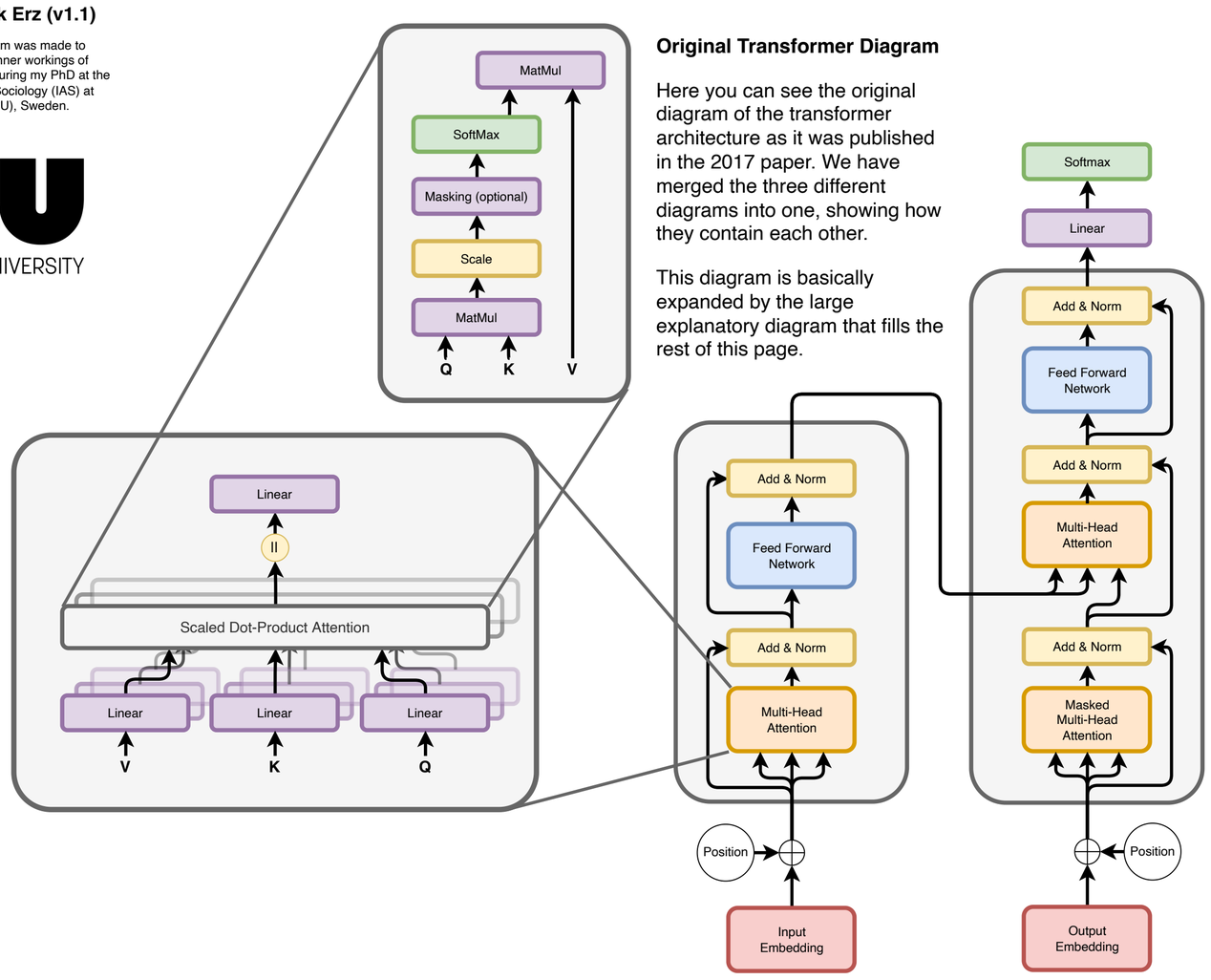
This transformer diagram was made to better understand the inner workings of transformer networks during my PhD at the Institute for Analytical Sociology (IAS) at Linköping University (LiU), Sweden.

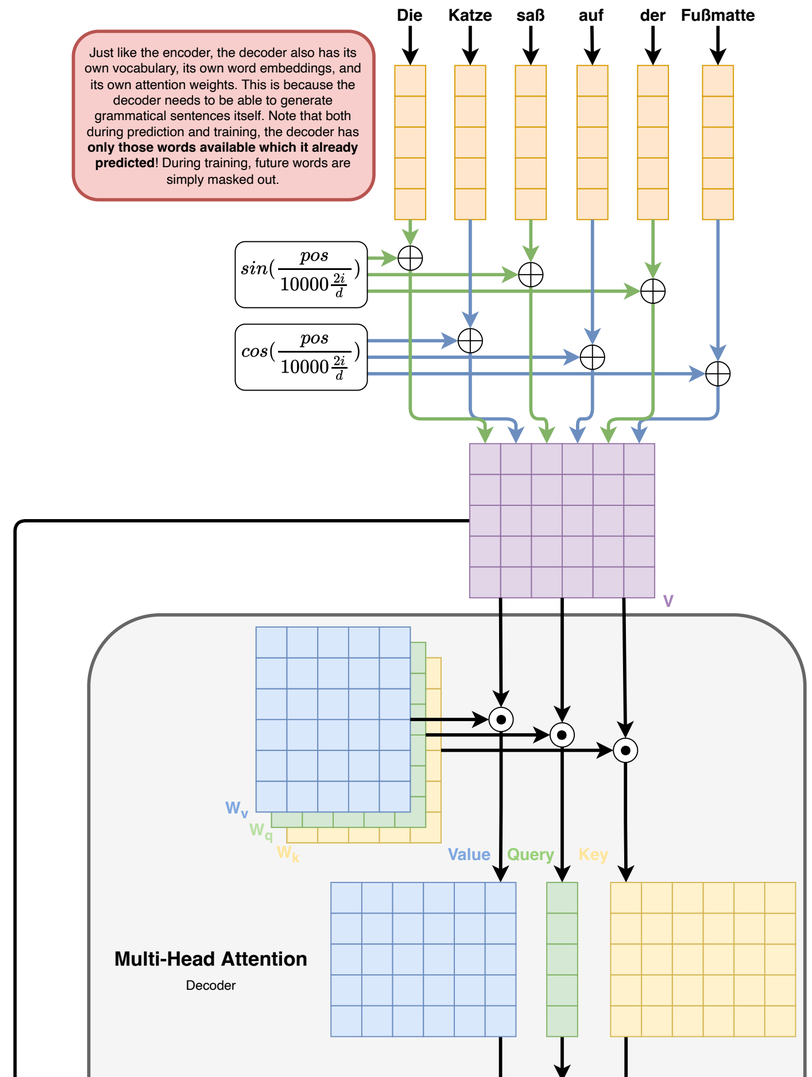
LINKÖPING UNIVERSITY

Original Transformer Diagram

Here you can see the original diagram of the transformer architecture as it was published in the 2017 paper. We have merged the three different diagrams into one, showing how they contain each other.

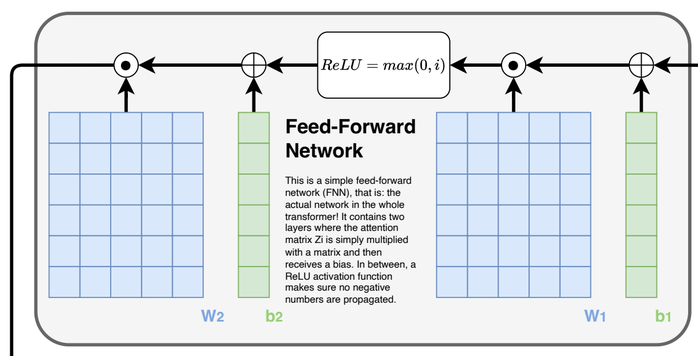
This diagram is basically expanded by the large explanatory diagram that fills the rest of this page.

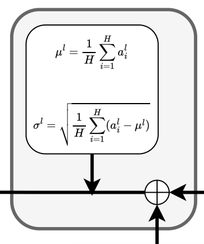


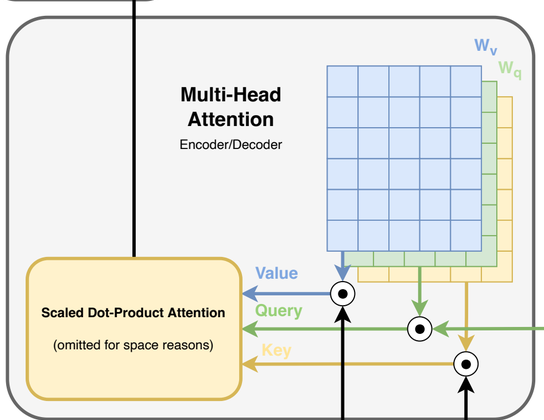


Just like the encoder, the decoder also has its own vocabulary, its own word embeddings, and its own attention weights. This is because the decoder needs to be able to generate grammatical sentences itself. Note that both during prediction and training, the decoder has **only those words available which it already predicted**! During training, future words are simply masked out.

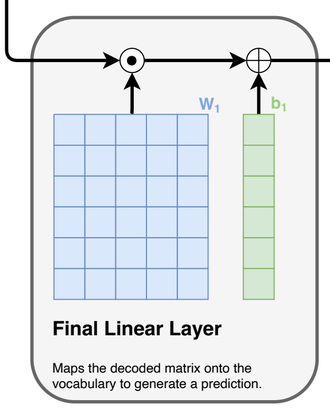
Notice how often there are residual connections and afterwards layer normalizations. The very fact that the different states have to be normalized that often gives a hint at how big that model is.

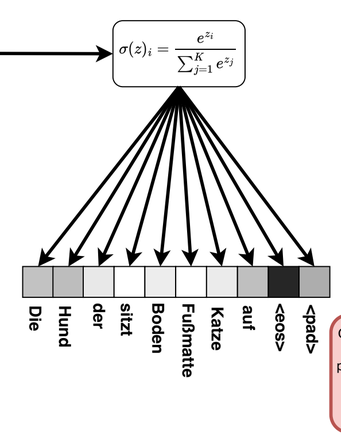




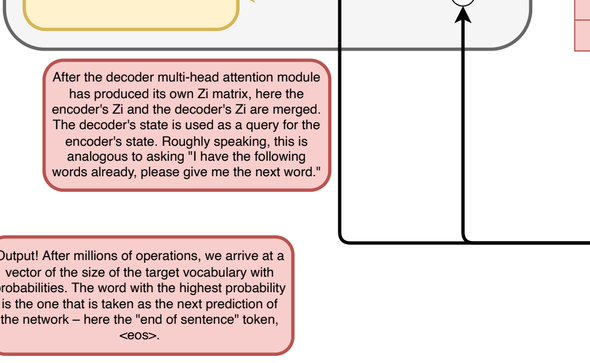


After the decoder multi-head attention module has produced its own Zi matrix, here the encoder's Zi and the decoder's Zi are merged. The decoder's state is used as a query for the encoder's state. Roughly speaking, this is analogous to asking "I have the following words already, please give me the next word."



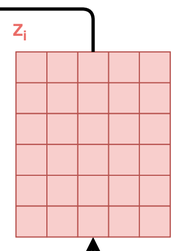


Output! After millions of operations, we arrive at a vector of the size of the target vocabulary with probabilities. The word with the highest probability is the one that is taken as the next prediction of the network – here the "end of sentence" token, <eos>.



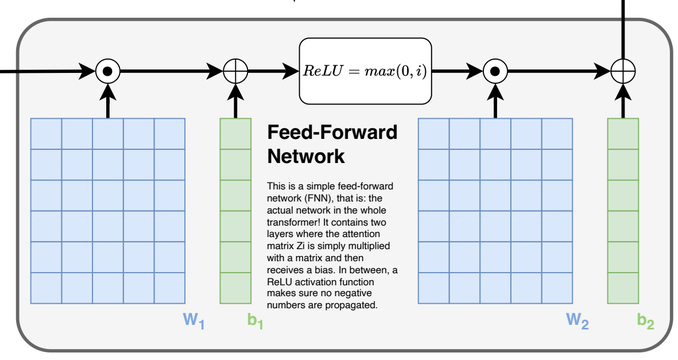
This matrix is the final output of the multi-head attention module. It is in quotes since there is only a loose connection to the "hidden layers" of feed-forward networks.

Encoder Output



This is it! After the feed-forward network, we have reached the end of the encoder! The matrix Zi contains the encoded information about how each word in the input sequence relates to each other, from multiple attention heads and including a transformation via the feed-forward network below.

After the residual connection and layer normalization to the left, you can stack one or more encoders on top of this (e.g. eight). The Zi matrix from the final encoder stage is then passed to the decoder.

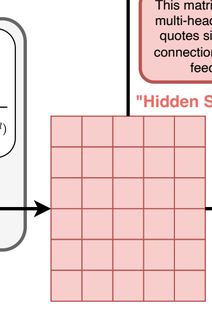


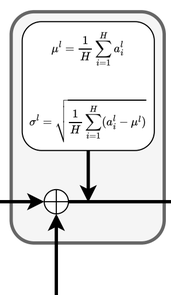
Residual Connection & Layer Normalization

After the attention matrix has been calculated, the original word embeddings with positional encodings (V) are then added to it, before the whole resulting matrix is normalized.

Adding the original input to the output of a layer (here: the multi-head attention) is called a **residual connection** since it re-introduces the unaltered input once again. This prevents the phenomenon of vanishing or exploding gradients.

Layer normalization simply put means to calculate the mean and standard deviation of all the numbers in the matrix and then re-center it.





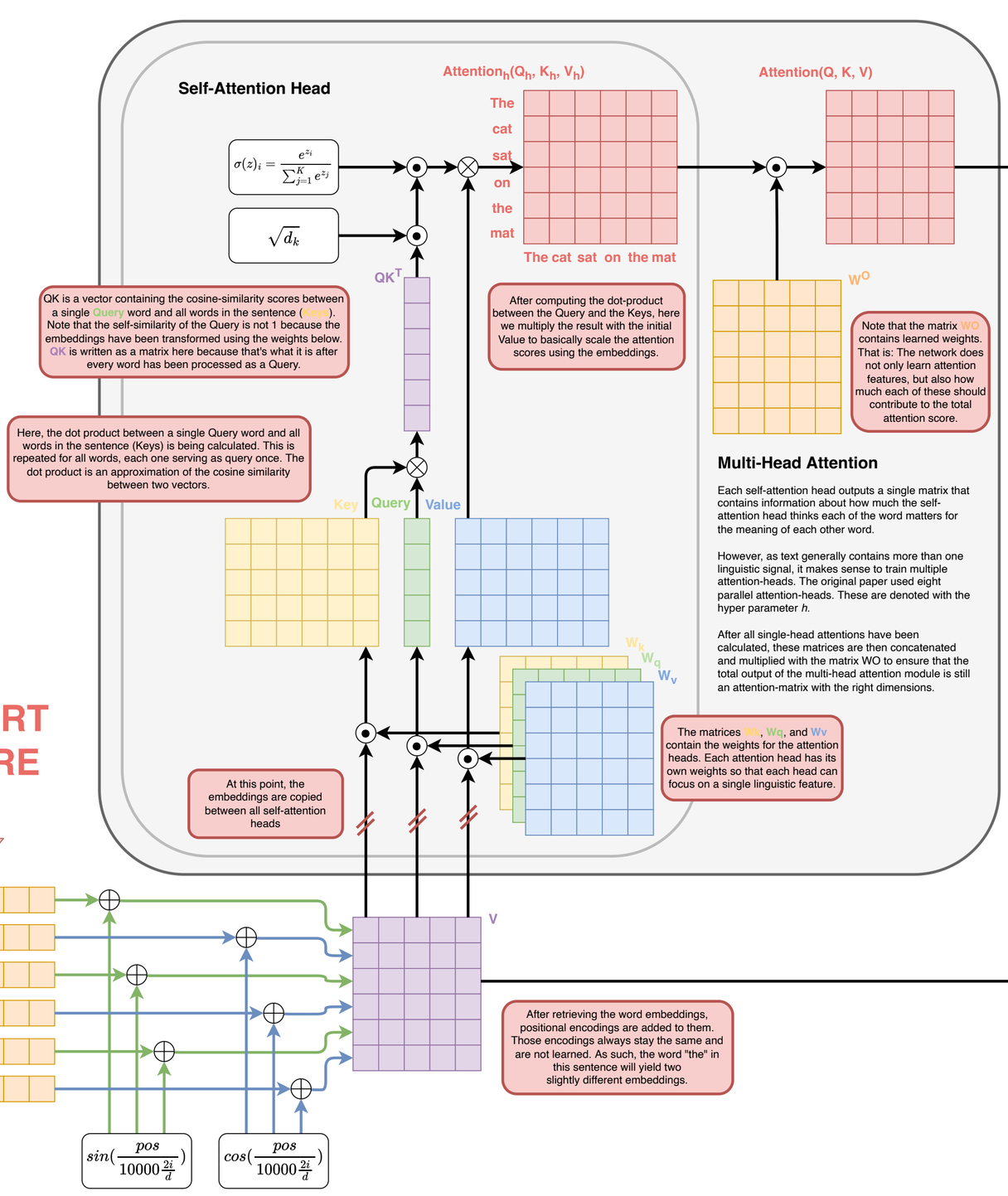
Note that the matrix W^O contains learned weights. That is: The network does not only learn attention features, but also how much each of these should contribute to the total attention score.

Multi-Head Attention

Each self-attention head outputs a single matrix that contains information about how much the self-attention head thinks each of the word matters for the meaning of each other word.

However, as text generally contains more than one linguistic signal, it makes sense to train multiple attention-heads. The original paper used eight parallel attention-heads. These are denoted with the hyper parameter *h*.

After all single-head attentions have been calculated, these matrices are then concatenated and multiplied with the matrix WO to ensure that the total output of the multi-head attention module is still an attention-matrix with the right dimensions.



QK is a vector containing the cosine-similarity scores between a single Query word and all words in the sentence (Keys). Note that the self-similarity of the Query is not 1 because the embeddings have been transformed using the weights below. QK is written as a matrix here because that's what it is after every word has been processed as a Query.

After computing the dot-product between the Query and the Keys, here we multiply the result with the initial Value to basically scale the attention scores using the embeddings.

Here, the dot product between a single Query word and all words in the sentence (Keys) is being calculated. This is repeated for all words, each one serving as query once. The dot product is an approximation of the cosine similarity between two vectors.

The matrices W_k, W_q, and W_v contain the weights for the attention heads. Each attention head has its own weights so that each head can focus on a single linguistic feature.

At this point, the embeddings are copied between all self-attention heads

START HERE

After retrieving the word embeddings, positional encodings are added to them. Those encodings always stay the same and are not learned. As such, the word "the" in this sentence will yield two slightly different embeddings.

The Transformer Architecture

Transformers have completely changed the way we build neural architectures for natural language processing (NLP) tasks. They have replaced the previous recurrent neural networks (RNN) and their modifications using general rectified unit (GRU) and long short-term memory (LSTM) cells.

In the form of BERT, GPT, ERNIE, and some other questionably named architectures they are much more powerful than their predecessors. They are more accurate, faster, better to train due to parallelization, and produce much better results. However, all of this comes at the cost that they are much harder to interpret. Whereas LSTM cells still had a vague resemblance to the sequential nature of text, transformers are just three attention modules in a trench coat.

"Attention is all you need" becomes apparent in this visualization of the complete, unabridged transformer architecture, since the actual "neural network" part – the two feed-forward networks (FFN) – make up only a fraction of the whole.

References

Alammar (2018): The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Doshi (2021a): Transformers Explained Visually — Not Just How, but Why They Work So Well. Towards Data Science.

Doshi (2021b). Transformers Explained Visually (Part 3): Multi-head Attention, deep dive. Towards Data Science.

Kurbiel (2020). Drawing the Transformer Network from Scratch (Part 1). Towards Data Science.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv:1706.03762 [Cs]. http://arxiv.org/abs/1706.03762

Legend

- Addition.
- Multiplication.
- Dot-product.
- $max(0, i)$ White formulas are applied to everything in their path.
- Grey boxes denote different sub-modules of the network.
- Red boxes contain further explanations.