Transformer

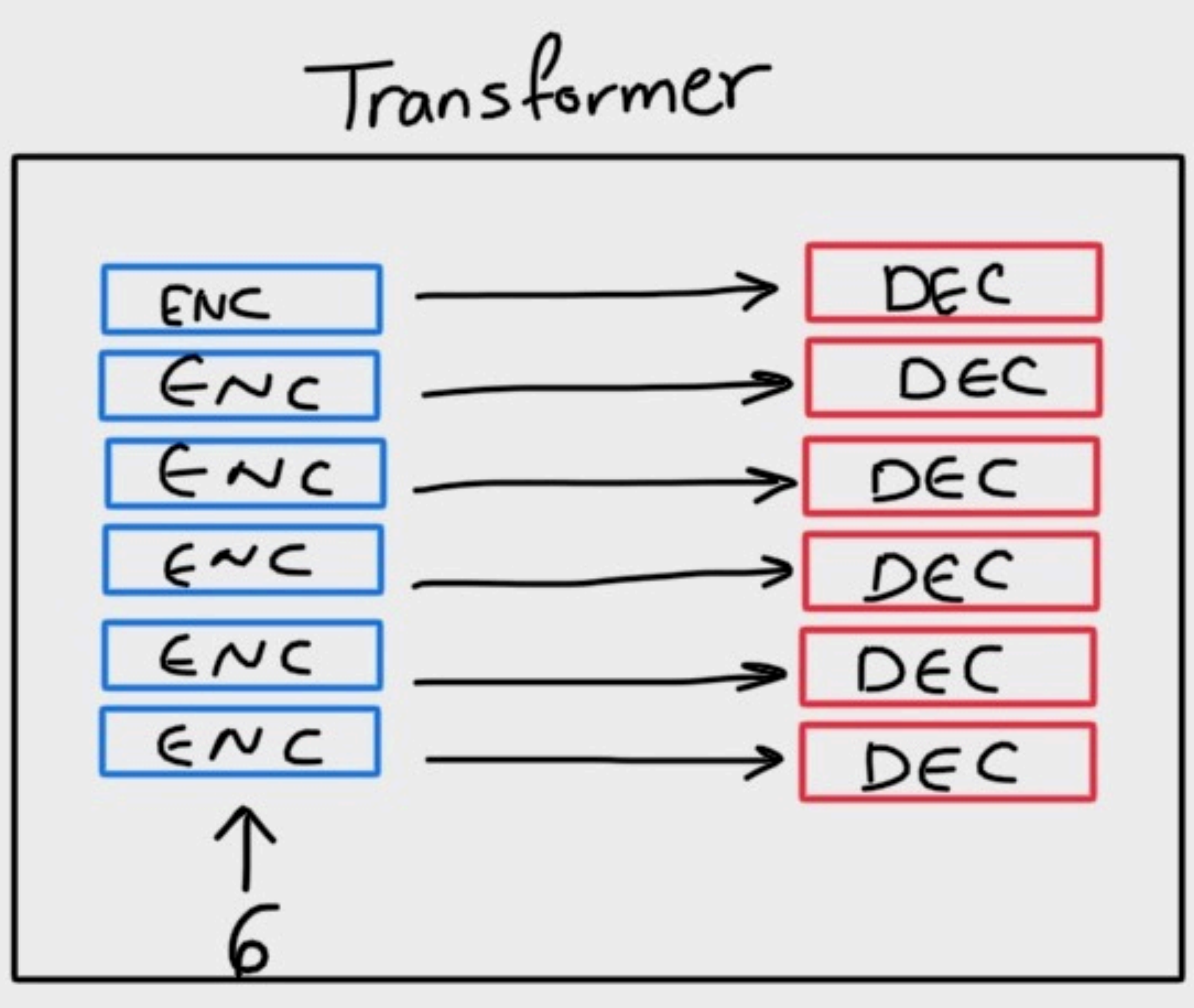
improve Performance due to parallels:

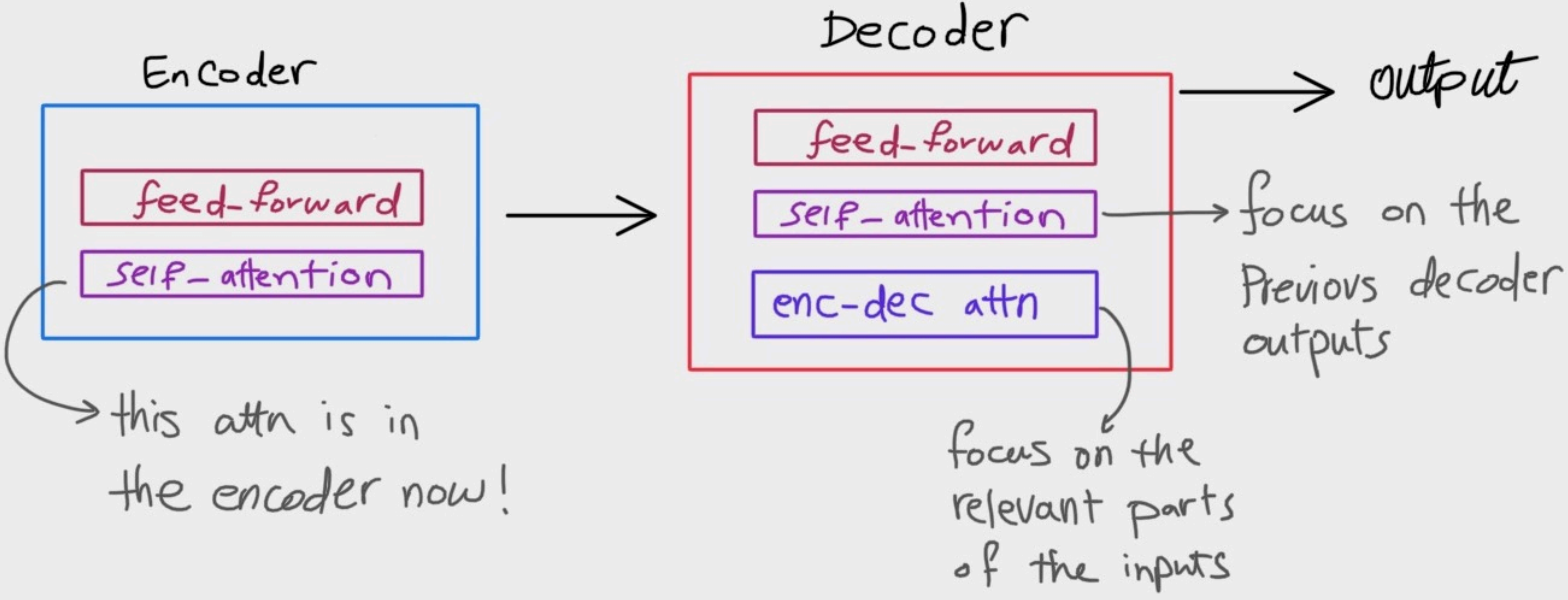
As opposed to RNN models that take the input sequence token_by_token, the transformer takes the input sequence as a whole (parallel), then generates the output one-by-one as before.

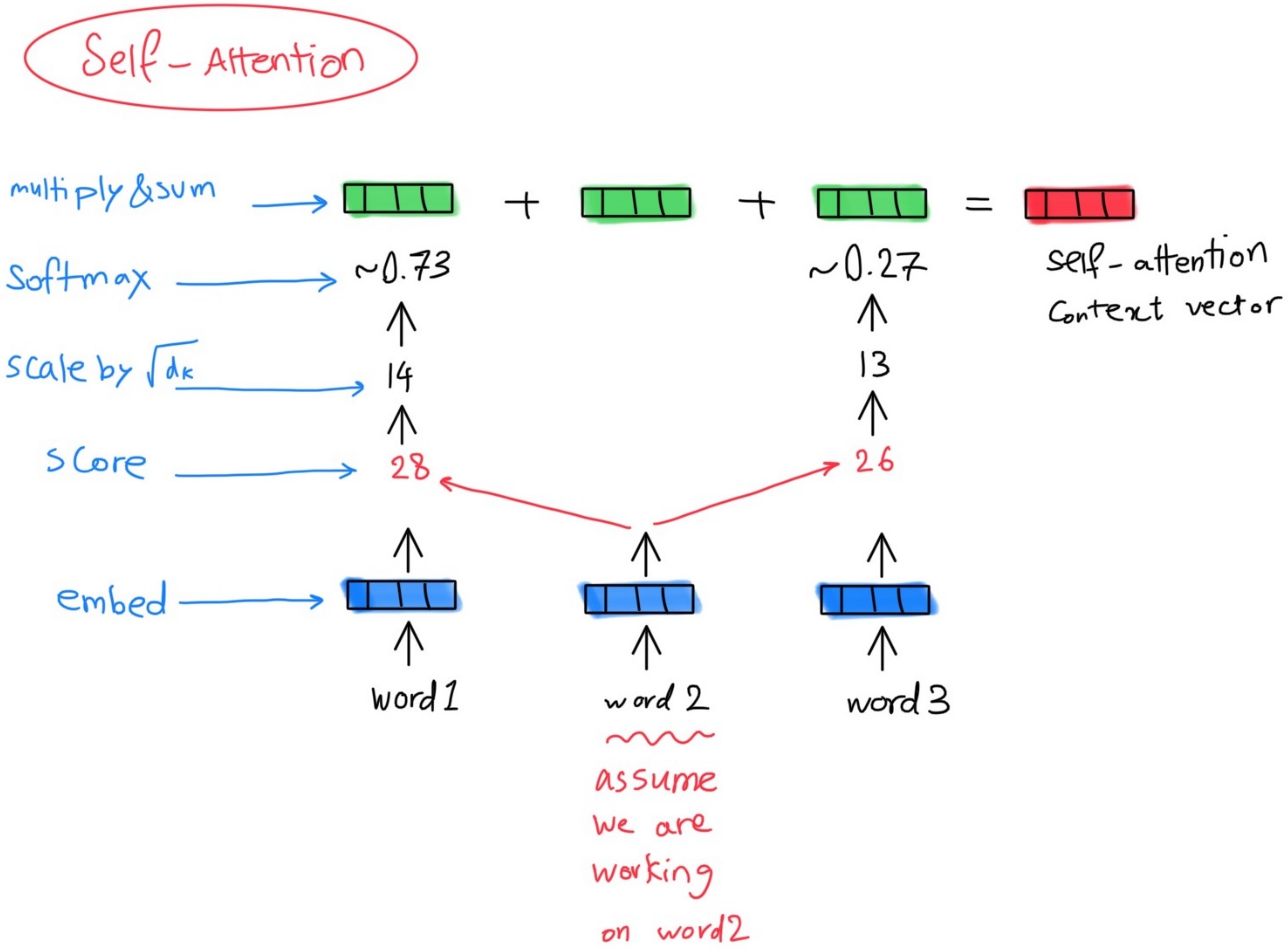
Another difference between transformer seq2seq models and RNN seq2seq is that they use feed-forward neural networks instead of RNNs.

The third major novelty in transformers is a concept called self_attention.

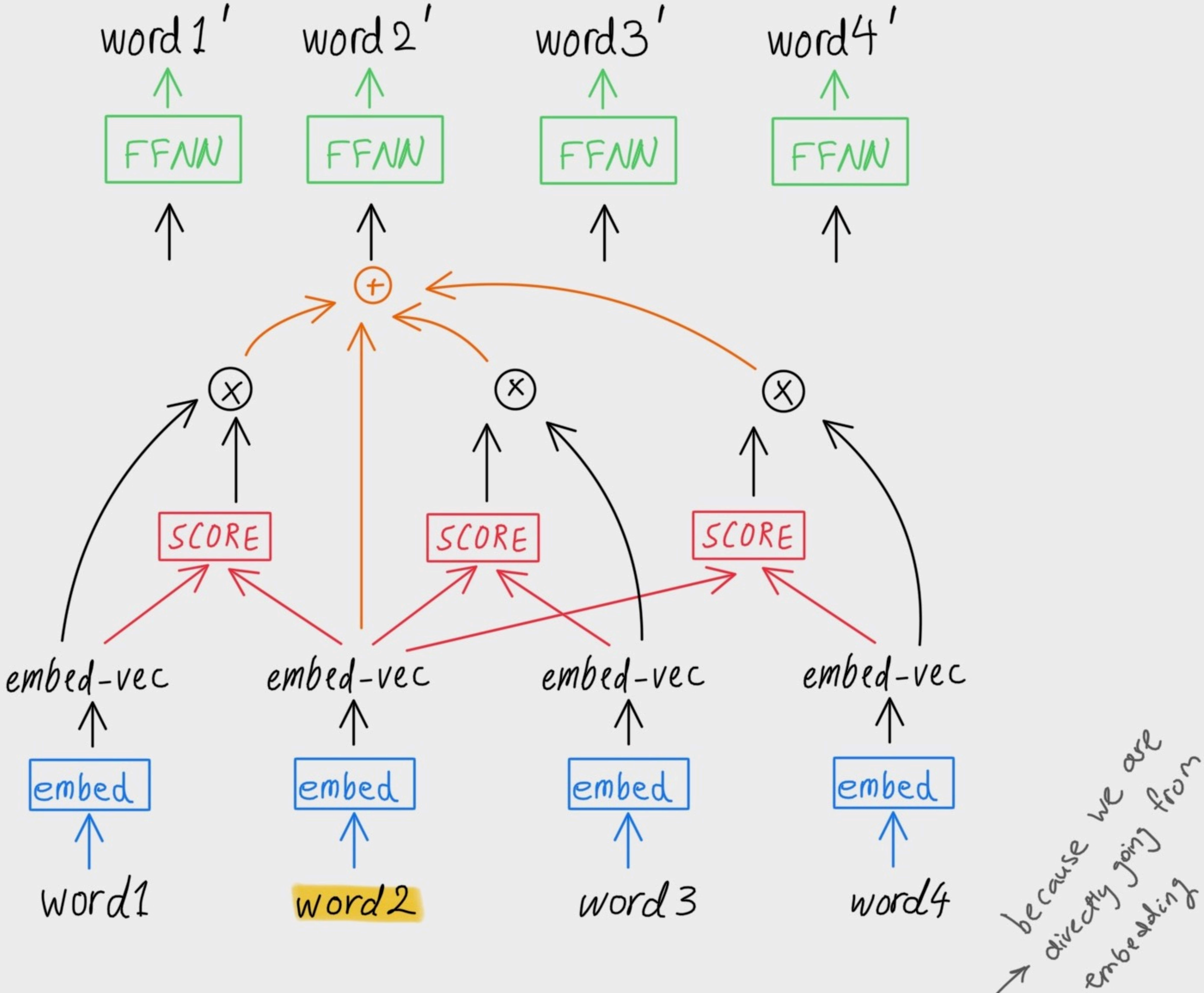
It has a Stack of encoders and decoders.



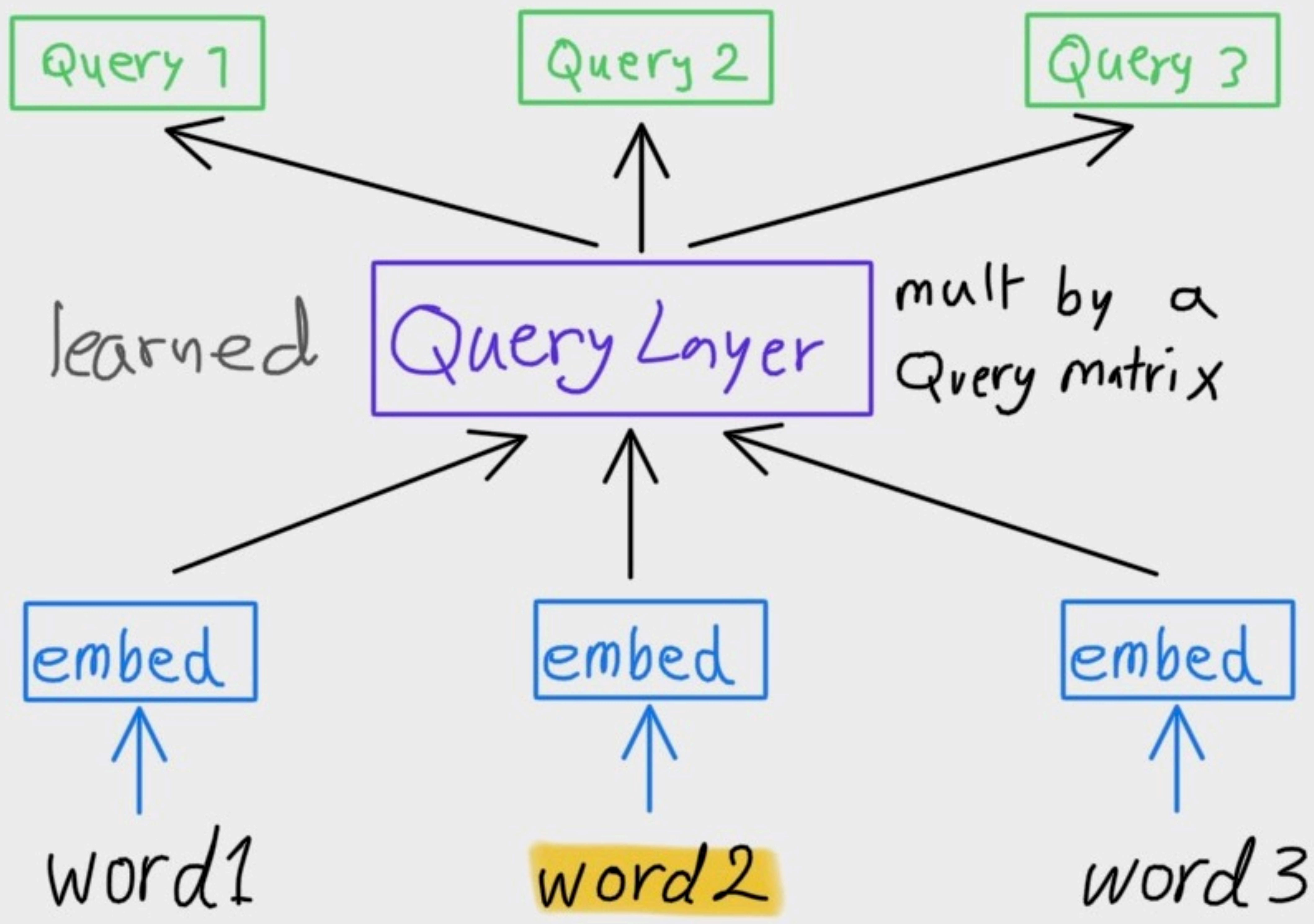

Encoder
feed-forward
self-attention
this attn is in the encoder now!
Decoder
feed-forward
self-attention
enc-dec attn
output
focus on the Previous decoder outputs
focus on the relevant parts of the inputs

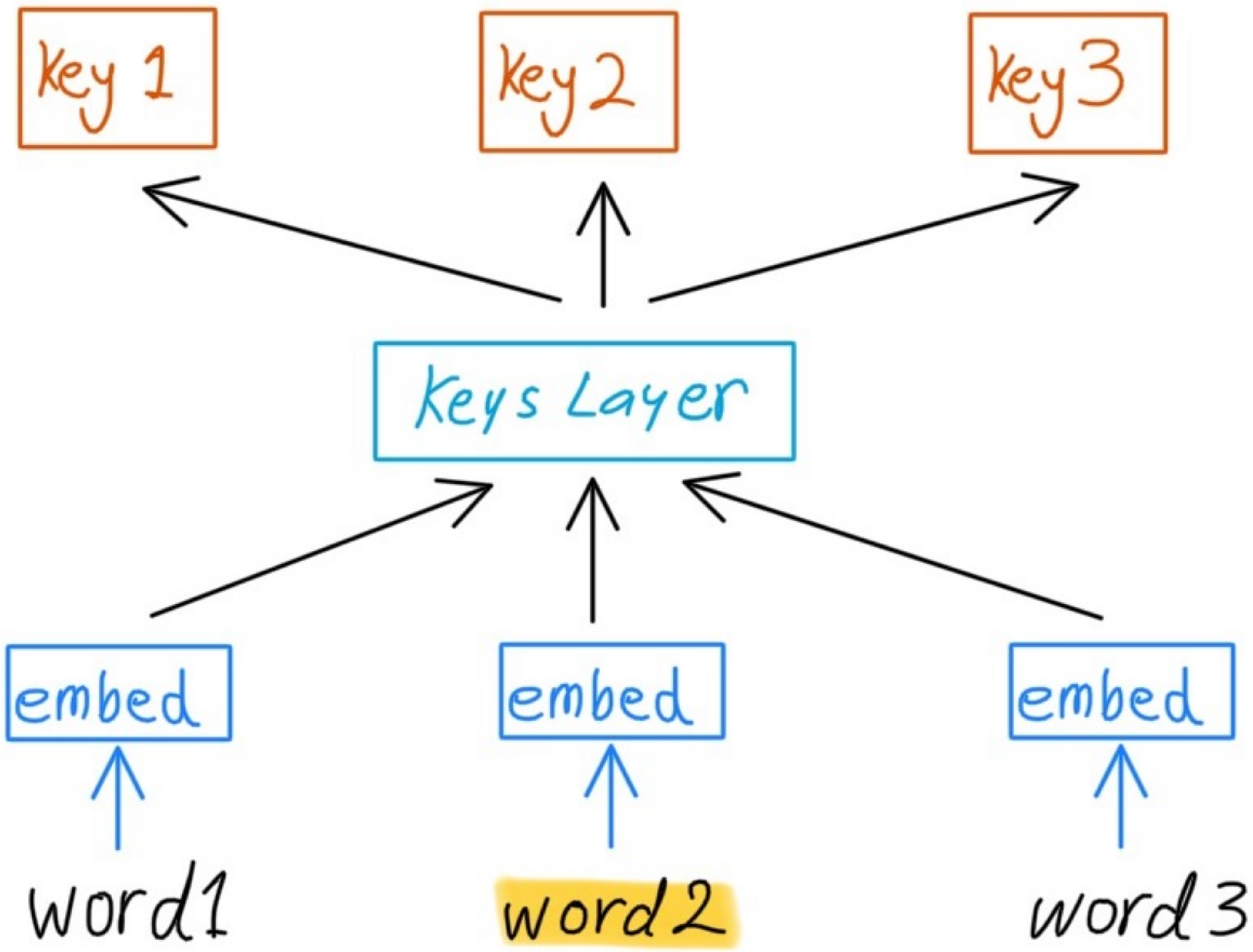

Self-Attention
multiply &sum
+
+
=
Softmax
~0.73
~0.27
self-attention
Context vector
scale by $\sqrt{d_k}$
14
13
score
28
26
embed
word 1
word 2
word 3
assume
we are
working
on word2

De-crypt the transformer paper



- Problem in this figure: model is mainly focusing on other similar words. → Let's modify it!


Query 1
Query 2
Query 3
learned
Query Layer
mult by a
Query matrix
embed
embed
embed
word1
word2
word3



• Using the keys and queries, we calculate like this :

		Q	K	score(Q, K)	$*\sqrt{d_k}$	softmax	softmax * K
word1	$embed^1$		key1	28	14	0.2	vec1
word2	$embed^2$	Query	key2				vec2
word3	$embed^3$		key3	26	13	0.8	vec3

vec1 + vec2 + vec3 self-attn Context vector → Passed to FFNN

- The authors also add a third thing: values. So:

		Q	K	V	score(Q, K)	$*\sqrt{d_k}$	softmax	softmax* V
word1	embed¹		Key1	Val1	28	14	0.2	vec1
word2	embed²	Query	Key2	Val2				vec2
word3	embed³		Key3	Val3	26	13	0.8	vec3

Q matrix: learned during training process learned

K : by multiplying embeddings by K matrix

V : by multiplying embeddings by V matrix learned