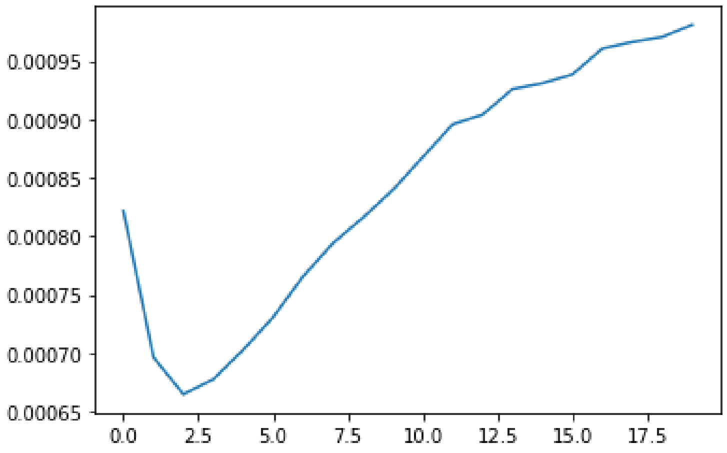


Figure 4: Validation Loss

6 References

1. **Bahadanau Attention**
2. **Google Neural Machine Translation System**
3. **Dataloaders and Datasets**