香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMP 4901B
Large Language Models

Transformers

Junxian He

Sep 17, 2025

Please Download HKUST iLearn in Your Mobile Phone or iPad

HKUST iLearn

HKUST Learning

Designed for iPad. Not verifi

HKUST iLearn

Canvas

This will open the 'Canvas Student' app which provides an easy access to the online content of your courses at HKUST - watch videos, post to discussions, submit quizzes, etc.

SFQ

Allows you to complete the Student Feedback Questionnaire for all your courses at HKUST on the move.

iPRS

Enables you to quickly respond to questions or polls created by your instructor in class.

We are going to use iPRS to do quizzes in the future

Audit

pass

group project

audit

enrolled

attendance

10%

Recap: Autoregressive Language Models

$$
\begin{aligned}
p(\text{the, mouse, ate, the, cheese}) = {} & p(\text{the}) \\
& p(\text{mouse} \mid \text{the}) \\
& p(\text{ate} \mid \text{the, mouse}) \\
& p(\text{the} \mid \text{the, mouse, ate}) \\
& p(\text{cheese} \mid \text{the, mouse, ate, the}).
\end{aligned}
$$

$$p(x_1, x_2, \ldots, x_I) = \prod_{i=1}^{I} p(x_i \mid x_{1:i-1})$$

Next Word Context

Recap: Neural Language Models

Recap: Neural Language Models

Neural language models are typically autoregressive

Recap: Neural Language Models

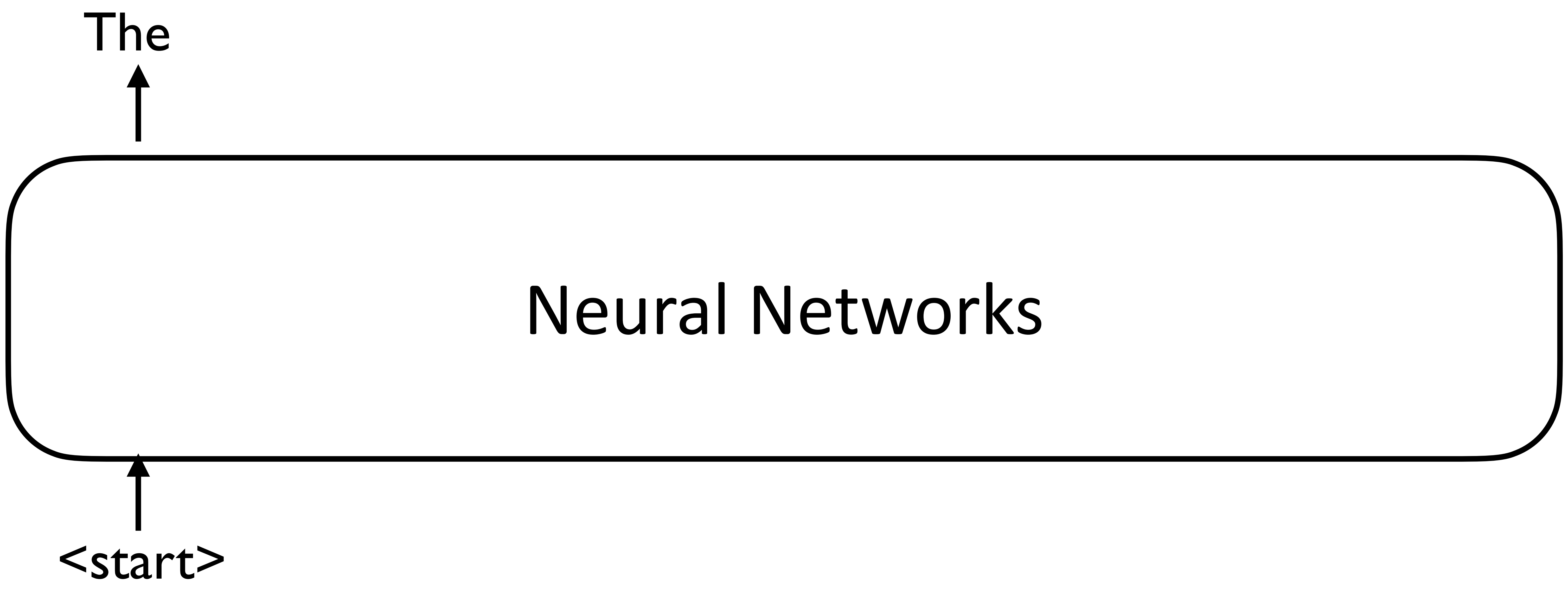
Neural language models are typically autoregressive

Data: "The mouse ate the cheese ."

Recap: Neural Language Models

Neural language models are typically autoregressive

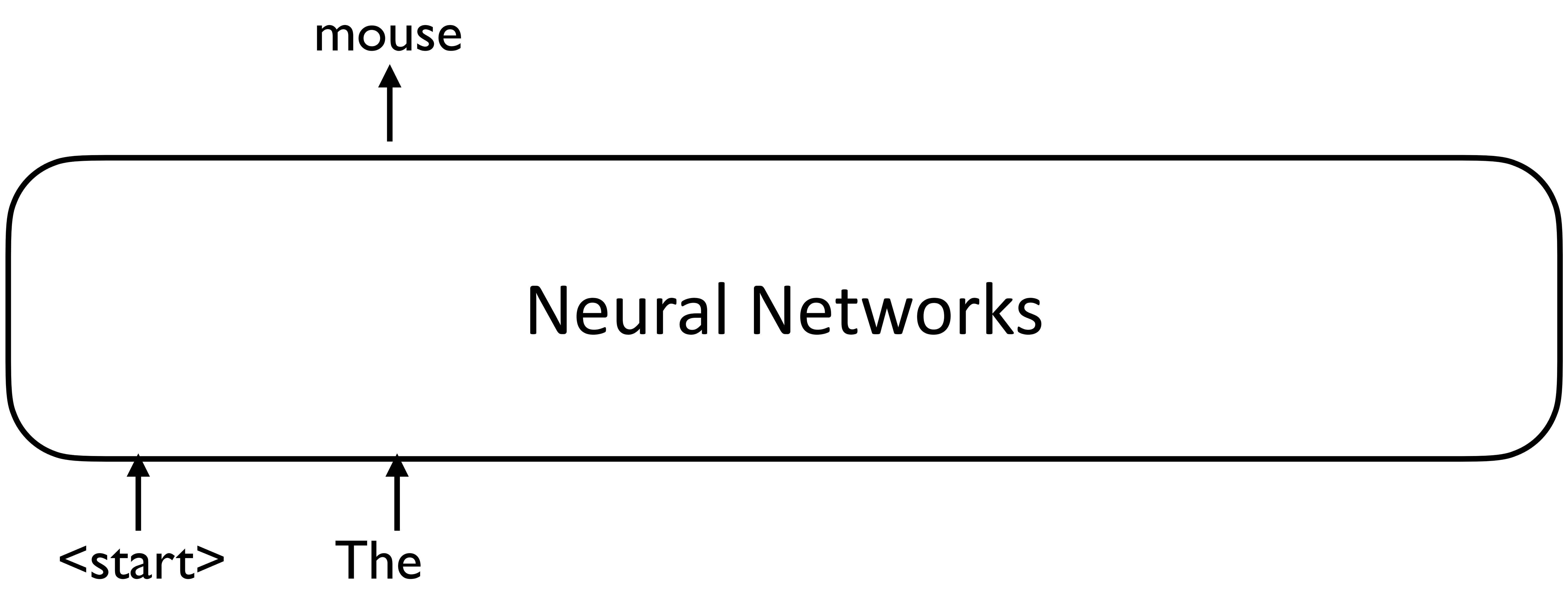
Data: “The mouse ate the cheese .”



Recap: Neural Language Models

Neural language models are typically autoregressive

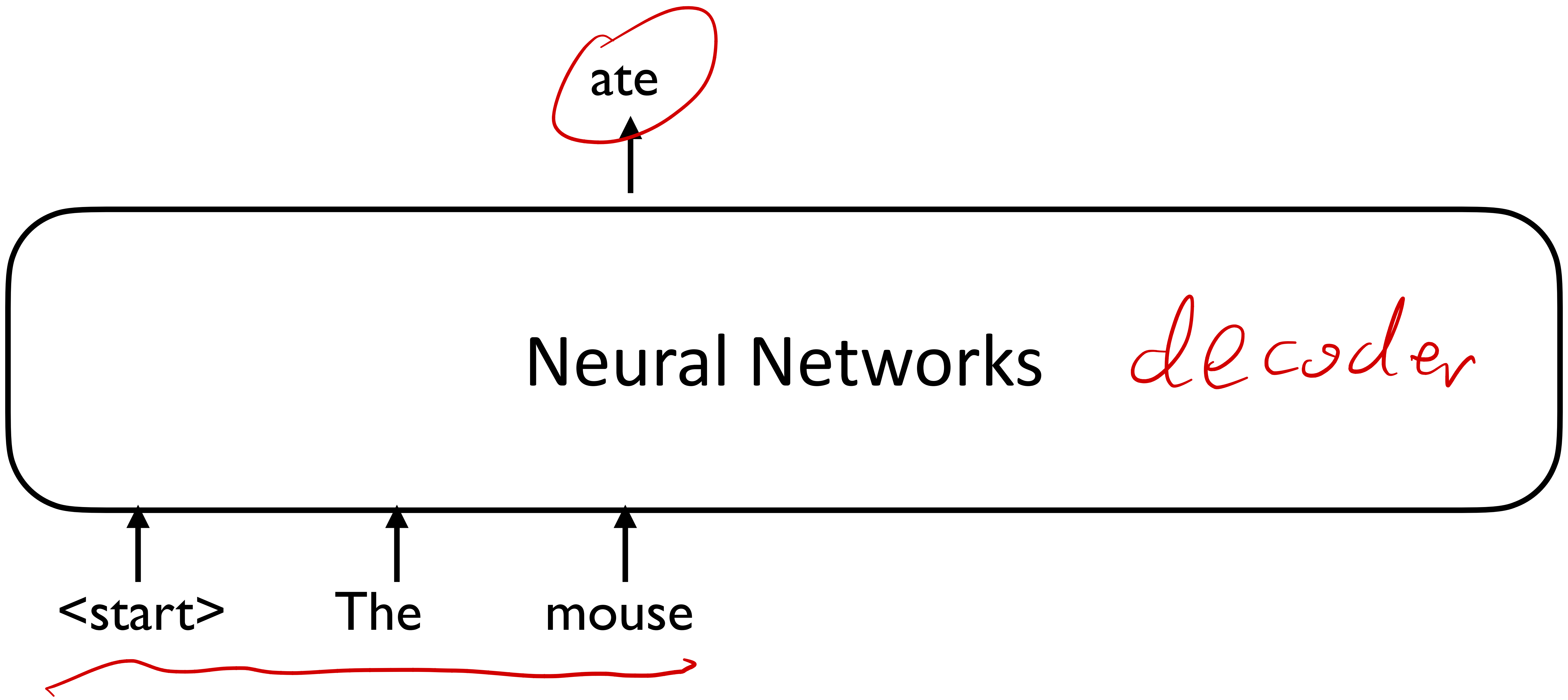
Data: "The mouse ate the cheese ."



Recap: Neural Language Models

Neural language models are typically autoregressive

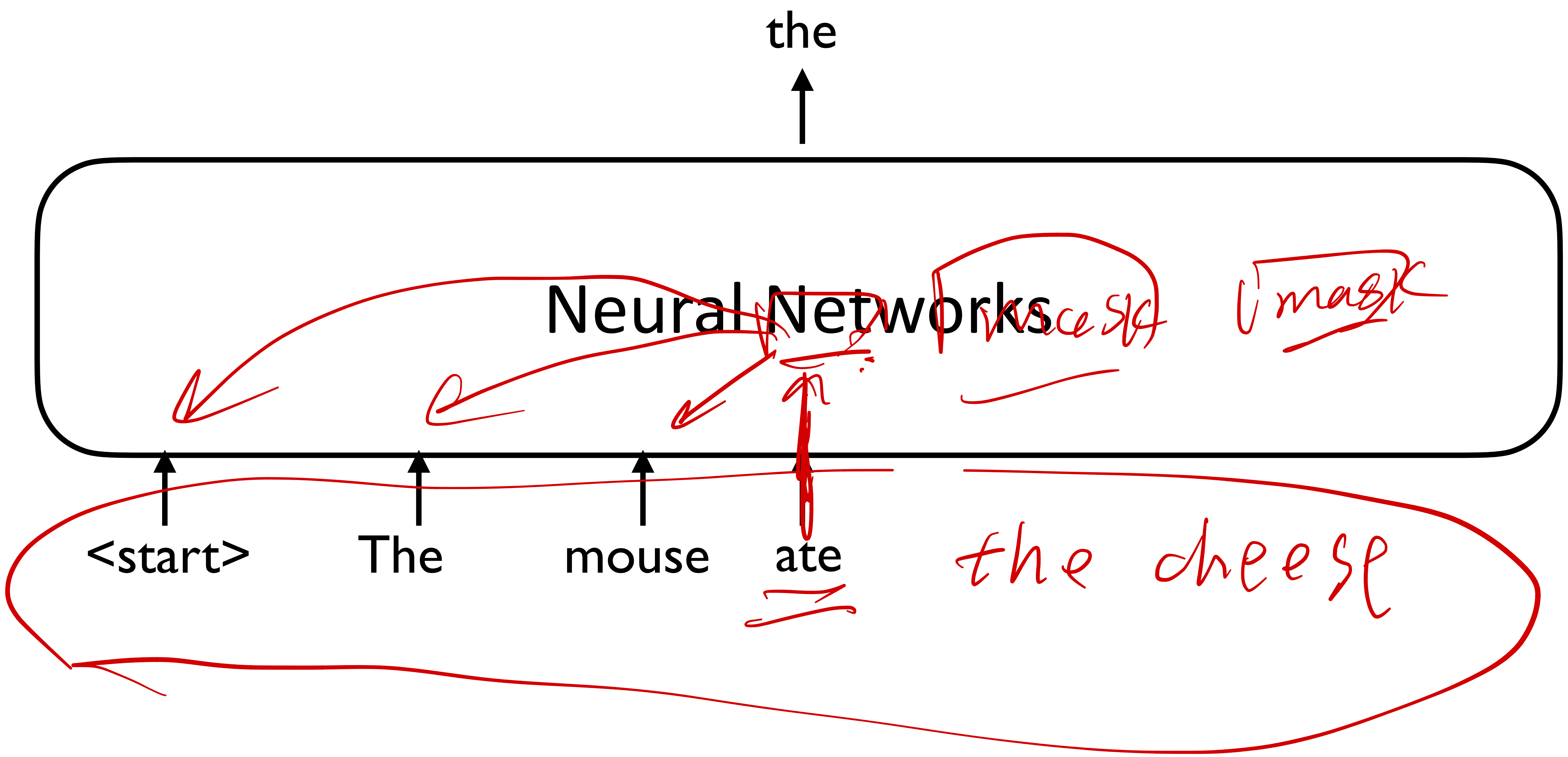
Data: "The mouse ate the cheese ."



Recap: Neural Language Models

Neural language models are typically autoregressive

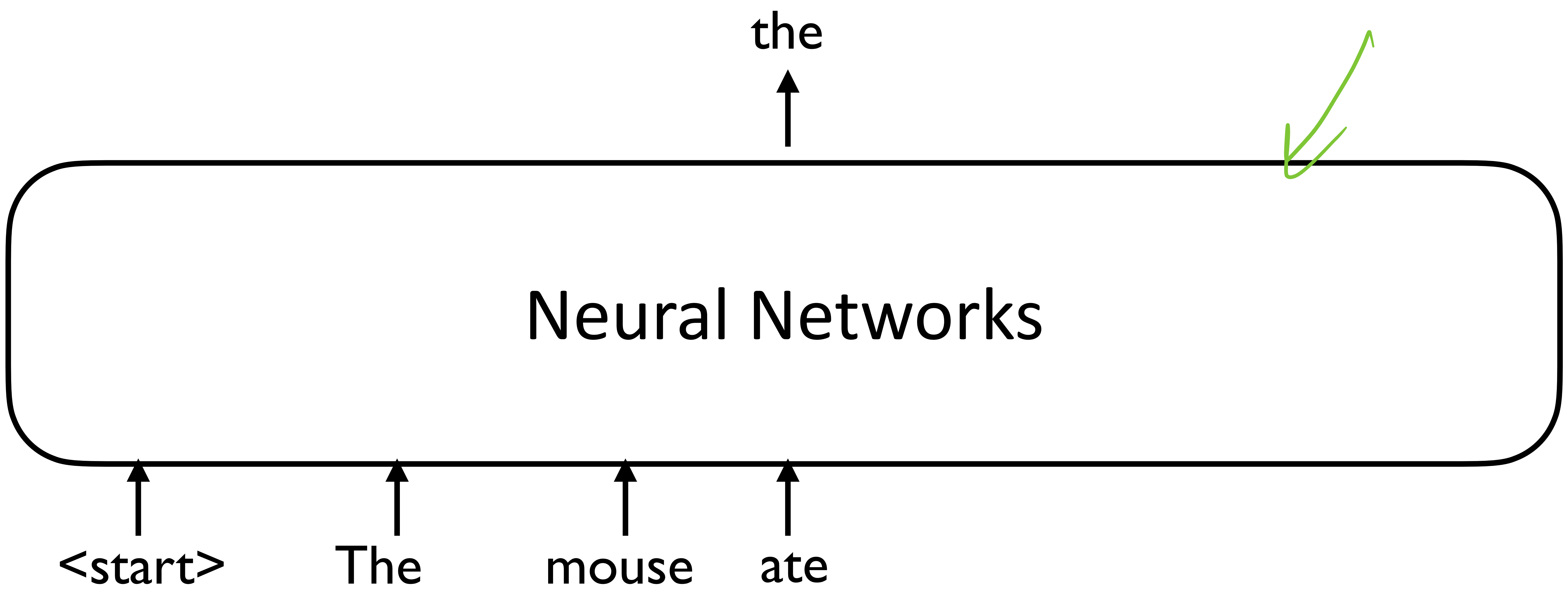
Data: “The mouse ate the cheese .”



Recap: Neural Language Models

Neural language models are typically autoregressive

Data: "The mouse ate the cheese ."

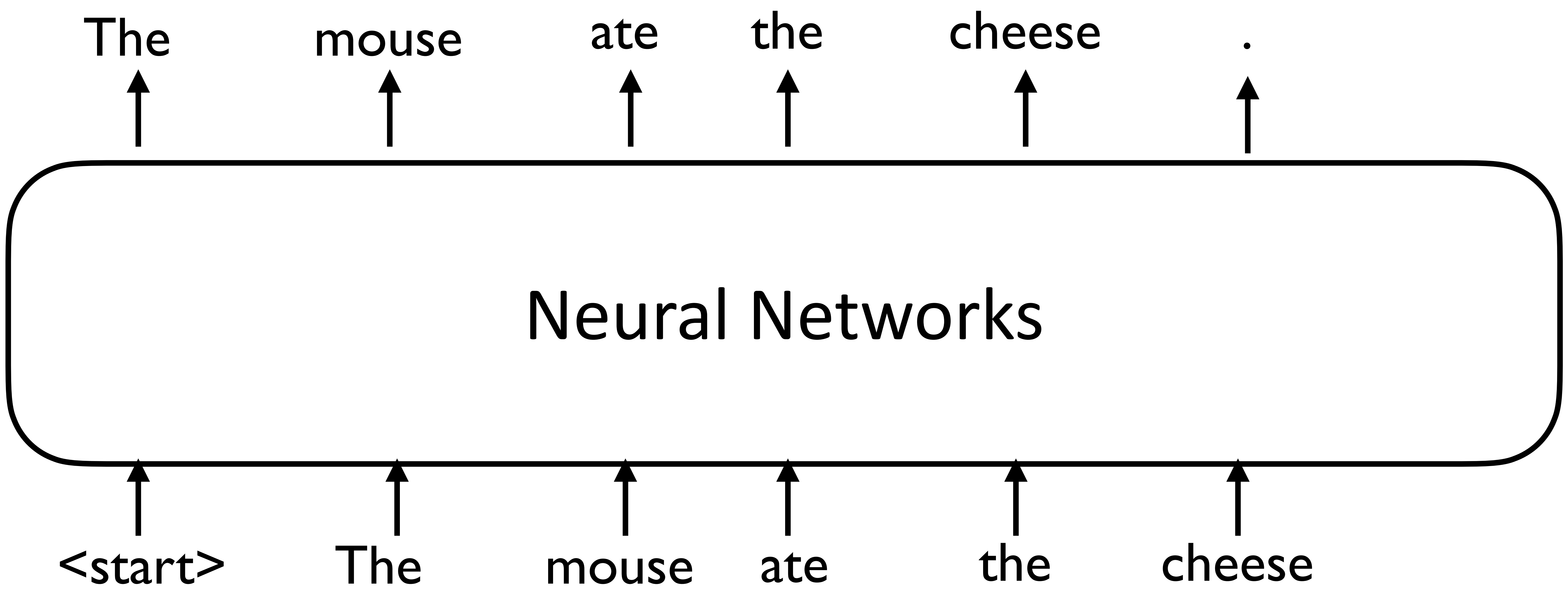


We can compute the loss on every token in parallel

Recap: Neural Language Models

Neural language models are typically autoregressive

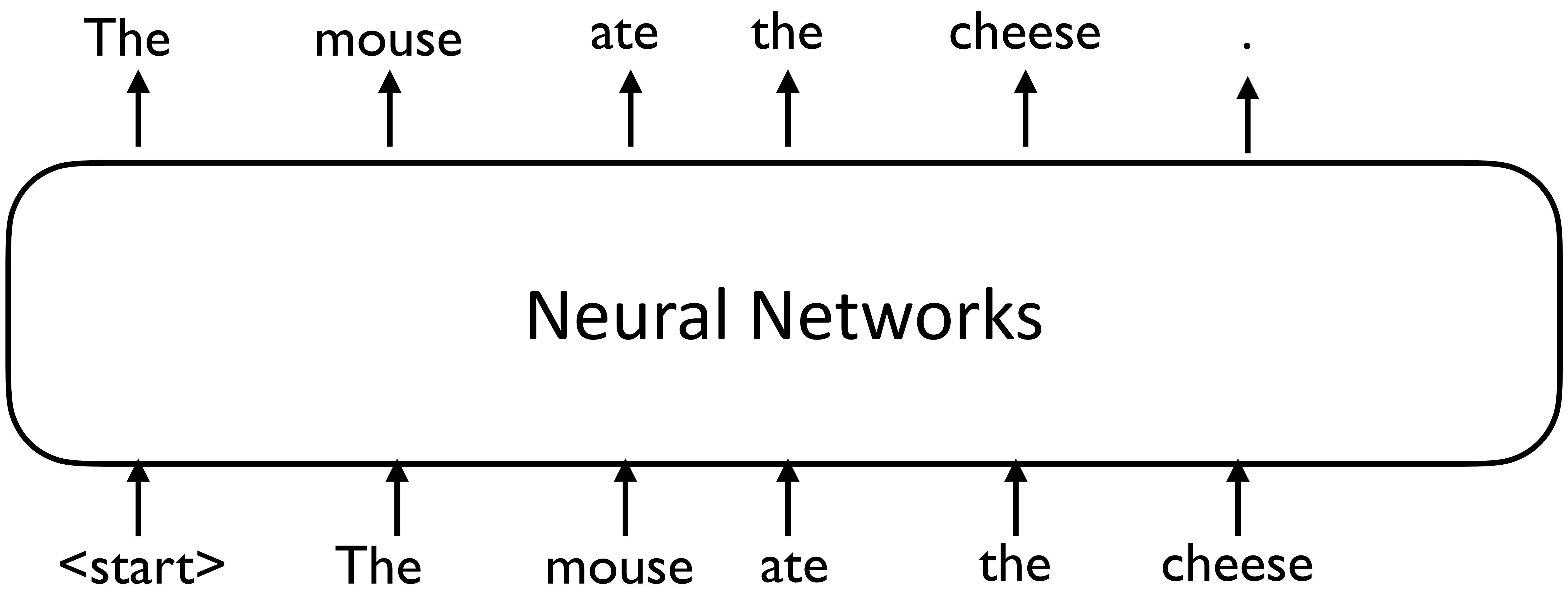
Data: “The mouse ate the cheese .”



Recap: Neural Language Models

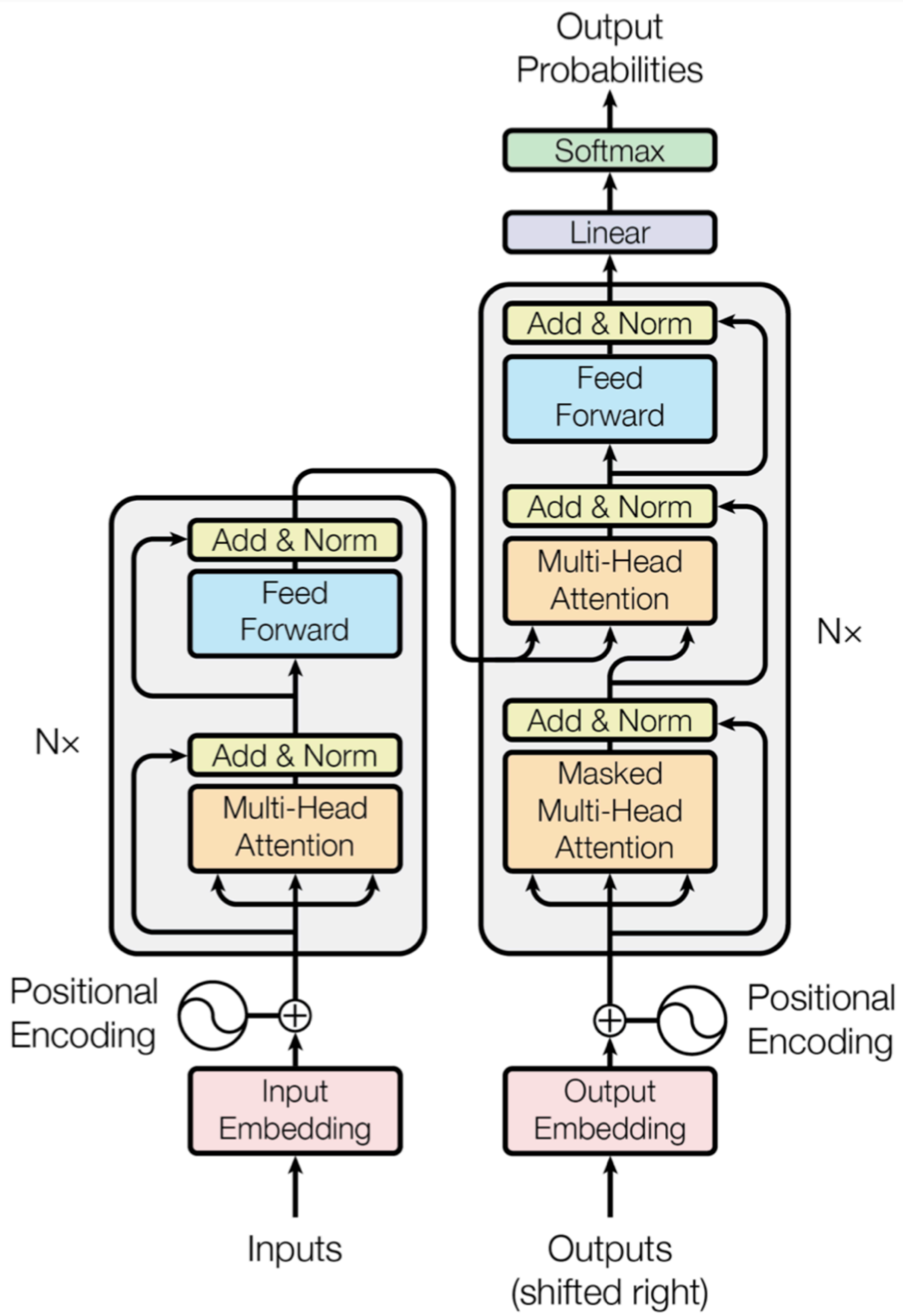
Neural language models are typically autoregressive

Data: "The mouse ate the cheese ."



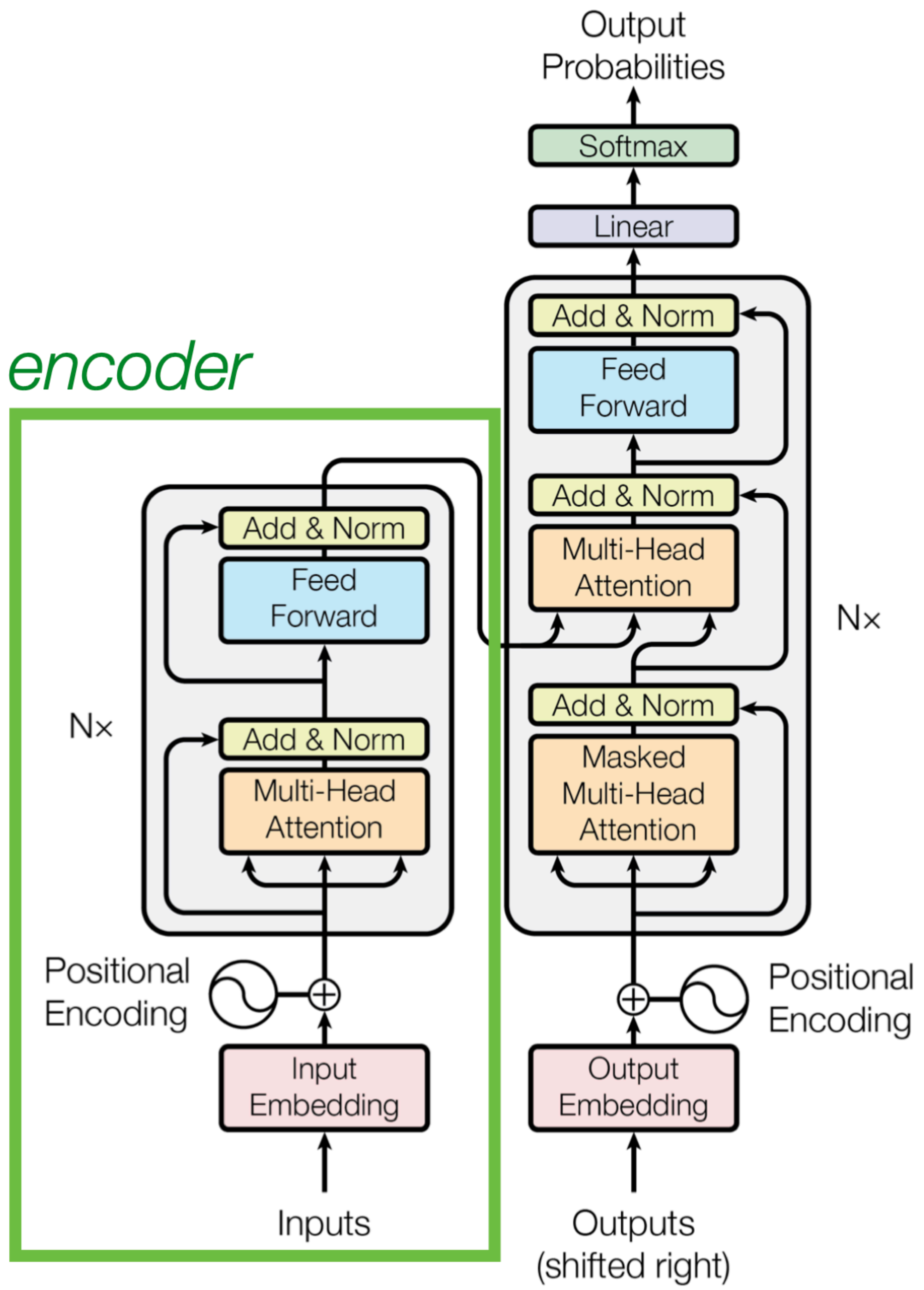
Each prediction only sees the inputs on its left

Recap: Transformer

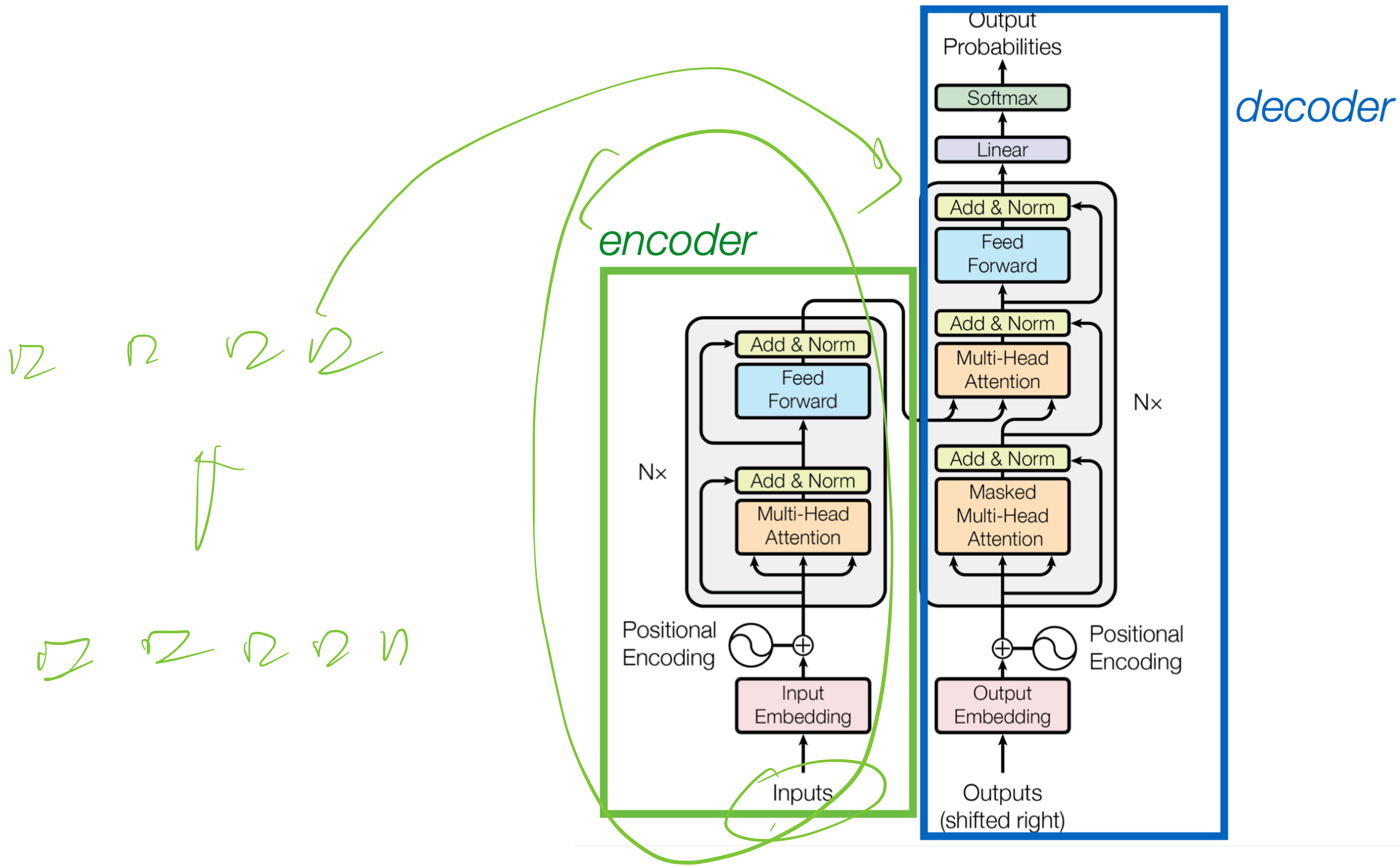


Vaswani et al. Attention is All You Need. NeurIPS 2017.

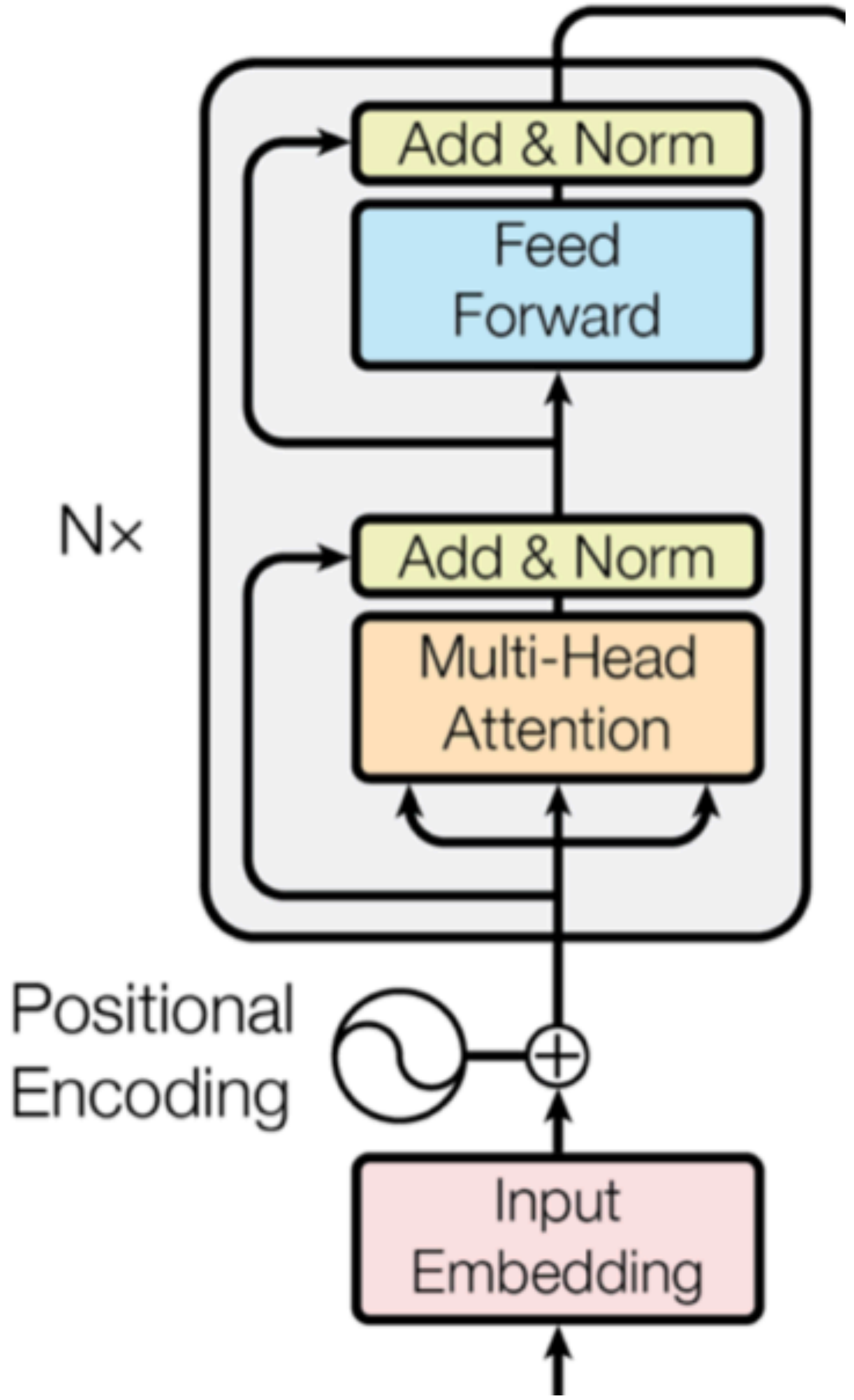
Recap: Encoder



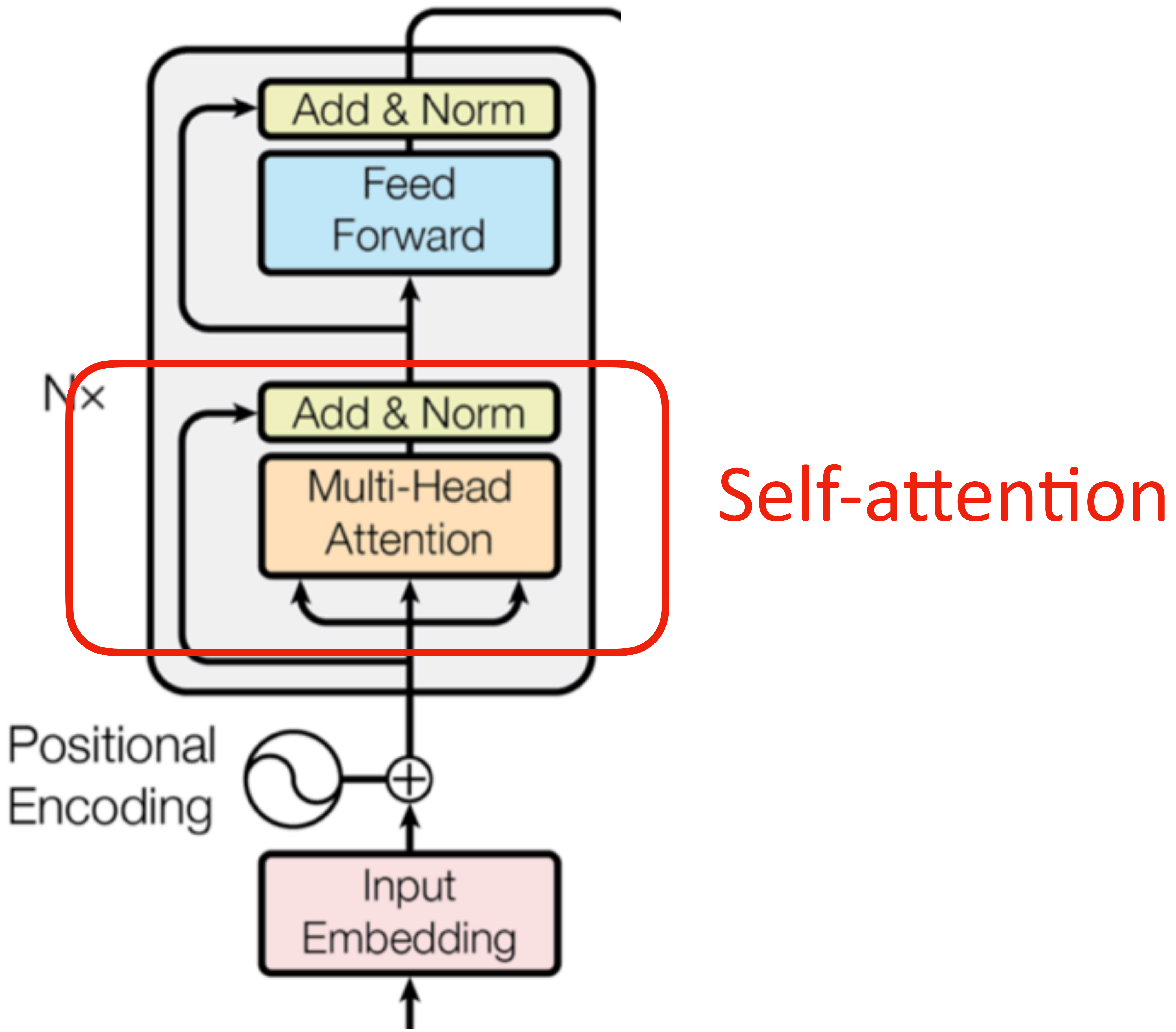
Recap: Decoder



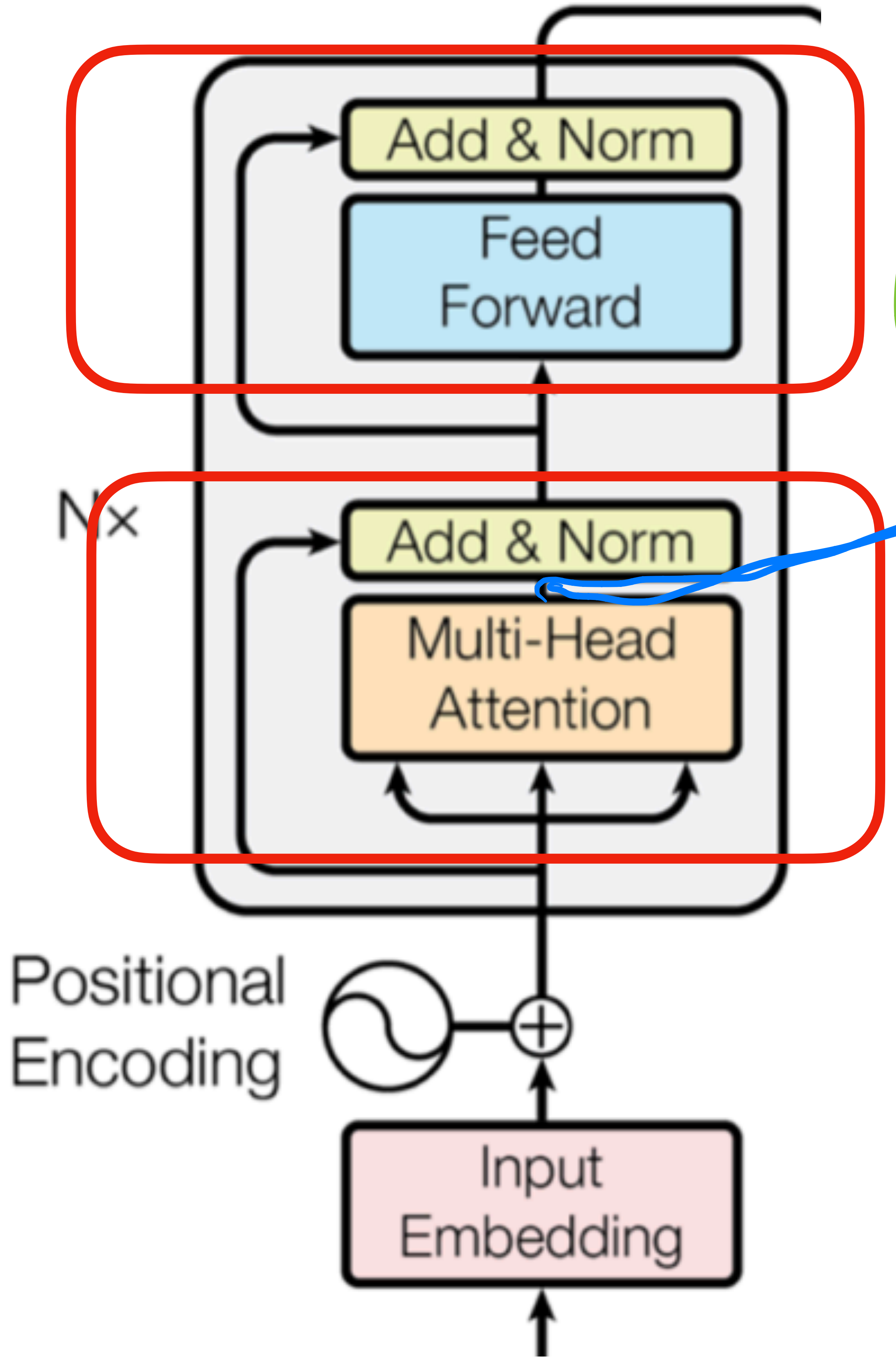
Recap: Transformer Encoder



Recap: Transformer Encoder



Recap: Transformer Encoder



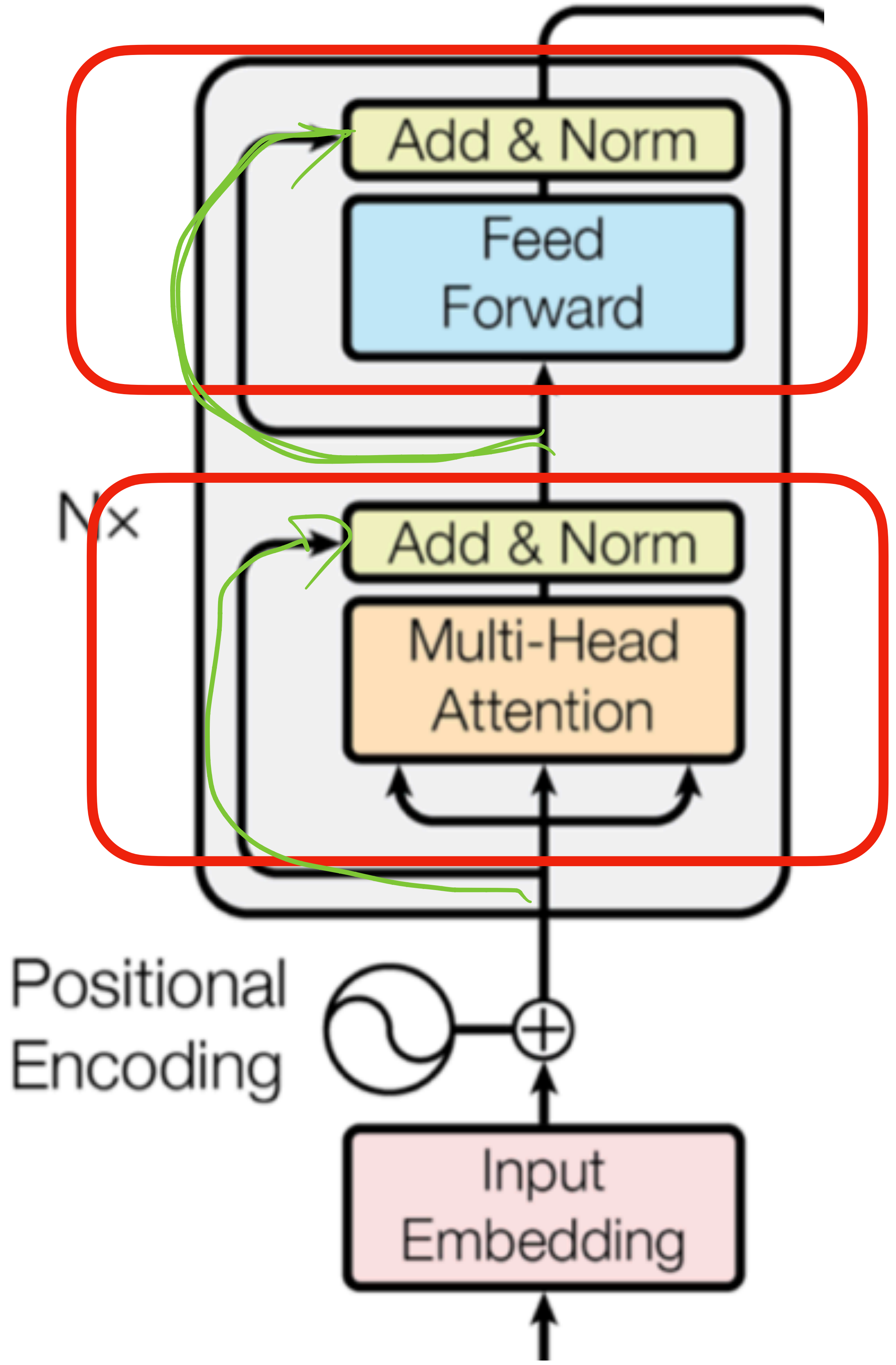
MLP

Self-attention

attn output

Recap: Transformer Encoder

Residual connection

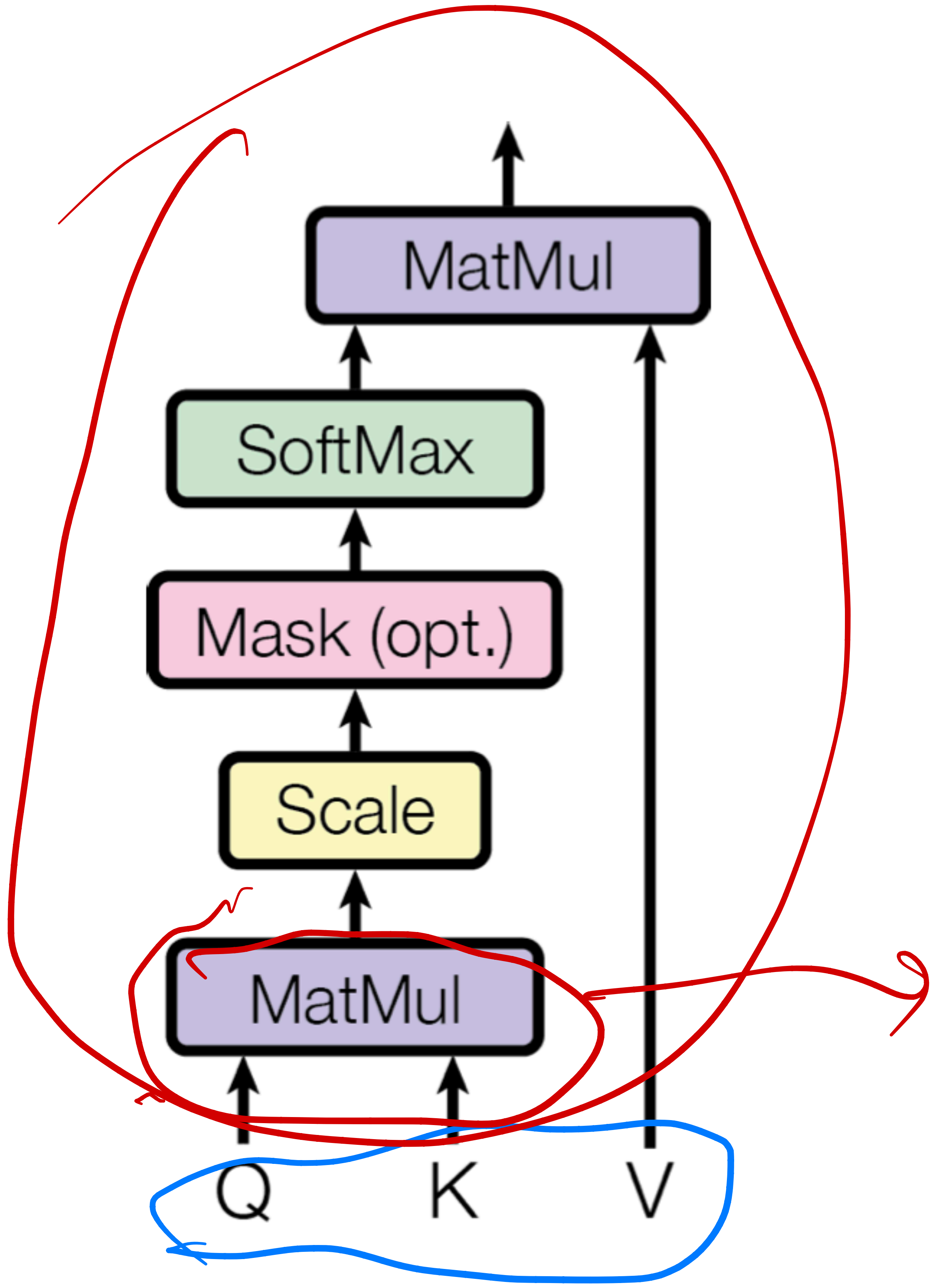


MLP

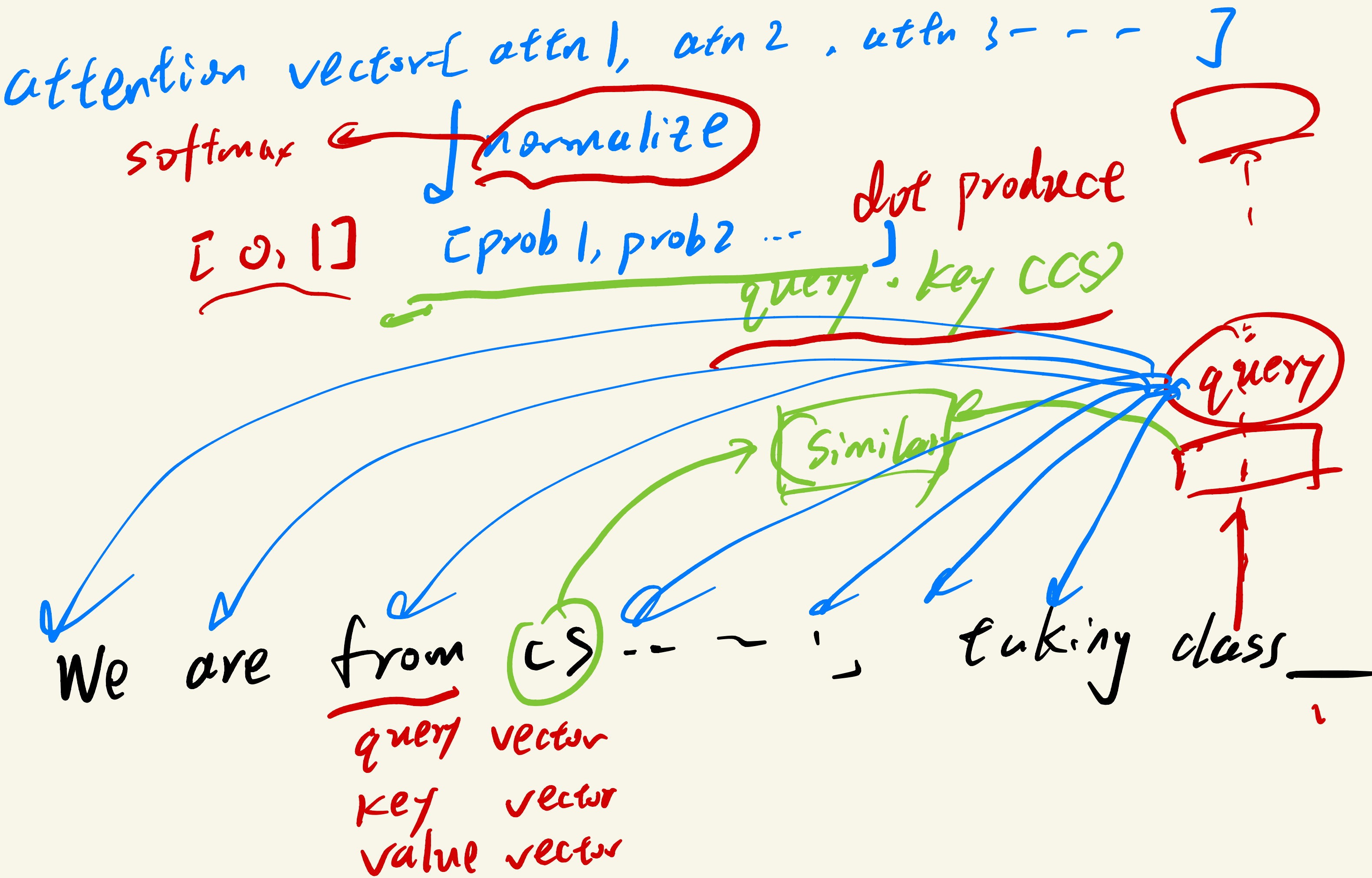
Self-attention

Recap: What is Attention

Scaled Dot-Product Attention



Q: Query
K: key
V: value



attention vector = [attn 1, atn 2 , attn 3 - - -]
softmax
normalize
[0, 1]
[prob 1, prob2 ...]
dot produce
query · key CCS
query
similar
We are from CS -- ~ , taking class ___
query vector
key vector
value vector

$$\text{softmax}(t_1, t_2, t_3 \cdots t_n)$$

$$= \left[\frac{\exp(t_1)}{\sum_i \exp(t_i)}, \frac{\exp(t_2)}{\sum_i \exp(t_i)} \cdots\cdots \right]$$

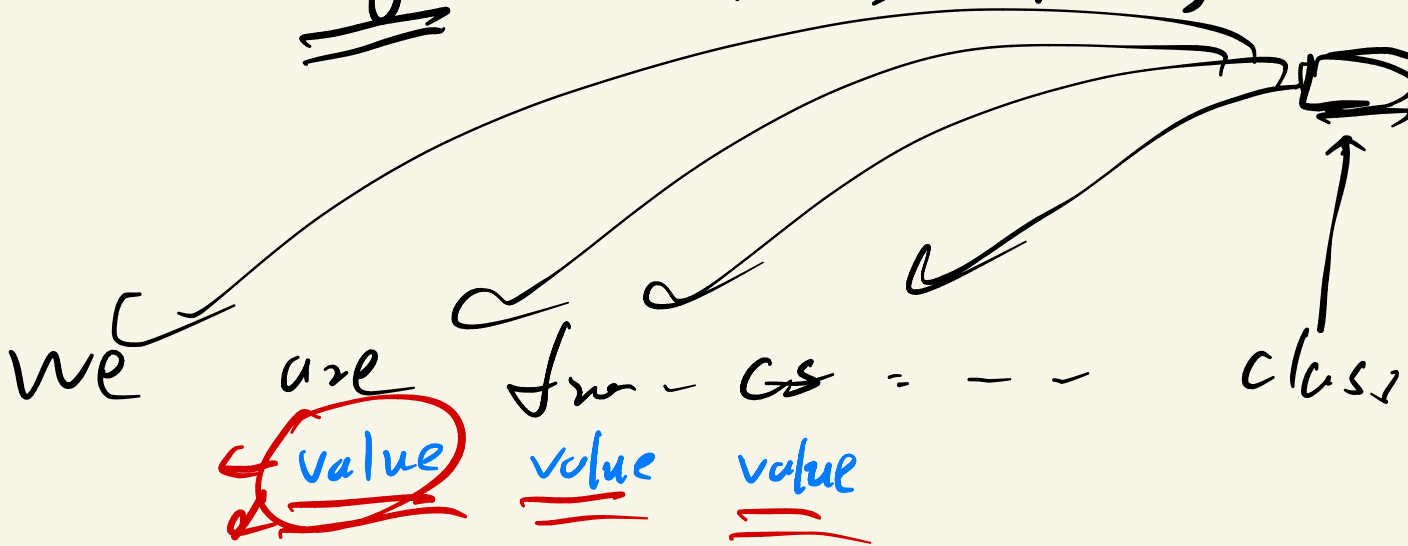
probability

$\overrightarrow{effect} = prob1 \cdot value1 + prob2 \cdot value2 + \dots$

how much it affects

not how it affects

attn weight = [prob1, prob2, ... prob n]



high level

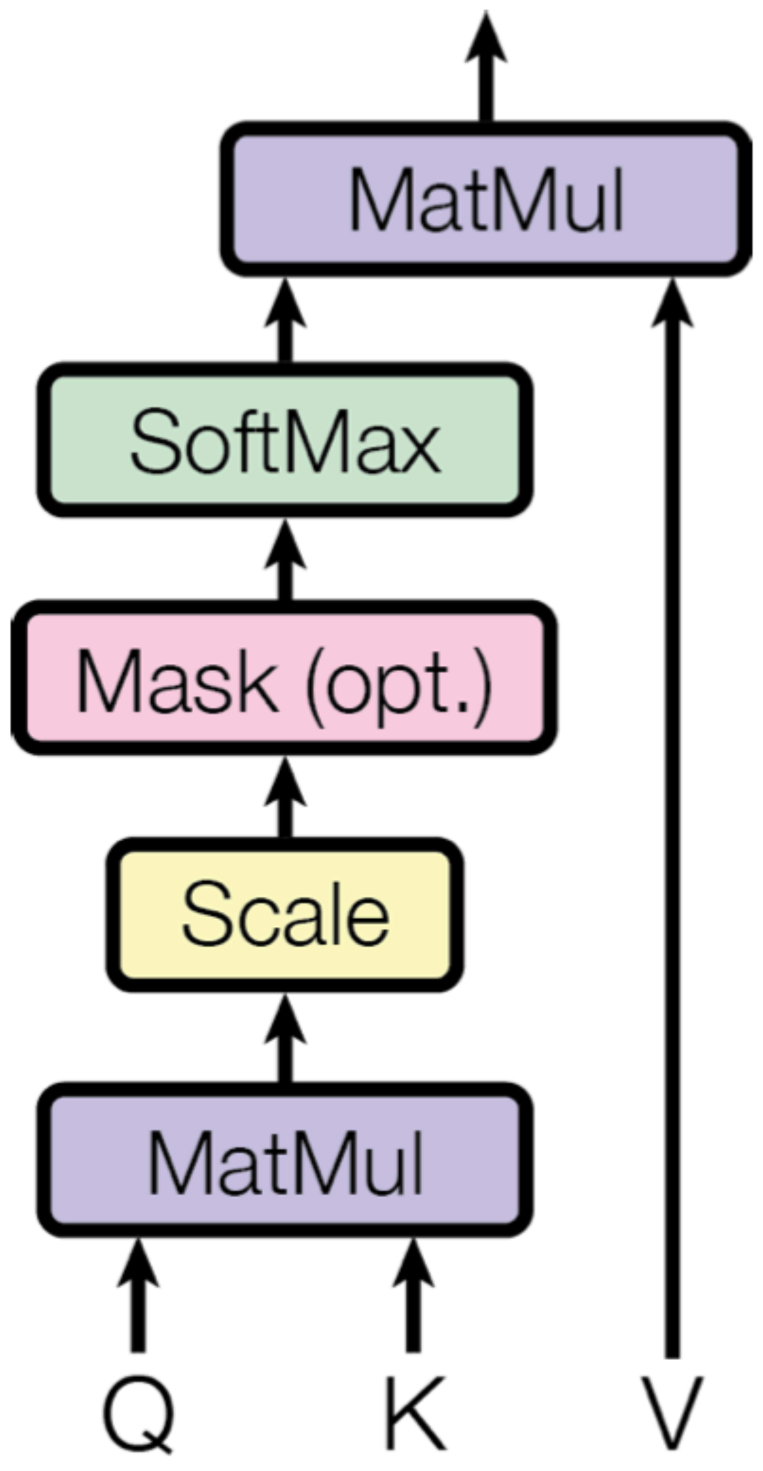
attention weight

effect (attention output)
(weighted addition of values)

Recap: What is Attention

$Q \in R^{n \times d}$ $\quad K \in R^{m \times d}$ $\quad V \in R^{m \times d}$

Scaled Dot-Product Attention



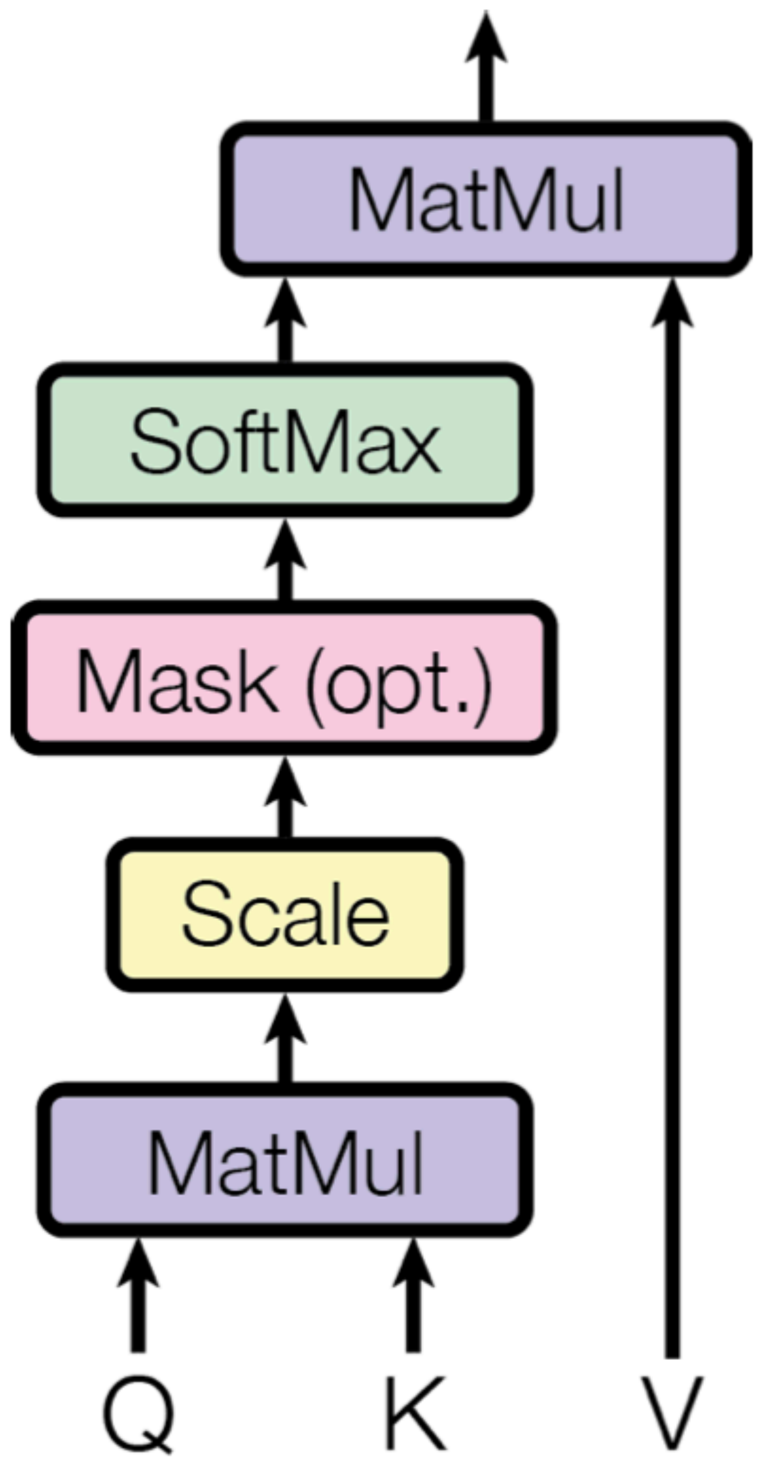
Q: Query
K: key
V: value

Recap: What is Attention

$$Q \in R^{n \times d} \qquad K \in R^{m \times d} \qquad V \in R^{m \times d}$$

We have n queries, m (key, value) pairs

Scaled Dot-Product Attention



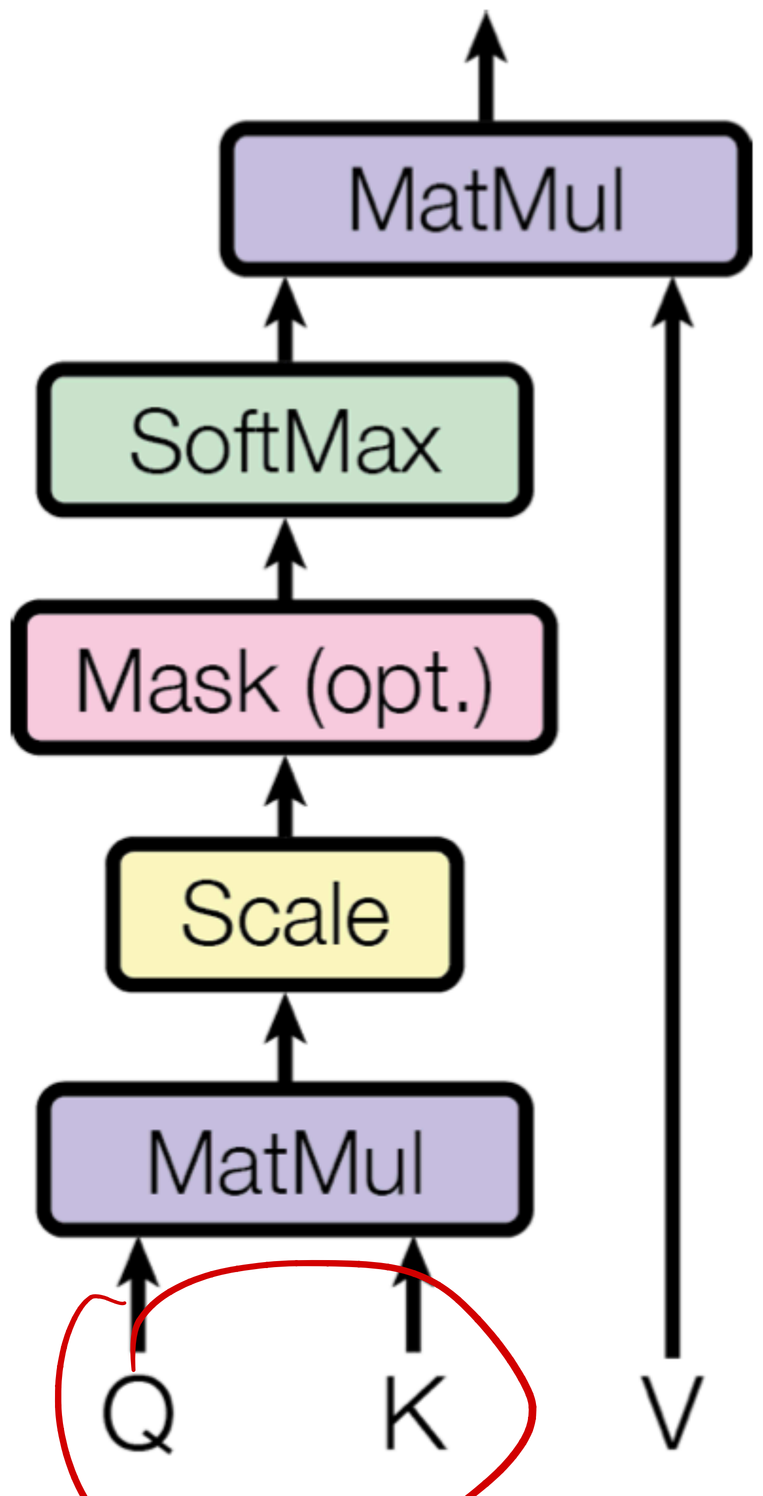
Q: Query
K: key
V: value

Recap: What is Attention

$Q \in R^{n \times d}$ $K \in R^{m \times d}$ $V \in R^{m \times d}$

We have n queries, m (key, value) pairs

Scaled Dot-Product Attention



Attention weight = softmax(QK^T)

pairwise attn score

QK

QK^T

Q: Query
K: key
V: value

$Q \in R^{n\times d}$ $\quad K \in R^{m\times d}$ $\quad V \in R^{m\times d}$

$n \neq m$

self attention: $n = m$

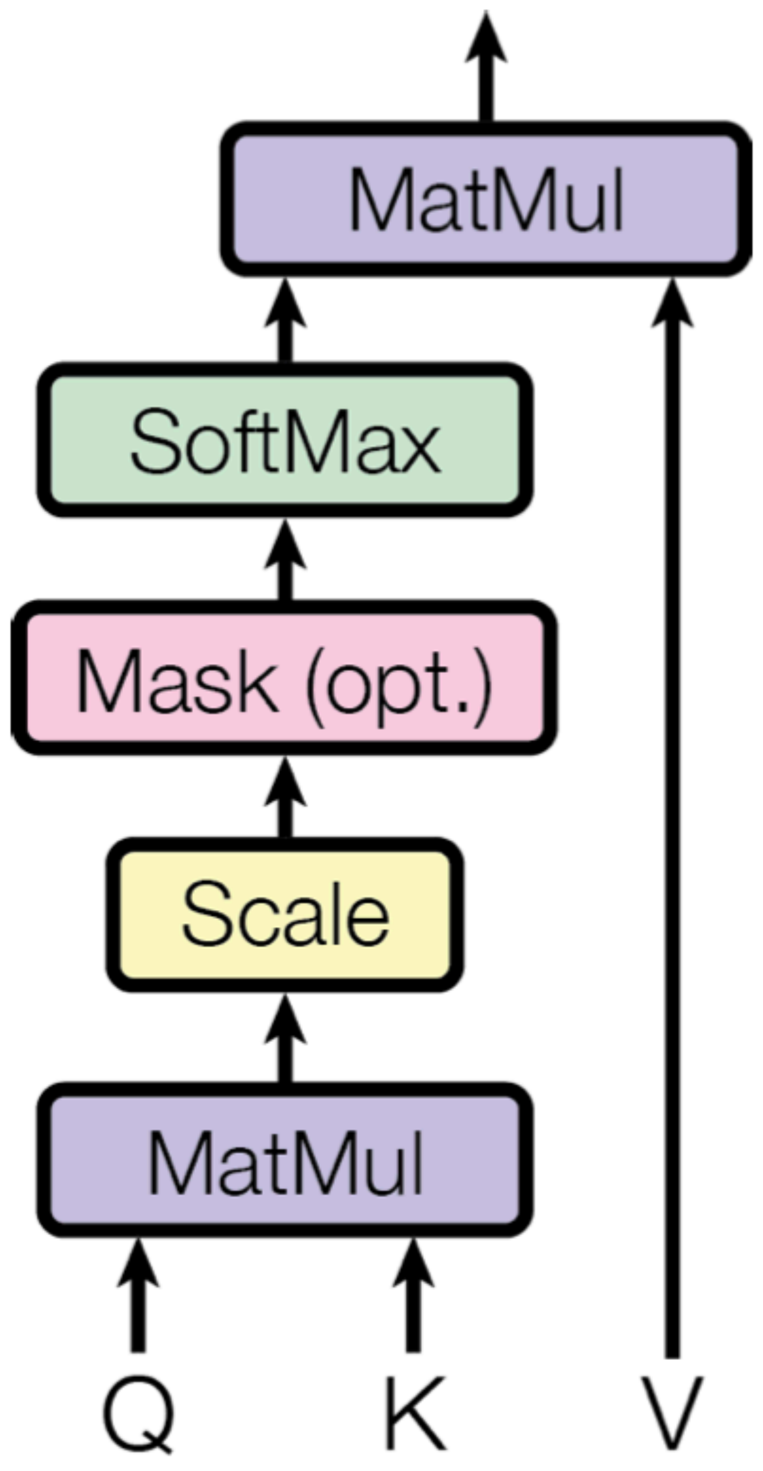
softmax

$$QK^T \in R^{n\times m} = \begin{bmatrix} q_1\cdot k_1 & q_1\cdot k_2 \\ q_2\cdot k_1 & \\ & \end{bmatrix}$$

$$Q = \begin{bmatrix} \vec{q_1}^T \\ \vec{q_2}^T \\ \vec{q_3}^T \\ \vdots \end{bmatrix} \qquad K = \begin{bmatrix} \vec{k_1}^T \\ \vec{k_2}^T \\ \vdots \\ \vdots \end{bmatrix} \qquad V = \begin{bmatrix} \\ \end{bmatrix}$$

Recap: What is Attention

Scaled Dot-Product Attention



Q: Query
K: key
V: value

$$Q \in R^{n \times d} \qquad K \in R^{m \times d} \qquad V \in R^{m \times d}$$

We have n queries, m (key, value) pairs

Attention weight = softmax(QK^T)

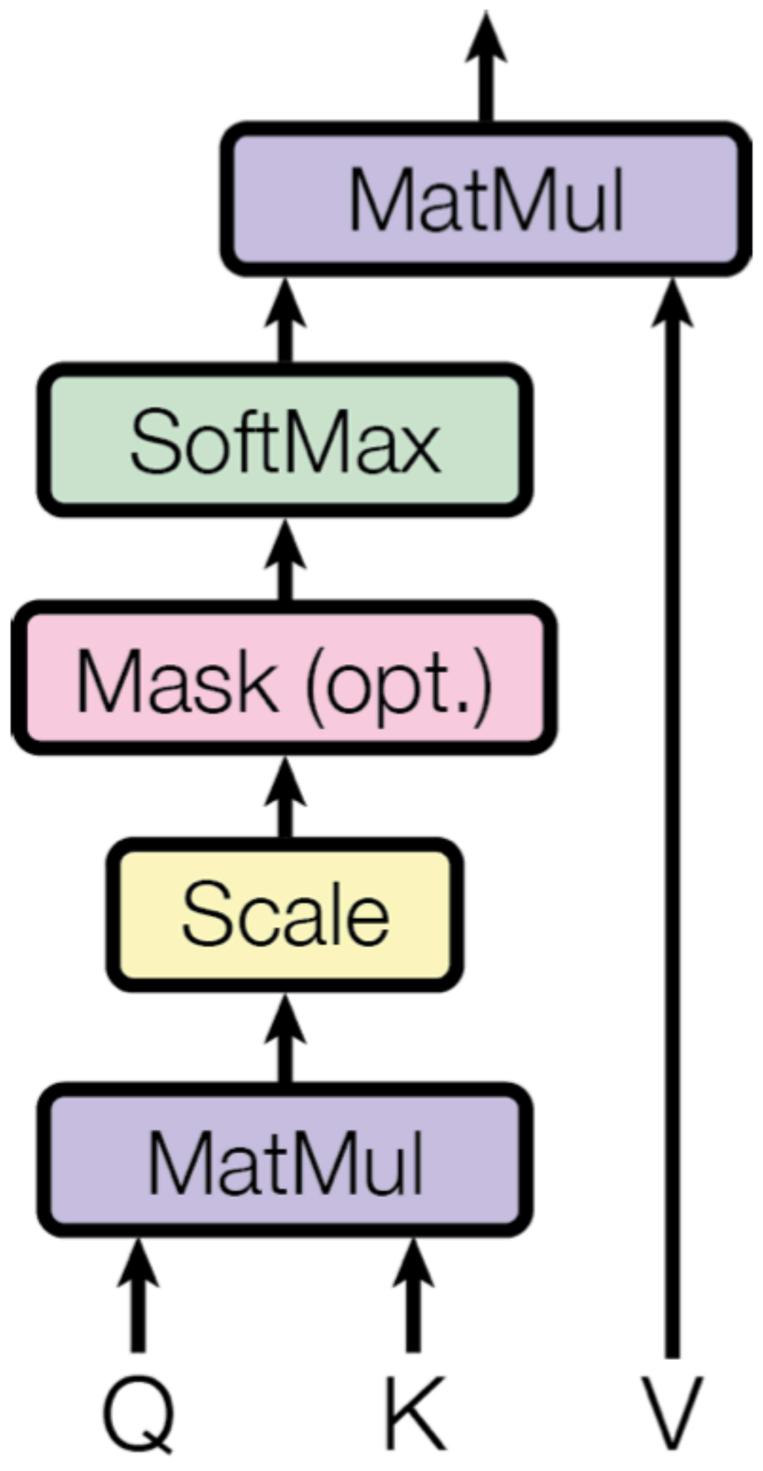
Dot-products grow large in magnitude

Recap: What is Attention

$Q \in R^{n \times d}$ $\quad K \in R^{m \times d}$ $\quad V \in R^{m \times d}$

We have n queries, m (key, value) pairs

Scaled Dot-Product Attention



Q: Query
K: key
V: value

Attention weight = softmax(QK^T)

Dot-products grow large in magnitude

Scaled Attention weight = softmax$\left(\frac{QK^T}{\sqrt{d_k}}\right)$

dot product

query $= \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_d \end{bmatrix}$ d key $\begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_d \end{bmatrix}$

$Var(q_1k_1 + q_2k_2 + \cdots q_dk_d)$

$= C_3 \cdot d$ $\sqrt{d}$ $\sqrt{d}$

$= q_1k_1 + q_2k_2 + \cdots q_dk_d$

magnitude grows with d

variance is small

mean$(q_1) = 0$

$Var(q_1) = C_1$

mean$(k_1) = 0$

$Var(k_1) = C_2$

$Var(q_1k_1) = C_3$

$\underline{\underline{x}}$

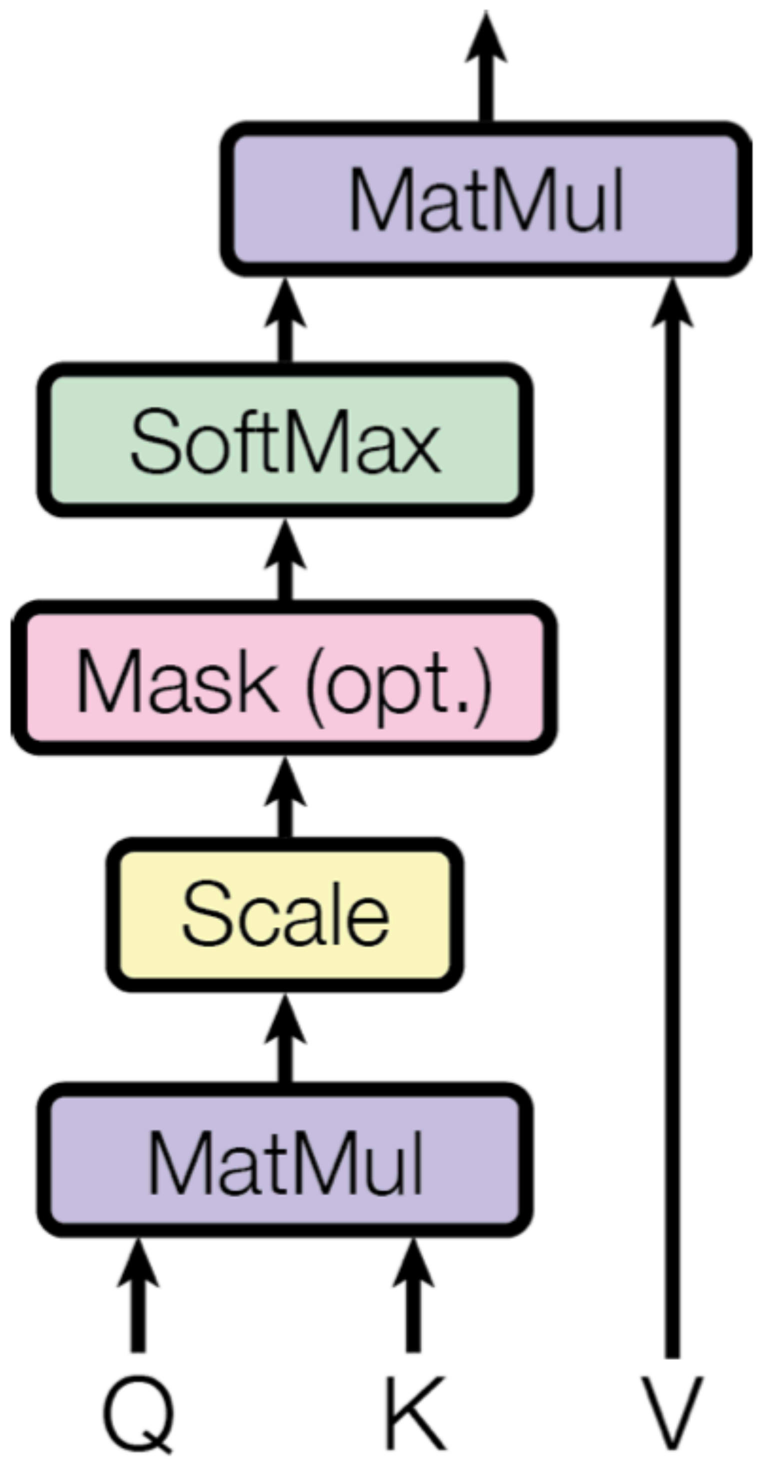
$\underline{Var(x)}$

$Var(\underset{\sim}{a \cdot x}) = a^2 Var(x)$

$$Var\left(\frac{x}{\sqrt{d}}\right) = \frac{Var(x)}{d}$$

Recap: What is Attention

Scaled Dot-Product Attention



Q: Query
K: key
V: value

$$Q \in R^{n \times d} \qquad K \in R^{m \times d} \qquad V \in R^{m \times d}$$

We have n queries, m (key, value) pairs

Attention weight = softmax(QK^T)

Dot-products grow large in magnitude

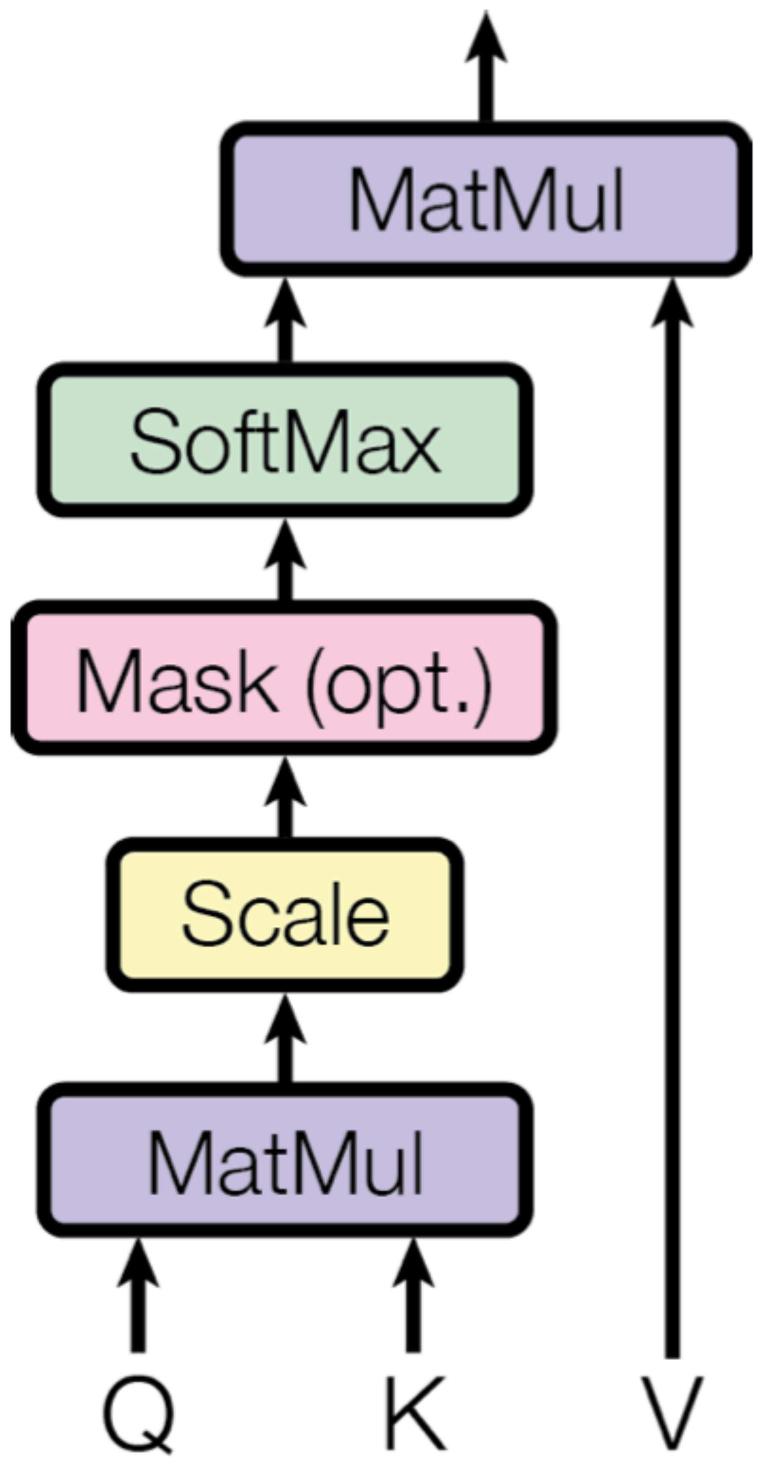
Scaled Attention weight = softmax$\left(\frac{QK^T}{\sqrt{d_k}}\right)$ Shape is mxn

Recap: What is Attention

$$Q \in R^{n \times d} \qquad K \in R^{m \times d} \qquad V \in R^{m \times d}$$

We have n queries, m (key, value) pairs

Scaled Dot-Product Attention



Q: Query
K: key
V: value

Attention weight = softmax(QK^T)

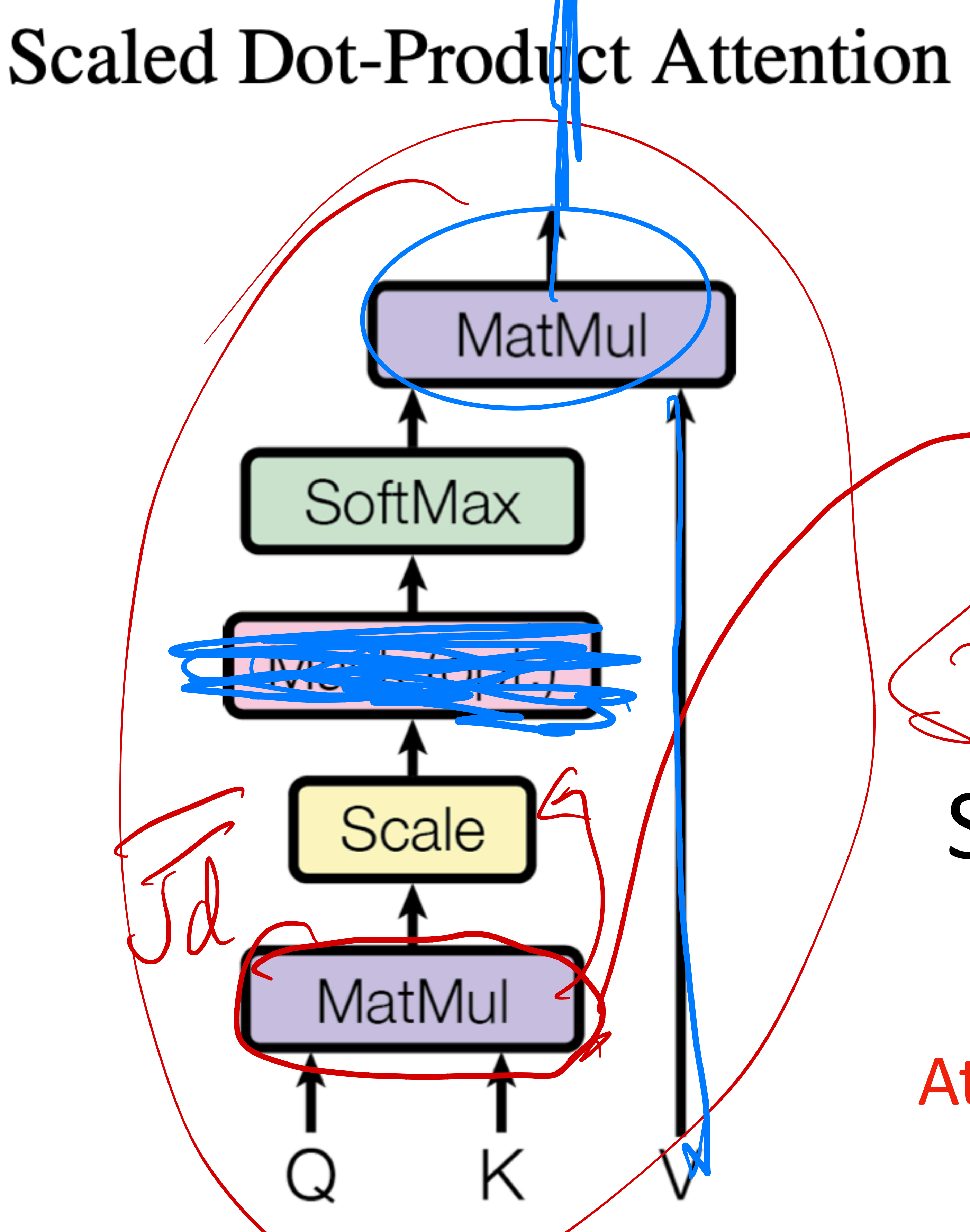
Dot-products grow large in magnitude

Scaled Attention weight = softmax$\left(\frac{QK^T}{\sqrt{d_k}}\right)$

Shape is mxn

Attention weight represents the strength to "attend" values V

Recap: What is Attention



Q: Query
K: key
V: value

$Q \in R^{n\times d}$ $K \in R^{m\times d}$ $V \in R^{m\times d}$

We have n queries, m (key, value) pairs

Attention weight = softmax(QK^T)

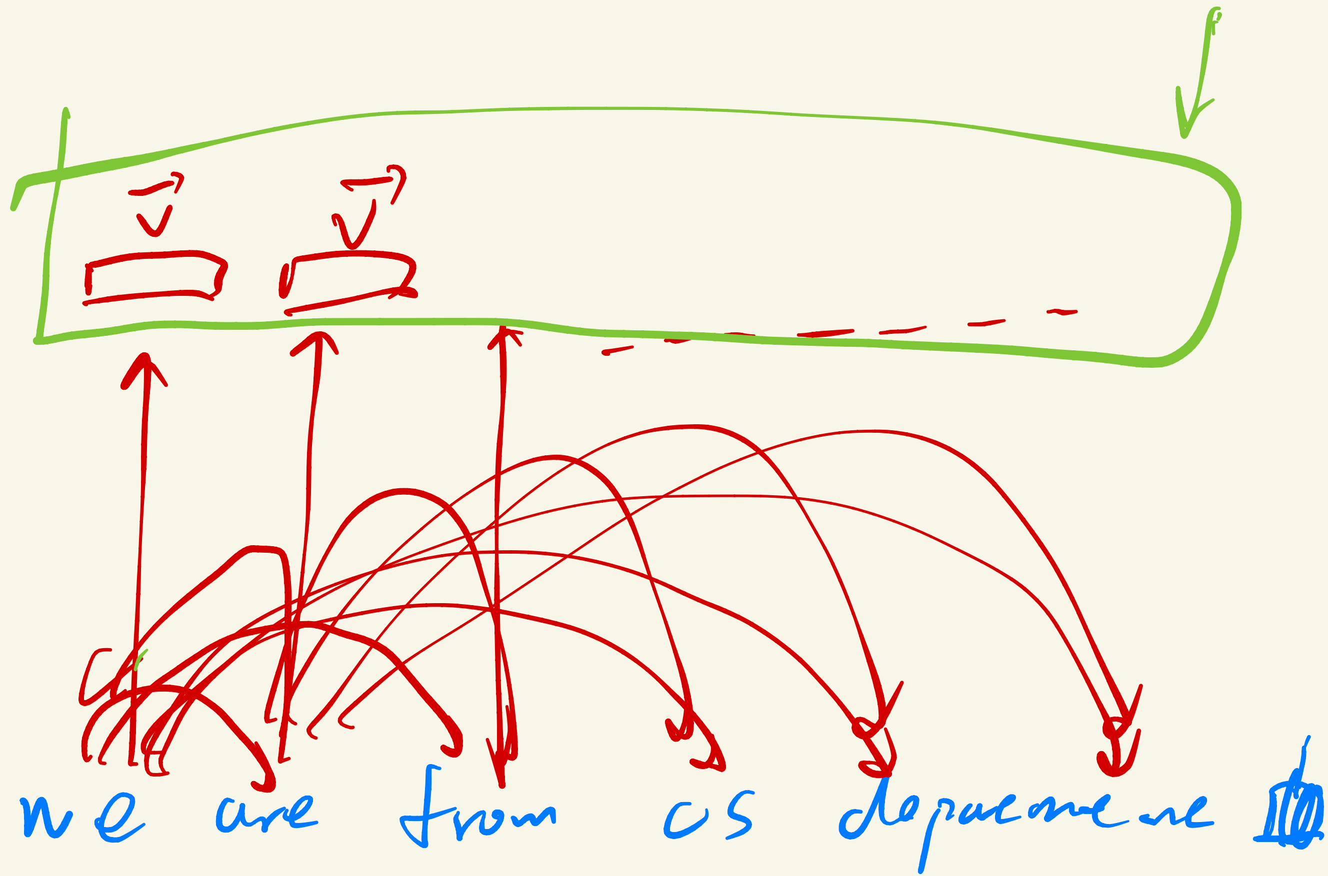
Dot-products grow large in magnitude

Scaled Attention weight = softmax$\left(\frac{QK^T}{\sqrt{d_k}}\right)$ Shape is mxn

d_{key}

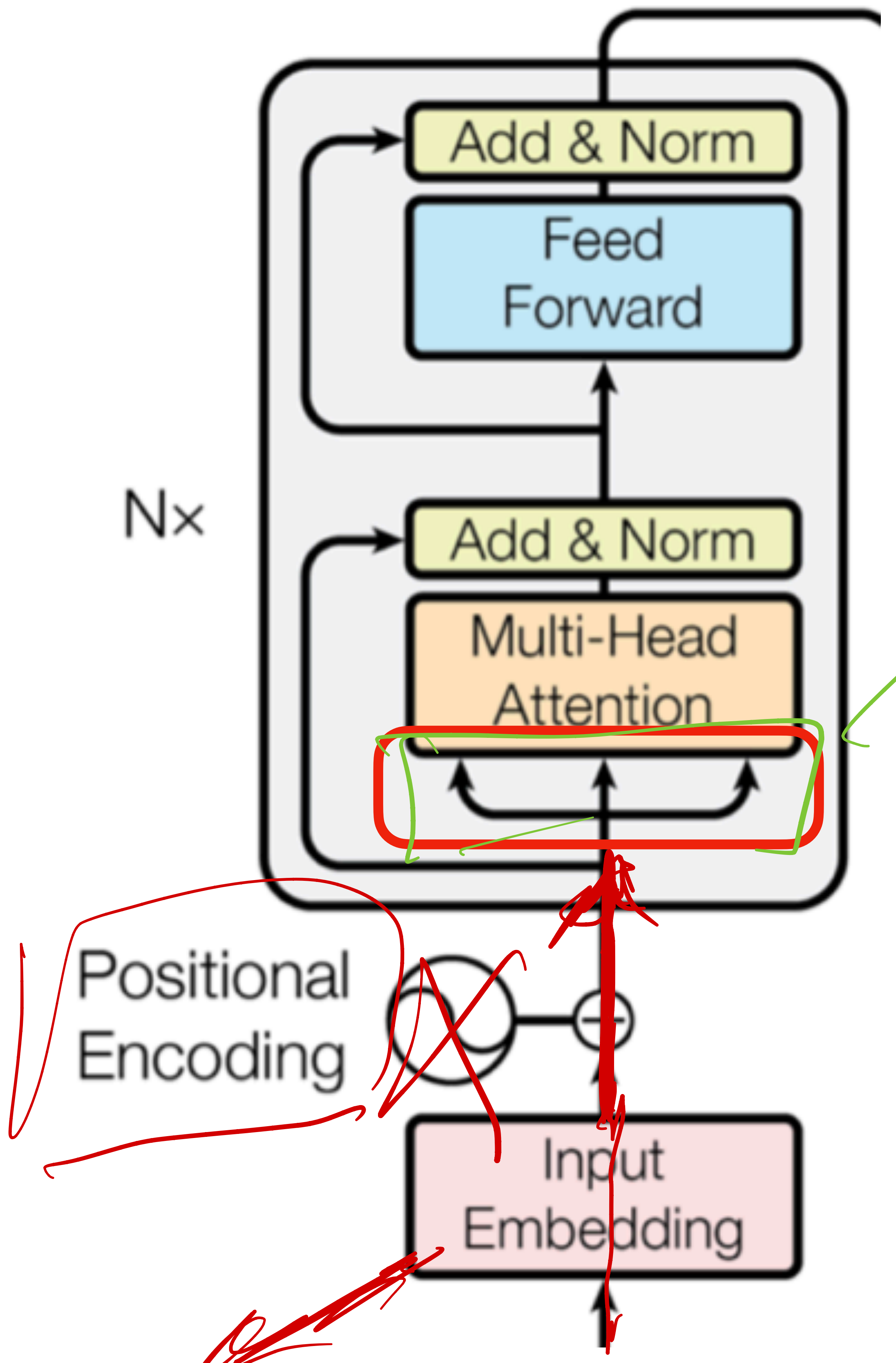
Attention weight represents the strength to "attend" values V

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$\vec{v}$
$\vec{v}$
we are from cs department

Q, K, V

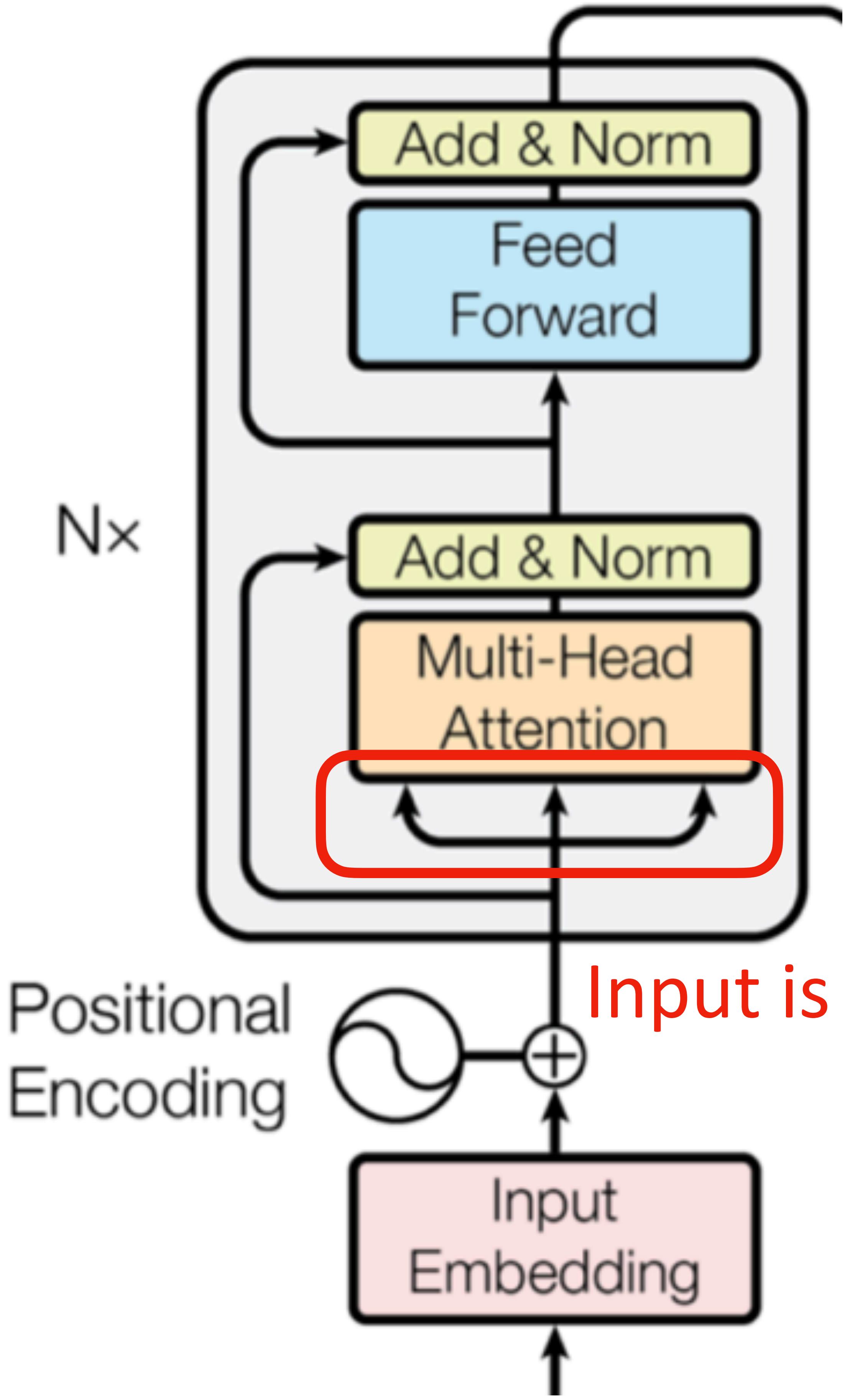


What are Q, K, V in the transformer

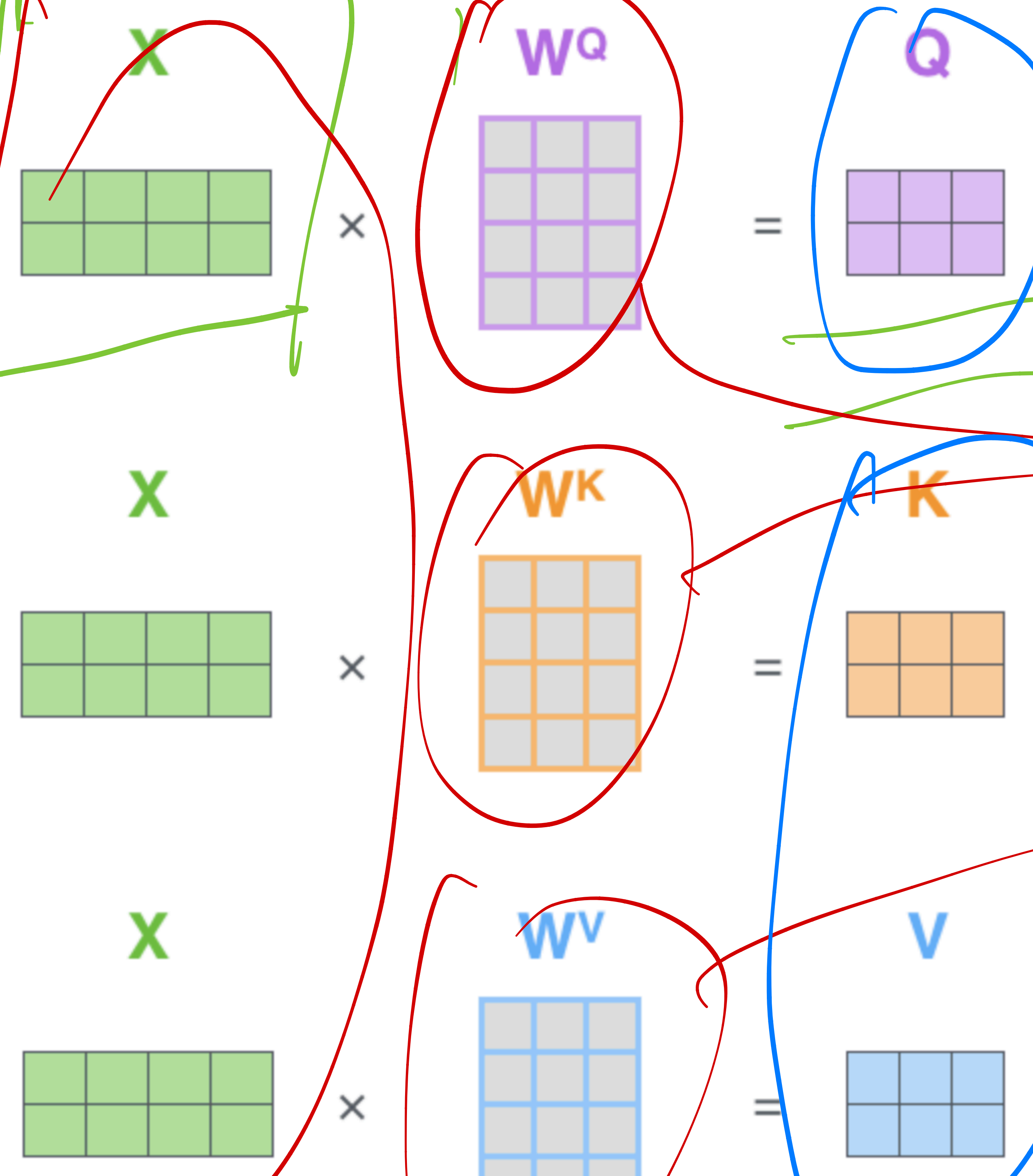
CS

embedding (CS)

Self-Attention



Input is X



$x = \begin{bmatrix} emb_1 \\ emb_2 \\ emb_3 \end{bmatrix}$

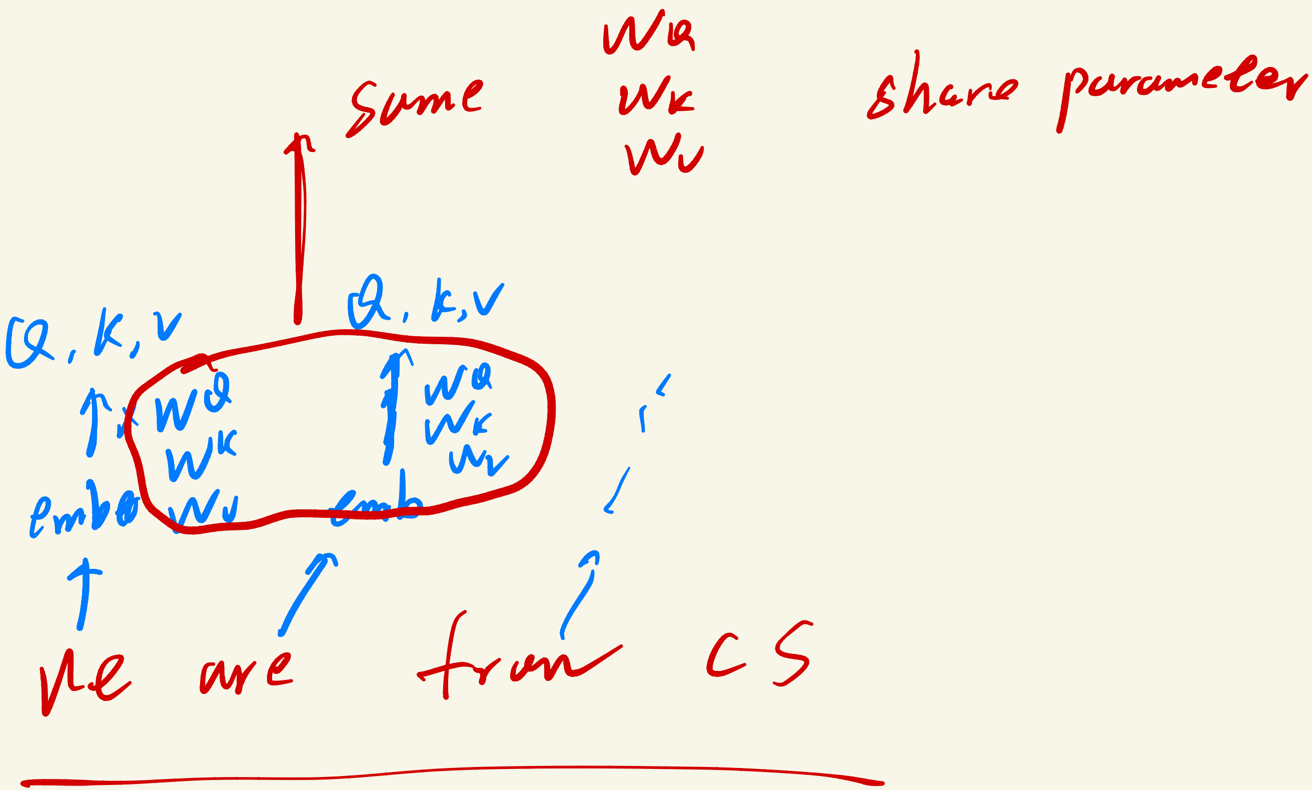
parameters

$x \to Q$

$x \to K$

15

Jay Alammar. The Illustrated Transformer.



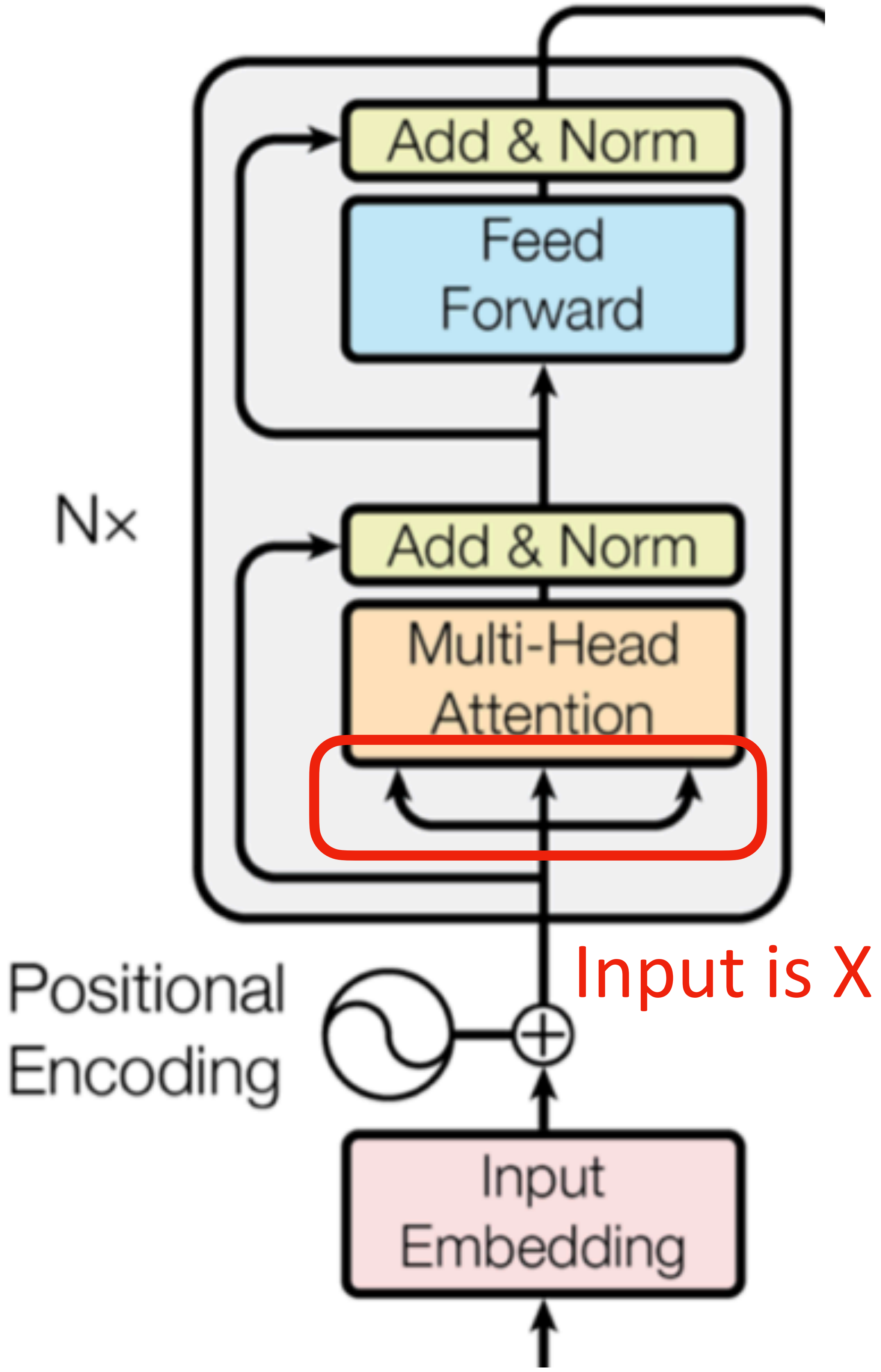
Same
W_Q
W_K
W_V
share parameter
Q, K, V
W^Q
W^K
W^V
emb
Q, K, V
W_Q
W_K
W_V
emb
We are from CS

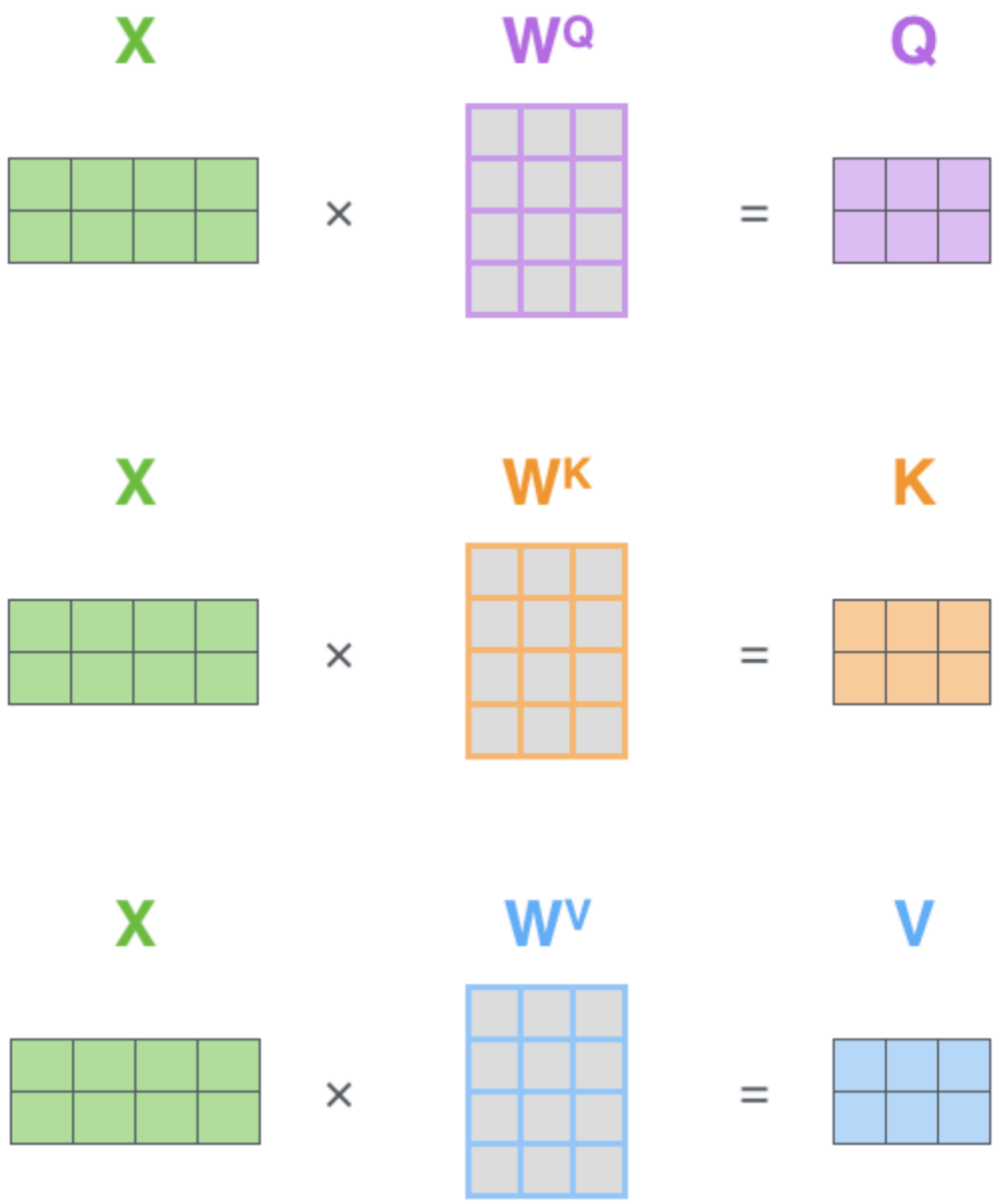
linear multiplication:

$$\underset{3}{\vec{q}} \cdot \underset{3\times 5}{M}$$

linear transformation

Self-Attention



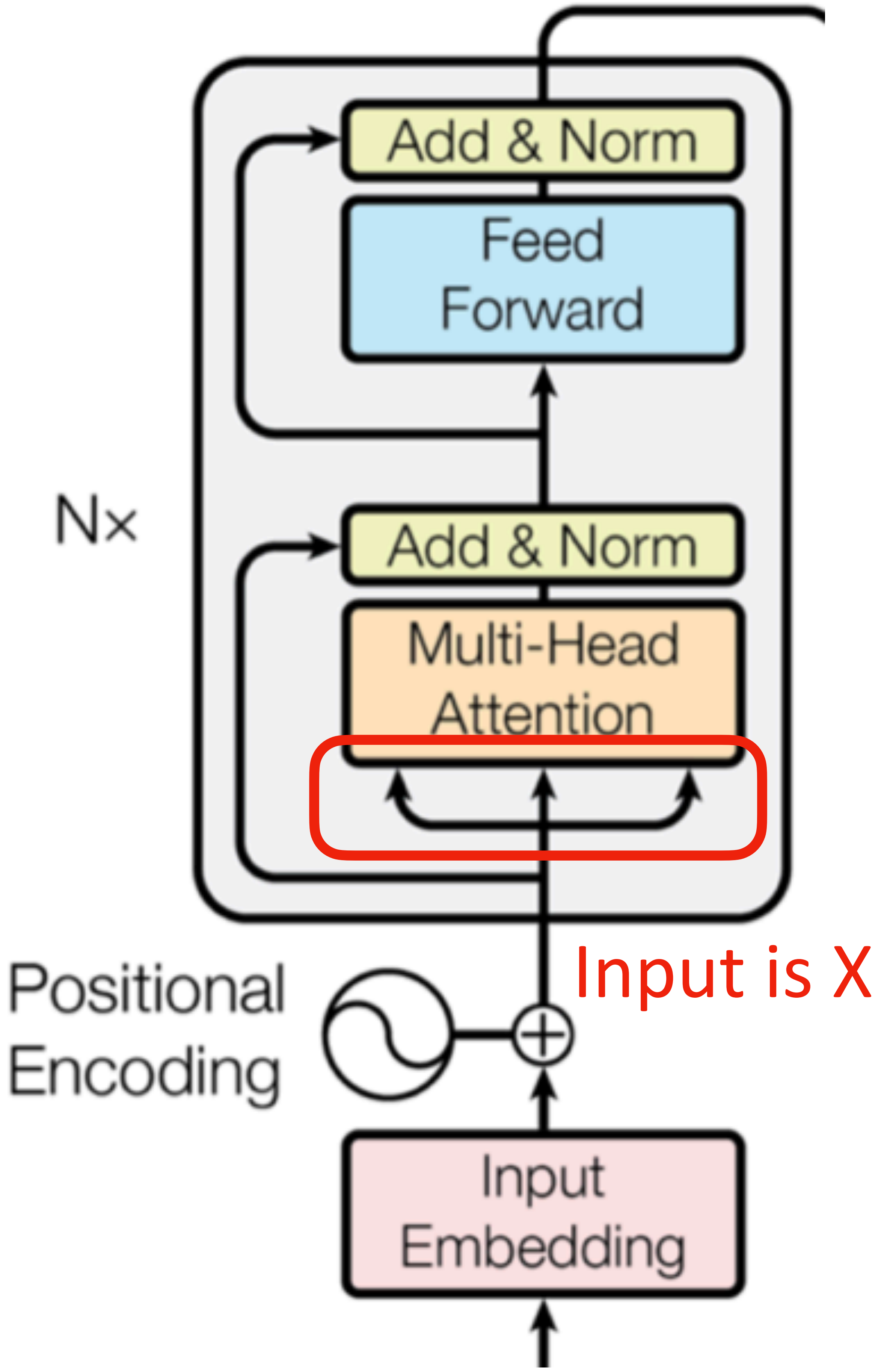


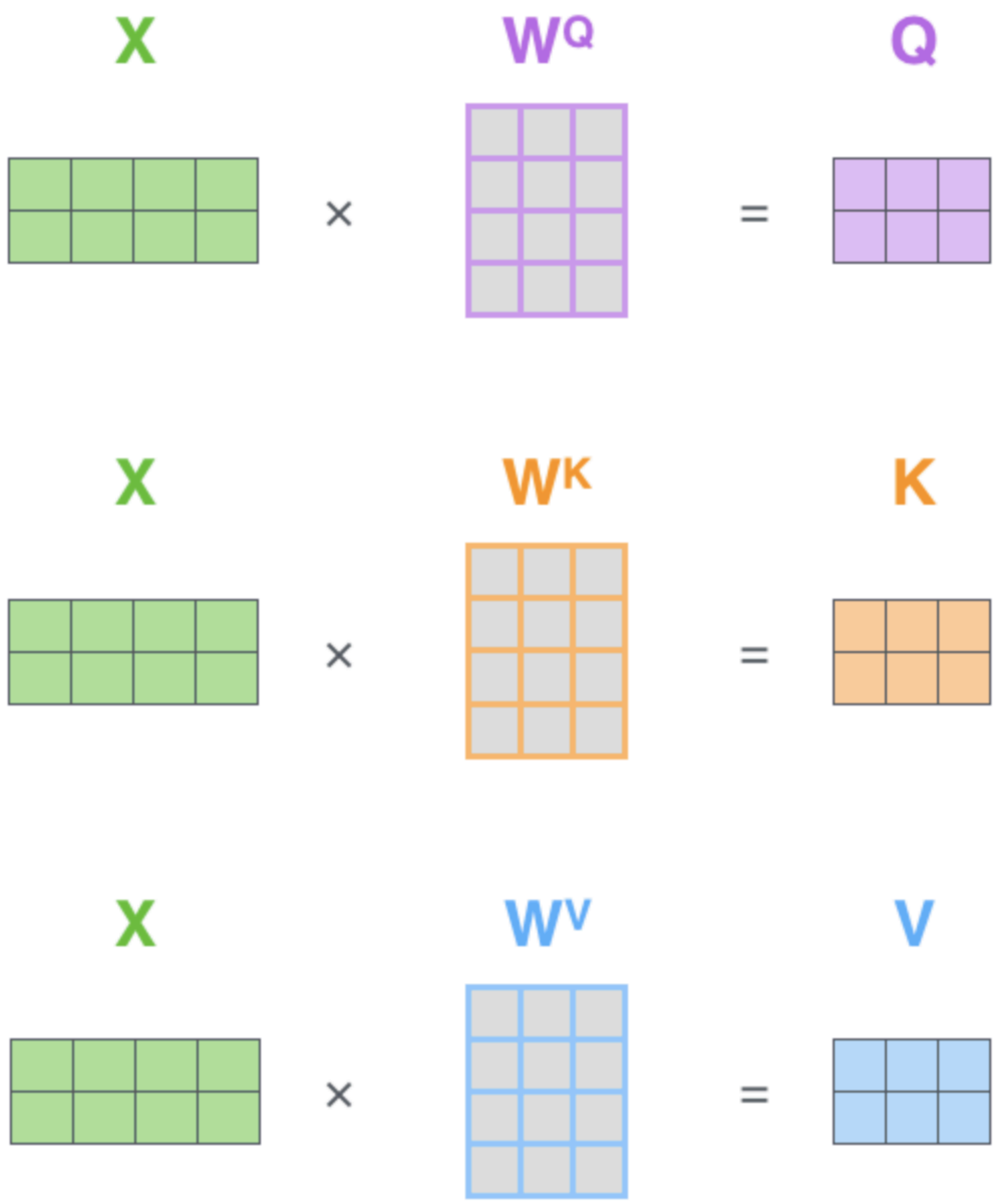
Query, key, and value are from the same input, thus it is called “self”-attention

15

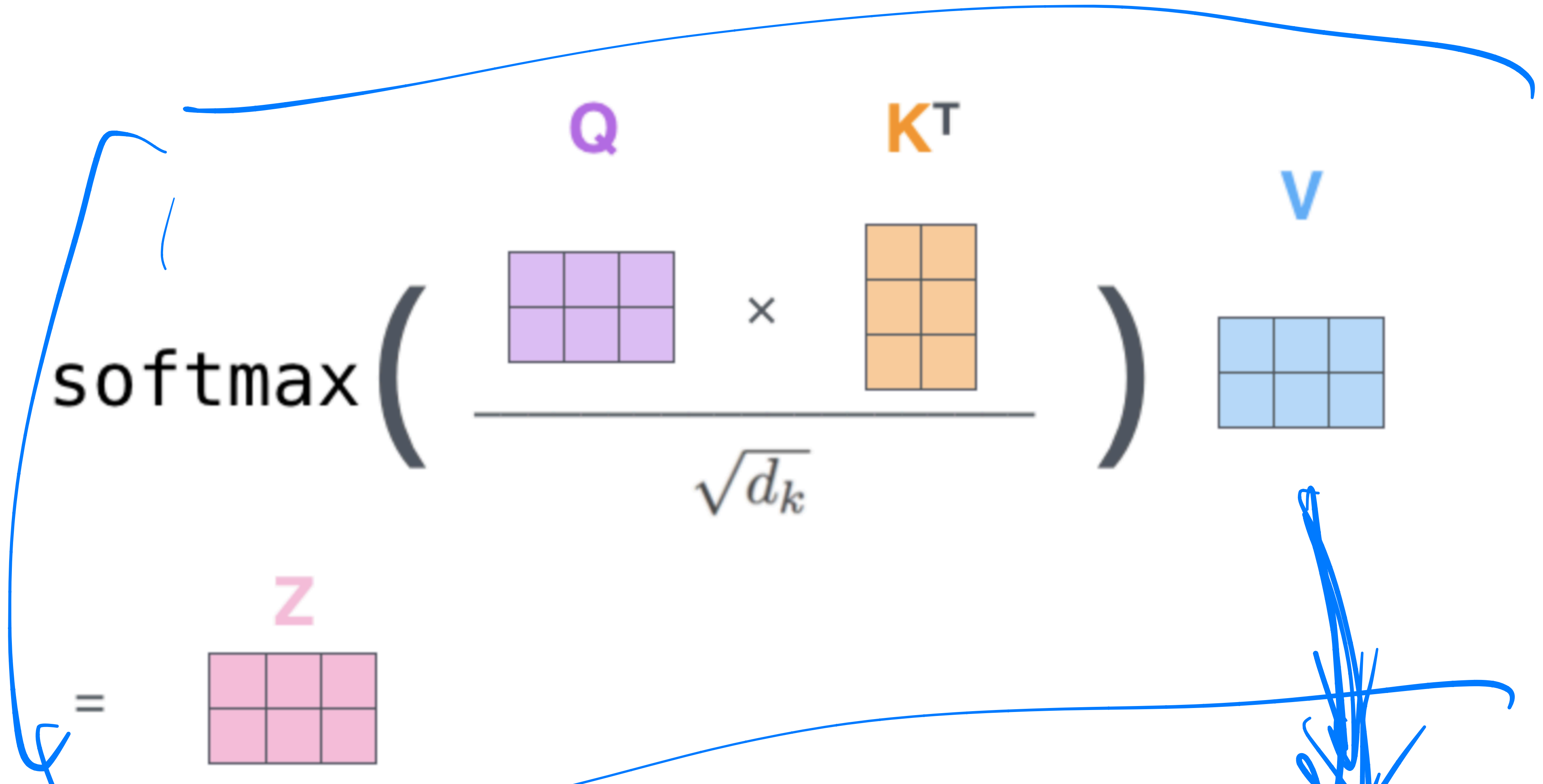
Jay Alammar. The Illustrated Transformer.

Self-Attention



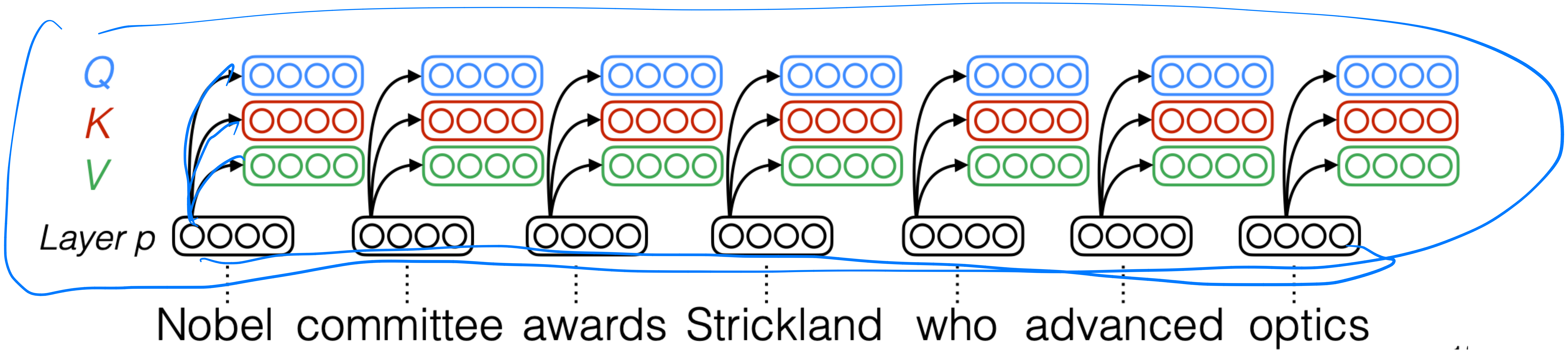


Query, key, and value are from the same input, thus it is called "self"-attention



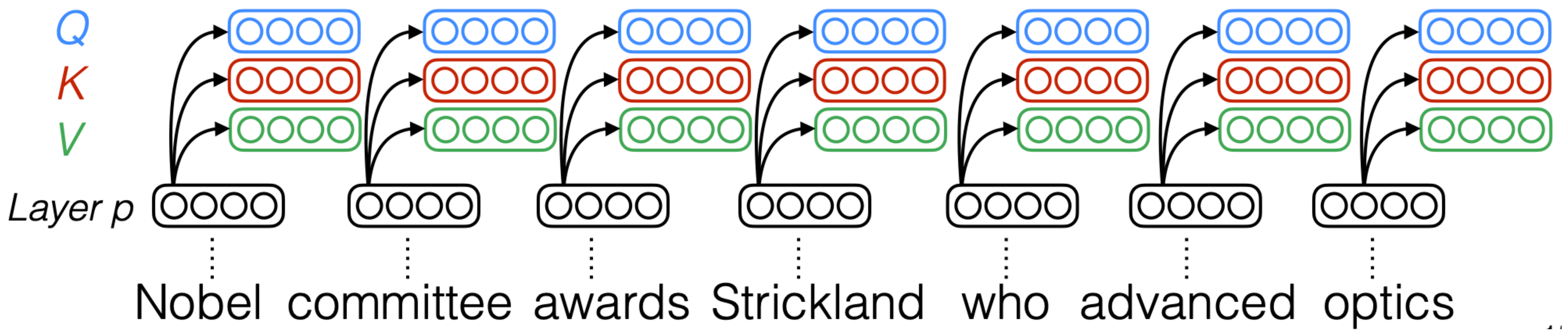
Jay Alammar. The Illustrated Transformer.

Self-Attention

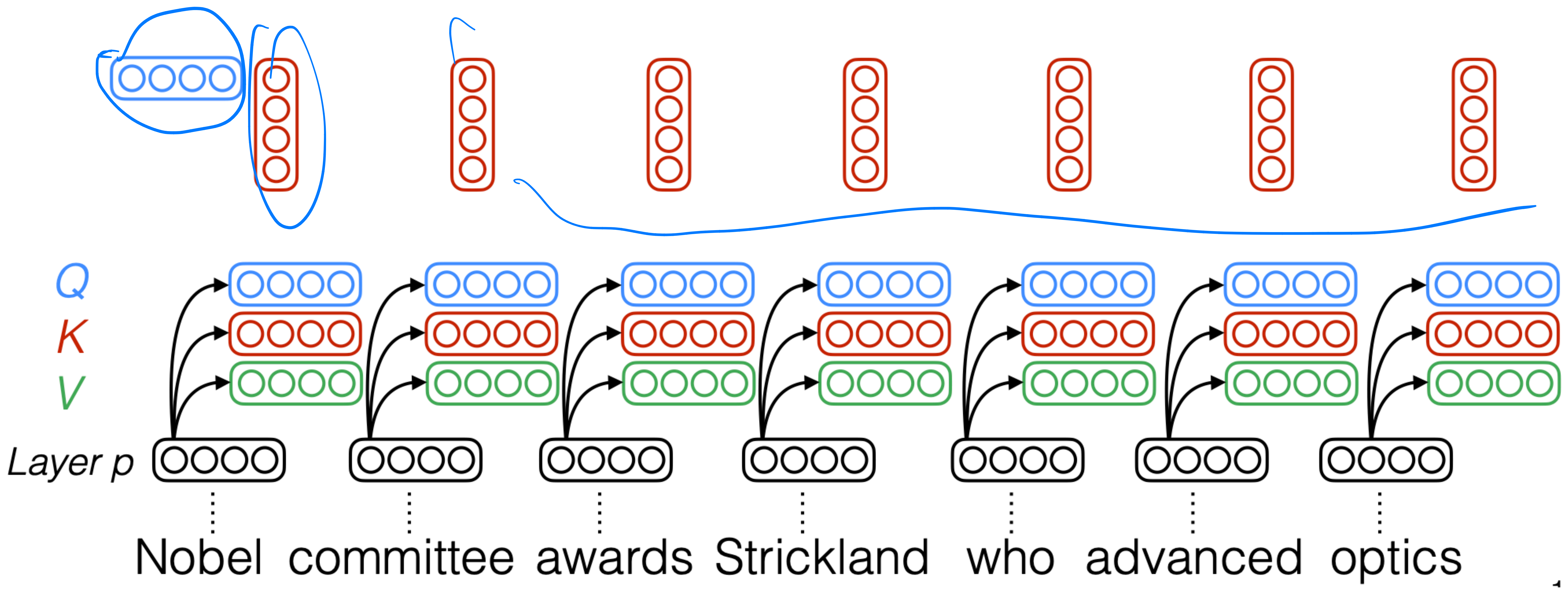


Self-Attention

At each step, the attention computation attends to all steps in the input example

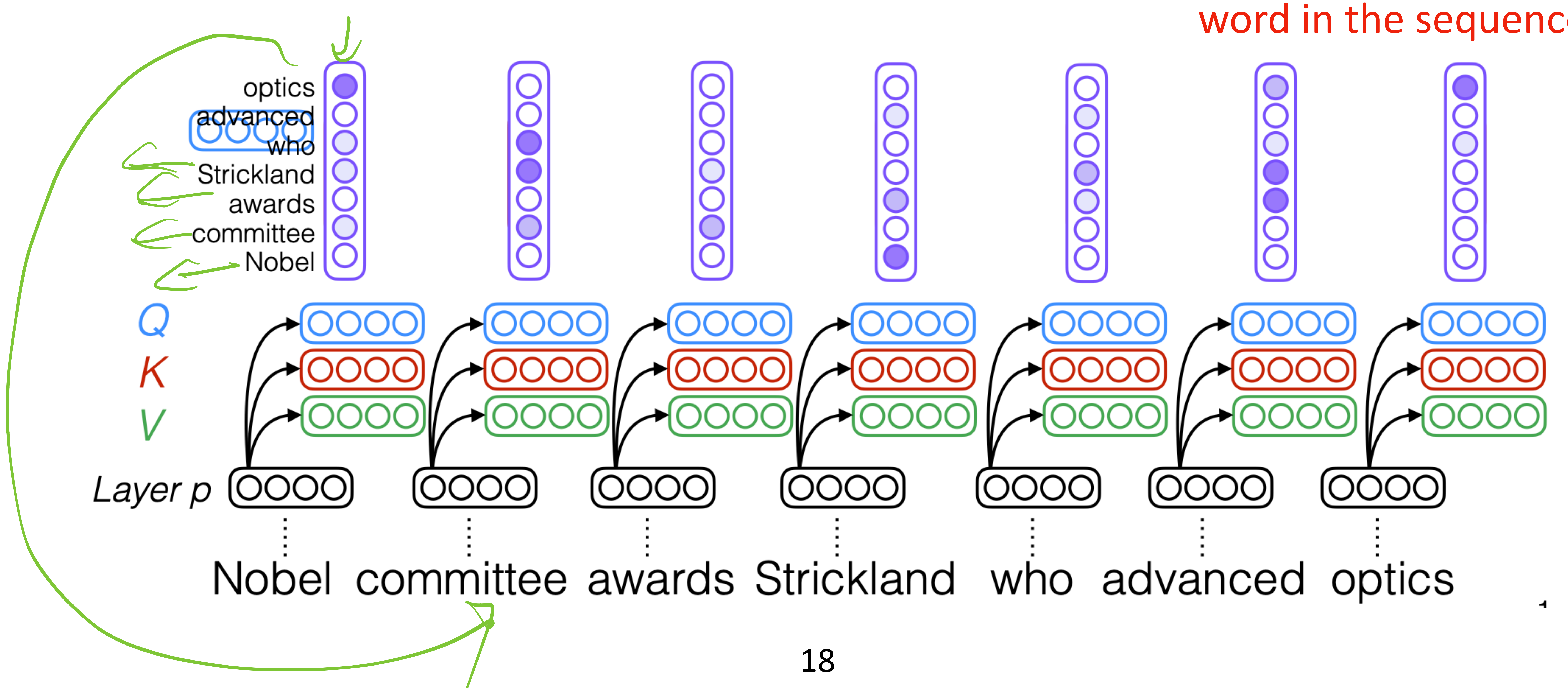


Self-Attention

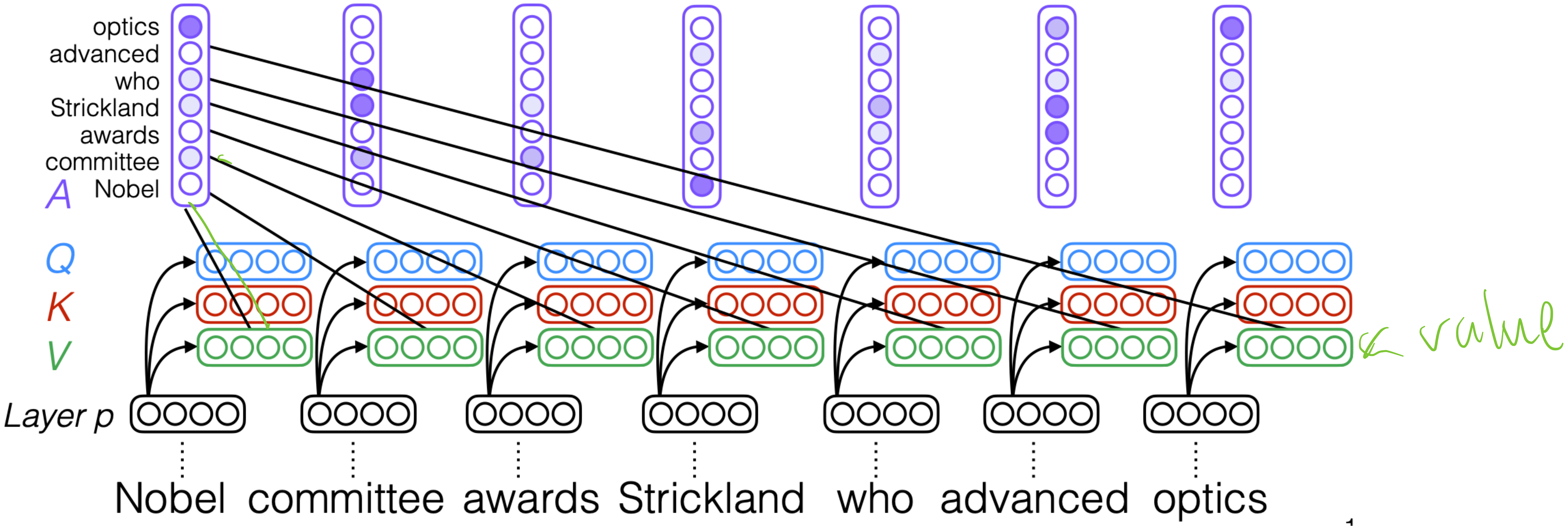


Self-Attention

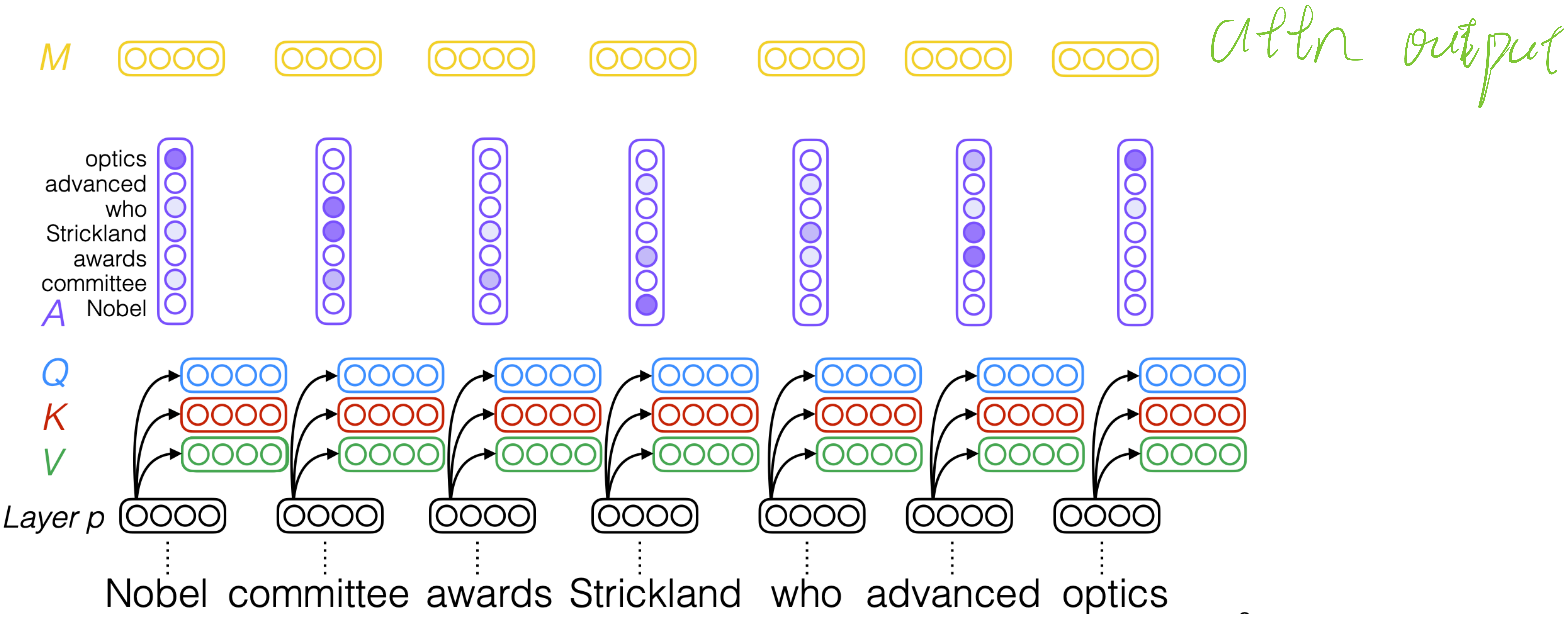
Attention weight on every word in the sequence



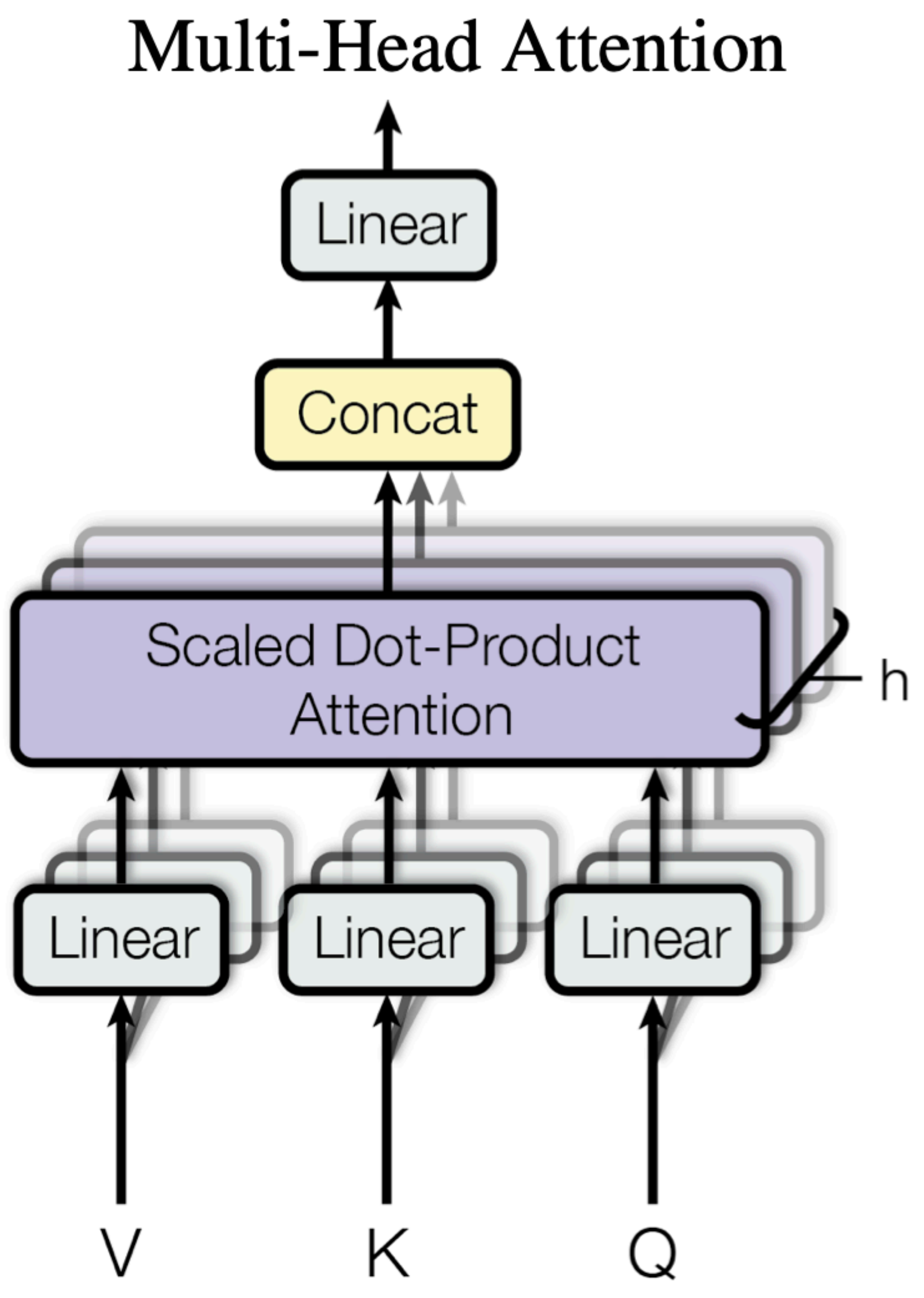
Self-Attention



Self-Attention



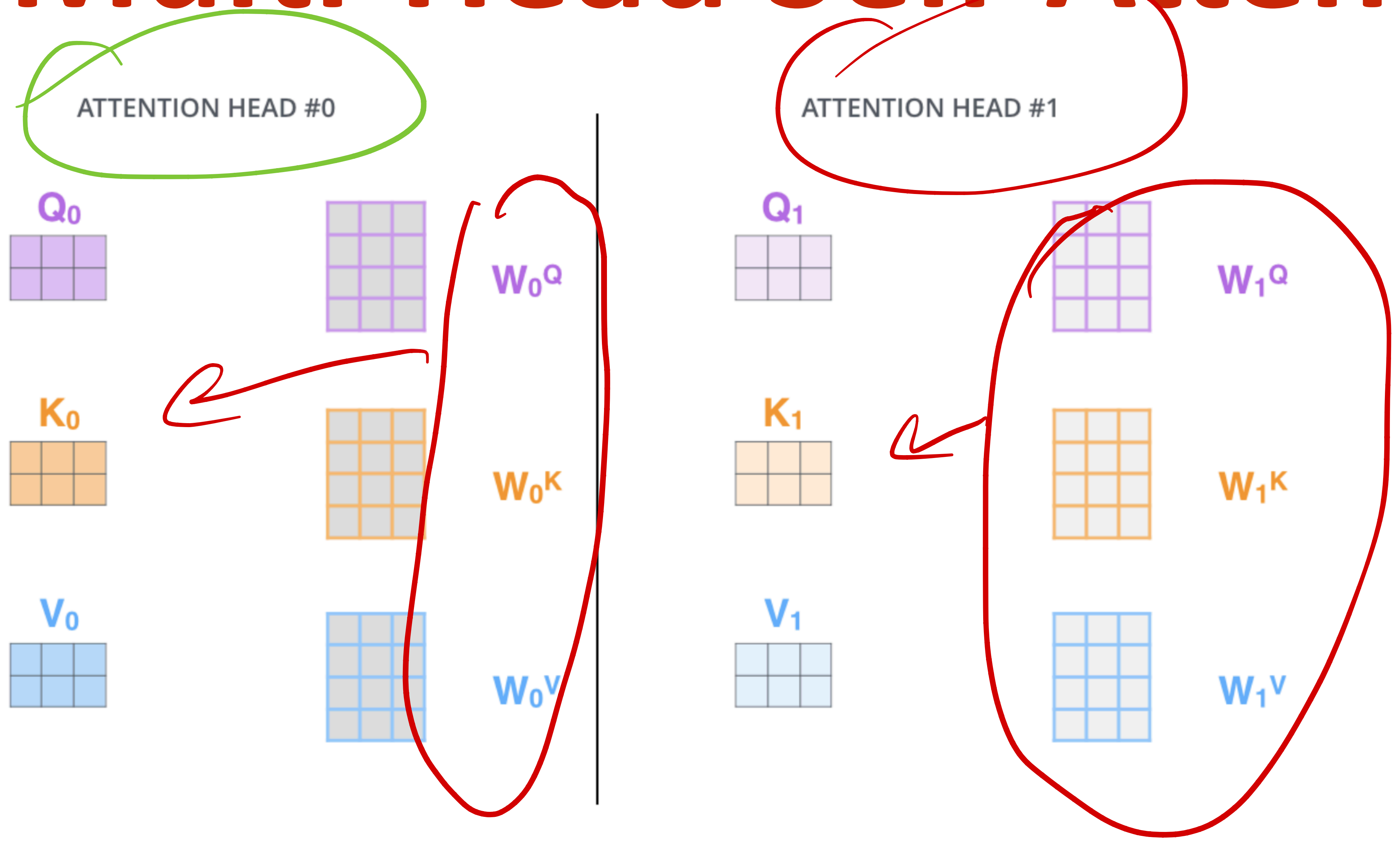
Multi-Head Attention



Multi-Head Self-Attention

Jay Alammar. The Illustrated Transformer.

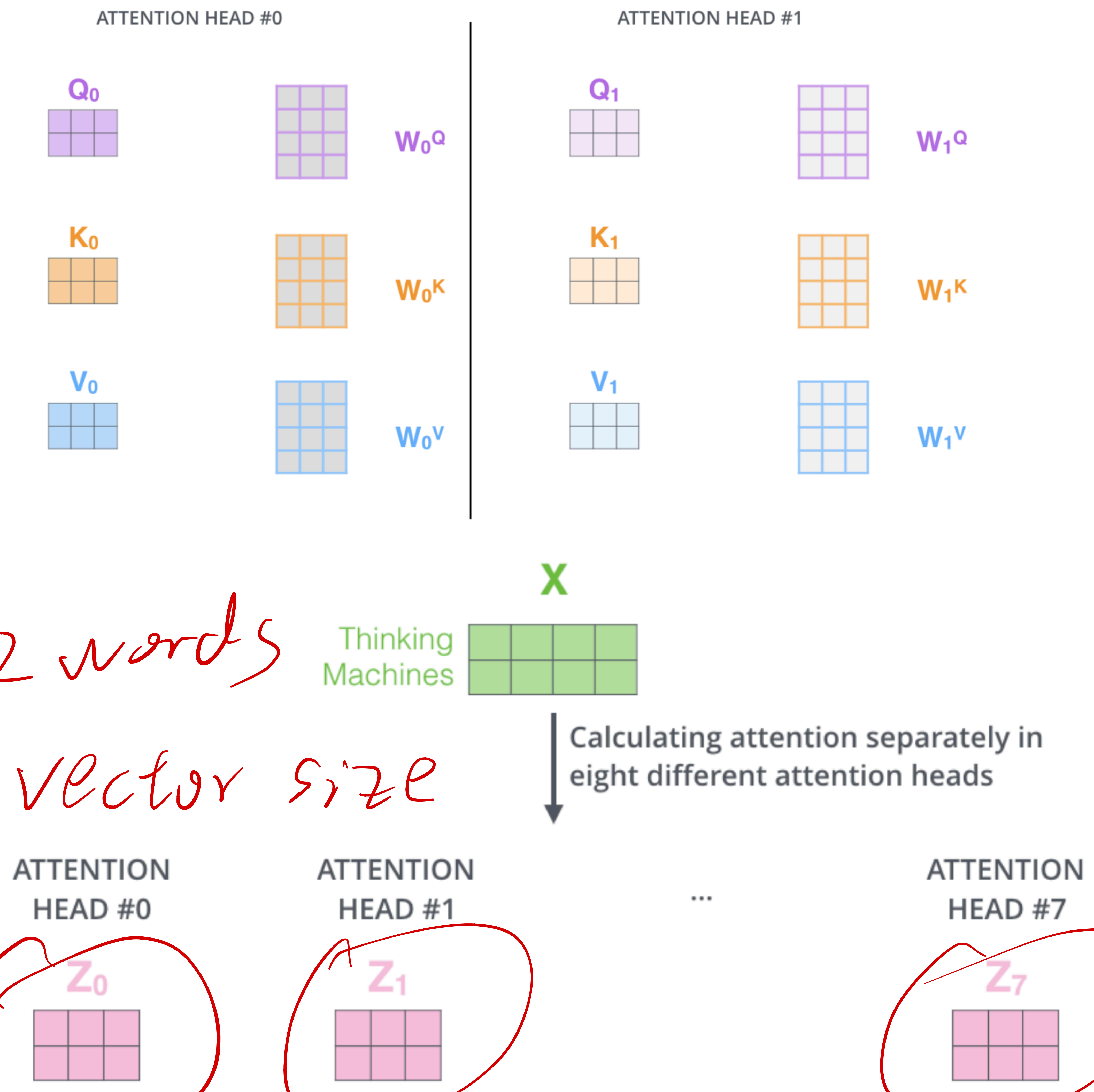
Multi-Head Self-Attention



Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention

differen paramerer



2×3

2: 2 words

3: vector size

3

8 heads

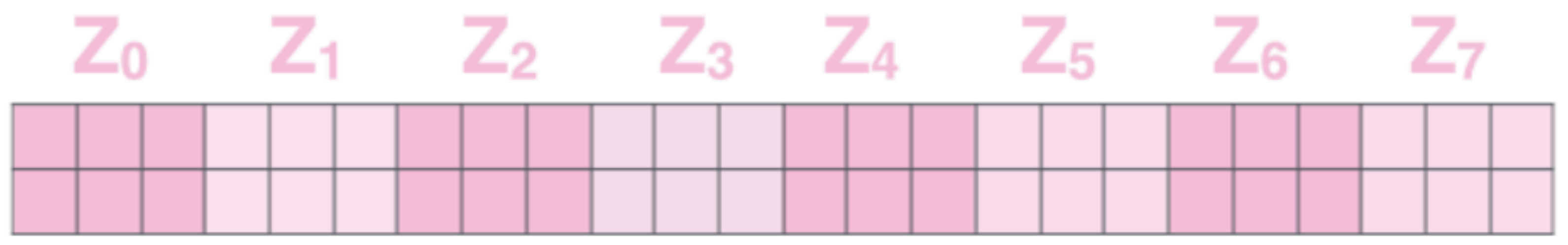
Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention

Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention

1) Concatenate all the attention heads



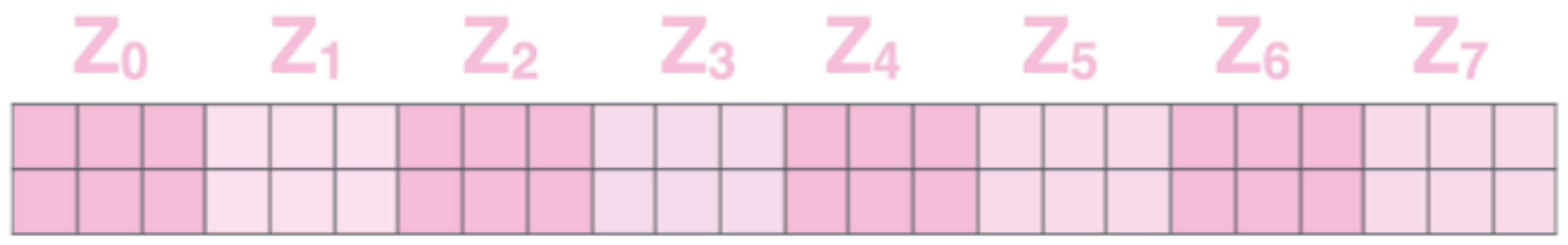
2 words, vector size 24

4

Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention

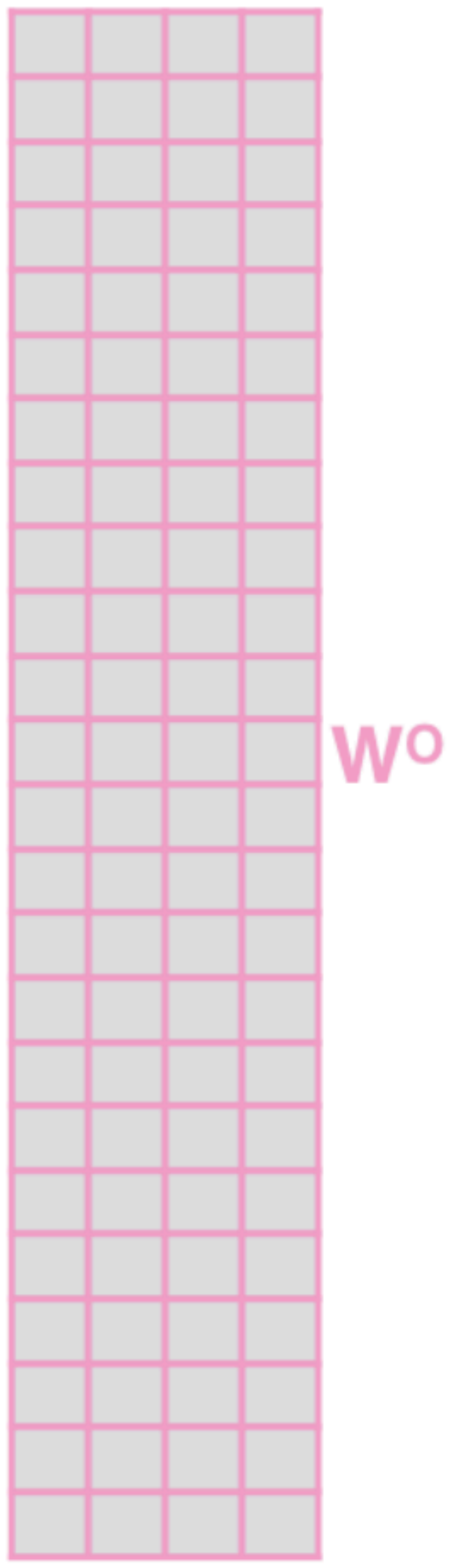
1) Concatenate all the attention heads



24

2) Multiply with a weight matrix W^O that was trained jointly with the model

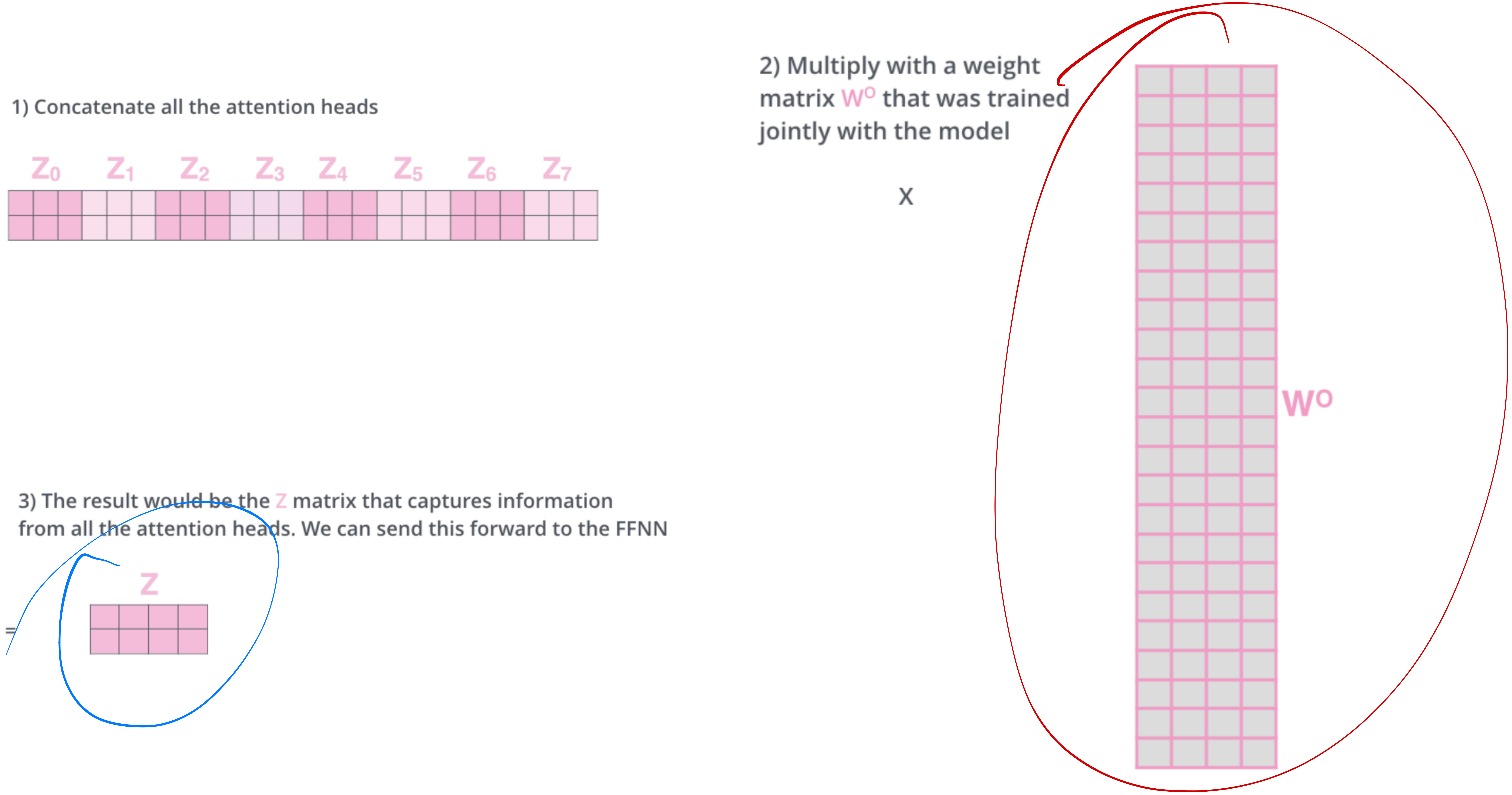
X



$= 2 \times 4$

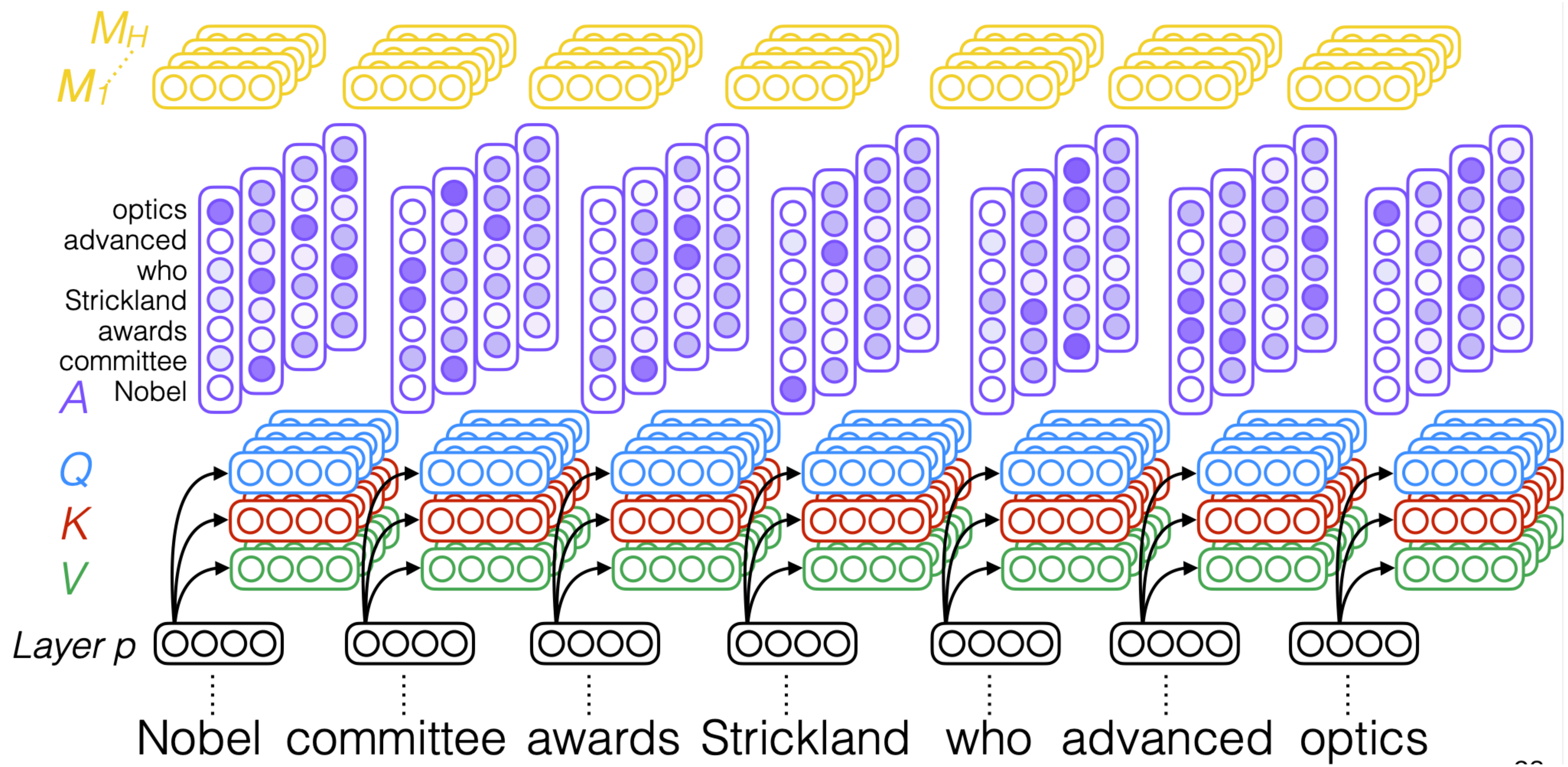
Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention



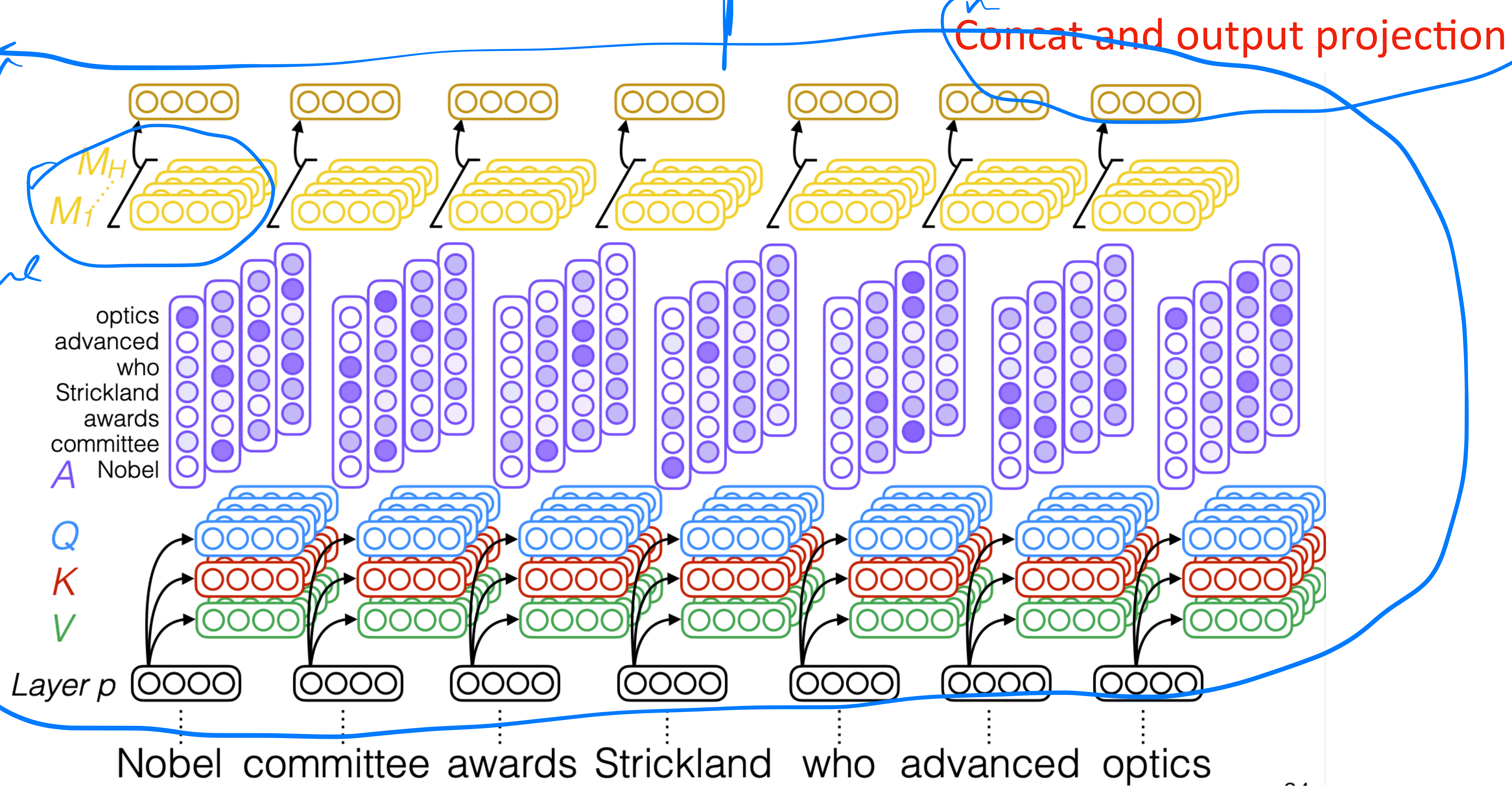
Jay Alammar. The Illustrated Transformer.

Multi-head Self-Attention



Multi-head Self-Attention

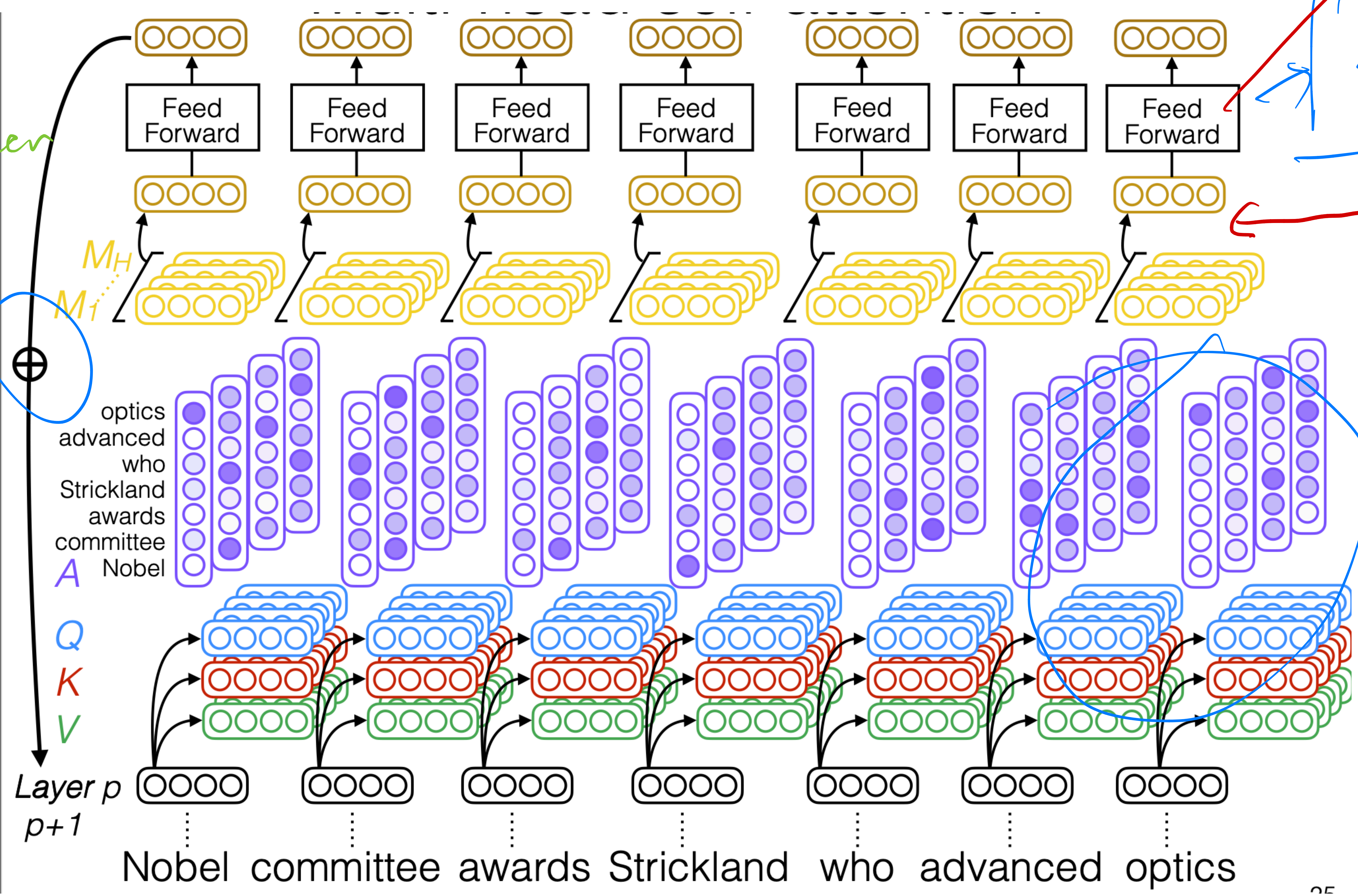
self attn
in transforme



Slides by Emma Strubell

Multi-head Self-Attention + FFN

One transformer layer



parameters

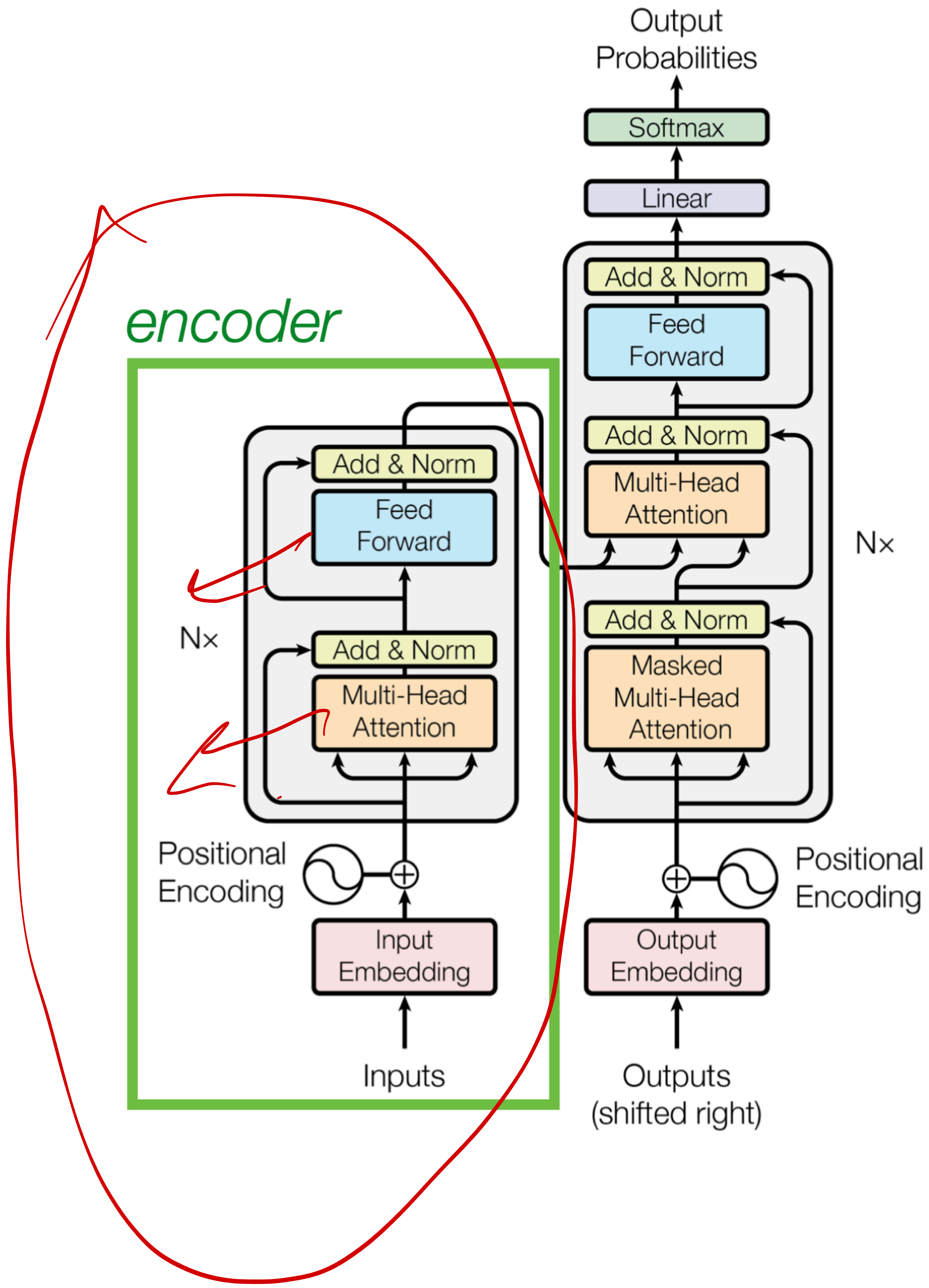
2-layer

output projection

Wq
Wk
Wv
for each head

Transformer Encoder

Currently we only cover the encoder side

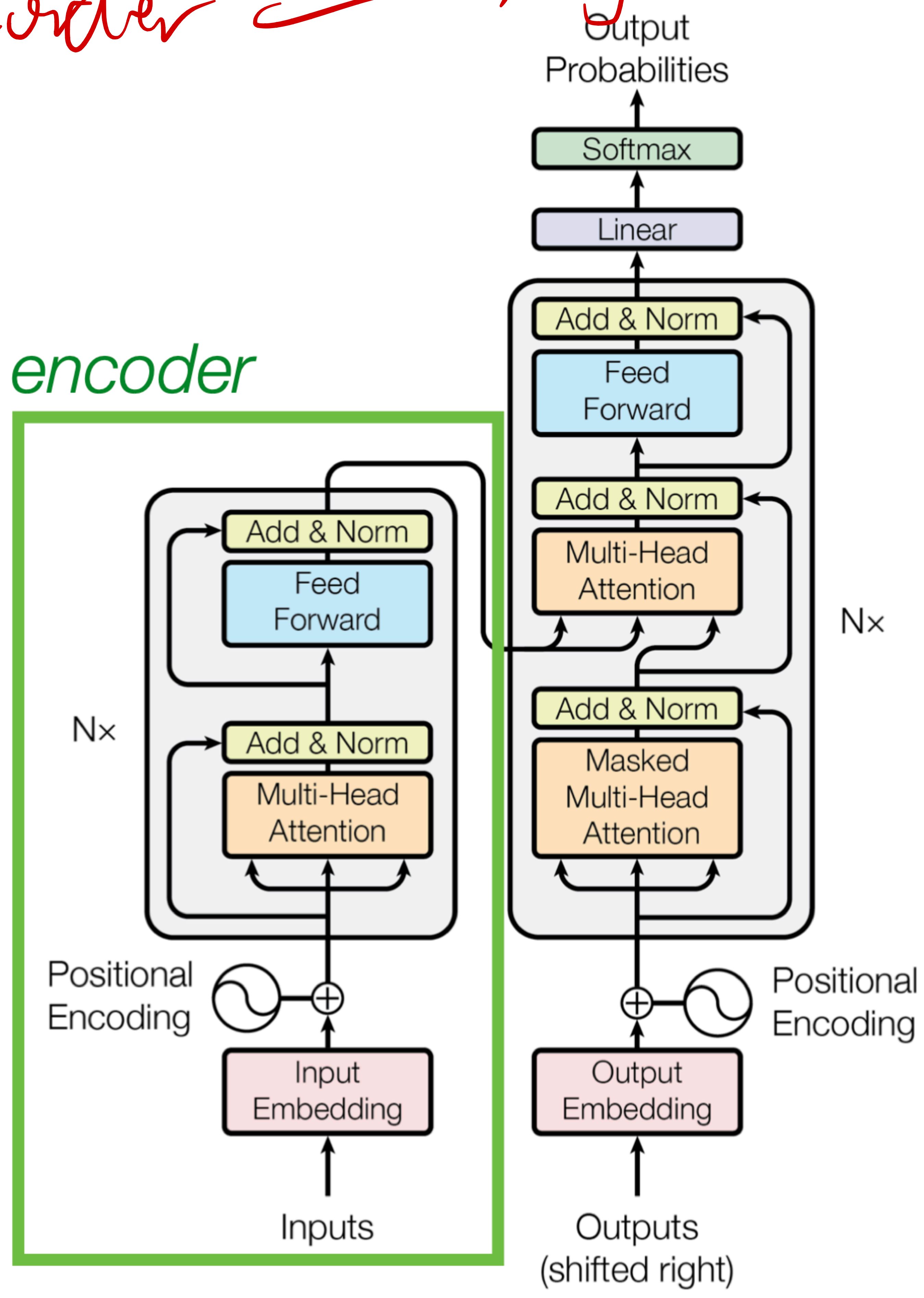


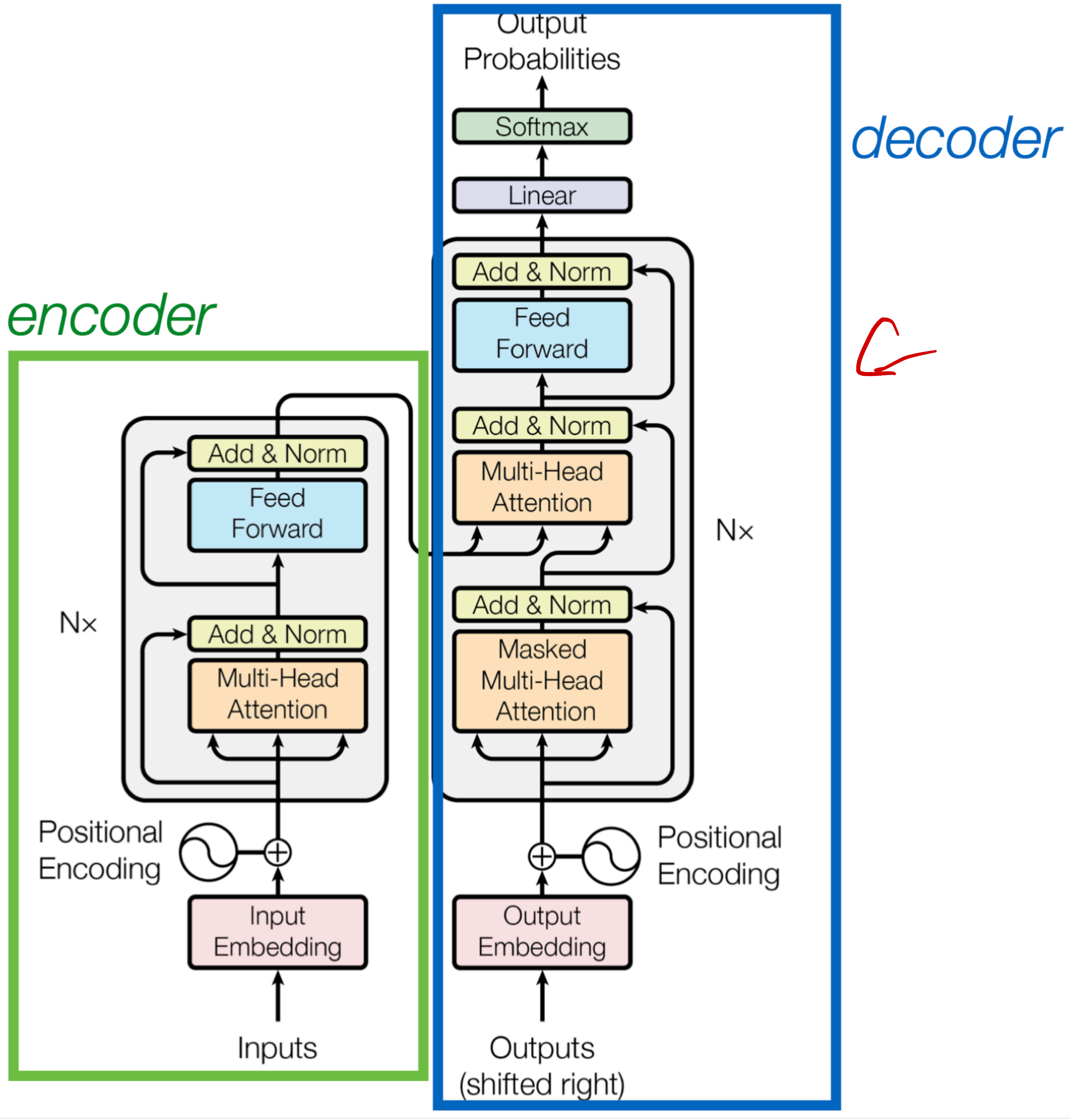
encoder → get representation

Transformer Encoder

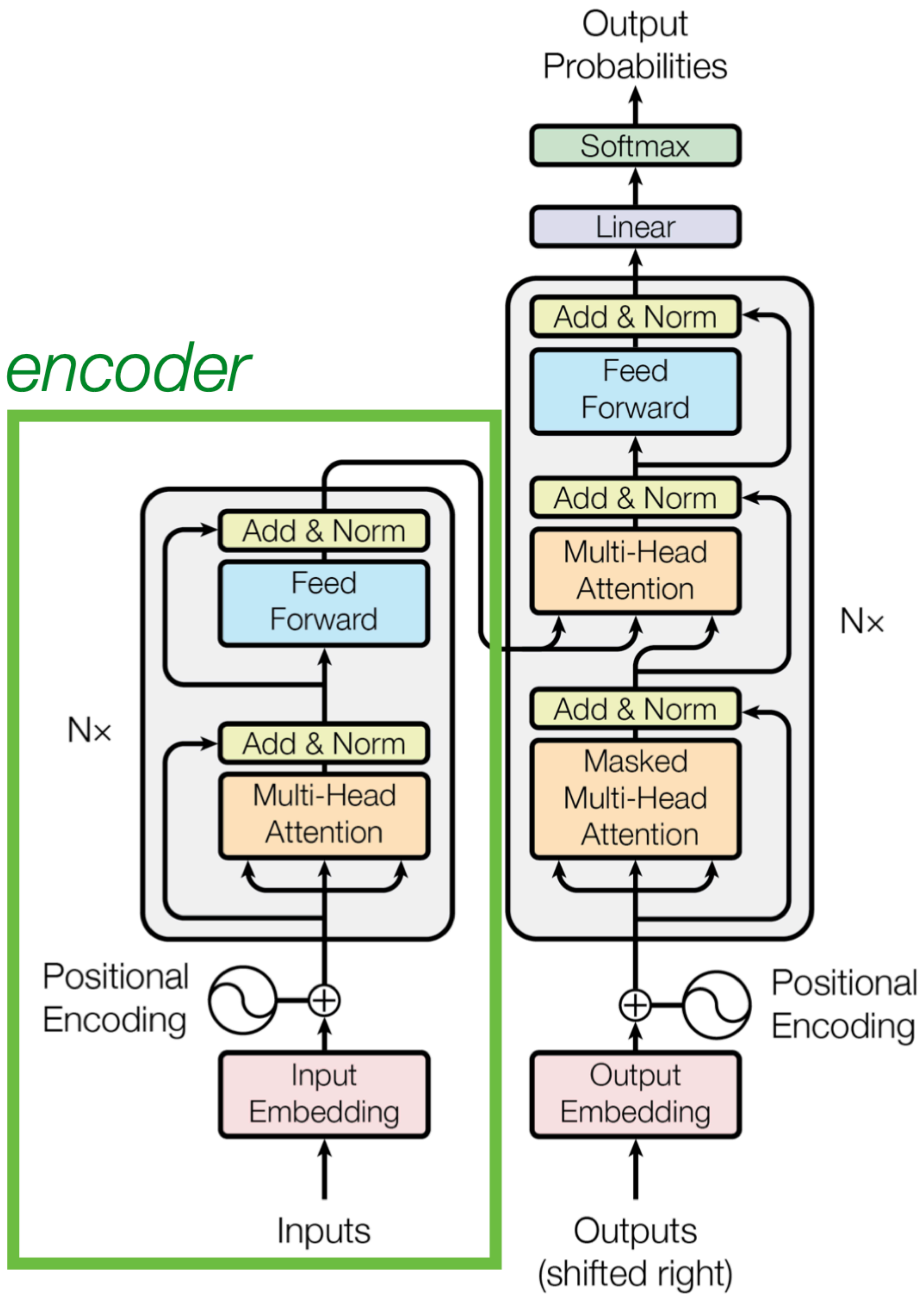
decoder → generate

Currently we only cover the encoder side

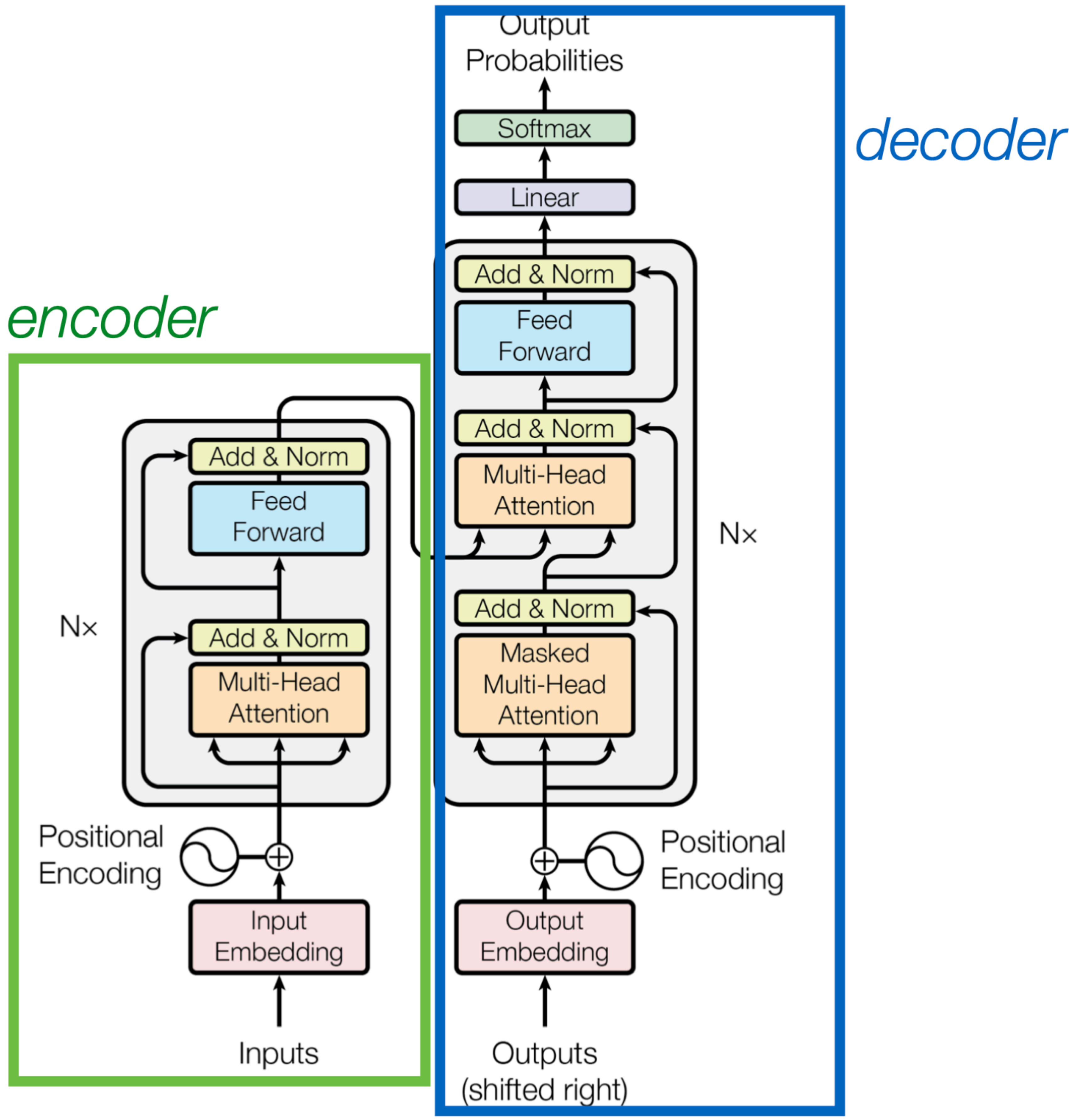




Transformer Encoder

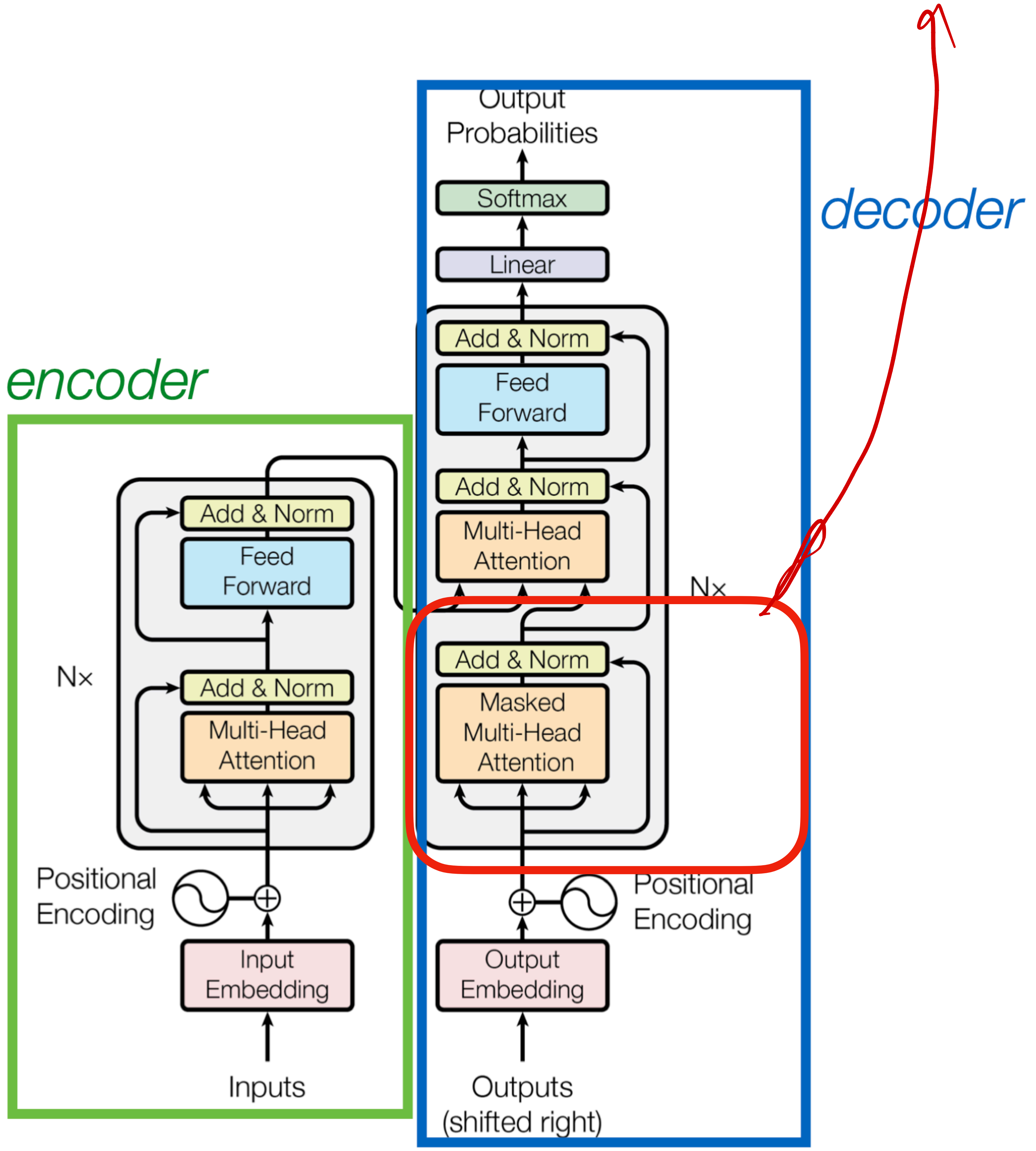


Currently we only cover the encoder side

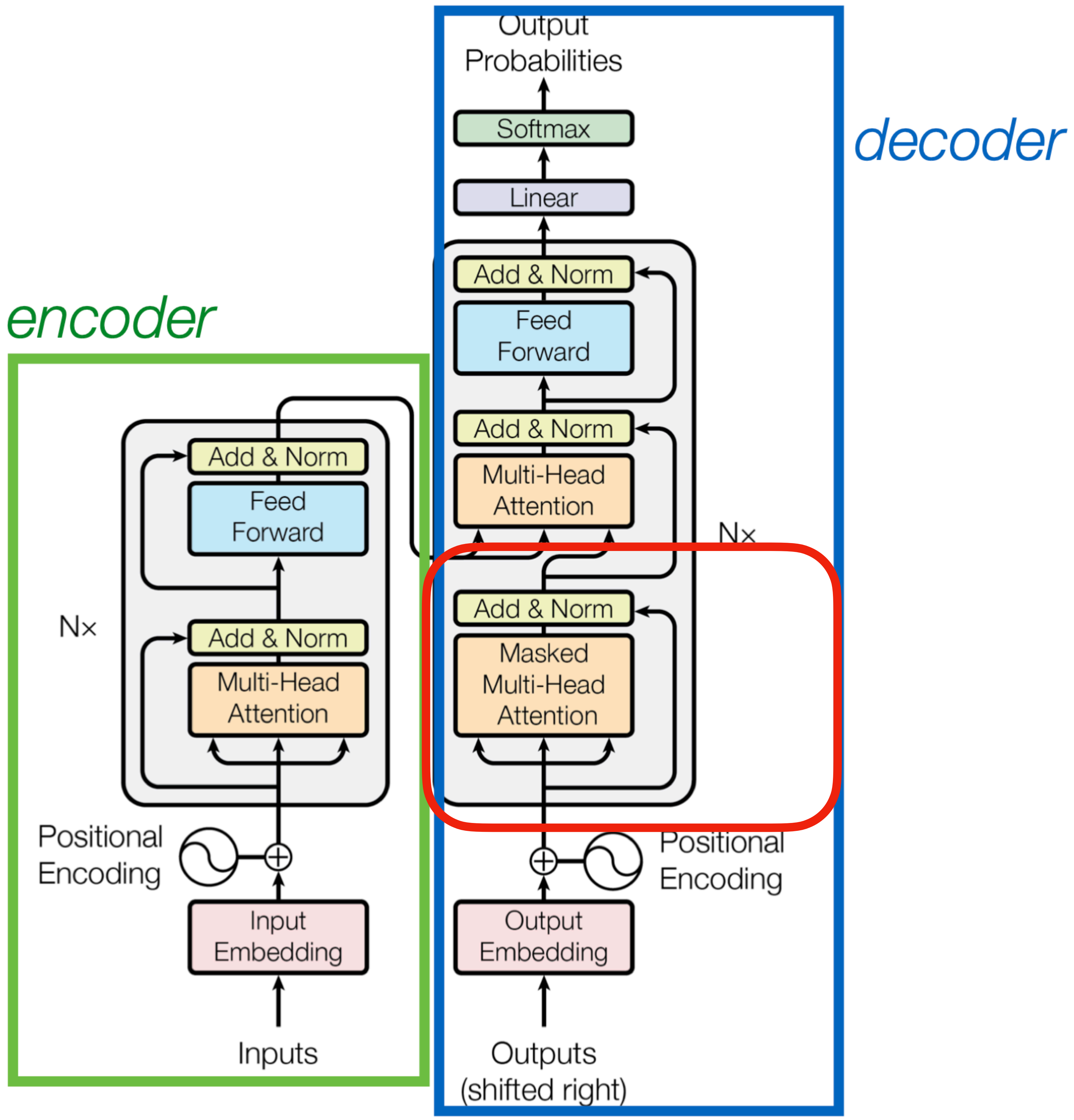


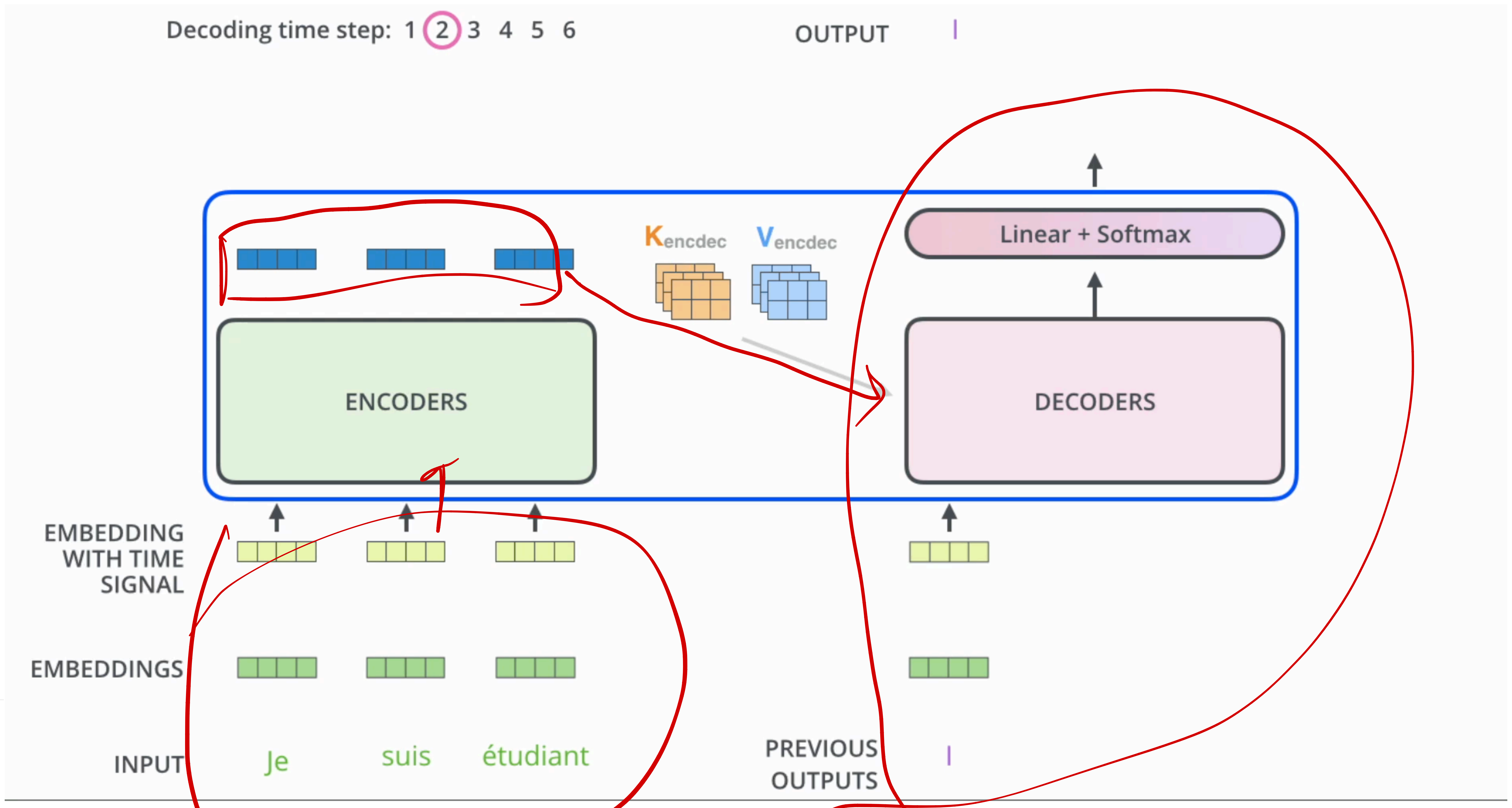
This encoder-decoder arch is originally proposed as a seq2seq arch, for classification tasks, often only encoder is used. And language models often only have a decoder

Masked Attention



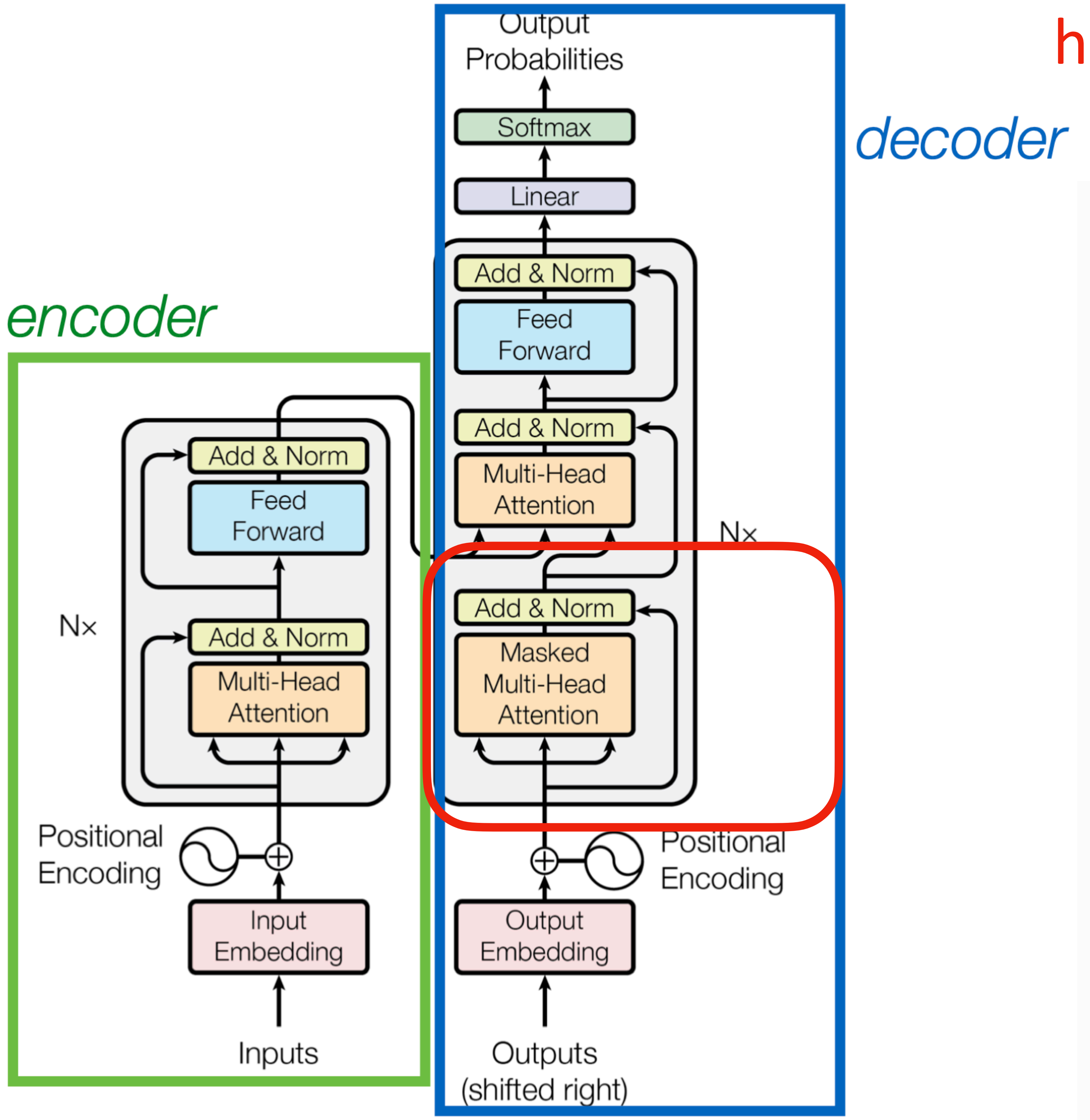
Masked Attention

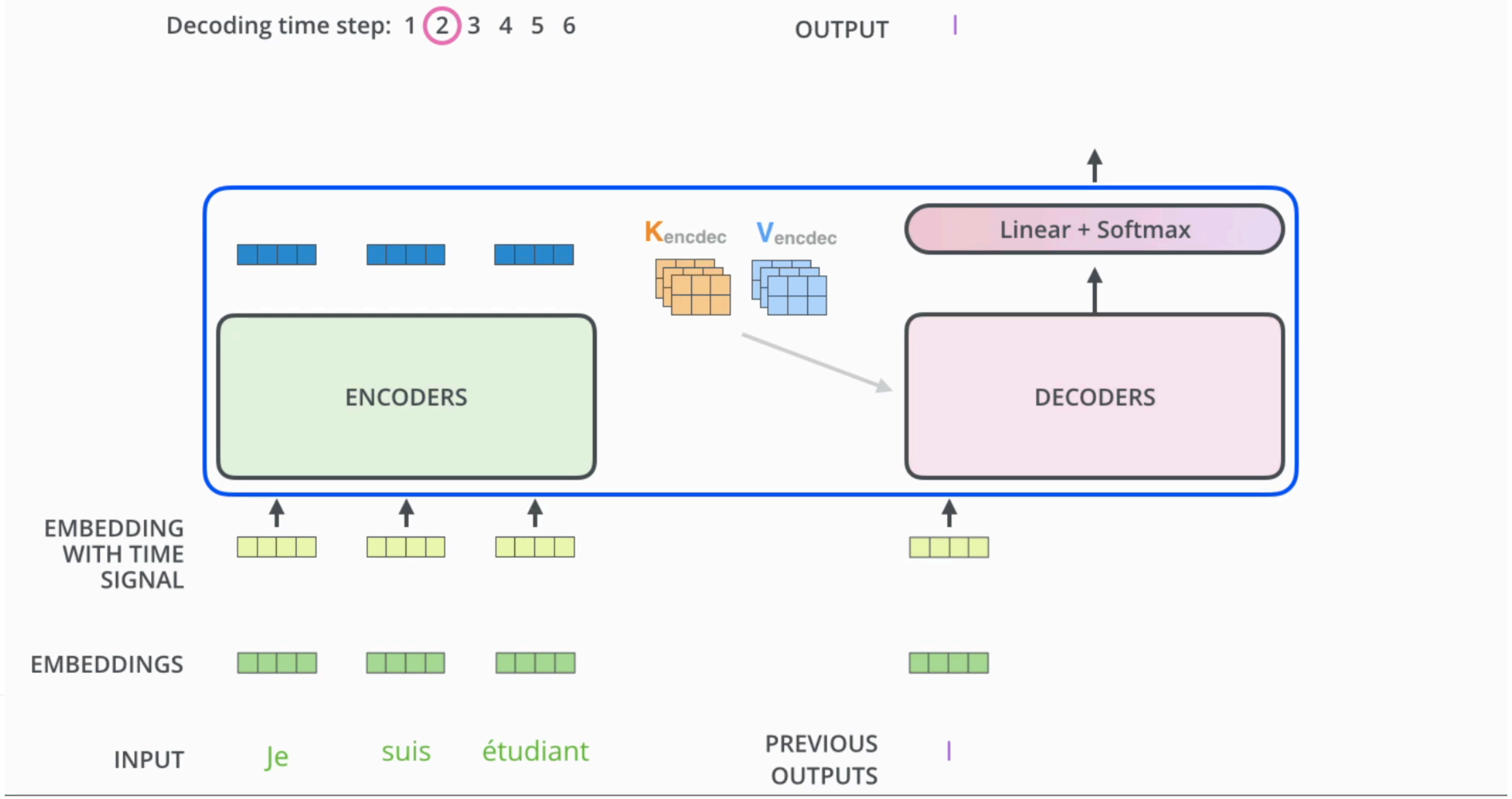




Masked Attention

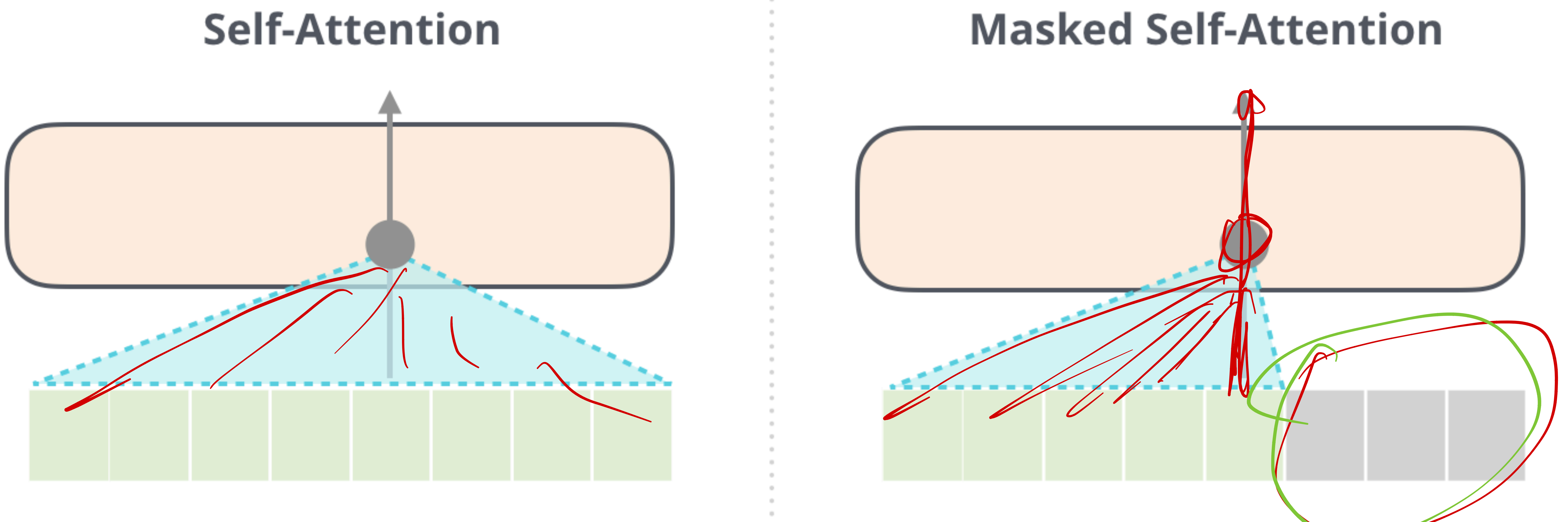
Typical attention attends to the entire sequence, while masked attention only attends to the ones on the left because future words have not been generated





Masked Attention

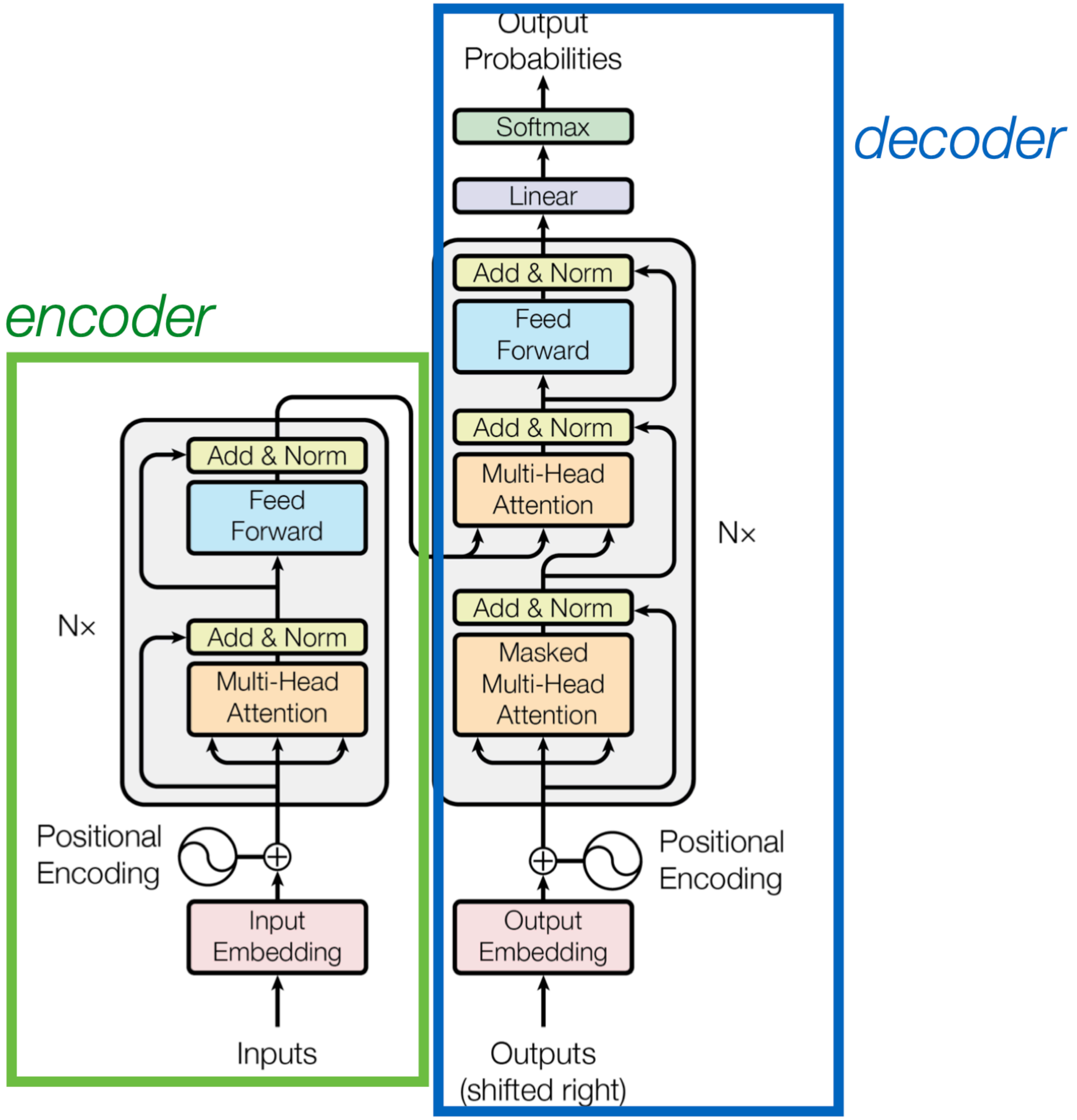
mask implementation? → attention weight = 0



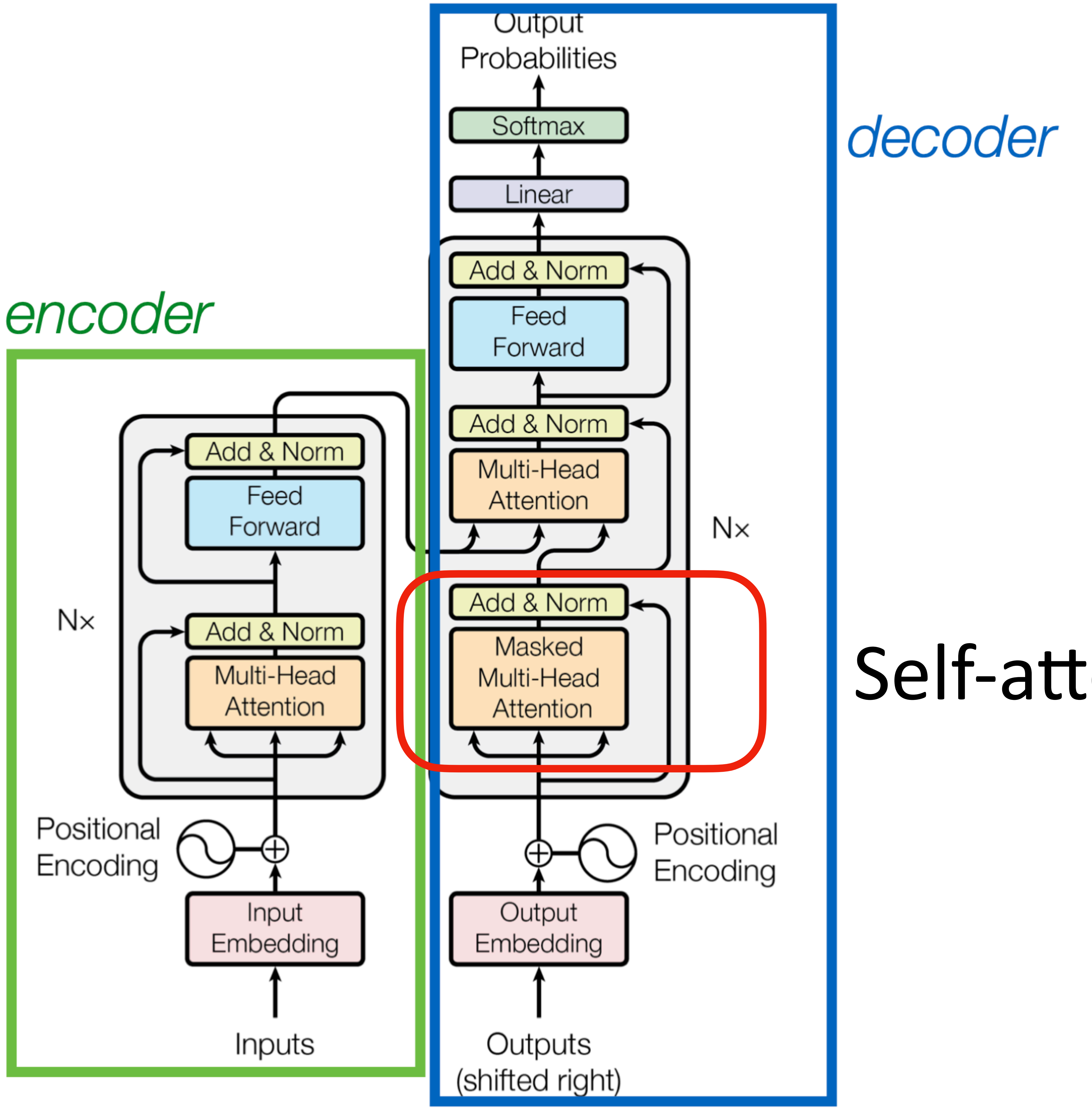
att weight 1 × value 1

+ weight 2 × value 2 + ...

Transformer Decoder in Seq2Seq

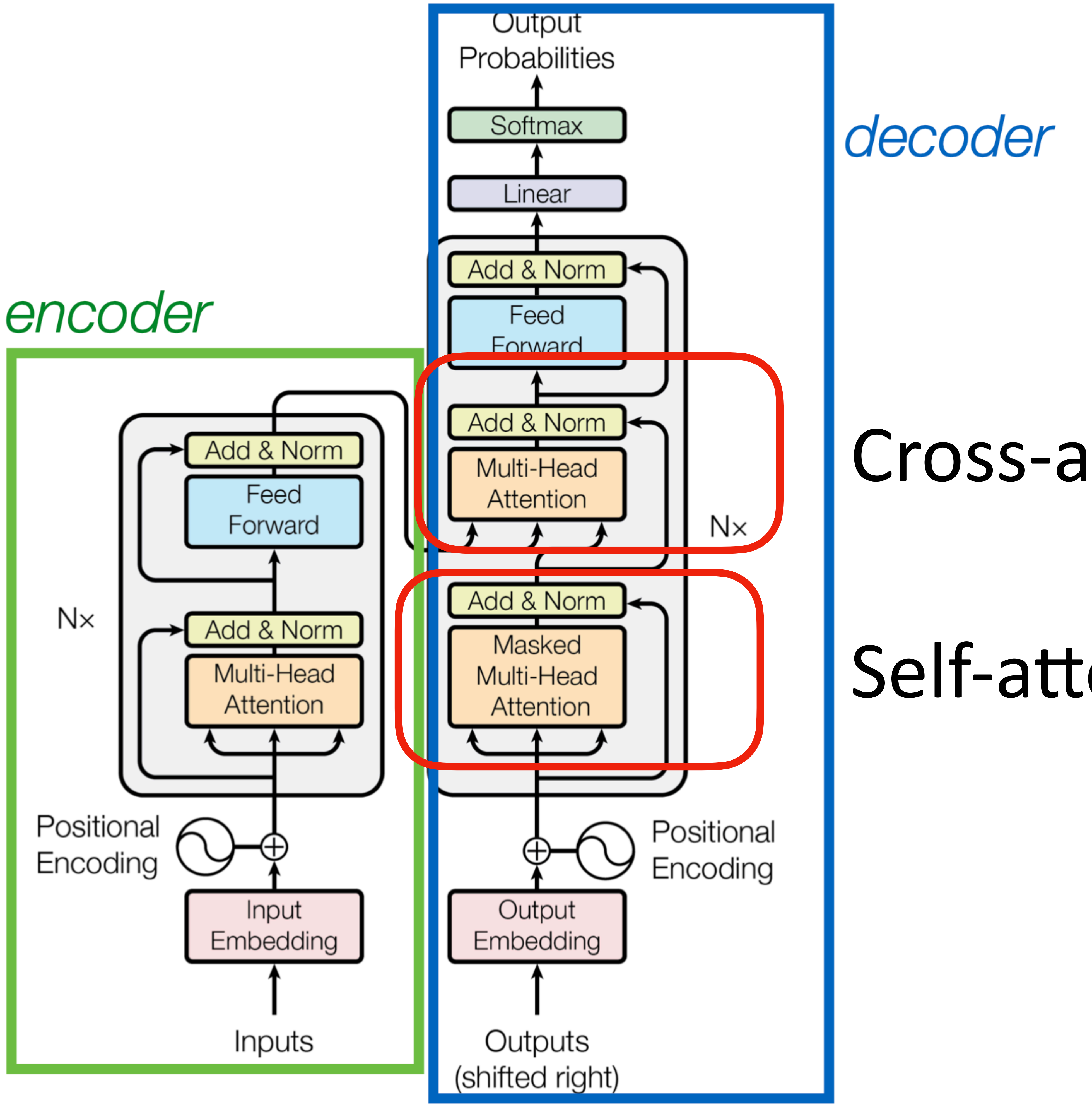


Transformer Decoder in Seq2Seq



Self-attention

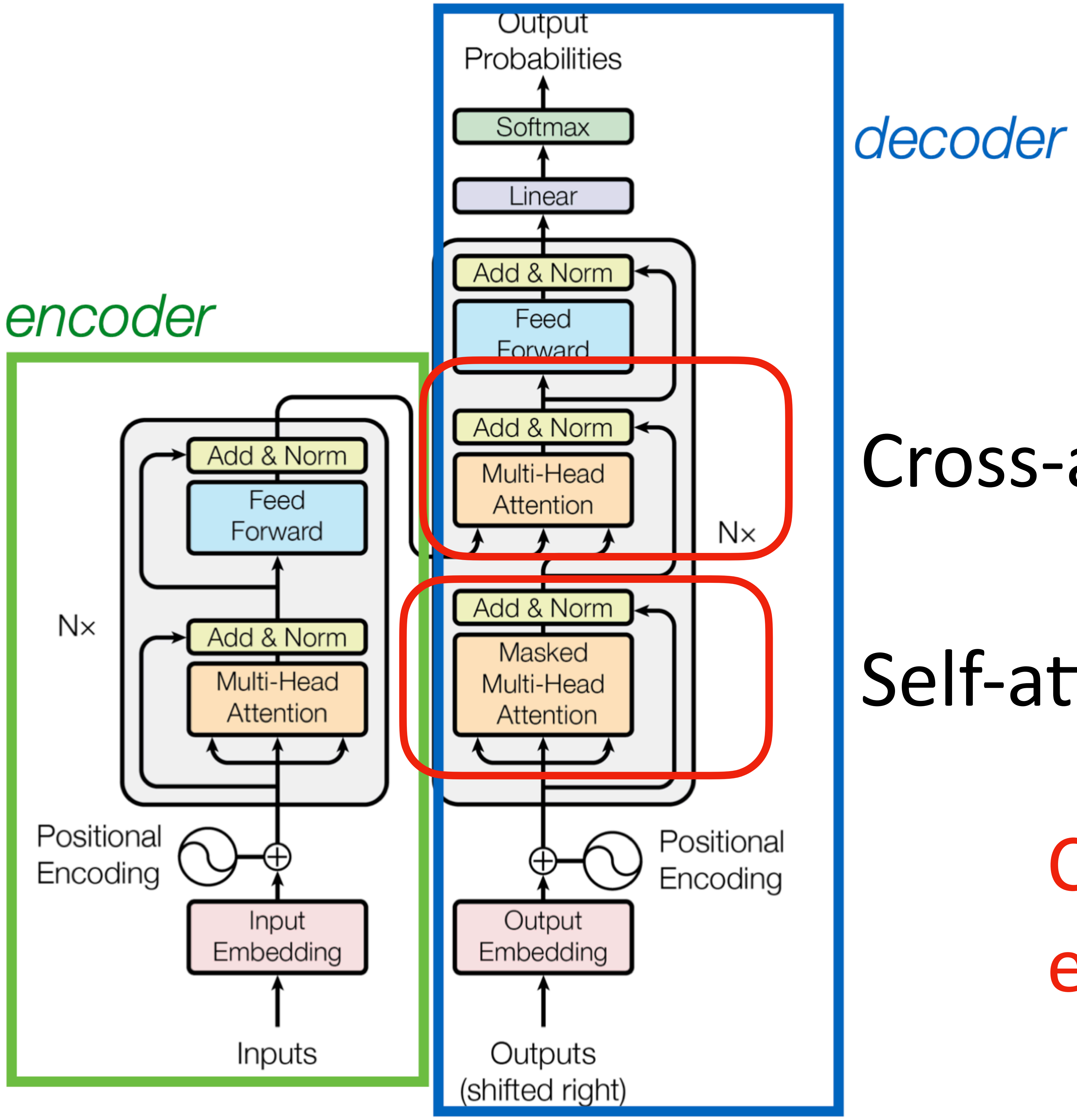
Transformer Decoder in Seq2Seq



Cross-attention

Self-attention

Transformer Decoder in Seq2Seq

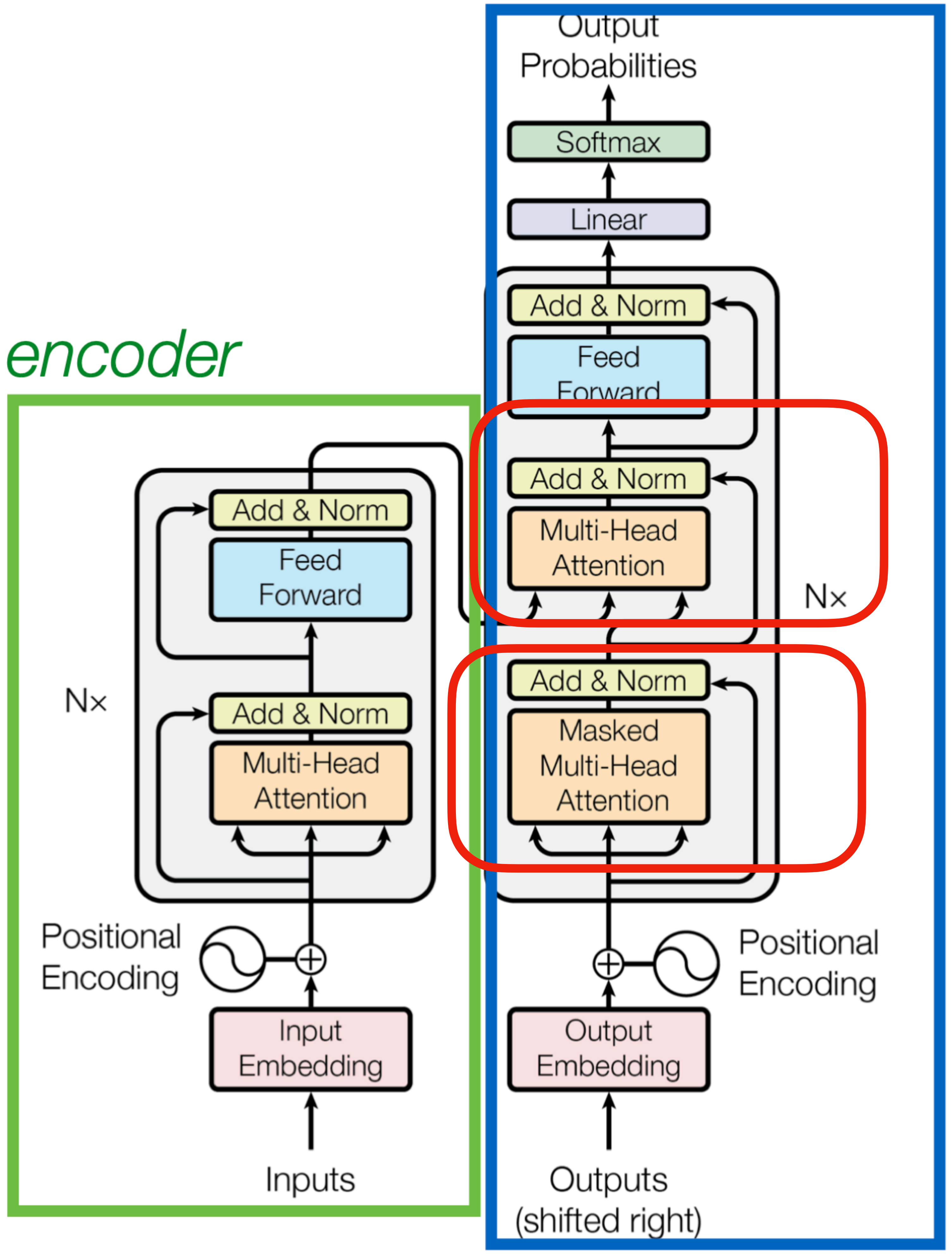


Cross-attention

Self-attention

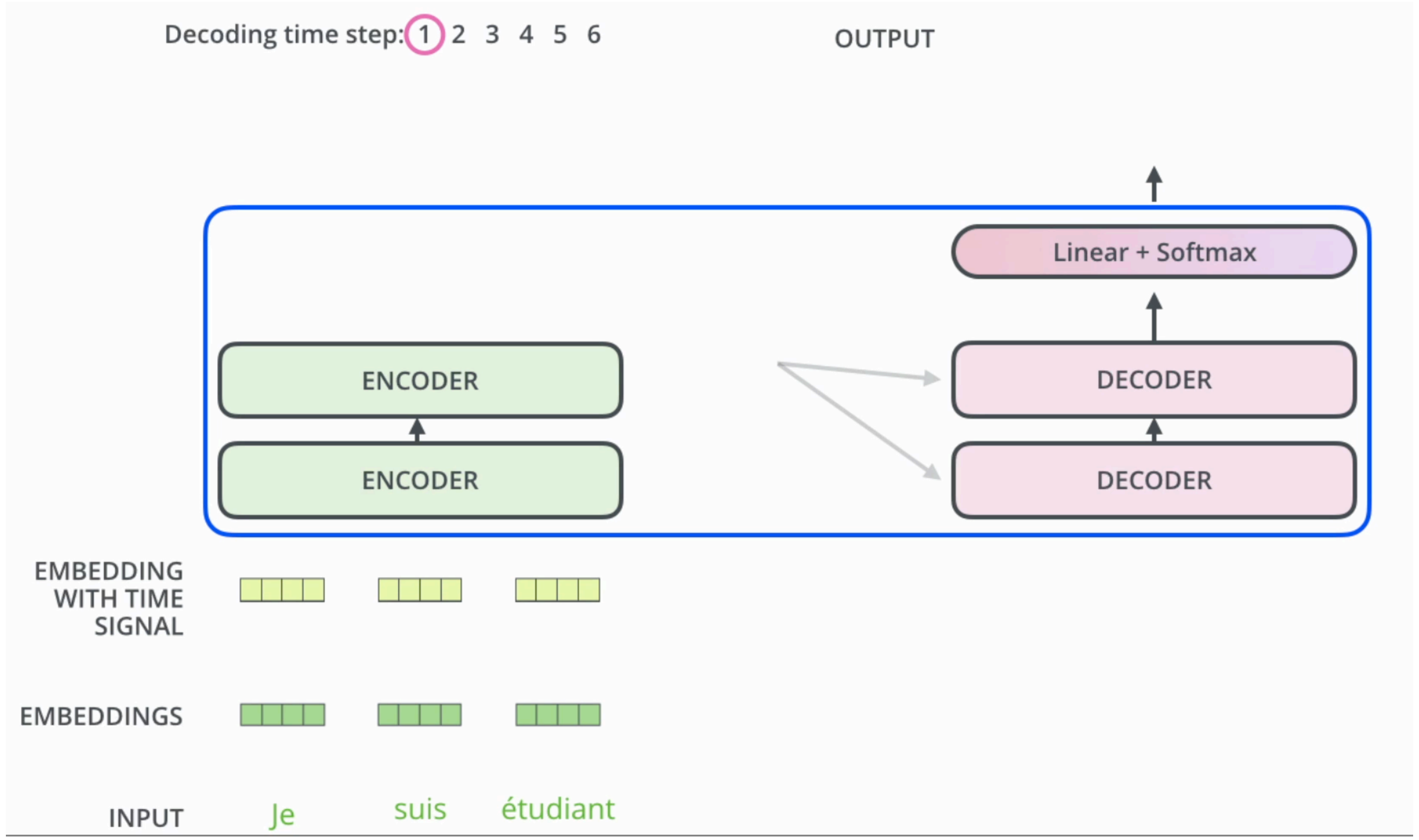
Cross-attention uses the output of encoder as input

Transformer Decoder in Seq2Seq



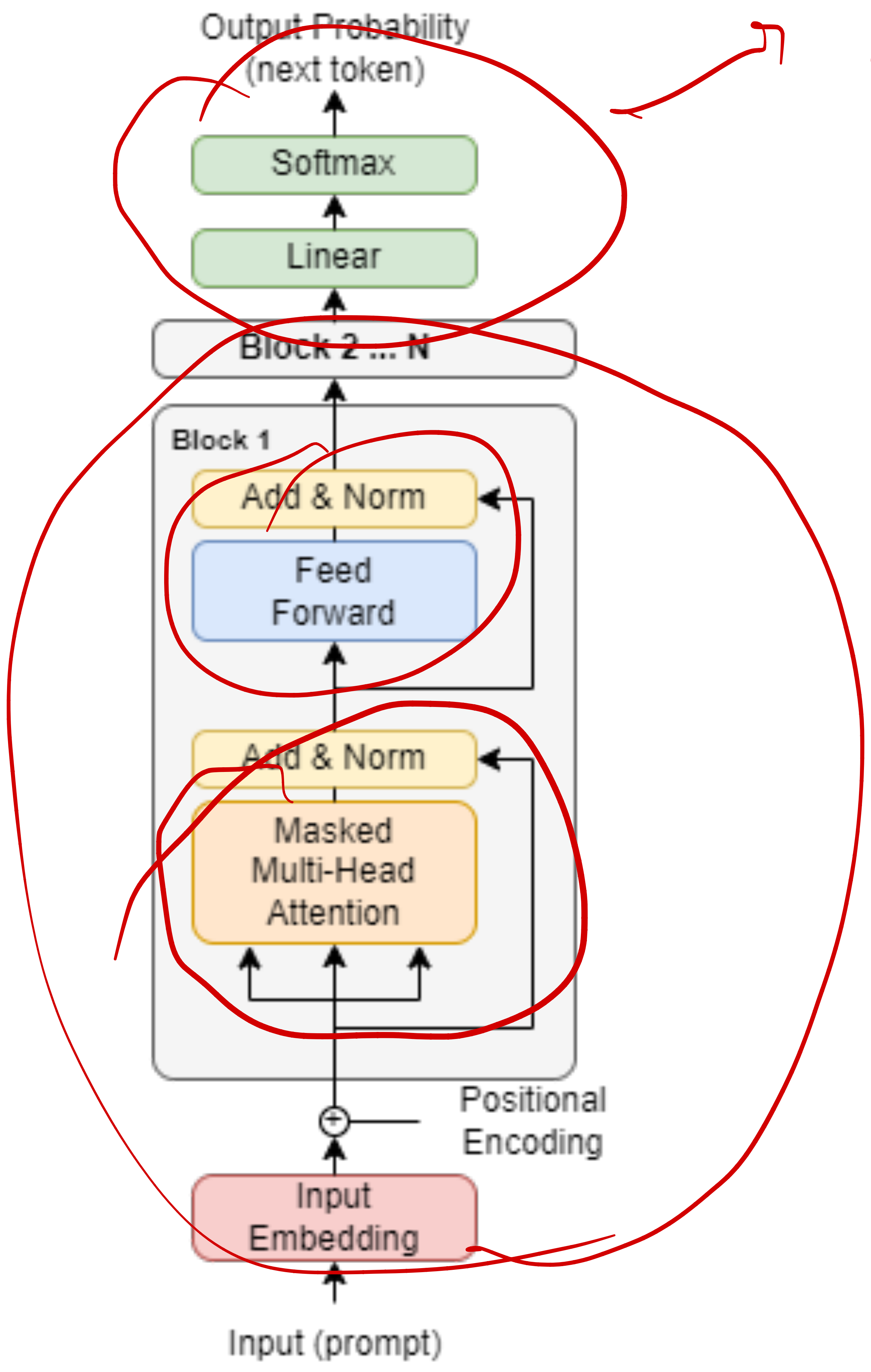
Cross-attention

Self-attention

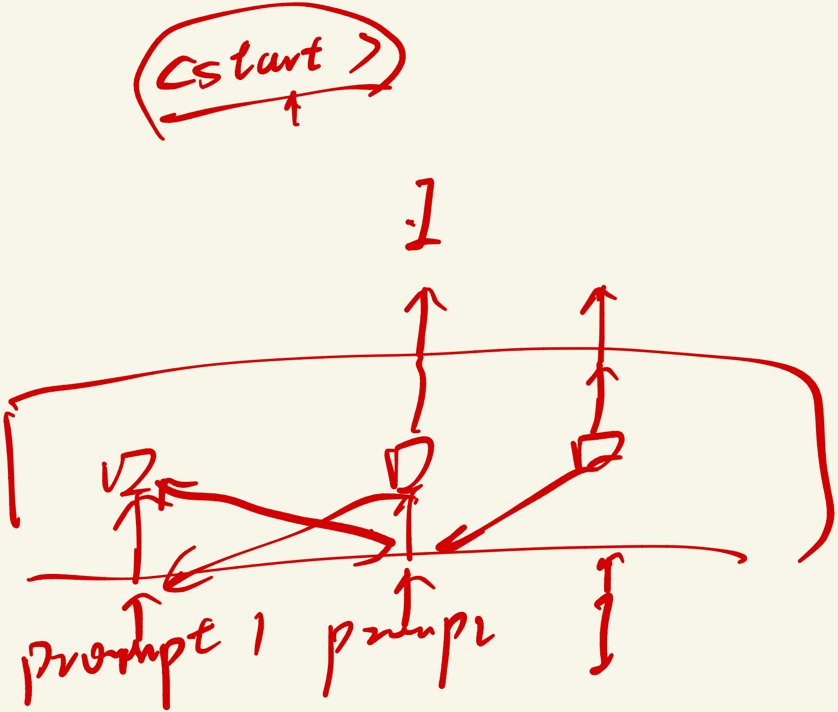


Cross-attention uses the output of encoder as input

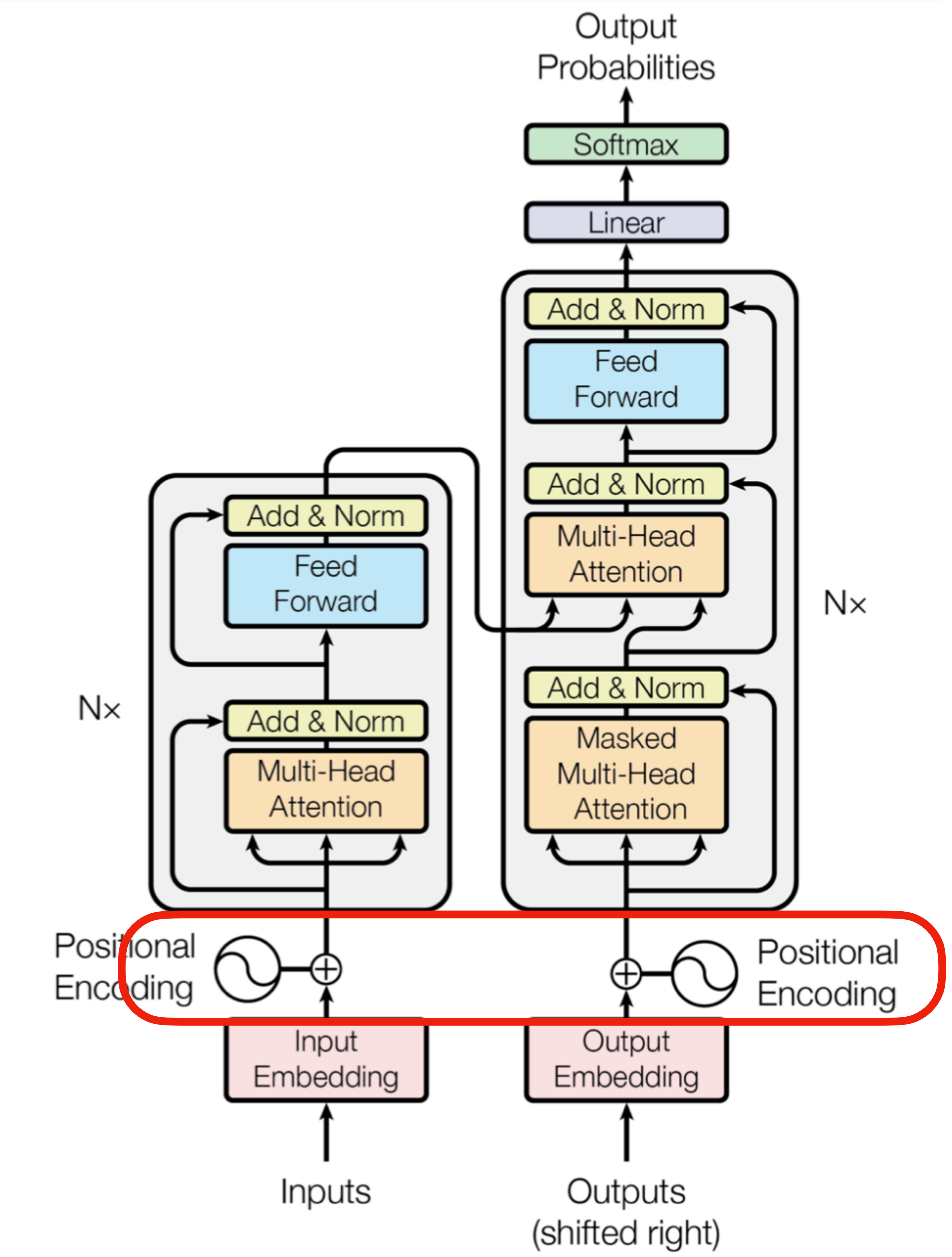
Transformer Language Model (e.g., ChatGPT)



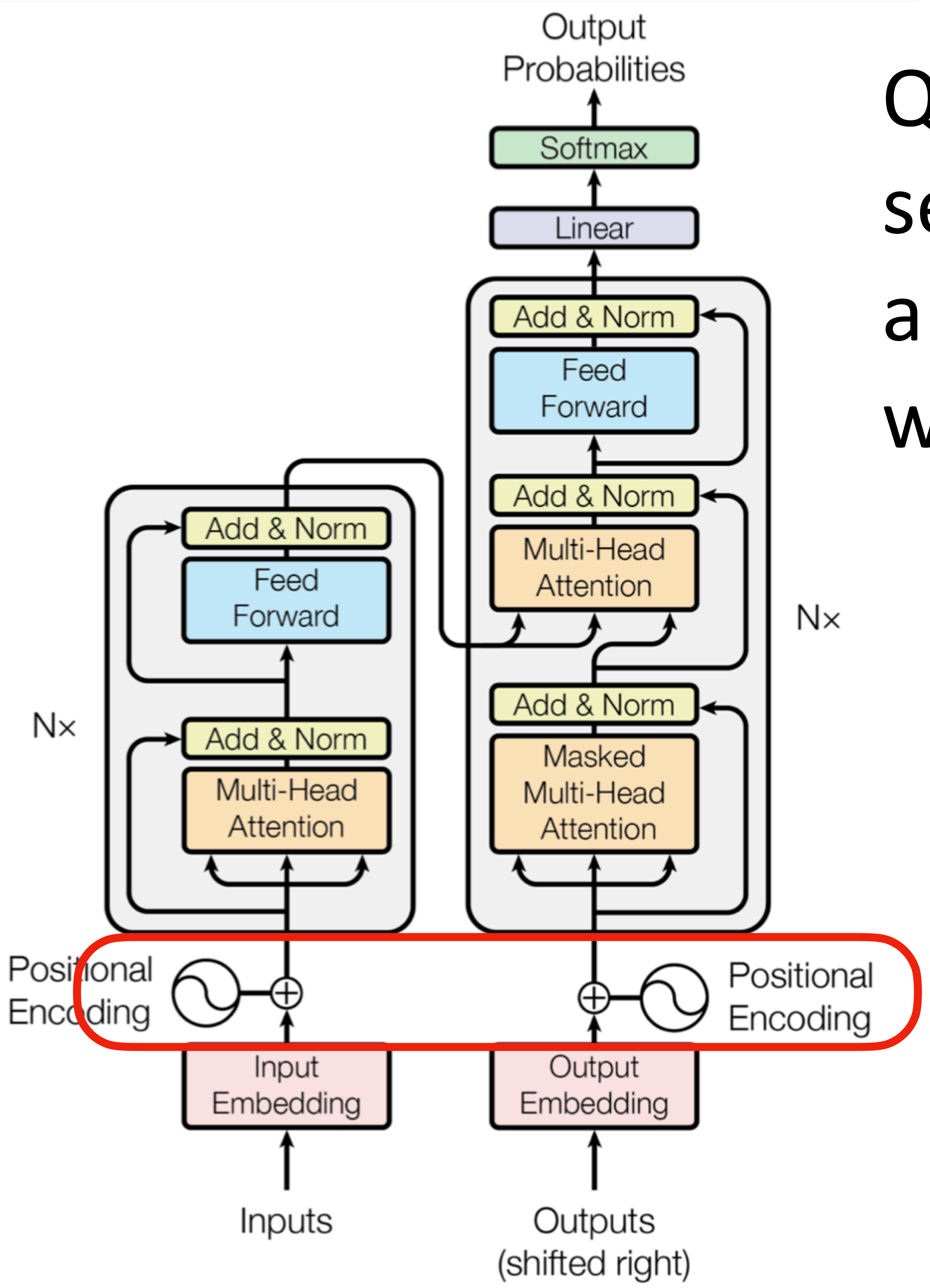
softmax


<start>
1
prompt 1

Position Embeddings

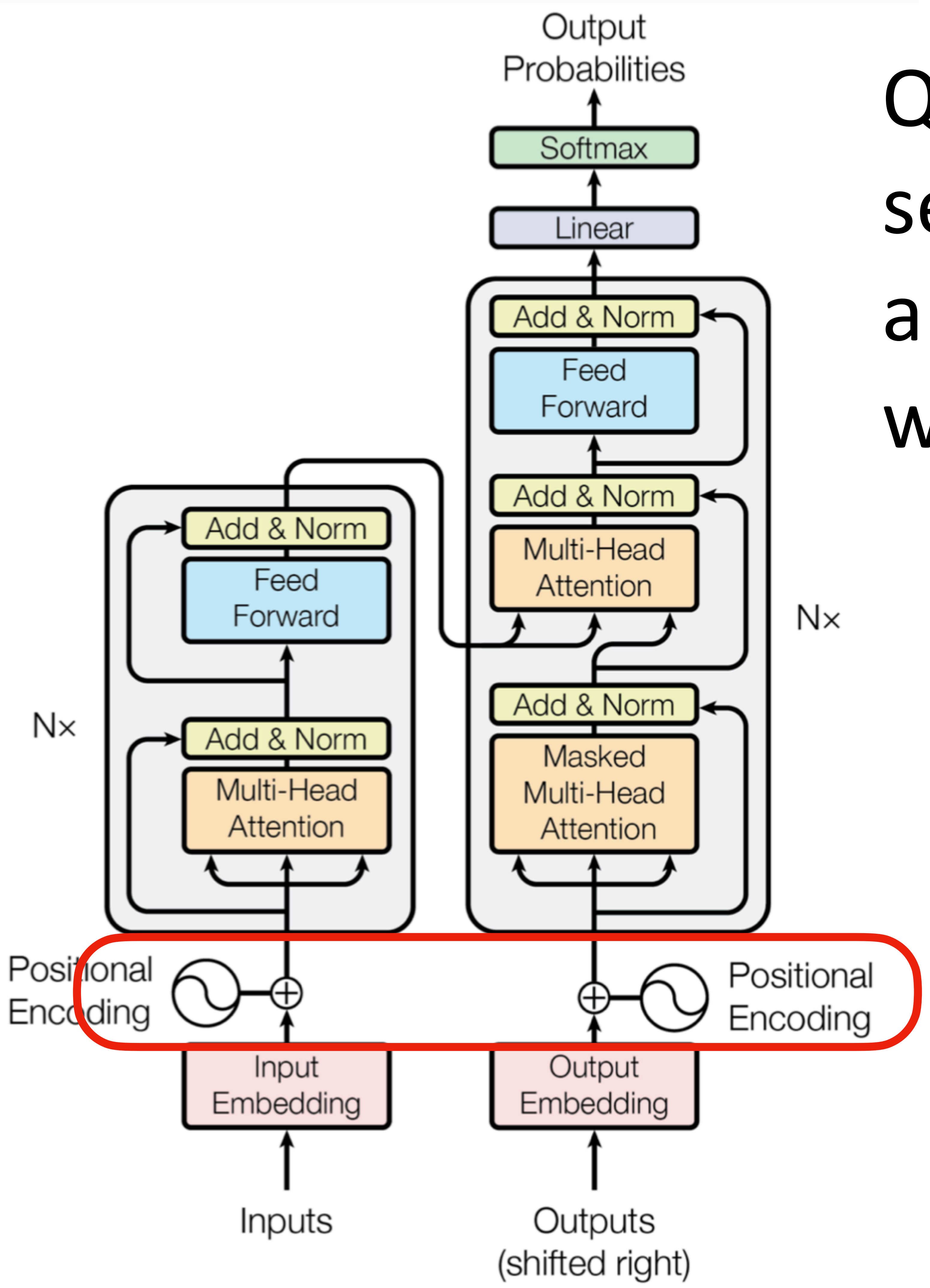


Position Embeddings



Question: If we shuffle the order of words in the sequence, will that change the attention output and feed forward output of the corresponding word?

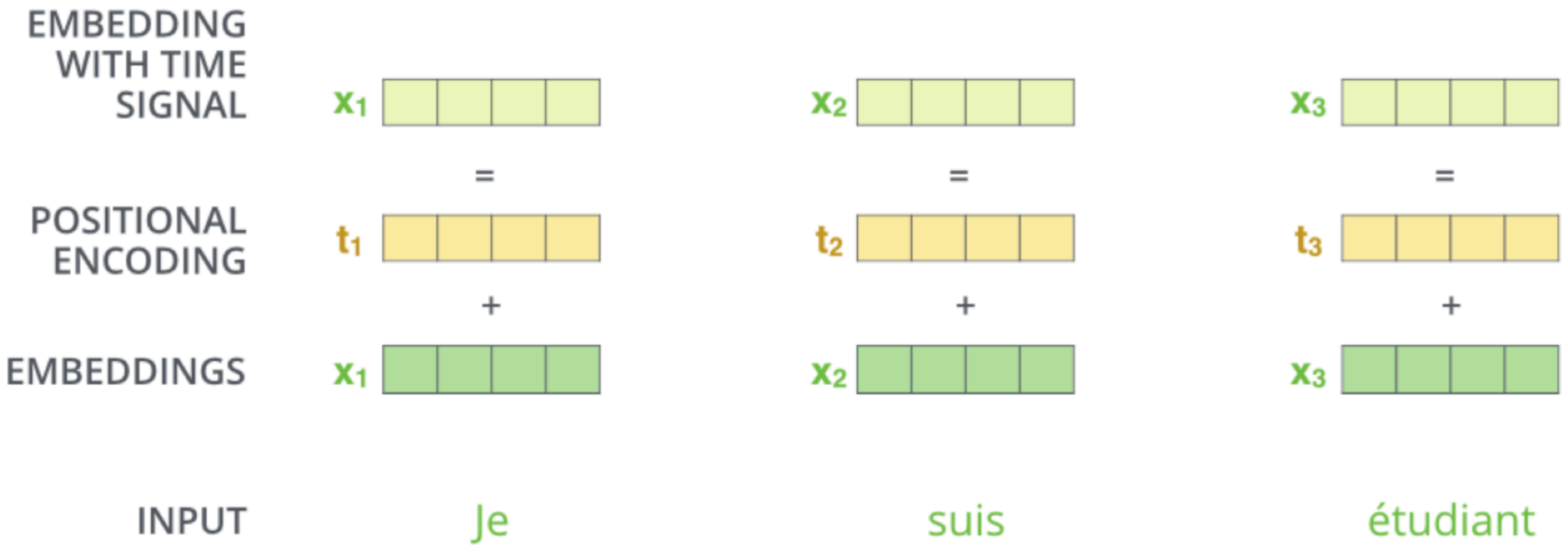
Position Embeddings



Question: If we shuffle the order of words in the sequence, will that change the attention output and feed forward output of the corresponding word?

Position embeddings are added to each word embedding, otherwise our model is unaware of the position of a word

Positional Encoding



Transformer Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Positional encoding is a 512d vector
i = a particular dimension of this vector
pos = dimension of the word
d_model = 512

Complexity

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

n is sequence length, d is embedding dimension.

Complexity

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

n is sequence length, d is embedding dimension.

Restricted self-attention means not attending all words in the sequence, but only a restricted field

Complexity

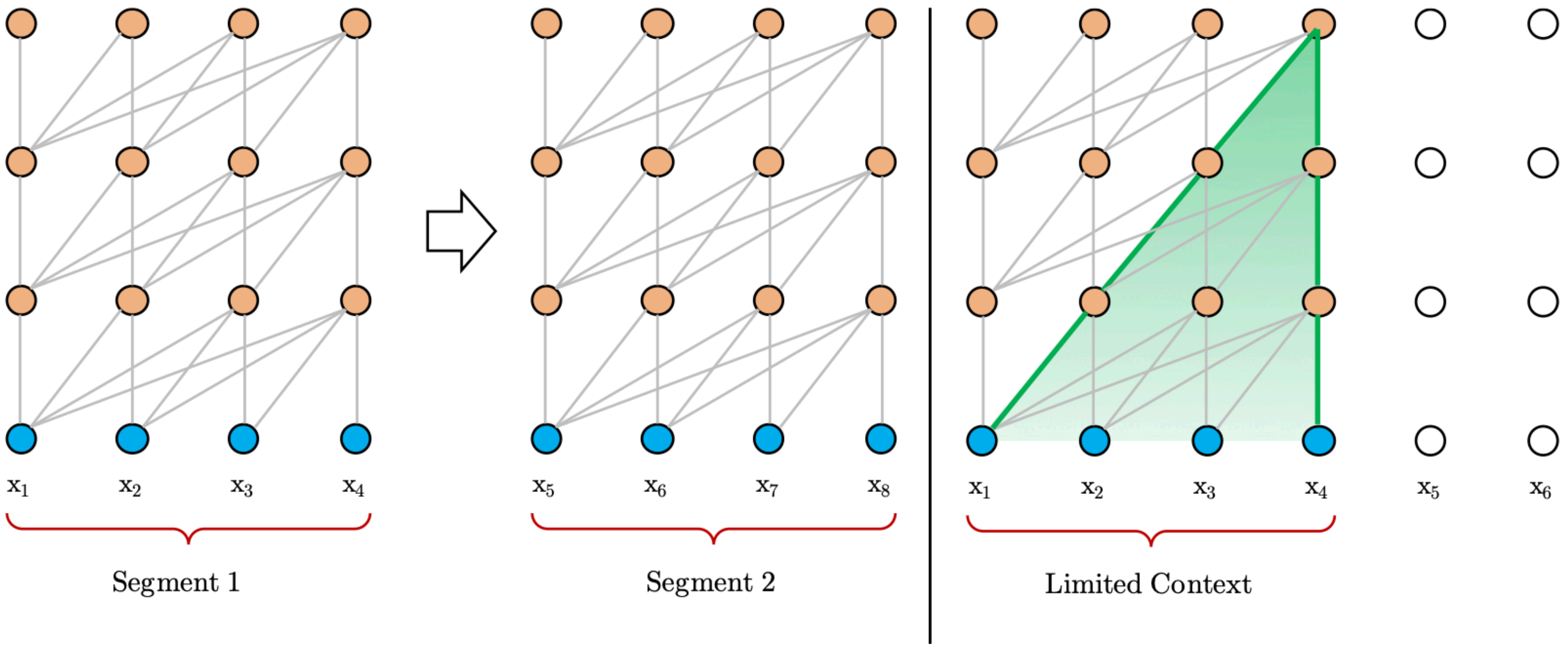
Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

n is sequence length, d is embedding dimension.

Restricted self-attention means not attending all words in the sequence, but only a restricted field

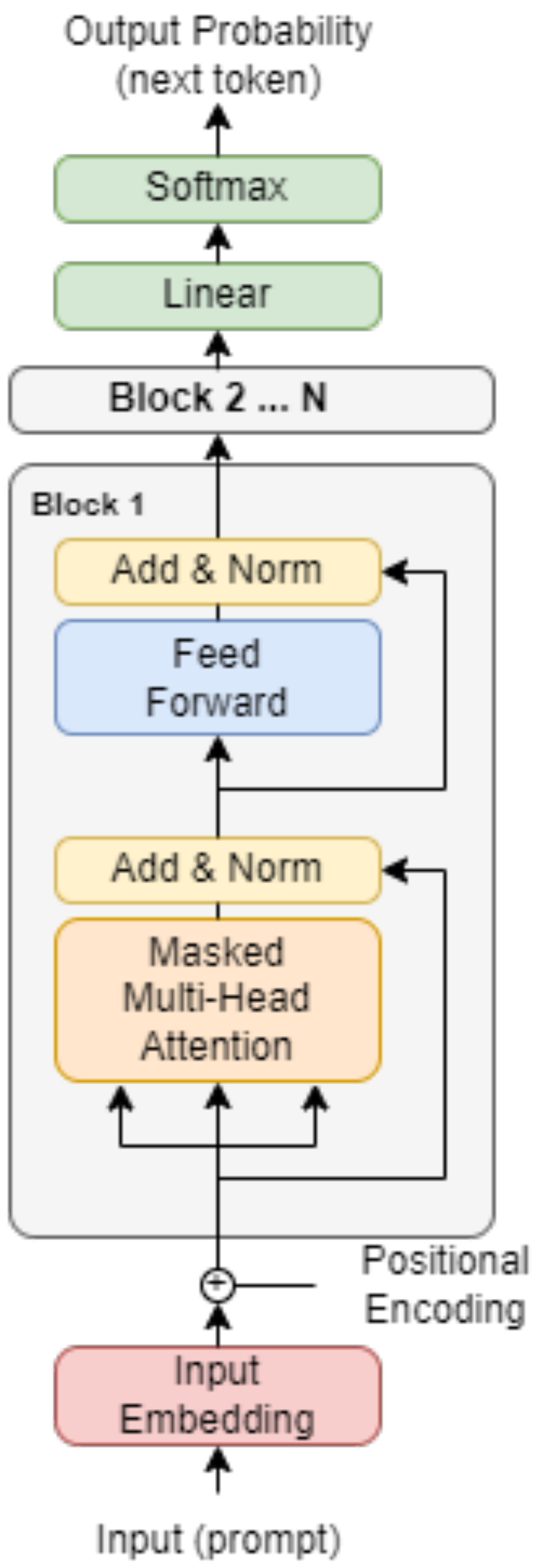
Square complexity of sequence length is a major issue for transformers to deal with long sequence

Language Model Training with Limited Context



Dai et al. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. 2019.

Transformer Language Model (e.g., ChatGPT)



Language Model Pretraining

Pretraining

Pretraining

Target Data B

Pretraining

Source Data A (maybe a different task)

Target Data B

Pretraining

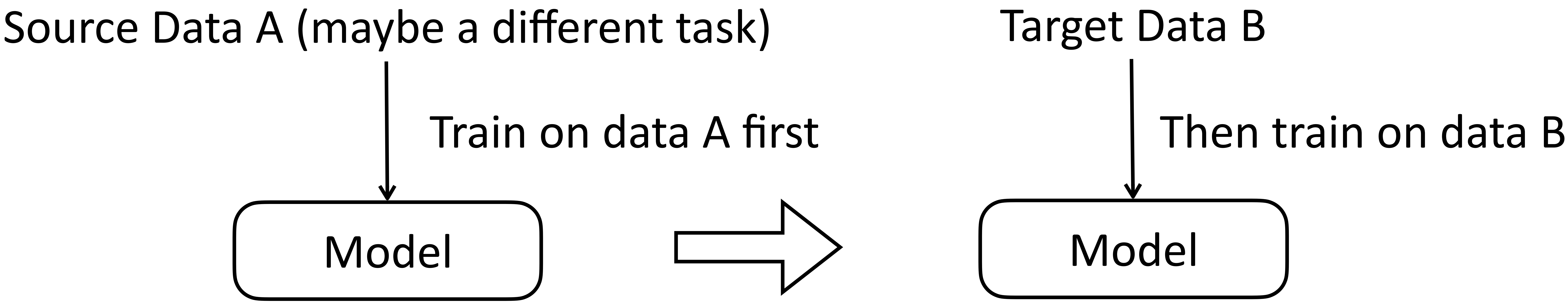
Source Data A (maybe a different task)

Target Data B

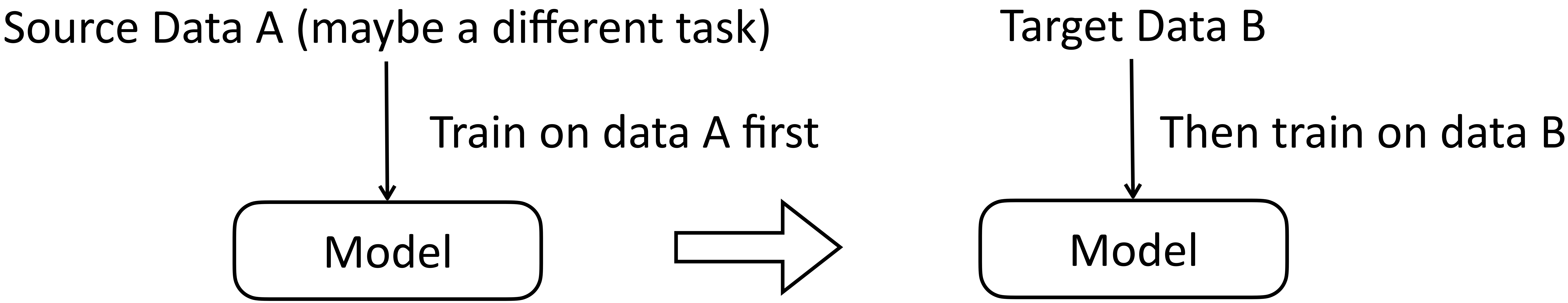
Train on data A first

Model

Pretraining

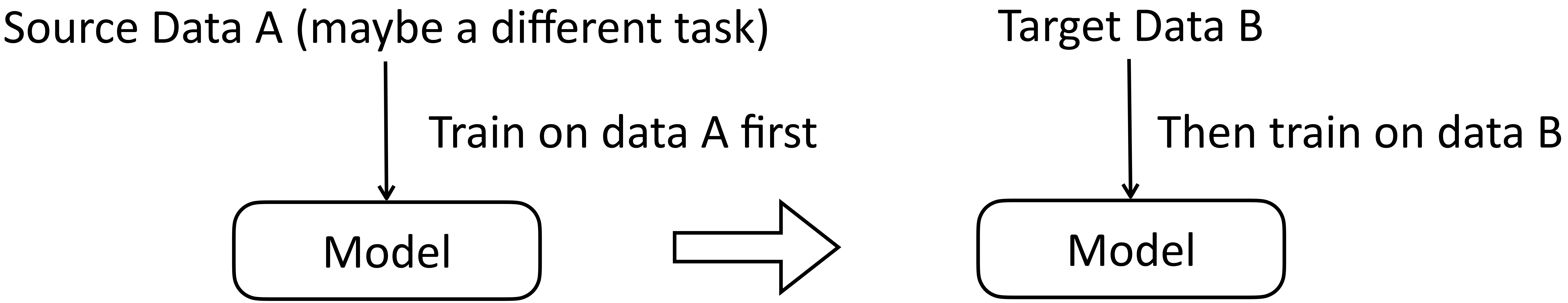


Pretraining



Classically, this is transfer Learning

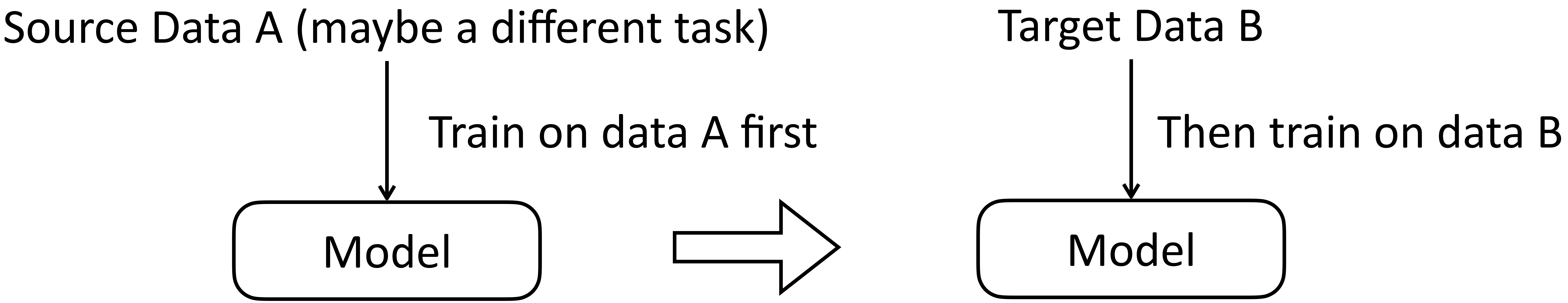
Pretraining



Classically, this is transfer Learning

It is now called pretraining because of the scale of A

Pretraining



For supervised training, data A is often limited

How can we find large-scale data A to train?

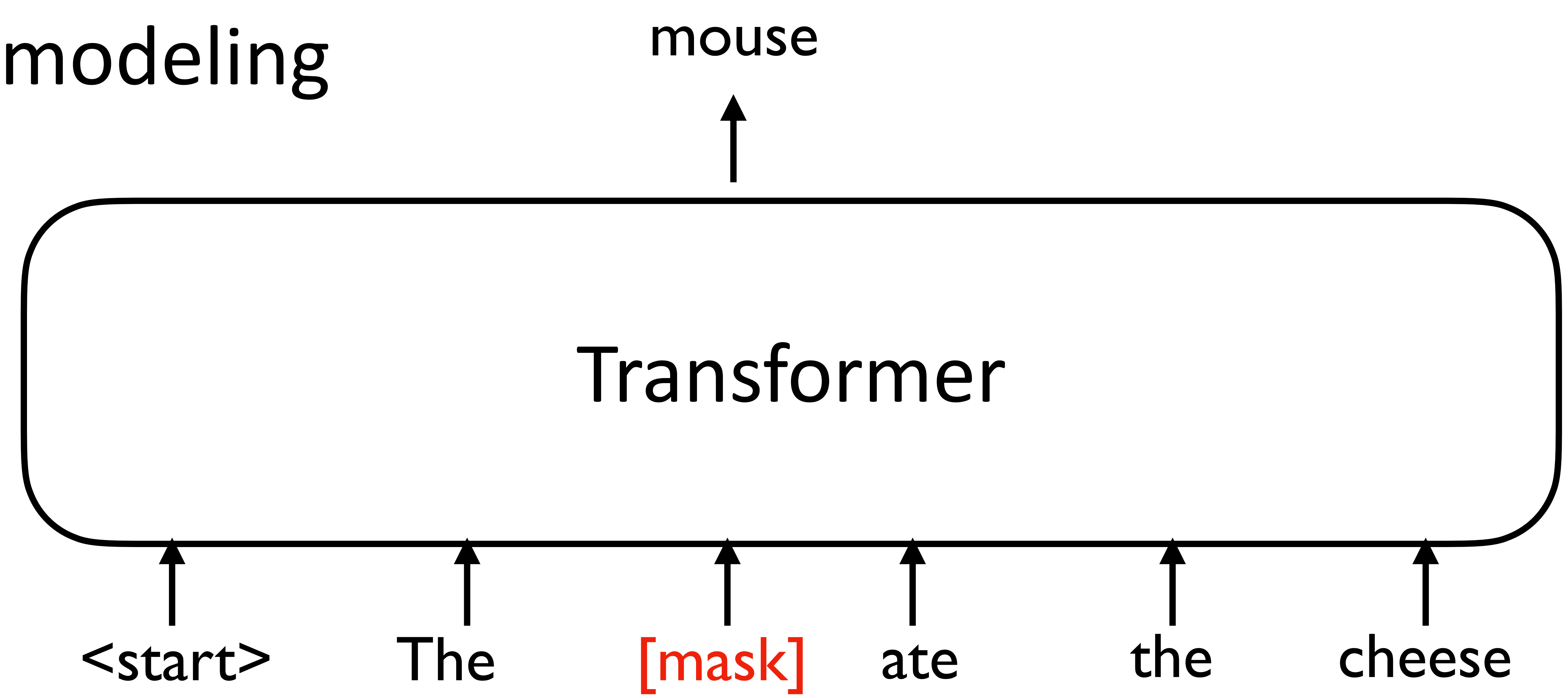
BERT

Mask language modeling

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

BERT

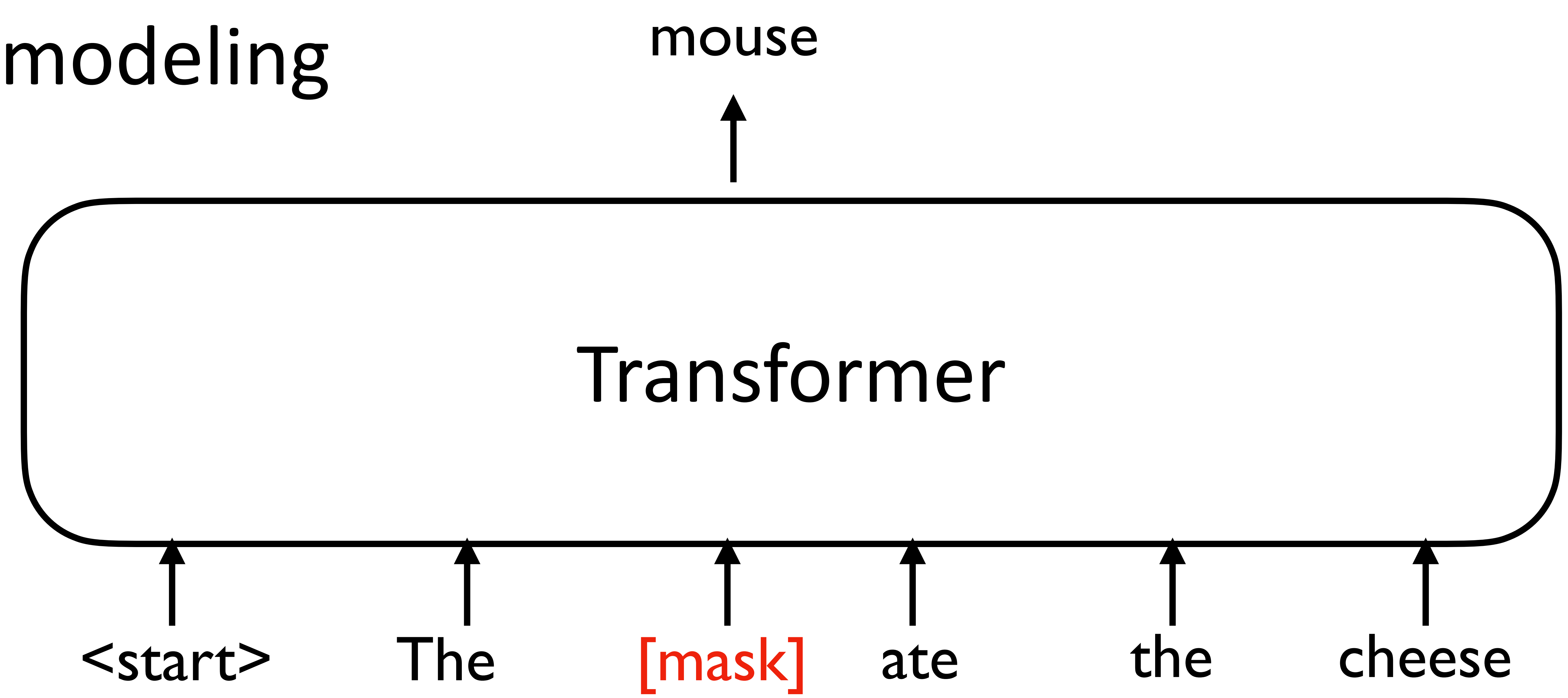
Mask language modeling



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

BERT

Mask language modeling

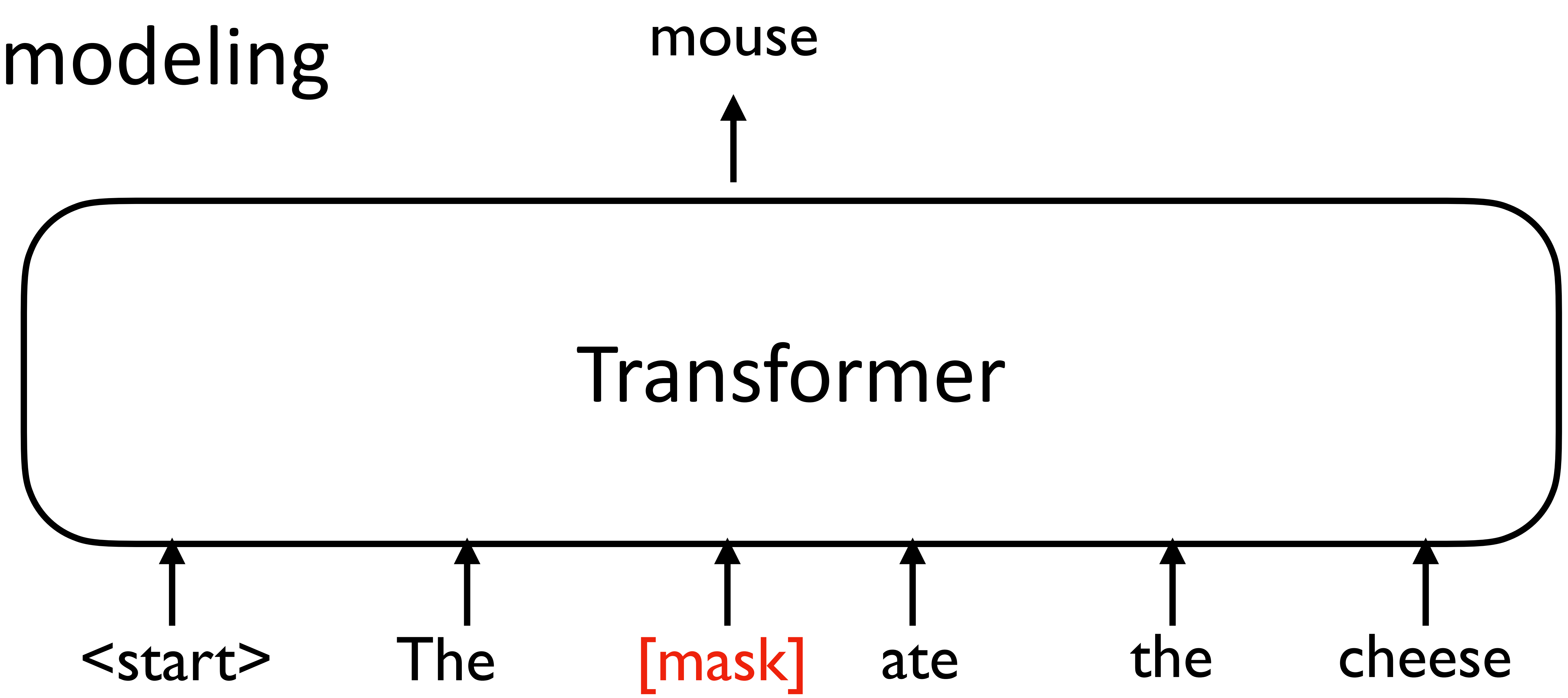


Construct a synthetic task from raw text only

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

BERT

Mask language modeling



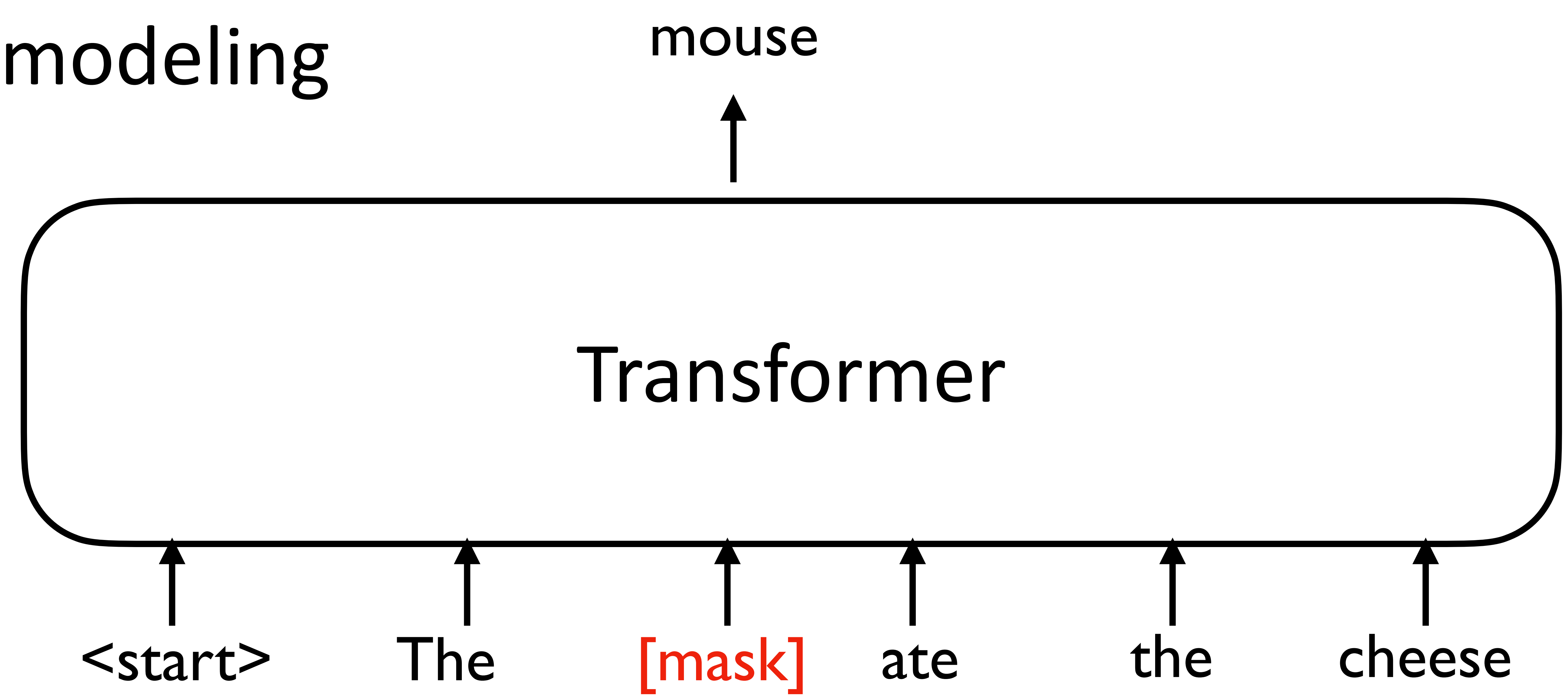
Self-supervised Learning

Construct a synthetic task from raw text only

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

BERT

Mask language modeling



Self-supervised Learning

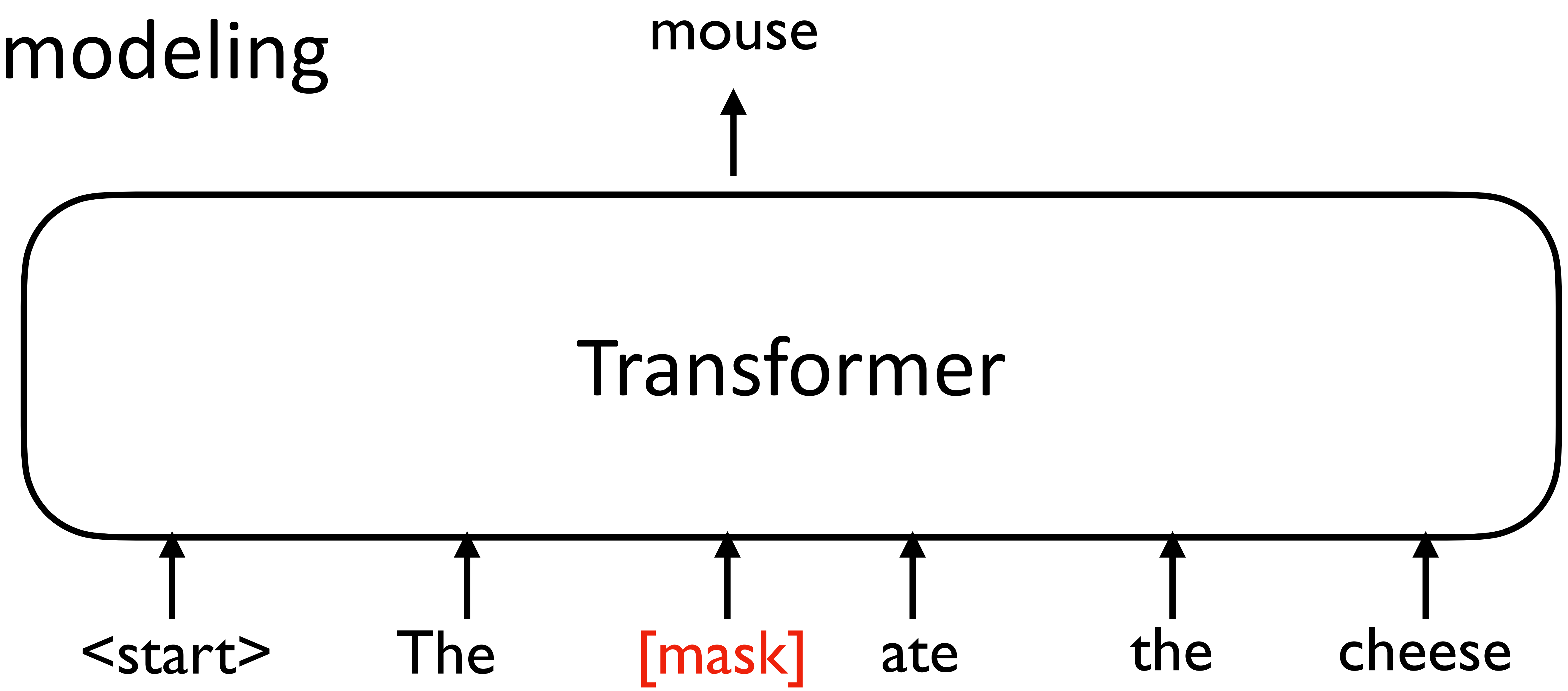
Construct a synthetic task from raw text only

Can be made very large-scale

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

BERT

Mask language modeling



Self-supervised Learning

Construct a synthetic task from raw text only

Can be made very large-scale

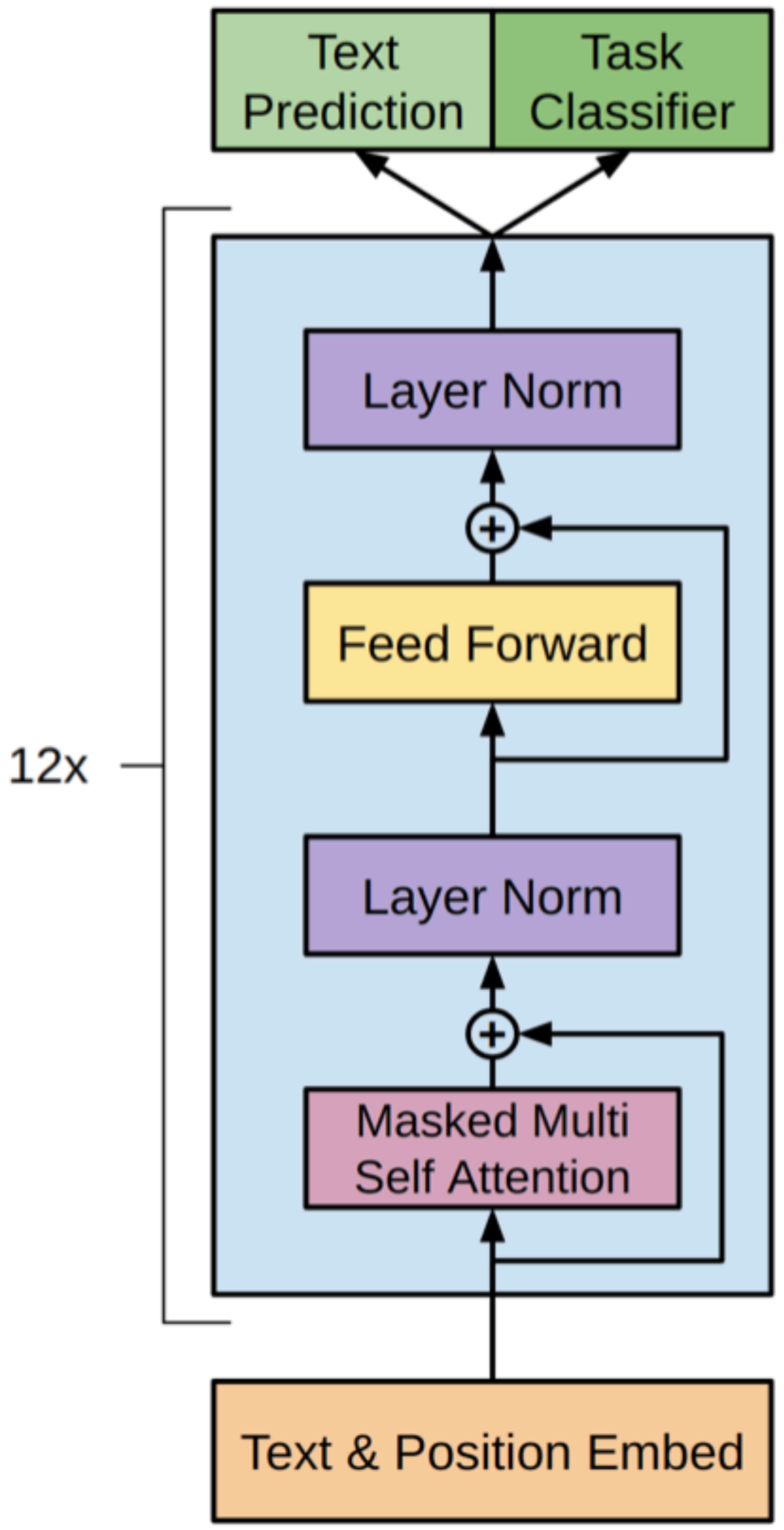
Is Bert a language model? Is it a generative model?

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019.

Generative Pre-Training (GPT)

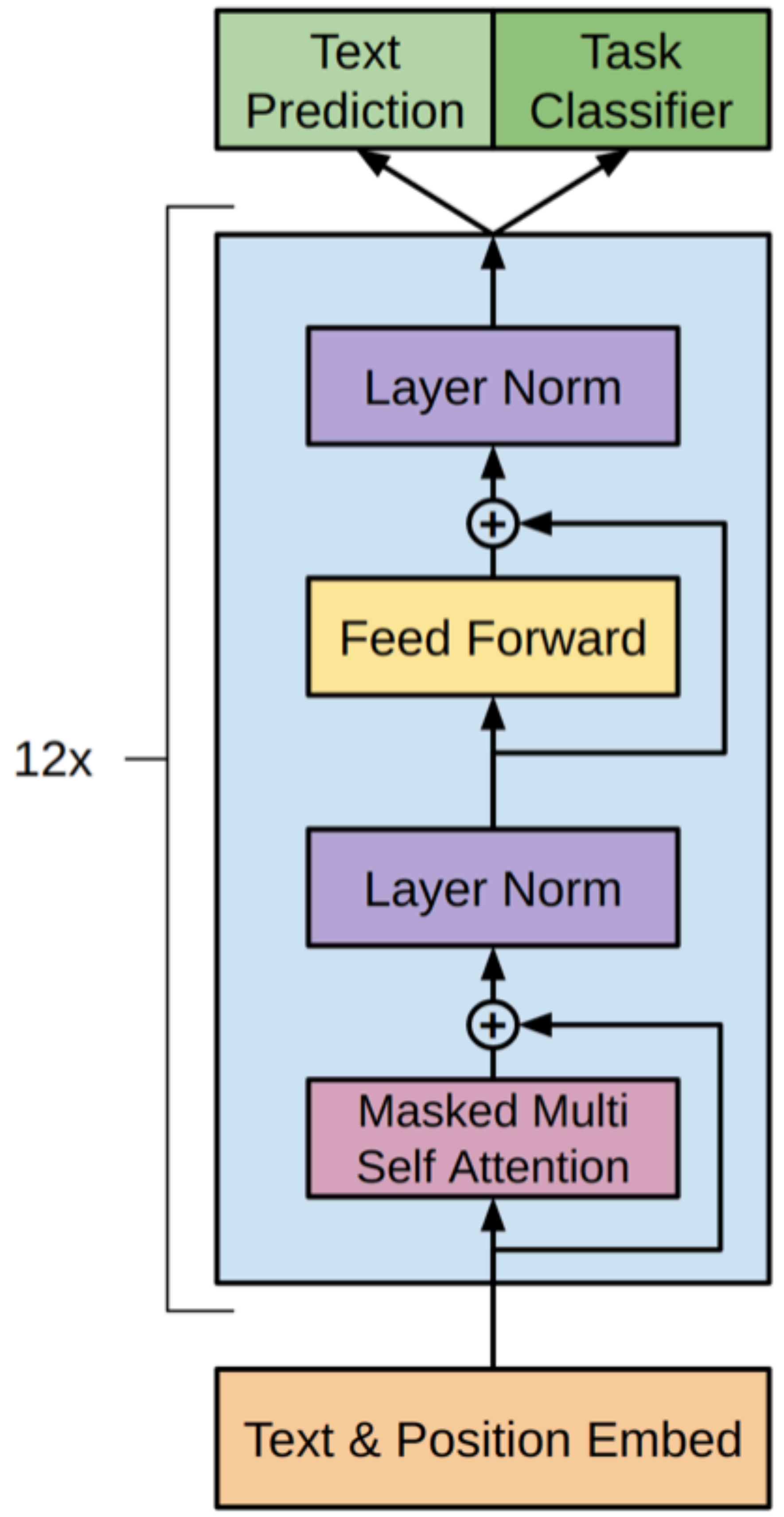
Radford et al. Improving Language Understanding by Generative Pre-Training. 2018

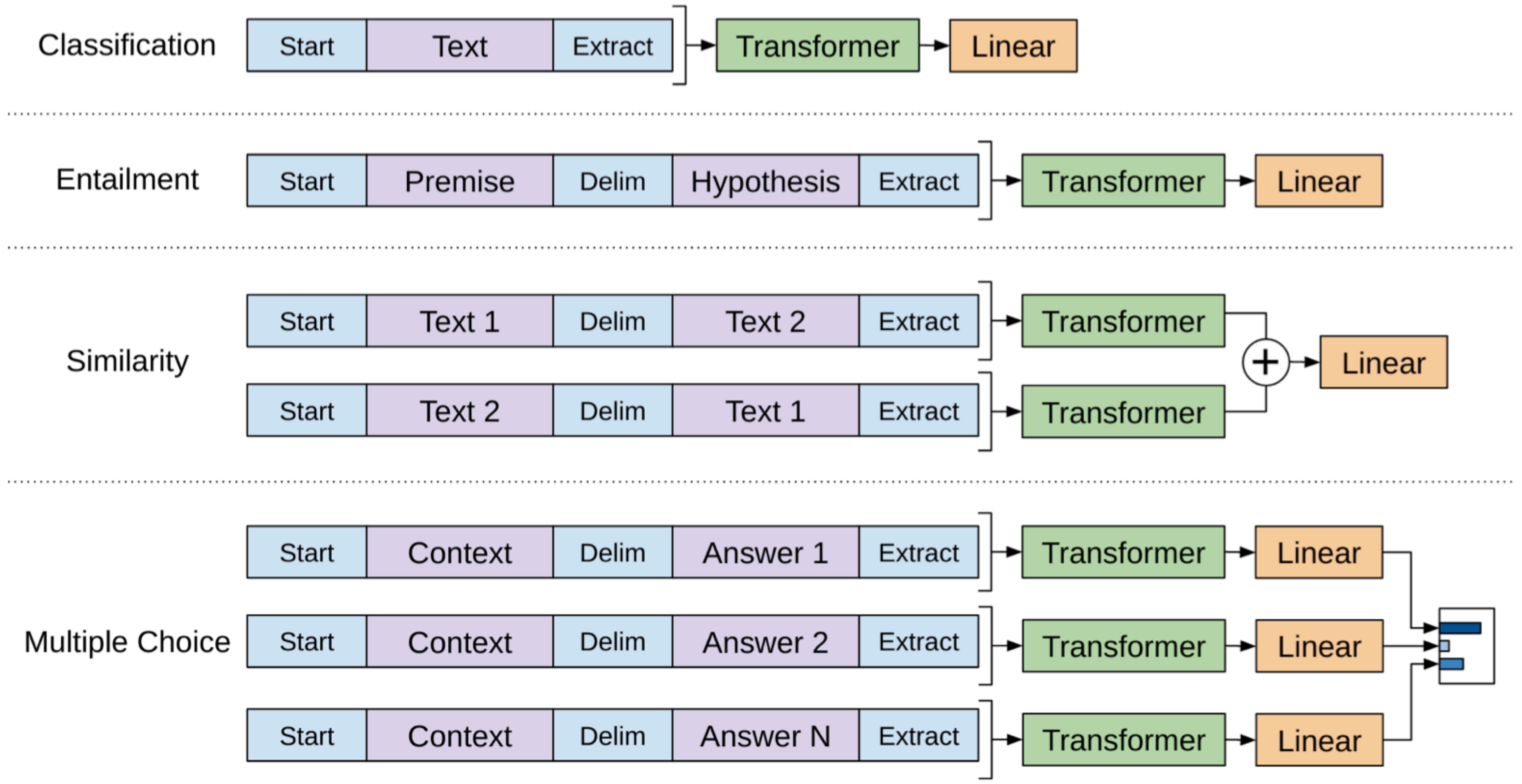
Generative Pre-Training (GPT)



Radford et al. Improving Language Understanding by Generative Pre-Training. 2018

Generative Pre-Training (GPT)





Radford et al. Improving Language Understanding by Generative Pre-Training. 2018

Thank You!