Open in app ↗ Sign up Sign In

HANDS-ON TUTORIALS, INTUITIVE TRANSFORMERS SERIES NLP

Transformers Explained Visually — Not Just How, but Why They Work So Well

A Gentle Guide to how the Attention Score calculations capture relationships between words in a sequence, in Plain English.

Ketan Doshi · Follow
Published in Towards Data Science
10 min read · Jun 3, 2021

Listen Share

Photo by Olav Ahrens Røtne on Unsplash

Transformers have taken the world of NLP by storm in the last few years. Now they are being used with success in applications beyond NLP as well.

The Transformer gets its powers because of the Attention module. And this happens because it captures the relationships between each word in a sequence with every other word.

But the all-important question is *how* exactly does it do that?

In this article, we will attempt to answer that question, and understand *why* it performs the calculations that it does.

I have a few more articles in my series on Transformers. In those articles, we learned about the Transformer architecture and walked through their operation during training and inference, step-by-step. We also explored under the hood and understood exactly how they work in detail.

Our goal is to understand not just how something works but why it works that way.

1. Overview of functionality (*How Transformers are used, and why they are better than RNNs. Components of the architecture, and behavior during Training and Inference*)
2. How it works (*Internal operation end-to-end. How data flows and what computations are performed, including matrix representations*)
3. Multi-head Attention (*Inner workings of the Attention module throughout the Transformer*)

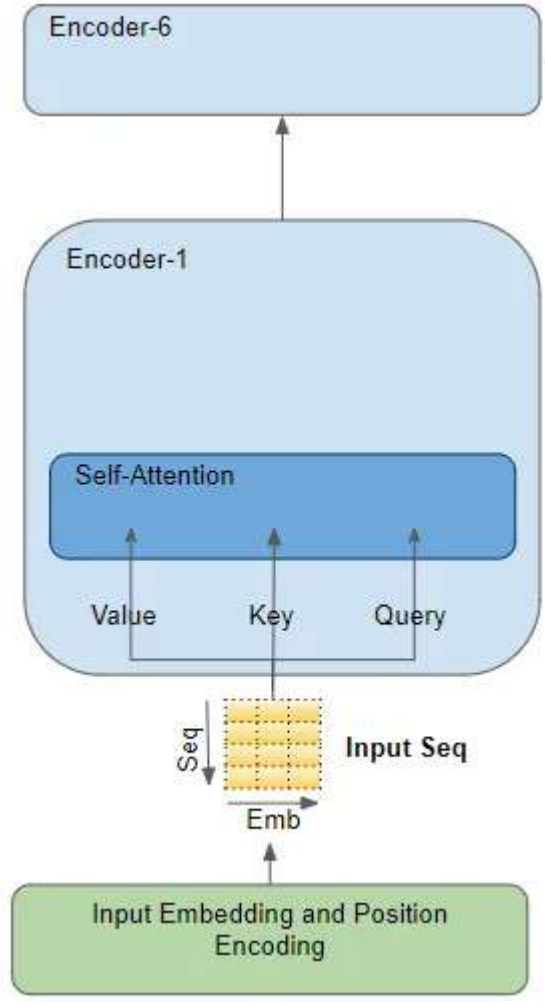
And if you're interested in NLP applications in general, I have some other articles you might like.

1. Beam Search *(Algorithm commonly used by Speech-to-Text and NLP applications to enhance predictions)*
2. Bleu Score (*Bleu Score and Word Error Rate are two essential metrics for NLP models*)

To understand what makes the Transformer tick, we must focus on Attention. Let's start with the input that goes into it, and then look at how it processes that input.

How does the input sequence reach the Attention module

The Attention module is present in every Encoder in the Encoder stack, as well as every Decoder in the Decoder stack. We'll zoom in on the Encoder attention first.



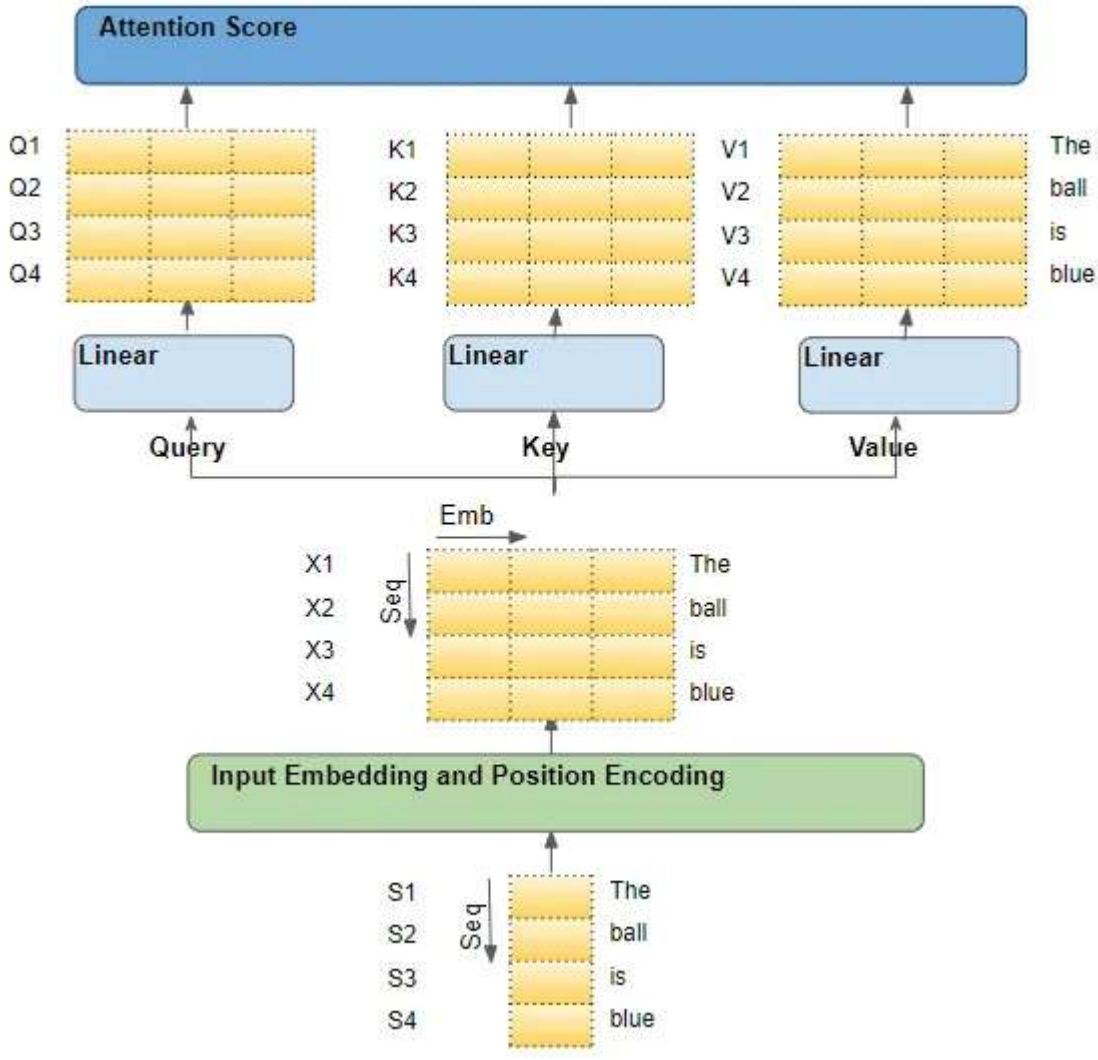
Attention in the Encoder (Image by Author)

As an example, let's say that we're working on an English-to-Spanish translation problem, where one sample source sequence is "The ball is blue". The target sequence is "La bola es azul".

The source sequence is first passed through the Embedding and Position Encoding layer, which generates embedding vectors for each word in the sequence. The embedding is passed to the Encoder where it first reaches the Attention module.

Within Attention, the embedded sequence is passed through three Linear layers which produce three separate matrices — known as the Query, Key, and Value. These are the three matrices that are used to compute the Attention Score.

The important thing to keep in mind is that each 'row' of these matrices corresponds to one word in the source sequence.



The flow of the source sequence (Image by Author)

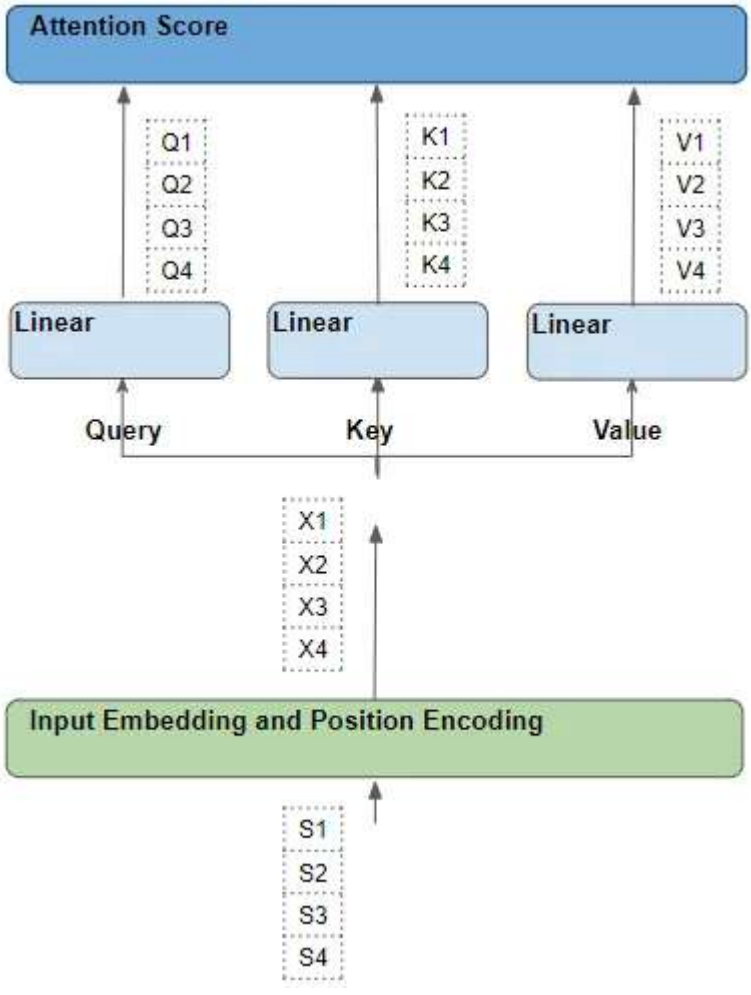
Each input row is a word from the sequence

The way we will understand what is going on with Attention, is by starting with the individual words in the source sequence, and then following their path as they make their way through the Transformer. In particular, we want to focus on what goes on inside the Attention Module.

That will help us clearly see how each word in the source and target sequences interacts with other words in the source and target sequences.

So as we go through this explanation, concentrate on what operations are being performed on each word, and how each vector maps to the original input word. We do not need to worry about many of the other details such as matrix shapes, specifics of the arithmetic calculations, multiple attention heads, and so on if they are not directly relevant to where each word is going.

So to simplify the explanation and the visualization, let's ignore the embedding dimension and track just the rows for each word.



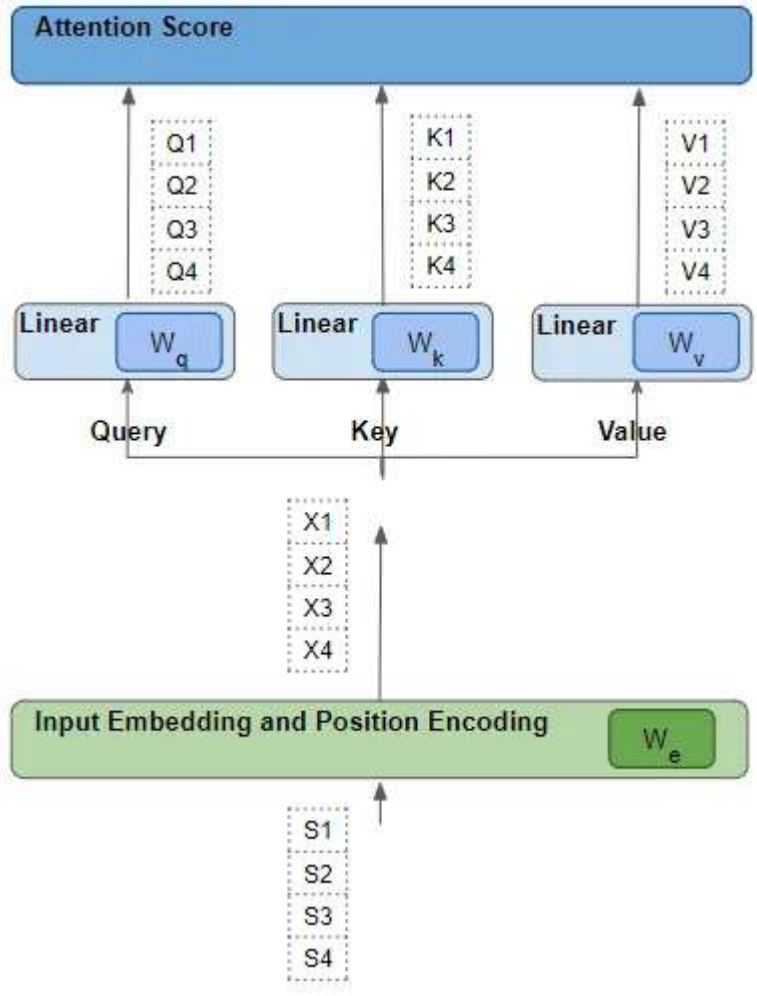
The flow of each word in the source sequence (Image by Author)

Each word goes through a series of learnable transformations

Each such row has been generated from its corresponding source word by a series of transformations — embedding, position

encoding, and linear layer.

All of those transformations are trainable operations. This means that the weights used in those operations are not pre-decided but are learned by the model in such a way that they produce the desired output predictions.

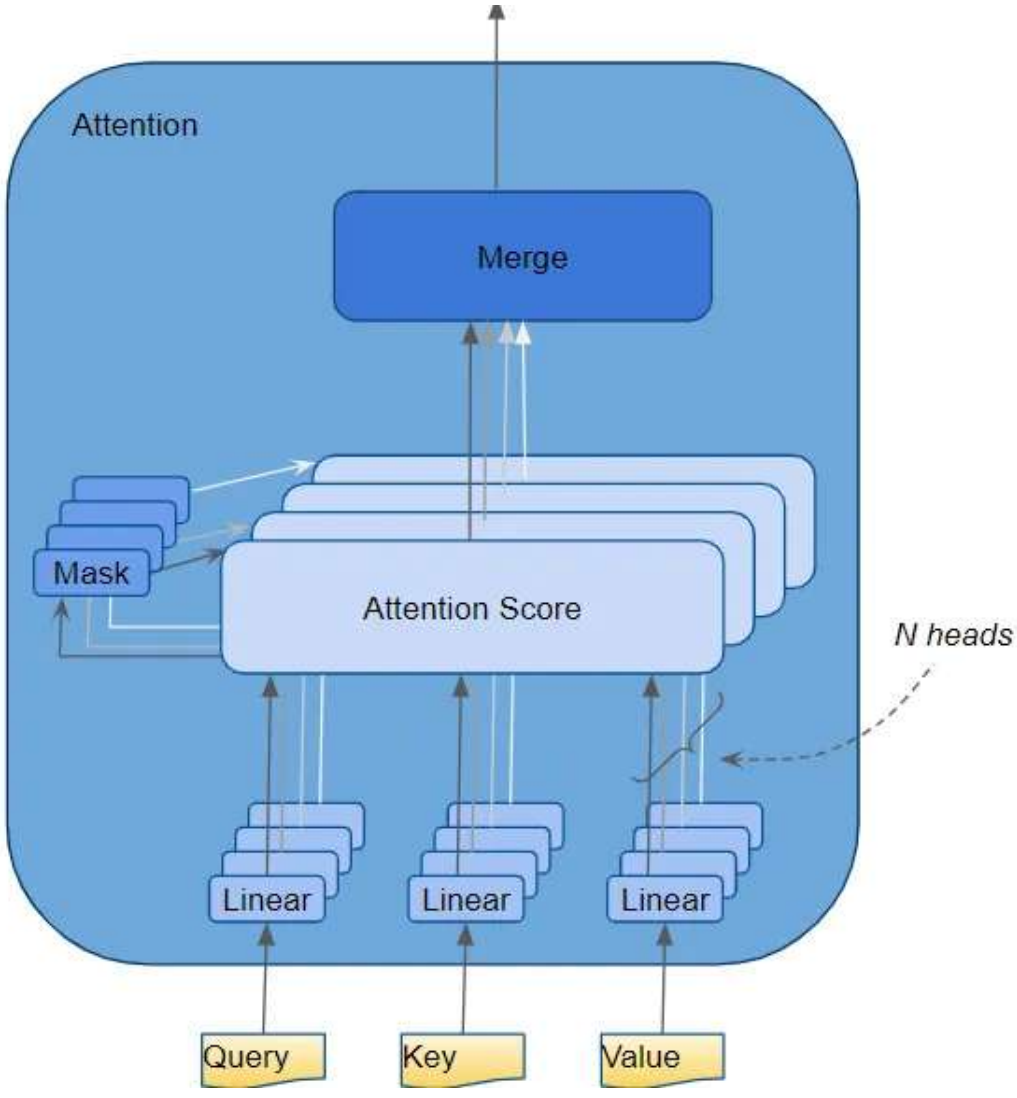


Linear and Embedding weights are learned (Image by Author)

The key question is, how does the Transformer figure out what set of weights will give it the best results? Keep this point in the back of your mind as we will come back to it a little later.

Attention Score — Dot Product between Query and Key words

Attention performs several steps, but here, we will focus only on the Linear layer and the Attention Score.



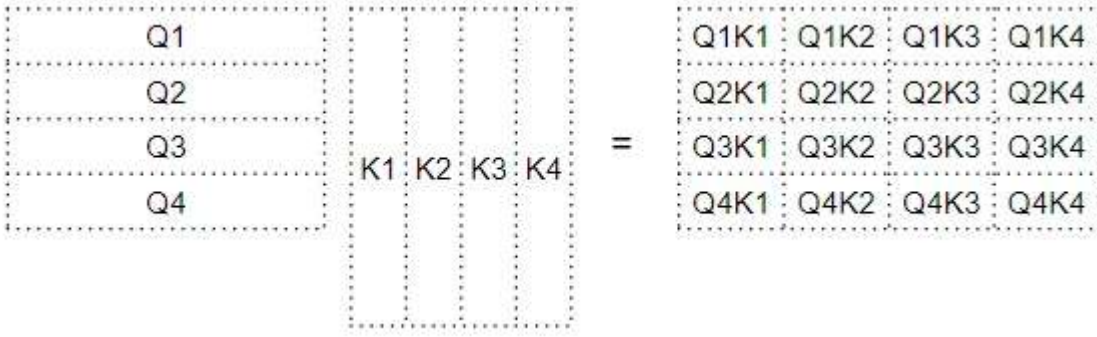
Multi-head attention (Image by Author)

$$Z = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Attention Score calculation (Image by Author)

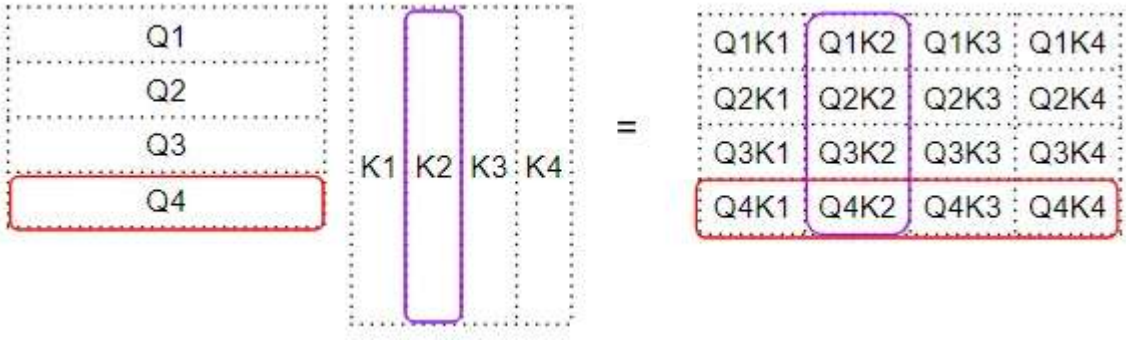
As we can see from the formula, the first step within Attention is to do a matrix multiply (ie. dot product) between the Query (Q) matrix and a transpose of the Key (K) matrix. Watch what happens to each word.

We produce an intermediate matrix (let's call it a 'factor' matrix) where each cell is a matrix multiplication between two words.



Dot Product between Query and Key matrices (Image by Author)

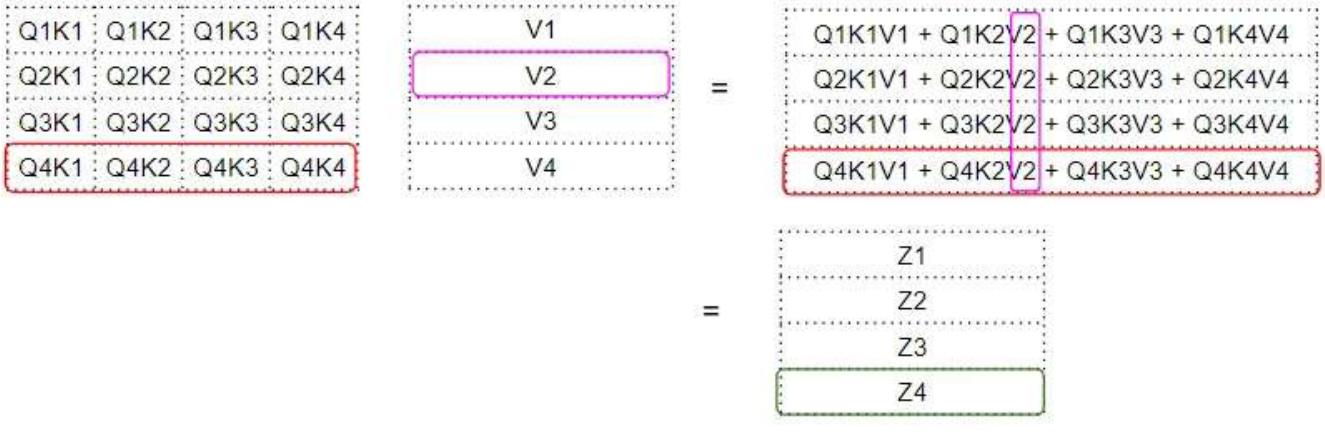
For instance, each column in the fourth row corresponds to a dot product between the fourth Query word with every Key word.



Dot Product between Query and Key matrices (Image by Author)

Attention Score — Dot Product between Query-Key and Value words

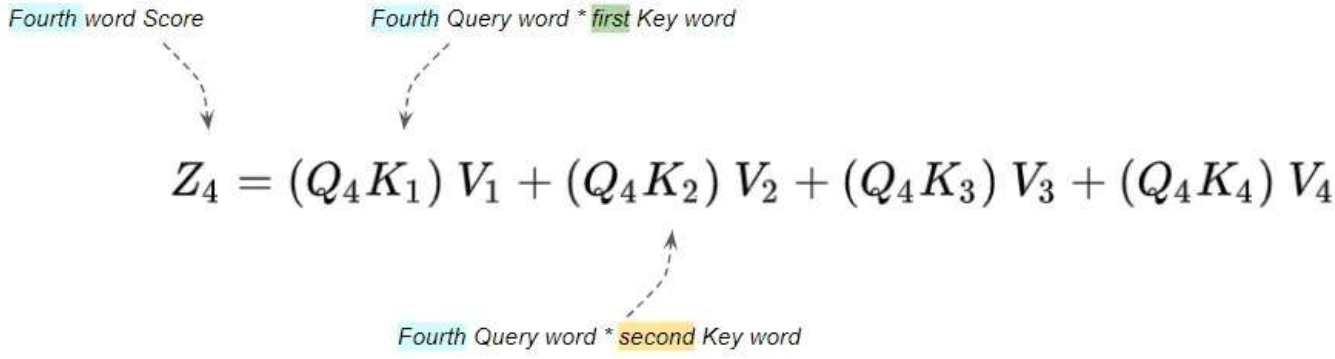
The next step is a matrix multiply between this intermediate ‘factor’ matrix and the Value (V) matrix, to produce the attention score that is output by the attention module. Here we can see that the fourth row corresponds to the fourth Query word matrix multiplied with all other Key and Value words.



Dot Product between Query-Key and Value matrices (Image by Author)

This produces the Attention Score vector (Z) that is output by the Attention Module.

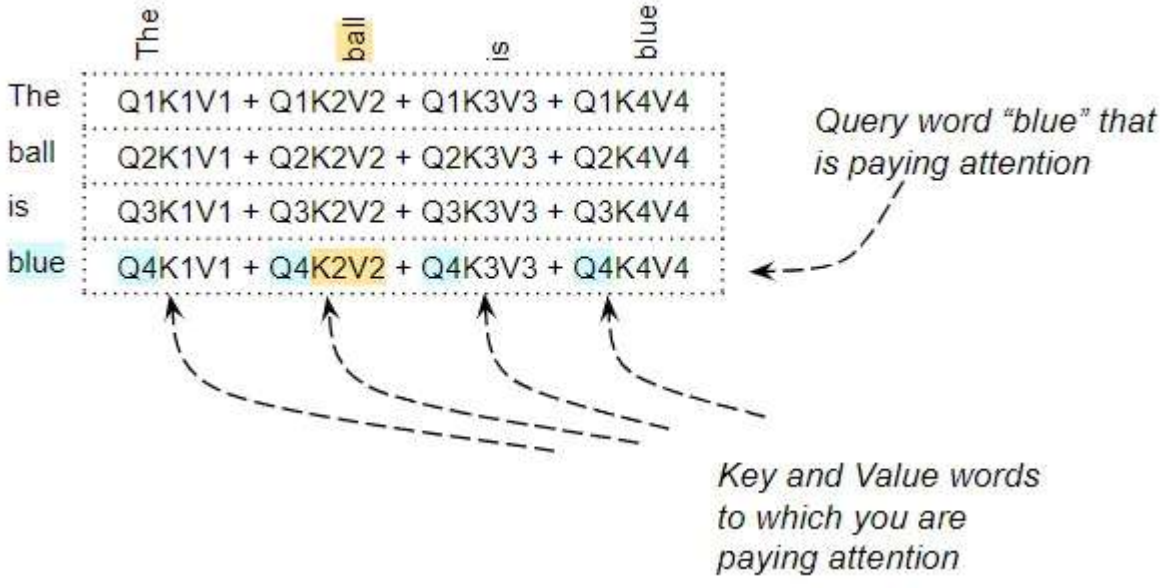
The way to think about the output score is that, for each word, it is the encoded value of every word from the "Value" matrix, weighted by the "factor" matrix. The factor matrix is the dot product of the Query value for that specific word with the Key value of all words.



Attention Score is a weighted sum of the Value words (Image by Author)

What is the role of the Query, Key, and Value words?

The Query word can be interpreted as the word *for which* we are calculating Attention. The Key and Value word is the word *to which* we are paying attention ie. how relevant is that word to the Query word.



Attention Score for the word "blue" pays attention to every other word (Image by Author)

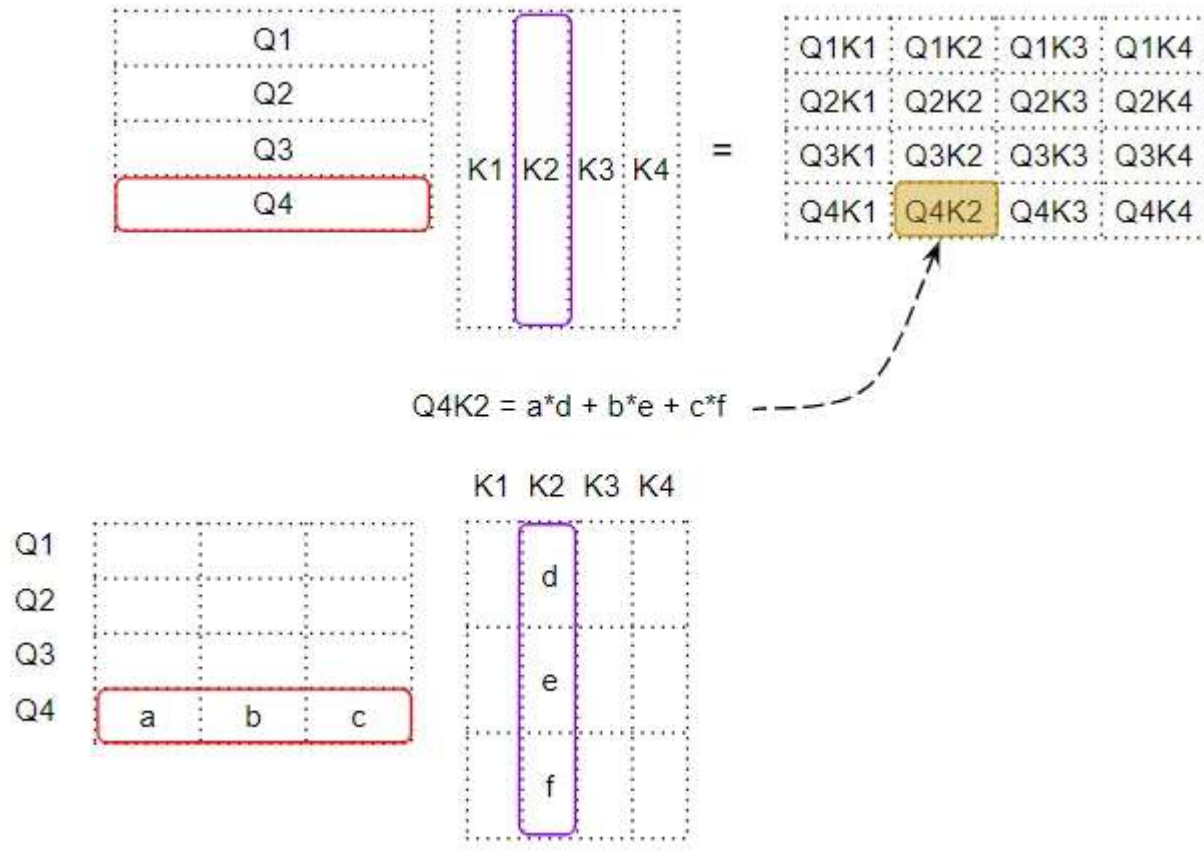
For example, for the sentence, “The ball is blue”, the row for the word “blue” will contain the attention scores for “blue” with every other word. Here, “blue” is the Query word, and the other words are the “Key/Value”.

There are other operations being performed such as a division and a softmax, but we can ignore them in this article. They just change the numeric values in the matrices but don’t affect the position of each word row in the matrix. Nor do they involve any inter-word interactions.

Dot Product tells us the similarity between words

So we have seen that the Attention Score is capturing some interaction between a particular word, and every other word in the sentence, by doing a dot product, and then adding them up. But how does the matrix multiply help the Transformer determine the relevance between two words?

To understand this, remember that the Query, Key, and Value rows are actually vectors with an Embedding dimension. Let’s zoom in on how the matrix multiplication between those vectors is calculated.



Each cell is a dot product between two word vectors (Image by Author)

When we do a dot product between two vectors, we multiply pairs of numbers and then sum them up.

- If the two paired numbers (eg. 'a' and 'd' above) are both positive or both negative, then the product will be positive. The product will increase the final summation.
- If one number is positive and the other negative, then the product will be negative. The product will reduce the final summation.
- If the product is positive, the larger the two numbers, the more they contribute to the final summation.

This means that if the signs of the corresponding numbers in the two vectors are aligned, the final sum will be larger.

How does the Transformer learn the relevance between words?

This notion of the Dot Product applies to the attention score as well. If the vectors for two words are more aligned, the attention score will be higher.

So what is the behavior we want for the Transformer?

We want the attention score to be high for two words that are relevant to each other in the sentence. And we want the score to be low for two words that are unrelated to one another.

For example, for the sentence, “The black cat drank the milk”, the word “milk” is very relevant to “drank”, perhaps slightly less relevant to “cat”, and irrelevant to “black”. We want “milk” and “drank” to produce a high attention score, for “milk” and “cat” to produce a slightly lower score, and for “milk” and “black”, to produce a negligible score.

This is the output we want the model to learn to produce.

For this to happen, the word vectors for “milk” and “drank” must be aligned. The vectors for “milk” and “cat” will diverge somewhat. And they will be quite different for “milk” and “black”.

Let’s go back to the point we had kept at the back of our minds — how does the Transformer figure out what set of weights will give it the best results?

The word vectors are generated based on the word embeddings and the weights of the Linear layers. Therefore the Transformer can learn those embeddings, Linear weights, and so on to produce the word vectors as required above.

In other words, it will learn those embeddings and weights in such a way that if two words in a sentence are relevant to each other, then their word vectors will be aligned. And hence produce a higher attention score. For words that are not relevant to each other, the word vectors will not be aligned and will produce a lower attention score.

Therefore the embeddings for “milk” and “drank” will be very aligned and produce a high attention score. They will diverge somewhat for “milk” and “cat” to produce a slightly lower score

and will be quite different for “milk” and “black”, to produce a low score.

This then is the principle behind the Attention module.

Summarizing — What makes the Transformer tick?

The dot product between the Query and Key computes the relevance between each pair of words. This relevance is then used as a “factor” to compute a weighted sum of all the Value words. That weighted sum is output as the Attention Score.

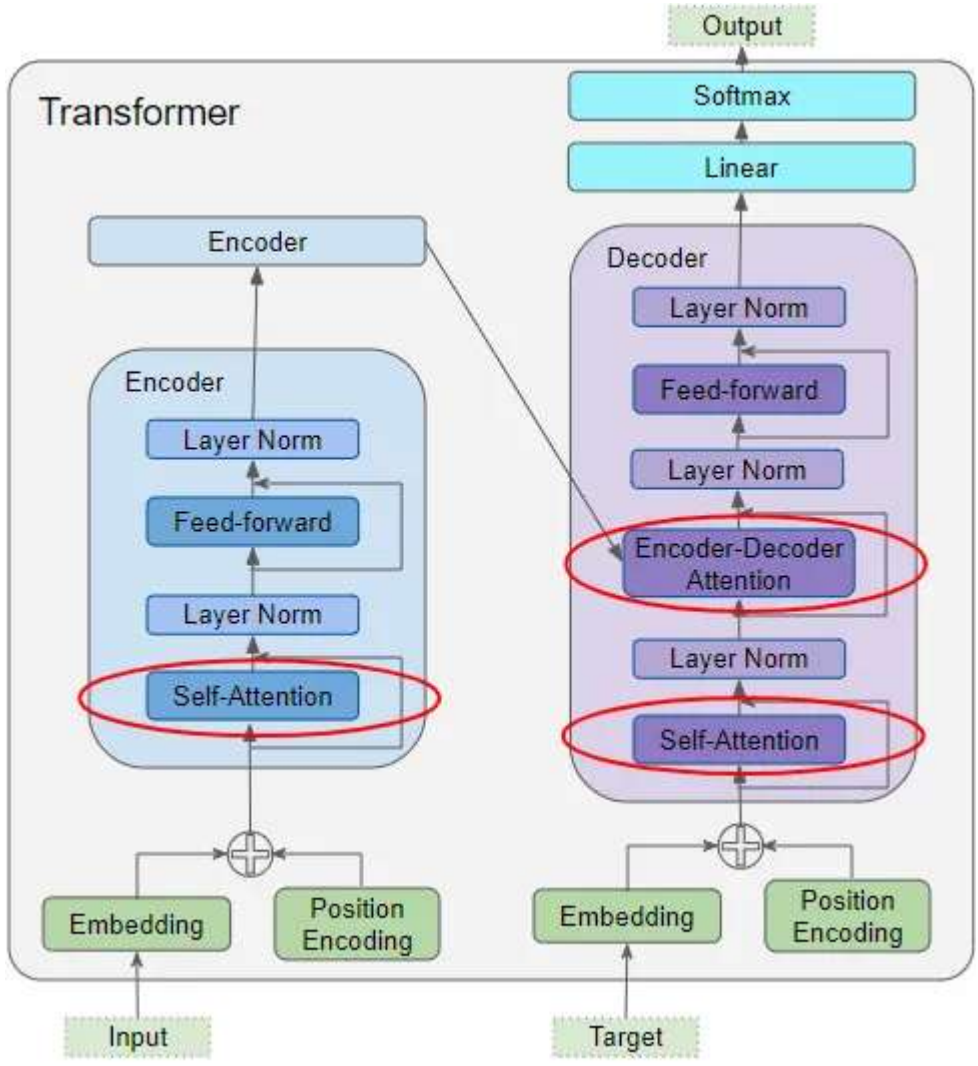
The Transformer learns embeddings etc, in such a way that words that are relevant to one another are more aligned.

This is one reason for introducing the three Linear layers and making three versions of the input sequence, for the Query, Key, and Value. That gives the Attention module some more parameters that it is able to learn to tune the creation of the word vectors.

Encoder Self-Attention in the Transformer

Attention is used in the Transformer in three places:

- Self-attention in the Encoder — the source sequence pays attention to itself
- Self-attention in the Decoder — the target sequence pays attention to itself
- Encoder-Decoder-attention in the Decoder — the target sequence pays attention to the source sequence

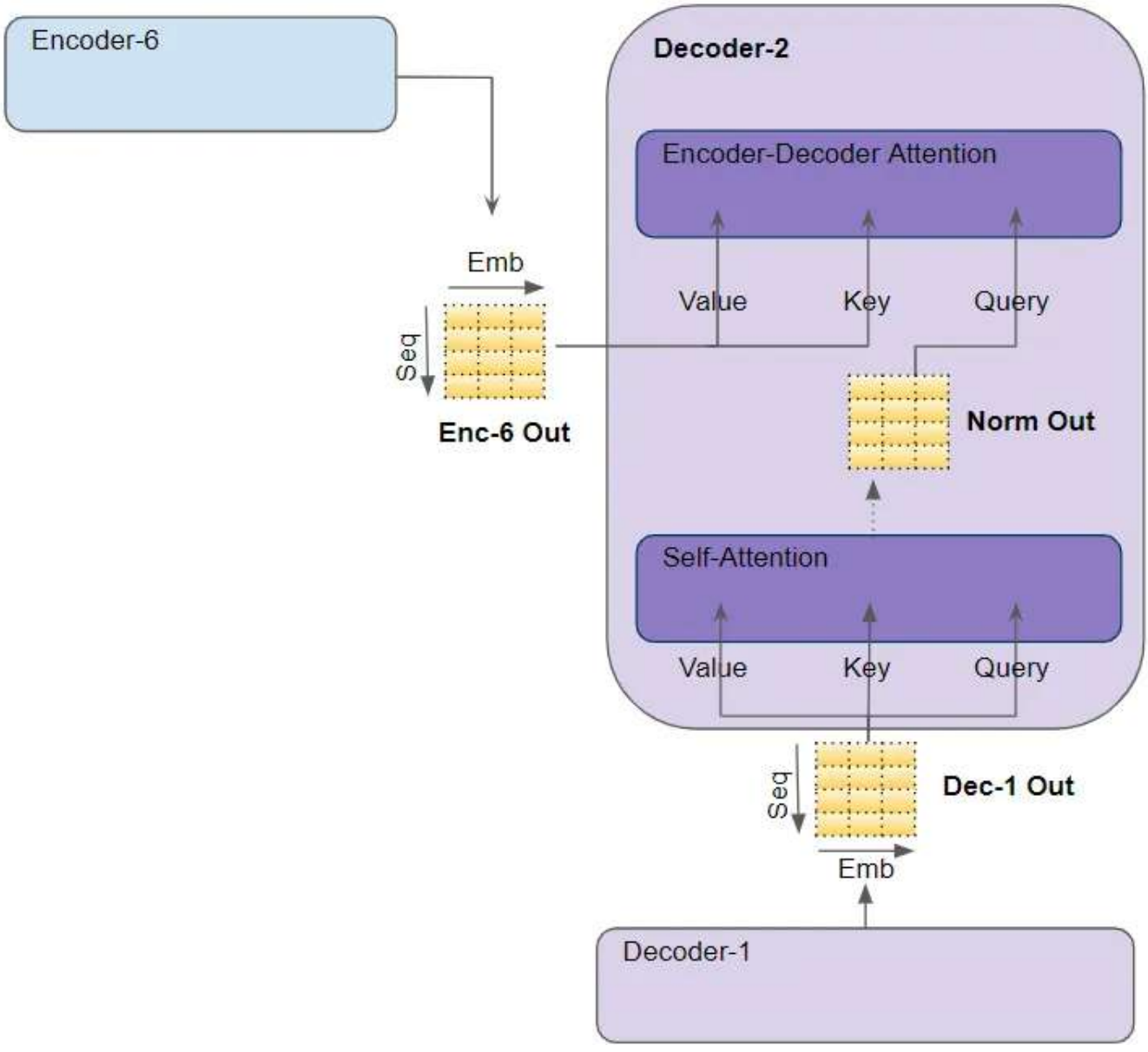


Attention in the Transformer (Image by Author)

In the Encoder Self Attention, we compute the relevance of each word in the source sentence to each other word in the source sentence. This happens in all the Encoders in the stack.

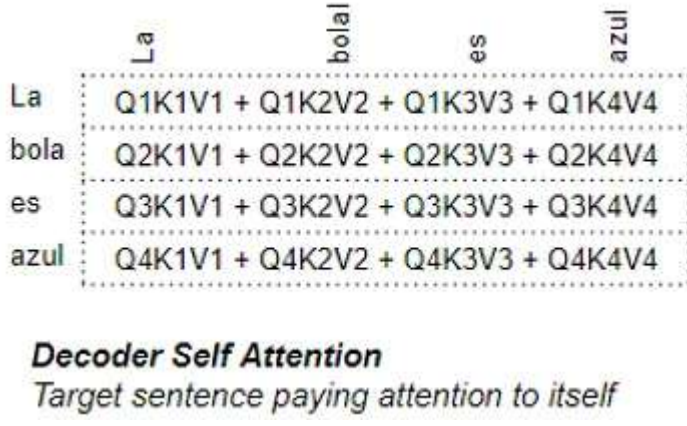
Decoder Self-Attention in the Transformer

Most of what we've just seen in the Encoder Self Attention applies to Attention in the Decoder as well, with a few small but significant differences.



Attention in the Decoder (Image by Author)

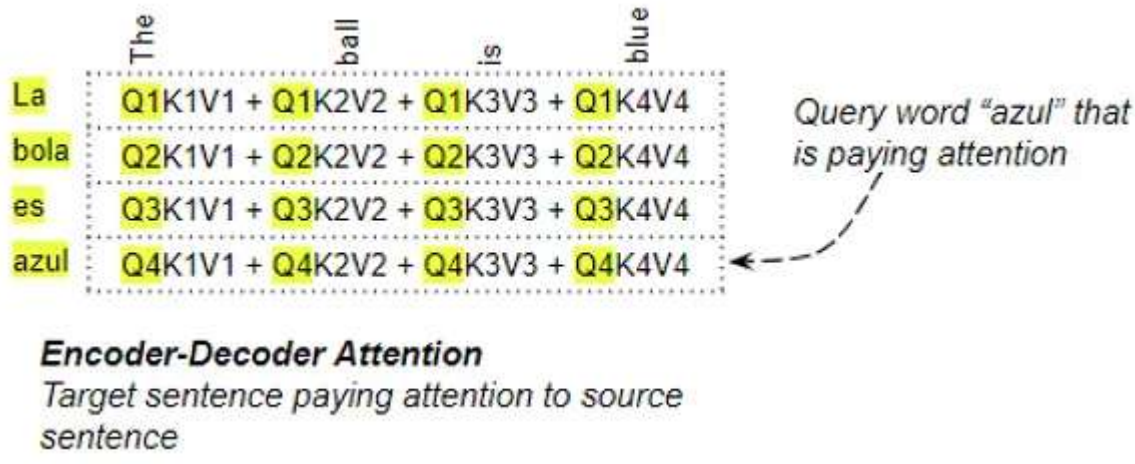
In the Decoder Self Attention, we compute the relevance of each word in the target sentence to each other word in the target sentence.



Decoder Self Attention (Image by Author)

Encoder-Decoder Attention in the Transformer

In the Encoder-Decoder Attention, the Query is obtained from the target sentence and the Key/Value from the source sentence. Thus it computes the relevance of each word in the target sentence to each word in the source sentence.



Encoder-Decoder Attention (Image by Author)

Conclusion

Hopefully, this gives you a good sense of the elegance of the Transformer design. Do also read the other Transformer articles in my series to get an in-depth understanding of why the Transformer has now become the architecture of choice for so many deep learning applications.

And finally, if you liked this article, you might also enjoy my other series on Audio Deep Learning, Geolocation Machine Learning, and Batch Norm.

Audio Deep Learning Made Simple (Part 1): State-of-the-Art Techniques

A Gentle Guide to the world of disruptive deep learning audio applications and architectures. And why we all need to...

towardsdatascience.com

Leveraging Geolocation Data for Machine Learning: Essential Techniques

A Gentle Guide to Feature Engineering and Visualization with