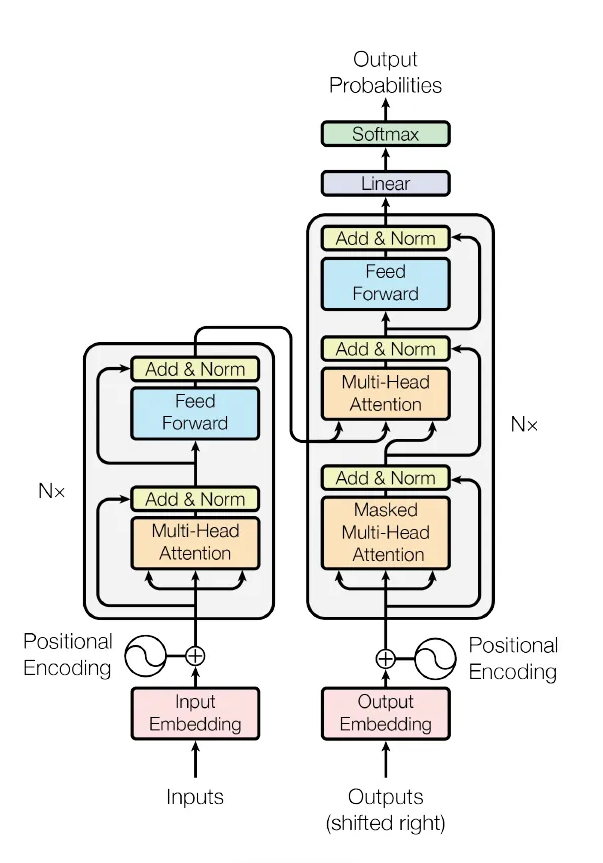
[“Attention Is All You Need” by Vaswani et al., 2017](https://arxiv.org/abs/1706.03762) was a landmark paper that proposed a completely new type of model — the Transformer. The model introduced a new mechanism that caused a revolution in the field of Natural Language Processing – The Attention Mechanism. Nowadays, the Transformer model plays an essential role in the realms of machine learning, but its algorithm is quite complex and hard to chew on.

In general, the Transformer model is based on an encoder-decoder architecture. The encoder is the grey rectangle on the left-hand side, the decoder the one on the right-hand side. Both the encoder and decoder consist of two and three sub-layers, respectively: **multi-head self-attention**, a **fully-connected feed forward network** and — in the case of the decoder —**encoder-decoder self-attention.**



What cannot be seen as clearly in the picture is that the Transformer actually stacks multiple encoders and decoders (which is denoted by Nx in the image, i.e., encoders and decoders are stacked n times). This means that the output of one encoder is used as the input for the next encoder — and the output of one decoder as the input for the next decoder.

**I. Attention Mechanism in the Encoder:**

The attention mechanism in the encoder of a Transformer model is a key component that allows the model to efficiently process input sequences and capture relevant information from them.

One of the problems of recurrent models is that long-range dependencies (within a sequence or across several sequences) are often lost. That is, if a word at the beginning of a sequence carries importance for a word at the end of a sequence, the model might have forgotten the first word once it reaches the last word. And to overcome this problem, self-attention is invented for the model to memorize the sentence better!



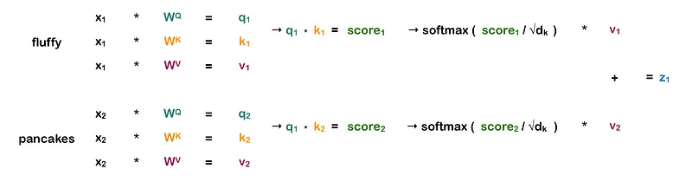
For instance, when a NLP model process the above sentence, it may not remember what is the proper pronounce to use at the end of the sentence, it could be ‘him’, ‘her’ or ‘it’? It does not know!

In the self-attention layer, an input x (represented as a vector) is turned into a vector z via three representational vectors of the input: q (Queries), k (Keys) and v (Values). These are used to calculate a score that shows how much attention that particular input should pay to other elements in the given sequence.

**This is the formula to calculate the Attention:**



That might still be hard to get the idea, we can express it this way:



Say we want to calculate **self-attention for the word “fluffy”** in the sequence “fluffy pancakes”. First, we take the input vector x1 (representing the word “fluffy”) and multiply it with three different weight matrices Wq, Wk and Wv (which are continually updated during training) in order to get three different vectors: q1, k1 and v1. The exact same is done for the input vector x2 (representing the word “pancakes”). We now have a query, key and value vector for both words.

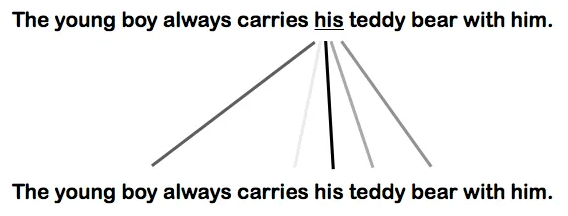
**And what are these Query, Key and value mean? Here’s what they do:**

The **query** is the representation for the word we want to calculate self-attention for. So since we want to get the self-attention for “fluffy”, we only consider its query, not the one of “pancakes”. As soon as we are finished calculating the self-attention for “fluffy”, we can also discard its query vector.

The **key**is a representation of each word in the sequence and is used to match against the query of the word for which we currently want to calculate self-attention.

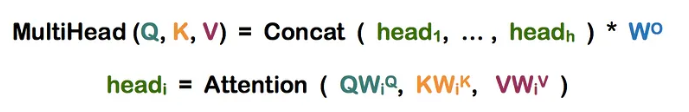
The **value**is the actual representation of each word in a sequence, the representation we really care about. Multiplying the query and key gives us a score that tells us how much weight each value (and thus, its corresponding word) obtains in the self-attention vector. Note that the the value is not directly multiplied with the score, but first the scores are divided by the square root of the dk, the dimension of the key vector, and softmax is applied.

The result of these calculations is one vector for each word. As a final step, these two vectors are summed up, and voilà, we have the self-attention for the word “fluffy”.

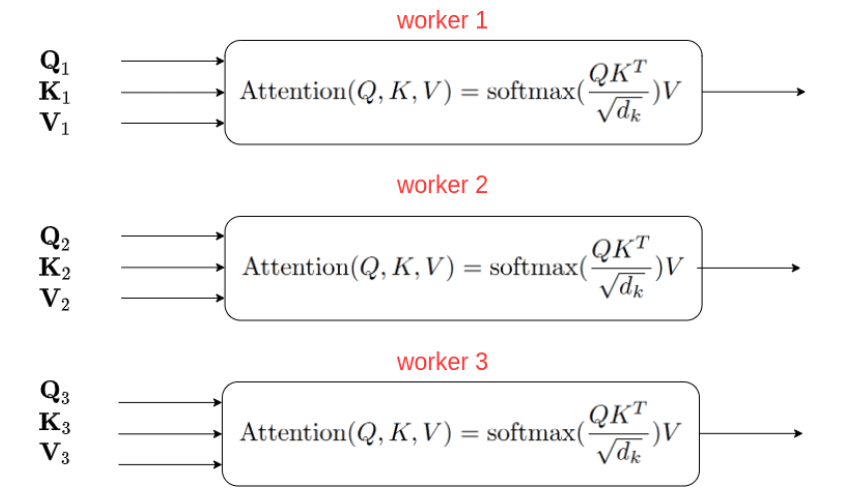


The picture above illustrates how much Attention the word ‘his’ pays to any other word in the sentence.

The process above is carried out multiple times with different weight matrices, which means we end up with multiple vectors (called heads in the formulae below). These heads are then concatenated and multiplied with a weight matrix Wo. This means that each head learns different information about a given sequence and that this knowledge is combined at the end.



But the most important trait of Transformer is that all of the calculations above can be parallelized, and that is what makes the Transformer model even more beautiful!



Let’s look at RNNs first. They need to process sequential data in order, each word of a sequence is passed to the model one by one, one after the other. Transformer models, however, can process all inputs at once. And this makes these models incredibly fast, allowing them to be trained with huge amounts of data.

As for the Transformer model, it can process all words in a sequence in parallel. However, this means that some important information is lost: the **word position in the sequence**. To retain this information, the position and order of the words must be made explicit to the model. This is done via positional encodings. These positional encodings are vectors with the same dimension as the input vector and are calculated using a sine and cosine function. To combine the information of the input vector and the positional encoding, they are simply summed up.

**2. Attention Mechanism in the Encoder-Decoder:**

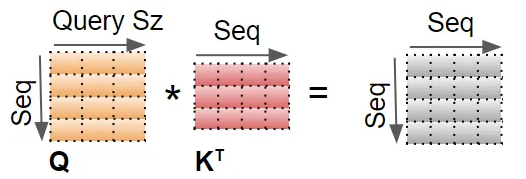
The encoder-decoder attention mechanism is used to weight the encoder outputs based on their relevance to the current position in the decoder.

Unlike the self-attention mechanism used within each layer of the Transformer encoder and decoder, which computes attention scores between all pairs of positions within a single sequence, the encoder-decoder attention mechanism computes attention scores between each position in the decoder and all positions in the encoder.

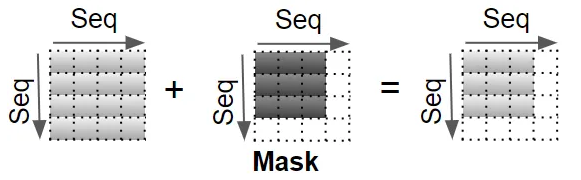
Specifically, at each position in the decoder, the encoder-decoder attention mechanism computes a set of attention scores between that position and all positions in the encoder. These attention scores are then used to weight the encoder outputs, allowing the decoder to attend to different parts of the input sequence as needed.

To compute the attention scores, the Transformer model uses three learned parameter matrices: W\_q, W\_k, and W\_v. These matrices are used to transform the decoder input, encoder outputs, and encoder outputs, respectively, into query, key, and value representations that are used to compute the attention scores.

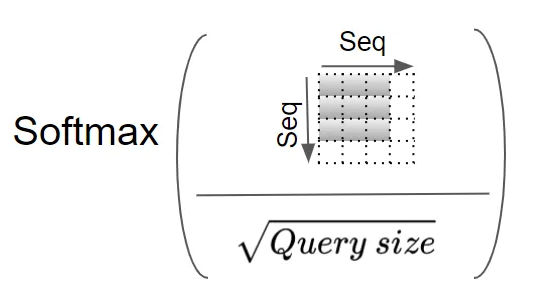
The first step is to do a matrix multiplication between Q and K.



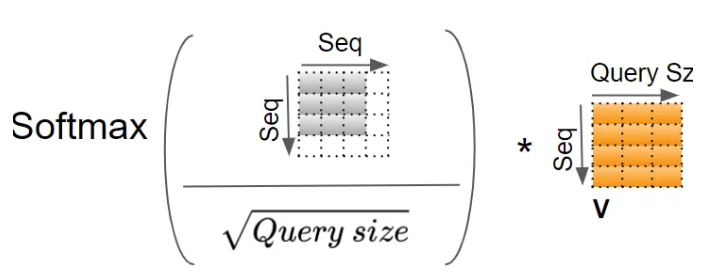
A Mask value is now added to the result. In the Encoder Self-attention, the mask is used to mask out the Padding values so that they don’t participate in the Attention Score.



The result is now scaled by dividing by the square root of the Query size, and then a Softmax is applied to it.



Another matrix multiplication is performed between the output of the Softmax and the V matrix.



The resulting attention output is then concatenated with the decoder input at the current position and passed through a feedforward network to produce the final output for that position.

The encoder-decoder attention mechanism is a key component of many natural language processing tasks, including machine translation, where it allows the model to align the source and target sentences and generate accurate translations.

**3. Attention Mechanism in the Decoder:**

All of the basics of the Attention mechanism in the Encoder will still be used in the Decoder Attention mechanism.

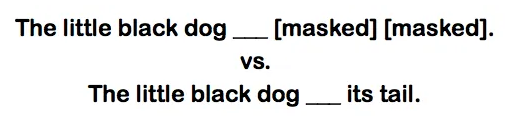
The decoder, however, uses what is called **masked multi-head self-attention**. This means that some positions in the decoder input are masked and thus ignored by the self-attention layer.

The reason some positions in the decoder attention mechanism are masked is to prevent the model from "cheating" by attending to future positions in the output sequence.

In the decoder of a Transformer model, the output sequence is generated one position at a time, with each position being generated based on the previous positions. When computing the attention scores between a given position in the decoder and the encoder outputs, it's important to ensure that the decoder can only attend to information that was available at the time of generation, and not to any information that comes after the current position.

To achieve this, the decoder attention mechanism typically includes a masking step that sets the attention scores for any future positions to a very large negative value. This effectively excludes those positions from consideration, as the resulting softmax over the attention scores will assign a near-zero weight to any future positions.

By masking future positions in the decoder attention mechanism, the model is forced to rely only on information that was available at the time of generation, which encourages it to generate outputs that are more coherent and realistic.



In the first sentence (masked), the next word is far more difficult to predict than in the second sentence (unmasked). The words “its tail” make it clear the word to predict is probably “wiggled”.