Attention is all you need: understanding with example

Mehul Gupta · Following
Published in Data Science in your pocket · 14 min read · May 4, 2021

'Attention is all you need' has been amongst the breakthrough papers that have just revolutionized the way research in NLP was progressing. Thrilled by the impact of this paper, especially the 'Attention' layer, I tried my hands on this paper & penned my understanding in this post.

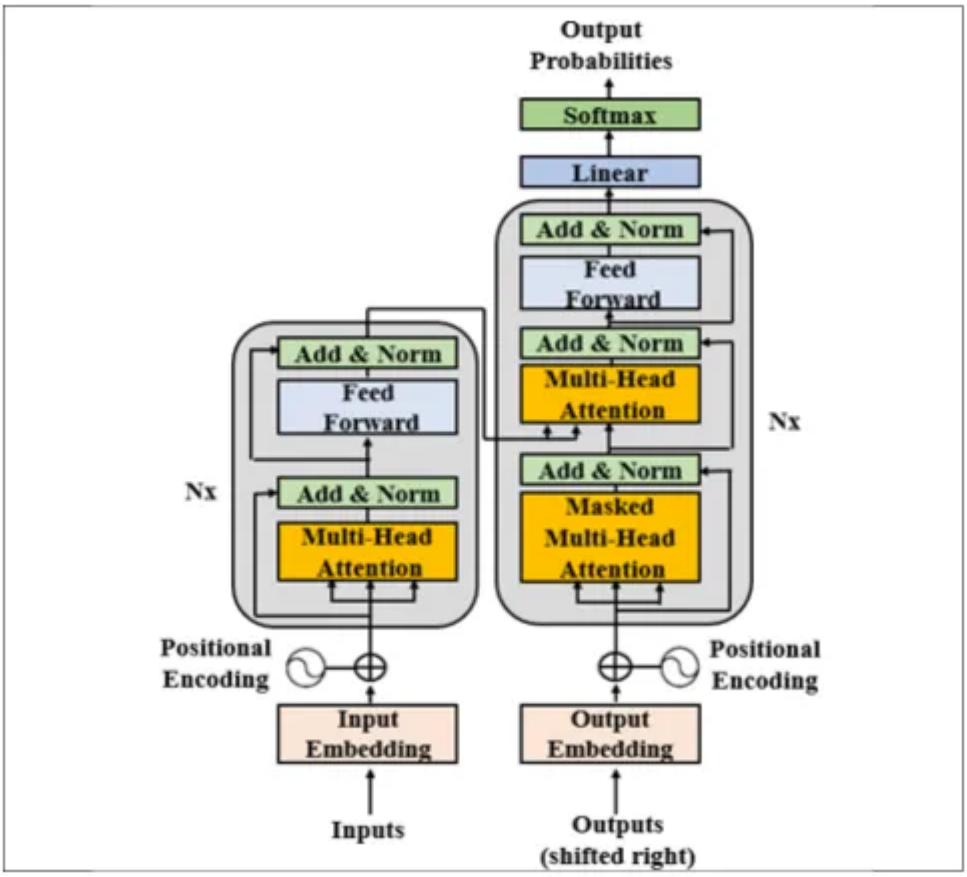
Before starting, we must know what a Transformer is.

A Transformer model basically helps in transforming one sequence of input into another depending on the problem statement. This transformation can be

> *Translation from one language to another (say English to Hindi)*
>
> *An answer (output sequence) for a question (input sequence)*

...etc. with the help of an Encoder & Decoder model stacked together. I guess you are aware of the core idea of Encoder & Decoder !!

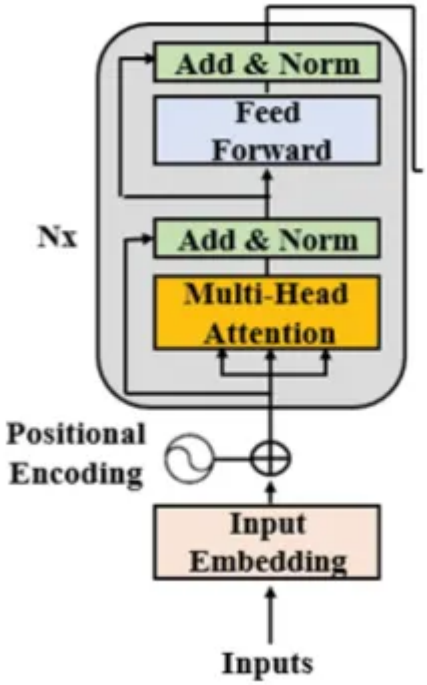
The paper explains the 'Transformer' model shown below



Looks mammoth !! isn't it?

Let's start off with the Encoder (left-hand side) of this daddy network.

ENCODER



The aim of an Encoder is to see the entire input sequence at once & decide which tokens are important corresponding to other tokens in the sequence using the Attention Layer described in the later sections producing altered embeddings incorporating this 'attention'

I will take a bottom-up approach to explain the structure starting from 'Inputs'

'Inputs' is the numeric representation (not embedding) of the sequence that has to be transformed. Say, we aim to convert **'I am a boy'** to German. As text directly can't be used as an input for any neural network, a numeric representation generated for each token (I, am, a, boy) using a tokenizer is generated & fed to the encoder. We won't be discussing the tokenizer part

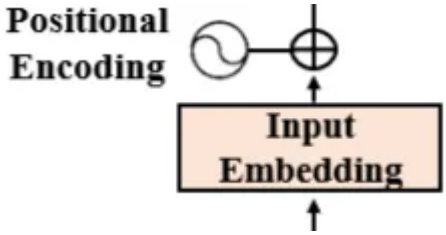
The Input Embedding layer helps in generating meaningful embeddings of dimension 512 for each token. Hence, if the naïve numeric representation of **'I am a boy'** is, say, [1,2,3,4] (or mostly one-hot -encoded), then after the input embedding layer, it would be of the dimension 4x512 (1 vector of size

512 for each token). Also, related words will have approximately similar embeddings & vice versa. Hence, if we get the words ‘child’ and ‘children’, their embedding should be similar but not for ‘child’ and ‘water’.

So, this was something I assume you might know pretty well even before the post.

> Now comes the real meat !!

POSITIONAL ENCODING



It has been observed that though embeddings generated in Input Embedding layers helps to determine how similar two words are, it generally doesn't account for the position at which the two words appear. Let's say we have two sentences:

> *He has a black Ferrari. He also has a white cat.*
>
> *He has a black & white Ferrari*

Now, if we only use Input Embedding Layer, both 'black' & 'white' will have similar embedding in the 2 sentences. Though, if you notice in sentence 1, both the words are far away & are related to totally different entities. Right?

Hence, Another embedding is generated which is added to the output of the Input Embedding layer to ensure information about the position of every token is also considered.

How is this Positional Embedding calculated?

$$PE_{(pos\ 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos\ 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Breaking the formulae down,

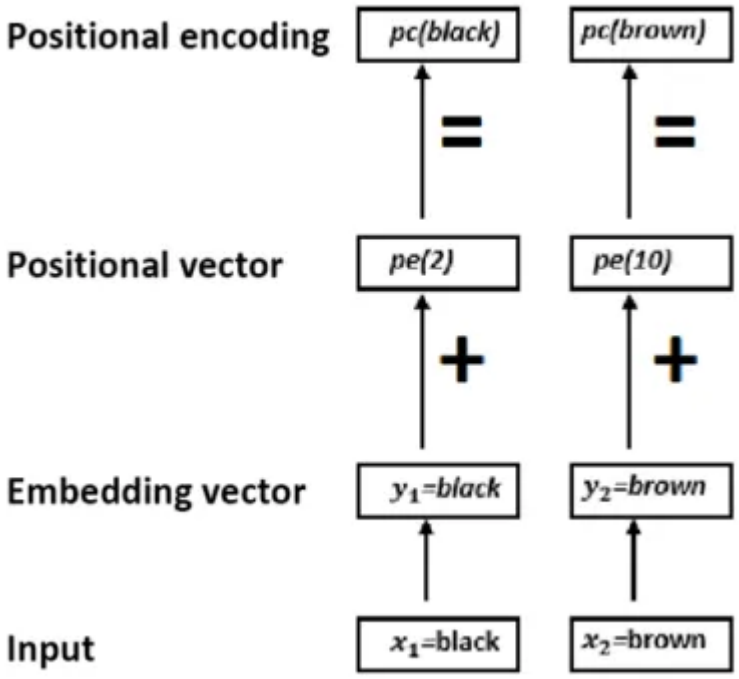
> *For every token at index 'pos' in output Embedding from 'input embedding' layer (pink box in the diagram above), we will generate a psoition vector of dimension 512 (i.e. d_model which is equal to embedding dimension for each token) where for every even embedding index (from 0 →512 i.e 0,2,4,6,.....510), we follow the 1st formula else the 2nd one for odd indexes(1,3,5,7....511).*

Hence, suppose we need to generate Positional Encoding for '**boy**' in '**I am a boy**', then we have — pos=3 ('boy' is the 4th token of the sequence)

Let the original Embedding be [1.2,3.2.....,1.1] of the size 1x512 for 'boy'

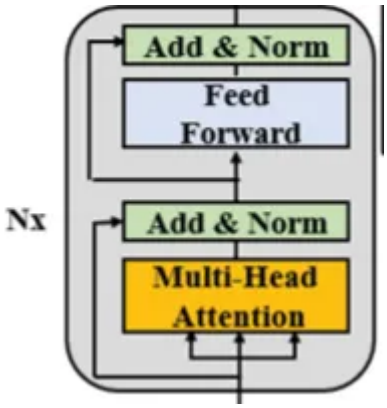
> *The Position Vector will be:*
>
> *[sin(3/1000^0),cos(3/1000^0),sin(3/1000^{4b}),cos(3/1000^{4b}).....cos(3/1000^{1022b})] of the size 1x512 where b=1/512 for 'am'. Similarly, it can be calculated for other tokens as well*

Now, this Positional vector will be added to the original Embedding.



How Positional Encoding generated for words ‘Black’ & ‘Brown’

Once this Positional Embedding is calculated, comes the core of this Encoder



It comprises 4 segments that are repeated for ‘Nx’ times. Nx=6 in the paper. Let us dive into each of the 4 components one by one

The Multi-Head Attention Layer

> *But, what is attention?*

Attention is basically a mechanism that dynamically provides importance to a few key tokens in the input sequence by altering the token embeddings.

In any sentence, there exist a few keywords that contain the gist of it. Say, in ' I am a boy ', 'am' is of not the same importance as of 'boy' when it comes to understanding the meaning of the sentence. So, to make **our model focus on important words of a sentence**, the attention layer comes in very handy & improves the quality of results produced by neural networks

> *But, can't LSTM or RNN or any other seq2seq model be used?*

Not really, as LSTM or RNN may have some memory, but when it comes to complex tasks like Language Translation, the model may need to remember more than an LSTM's or RNN's capacity.

A Multi-Head Attention Layer can be considered a **stack of parallel Attention Layers** which can help us in understanding different aspects of a sentence or rather a language. Assume it to be different people given a common question. Each one of them will understand it differently & answer accordingly which may/may not be the same. For example: 'He sat on the chair & it broke'. Here, one of the attention heads may associate 'it' with chair & other may associate it with 'He' . Hence, to get a generalized & different perspective, multiple attention heads are used.

> *Now, as we have are clear with the motto, let's jump into mathematics*

Each head in the Multi-Head Attention Layer intakes the new embedding (Positional Encoding generated in the last step) which is **n x 512** in the dimension where 'n' is the tokens in the sequence & produces an output of shape **n x 64 each.** This output from all heads is then concatenated to produce a single output of the Multi Headed Attention module of the dimension **n x 512.** In the paper, 8 attention heads are used.

> *All this is fine, but how does an Attention Layer work?*

Before moving onto that, a few concepts are a must to know. We should start with 3 matrices pair used in Attention Layer

- Query & Query_weights
- Key & Key_weights

- Value & Value_weights

Where Query, Key & Value have dimension **n x 64** where 'n'= tokens in the input sequence. Here, will define a few generally used notations throughout the paper

> ***d_model*** *= 512 (dimension of embedding for each token)*
>
> ***d_k*** *= 64 (dimension of Query & Key vector)*
>
> ***d_v*** *= 64 (dimension of Value vector)*

Note*: It must be noted that the dimension can be changed accordingly & there exists no specific reason to choose 512,64 & 64 respectively.*

Considering dimensions of Query, Key & Value matrices above, dimensions of the 3 weights matrices are

- Q_w = d_model x d_k =512 x 64
- K_w = d_model x d_k = 512 x 64
- V_w = d_model x d_v = 512 x 64

> *But, what it has to do with* ***attention****?*

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

So, using Query, Key & Value matrices, Attention for each token in a sequence is calculated using the above formula.

> *Will follow up with a small mathematical example to make life easier!!*

Let us assume

- We have an input **'Raj is good'.**
- After passing through the tokenizer, we get [1,2,3] as input for the transformer (assume label encoding)
- Let's assume d_model = 4, hence embedding per token=4(just for easy representation, I can't demo using 512 as embedding size). Hence the output of 'Input Embedding' layers would be
- 3(tokens in 'Raj is good') x 4(d_model).

```
[[1.0, 0.0, 1.0, 0.0],
 [0.0, 2.0, 0.0, 2.0],
 [1.0, 1.0, 1.0, 1.0]]
```

Here, each row represent Positional Encoding of each token of the input sequence

Now, let's understand Attention considering the above matrix as Input Embeddings

- Let d_k,d_v = 3 & not 64 for our example. Hence Q_w,K_w & V_w will be of the dimension 4(d_model)x 3(d_k or d_v). Let us initialize these matrices as below:

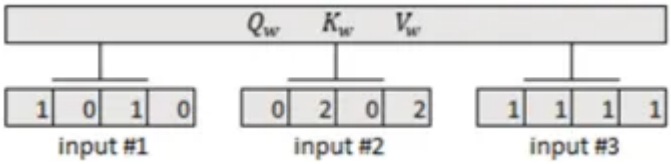
```
[[1, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 1, 1]]
```

```
[[0, 0, 1],
 [1, 1, 0],
 [0, 1, 0],
 [1, 1, 0]]
```

```
[[0, 2, 0],
 [0, 3, 0],
 [1, 0, 3],
 [1, 1, 0]]
```

Query, Key & Value matrices from Left to Right

Now, we are ready with input embeddings & our initial weight matrices. The below picture gives a clear picture of what has happened till now



Each row of Input Embedding represents a token of the input sequence. Row 1 represent Raj, Row 2: is, Row 3:good

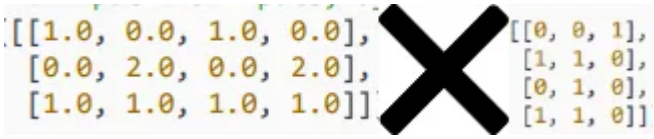
- The next step is pretty simple, we need to multiply the above-assumed Input Embedding (3 x 4 matrix defined in point 3) with Q_w, K_w & V_w respectively.

```
[[1.0, 0.0, 1.0, 0.0],
 [0.0, 2.0, 0.0, 2.0],
 [1.0, 1.0, 1.0, 1.0]]
```

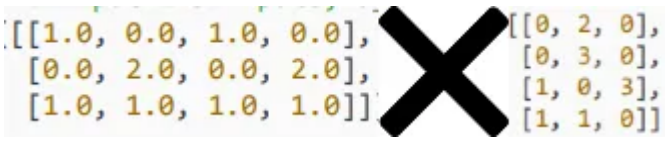
×

```
[[1, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 1, 1]]
```

Positional Embedding x Query_weights

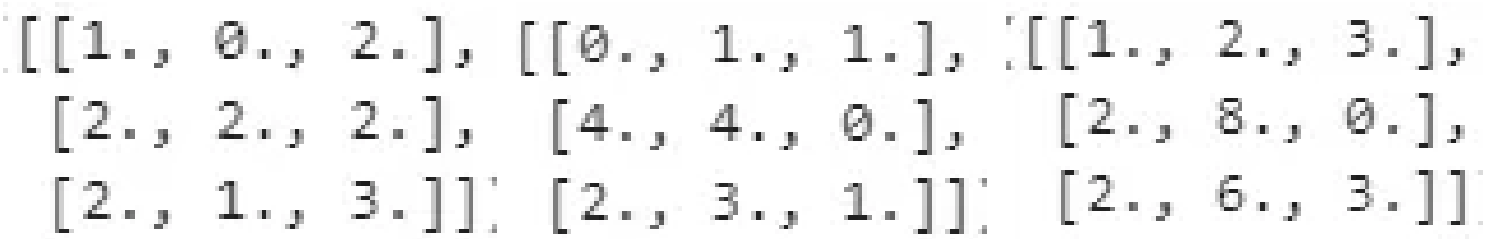


Positional Embedding x Key_weights



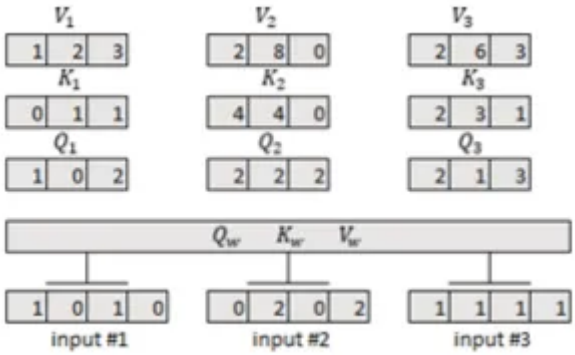
Positional Embedding x Value_weights

The output Query, Key & Value matrix are:



Query, Key & Value (Left-Right)

We must keep in mind that each row of the above matrices corresponds to one of the input sequence tokens as shown in the below diagram



Q1, K1, V1 corresponds to Query, Key & Value for 1st token & likewise for other tokens. Observe the similarity between this & the above figure

But, where did the monstrous formula for Attention go?

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

Going one term at a time, Let us 1st calculate the below term 1st:

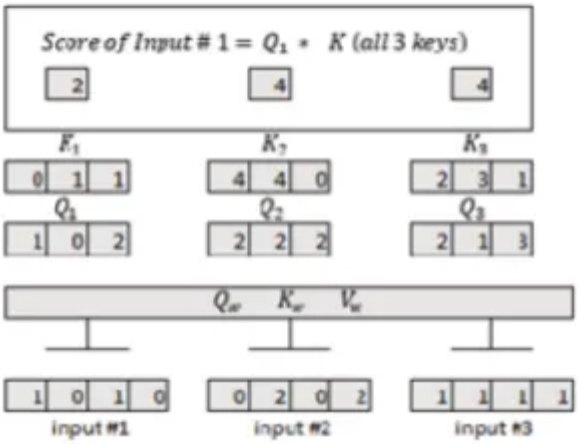
$$\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

As d_k = 3(assumed earlier), root(d_k) is approximated to 1 for ease of calculation.

Hence, Q x K_Transpose/1 =

```
[[ 2. 4. 4.]
 [ 4. 16. 12.]
 [ 4. 12. 10.]]
```

The pictorial representation of what's going on has been added below. The above matrix can be called as **Score matrix.** The below image shows the Scores for 1st token.



The score for 1st token (corresponds to 1st row of Score matrix)

Now, we need to apply softmax across each row of the above output.

Hence, softmax(QxK_Transpose/1)=

```
[[0.06337894, 0.46831053, 0.46831053],
 [6.03366485e-06, 0.982007865, 0.0179861014],
 [0.000295387223, 0.880536902, 0.119167711]]
```

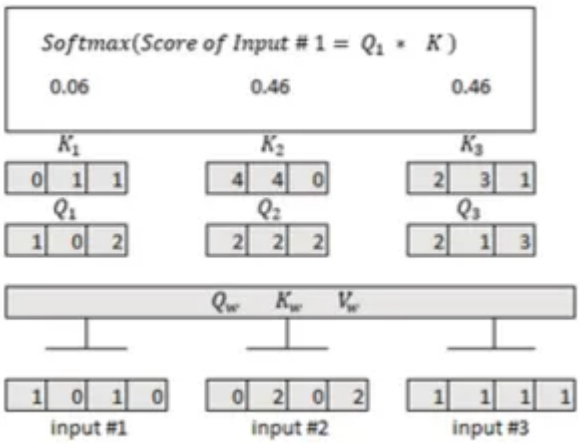
Softmaxed scores for each token. Here also, 1st Row: Raj, 2nd Row:is & 3rd Row:good

The softmaxed scores for a token represent the importance of tokens corresponding to other tokens. For example, Softmaxed score for 1st token ‘Raj’ =[0.06,0.46,0.46] implies the Importance of ‘Raj’ for

> *Raj=0.06*
>
> *is=0.46*
>
> *good=0.46*

More the score, the more the importance of that token corresponding to that token (including itself).

After applying softmax() on the Score matrix, this is what happens in the Attention Layer for 1st token



The softmaxed score for 1st token (corresponds to 1st row of softmaxed Score matrix)

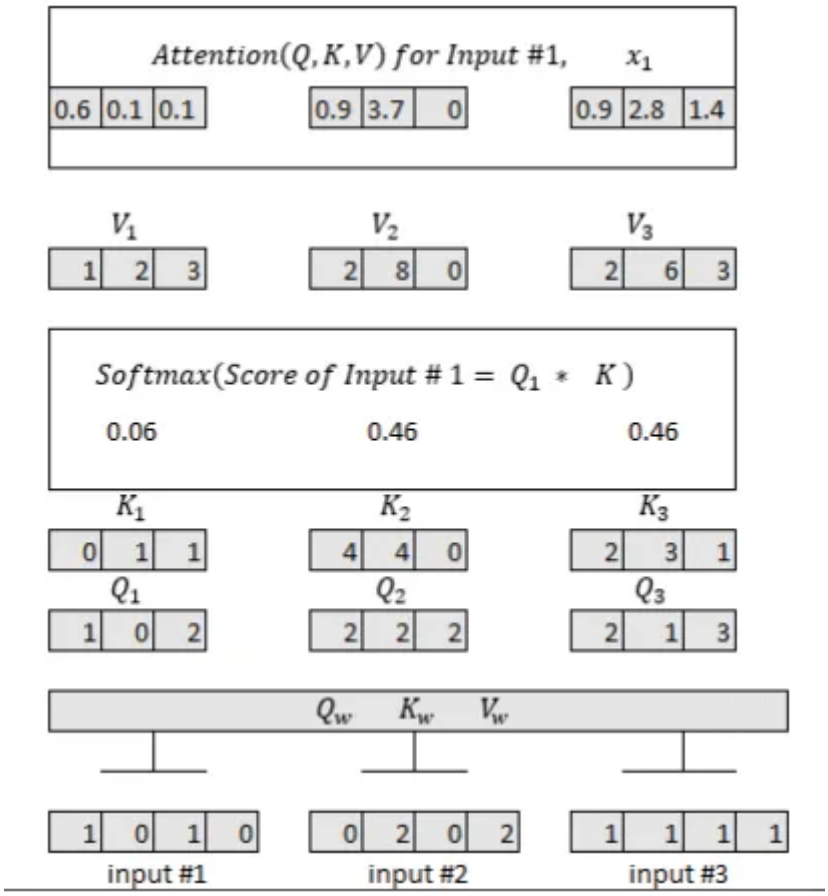
So, left with the last part of the equation, multiplication of the above-**softmaxed value with Value matrix.** But with a twist. We will be calculating 3 attention values for each token corresponding to all other tokens in the sequence including itself. If total tokens were 5,6 or any other number, that many attention values would have been calculated

What we will be doing is calculate Attention for 1st token i.e. row 1 below. For this, we need to multiply each value of row 1 in the softmaxed score matrix with the corresponding index row in the Value matrix (observe Value matrix declared above) i.e.

> *0.063337894([0,0] in softmaxed matrix)*[1. 2. 3.](1st row, Value matrix) =*
>
> *[0.06337894, 0.12675788, 0.19013681] i.e. A1*
>
> *0.46831053([0,1] in softmaxed matrix)*[2. 8. 0.](2nd row, Value matrix) =*

> *[0.93662106, 3.74648425, 0.] i.e. A2*
>
> *0.46831053([0,2] in softmaxed matrix)*[2. 6. 3.](3rd row, Value matrix)=*
>
> *[0.93662106 2.80986319 1.40493159] i.e. A3*

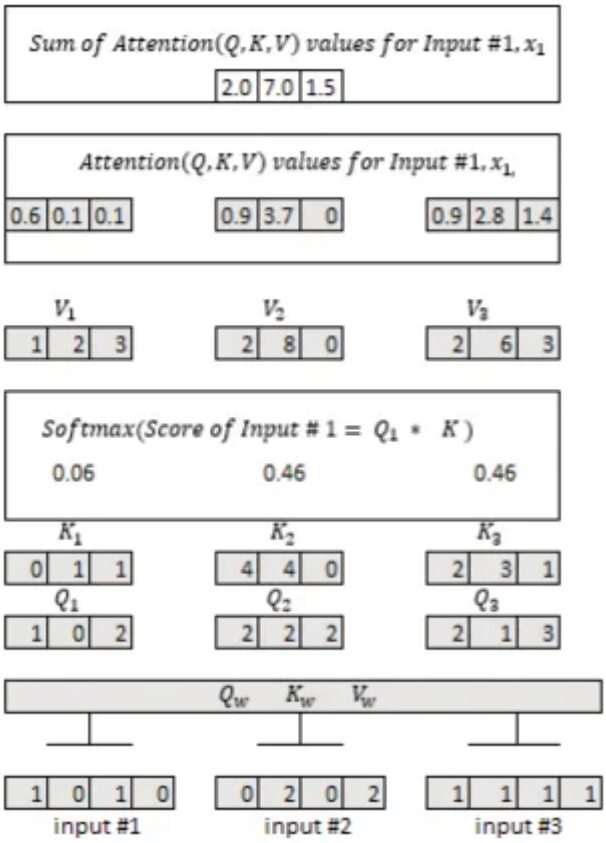
Observe how the 3 attention vectors are calculated for 1st token (values are rounded off for ease of understanding)



Now, we need to add these 3 vectors A1+A2+A3=

[1.93662106 6.68310531 1.59506841]]

And finally, attention for 1st token is calculated !!



Similarly, we can calculate attention for the remaining 2 tokens (considering 2nd & 3rd row of softmaxed matrix respectively) & hence, our Attention matrix will be of the shape, n x d_k i.e. 3 x 3 in our case.

Now, coming back to the paper where we have 8 such attention heads. In this case, we will concatenate output matrices from all heads & this concatenated matrix is multiplied with a weights matrix such that output = n x d_model which was the input shape for this Multi-Head Attention layer.

$$MultiHead(output) = Concat(z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7,)\ W^0 = x,\ d_{model}$$

Here, z_i represent attention matrix output from the ith head

Hence, to summarize

> *Calculate Q, K & V using Q_w, K_w & V_w & Input sequence Embeddings*
>
> *Calculate Attention Matrix of dimension* ***n x d_k.*** *This involves a few steps shown above*
>
> *Concatenate Attention matrix from all attention heads & multiply with a weights matrix such that the output = n x d_model.*

Just before ending, we must know why all this mathematics helps in calculating attention values? What is the significance of Query, Key & Value matrices? To know this, do read this section.

That's all for attention !!

Post Layer Normalization

With an idea to not lose out on essential information, normalization uses residual connections to look back at the Input of the previous layer & its output simultaneously. This is done in the following steps

1. Add Input & Output of the previous layer, whatever it may be. In this case, the Multi-Head Attention Layer is the previous layer. Hence Input & Output of this layer is added.

Let this be V (of the dimension n x d_model)

2. Normalize V using the below formula

$$\gamma \frac{v - \mu}{\sigma} + \beta$$

Here, μ= Mean, σ=Standard Deviation, γ= Scaling/Damping factor, β=Regularization constant. It must be kept in head that μ & σ are calculated separately for each row 'r' of V (Input + Output matrix, previous layer).

Next is a **Feed-Forward Network** comprising of 2 layered neural networks applying ReLU. Also, though the input & output dimension remains same, the 1st layer has an output dimension= 2048

After this FNN, again a Post Layer Normalization is done with Input(FNN) & Output(FNN) similar to what happened after Multi-Head Attention. Now, the 4 segments:

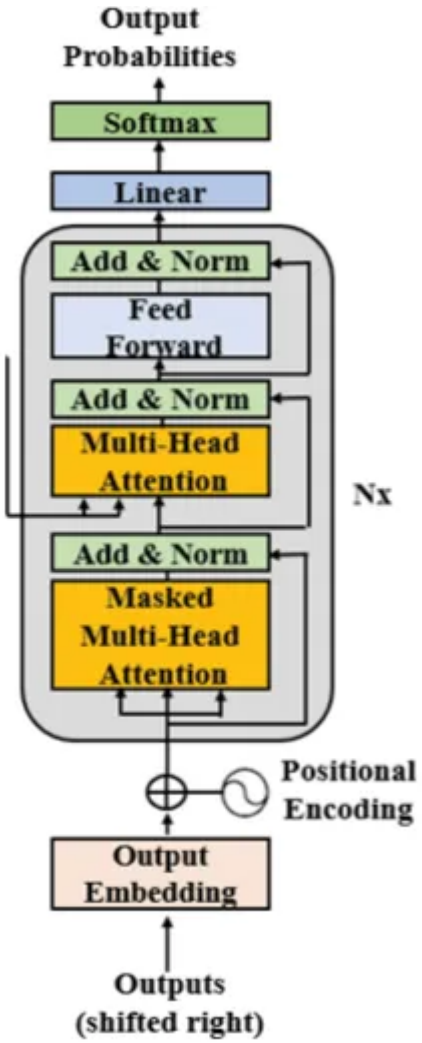
> *Multi-Head Attention (8 heads)*

> *Normalization*
>
> *Fee-Forward Network*
>
> *Normalization*

is repeated for 6 iterations

The final output from the 2nd Normalization Layer in the 6th iteration is of the dimension **n x d_model** & is passed to the Decoder which is the attention matrix for the input sequence. We will discuss how & to which segment does this goes in the below explanation

DECODER



Another monster in the house !!

The major aim of using a Decoder is to determine the output sequence's tokens one at a time by using:

- Attention known for all tokens for the input sequence from Encoder
- All predicted tokens of output sequence so far.
- Once a new token is predicted, it is considered to determine the next token. This chain goes till the 'End Of Sentence' token isn't predicted.

As followed in Encoder, I will go through this network in a bottom-up approach

1. **'Outputs'** is the numeric representation of the Output Sequence generated using a tokenizer as done in Encoder but with a difference. This numeric representation is right-shifted.

But why right-shifted?

As the decoder is trained in such a way that it is able to predict the next word of the sequence given the previous tokens & attention from Encoder. Now, for the 1st token, this looks troublesome as there exist no previous tokens. This would have lead to a problem to predict the 1st token every time. Hence, the output sequence is shifted & a 'BOS' (Beginning of Sentence) is inserted at the beginning. Now, when we need to predict the 1st token, this BOS becomes the previous token of the output sequence.

2. The Output Embedding & Positional Embedding layers have the same role & structure as in Encoder.

3. The core of the decoder changes a bit compared to the Encoder's core with the addition of Masked Multi-Head Attention, though, repeated for 6 iterations similar to Encoder's core.

4. In **Masked Multi-Head Attention Layer,** attention is applied on tokens up to current position (index till which prediction is done by transformer) & not future tokens(as aren't predicted till now). This is in stark difference from Encoder where attention is calculated for the entire sequence at once.

> ***For example****: If we wish to translate 'I am good' (input for the encoder, attention will be calculated for all tokens all at once) into French i.e 'je vais bien' (input for decoder), & the translation has reached till 'vais'(2nd token), hence, 'bien' would be masked & attention will be applied for the 1st 2 tokens. This is done by setting future tokens (embedding for 'bein')as infinite values.*

After this, a Normalization Layer same as in Encoder is followed.

All this is good but where did the output from Encoder go?

How Decoder is using it?

After the Normalization layer, a Multi-Head Attention layer follows which

- Intakes the output from Encoder (n x d_model; remember the final output from Encoder!), calls this output K & V which is used as Key & Value by Decoder's Multi-Head Attention.
- Also, the Query matrix from the previous Masked Multi-Head Attention layer is taken. Hence, this attention layer doesn't require any training as takes pretrained values for Query, Key & Value matrices.

$$Input\_Attention=(Output\_decoder\_sub\_layer\text{-}1(Q), Output\_encoder\_layer(K,V))$$

This layer uses the pretrained information from Encoder. As Query vector will be available for just 'seen' tokens (predicted tokens), even this Multi-Head Attention Layer can't see beyond what isn't predicted in the output sequence similar to Masked Multi-Head Attention Layer.

After this layer, The following layers comes in

- Normalization
- FNN
- Normalization

Functioning similarly to as in Encoder. Hence, the decoder core has

> *Masked Multi-Head Attention Layer + Normalization*
>
> *Multi-Head Attention Layer + Normalization*
>
> *FNN + Normalization*

- After these repeated code blocks, we have a **linear layer followed by a softmax** function giving us the probability for the aptest token in the predicted sequence
- Once the most probable token is predicted, it goes back in the tail of the output sequence (remember the right-shifted sequence).

Hence, if we have 2 tokens in the output sequence till now out of 4 tokens (apart from BOS, i.e. two tokens have been predicted),

- The current aim of the decoder is to predict the 3rd token of the output

- Once its 3rd token is predicted, it goes to the tail of the output sequence & a new iteration starts
- In this new iteration, we have 3 tokens in the output sequence(apart from BOS, the previous two tokens & newly predicted token) & Decoder now aims to predict the 4th token
- If the predicted token is 'End Of Sentence'(EOS), the transformation is done & the output sequence is completely predicted.

Next, will come up with BERT, one of the breakthrough models that uplifted the entire NLP game.

A big thanks to:https://www.amazon.in/Transformers-Natural-Language-Processing-architectures-ebook/dp/B08S977X8K

> With this, it's a sign-off!!

NLP Transformers Attention Machine Learning Artificial Intelligence

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

15 min rea

View list

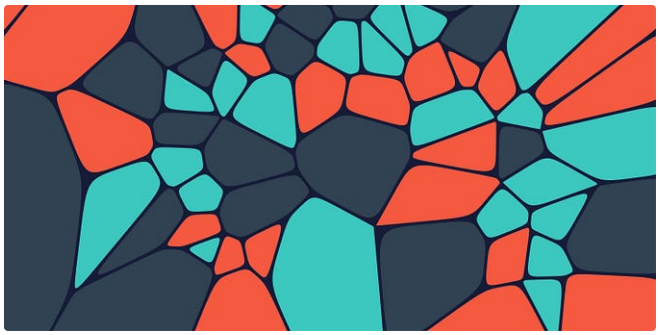
Written by Mehul Gupta

Following

1.4K Followers · Writer for Data Science in your pocket

Data Scientist @ DBS Bank | Youtube: https://www.youtube.com/channel/UCQoNosQTIxiMTL9C-gvFdjA

More from Mehul Gupta and Data Science in your pocket



Mehul Gupta in Data Science in your pocket

Langchain tutorials for newbies

Langchain use cases with demo explained

4 min read · Aug 20

73 1

Mehul Gupta in Data Science in your pocket

What are Vector Databases and How Langchain uses Vector DBs

with codes and examples

6 min read · Sep 21

99

Mehul Gupta in Data Science in your pocket

How to create a custom OpenAI Gym environment? with codes

Creating a game environment in OpenAI-gym from scratch

5 min read · Jul 11

32

Mehul Gupta in Data Science in your pocket

Deep Q Networks (DQN) explained with examples and codes in...

Value-based methods in reinforcement learning explained with code

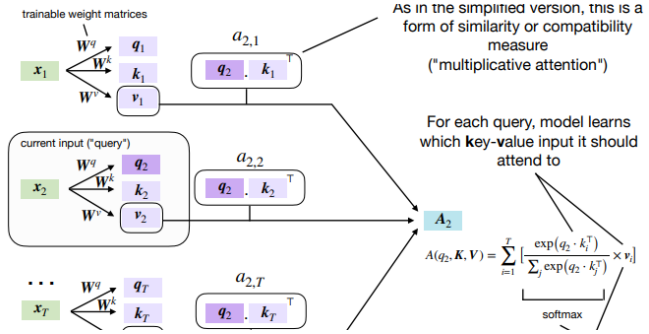
7 min read · Apr 8

105

See all from Mehul Gupta

See all from Data Science in your pocket

Recommended from Medium



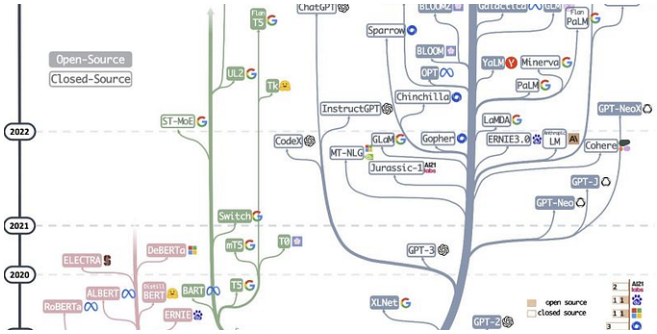
Zain ul Abideen

Attention Is All You Need: The Core Idea of the Transformer

An overview of the Transformer model and its key components.

6 min read · Jun 26

144



Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

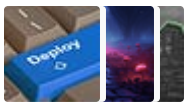
15 min read · Sep 14

372

Lists

Natural Language Processing

669 stories · 283 saves

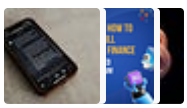


Predictive Modeling w/ Python

20 stories · 452 saves

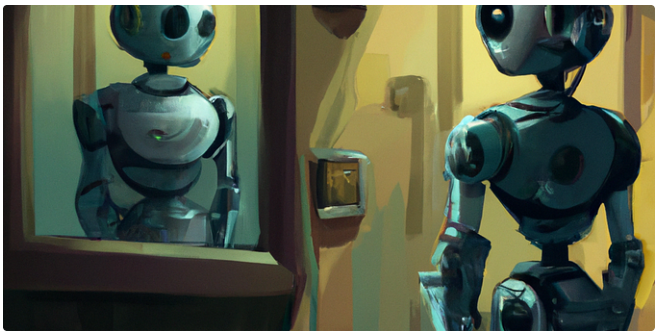
AI Regulation

6 stories · 138 saves



ChatGPT prompts

24 stories · 459 saves



Thomas van Dongen in Towards Data Science

Demystifying efficient self-attention

A practical overview

20 min read · Nov 7, 2022

477 2

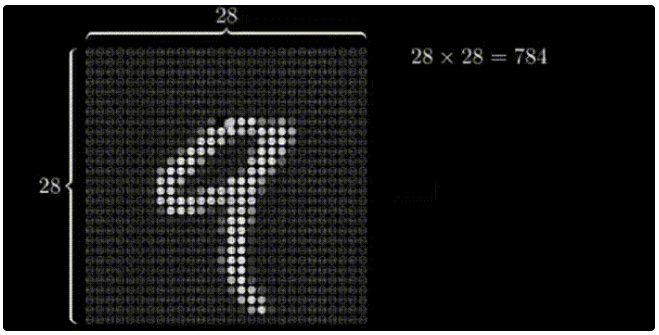
David Shapiro

A Pro's Guide to Finetuning LLMs

Large language models (LLMs) like GPT-3 and Llama have shown immense promise for...

12 min read · Sep 23

283 6



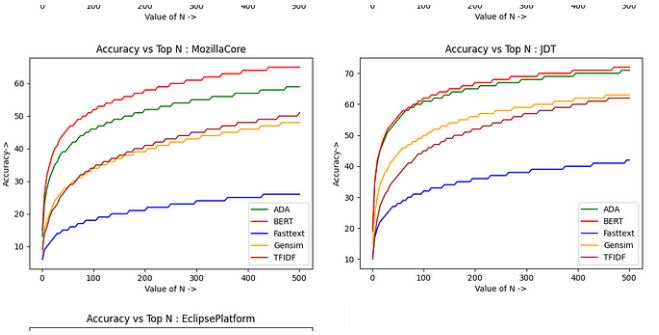
Sadaf Saleem

Neural Networks in 10mins. Simply Explained!

What are Neural Networks?

9 min read · May 15

252 2



Avinash Patil

Embeddings: BERT better than ChatGPT4?

In this study, we compared the effectiveness of semantic textual similarity methods for...

4 min read · Sep 19

3 1

See more recommendations