Attention is all you need - Fully grasping the legendary paper

Posted **May 6, 2024** • Updated **May 7, 2024**
By **notiona** **16 min** read

A review that is 8 years late

As an AI engineer practicing data analysis and machine learning in the industry, I always had a thirst for one of the most cutting-edge technology leading the AI today: the transformer. Yet all I knew about it was high-level ideas, which I usually forgot in a couple of days. I had little hands-on experience in using deep learning professionally, and there were more prerequisite knowledge than I thought to fully understand the concept and motivation for the transformer.

Over the last few weeks, I have been studying prerequisite concepts such as deep learning fundamentals, pytorch, NLP fundamentals, RNN, LSTM, seq2seq, and attention. Also, I have looked at several posts and videos on transformers to finally read and analyze the legendary paper, “Attention is all you need(2017)”, the first paper that brought transformers to the world.

As I was reading the original paper, I was analyzing, visualizing, and implementing some parts of the paper for a clearer understanding, and planned to post it as this month’s blog post. Then I soon faced a problem. My review was around 8 years late 😅

As of 2024, the concept of transformers is almost common sense to deep learning researchers, and there are thousands of reviews for this particular paper. As I will introduce in the next section, some of these are beyond imagination in their quality and straightforwardness. I found no point in adding another petty review of mine among these world-class analyses.

Instead, this post aims to depict two types of meaningful information regarding the monumental paper.

First of all, there is a roadmap for those who want to start learning transformers and Gen AI. I was struggling to find the appropriate (and preferably free) material as I was learning, so I have backtracked and hand-picked the material that I have gone through to finally read and understand the original paper. I wish it could help those with a similar interest.

Secondly, there is a FAQ section illustrating the questions and answers I had when I read the paper. I hope this will help those who have already read the paper and wish to gain a deeper understanding on transformers.

Roadmap to understanding the transformer

The next section of this blog post, “Attention is all you need - FAQ” requires more prerequisite knowledge than any other blog posts so far, as you need to understand the original transformer manuscript. More specifically, you would need all the relevant knowledge in order to even follow the gist of the paper let alone understand the core ideas.

I cannot say I have understood the paper 100%, but I have backtracked how I have built my knowledge to collect the sufficient knowledge from the ground up. And the following are the list of resources (blog posts, documentations, Youtube videos) that I have found especially helpful in building these prerequisite knowledge. Some are more popular than others, but all of them were essential in building a firm understanding and correcting misunderstood concepts. I have chosen the material such that there is as little duplicate as possible.

> ❗ For others joining the journey, please feel free to leave any material you found useful in ramping up your deep learning knowledge in the comment section!

> ❗ The following roadmap assumes you are generally proficient with Python and basic machine learning concepts (model, training, inference, etc.), so we can stack up deep learning concepts from thereon.

Deep learning fundamentals

The first concept to grasp is the deep learning fundamentals regarding its training and inference setup, as opposed to traditional machine learning. More specifically, it is important to fully understand the most primitive deep learning framework, also known as vanilla deep learning network, feed-forward network, or fully connected layer. One should be able to explain what each of the concepts mean to be sure they have understood the basics.

Multi layer perceptron, activation function, forward propagation, minibatch training, loss function, backpropagation, computational graph, optimizer, learning rate scheduler, batch normalization, dropout, hyperparameter optimization

These are the material I have seen to materialize the above concepts.

- Deep learning from scratch (Saito Goki): This is a book which explains and implements core deep learning blocks from scratch, only using Python and numpy. I read this book in Korean, thinking the original book is in English, but it turned out that the original book was in Japanese, so there is no English version. Instead, for those English speakers you can refer to the famous Stanford C231n lecture, as this book refers to a lot of concepts and key ideas from this lecture. Especially the section on computational graphs completely demystified the mathematics and implementation of backpropagation, a concept commonly lacking concrete explanation.
- Youtube 3Blue1Brown Neural Network Series #1~#4: This is part of a Youtube series made by 3Blue1Brown, one of the greatest Youtubers for visualizing math and statistics. These four videos on deep learning's training and inference almost creates a delusion that deep learning is simple, by visualizing and breaking concepts down into manageable pieces. Each video is very dense, so there are always new insights to pick up in multiple replays.
- Youtube StatQuest Neural Networks/Deep Learning Series #1~#Part 7: StatQuest is also a famous Youtuber who explains machine learning and statistics. There were usually a lot of traditional machine learning related content, then this video series came up. From the beginning to part 7 (cross entropy and backpropagation) is where you can pick up deep learning basics. StatQuest uses the method of breaking a complex concept down into the number-wise level, so you can see all the operations and arithmetic happening with absolutely no abstraction. There is no "roughly speaking", "if you take the derivative of this equation", "with these matrix operations" kind of skipping, which makes his explanation ever so clear. Especially for people like me who understand better with examples than generalized equations, StatQuest is the way to go. In the deep learning series you can pick up useful insights on why each layer is required and how deep learning networks can depict a complex data distribution. At first due to the cheerful and rather informal nature of the video the material may seem "easy" or even "childish", but that is what makes

StatQuest so frighteningly brilliant. He (who I am aware is a professor in biostatistics) can explain college level concepts as if you are explaining to a middle school student, which is an incredible ability.

Pytorch

- Pytorch Official Documentation “Introduction to PyTorch”, “Introduction to PyTorch on YouTube”: We have looked at the theoretical aspect of deep learning with the above material, but implementing the topic is a different problem. I have gone through many resources, but found no rival to Pytorch’s official documentation. As you see how Pytorch implements the concepts of forward propagation, backward propagation and how the gradients are applied to the parameters on the code level, you can solidify each concept even more. Another plus you can gain here is because Pytorch tutorial docs mainly handle image data as input, you can familiarize yourself with CNN based architectures.

RNN~Transformer

Now that the core fundamentals of deep learning is in place, it is time to follow the journey and history of how deep learning models evolved to handle variable length data (usually NLP).

- Youtube StatQuest Neural Networks/Deep Learning Series #RNN~#Attention: Enough of praise for StatQuest, this is the next part of the StatQuest deep learning series. Starting from RNN, going to LSTM, word embedding, seq2seq, and attention, you can follow the researchers’ footsteps on ways to handle variable length data more effectively. Absorbing each concept with StatQuest’s crystal clear explanation is important, but you will learn a lot more if you keep asking questions such as “why this is operation requried here?” or “why don’t they do this instead of what is shown here?”. By asking these critical questions you can more effectively understand the motivation behind each new technique. This section also requires many replays to fully absorb all the dense information.

Transformer

Now finally moving on to transformers!

- Youtube StatQuest Neural Networks/Deep Learning Series #Transformer~#End: These videos now deal with the transformers and its variants (such as the decoder-only transformer) and query key value operations. It would be helpful to focus on the difference between the transformer and its predecessors, which rely on recurrent or convolutional layers. It would be also nice to look at how transformers deal with variable length input and output data, and also utilizing parallelism simultaneously.

- Youtube 3Blue1Brown Neural Network Series #5~#End): The first four videos of 3Blue1Brown was a complete series on fundamental deep learning, but for some reason 3Blue1Brown has recently uploaded new videos to the series regarding transformers, especially GPT-3. 3Blue1Brown uses a different approach than StatQuest in that he illustrates transformers as a sum of all the large layers of GPT-3, breaking down all the massive number of weights into manageable chunks of information. I believe that these two equally brilliant approaches can compliment each other in understanding transformers better.

- “The illustrated transformer” Jay Alammar blog post: A magnificent hands-on illustration of the trasnformer by the famous deep learning researcher and educator, Jay Allamar. You can actually feel you are reading the “Attention is all you need” paper by just reading the blog post. It is that clear and straightforward. Though it is a superb explanation, to test your understanding and to appreciate the legendary manuscript per se one your own first, I recommend to come back to this blog post if you get stuck reading the original paper.

- Youtube “How a Transformer works at inference vs training time”: This may be the most unpopular material so far, but it was crucial in building my final piece of understanding. It is a video by an AI engineer in Hugging Face, and it clearly illustrates the differences of the transformer in training and inference time. In many resources people tend to focus on the inference time of the transformer and usually do not draw a clear line on how the transformer behaves differently in training and inference time. I was a little confused on how the transformer exploits its parallelism in its query key value operations, and this video cleared it all. Also, on the side the video also introduces a little bit of Hugging Face API and code conventions, which was a delightful bonus.

If you managed to get to this far, now you can fully appreciate and savor “Attention is all you need” quite naturally!

Attention is all you need - FAQ

In this section I will enumerate my questions and (later found) answers as I was reading the "Attention is all you need" paper. It is aimed for those who have already read the paper and looking for a deeper understanding.

❗ Why the sudden division by $\sqrt{d_k}$?

$$Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}})V$$

This equation holds the essence of transformer's attention. After understanding the idea of linearly combining values based on the similarity between the queries and the keys, the next question I had was, the purpose of that suspicious division by $\sqrt{d_k}$. The authors state that without it, the similarities "pushes the softmax function into regions where it has extremely small gradients", but I had little idea on what it meant exactly. Intuitively I can see that if a specific (query, key) pair is dominant all the other pairs will have a softmax value very close to 0.0. Only the value for the most dominant pair will be used, not the linear combination, and this will have a negative impact on training. But as there is the word "gradient" I thought this explanation is not enough, and it had to be related to the backpropagation of the softmax layer. Ultimately, I realized if there is a dominant (query, key) pair, the gradient when backpropagating will be very close to 0.0, creating a sort of vanishing gradient problem. I have found the exact equation and the backpropagation of the softmax function in this wonderful article. .

❗ What does Outputs (shifted right) mean?

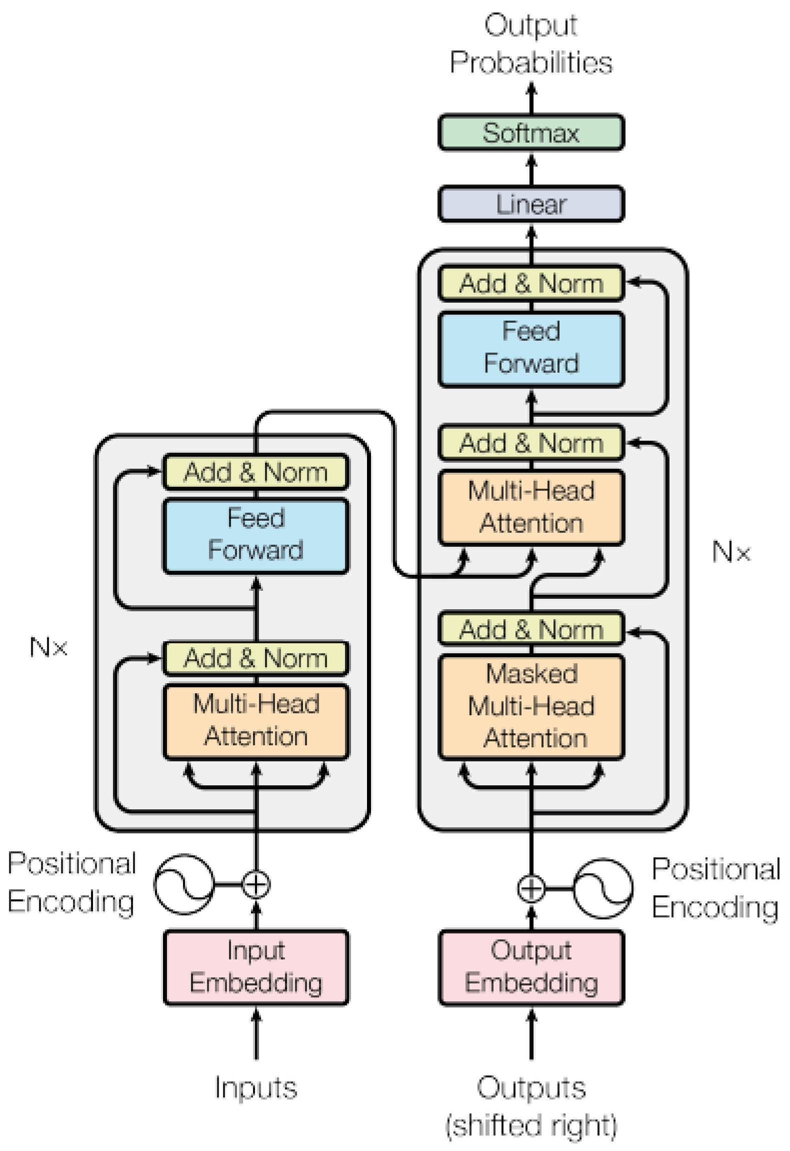

Figure 1: The Transformer - model architecture.

I was wondering what would the "Outputs (shifted right)" mean in the bottom right of this famous figure. This setup is very popular in variable input variable output models, and this is used because such models tend to use all the words (or more correctly tokens) before a specific point to predict what comes right after. In the initial decoding stage of the inference time there is no previous output, so we have to use the SOS (start of sentence) token as the input to generate the first predicted output token. Then with the SOS token and the previous output as the input we generate the next token, and so on. This is similar in training time, where we instead push in the SOS token, followed by the whole input token to the decoder, to calculate the loss for the whole sequence length. Thus, for variable input variable output models, we need the initial SOS token for the decoder, hence the name "shifted right" makes sense.

❗ How can we perform Q, K, V operations in parallel if in the decoder we have to wait for the previous prediction as the new input?

In most transformer explanation materials, there tends to be a focus on inference time, discussing decoder-only transformers like ChatGPT. This question arose because I didn't understand the setup differences between the inference time and the training time of transformers. One of the biggest advantages of transformers is their ability to leverage parallel computation using GPUs in NLP tasks, which were traditionally known to require recurrent networks. However, I was curious whether this advantage loses some of its meaning if the decoder layer has to wait for the previous inference results, thus potentially diminishing the purpose of parallelization.

This curiosity was resolved through the video discussing the difference between transformer inference and training times.

During inference, as we saw above, sequential operations are required because the output from the previous step in the decoder directly becomes the input for the next step. However, this is the inference phase, so even though there isn't much parallelization, it doesn't greatly impact performance compared to training. On the other hand, during training, all input tokens are provided to the encoder, and the decoder receives right-shifted correct answer tokens. The actual encoder and decoder only need to perform a forward/backward pass once per input, allowing parallel operations without waiting for inference on each word. During training, the act of ignoring the decoder's output (likely different from the correct answer) and giving the correct right-shifted answer tokens is called "teacher forcing".

❗ How can we properly perform self attention operations in parallel and memory effieciently if the lenghth of input for the encoder and the decoder constantly change?

After examining the operation of training and inference times in transformers, the next thought that came to mind was whether efficient parallelization and memory allocation optimization could be achieved when processing short input sentences like "I am a student" versus longer passages of several hundred words. Logically, it seemed that the shift from O(n^2) memory and time complexity when handling short versus long sentences would not favor parallelization.

Therefore, actual implementations of transformers often set a maximum context length (the length of input tokens) and adopt a strategy where shorter sentences are padded to match this maximum length. This approach allows transformer models to maintain a consistent level of memory and computational resources during training.

> Does the sequence size of the encoder and decoder has to match in the encoder-decoder self attention layer?

Many resources explaining transformer self-attention often assume that the sequence sizes of Q, K, and V are equal, which typically holds true for self-attention within the encoder and masked self-attention within the decoder. However, during encoder-decoder attention, due to the variable-length nature of models, the sequence size on the encoder side may not match that on the decoder side. Nevertheless, the attention mechanism operates effectively even in such cases.

In encoder-decoder attention:

The encoder provides K and V. The decoder provides Q. Looking solely at the matrix dimensions, the matrix multiplication involved can be described as follows. I will omit the scaling with $\sqrt{d_k}$ for simplicity. I will use $d_{model} = 512$ as stated in the paper.

$$K : (n, 512), Q : (m, 512), V : (n, 512)$$

$$QK^t : (m, n)$$

$$(QK^t)V : (m, 512)$$

Thus, as you can see self attention is correctly performed even if encoder and decoder's input sequence lengths are different.

> Why do we need to multiply $\sqrt{d_{model}}$ to the input before adding the positonal encoding value?

I had difficulty understanding the rationale behind the statement "In the embedding layers, we multiply those weights by $\sqrt{d_{model}}$." in the paper. While it's quite noticeable why we divide by $\sqrt{d_{model}}$ in the attention mechanism, I found it puzzling why we multiply by $\sqrt{d_{model}}$ before adding positional encoding. Upon further investigation, I came across a hypothesis that seemed plausible, similar to the answer provided in this post. The explanation suggests that embedding weights are initialized to follow a standard normal distribution, and using them directly would overly influence the positional encoding values, which typically range between -1 and 1. Therefore, intentionally scaling up the embedding vector weights compared to the positional encoding helps to mitigate this effect.

❗ Please feel free to share your interesting questions about the paper in the comments!

Equivalent Korean Post

Attention is all you need - 전설의 논문을 온전히 흡수하는 방법

References

- https://arxiv.org/abs/1706.03762
- https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk
- https://pytorch.org/tutorials/beginner/basics/intro.html
- https://www.youtube.com/watch?v=zxagGtF9MeU&list=PLblh5JKOoLUIxGDQs4LFFD–41Vzf-ME1
- https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- https://jalammar.github.io/illustrated-transformer/
- https://www.youtube.com/watch?v=IGu7ivuy1Ag
- https://towardsdatascience.com/transformer-networks-a-mathematical-explanation-why-scaling-the-dot-products-leads-to-more-stable-414f87391500
- https://stats.stackexchange.com/questions/534618/why-are-the-embeddings-of-tokens-multiplied-by-sqrt-d-note-not-divided-by-s

❗ Please feel free to point out any inaccurate or insufficient information. Also, please feel free to leave any questions or suggestions in the comments. 🙇

DEEP LEARNING

transformers nlp

Share:

Further Reading

Apr 30, 2024

Attention is all you need - 전설의 논문을 온전히 흡수하는 방법

약 8년 늦은 리뷰? 나름 데이터 분석과 머신러닝을 공부하고 실무에 사용하고 있지만, 정작 현재 AI 기술을 이끌고 있는 트랜스포머에 대해서는 교양 수준으로 이따금씩 전해 들은(그리고 곧 까먹은) 것이 전부라 …

OLDER

Attention is all you need - 전설의 논문을 온전히 흡수하는 방법

NEWER

Dkron full setup - towards a more reliable and highly available cron