English to Hindi Translation

Saurabh Shashikant Burewar (B18CSE050)
Rituraj Kulshresth (B18CSE046)
Nisar Ahmed (B18CSE038)

Paper reviews

Paper review of "Very Deep Transformers for Neural Machine Translation".

In this paper, we studied Transformers as the state-of-the-art models in recent machine translation evaluations. Neural machine translation (NMT) models have advanced the previous state-of-the-art by learning mappings between sequences via neural networks and attention mechanisms.

We explore the application of very deep Transformer models for Neural Machine Translation (NMT). Using a simple yet effective initialization technique that stabilizes training, we show that it is feasible to build standards.

Transformer-based models with up to 60 encoder layers and 12 decoder layers. Transformer Network has also become a state-of-the-art model in Neural Machine Translation(NMT) and Multimodal Machine Translation (MMT), two hot topics in the domain of NLP. A deep Transformer based NMT model can outperform Transformer-Big with the proper utilization of the normalization layer.

We also saw a 16-layer Transformer encoder by using an enhanced attention model. In this work, we continue the line of research and go towards a much deeper encoder for Transformer. We choose encoders to study because they have a greater impact on performance than decoders and require less computational cost.

In this paper, we have learnt that the deep Transformer models can be easily optimized by proper use of layer normalization, and have explained the reason behind it. Moreover, we also saw an approach based on a dynamic linear combination of layers and successfully trained a 30-layer Transformer system. It is the deepest encoder used in NMT so far. Experimental results show that our thin-but-deep en-coder can match or surpass the performance of Transformer-Big. Also, its model size is 1.6X smaller. In addition, it requires 3X fewer training epochs and is 10% faster for inference.

This article traced back the origin of NMT to word and sentence embeddings and neural language models. We reviewed the most commonly used building blocks of NMT architectures – recurrence, convolution, and attention – and discussed popular concrete architectures such as RNNsearch, GNMT, ConvS2S, and the Transformer.

Paper review of “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”.

In this paper, we see and study the task of learning universal representations of sentences, i.e., a sentence encoder model that is trained on a large corpus and subsequently transferred to other tasks.

We also learn how Natural Language Inference dataset can consistently outperform unsupervised methods like SkipThought vectors on a wide range of transfer tasks.
Pre-training on natural language inference data, has been proven quite effective for learning dense latent representations of text.

Natural Language Inference is recognized that pretraining embedding models on natural language inference (NLI) data is effective for semantic text similarity tasks . As such, we merge two popular NLI datasets, namely SNLI and MultiNLI into one dataset (referred to as NLI) and perform pre-training with the classification objective function.

Models learned on NLI can perform better than models trained in unsupervised conditions or on other supervised tasks. BiLSTM with max pooling is adopted as a well-performed supervised universal sentence encoder. Max pooling is a common mechanism to get a fixed-size sentence .This pooling operation selects the information from the hidden states of the BiLSTM, for our trained and untrained BiLSTM-max models (for both models, word embeddings are initialized with GloVe vectors). When trained, the model learns to focus on specific words that carry most of the meaning of the sentence without any explicit attention mechanism.

This paper focuses only on the surface of possible combinations of models and tasks for learning generic sentence embeddings. Larger datasets that rely on natural language understanding for sentences could bring sentence embedding quality to the next level.

Paper review of "Neural Machine Translation as a Novel Approach to Machine Translation".

The authors of the paper present the most used machine translation-Statistical Machine Translation system and introduce a novel system- Neural Machine Translation. Neural Machine Translation structure is built on an encoder-decoder framework. The encoder transforms a source language sentence into continuous space representation through a recurrent neural Network.

The advantage of statistical machine translation systems is the removal of manual translators work for each language pair. On the other hand, the disadvantage is the restriction to a single domain of texts, i.e. if the system is trained on one type of corpus (e.g. administrative), then it should be used to translate administrative texts, not e.g. technical texts.

On the other hand, the Neural Machine Translation (NMT) structure is built on an encoder-decoder framework. The encoder transforms a source language sentence into continuous space

representation through a recurrent neural network (RNN) from which the decoder generates a target language sentence using another RNN.

Machine learning could simply be understood by algorithms that process data from which they can learn, and on that basis, they can make decisions or predict solutions to certain problems. Origin of neural networks was inspired by the understanding of the functioning of the human brain, or all connections between neurons. However, in contrast with the human brain, where neurons can freely interconnect, artificial Neural networks consist of discrete layers, connections, and data dissemination. Neural networks use distributed, parallel information processing to perform calculations. Knowledge is stored primarily through the strength of links between individual neurons. The neuron receives signals from the environment from other neurons, processes them and sends them as input signals for the neurons in its surroundings.

Multi layer NN can be divided into -

1. Input layer - the input is from the external world and the output is another neuron,
2. Hidden layer - input is from the external world or from other neurons, the output is another neuron,
3. Output layer - the input is similar to the hidden layer and the output is directed to the external world.

When a fixed sentence representation of a source sentence is created using the encoder and RNN, we use the decoder with RNN to create a translation. Starting from RNN, the internal state of RNN is calculated based on the source sentence summation vector, the preceding word, and the previous internal state. Using the internal hidden state it is possible to score each target word based on how likely it will follow all previously translated words based on the source sentence.Based on the score, the next step is to calculate the probability that serves to select a word by choosing from a multinomial distribution. After selecting the i-th word, it returns to the first step, calculating the hidden state of the decoder, evaluating and normalizing the target word, and selecting the next word. The procedure is repeated until the end of the sentence.

The result is a neural network that can process source segments and transform them into target segments, with NMT passing through whole sentences, not just phrases. The advantage of this approach is precisely the appropriate context of the translation, which also improves the fluency of the translation. But the accuracy of the terminology can sometimes be insufficient.

Our model and Experiments

Dataset

We use a Hindi_English_Truncated_Corpus which provides pairs of english and hindi sentences. For simplicity and better training, we drop sentences that are too small (less than 20 characters) and too big

(more than 200 characters) and put start and end tags for each sentence. Since we cannot work with the full dataset here, we sample 64,000 sentences for our experiment.

Processing the data consists of tokenizing the sentences and creating a vocabulary. Also, since we cannot use strings as inputs in our model, we convert our string of sentences into a sequence of integers. Also, since sequences are of different lengths, we need to pad them. Lastly, we convert them to tensors to form our dataset

We also divide our data into batches of size 64 and shuffle our batches to optimize the dataset.
Now, we are ready with the final dataset to train the model on.

Model

We define a transformer model with an encoder, decoder and a final dense layer. The encoder creates the embeddings and the positional embeddings followed by four encoder layers where each of these layers consists of a multi head attention followed by layer normalization and a feed forward network including our RELU activation function. The decoder is similar to the encoder but also produces attention weights. Each of the multiple decoder layers consist of similar multi head attention followed by layer normalization and a feed forward network. We implemented the standard implementation of a multi-head attention layer by tensorflow. Each head in the multi head attention gets three values, Q (query), K (key), V (value). Instead of one single attention head, Q, K, and V are split into multiple heads because it allows the model to jointly attend to information from different representation subspaces at different positions.

Training and results

We define our loss function using the Sparse Categorical Cross entropy and accuracy function. For our experiments, we train the model on our dataset for 20 epochs where we train while iterating through all our batches and saving a checkpoint to start where we left off.

At each training step,

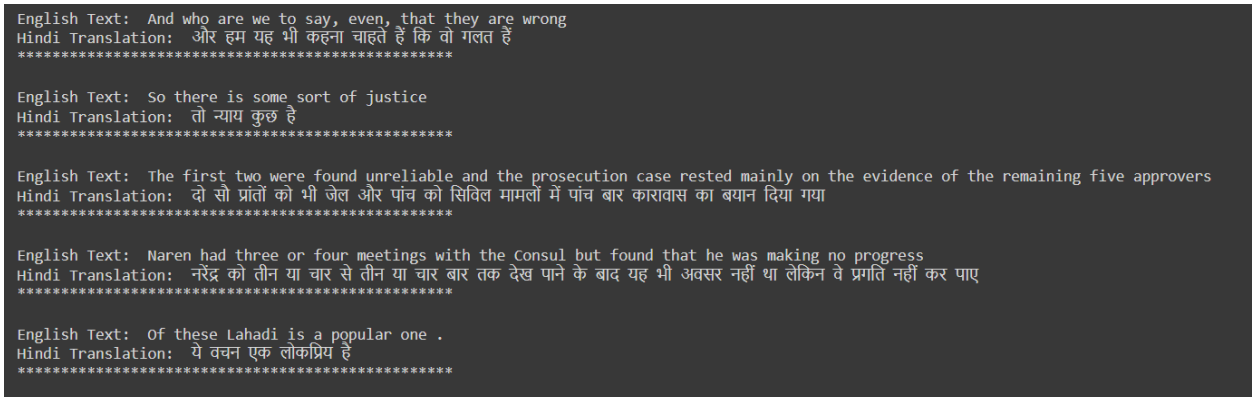
- We create our encoding, decoding and look ahead masks and compute predictions.
- To record operations and compute gradients, we use Gradient Tape from tensorflow which we then apply to our Adam optimizer which has been initialized with a custom schedule learning rate and update our weights.
- Finally, we compute the accuracy and loss using functions we defined.

On average, each epoch takes about 310 seconds on a NVIDIA Tesla K80 provided by Google Colab.

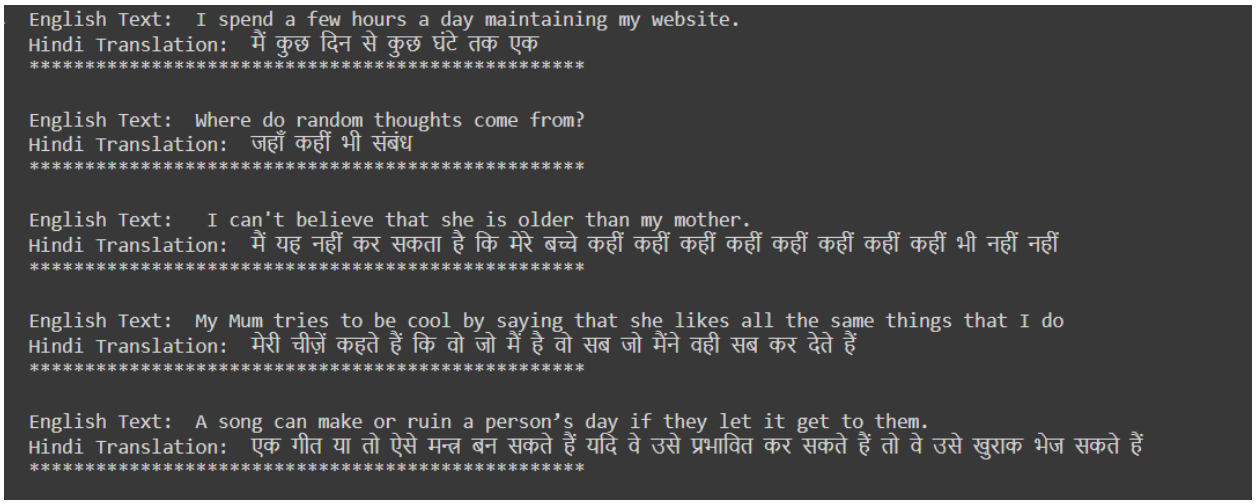
Once training is complete we test our model on some data. Computing predictions is similar to training where we first tokenize our text and convert it to sequence, then create our masks and compute

predictions for every entry in the sequence with ids for correct positions. The predicted sequence is concatenated and then converted back into text.

Some good results given by the model -



Some bad results given by the model -



Contribution

B18CSE038 - 1 Paper review and bit of code
B18CSE046 - 1 Paper review and bit of code
B18CSE050 - 1 Paper review and bit of code

All members did coding together and code is a bit referenced, so contribution in code is hard to say.

Code

https://colab.research.google.com/drive/1K5vIfxeECq_YHI6sA1Viu_3B3PPZF3uY?usp=sharing

References

- https://www.tensorflow.org/text/tutorials/transformer
- https://arxiv.org/abs/2008.07772
- https://arxiv.org/abs/1705.02364
- https://www.researchgate.net/publication/344476131_Neural_Machine_Translation_as_a_Novel_Approach_to_Machine_Translation
- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/