Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 10

Transformers II

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

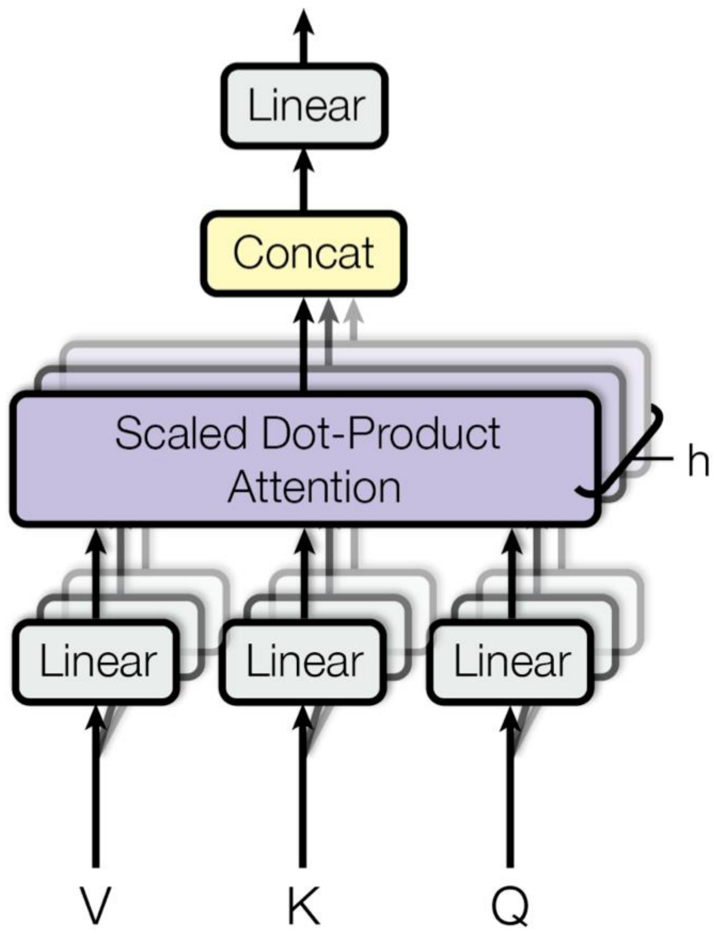
Outline

- Multi-Head Attention
- Encoder Output
- Decoder
- Masked Multi-Head Attention
- Encoder-Decoder Attention
- Output
- Transformer's pipeline

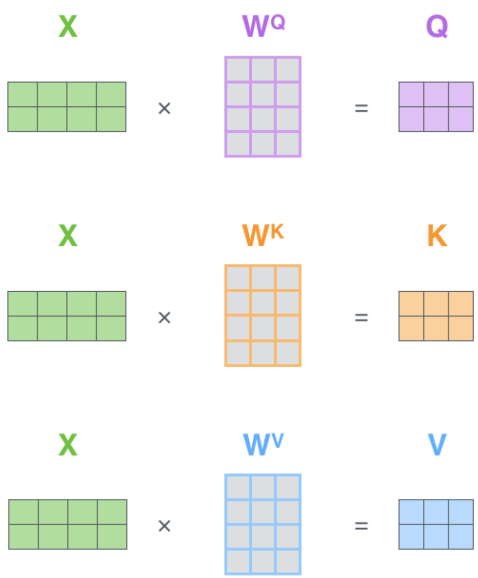
Multi-head attention

Multi-head Attention

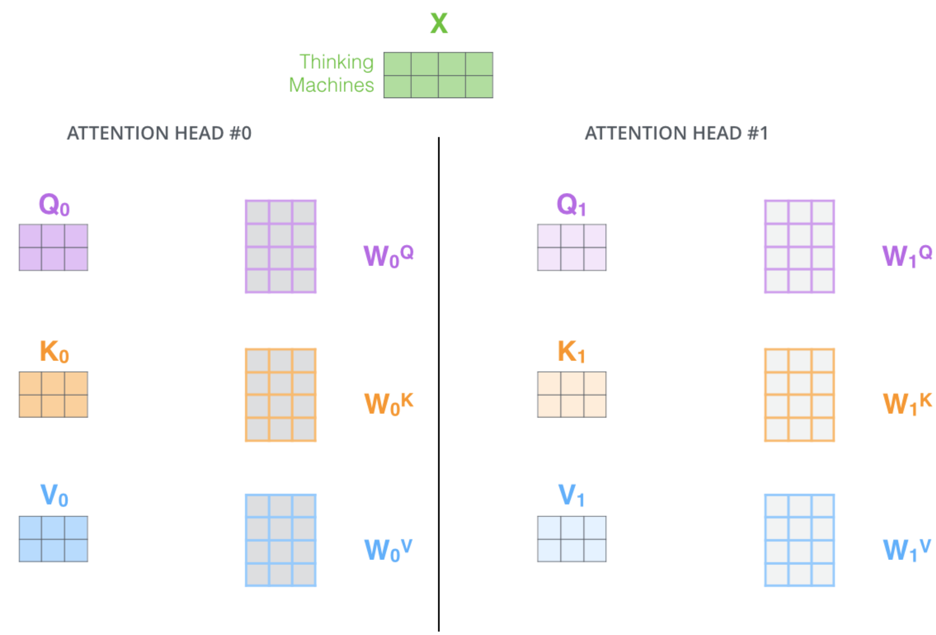
- By using different self attention heads it is possible to encode different meanings of the context:
  - Several scaled-dot product attention computations are performed in parallel (using different weight matrices)
  - The results are concatenated row-by-row forming a larger matrix (with the same number of rows m)
  - This matrix is finally multiplied by a final weight matrix
  - This scheme is called **multi-head** attention



Multi-head Attention



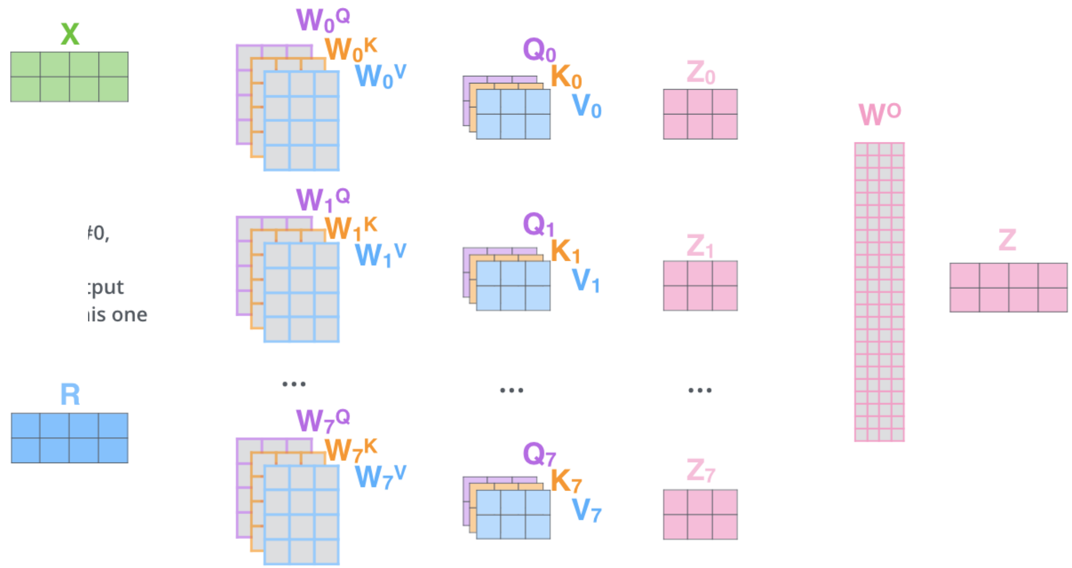
one head



two heads

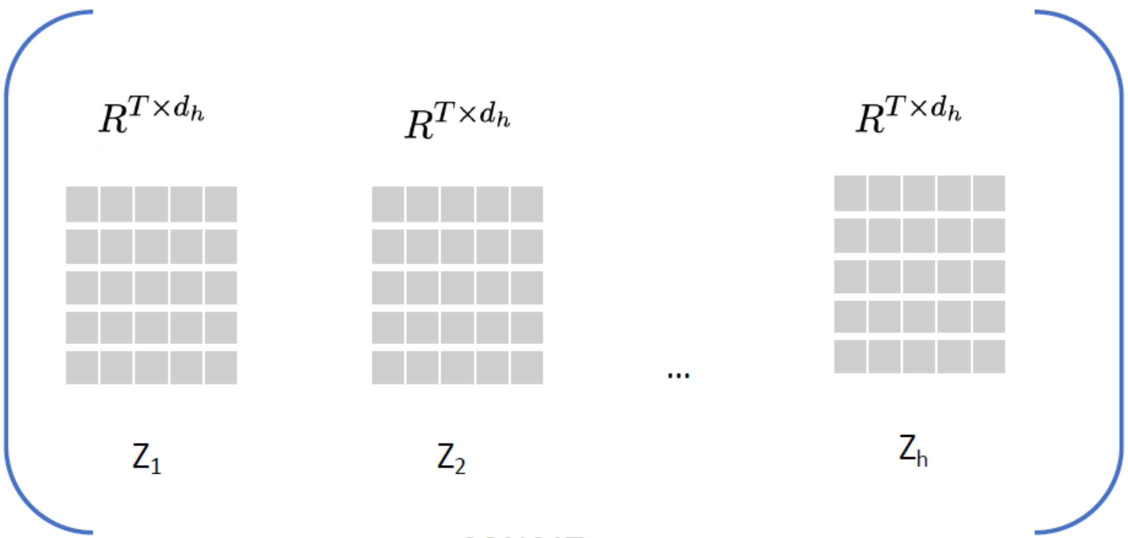
Multi-head Attention

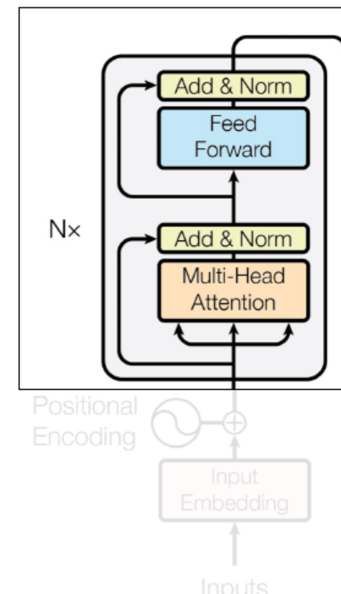
- The outputs of the heads are concatenated and then are multiplied by an additional weight matrix to combine several representations at the same network level.



Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this possibility.

Multi-head Attention





CONCAT

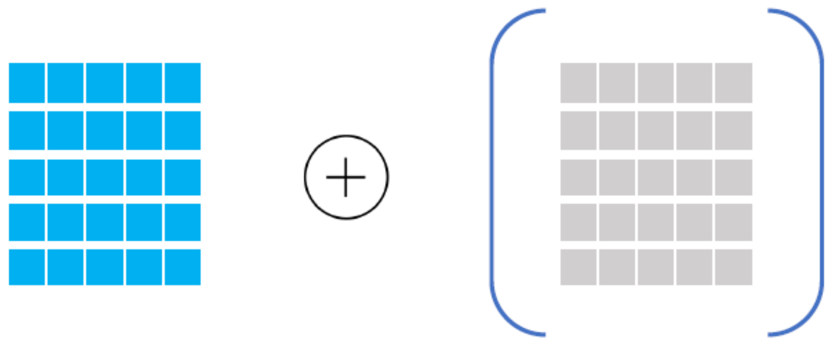
Multi Head Attention : Z

$$d_h = \frac{d_{model}}{h}$$

$$R^{T\times d_{model}}$$

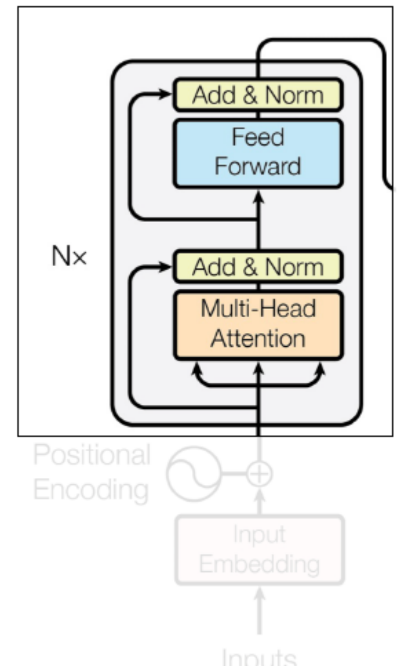
Add (skip connections) & Norm

Add & Norm



Input

Norm(Z)



Normalization(Z)

- Mean 0, Std dev 1
- Stabilizes training
- Regularization effect

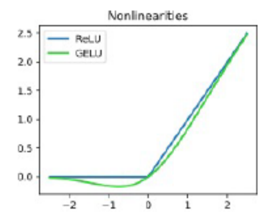
Add -> Residuals

- Avoid vanishing gradients
- Train deeper networks

Feed Forward

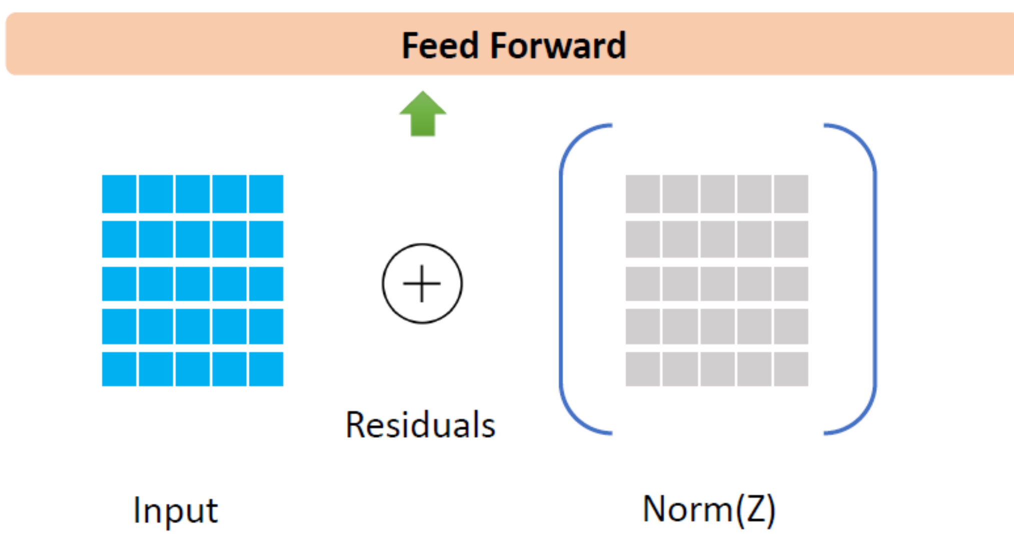
Feed Forward

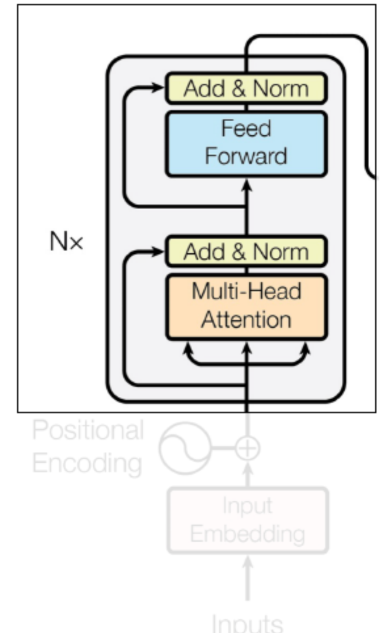
Add & Norm

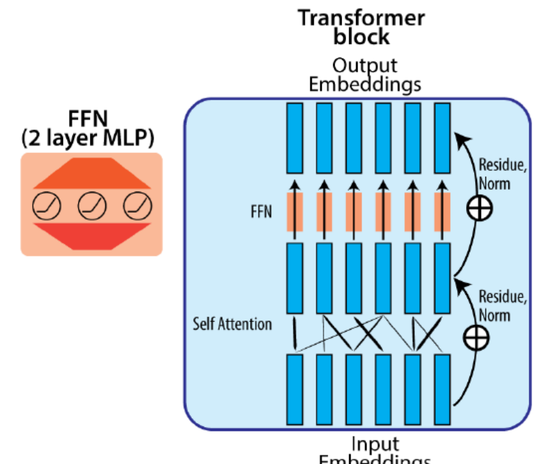


Feed Forward

- Non Linearity
- Complex Relationships
- Learn from each other



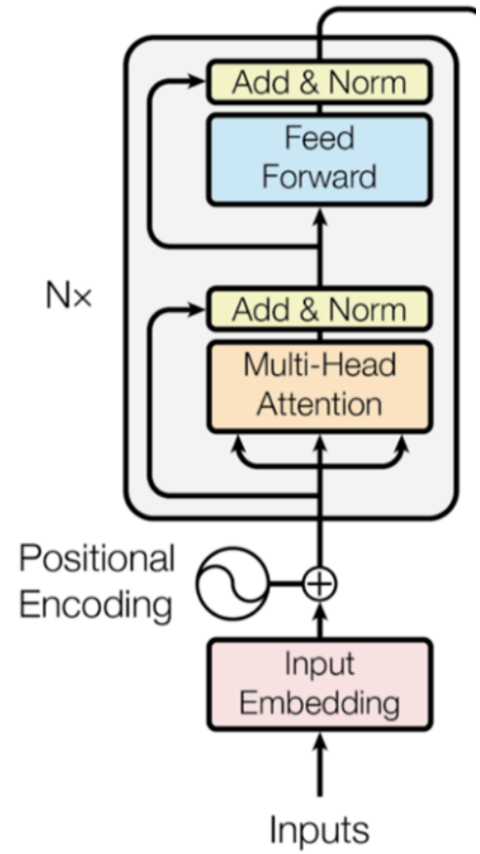




Transformer’s Encoder

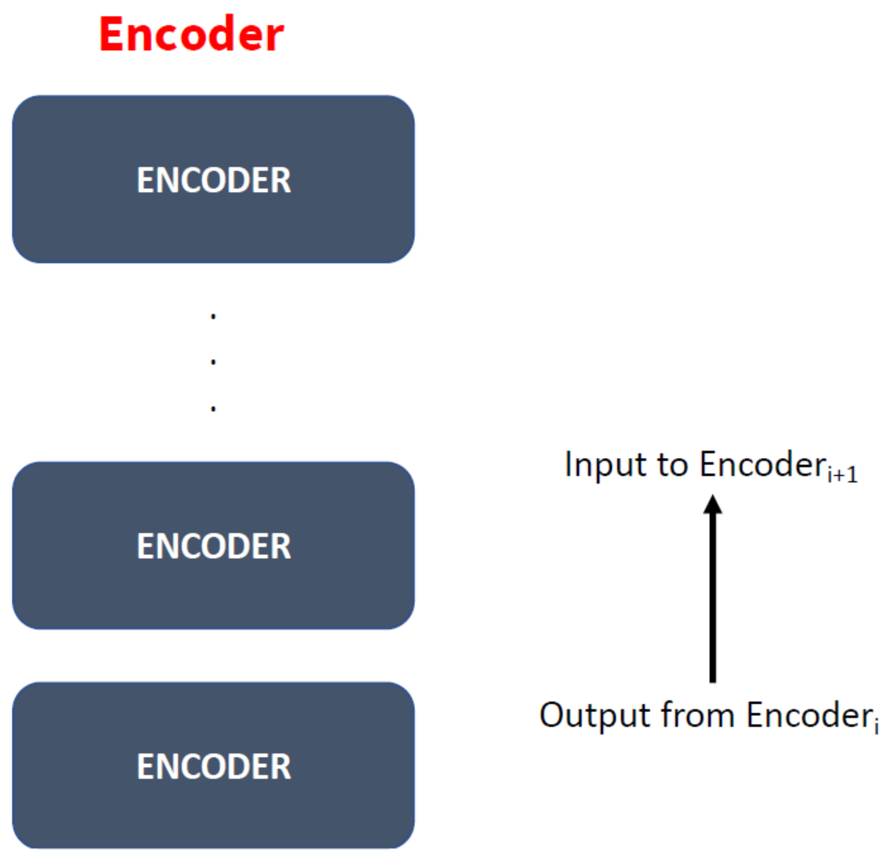
Transformer's Encoder

- Used for computing a representation of the input sequence
- Uses an additive positional encoding to deal with the order-agnostic nature of the self-attention
- Uses residual connection to foster the gradients flow
- Adopts normalization layers to stabilize the network training
- Position-Wise Feed-Forward layer to add non-linearity
  - Applied to each sequence element independently



Transformer's Encoder

- Since each encoder produces an output whose dimensionality is the same of the input, it is possible to stack an arbitrary number of encoder's blocks.
- The output of the first block is fed to the second block (no word embeddings) and so on.

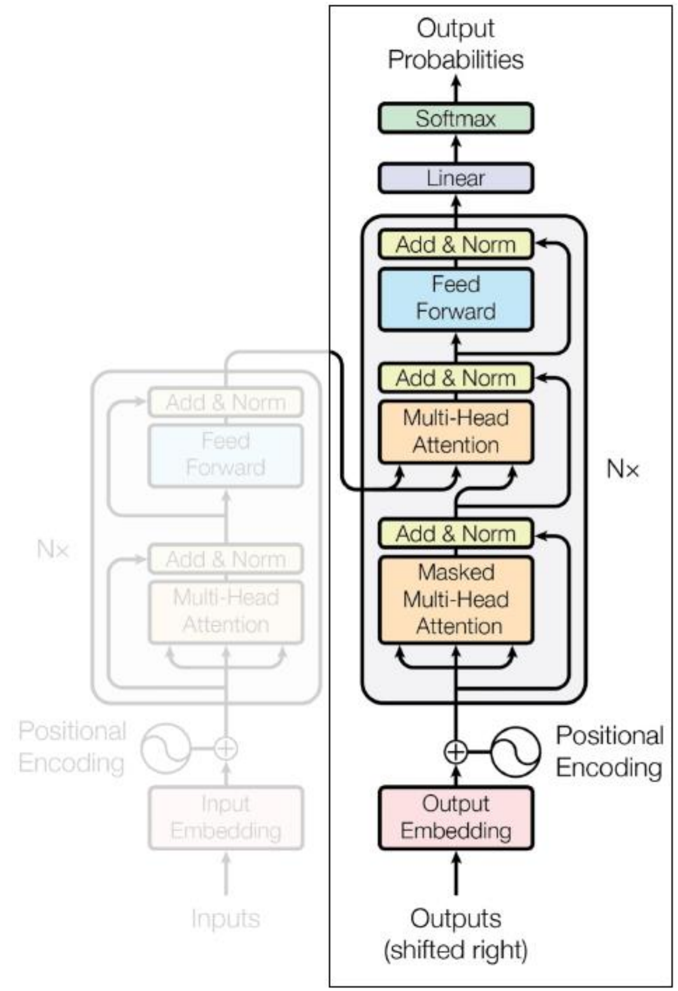


Decoder

Decoder

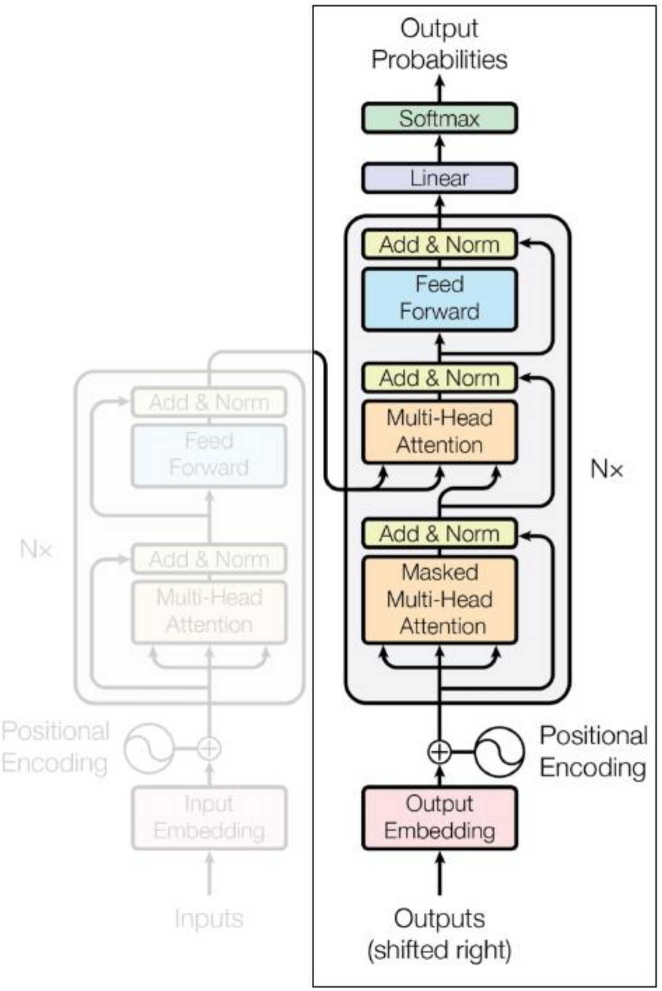
- The **Decoder** uses the information contained in the intermediate representation $z_1, \dots, z_t$ to generate the output sequence $y_1, \dots, y_m$

- The Decoder works **sequentially**; at each step the decoder uses $z_1, \dots, z_t$ and $y_{1,\dots,}y_{i-1}$ to generate y_i

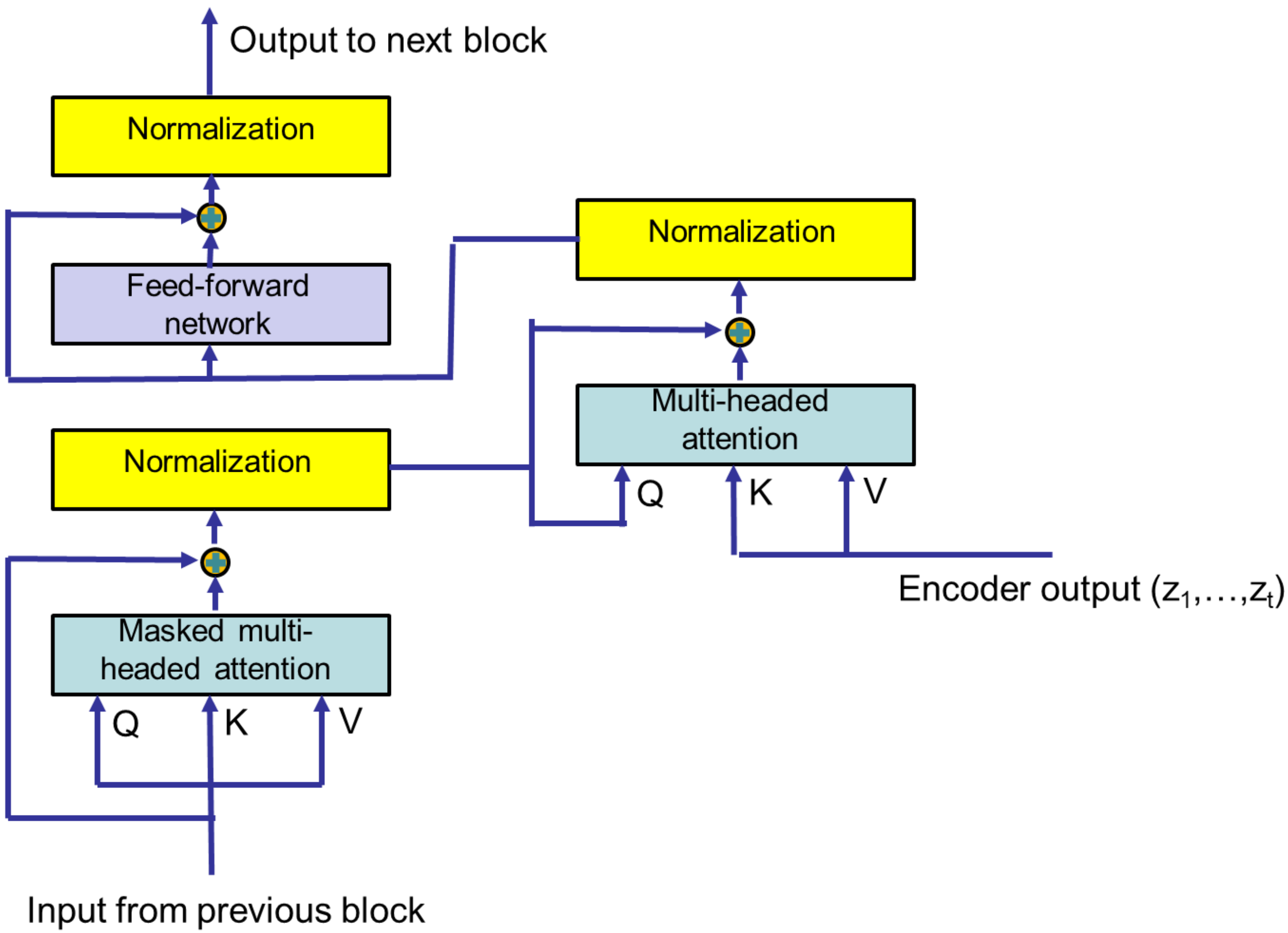


Decoder

- The decoder is made of a sequence of **decoder blocks** having the same structure
  - The original paper used 6 decoder blocks
- The decoder blocks, in addition to the same modules used in the encoder block, add an attention module where the keys and values are taken from the encoder's intermediate representation $z_1, \dots, z_t$
  - Also, the self-attention module is slightly modified so as to ensure that the query at position i only uses the values at positions 1,…,i

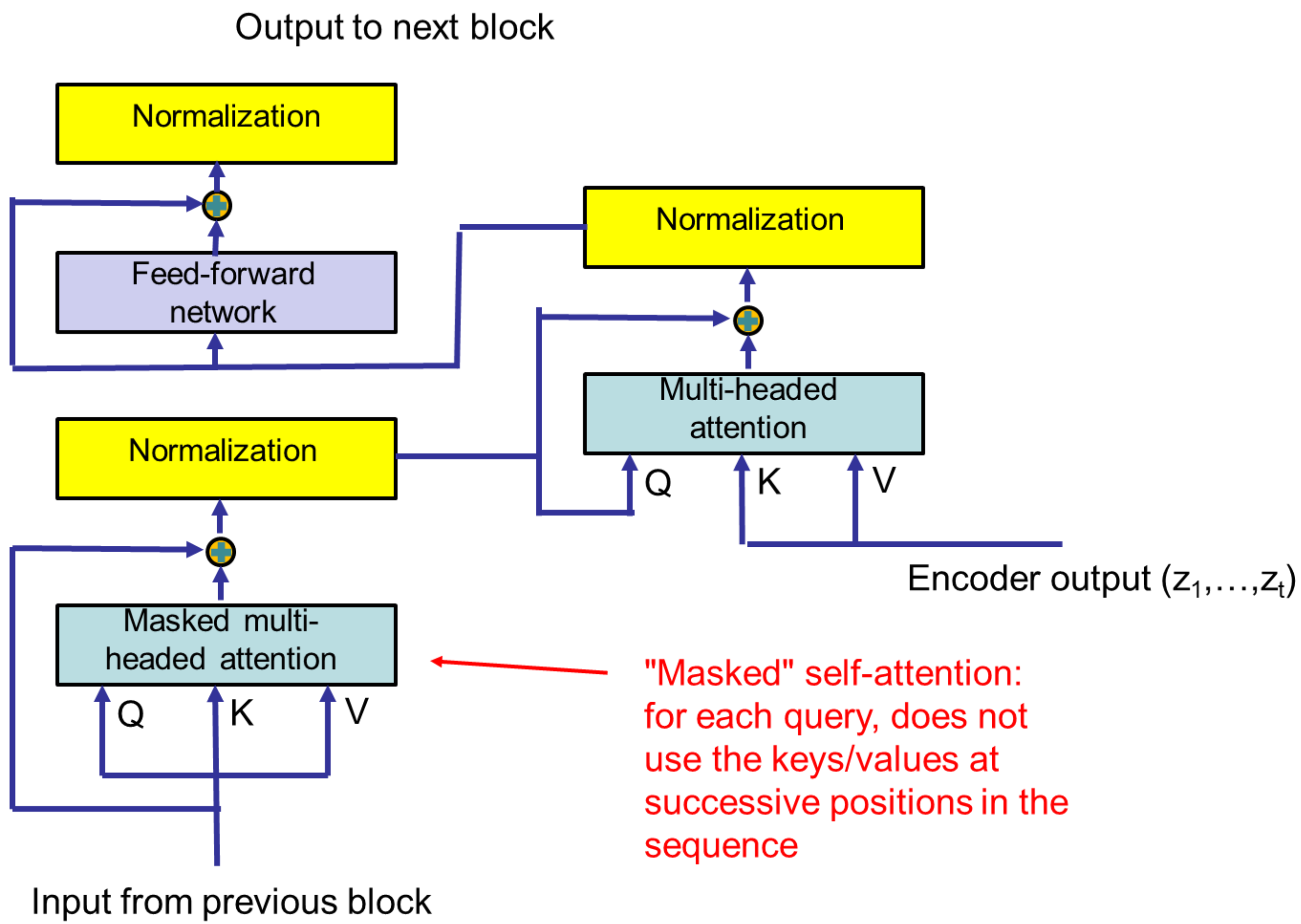


Decoder

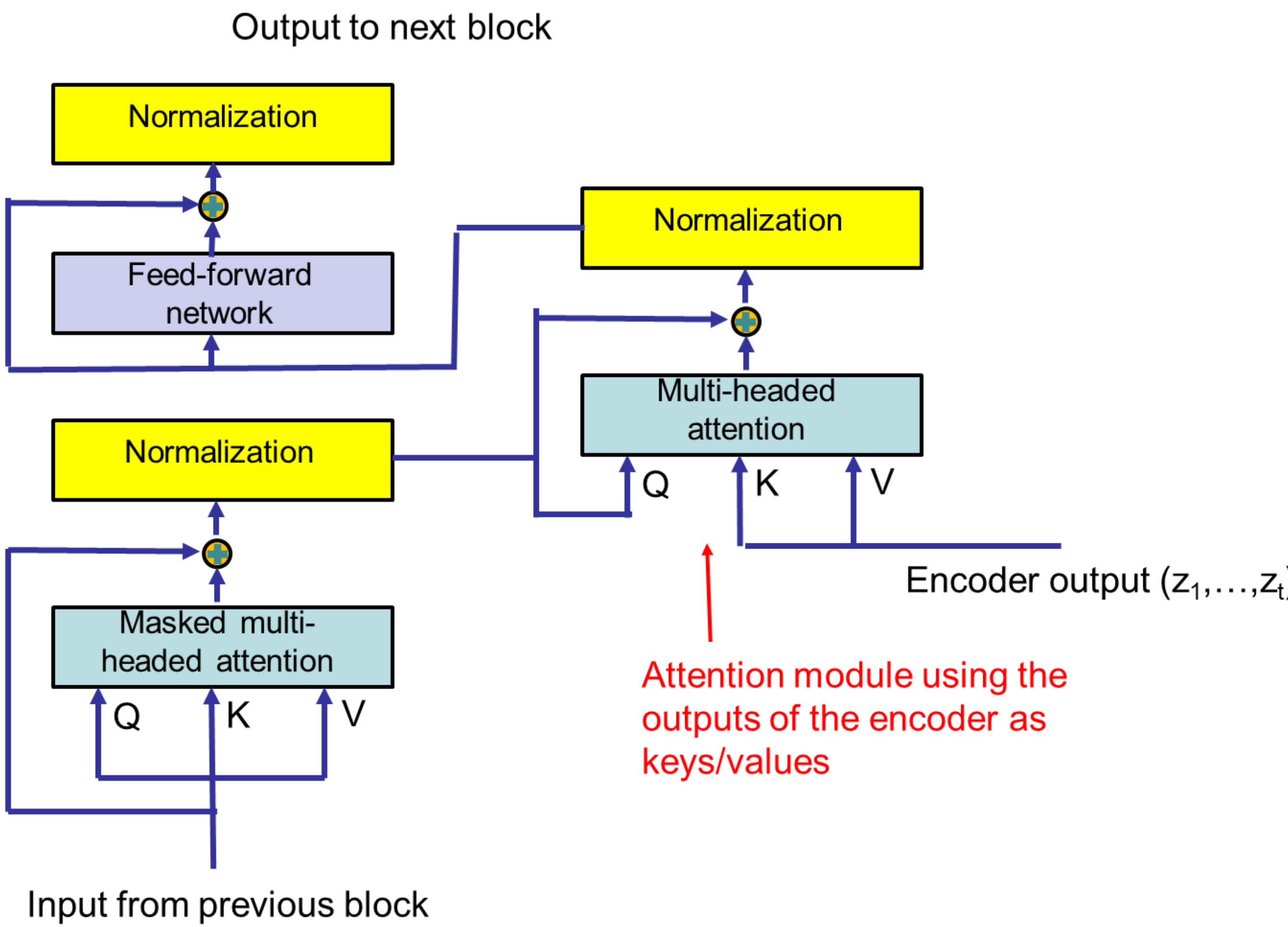


Output to next block
Normalization
Feed-forward network
Normalization
Multi-headed attention
Q
K
V
Normalization
Masked multi-headed attention
Q
K
V
Encoder output ($z_1,\ldots,z_t$)
Input from previous block

Decoder

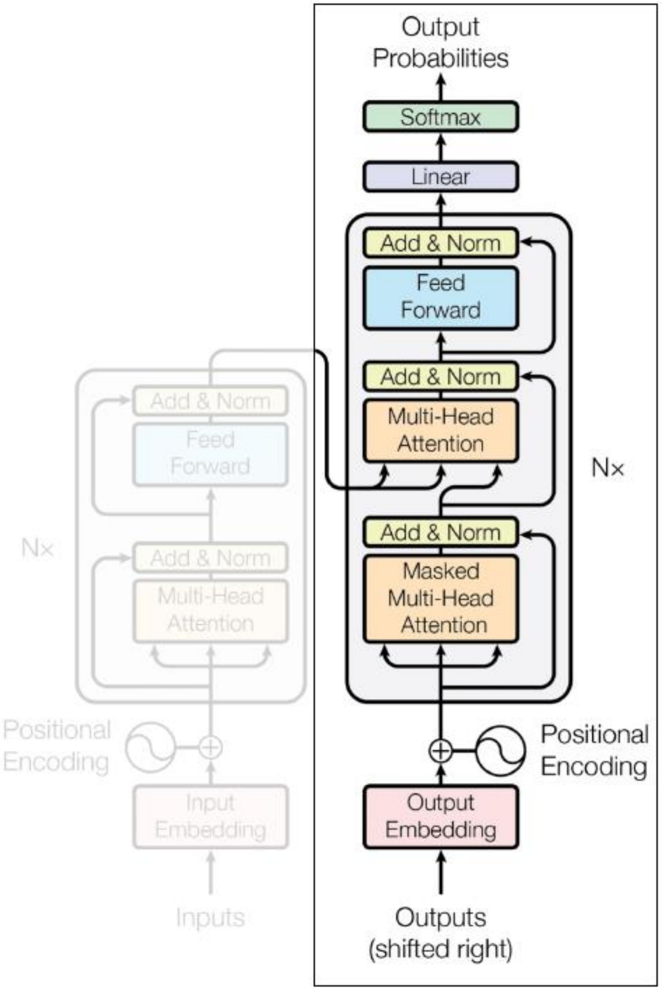

Output to next block
Normalization
Feed-forward network
Normalization
Multi-headed attention
Q
K
V
Normalization
Masked multi-headed attention
Q
K
V
Encoder output ($z_1,\ldots,z_t$)
Input from previous block
"Masked" self-attention: for each query, does not use the keys/values at successive positions in the sequence

Decoder


Output to next block
Normalization
Feed-forward network
Normalization
Multi-headed attention
Q
K
V
Encoder output (z_1,…,z_t)
Normalization
Masked multi-headed attention
Q
K
V
Attention module using the outputs of the encoder as keys/values
Input from previous block

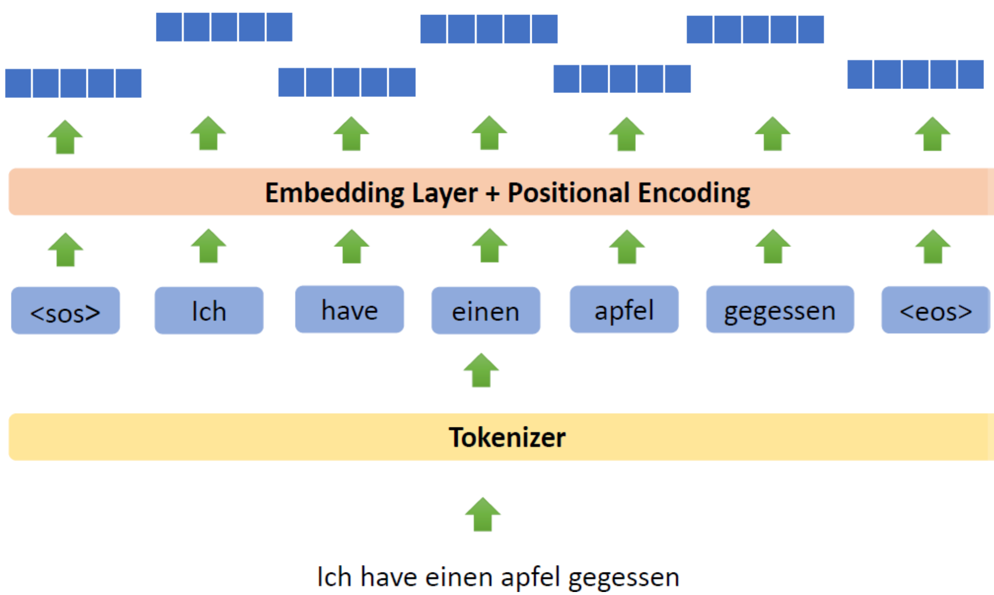
Decoder

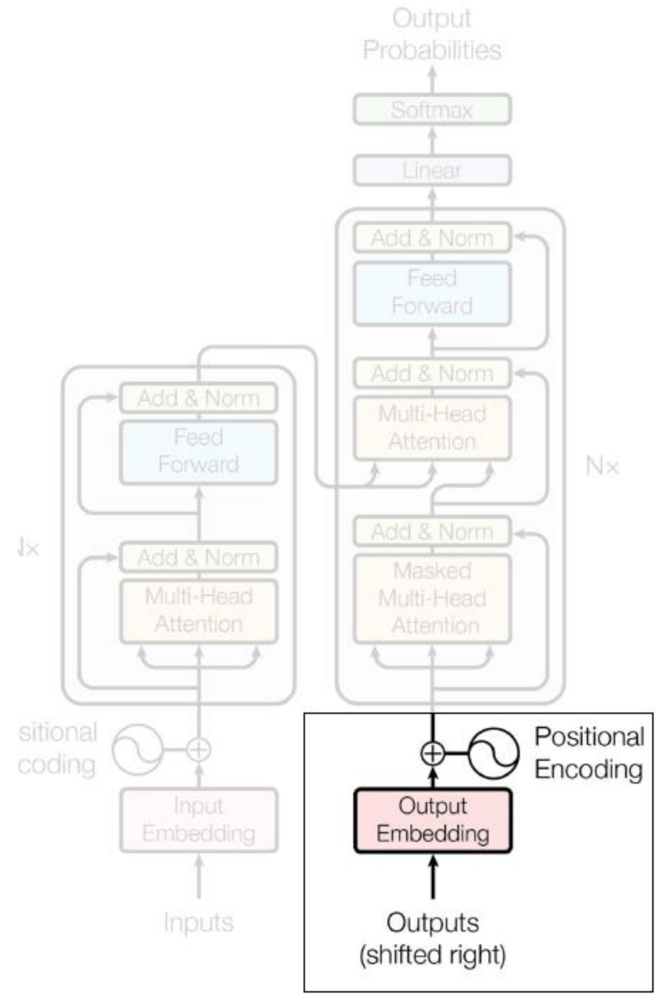
- On top of the last decoder block, the decoder adds an additional linear layer and a softmax activation function, for computing the probability of the next output element y_i
- Thus, the last layers has a number of neurons corresponding to the cardinality of the output set



Masked Multi-Head Attention

Output embedding

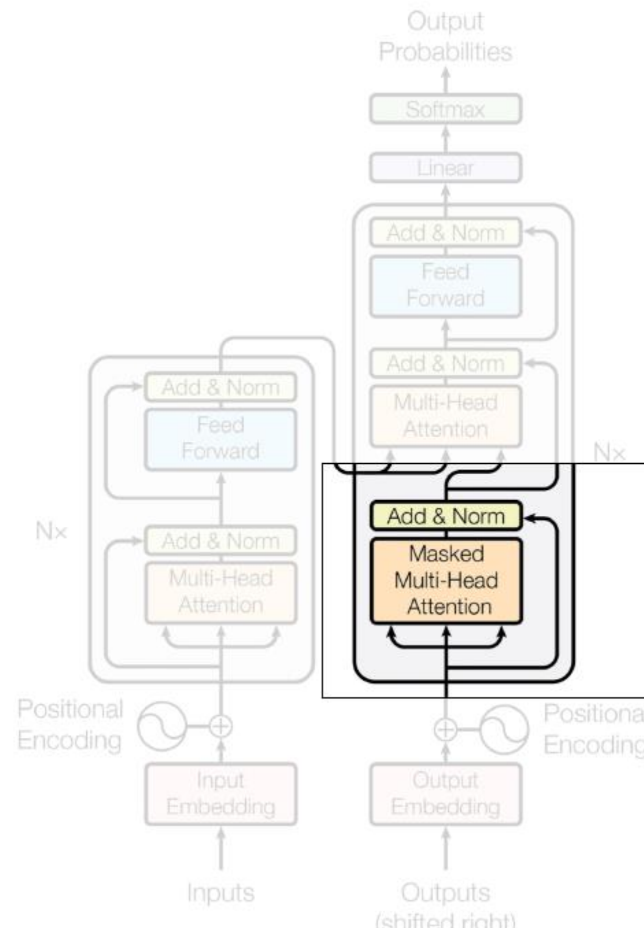




Masked Multi-Head Attention

<sos> Ich have einen apfel gegessen <eos>

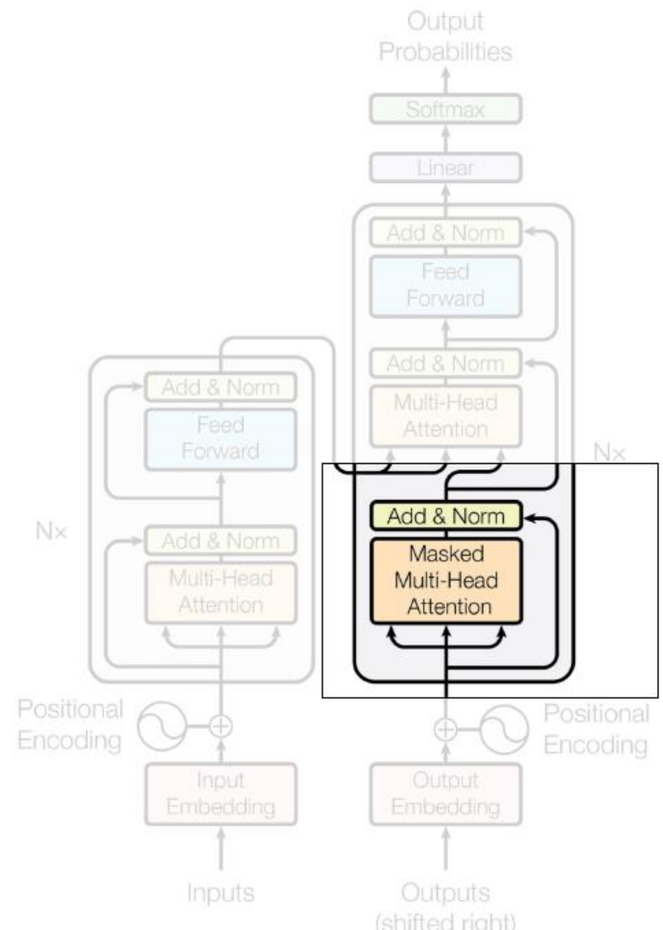
Outputs at time T should only pay attention to outputs until time T-1



Masked Multi-Head Attention

1	<sos>	Ich	have	einen	apfel	gegessen	<eos>
2	<sos>	Ich	have	einen	apfel	gegessen	<eos>
3	<sos>	Ich	have	einen	apfel	gegessen	<eos>
4	<sos>	Ich	have	einen	apfel	gegessen	<eos>
5	<sos>	Ich	have	einen	apfel	gegessen	<eos>
6	<sos>	Ich	have	einen	apfel	gegessen	<eos>
7	<sos>	Ich	have	einen	apfel	gegessen	<eos>

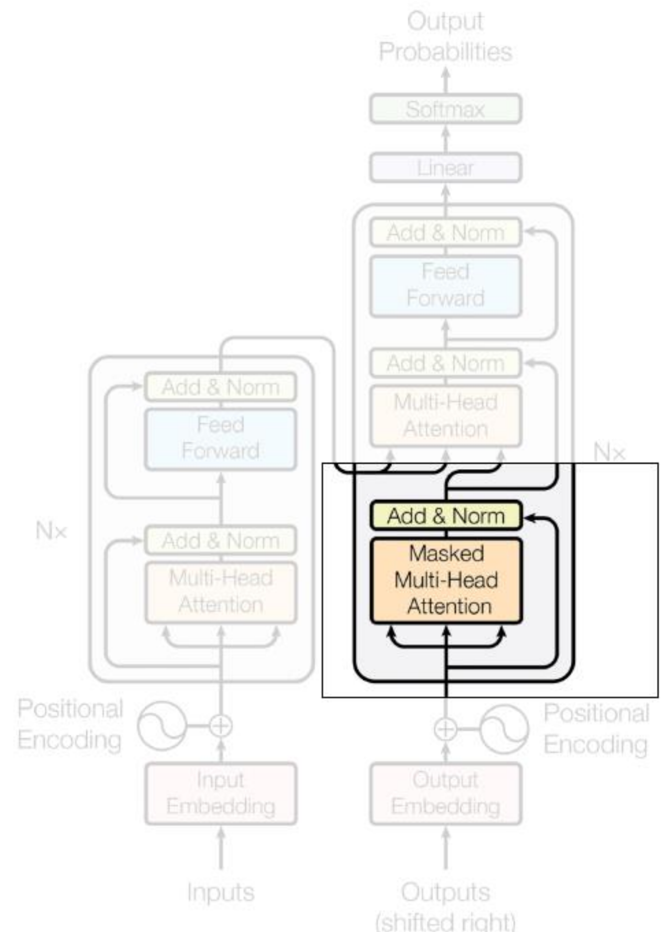
Mask the available attention values ?



Masked Multi-Head Attention

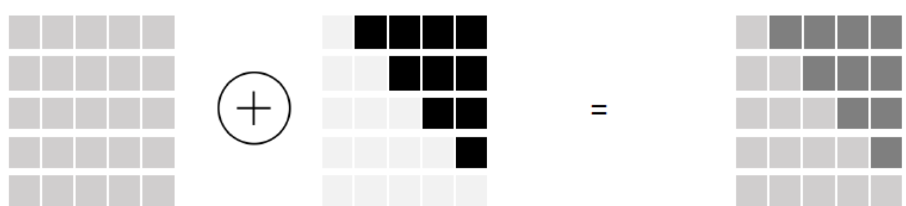
1	<sos>	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
2	<sos>	Ich	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
3	<sos>	Ich	have	$-\infty$	$-\infty$	$-\infty$	$-\infty$
4	<sos>	Ich	have	einen	$-\infty$	$-\infty$	$-\infty$
5	<sos>	Ich	have	einen	apfel	$-\infty$	$-\infty$
6	<sos>	Ich	have	einen	apfel	gegessen	$-\infty$
7	<sos>	Ich	have	einen	apfel	gegessen	<eos>

Softmax -> $-\infty$ -> 0



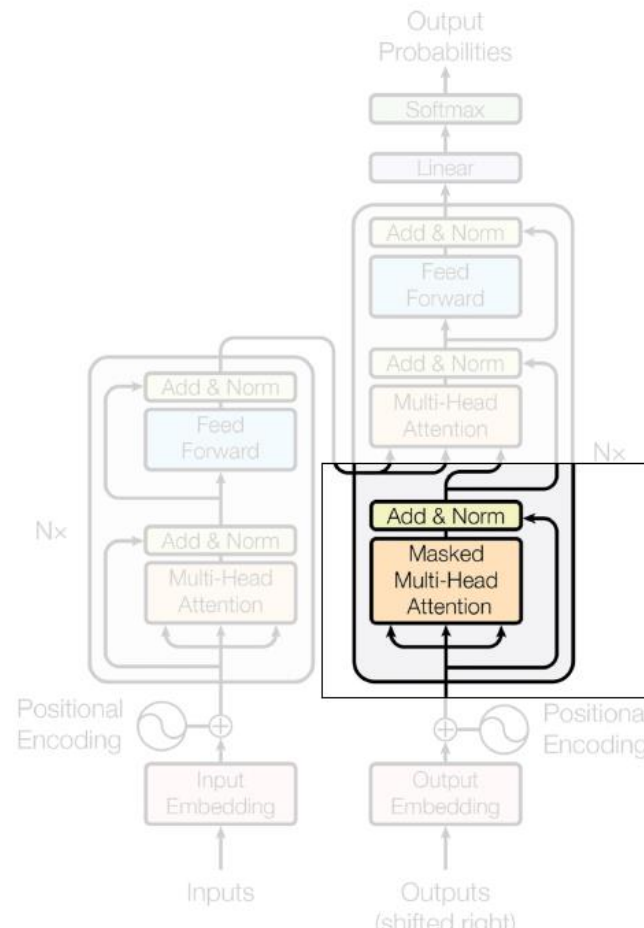
Masked Multi-Head Attention

$R^{T\times T}$ $R^{T\times T}$

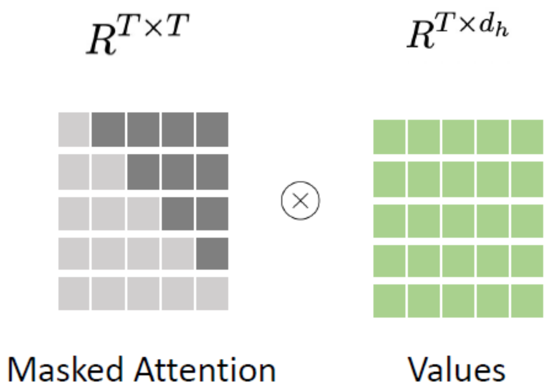


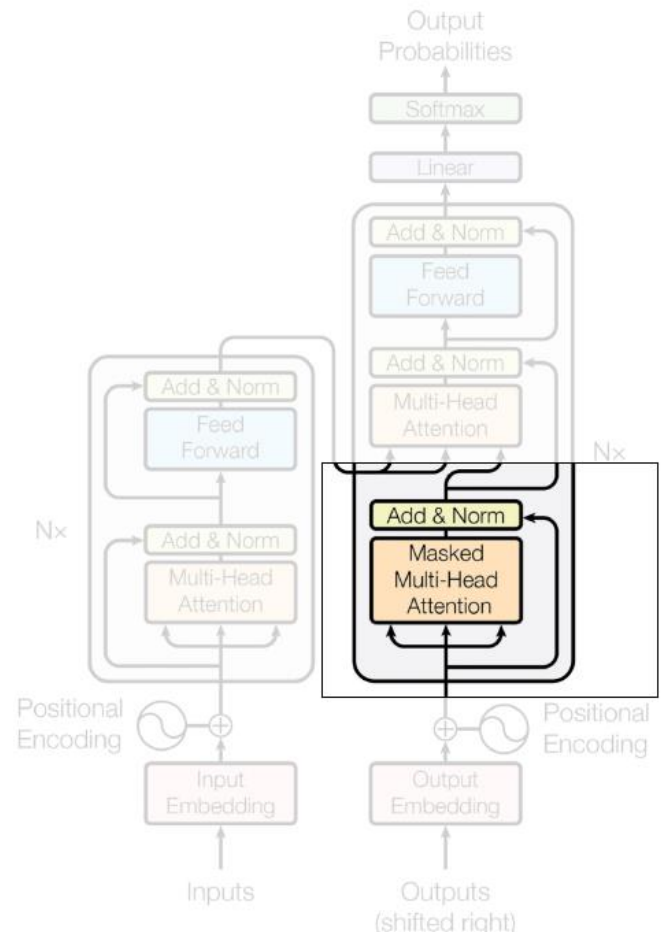
QK^T Attention Mask: M Masked Attention

Masked Multi Head Attention : Z’



Masked Multi-Head Attention





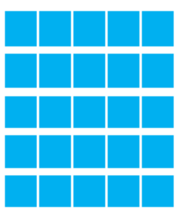
Masked Multi Head Attention : Z’

Encoder Decoder Attention

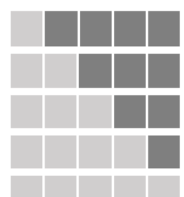
Encoder Decoder Attention

Encoder Decoder Attention ?

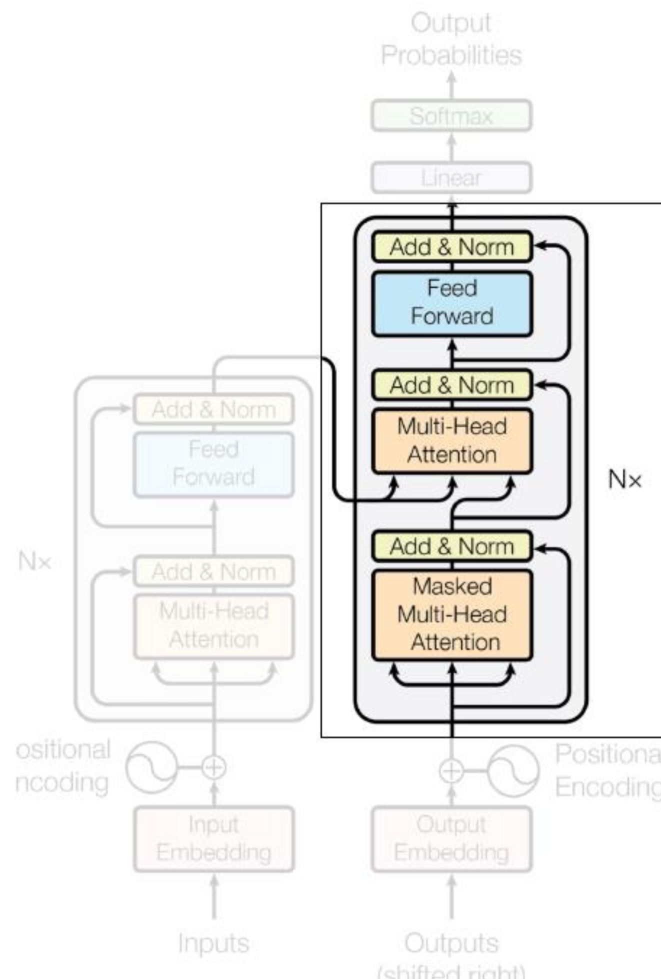
Add & Norm



Input



Norm(Z')



Encoder Decoder Attention

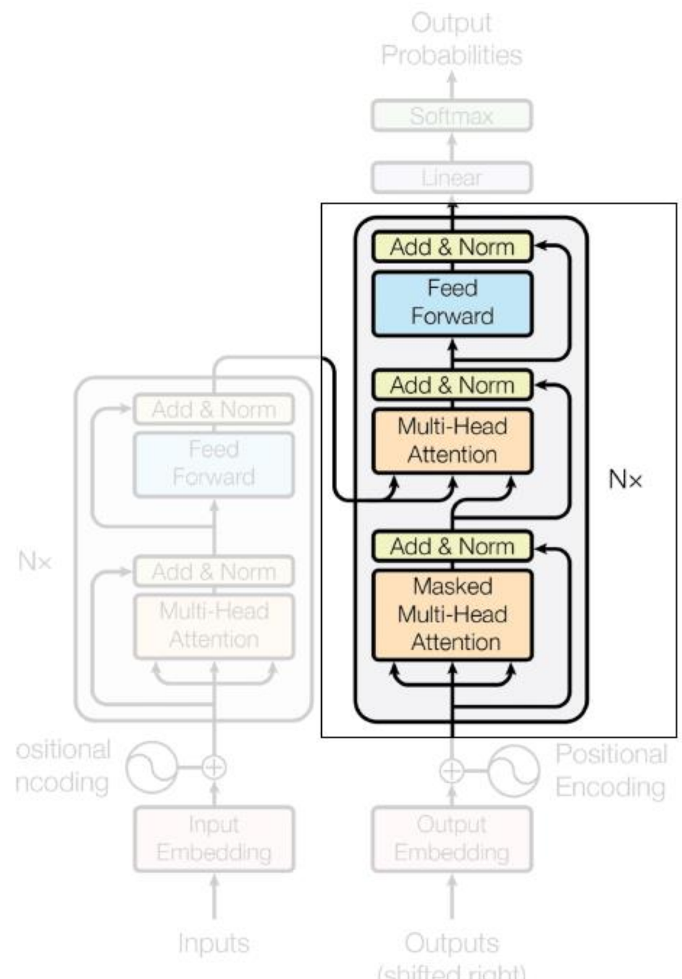
Encoder **Self** Attention

1. Queries from Encoder Inputs
2. Keys from Encoder Inputs
3. Values from Encoder Inputs

Decoder **Masked Self** Attention

1. Queries from Decoder Inputs
2. Keys from Decoder Inputs
3. Values from Decoder Inputs

Encoder Decoder Attention ?



Encoder Decoder Attention

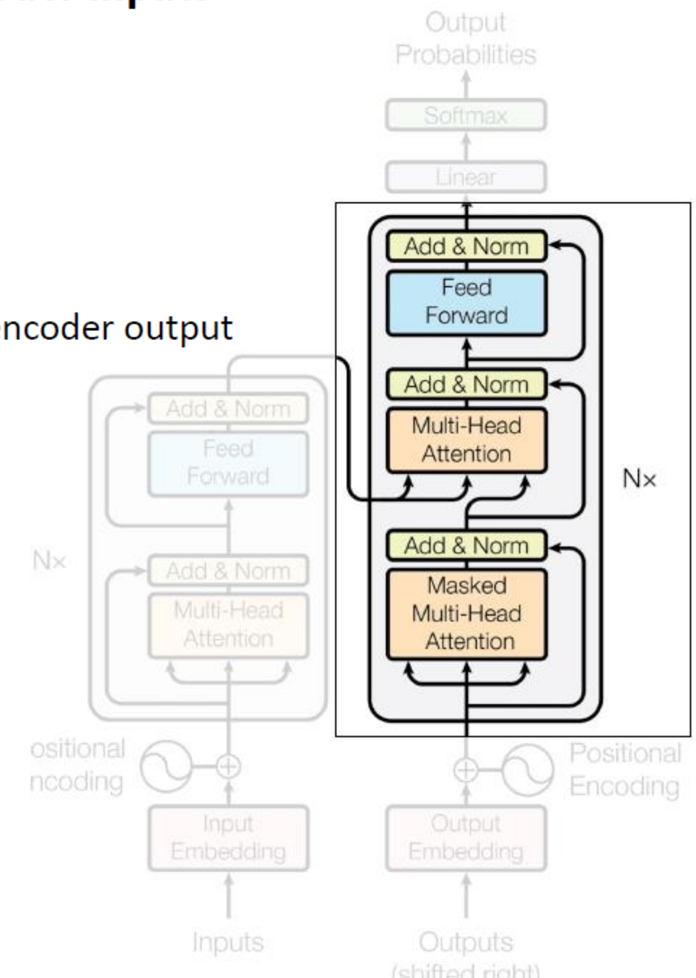
Encoder

Keys from **Encoder Outputs**
Values from **Encoder Outputs**

Decoder

Queries from **Decoder Inputs**

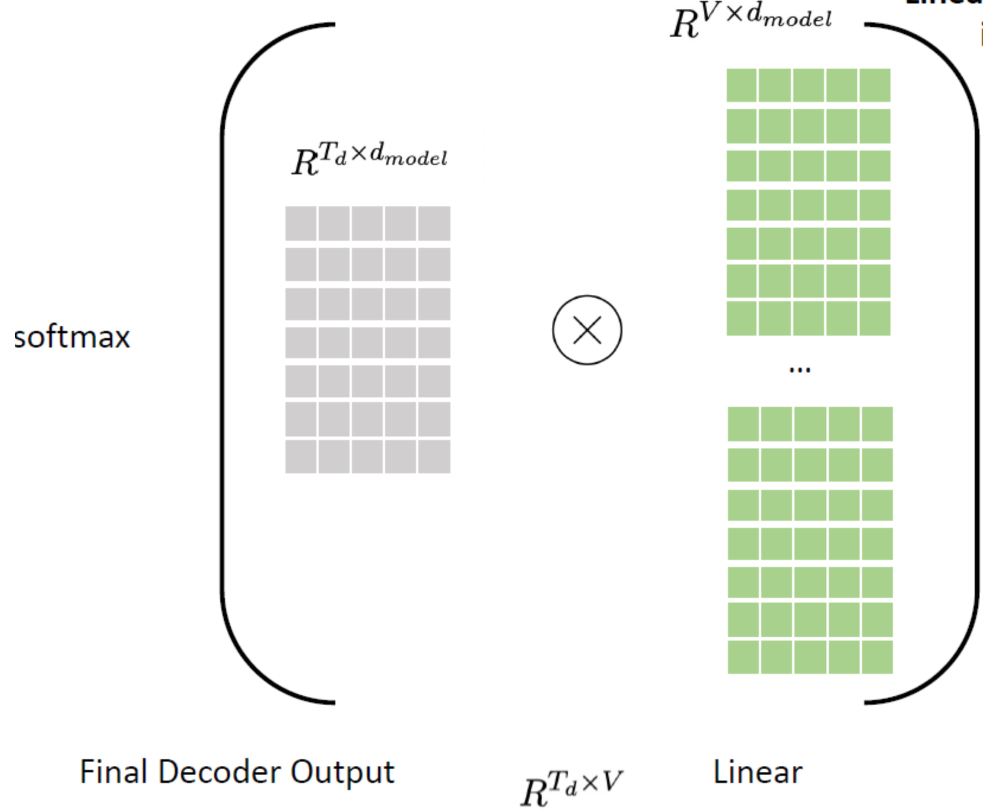
NOTE: Every decoder block receives the same FINAL encoder output

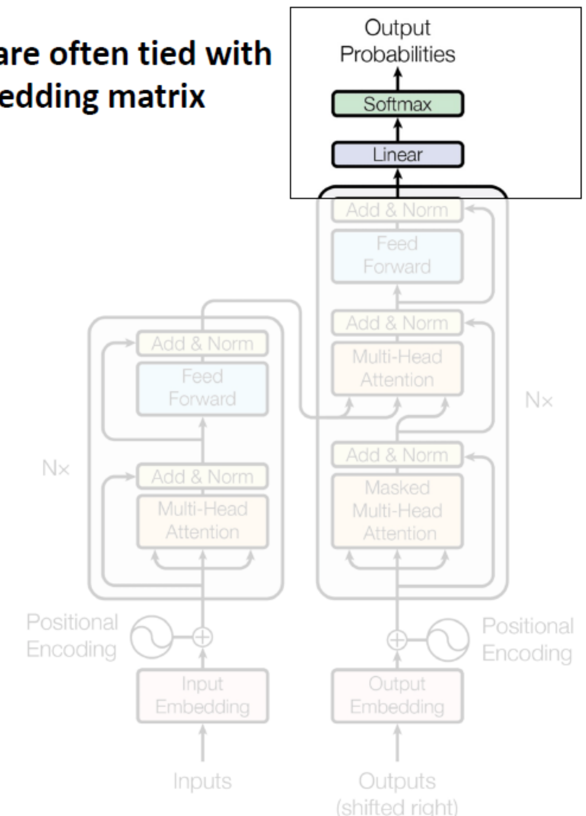


Output

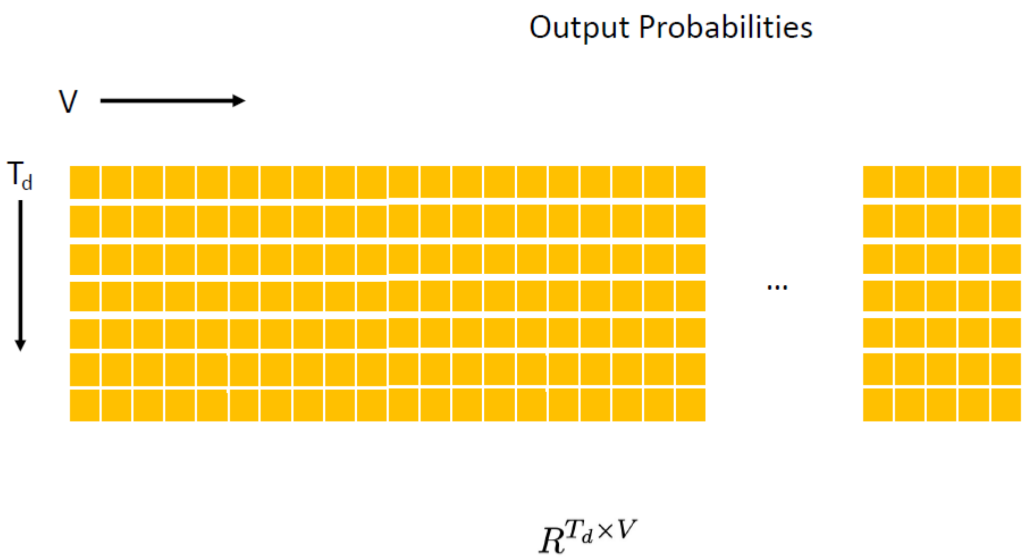
Linear

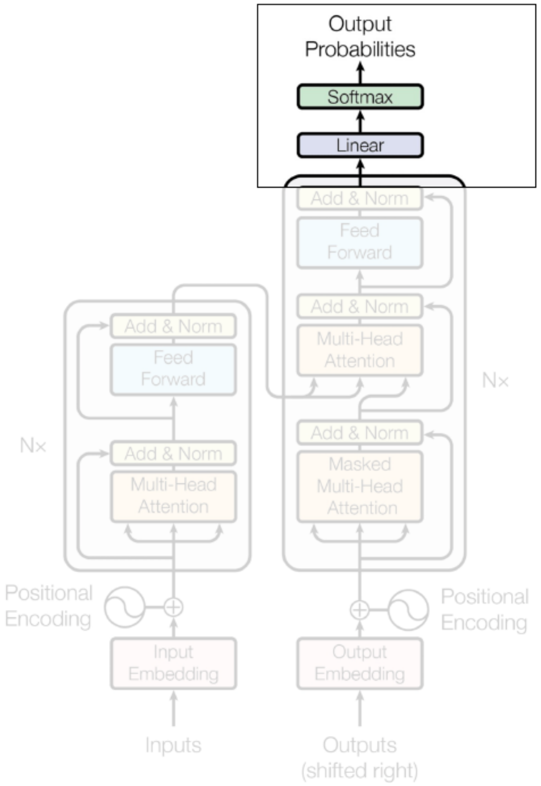
Linear weights are often tied with input embedding matrix





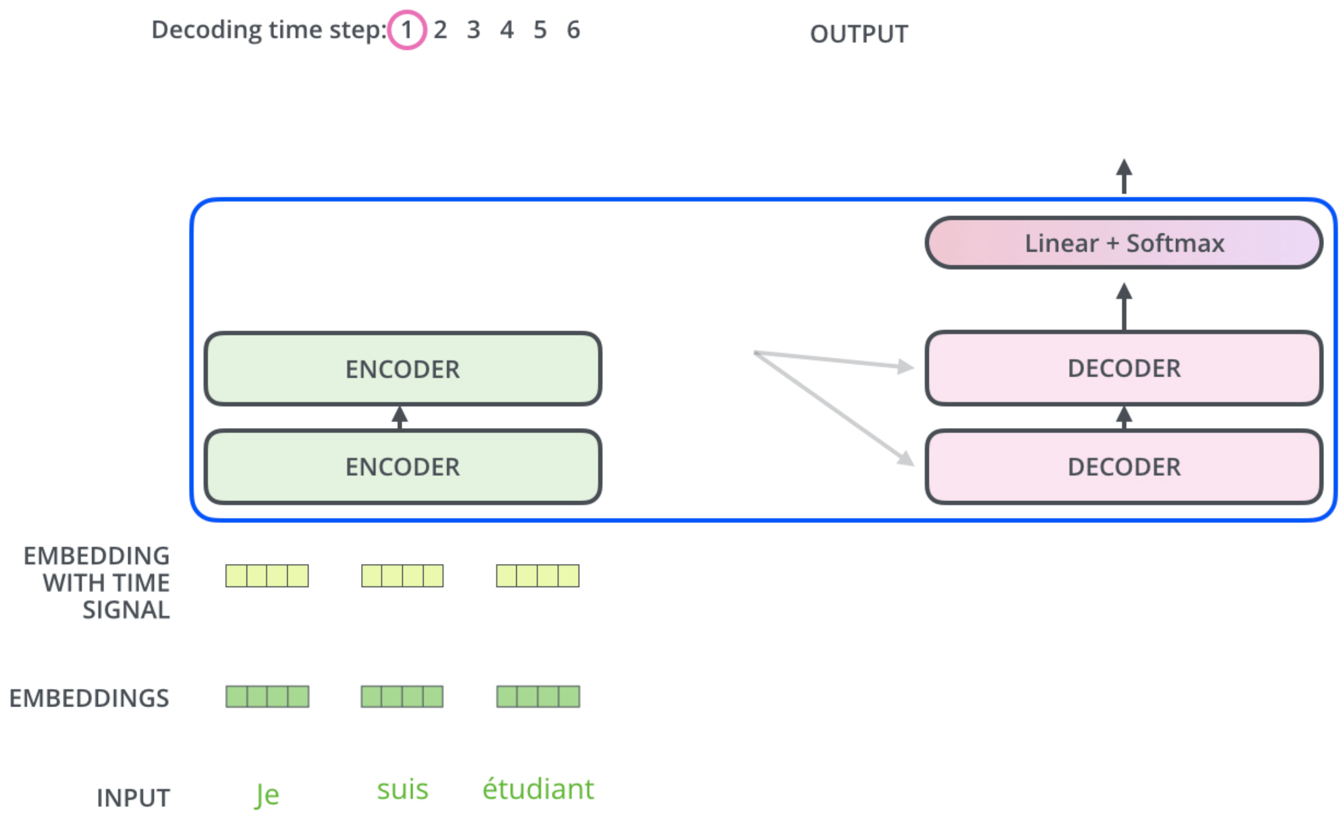
Softmax


Output Probabilities
V
T_d
...
$R^{T_d \times V}$

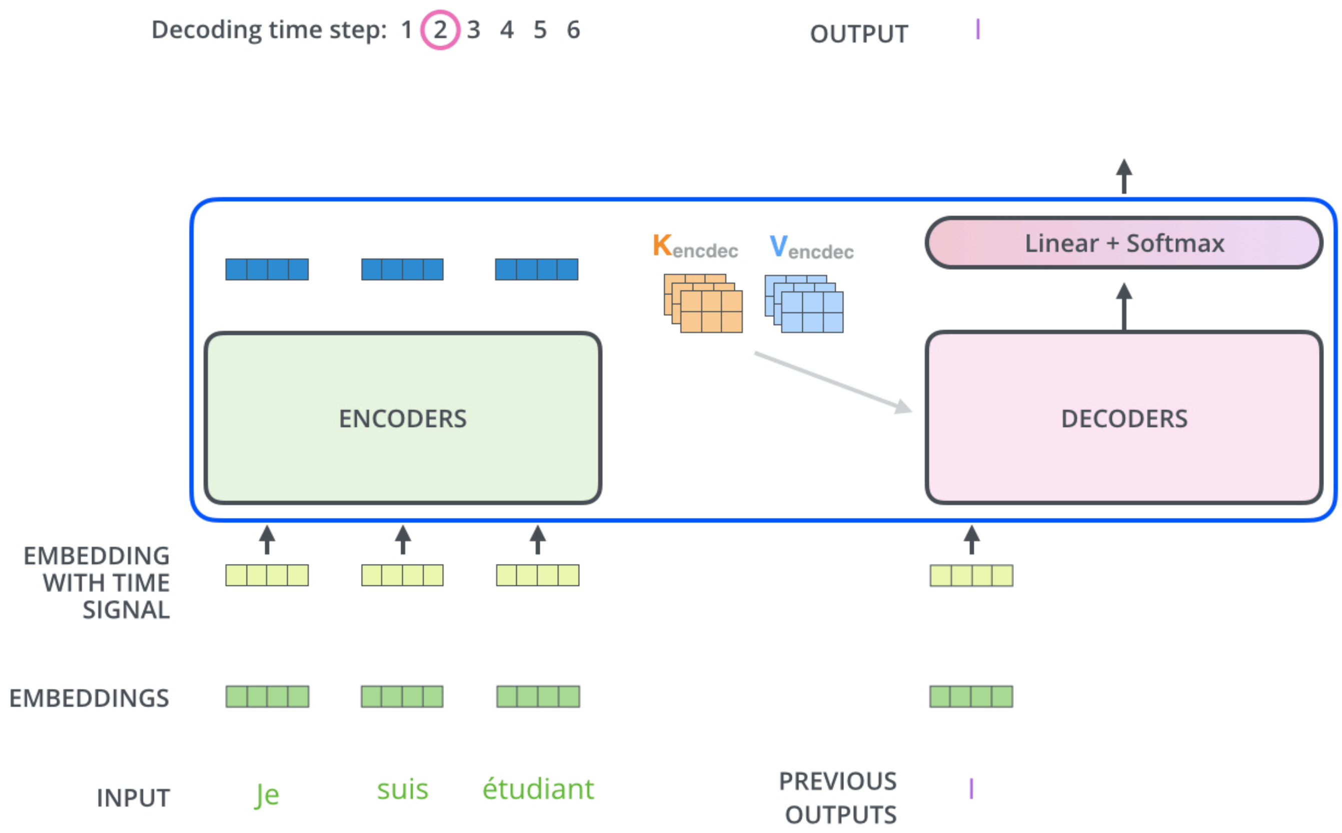

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
N×
N×
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Transformer’s pipeline

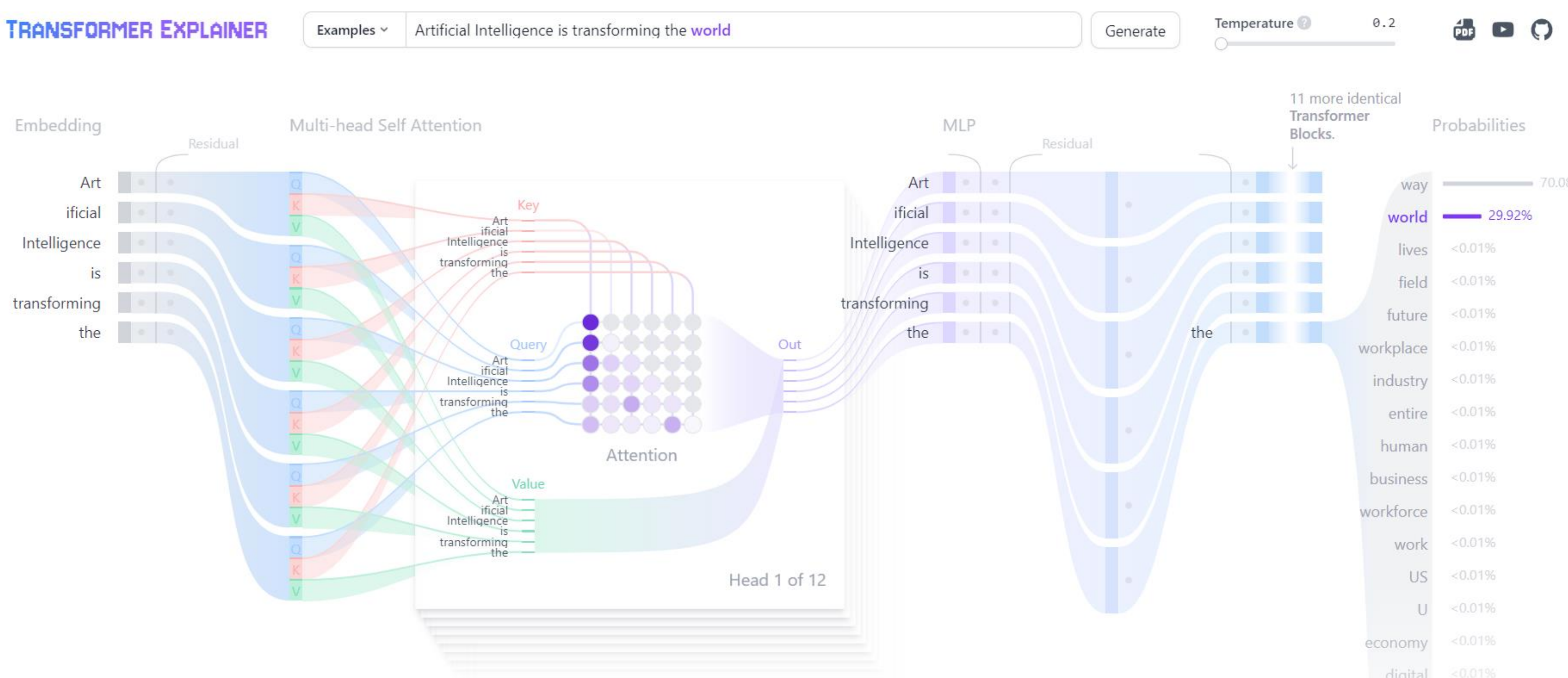
Transformer's pipeline



Transformer's pipeline



Transformer's pipeline



https://poloclub.github.io/transformer-explainer/

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 10

Transformers II

Nicola Capuano and Antonio Greco

DIEM – University of Salerno