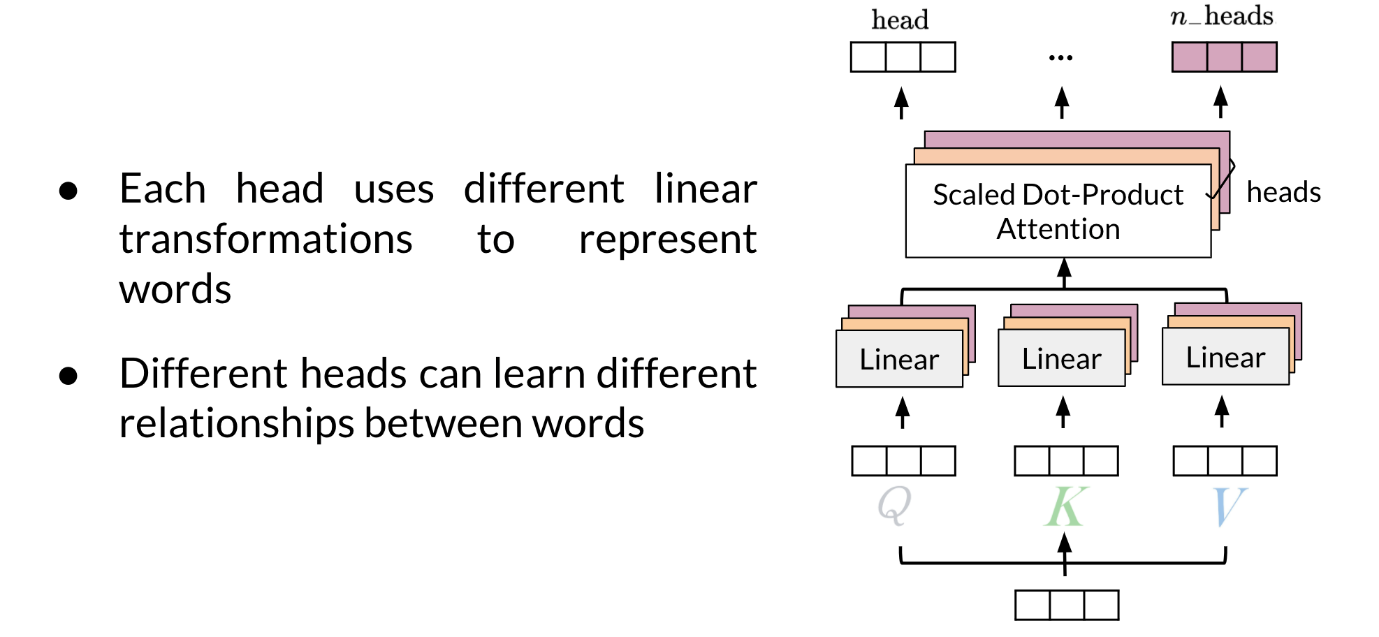
Multi-head Attention

In this reading, I will summarize the intuition behind multi-head attention and scaled dot product attention.



Given a word, you take its embedding then you multiply it by the Q, K, V matrix to get the corresponding queries, keys and values. When you use multi-head attention, a head can learn different relationships between words from another head.

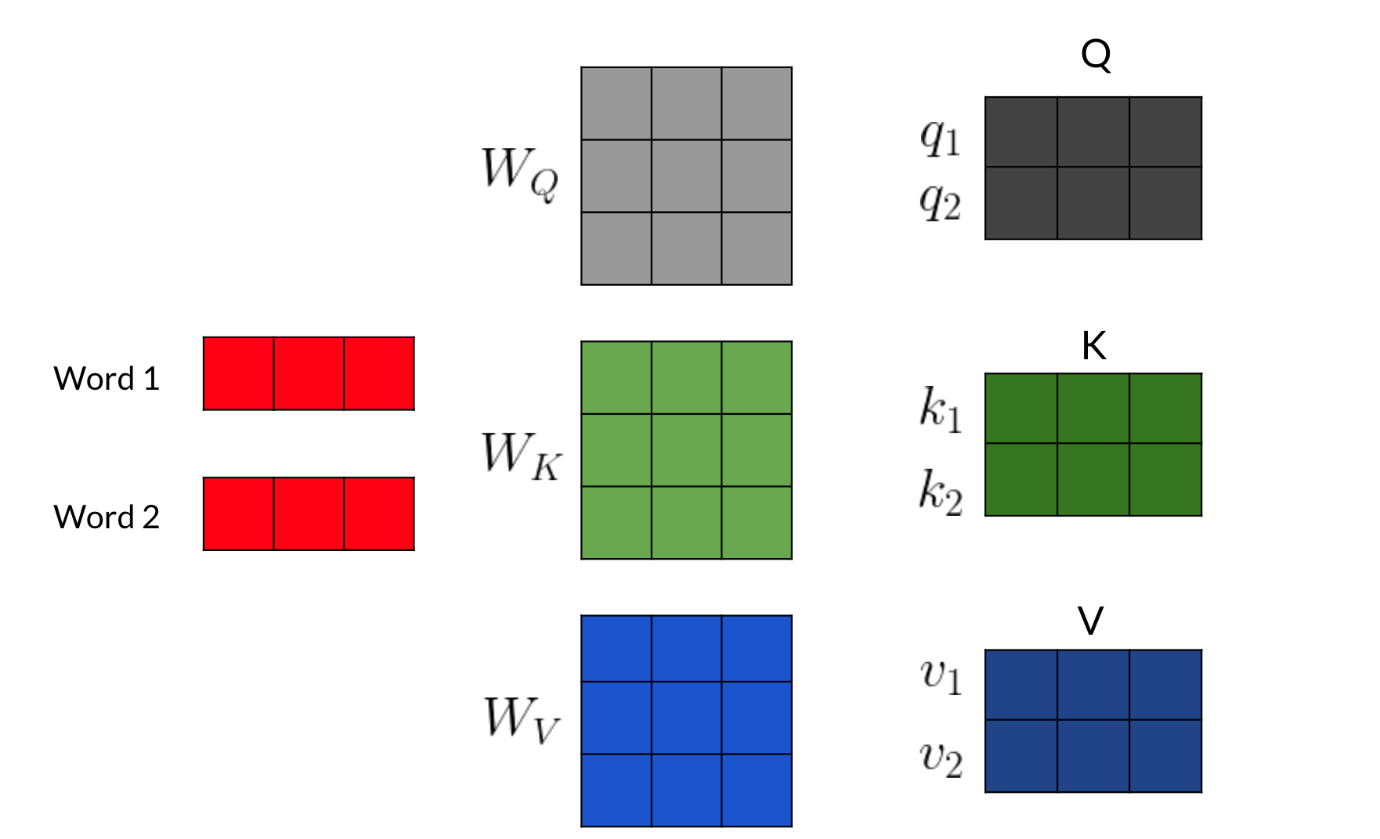
Here's one way to look at it:

- First, imagine that you have an embedding for a word. You multiply that embedding with Q to get q\_1*q*1​, K to get k\_1*k*1​, and V to get v\_1*v*1​.

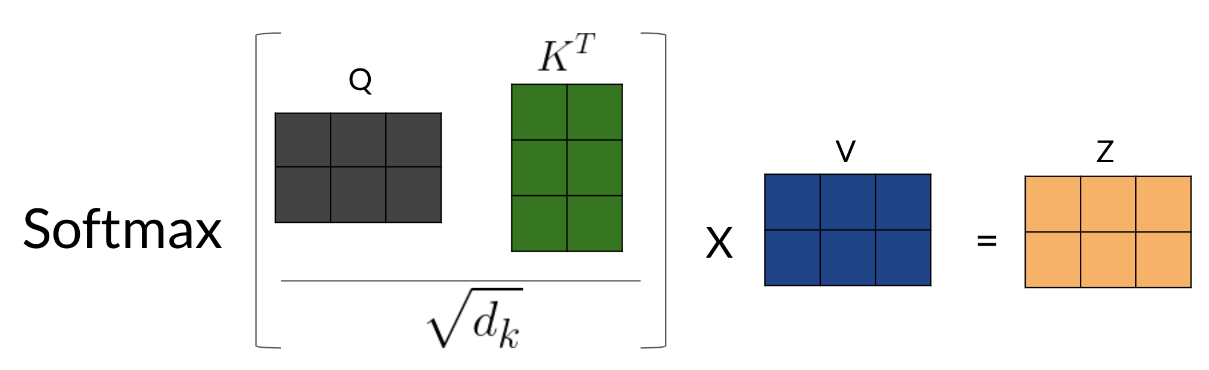
- Next, you feed it to the linear layer, once you go through the linear layer for each word, you need to calculate a score. After that, you end up having an embedding for each word. But you still need to get the score for how much of each word you are going to use. For example, this will tell you how similar two words are q\_1*q*1​ and k\_1*k*1​ or even q\_1*q*1​ and k\_2*k*2​ by doing a simple q\_1 \dot k\_1*q*1​*k*˙1​. You can take the softmax of those scores (the paper mentions that you have to divide by \sqrt(d)(​*d*) to get a probability and then you multiply that by the value. That gives you the new representation of the word.

If you have many heads, you can concatenate them and then multiply again by a matrix that is of dimension (dim of each head by num heads - dim of each head) to get one final vector corresponding to each word.

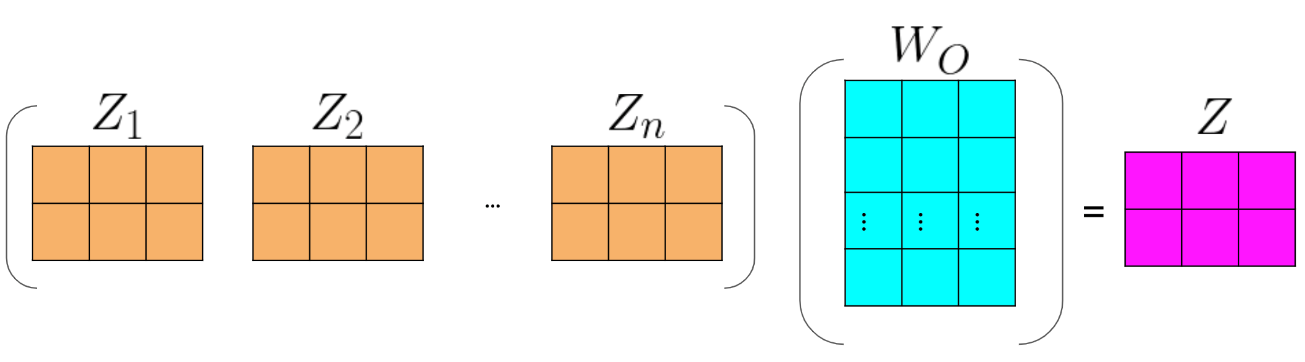
Here is step by step guide, first you get the Q, K, V matrices:



For each word, you multiply it by the corresponding W\_Q, W\_K, W\_V *WQ*​,*WK*​,*WV*​ matrices to get the corresponding word embedding. Then you have to calculate scores with those embedding as follows:



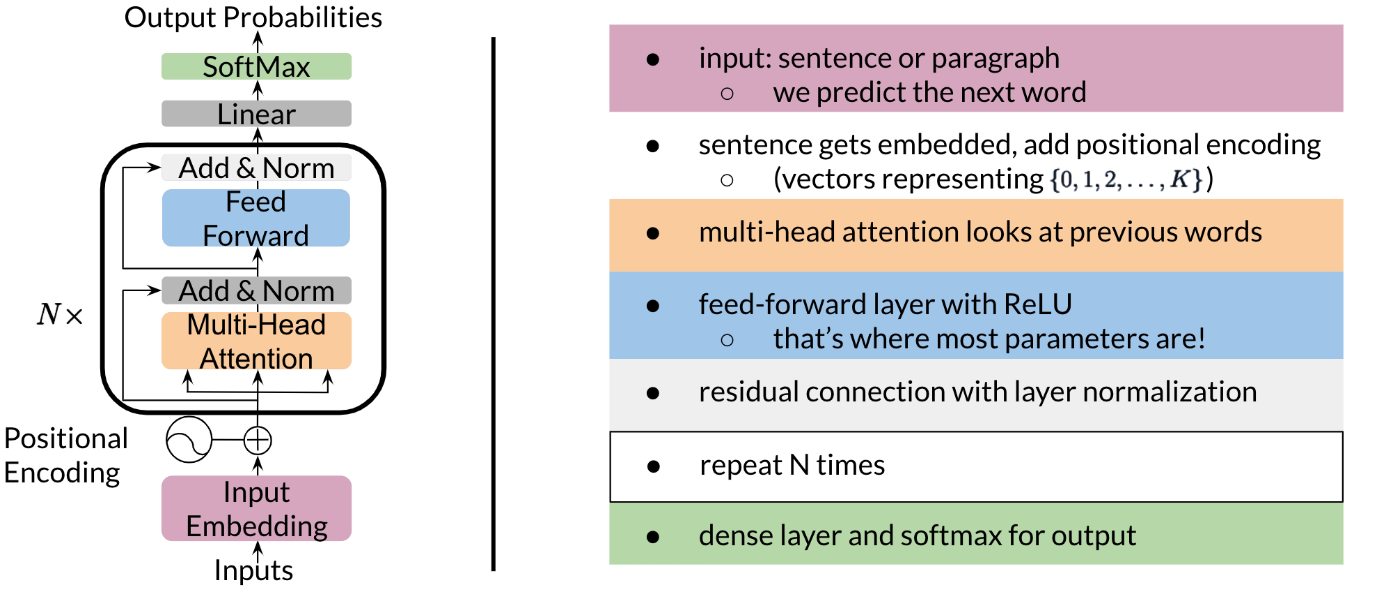
Note that the computation above was done for one head. If you have several heads, concretely n*n*, then you will have Z\_1, Z\_2, ..., Z\_n *Z*1​,*Z*2​,...,*Zn*​. In which case, you can just concatenate them and multiply by a W\_O*WO*​ matrix as follows:



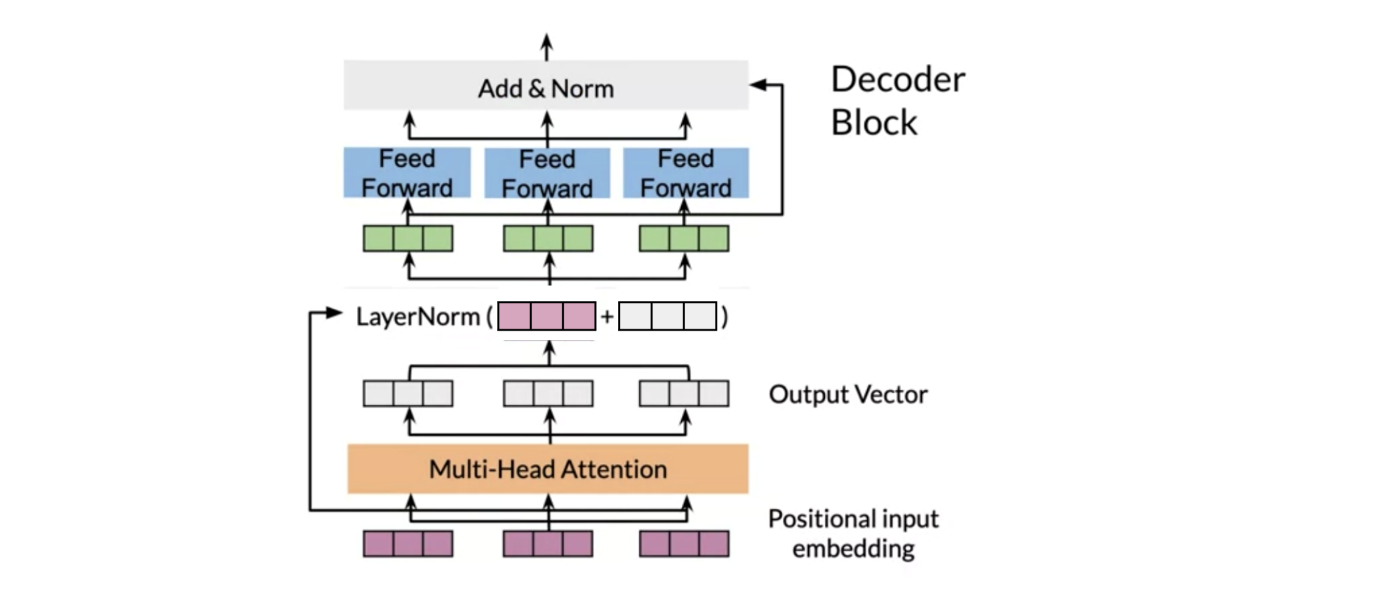
Hence, the more heads you have, the more Zs you will end up concatenating and as a result, that will change the inner dimension of W\_O*WO*​, which will then project the combined embeddings into one final embedding.

Transformer Decoder

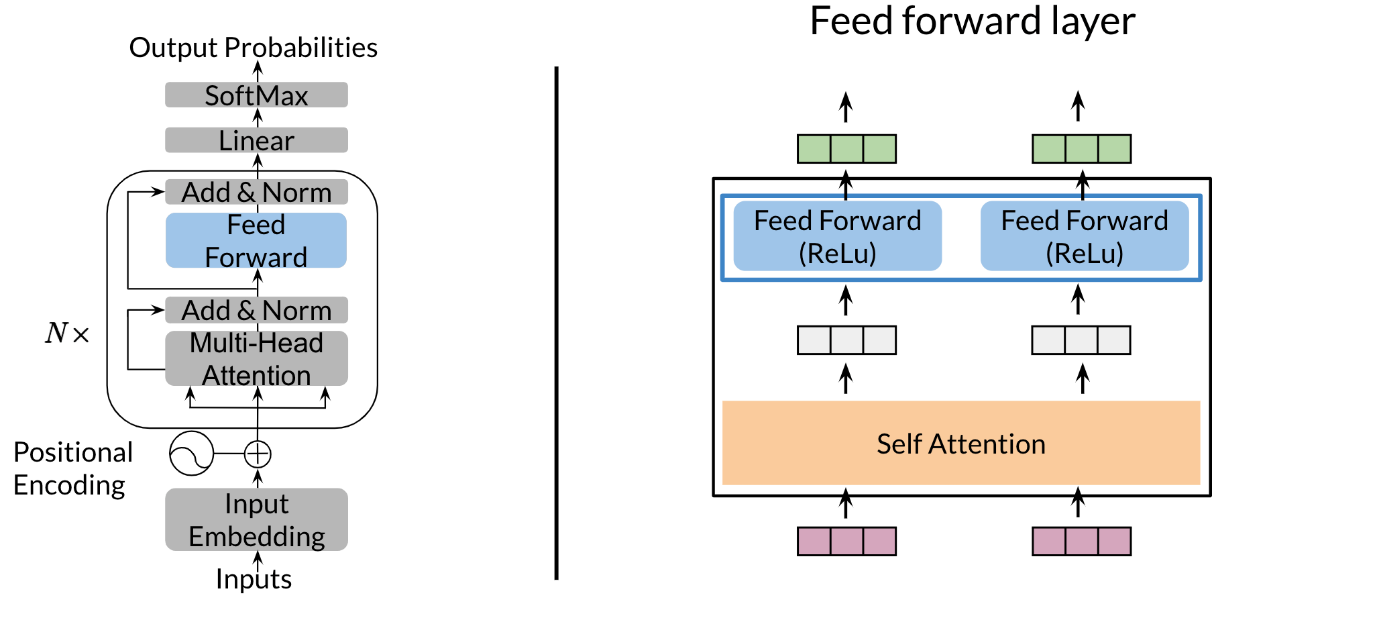
That was a lot of information! Don't worry if you did not understand everything, we will go over everything step by step.



Once you get the embeddings, you append to it the positional encoding, which you can think of just a learned number that tells you information about the position of the word. Then, you do multi-head attention as explained in the previous video/reading. There is a feedforward layer with a ReLU after this, then a residual connection with layer normalization (repeat the block shown above N times), finally a linear layer with a softmax. Zoning into the block that gets repeated N times, you get the following:



Now for the feedforward block, you can just implement the following:



You get the input, (red vector) run it through self-attention and then a feedforward with ReLU. At the end of the decoder, you can just run a softmax.