✦ **Jump-start your best year yet:** Become a member and get 25% off the first year

All you need to know about 'Attention' and 'Transformers' — In-depth Understanding — Part 2

Attention, Self-Attention, Multi-head Attention, Masked Multi-head Attention, Transformers, BERT, and GPT

Arjun Sarkar · Follow
Published in Towards Data Science
9 min read · Sep 13, 2022

Listen Share ••• More

In the previous story, I have explained what is the Attention mechanism, and some important keywords and blocks associated with Transformers, such as Self Attention, Query, Keys and Values, and Multi-head Attention.

To understand more about these topics please visit Part 1 of this story — All you need to know about 'Attention' and 'Transformers' — In-depth Understanding — Part 1.

In this part, I will explain how these blocks of Attention help create the Transformer network and discuss in detail all the blocks in the network.

Contents:

4. Basics of GPT (To be covered in Part 3)

5. Basics of BERT (To be covered in Part 3)

3. The Transformer Network

Paper — Attention Is All You Need (2017)

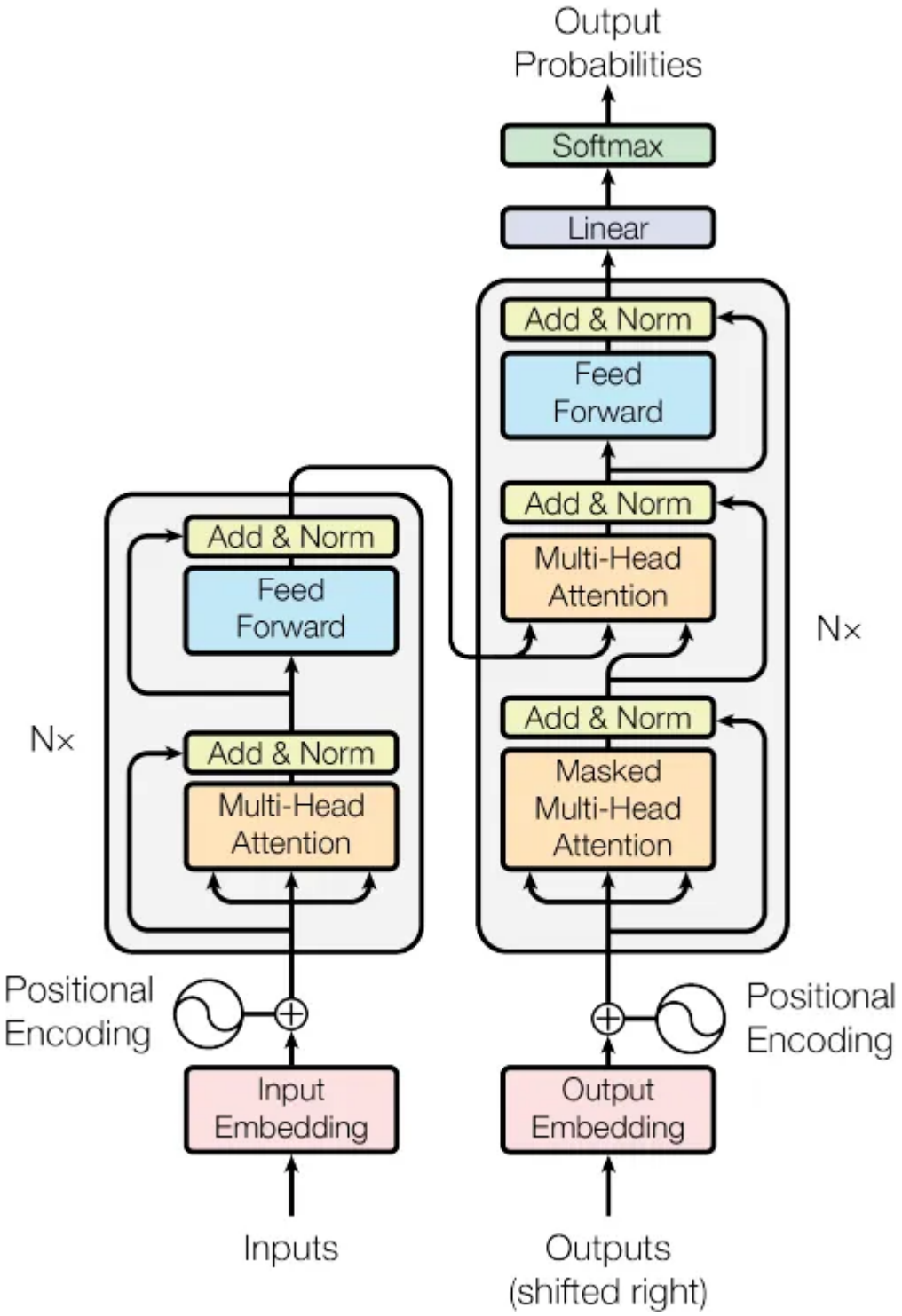

Figure 1. The Transformer Network (Source: Image from the original paper)

Figure 1 shows the Transformer network. This network has replaced RNNs as the best model for NLP and even in computer vision (*Vision Transformers*).

The network contains two parts — The encoder and the decoder.

In machine translation, the encoder is used to encode the initial sentence and the decoder is used to produce a translated sentence. The encoder of the transformer can

process the whole sentence in parallel, making it much faster and better than RNNs — which process one word of the sentence at a time.

3.1 The Encoder Blocks

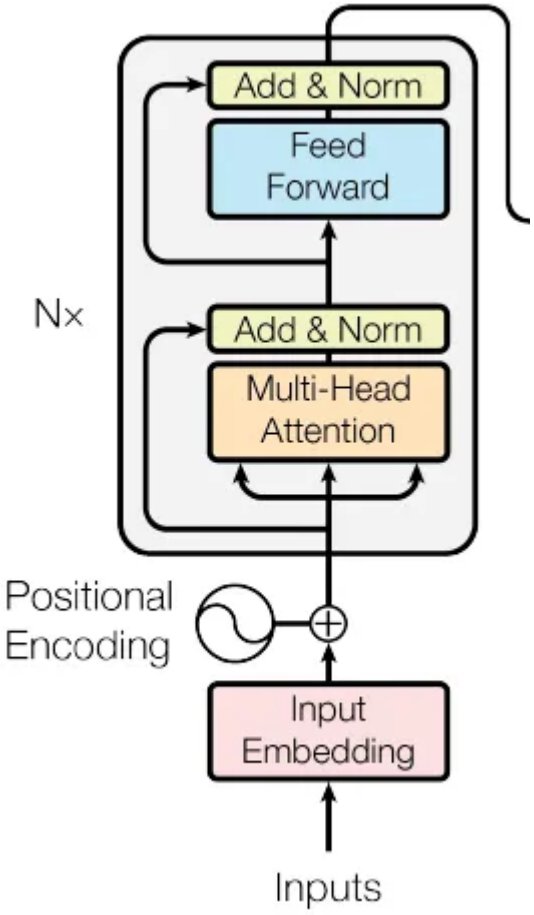

Figure 2. The Encoder part of the transformer network (Source: image from the original paper)

The encoder network begins with the inputs. Here the entire sentence is fed at once. They are then embedded at the '**input embedding**' block. Then a '**positional encoding**' is added to each word in a sentence. This encoding is essential in understanding the position of each word in a sentence. Without positional embedding, the model will treat the entire sentence as just a bag full of words without having any sequence or meaning.

In detail:

3.1.1 Input embedding — The word 'dog' in a sentence can use the embedding space to obtain a vector embedding. Embedding is just converting a word in any language to its vector representation. An example is shown in figure 3. In the embedding space, similar words have similar embeddings, for example, the word 'cat' and the word 'kitty' will fall very close in the embedding space, and the word 'cat' and 'emotion' will fall further away in the space.

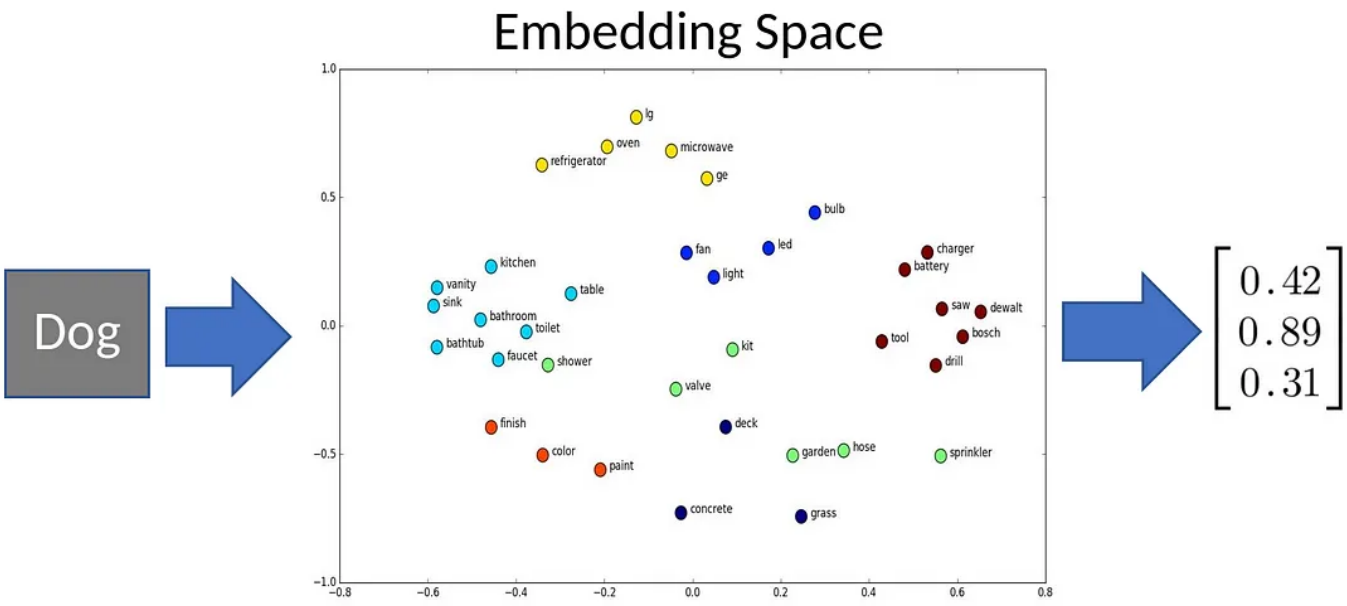

Figure 3. Input embedding (Source: image created by author)

3.1.2 Positional encoding

A word in different sentences can have different meanings. For example, the word dog in a. I own a cute dog (animal/pet — position 5) and b. What a lazy dog you are! (worthless -position 4), have different meanings. To help with that positional encodings come in. It is a vector that gives information based on the context and position of the word in a sentence.

In any sentence, the words come one after the other to have a significant meaning. If the words in the sentence are jumbled, then the sentence will have no meaning. But when a transformer loads the sentences, it does not do so in sequential order but loads them in parallel. Since the transformer architecture does not include the order of the words while loading in parallel, we must explicitly define the position of the words in a sentence. This helps the transformer to understand that one word comes after the other in a sentence. This is where positional embeddings come in handy. This is sort of

a vector encoding defining the position of a word. This positional embedding is added to the input embedding before going into the attention network. Figure 4 gives an intuitive understanding of the input embedding and the positional embedding before it is fed into the attention network.

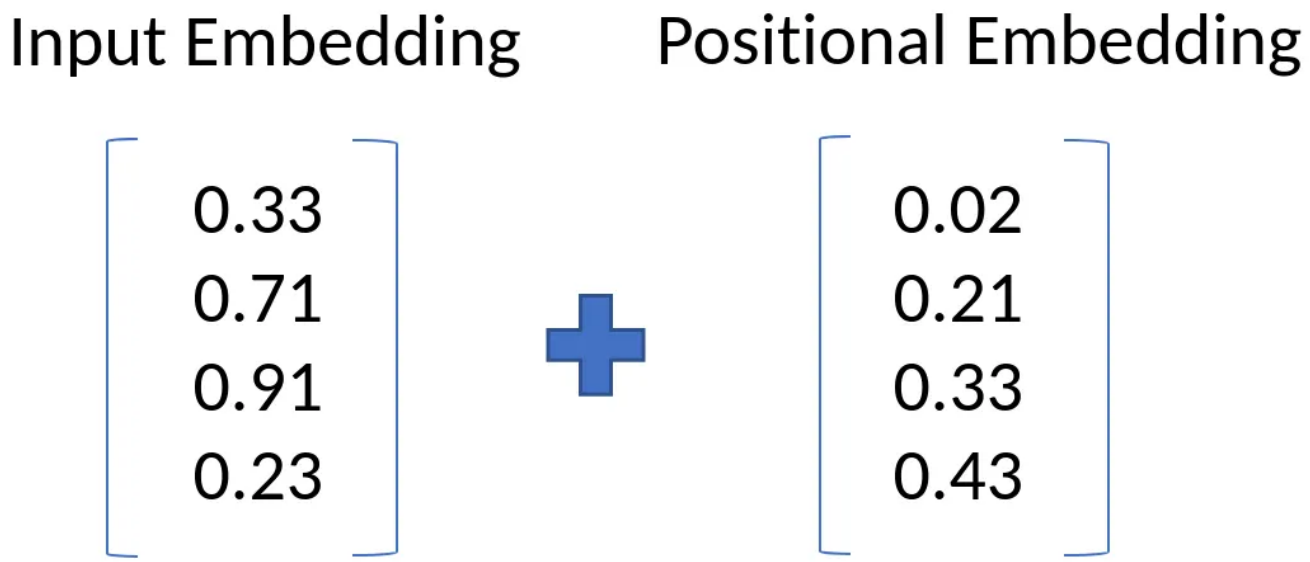

Figure 4. Intuitive understanding of positional embedding (Source: image created by author)

There are multiple ways of defining these positional embeddings. For example in the original paper — Attention is All You Need, the authors used an alternating sine and cosine function to define the embeddings, as shown in figure 5.

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

Figure 5. Positional embedding used in the original paper (Source: image from the original paper)

Though this embedding works well on text data, this embedding does not work with image data. So there can be multiple ways of embedding the position of an object (text/image), and they can be fixed or learned during training. The basic idea is that this embedding allows the transformer architecture to understand the position of the word in a sentence, and not mess up the meaning by jumbling the words.

After the word/input embedding and positional embedding is complete, the embeddings then flow into the most important part of the encoder which contains the two important blocks — a '**multi-head attention**' block and a '**feed-forward**' network.

3.1.3 Multi-Head Attention

This is the main block where the magic happens. To understand Multi-Head Attention please visit this link — 2.4 Multi-Head Attention.

As input, this block receives a vector (the sentence) that contains a sub-vector (words in a sentence). The multi-head attention then computes the attention between every position with every other position of the vector.

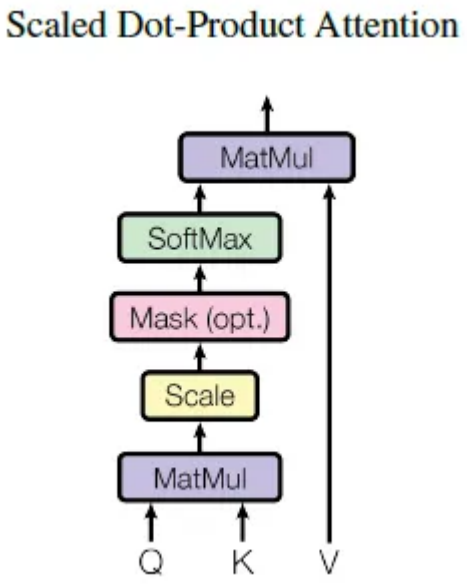

Figure 6. Scaled Dot-product Attention (Source: image from the original paper)

The figure above shows scaled dot-product attention. This is exactly the same as self-attention with an addition of two blocks (Scale and Mask). To know in detail about Sef-Attention please go to this link — 2.1 Self-Attention.

Scaled Attention, as seen in figure 6, is exactly the same, except it adds a scale after the first matrix multiplication (Matmul).

The scaling is shown below,

$$\text{Scale} = 1/\sqrt{d}$$
$$\textit{Output after Scale} = Q^T K/\sqrt{d}$$
$$\textit{where, } Q^T K \textit{ is MatMul of Query and Key}$$
$$\textit{and, } d \textit{ is the dimensionality (size) of the word embedding}$$

The output after scale goes into a Mask layer. This is an optional layer, useful for machine translation.

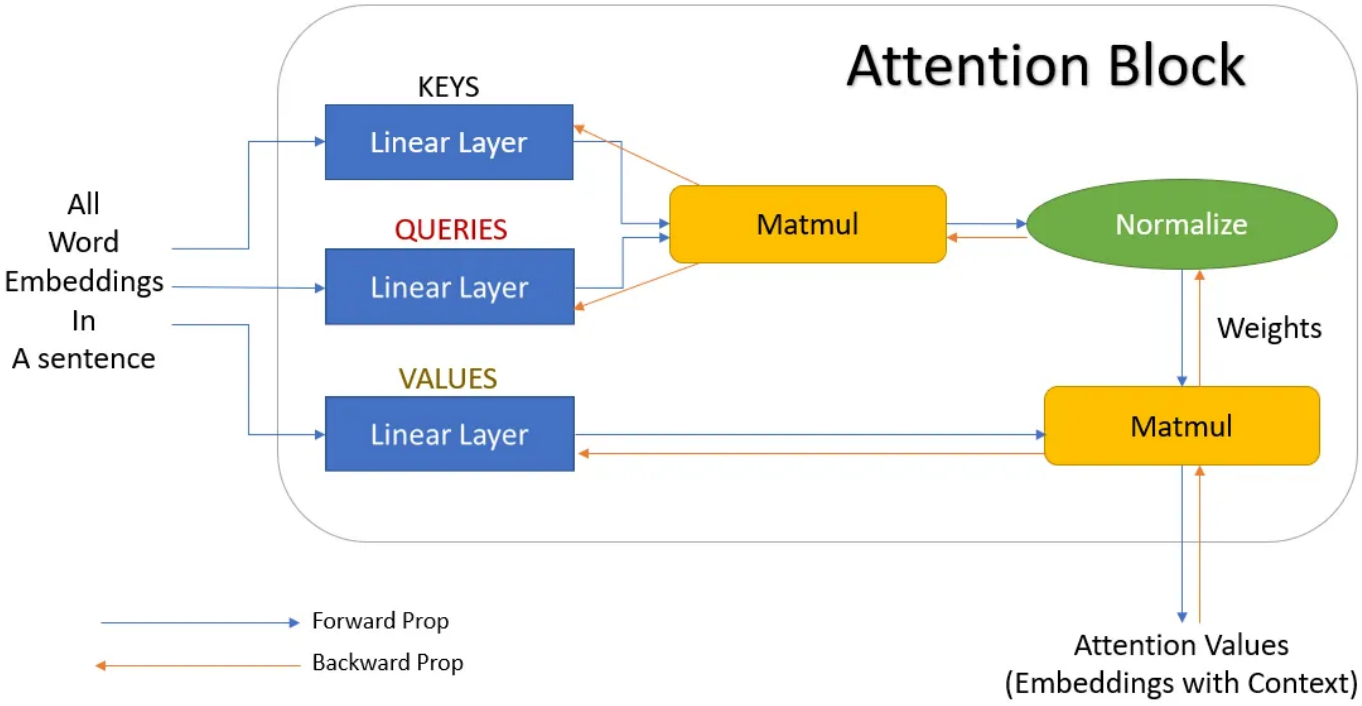

Figure 7. The Attention block (Source: image created by author)

Figure 7 shows the neural network representation of an attention block. The word embeddings are first passed into some linear layers. These linear layers do not have a 'bias' term, and hence are nothing but matrix multiplications. One of these layers is denoted as 'keys', the other as 'queries', and the last one as 'values'. If a matrix multiplication is performed between the keys and the queries and is then normalized, we get the weights. These weights are then multiplied by the values, and summed up, to get the final attention vector. This block can now be used in a neural network and is known as the Attention block. Multiple such attention blocks can be added to provide more context. And the best part is, that we can get a gradient backpropagating to update the attention block (weights of keys, queries, values).

Multi-head attention takes in multiple keys, queries, and values, feeds it through multiple scaled dot-product attention blocks, and finally concatenates the attention to give us a final output. This is shown in figure 8.

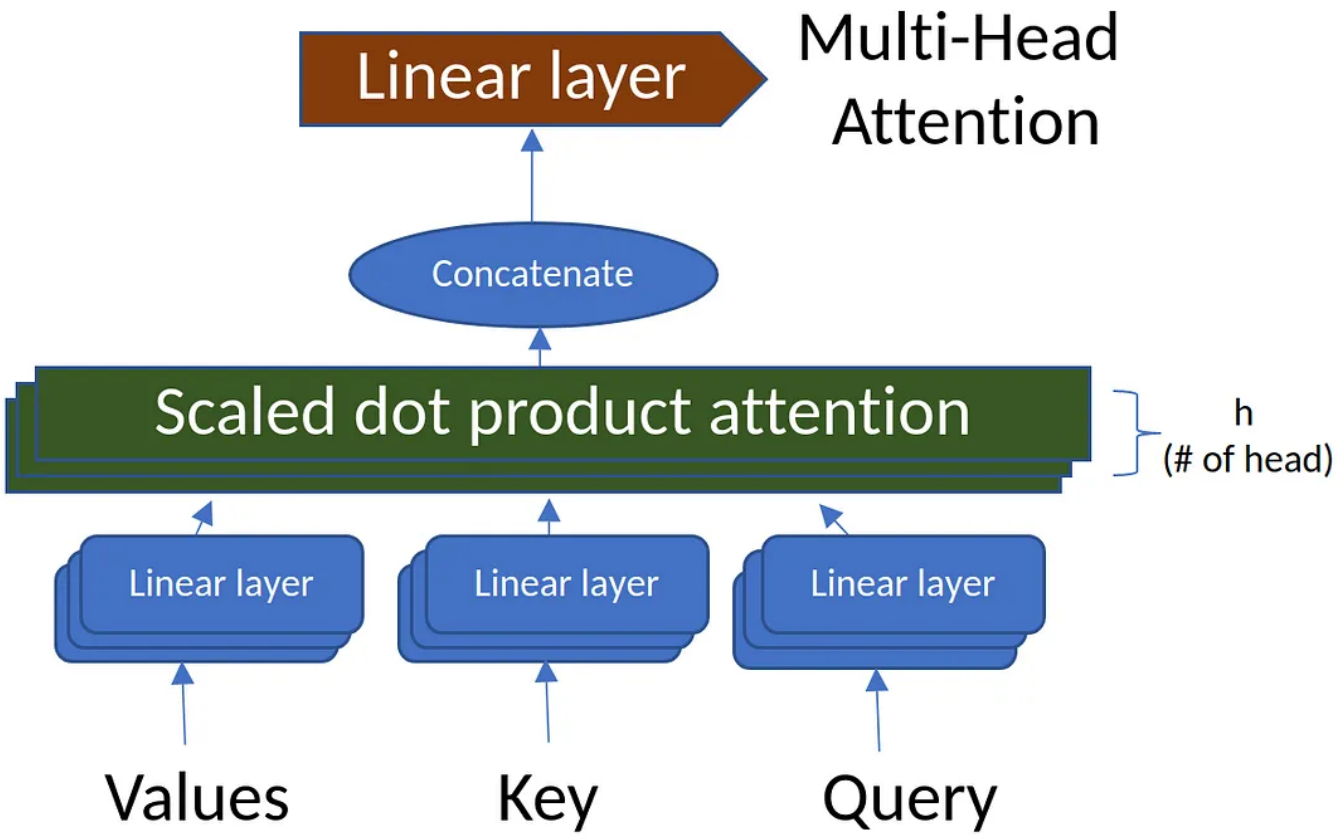

Figure 8. Multi-Head Attention (source: image created by author)

A simpler explanation: The main vector (sentence) contains sub-vectors (words) — with each word having a positional embedding. The attention computation treats each word as a '*query*' and finds some '*keys*' that correspond to some other words in the sentence, and then take a convex combination of the corresponding '*values*'. In multi-head attention, multiple values, queries, and keys are chosen which provides multiple attention (better word embedding with context). These multiple attentions are concatenated to give a final attention value (combination of context for all the words from all the multiple attentions), which works much better than using a single attention block.

In simple words, the idea of multi-head attention is to take a word embedding, combine it with some other word embedding (or multiple words) using attention (or multiple attentions) to produce a better embedding for that word (embedding with a lot more context of the surrounding words).

The idea is to compute multiple attentions per query with different weights.

3.1.4 Add & Norm and Feed-Forward

The next block is the '**Add & Norm**' which takes in a residual connection of the original word embedding, adds it to the embedding from the multi-head attention, and then normalizes it to have zero mean and variance 1.

This is fed to a '**feed forward**' block which also has an 'add & norm' block at its output.

The whole multi-head attention and feed-forward blocks are repeated n times (hyperparameters), in the encoder block.

3.2 The Decoder Blocks

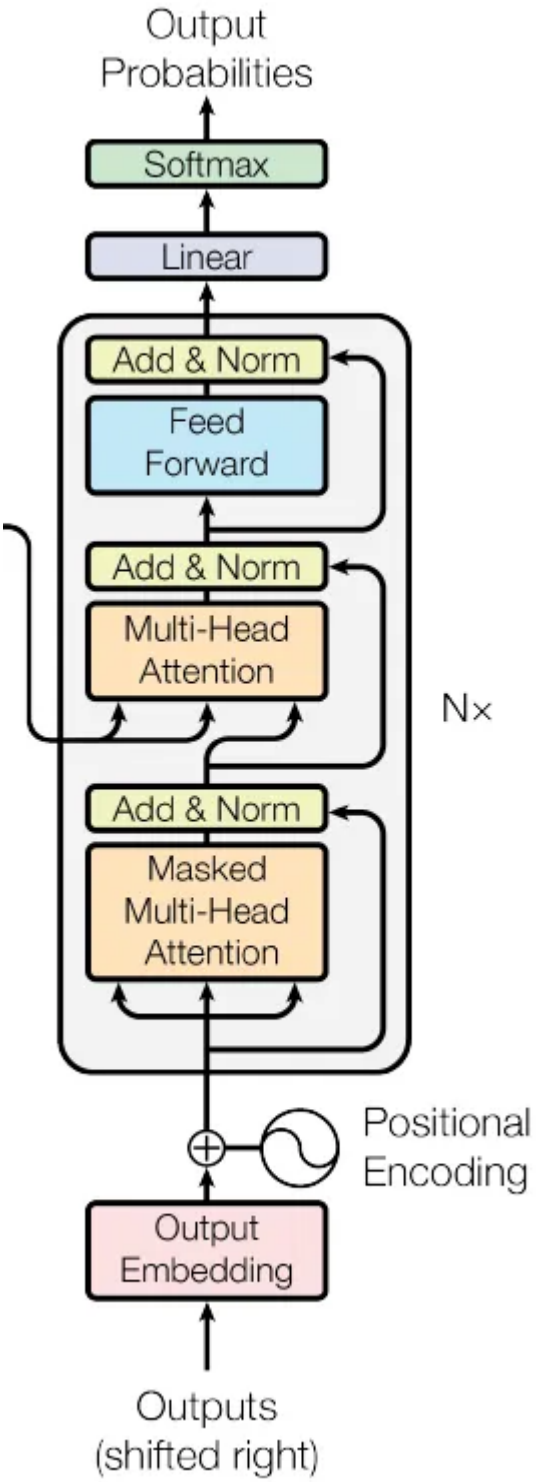

Figure 9. The Decoder part of the Transformer network (Souce: Image from the original paper)

The output of the encoder is again a sequence of embeddings, one embedding per position, where each position embedding contains not only the embedding of the original word at the position but also information about other words, that it learned using attention.

This is then fed into the decoder part of the transformer network, as seen in figure 9. The purpose of the decoder is to produce some output. In the paper, Attention is All You Need, this decoder was used for sentence translation (say from English to French). So the encoder will take in the English sentence, and the decoder will translate it to French. In other applications, the decoder part of the network is not necessary, and hence I will not elaborate on it too much.

Steps in decoder block —

1. In sentence translation, the decoder block takes in the French sentence (for English to French translation). Like the encoder, here we add a word embedding and a positional embedding and feed it to the multi-head attention block.

2. The self-attention block will generate an attention vector for each word in the French sentence, to show how much one word is related to the other in the sentence.

3. This attention vector from the French sentence is then compared with the attention vectors from the English sentence. This is the part where the English to French word mappings happen.

4. In the final layers, the decoder predicts the translation of the English word to the best probable French word.

5. The whole process is repeated multiple times to get a translation of the entire text data.

The blocks being used for each of the above steps are shown in figure 10.

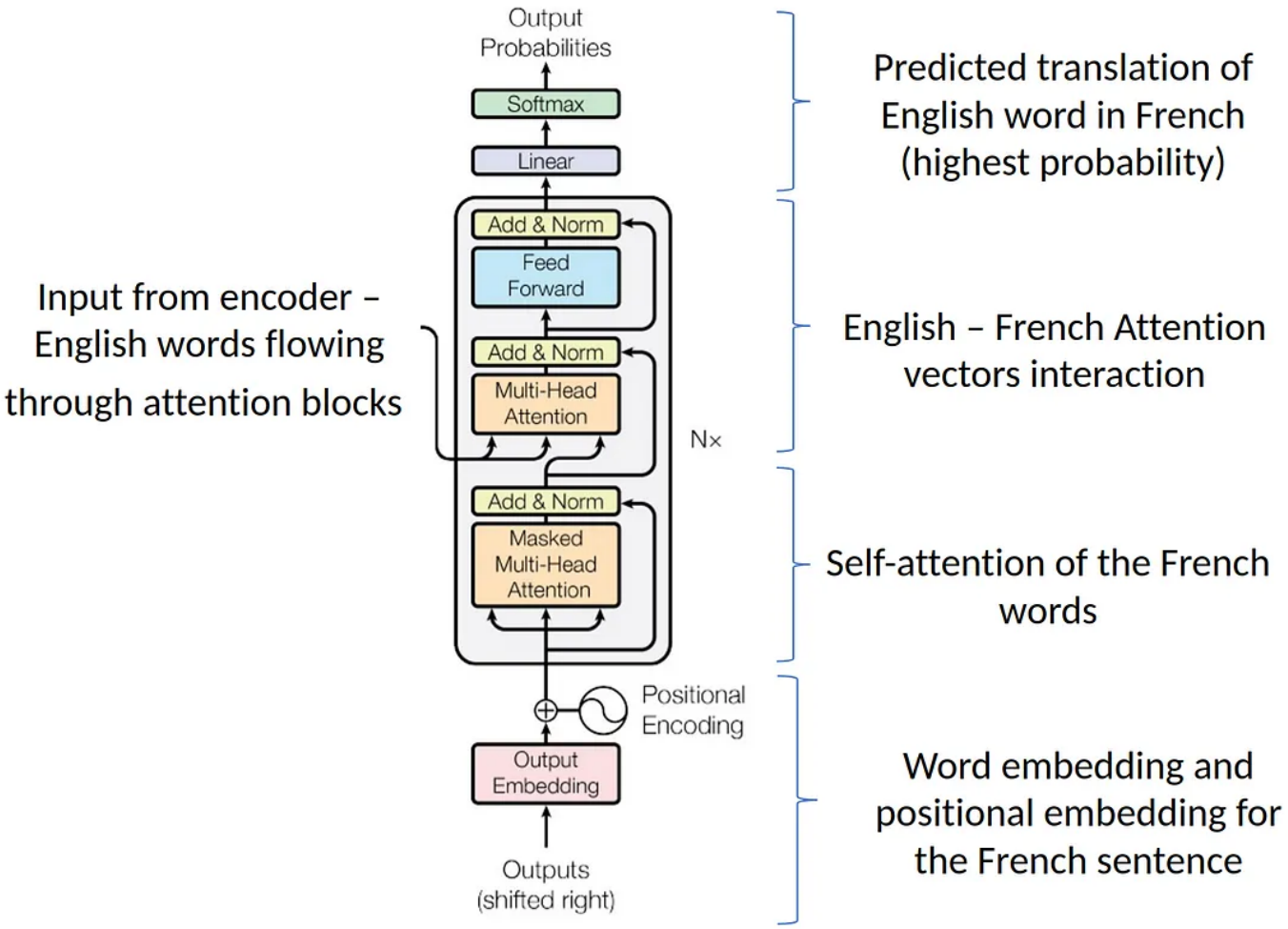

Figure 10. The function of different decoder blocks in sentence translation (Source: image created by author)

There is one block that is new in the decoder — the Masked Multi-head Attention. All the other blocks, we have already seen previously in the encoder.

3.2.1 Masked Multi-head Attention

This is a multi-head attention block where some values are masked. The probabilities of the masked values are nullified or not selected.

For example, while decoding, the output value should only depend on previous outputs and not future outputs. Then we mask the future outputs.

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

$$\text{Masked Attention}(Q,K,V) = \text{SoftMax}\left(\frac{Q^T K + M}{\sqrt{d_k}}\right) V$$

where M is a mask matrix of 0's and -∞'s

3.3 Results and Conclusion

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Figure 11. Results (source: image from the original paper)

In the paper, a comparison was done for language translation between English to German and English to French, with other state-of-the-art language models. BLEU is a measure used in language translation comparison. From figure 11, we see that the big Transformer model obtains a higher BLEU score for both translation tasks. What they also improve on significantly is the training cost.

In summary, a Transformer model can reduce computation costs while still obtaining state-of-the-art results.

In this part, I have explained the encoder and decoder blocks of a Transformer network, and how each block is used in language translation. In the next and final part (Part 3), I will discuss some important Transformer networks that have become extremely famous in recent times, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (General Purpose Transformer).

References:

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

Towards Data Science | Deep Learning | Machine Learning | Neural Networks | Naturallanguageprocessing

Follow

Written by Arjun Sarkar