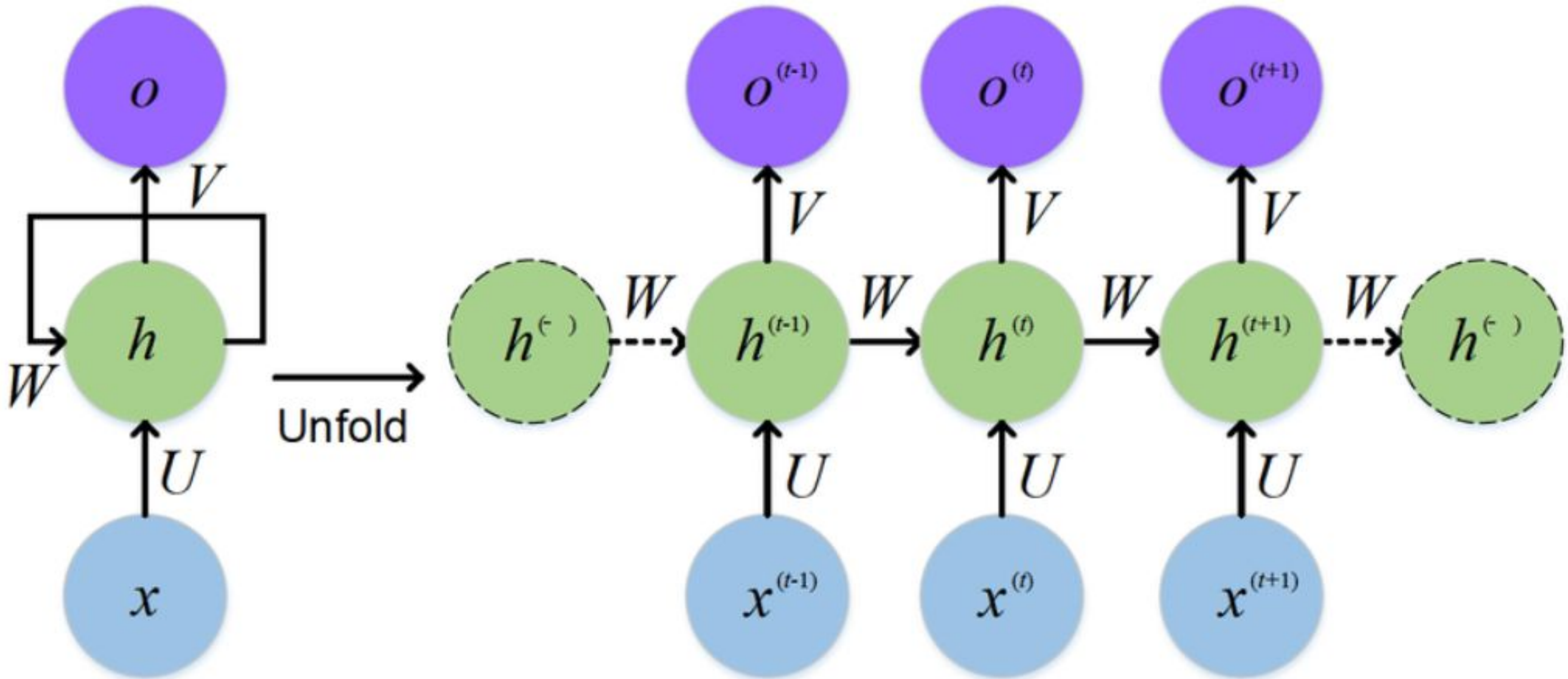
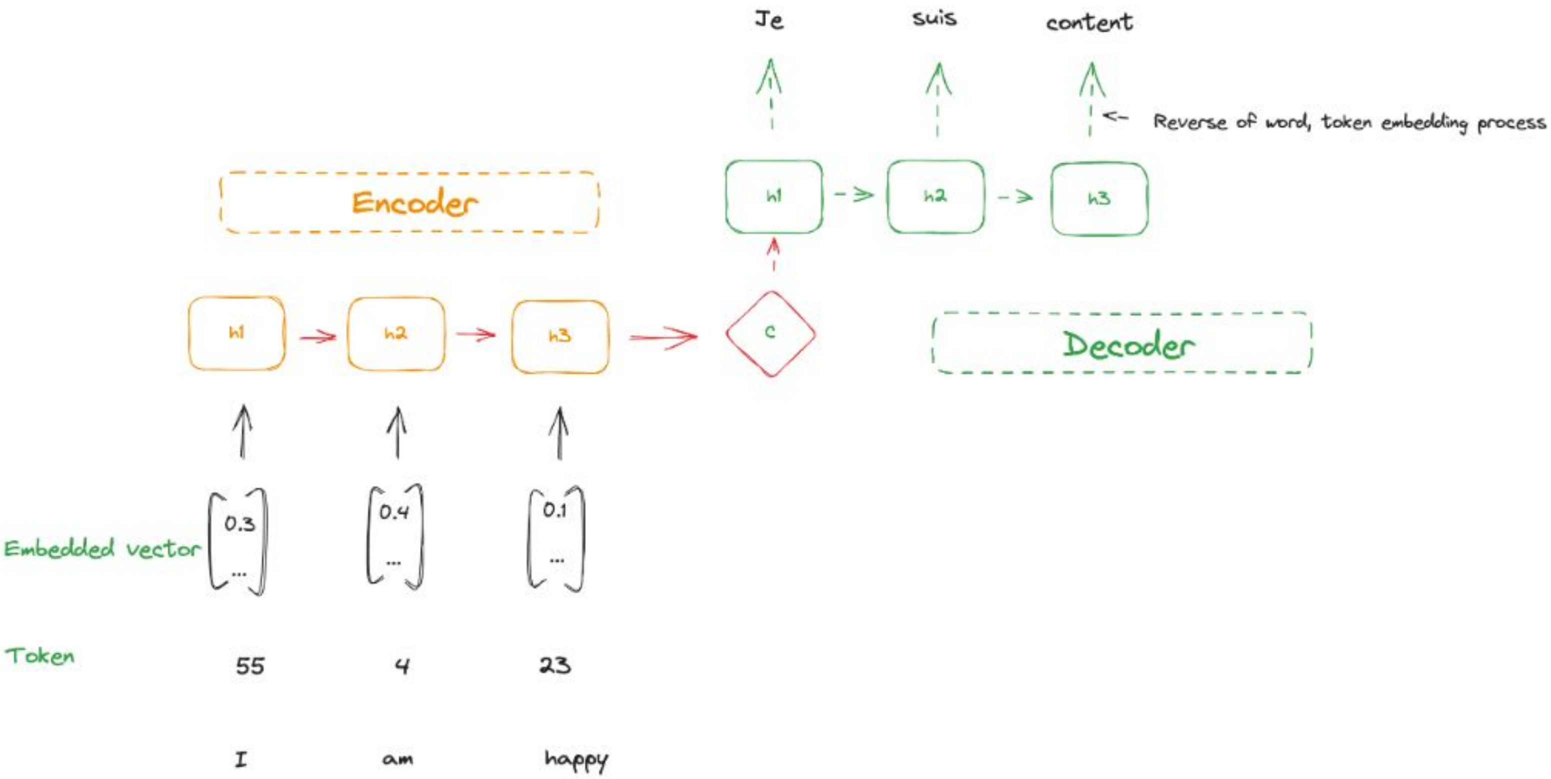
Transformers

1 Why should you care about Transformers?

ChatGPT

2 RNNs: Problems and progress


o
V
W
h
U
x
Unfold
o^(t-1)
o^(t)
o^(t+1)
h^(t-1)
h^(t)
h^(t+1)
x^(t-1)
x^(t)
x^(t+1)

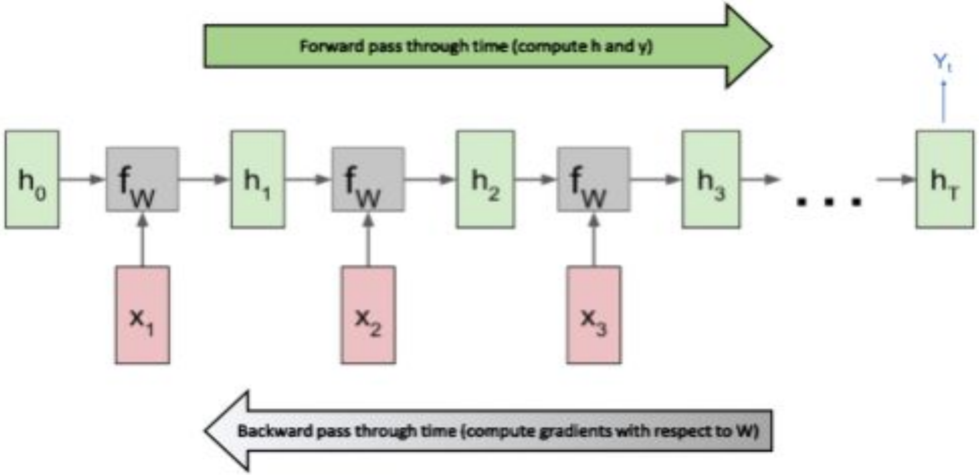


Je
suis
content
<- Reverse of word, token embedding process
h1
h2
h3
Encoder
c
Decoder
Embedded vector
0.3
...
0.4
...
0.1
...
Token
55
4
23
I
am
happy

What are the key issues we face here?

1. Information bottleneck at interface
2. Vanishing gradient problem
3. We have to compute the entire sequence recursively (makes scaling very hard!)

RNNs suffer from vanishing gradient through time



What does this mean for our performance?

A simplification of problems with RNNs:

> *Sally adored reading; when she received a book on her birthday she was older*

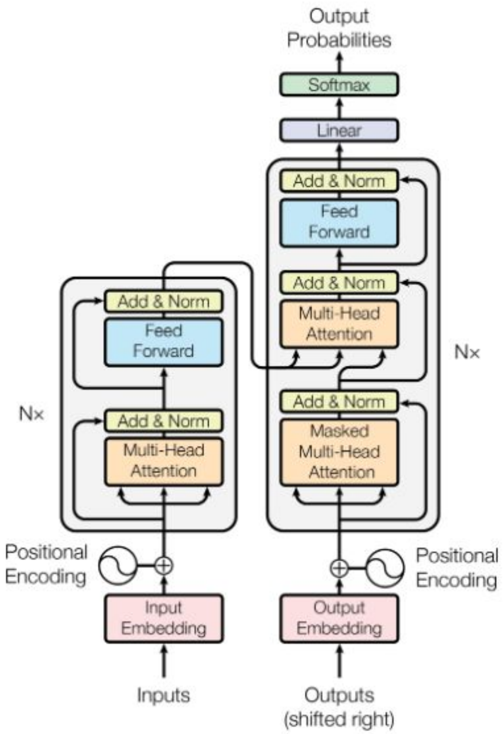
What we'd like:

> *Sally adored reading; when she received a book on her birthday she was happy!*

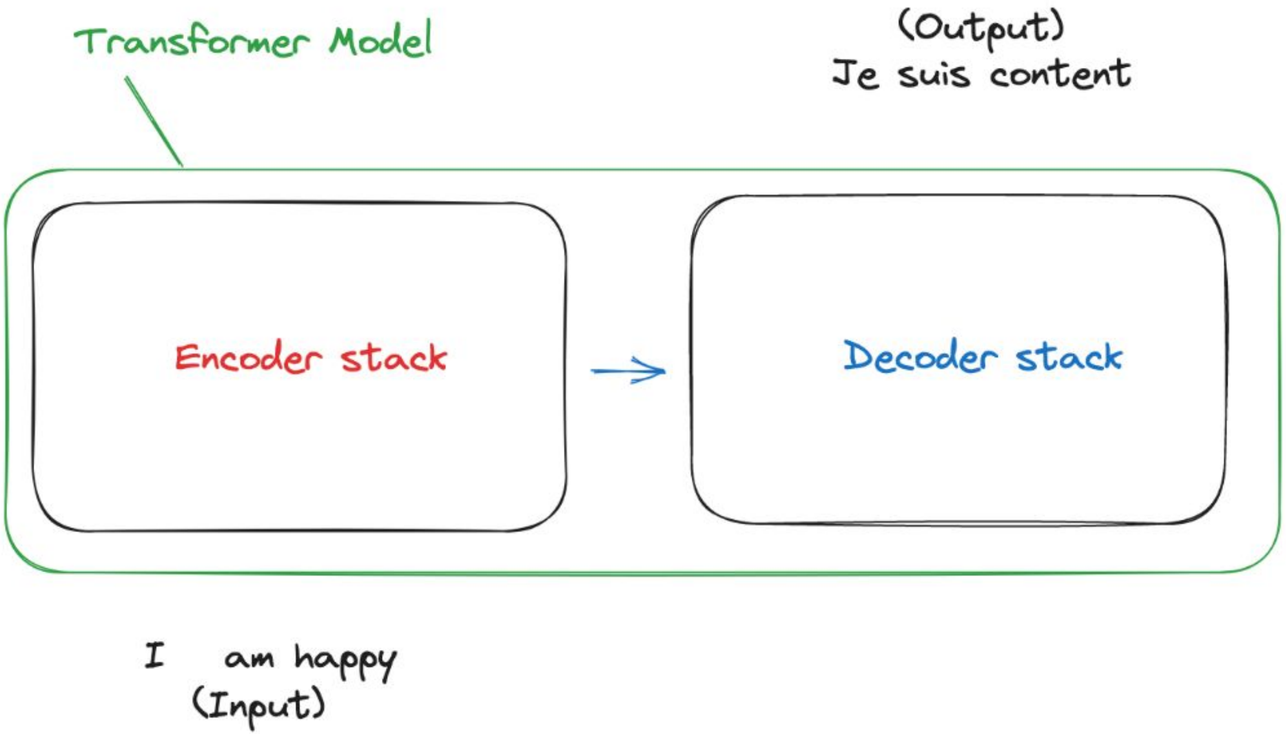
RNNs are likely to miss out on **important context** from earlier in the sentence because of their recency bias 🤗

3 Transformers

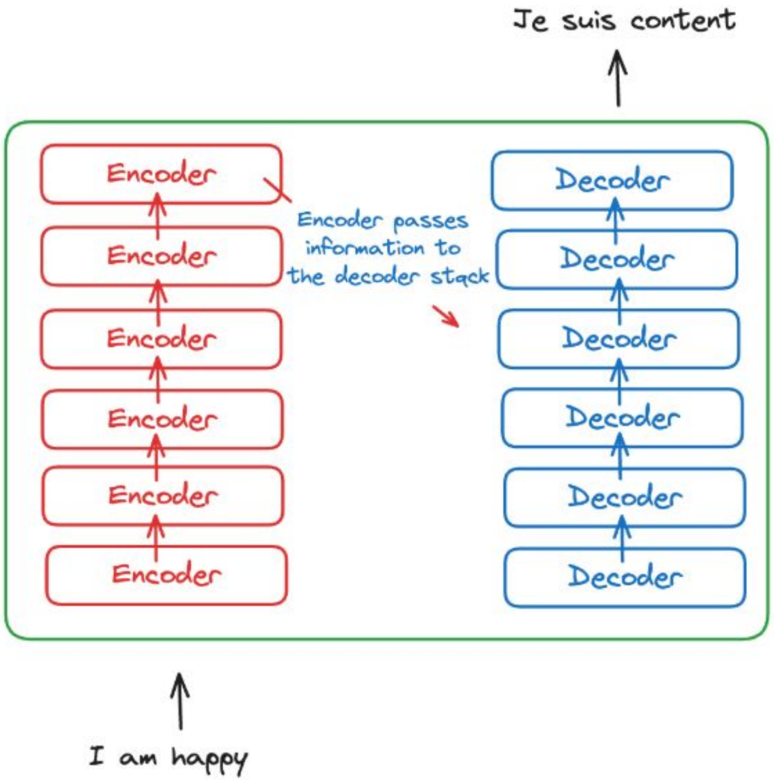
The paper that started it all: Attention is All You Need



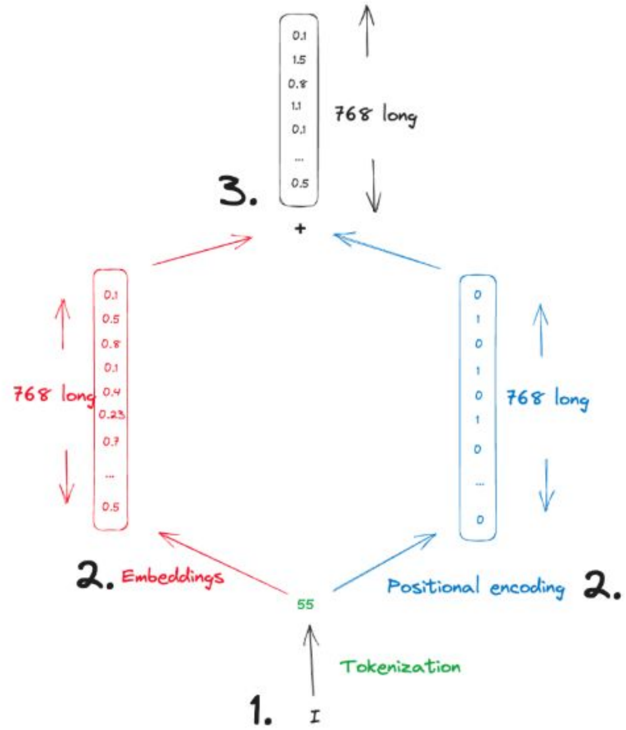
The highest level view:



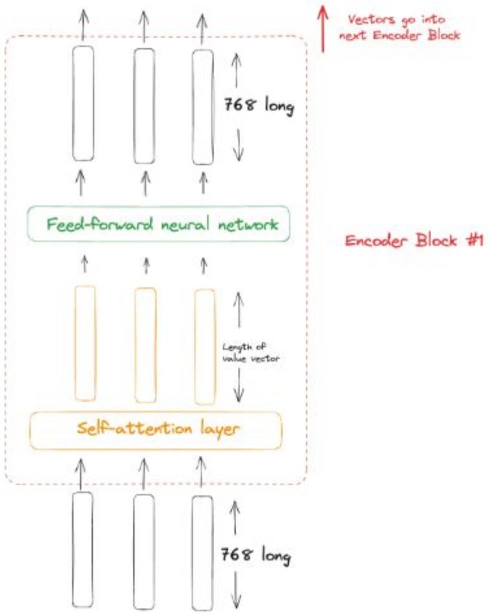
Broken down a bit more:



Before we talk about what's going on inside the encoder layers, let's talk about what's going into it!



Zooming in on one of the encoders:



- Usually the **embedding size** is 768
- The **hidden dimension** (the length of the projected Q, K and V vectors) is also 768
- In the example we'll use size 2 for simplicity

❓ What's going on in this strange self-attention layer?

1) Each token (word) embedding gets **projected** ➡️ into 3 further vectors: the **query, key and value vectors**

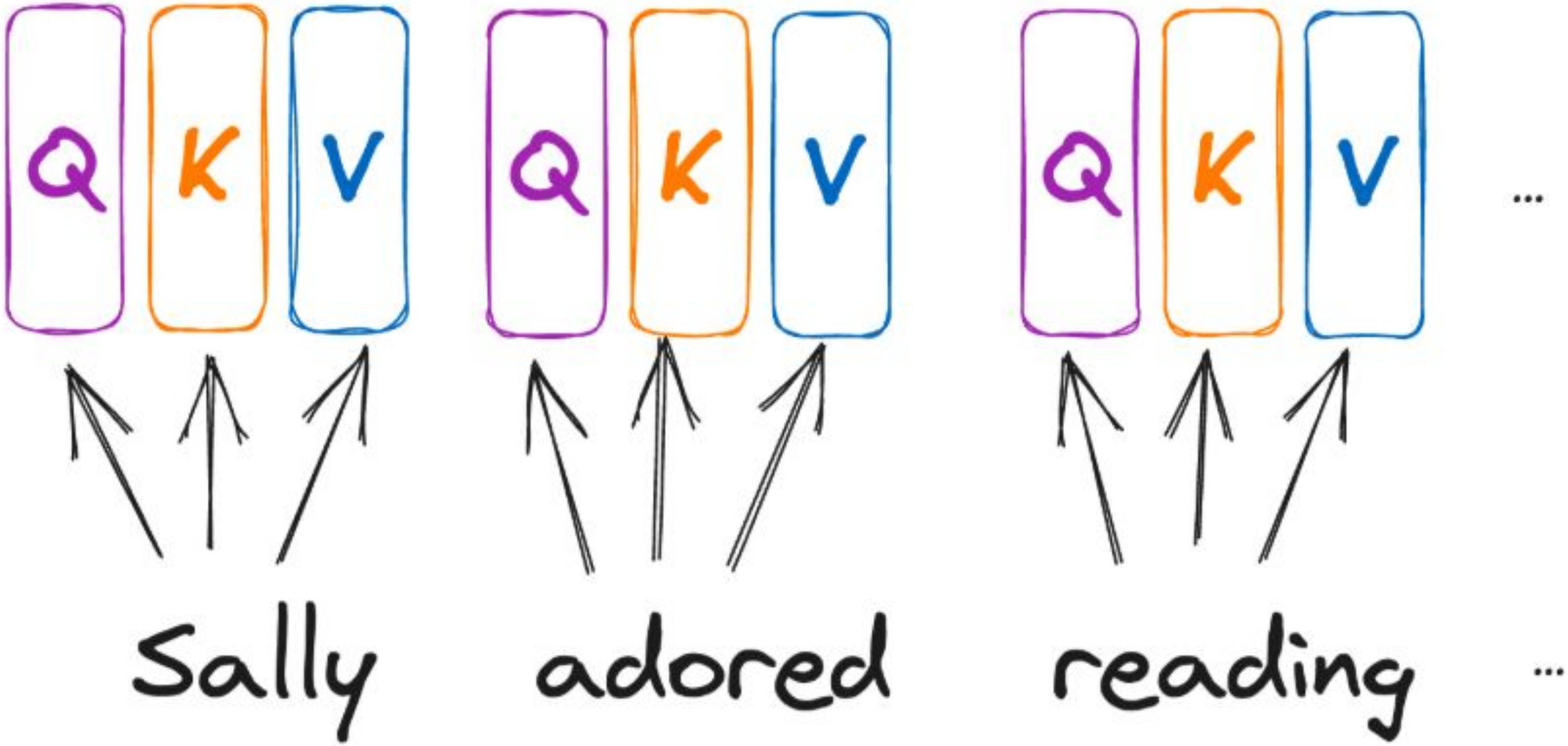
2) We compute a **scaled dot-product** 🔴 on the query and key vectors to work out how much each word relates to those around it

3) Take these scores and **normalize with softmax** ⤵️

4) **Multiply by our value vectors** ❎, sum and pass to our dense neural network.

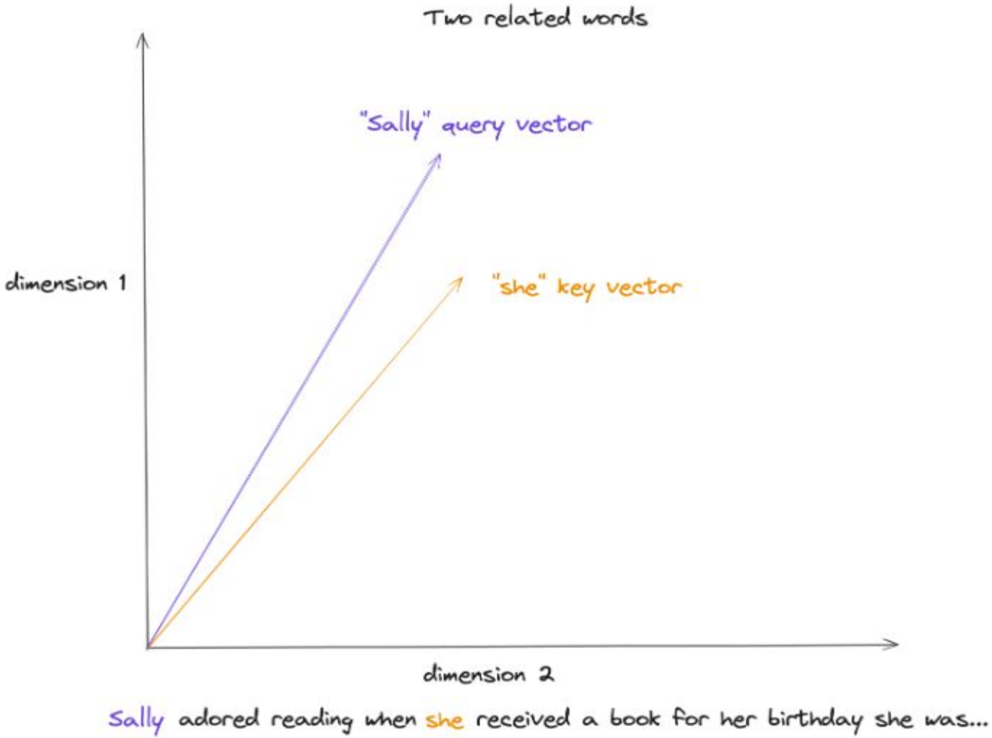
An example sentence

Sally adored reading; when she received a book for her birthday she was...

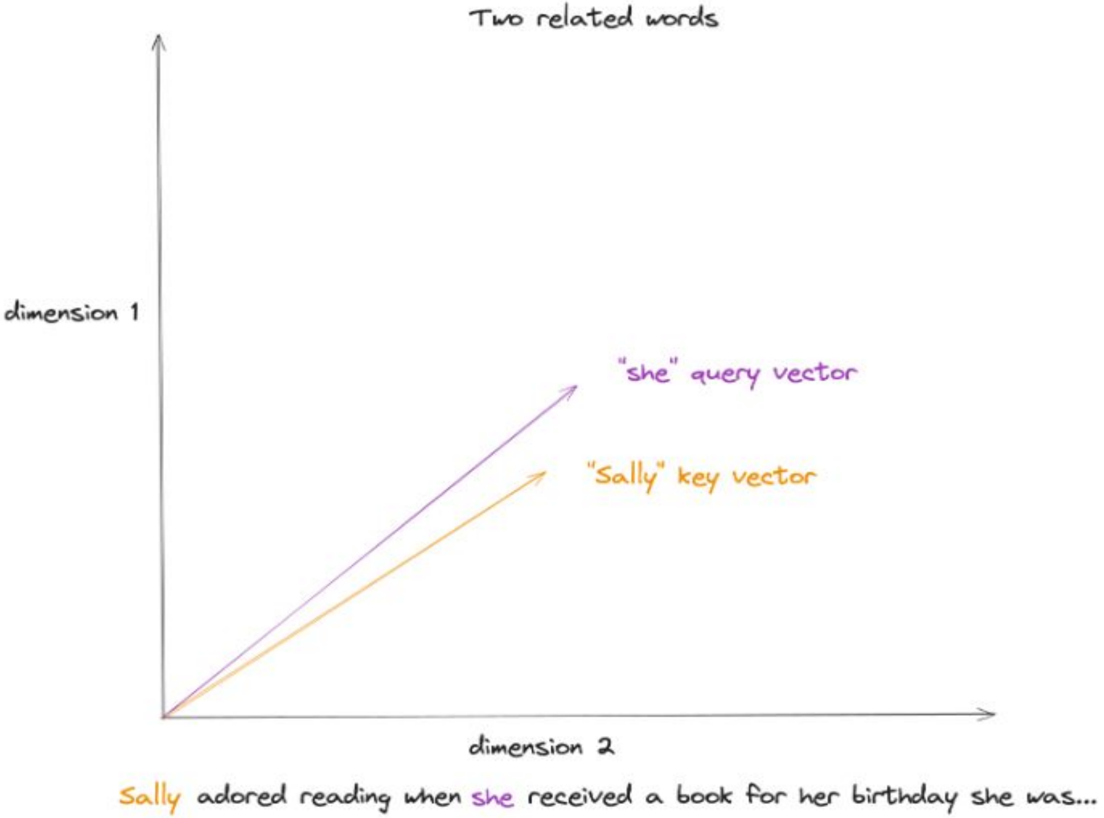

Q
K
V
Q
K
V
Q
K
V
...
Sally
adored
reading
...

Three zoomed-in examples:

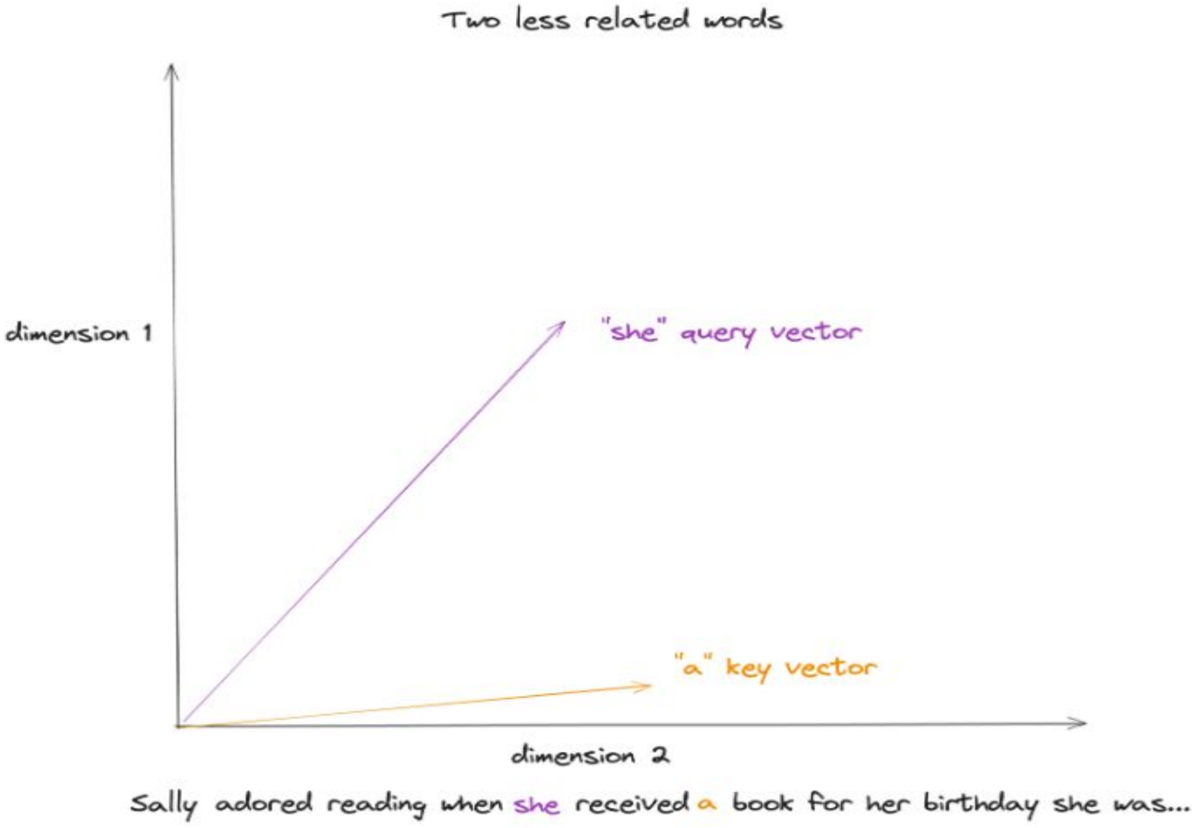
Two words in a sentence that are closely related



The same two words but seen from the other perspective:

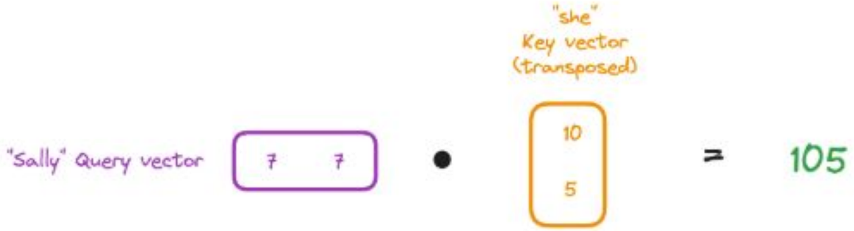


Finally, two words with a weak connection:



Let's look at one dot product

To keep it really simple, we're going to imagine our Q, K and V have only been projected into two dimensions



What happens once we have our dot products?

Dot-product between vectors

	Sally_K	adored_K	reading_K	when_K	she_K
Sally_Q	112	75	60	12	105

Then we scale:

Scaled dot-product between vectors

We divide by $\sqrt{\text{hidden\_dimension}}$ of embedding

In our case the root of $2 \cong 1.4$

	Sally_K	adored_K	reading_K	when_K	she_K
Sally_Q	80	53	43	8.5	75

Finally we apply softmax:

Apply softmax

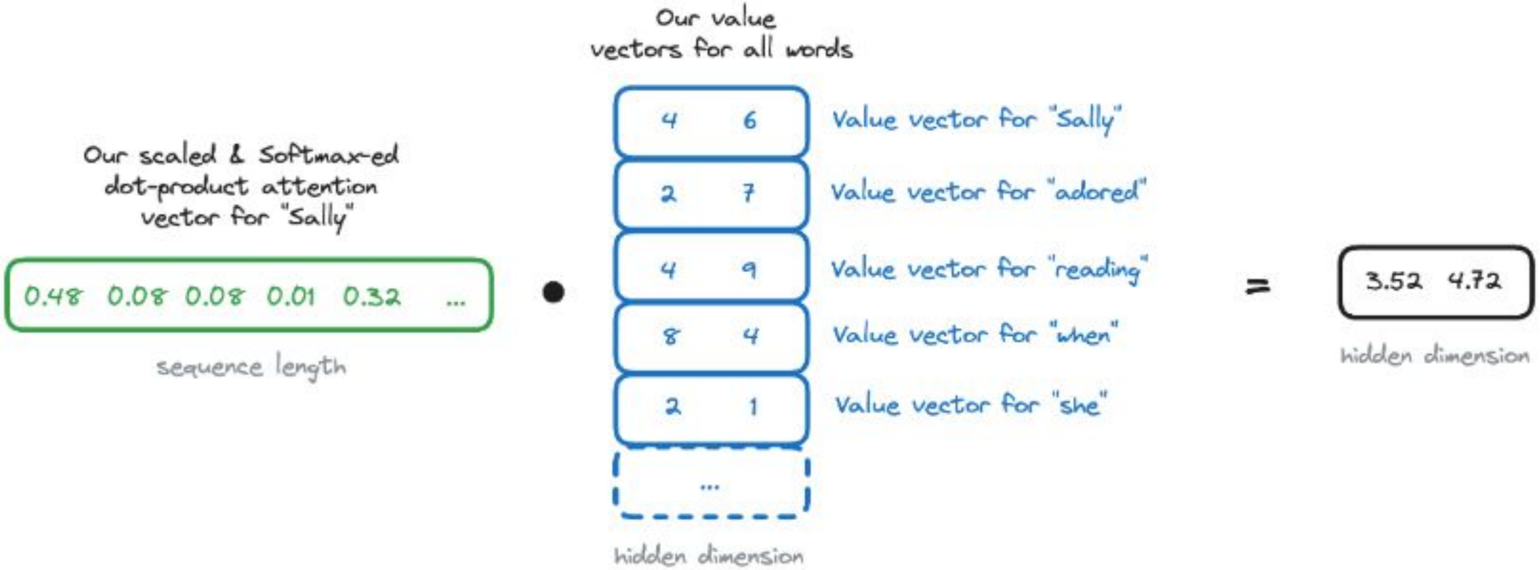
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

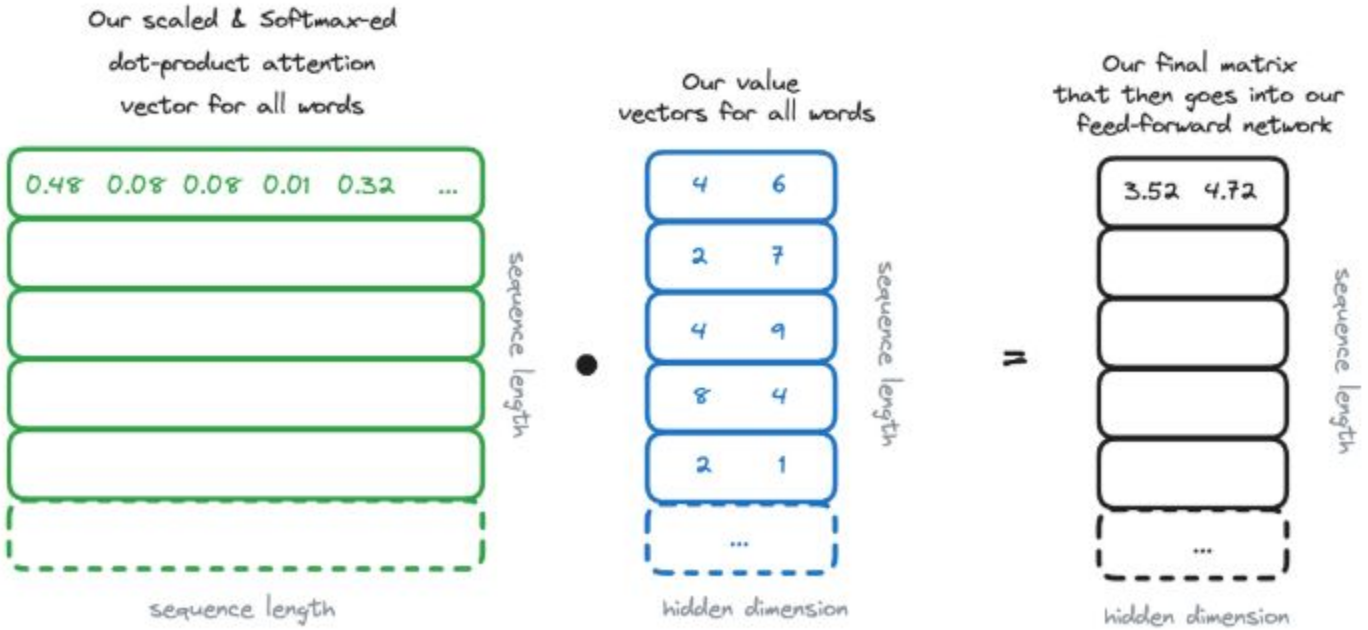
	$Sally_K$	$adored_K$	$reading_K$	$when_K$	she_K
$Sally_Q$	0.48	0.08	0.03	0.01	0.32

We have to do this for each word in our sentence
You can see how this becomes a matrix operation 💪
We get the scaled dot-product attention for all of our Query and Key vectors:

	$Sally_K$	$adored_K$	$reading_K$	$when_K$	she_K	...
$Sally_Q$	0.48	0.08	0.03	0.01	0.32	...
$adored_Q$	0.04	0.03	0.52	0.3	0.01	...
$reading_Q$	0.01	0.02	0.13	0.86	0.01	...
$when_Q$	0.01	0.03	0.02	0.58	0.01	...
she_Q	0.13	0.12	0.02	0.01	0.04	...
...	...	...	...	...	...	

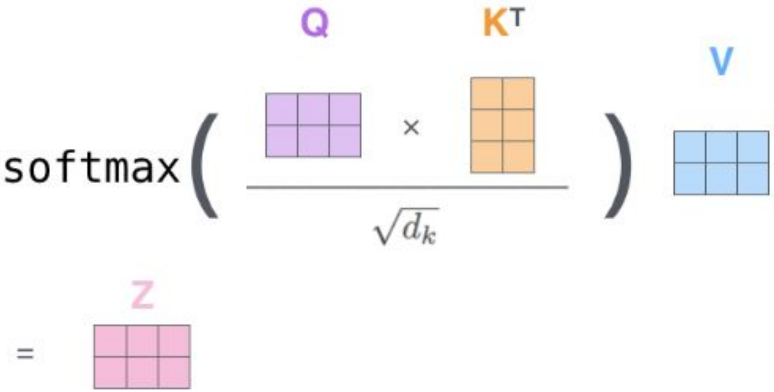
And then multiply our "similarity score" with all of our Value vectors





Our scaled & Softmax-ed dot-product attention vector for all words
0.48 0.08 0.08 0.01 0.32 ...
sequence length
sequence length
Our value vectors for all words
4 6
2 7
4 9
8 4
2 1
...
sequence length
hidden dimension
=
Our final matrix that then goes into our feed-forward network
3.52 4.72
...
sequence length
hidden dimension

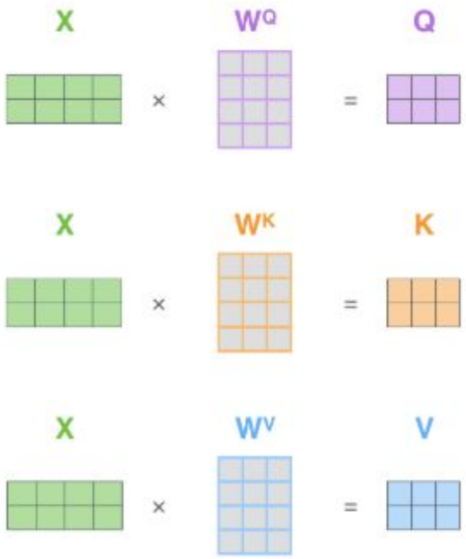
So really the entire thing can be written like this:



We are done with our multiplications
Now we just need to normalize and pass through to the feed-forward neural network
The neural network will output vectors of our original embedding dimension (e.g. 768)

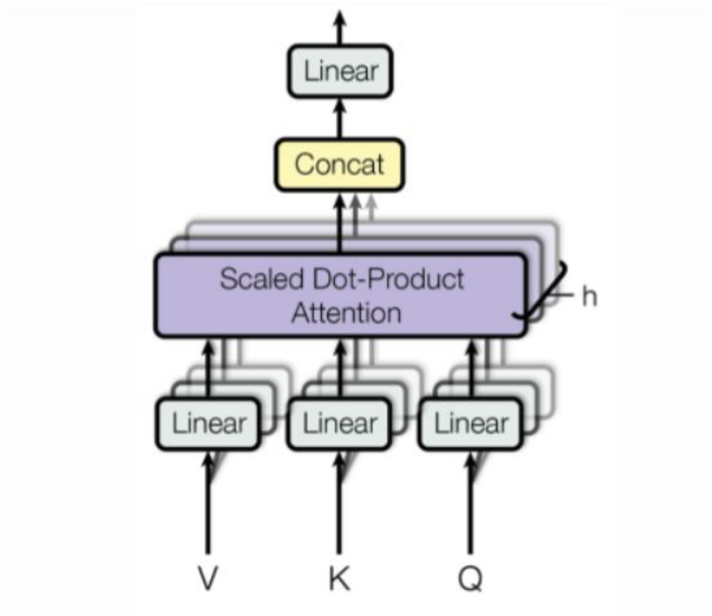
🤓 Computing one set of all of these Q, K, V multiplications and processes is what we call "single-headed attention".

When we are doing our initial linear projections (used to create the Q, K and V vectors) we can express these operations as matrices of weights too!



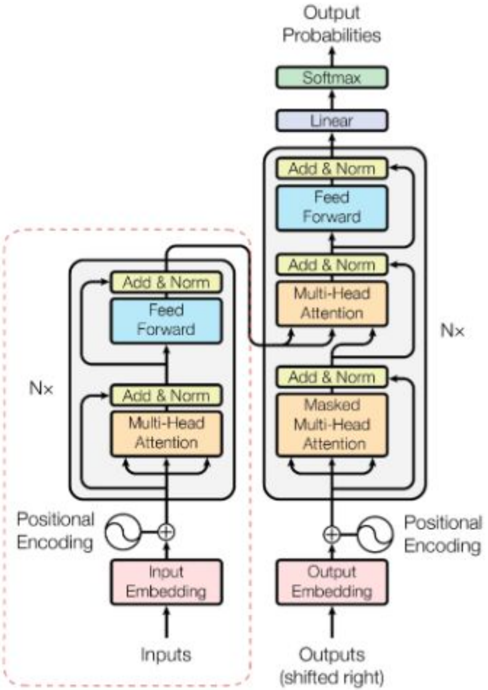
Multi-headed?

- We can use multiple heads to split up and analyze different parts of our embedding
- Each can focus a different part of the embedding eg. working on semantic vs syntactic features of our sentences 🤯 🤯 🤯



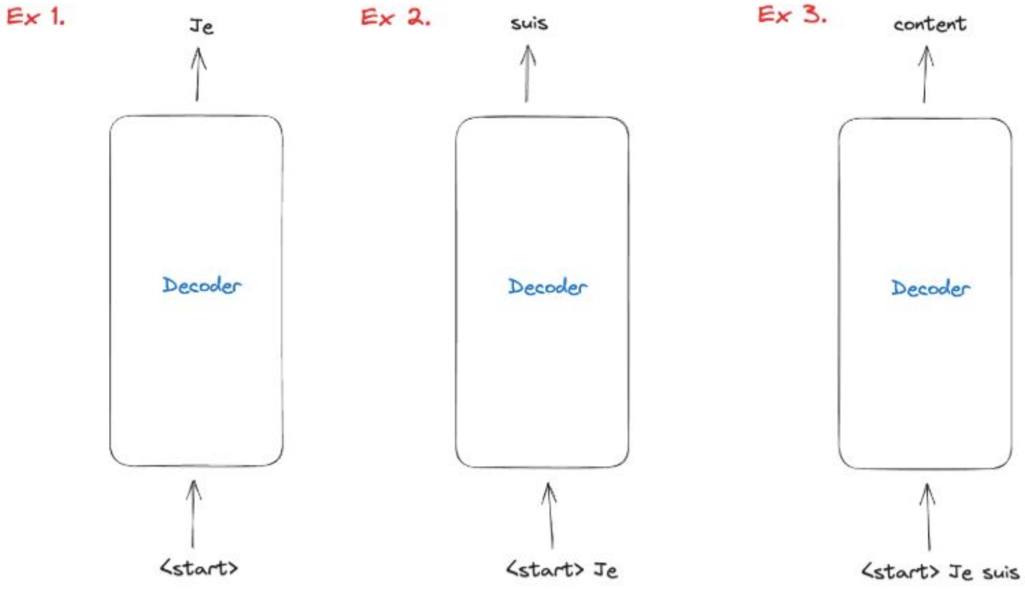
Let's check in with our original diagram:

One encoder block

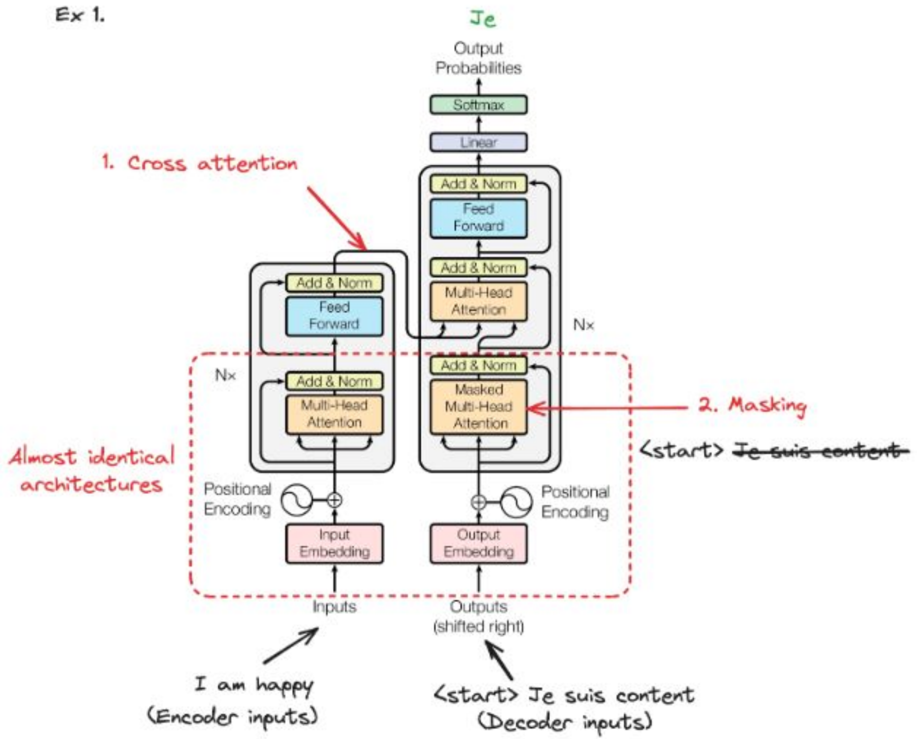


What about the decoder?

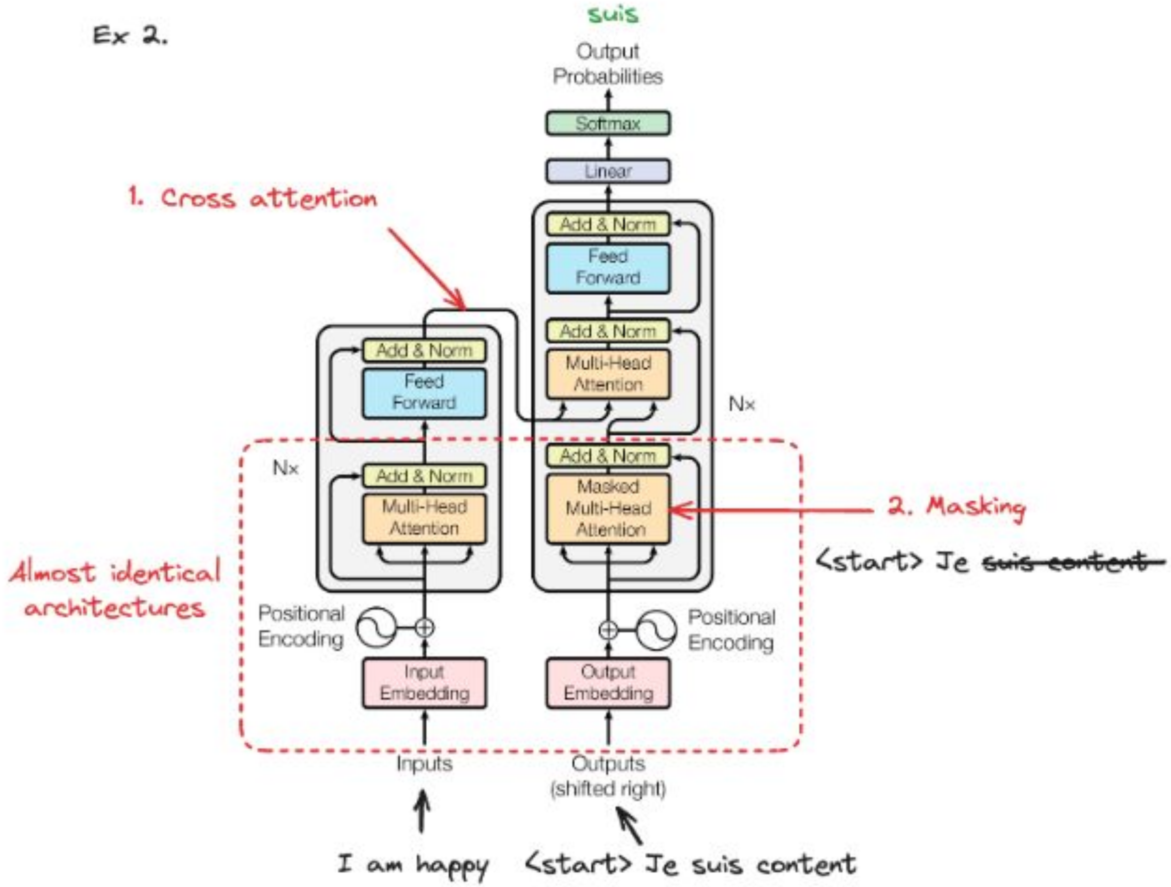
At the highest level view, its job is to choose the **most likely next token**:



How does it do this? And what happened to all the work the encoder did?

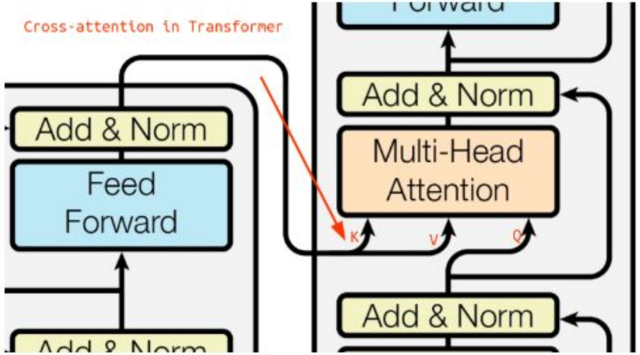


Ex 2.



suis
Output Probabilities
Softmax
Linear
1. Cross attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
2. Masking
<start> Je ~~suis content~~
Almost identical architectures
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
I am happy
<start> Je suis content

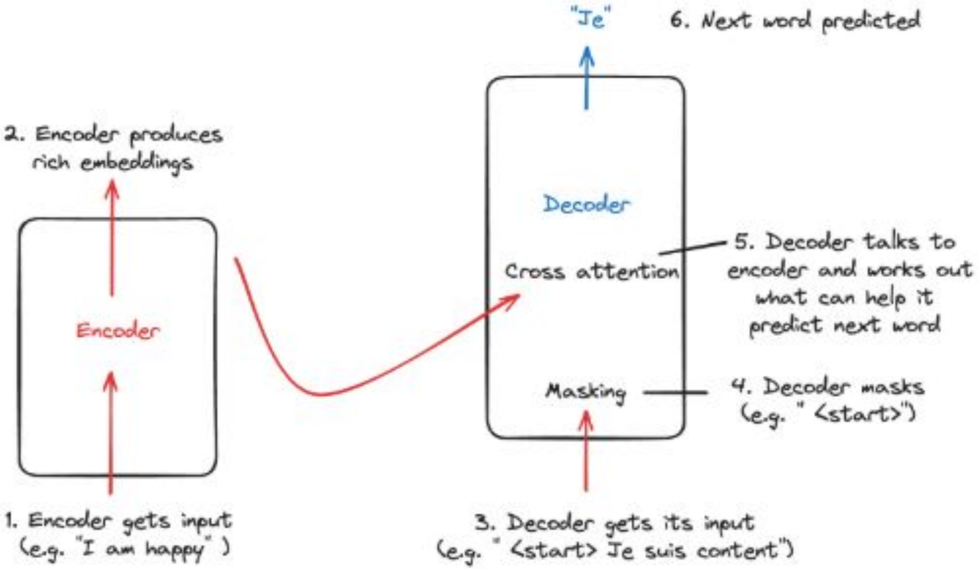
Cross attention



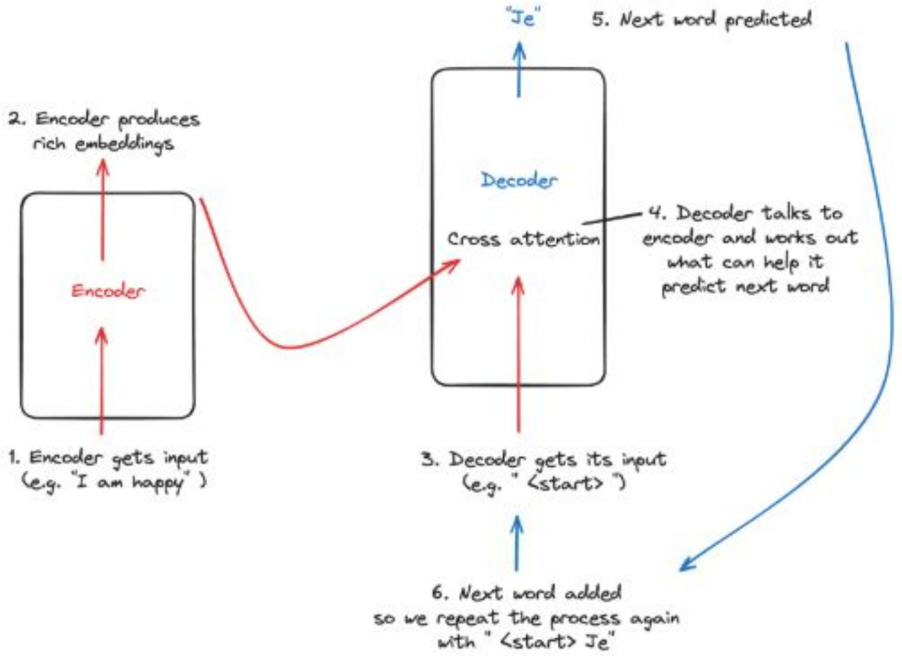
Attention between encoder and decoder (a.k.a. cross-attention):

- **Self-attention** operates within a single sequence and captures the relationships between tokens within that sequence.
- **Cross-attention** operates between two different sequences and captures the relationships between tokens from the source sequence and tokens from the target sequence, allowing the model to generate relevant output based on the information in the source sequence.

Let's recap the training process at a high level



So what does inference look like?



That's all there is to it!