Review of Attention, Transformer, and BERT

Natural language processing (NLP) has been a hot topic in academic and industry for some years. One of the biggest challenges in NLP is the lack of enough training data. Even though many texts are generated from everywhere globally, the data in specific datasets with annotated labels are far from enough for NLP models to run accurately. The basic idea for solving NLP problems is pre-training the unannotated text on the web and then fine-tuning the pre-trained models on task-specific datasets. The state-of-art technique for pre-training is known as BERT (bidirectional encoder representations from transformers). When we read articles related to BERT, we often encounter terms such as attention and transformer. In fact, we already found the term transformer in the very name of BERT. In this essay, I am going to have some general reviews of those techniques.

The transformer is a breakthrough improvement over the recurrent neural network (RNN), which had been dominantly used in the NLP world for years. Recurrent models typically factor computation along with the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states ht, as a function of the previous hidden state ht−1 and the input for position t. This approach has faced some critical challenges, such as inefficiency in computing and losing long range dependencies. When dealing with lengthy data, its sequential modeling would cause too many training steps, and its recurrence would prevent parallel computation. The attention mechanism had been integrated into RNNs to overcome those challenges. Then we have transformer model architecture, which is completely based on the attention mechanism. This approach soon became the state-of-art technique as it could dramatically increase efficiency and accuracy.

It is time to have a review of the attention mechanism. Attention is basically kind of like a retrieval based on the weighted combination of similarity. Say we have keys, such as k1, k2, k3, k4, values, such as v1, v2, v3, v4, and query q. Firstly, similar to we have done in text retrieval problems, we can vectorize the terms and query. Then the first layer is the similarity layer in which q would influence every term. The similarity function could be some function with respect to query and each term to get a scaler value si. There could be many options for this similarity function. For instance, the simplest one could be the dot product (si=dot(ki, q)) or cosine (si=cos(ki, q)). The second layer is the layer to compute weights. The data used in this layer come directly from the first layer, and again each term would associate with one weight. The weight function is a SoftMax function, which is a normalized exponential function (wi=exp(si)/exp(sk)). Finally, we can get the attention value by a linear combination of each vi and wi. The steps mentioned above from a general scheme of scaled dot-product attention. As we can see here, the attention mechanism does not have a recurrence and draws connections of any part of the sequence.

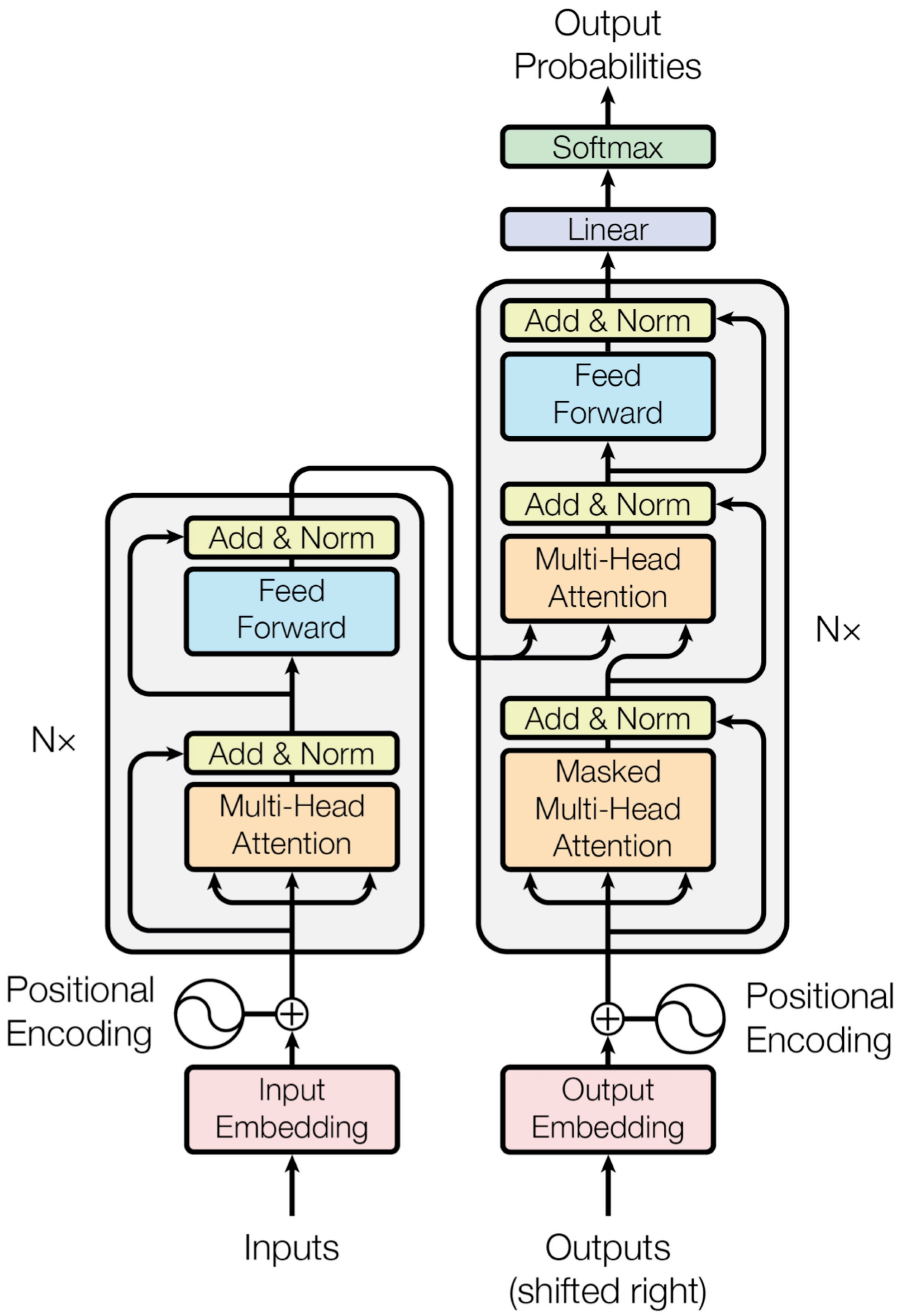
The transformer network is generally built up from many attention blocks. In 2017 Vaswani et al. published their paper "Attention Is All You Need." Since then, the transformer network has replaced the recurrence network as the most advanced technologies in the NLP world. The name of the paper has already shown that attention is all we need for the transformer network. The diagram of the whole transformer model architecture is shown at the end of this article. The transformer network consists of two parts. The first part corresponds to an encoder and the second part corresponds to a decoder. The encoder would process an entire sequence of terms in parallel because only some attention blocks in the encoder part and the attention blocks don't require recurrence. The encoder part also consists of two parts. The first part is the multi-head attention part in which all the essential computation was carried out. After the entire sequence of words are embedded (we can think of this as a technique of vectorizing terms) and are feed into the encoder, the multi-head attention layers will compute the attention between every position and every other position. The scheme is simple, the word in each position can be treated as a query, and the words in other positions can be treated as keys and values. Since there is no recurrence, the computation in multi-head attention can be carried out parallelly. For example, different words combinations can be computed in the different core of GPUs, and then the groups of results can be processed in parallel again. The add and norm sublayers in the figure are basically adding the residual connection from the lower layer input to the output of multi-head attention and then normalize the combination. Then the output of the attention layer was fed into the feed-forward layer (a feed-forward layer is basically a layer that does not have a feedback loop) that will generate a sequence of embeddings. Each embedding corresponds to a position. Each embedding in the output reflects the position and reflects the relative relations between that position and the other positions. The whole process would be repeated, and the number of repetitions is specified by N. The embeddings generated from the encoder part will go to the decoder part. Then again, the attention mechanism plays a central role in the decoder part. Since the decoder part would generate some text output, some extra layers, such as linear and SoftMax, are on the top. Inside the repetition block of the decoder, there is one lower part called masked multi-head attention. This layer is to do a self-attention between output words as there is no input embedding going to this layer. Since we are dealing with the sequence of output words, we need to only compute one output-word's attention based on the terms before it. The ordering is import in this layer. Then the upper layers are similar to their counterparts in the encoder part. The repetition in N times in the decoder part will gradually build up combinations and better embeddings to feed into the top layers to generate some probabilities. The whole transformer model scheme shows that there are only a few layers and no recurrence involved in the computation, and attention is everything we need in the transformer.

BERT is based on transformer architecture. Since the positions and relative relations between different positions are embedded into the transformer, BERT could easily process the sequence of words non-directionally. Not too long ago, processing a sequence of words needs to be done either from left to right or from right to left. Even the combined left-to-right and right-to-left models are just sequentially moving in one direction and then moving in the opposite direction. These approaches will soon lose the context information when the size of the sequence is getting large. The biggest advantage of BERT is that BERT can process the sequence without direction. The word in each position and the words in other positions are weighted simultaneously. In BERT, training data makes use of two strategies. The first one is called masked LM (MLM). The idea is that we randomly mask out 15% of the words in the input and run the entire sequence through the BERT attention-based encoder and then predict only the masked words. The simple masking scheme here may cause some mismatching between pre-training and fine-tuning. There are some tricks proposed by the researchers to deal with this issue. The second strategy is called next sentence prediction. The idea is also simple. We basically trained data like question-and-answer. During the training, we make 50% of the time the second sentence comes after the first sentence, and the other 50% of the time, the second sentence is just some random sentence. BERT is then required to predict whether the second sentence is the answer or not, assuming that the random sentence will be disconnected from the first one. By combing these two strategies, BERT has done a very great job in NLP subtasks. In sarcasm detection, one paper reaches 0.91 AUC by using BERT based trained sarcastic dataset.

The transformer-based BERT model has been one of the best models and the focus in the NLP world. It is good for us to understand the ideas behind the BERT model. This paper has gone through the general scheme of attention, transformer, and BERT. Some detailed tricks and standard mathematic forms are left out in this review. It is undoubtedly true that more readings and study are needed to go deep into each topic.

References:

1. Attention Is All You Need, Ashish Vaswani et al. https://arxiv.org/pdf/1706.03762.pdf
2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob Devlin et al. https://arxiv.org/pdf/1810.04805.pdf
3. A Transformer-based approach to Irony and Sarcasm detection, Rolandos Alexandros Potamias et al. https://arxiv.org/pdf/1911.10401.pdf


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)