Progress Report: Phase II Transformer Model

Author: Mingyu (Jerry) Liu

September 3, 2020 (draft)

Part I Introduction

The Transformer model is drastically changing the natural language processing world. It was first introduced in the paper "Attention is All you Need" (A Vaswani et al.) in mid-2017 and since then has been breaking multiple records NLP records and rapidly replacing the once-popular RNN sequence model and its variations by storm. It is behind many real-world applications that you might be using directly or indirectly every day. For instance, Google's search engine and language translation app rely on its pre-trained state-of-the-art BERT (Bidirectional Encoder Representations from Transformers) language model. The GPT language model series developed by Open AI, that are capable of generating highly-authentic human-like text, are widely applied to online conversational chatbots.

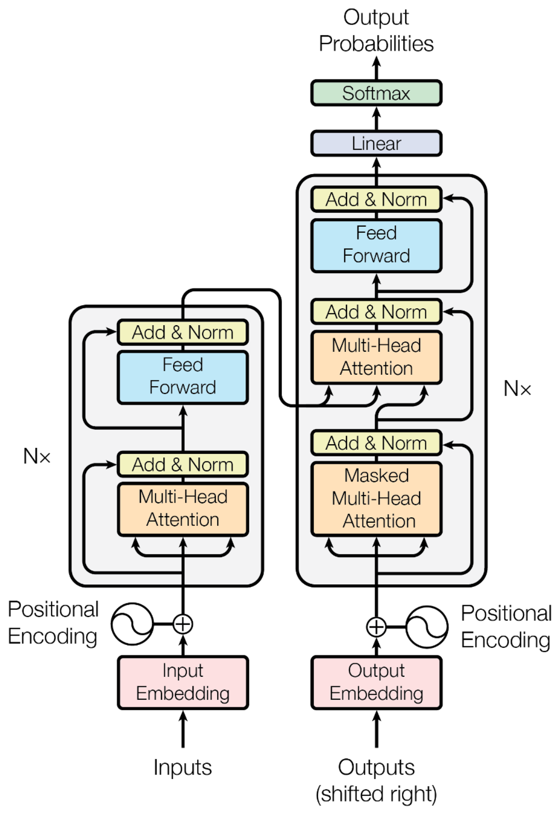
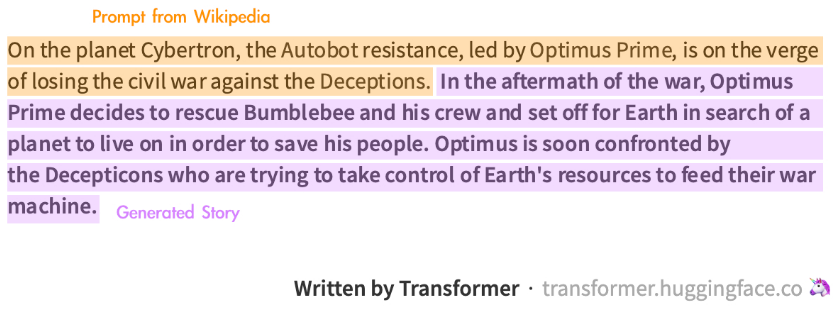


Figure 1: The Transformer architecture (source)

On a high level, the Transformer model is an attention-based encoder-decoder type of neural network. It leverages the power of attention mechanism and has infinite memory access. The encoder maps an input sequence into an abstract continuous representation that holds all the learned information from the input. The decoder then takes the continuous representation and step-by-step generates a single output while also being fed the previous output. The trained encoder-decoder can be used together for text generation, while the encoder alone can produce a latent representation of the input sequence which can subsequently be applied in various tasks including sentiment-classification and text-to-image synthesis.

Attention Mechanism

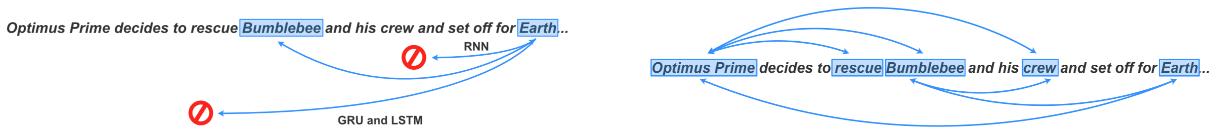
To understand Transformers, we must first understand the attention mechanism. To give an intuitive explanation of the attention mechanism, let's look at a short story generated by GPT with a prompt from Wikipedia (Figure 2). As the model generates text word by word, it has the ability to reference (or attend to) previous words that are relevant to the generated word. Deciding which words to attend to is all learned while training with backpropagation.



On the planet Cybertron, the Autobot resistance, led by Optimus Prime, is on the verge of losing the civil war against the Deceptions. **In the aftermath of the war, Optimus Prime decides to rescue Bumblebee and his crew and set off for Earth in search of a planet to live on in order to save his people. Optimus is soon confronted by the Decepticons who are trying to take control of Earth's resources to feed their war machine.**

Figure 2: A short piece of story generated with GPT-2 (source)

RNNs also have the capability of looking at previous inputs but have a very short window to reference from before suffering from the vanishing gradient problem. As the story gets longer, RNNs can't access word generated earlier in the sequence. The same is also true for the more advanced GRUs and LSTMs, although they are designed with the capacity to have a longer memory to reference from. The attention model, in theory and given enough computational resources, can reference from an infinite memory window and is capable of using the entire context of the story while generating text (Figure 3).


Optimus Prime decides to rescue Bumblebee and his crew and set off for Earth...
RNN
GRU and LSTM
Optimus Prime decides to rescue Bumblebee and his crew and set off for Earth...

(a) RNN, GRU and LSTM have limited memory (b) Attention has infinite memory window

Figure 3: Memory Capacity of RNNs vs. Attention

Part II Embedding

Input Embedding

Machine learning models take vectors as input. Therefore, when working with text, the first thing to do is to come up with a strategy to convert strings to numbers before it can be fed into the model. The naive approach is to "one-hot" encode each word in the sequence with a dictionary. However, this strategy suffers a number of weaknesses: a) One-hot encoding is arbitrary. It treats each word as an isolated entity and does not capture any connection between words with similar meaning (e.g. "orange" and "apple" is no closer than "orange" and "dog"), therefore limiting the generalization power of learned models. b) This representation is sparse and inefficient. The dimension of one-hot vectors grow linearly with the dictionary size, making training both time and memory consuming, especially when the dictionary size is large.

The technique called *word embedding* gives a way to use an efficient, compact representation in which similar words have a similar encoding. The embeddings do not need to be specified by hand. Instead, they are trainable parameters that can be learned during training. It should be noted that the input embedding technique is not confined to the Transformer model, it is a general approach of processing text data in NLP for better performance. (For curious readers, this blog provides a good overview of the different types of embedding algorithms, including the most popular Word2Vec and GloVe techniques).

Positional Encoding

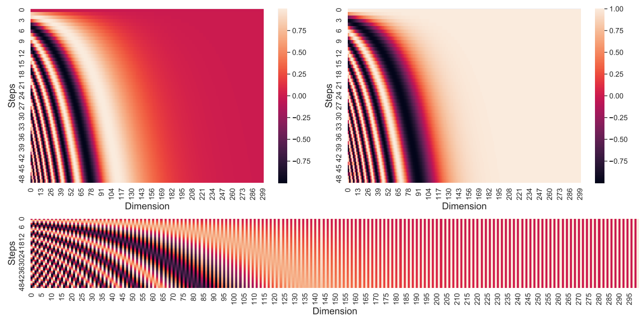
The Transformer has no recurrence like the RNNs. Instead, all tokens in a sequence simultaneously flow through the Transformer's encoder/decoder stack. Then, positional information of the sequence must be injected explicitly in order for the model to make use of the order of the sequence. This is done by adding *positional encodings* $\overrightarrow{p_t} \in \mathbb{R}^{d_{\text{emb}}}$ with compatible shape d_{emb} into the input embeddings. The encoding proposed by the paper employs an ingenious technique with alternating sine and cosine functions of different frequencies

$$\overrightarrow{p}_{(t,2i+1)} = \cos\left(\frac{t}{10000^{2i/d_{\text{emb}}}}\right) \qquad \overrightarrow{p}_{(t,2i)} = \sin\left(\frac{t}{10000^{2i/d_{\text{emb}}}}\right)$$

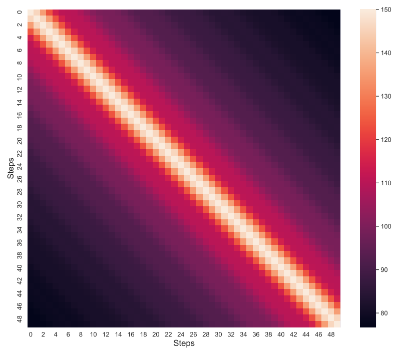
where t is the position in the input sequence and i is the position along the embedding dimension. That is, each time-step of the positional encoding corresponds to a sinusoid whose wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. The positional encoding can be written explicitly in the following vector form:

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_0 \cdot t) \\ \cos(\omega_0 \cdot t) \\ \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \vdots \\ \sin\left(\omega_{d_{\text{emb}}/2} \cdot t\right) \\ \cos\left(\omega_{d_{\text{emb}}/2} \cdot t\right) \end{bmatrix}_{d_{\text{emb}} \times 1} \qquad \text{where} \quad \omega_i = \frac{1}{10000^{2i/d_{\text{emb}}}}$$

A visualization of the sine-, cosine- and the interweaved-encoding is plotted in Fig 4a, where the horizontal axis represents the time-step and the vertical axis represents the embedding dimension. A useful property of this encoding is that the dot-products between neighbouring time-steps are symmetrical and decay nicely with time Fig 4b, which provides the attention mechanism meaningful information about relative distance between words during dot-product attention (as will be discussed soon).



(a) Sinusoidal positional encodings (top left: sin-only, top right: cosine-only, bottom: interweaved)



(b) Dot product of positional embeddings across all time-steps

Figure 4: Characteristics of sinusoidal positional encoding

Another key characteristic of this sinusoidal positional encoding is that the encoding $\overrightarrow{p}_{t+\phi}$ can be represented as a linear function of $\overrightarrow{p}_t$ for any fixed offset ϕ, thus enabling the self-attnetion module to attend effortlessly to relative positions. For a detailed proof, we direct the reader's attention to this great article. However, a less rigorous derivation is presented here to give a deeper understanding. Consider a sine-cosine pair with frequency ω_i, we shall prove that there exists a t-independent linear transformation $M \in \mathbb{R}^{2\times2}$ such that the following equation holds

$$M \cdot \begin{bmatrix} \sin(\omega_i \cdot t) \\ \cos(\omega_i \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_i \cdot (t+\phi)) \\ \cos(\omega_i \cdot (t+\phi)) \end{bmatrix}$$

by matrix inversion, the transformation matrix M is found to be

$$M_{i,\phi} = \begin{bmatrix} \cos(\omega_i \cdot \phi) & \sin(\omega_i \cdot \phi) \\ -\sin(\omega_i \cdot \phi) & \cos(\omega_i \cdot \phi) \end{bmatrix} \tag{1}$$

which takes a form analogues to the rotation matrix and is independent of t as desired. The sub-matrix $M_{i,\phi}$ can be found for a sine-cosine pair with any frequency ω_i, which eventually allows us to represent the entire positional encoding at $t+\phi$ as a linear function of the encoding at t for arbitrary offset ϕ. This property makes it easy for the model to learn to attend by relative positions. (source)

Part III Encoder

The encoder consists of two sub-layers a) a multi-head self-attention mechanism and b) a simple fully connected feed-forward network. The *multi-head attention* module computes attention

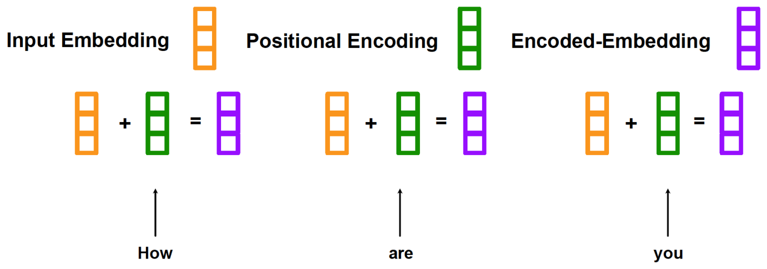


Figure 5: The input embedding & positional encoding illustrated

weights for the input and produces an output vector with encoded information on how each word should attend to other words in the sequence. A residual connection is employed around each of the two sub-layers followed by layer normalization. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. Several encoder blocks can be stacked on top of each other with each block potentially learning a different attention representation (weights not shared between stacked blocks).

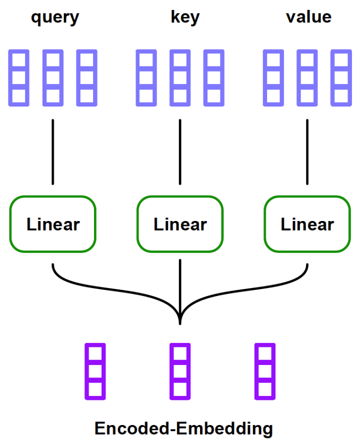


Figure 6: Creation of query, key and value vectors

Multi-headed Attention

Self-attention: The multi-headed attention in the encoder applies a specific attention mechanism called *self-attention*. Self-attention allows the model to associate each individual word to the rest of the sentence by an *attention weight*. (e.g. in the sentence "Hi how are you", the can learn to associate "you" with "how" and "are", and be aware that the sentence is a question and can answer appropriately.) To achieve self-attention, we feed the input to three distinct fully connected layers to create the **query**, **key** and **value** vectors (Fig 6). The query and key undergo a dot product matrix multiplication to produce a score matrix. [1]The score matrix determines how much focus should a word be put on other words. Then the score gets scaled down by dividing the square root of the dimension d_k of the queries and the keys and converted to softmax probability to get the attention weight. The matrix is normalized using $\sqrt{d_k}$ to prevent the dot products from growing large in magnitude, which could push the softmax function into regions where it has extremely small gradients. In the end, each value vector is multiplied by the attention weight matrix to get the output vector. The higher weights will keep the value of the words and the lower weights will drop out irrelevant words (Fig 7).

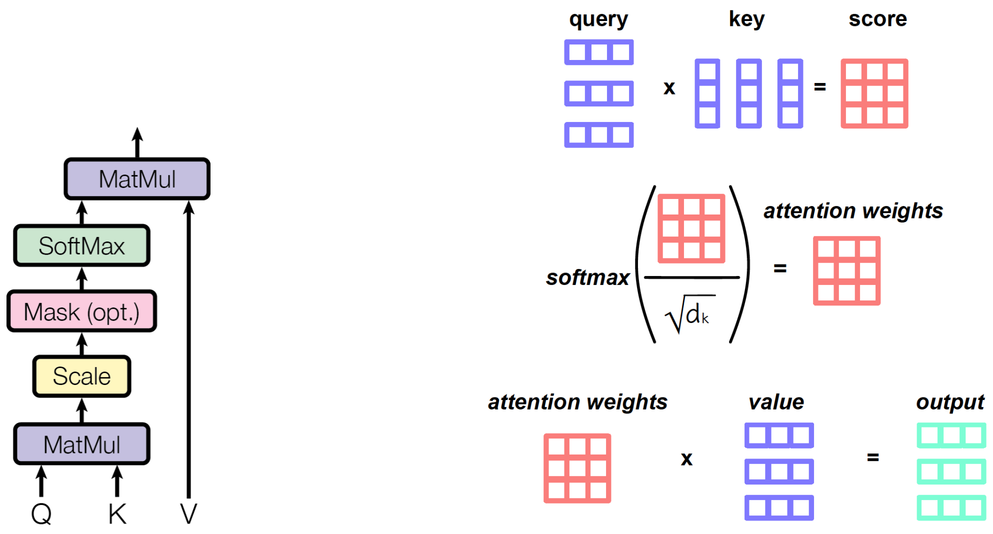


Figure 7: Self-attention module illustrated (left source)

Multi-head: To make the self-attention a multi-headed computation, we create a set of distinct query/key/value vectors, each initialized with unique weights and goes through its own self-attention module (called a *head*). Each head produces output a vector that gets concatenated before going through a final linear layer for further processing. In theory, this gives the attention multiple *representation subspaces*, where each head could potentially learn a different attention weight, therefore giving the encoder model more representing power.

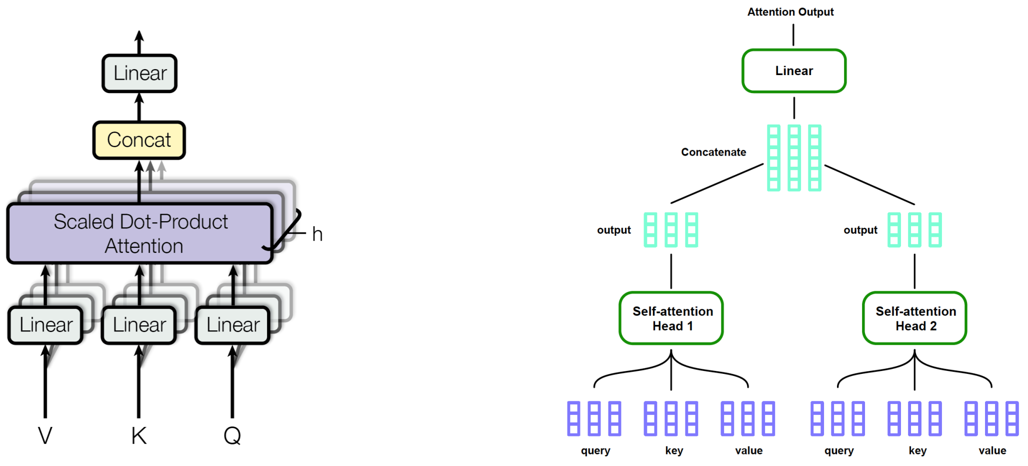
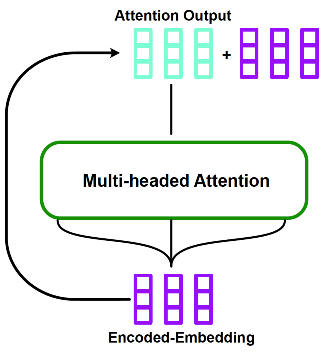


Figure 8: Multi-headed attention block illustrated (left source)

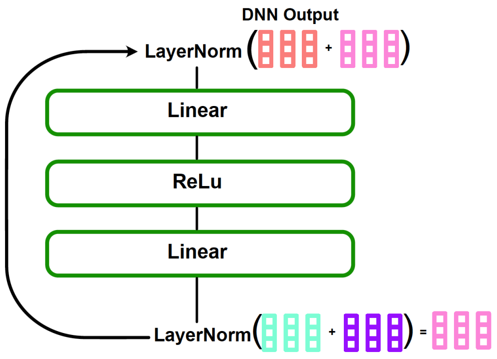
[1] The query/key/value concepts come from retrieval systems. For example, when you type a query to search for some video on Youtube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best-matched videos (**values**) (source)

Residual Connection

With the residual connection, the multi-headed attention vector is added to the original input. The residual connection allows gradients to flow through the network directly without passing through non-linear activation functions, thus makes the training of deep networks more easily and less prone to vanishing & exploding gradients. In the context of the transformer model, the residual connection allows the positional embeddings from the input to efficiently propagate to further layers where the more complex interactions are handled. The output of the residual connection then goes through a layer normalization, which helps stabilize the hidden state dynamics of the Transformer and speed up the training process.



(a) Residual connection



(b) Simple feed-forward neural network

Figure 9: Residual connection and point-wise feed-forward neural network visualized

Simple Feed-forward

The output from the layer normalization is fed into a point-wise feed-forward network for further processing. The output is again added to the input of the feed-forward network and further normalized.

In summary, the encoder block serves for the purpose of encoding the input into an abstract continuous representation with attention information. This will help the decoder focus on the appropriate words in the input during the decoding process. The encoder block can be stacked N-times to further encode the information potentially having each block learning more complex attention representation, therefore boosting the predictive power of the Transformer network.

Part IV Decoder

In addition to the two sub-layers in the encoder, the decoder inserts a "encoder-decoder attention" layer, which takes the input sequence as well as the encoder output that contains the attention information and helps the decoder focus on appropriate places in the input sequence. The self-attention module in the decoder operates in a slightly different way from the one in the encoder. Since the decoder is auto-regressive and generates sequence word-by-word, all future tokens in the decoder should be masked, ensuring that the predictions for position t can depend only on the known outputs at positions less than t. This is done by masking the section of the score matrix above the diagonal with $-\inf$. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization.

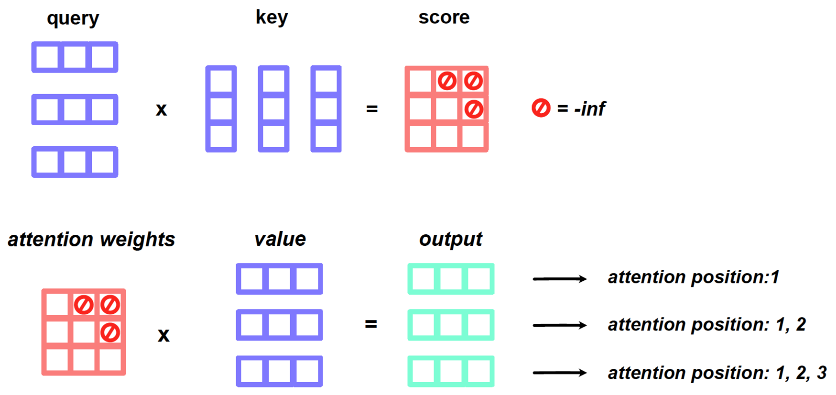


Figure 10: Masked score matrix in the decoder self-attention module

Part V Discussion

Comparision with RNNs

Transformers see the entire sentence as a whole whereas, in general, RNNs processes sequences and perform backpropagation sequentially, as calculating weights at a time-step t depends on the hidden state at the previous time-step $t-1$. The Transformer model completely discards the recurrence and processes the entire sequence simultaneously, without the need for backpropagation through time. This property can significantly reduce computation time by taking advantage of the parallel processing power of modern GPUs.

In addition, the time complexity of training a LSTM for one epoch is $\mathcal{O}\left(\text{seq\_len} * \text{feature\_dim}^2\right)$. The time complexity of a Transformer model, on the other hand, equals $\mathcal{O}\left(\text{seq\_len}^2 * \text{feature\_dim}\right)$. Therefore, a asymptotical reduction in computational cost can be achieved if seq_len $>$ feature_dim, which is most often the case with sentence representations used by state-of-the-art models in machine translations. This is extremely advantagous when we increase the hidden dimension, potentially to learn more complicated representation, without having to worry that the computational cost blows up.

As a side benefit, self-attention could yield more interpretable models. We could inspect attention distributions in the attention heads, which according to the author "(each head) clearly learn to perform different tasks, many appear to exhibit behaviour related to the syntactic and semantic structure of the sentences."

The edge brought by the Transformer model, however, does not come for free. There are two noticeable drawbacks compared to RNNs: a) Transformer can only process fixed-length sentences - where length is a hyperparameter b) Training will become expensive as the sequence length seq_len becomes long. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighbourhood in the input sequence centred around the respective output position.

Summary of self-attention in Transformer (adapted from the paper)

The Transformer uses multi-head attention in three different ways:

- In “encoder-decoder attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.
- The encoder contains self-attention layers. In a self-attention layer, all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property.