Neural Machine Translation with Transformers

Sapan Sharma
Department of Computer Science
Brock University
ss17hq@brocku.ca

Abstract

Transformer architectures have taken the neural network world by surprise as it continues to beat benchmark in new applications and is a major improvement on previous recurrent architectures. This paper explores the Transformer architecture by creating an application of Neural Machine Translation and testing it according to the "Attention is all you need paper."

1 Introduction

Neural machine translation is one of the most challenging artificial intelligence tasks given the fluidity of human language. Many different techniques were used to tackle this task such as classical methods which are rule based methods but recently with the development of deep neural networks, we are able to achieve state of the art results in neural machine translation. Many examples can be found in our day-to-day life which uses Neural Machine Translation, such as Google translate which is leading its development and use cases. This state-of-the-art algorithm is an application of deep learning in which massive data-sets of translated sentences are used to train the model, allowing it to be capable of translating between any two languages with high accuracy while preserving context.

Transformer architectures have allowed major improvements on Natural language processing tasks and their success is strongly noticed in many machine learning benchmarks. Using only attention mechanism instead of Recurrent based architectures have reduced many bottlenecks, provided parallelization potential and it also avoids catastrophic forgetting. Transformer was introduced in paper “Attention is all you need” by google brain team in 2017. Since then most famous models that are emerging in the field of Natural Language Processing and other sequence to sequence based problems consists of many transformers or its variants combined in different ways. In the “Attention is all you need” paper, the authors used a sequence-to-sequence network for neural machine translation that transforms a given sequence of elements such as sequence of the words in a sentence into another sentence. The transformer model, which is entirely based on positional encoding and attention, replaced the recurrent layers. It is commonly used in encoder to decoder architectures with multi-headed self-attention layers.

Previously in sequence-to-sequence problems Recurrent NN’s were used with Gated Recurrent Units’s or LSTM’s to retain context and timely dependencies in sequences but they have their limitations such as encoding bottlenecks, memory not long enough, and they are slow with no parallelization potential etc. Traditional RNN’s are not very effective in handling very long temporal dependencies which appear in large sequences. Before 2017, using LSTM’s with attention for translation was standard but its limitation implored researchers to find better methods. These limitations were dealt with, by introducing positional encoding and multi-headed attention mechanism. This model was proven to be highly effective for common natural language processing task which involves sequences. Transformer neural Networks have largely replaced LSTM networks for vector to sequence problems like sentiment analysis, sequence to sequence problems like neural machine translation, speech recognition, and vector to sequence problems like image captioning.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

2 Method

Method used to create Neural Machine Translation in this paper is driven from "Attention is all you need" paper, modifications are made to meet the current compute requirements and assessment requirements. Input and output embedding is required to understand a word by the network for both the encoder and decoder components. It is done by tokenizing the word and using learned embeddings from a pre-trained model to convert those tokens to vectors of dimension, which will be useful in the following layers. Learned embedding maps words to a point in space called embedding space, where similar words in meaning are closer to each other. A pre-trained embedding space is commonly used to save time. Bert pre-trained language representation is used to feed vectors to the transformer model. It enhances the language understanding, by retaining context unlike word2vec or GloVe representation where each word only has one vector without context. Bert gives bidirectional context using encoder based on other words in the sequence giving more exploitation potential to the model. Bert wordpiece vocabulary of tensorflow is used, it uses Bert's splitting algorithm to split the text into words before generating the subword vocabulary on the pre-trained bert by Tensorflow team. Example vocabulary for hello: he, hell, ello, etc.

Then in the next layer, positional encoding is given, the same word can be used in different sentences with different meanings; hence a need of some information about relative position of tokens in the sequence and provide context of sequence based on that relative positioning. That information can be added using positional encoding to each word in a sequence. Like input embedding, learned positional encoding can be used. The paper being referenced used the sine, and cosine functions. In the experiment, positional embedding was added to the input word embedding of the same dimension to get a vector containing both information.

The Encoder Component uses the input positional embedding and generates a continuous vector representation of inputs with all the information learned from the input sequence. The Encoder is made up of stacked identical N layers, and in the experiment, 4 of them were used. Each layer has two sub-layers: A Multi-head self-attention layer, and a position-wise fully connected feed-forward network. The decoder then takes the representation vector from the encoder and uses it with the information learned from the target language to predict the word step by step. Decoder is also made up of stacked identical N layers equal to encoder. Each layer has one masked multi-head attention layer, one multi-head attention layer operating with output of the encoder stack, and one feed forward layer. A position-wise feed forward layer is used where all parameters are shared across the input making it the same Feed Forward layer to improve training speed.

Layer normalization is applied after each sub-layer which normalizes the join of residual connection from input and output of the sub-layer. Function of layer normalization is to generate an output with the same dimension as initial input to the sub-layer. Residual connection also prevents the vanishing gradient problem.

The attention layer in general sense gives the relevancy context of each word to every other word in the input sequence. Every word in an index position of the input sequence is evaluated against the rest of the input sequence. The higher the relevancy of a word to another, the higher its attention vector, which is trained over timesteps. Improved attention is used in multi-head attention, Scaled Dot-product attention, where three weight matrices are used to compute three linear transformations for each word embedding in the sequence to get a query, key and value for each vector. Same vector is used because of self-attention model which help the word to associate with other words. These queries and key, value pairs will be used as a retrieval system to calculate attention vector scores by checking the compatibility of query with the key, which is the dot product of query and key. The score matrix from matrix multiplication operator determines how much attention one word should put on another. Then the scores are scaled down by dividing with dimensions of keys and softmax function is applied to get probabilities on which word to attend to. Softmax will act as mask when multiplied by value vector to get an output of words which needs more attention. To make this computation a multi-headed attention, the query, keys and values are split into h vectors before applying scaled dot-product attention, where h is number of heads. Output from each head is then concatenated before going through the final linear layer. Multi-heads are used to learn something different, giving models more representation.

Positional output embedding of target language is passed to masked multi-head attention layer which prevents next word in sequence to get attention as that needs to be predicted during training. Then

in the next attention block which is encoder decoder attention block which determines how related words from each language are and generates similar attention vector for every source and target vector. Then each attention vector is passed to a feed forward layer to make sense of our attention vectors. The final linear layer is another feed forward layer which is used to expand the dimensions into number of words in the target language and the next word is predicted to complete the sequence over multiple time steps. This completes the walk-through of the entire method.

3 Experiment

The architecture was created using Tensorflow library and a nightly build to get updated library. Dataset was imported from Google's tensorflow datasets library. The data-set have 51785 training, 1193 validation, and 1803 test examples containing Portuguese and English translations. Batches of 64 were created to train and test on the data-set. A pre-trained Bert language representation was used to feed vectors to the transformers with a vocabulary size of 5000.

The transformer architecture had 2 stacked layer of each decoder and encoder, 2 heads for attention, the embedding dimensions were 64, which is input dimensions and remain same in the subsequent layers and the units for inner feed forward layer is 256. An Adam optimizer is used with a variable learning rate as specified in the "Attention is all you need paper". A residual dropout layer was used to the output of each sub layer with a rate of 0.1

BLEU scores were for the test set using 300 examples comparing the predicted translation with the original translation. Run with 2 epochs produced a result of four different weights(1,0.5,0.3,0.25) as BLEU-1 results: 0.127 BLEU-2 results: 0.356 BLEU-3 results: 0.539 BLEU-4 results: 0.597

Comparisons using manual sentence translation was also done but due to very less accuracy number which was because of limited computation had sub-par results. An accuracy of 30 percent was achieved in 2 epochs but given more compute power it can easily achieve great results as shown by authors of the paper.

Referenced paper achieved BLEU scores better than any previous state-of-the-art models on English-to-german and english to french tests at fraction of training cost compared to other due to parallel nature of transformer architecture. It outperformed by more than 2.0 BLEU and reached a new 28.4 score, with a configuration of big transformer, 6 stacks, 16 heads, 1024 embedding dimension, 0.3 dropout layer using 300,000 steps completed in 4 days on a parallel P100 GPUs of 8.

4 Conclusion

In conclusion, the transformer model introduced a more effective approach in dealing with Sequence-to-Sequence problems. Previous solutions in applications like Neural Machine Translation, used recurrent networks in their encoder-decoder architecture. Although, the experiment was not used with very high power compute, the results were still impressive as can be seen from the experiment of Google Brain team. The flaw of losing information from the long sequence caused significant hurdles. As the transformer model is great with handling long sequences of input, many problems and applications involving large input sequences would be applicable to this model. Though language translation problems were the main application of the transformer model in this paper and papers referenced, it is worth noting that other applications of the Transformer Model would be feasible. This means not only text-based tasks, but tasks involving the requirement of handling large input sequences like image, sound, or video would also be applicable. There is still lots of room for exploration in Transformer Model applications. The transformer model's unprecedented success in text-based problems makes it an enticing topic for further exploration of potential applications.

Transformers are not perfect and have their flaws as well, the time complexity and scalability grows poorly with the length of the input sequence. Future directions and experiments will be done on Performer network which is more enhanced way of utilizing Transformer architecture with less compute allowing more potential to explore applications of this novel method.

References

[1] Allard, M. (2019, January 4). What is a Transformer? Retrieved from https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. (2017). Attention Is All You Need.

[3] Muñoz, E. (2020, November 2). Attention is all you need: Discovering the Transformer paper. Retrieved from https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634

[4] Phi, M. (2020, April 30). Illustrated Guide to Transformers- Step by Step Explanation. Retrieved from https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634

[5] Transformer model for language understanding : TensorFlow Core. (Documentation) (n.d.). Retrieved May 09, 2021, from https://www.tensorflow.org/tutorials/text/transformersetup$_input_pipeline$