10. LLMs in 2025

LING-581-Natural Language Processing 1

Instructor: Hakyung Sung

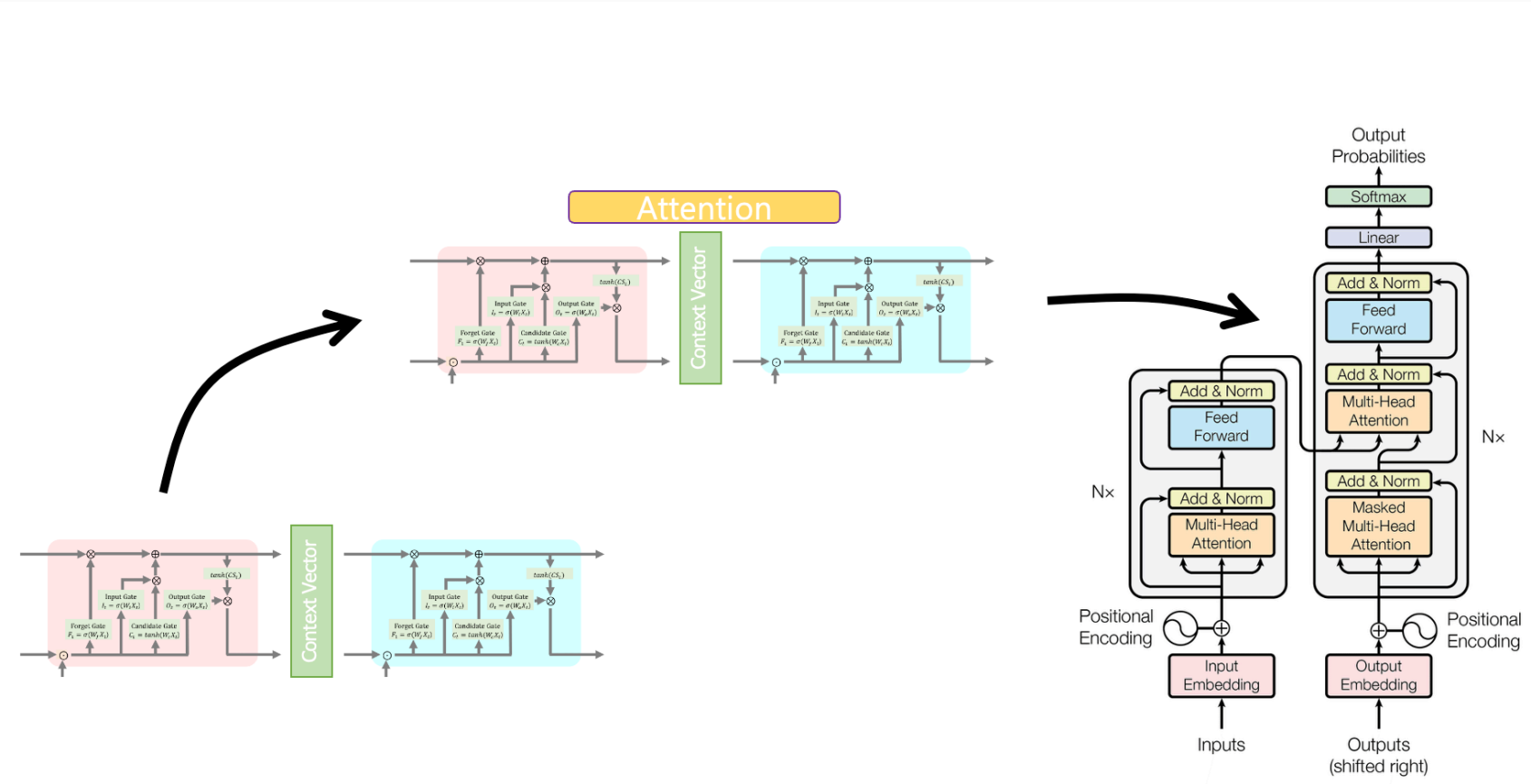
October 28, 2025

Table of contents

Review: Transformer

Recall

Towards the development of the **Transformer** model.



RNNs: Lack of parallelizability

- RNNs process input step by step — each hidden state depends on the previous one.

RNNs: Lack of parallelizability

- RNNs process input step by step — each hidden state depends on the previous one.
- GPUs are great at performing many independent computations in parallel, but RNNs don’t allow this because future states can’t be computed until past states are done.

RNNs: Lack of parallelizability

- RNNs process input step by step — each hidden state depends on the previous one.
- GPUs are great at performing many independent computations in parallel, but RNNs don’t allow this because future states can’t be computed until past states are done.
- As a result, training RNNs on very large datasets becomes slow and inefficient.

Transformer and self-attention

- In Transformers, attention occurs within a single sentence — all words attend to all words in the previous layer.
  (Self-attention + Cross-attention)

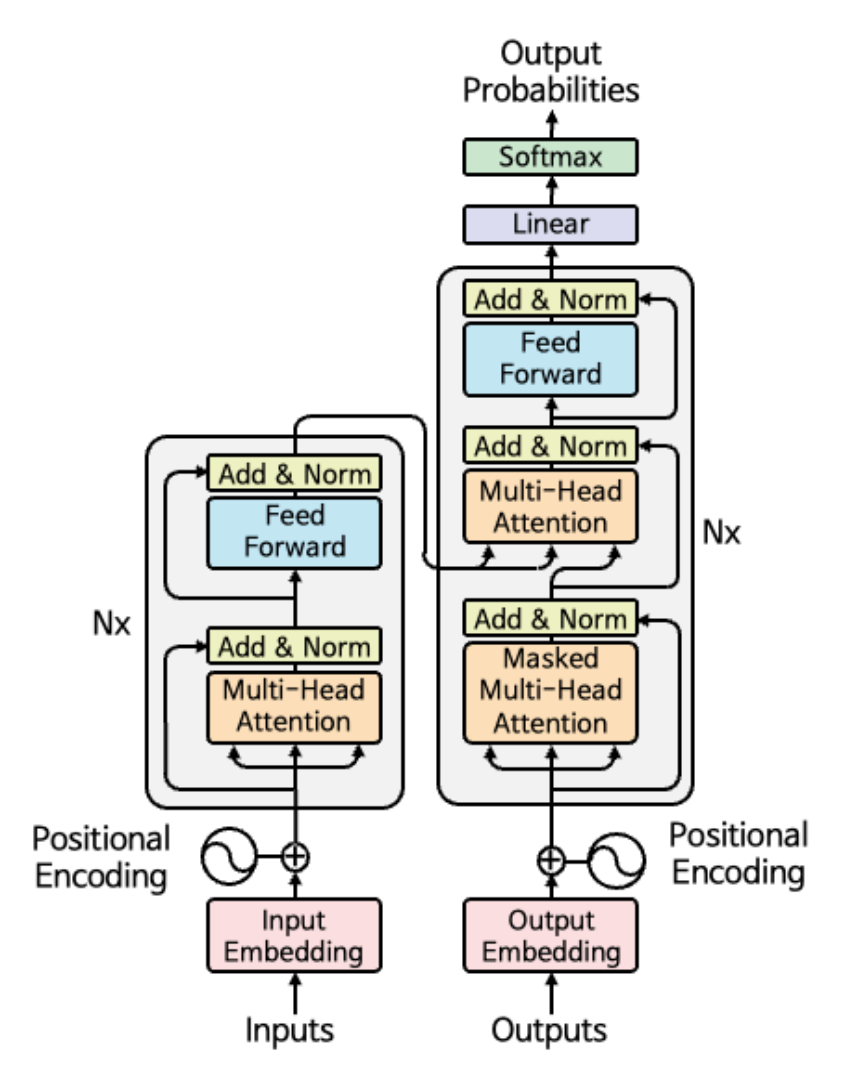
Transformer and self-attention

- In Transformers, attention occurs within a single sentence — all words attend to all words in the previous layer.
  (Self-attention + Cross-attention)
- As a result, Transformers overcome both long-distance dependency and lack of parallelizability.

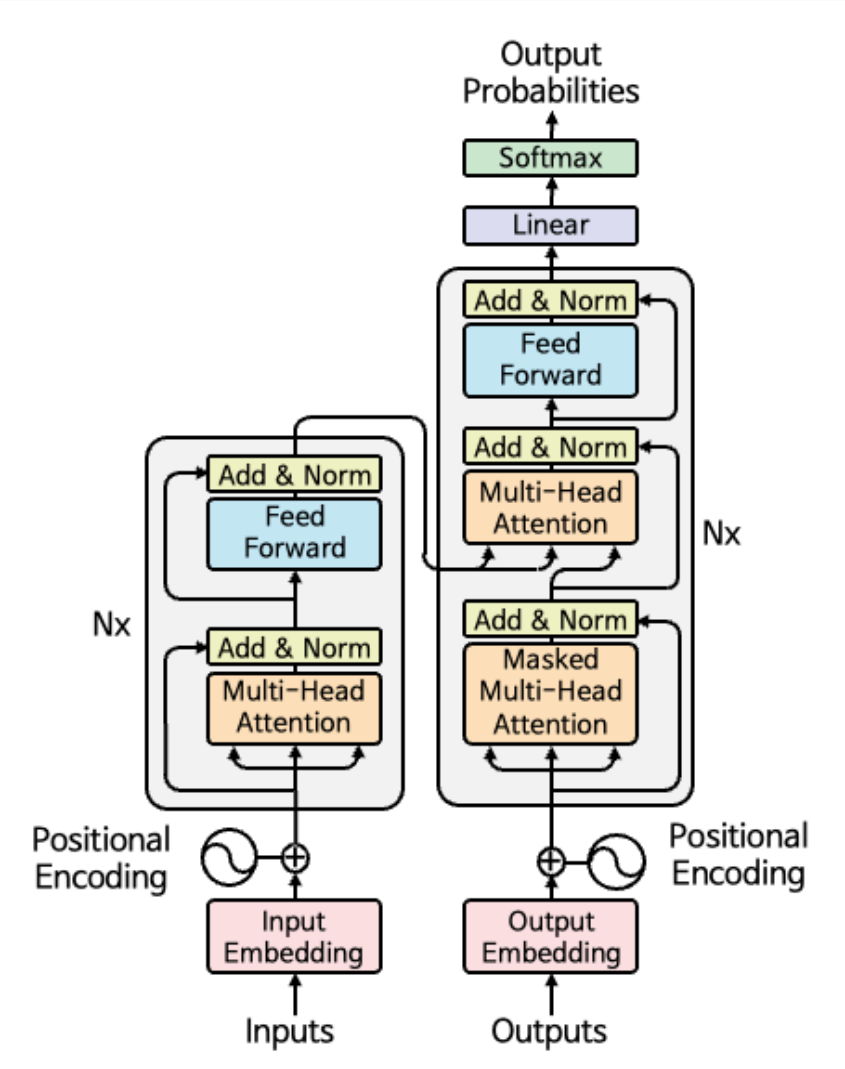
Transformer and self-attention

- In Transformers, attention occurs within a single sentence — all words attend to all words in the previous layer.
  (Self-attention + Cross-attention)
- As a result, Transformers overcome both long-distance dependency and lack of parallelizability.
- *Notes.* This was NOT an entirely new ways of looking NLP problems (e.g., probabilistic language models $\rightarrow$ neural network), but made a huge progress in the field.

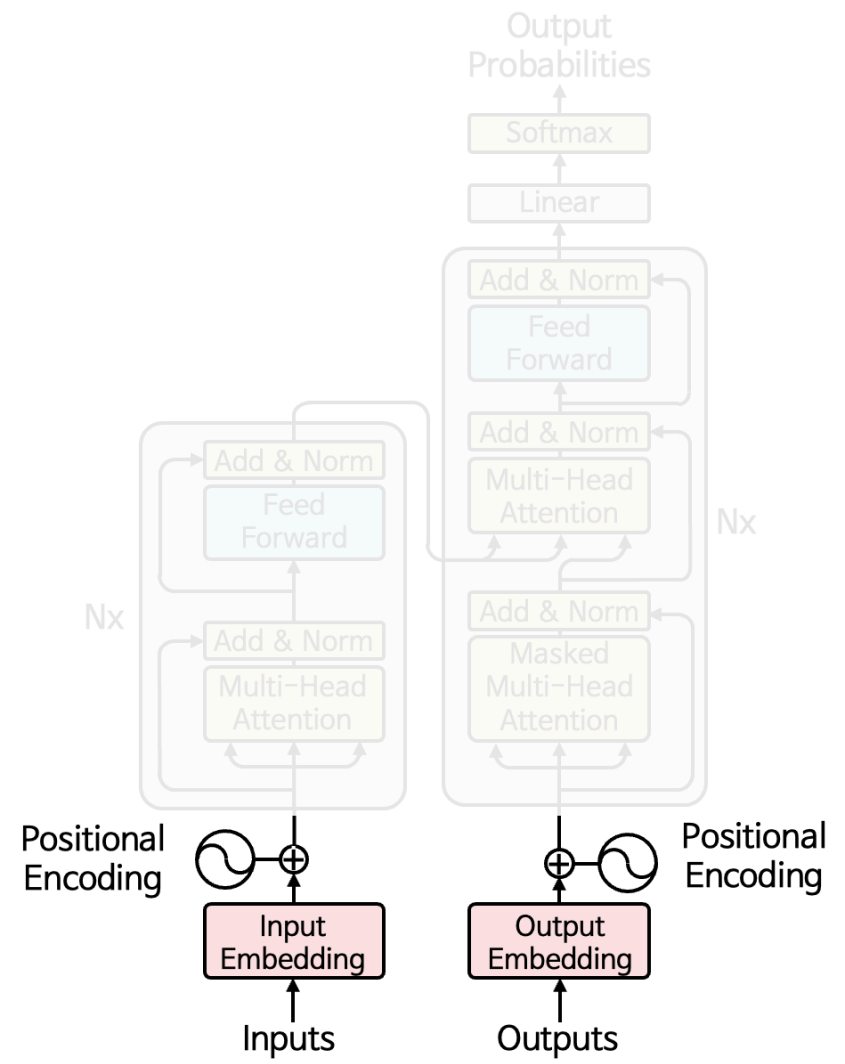
The structure of the Transformer (Vaswani et al., 2017):



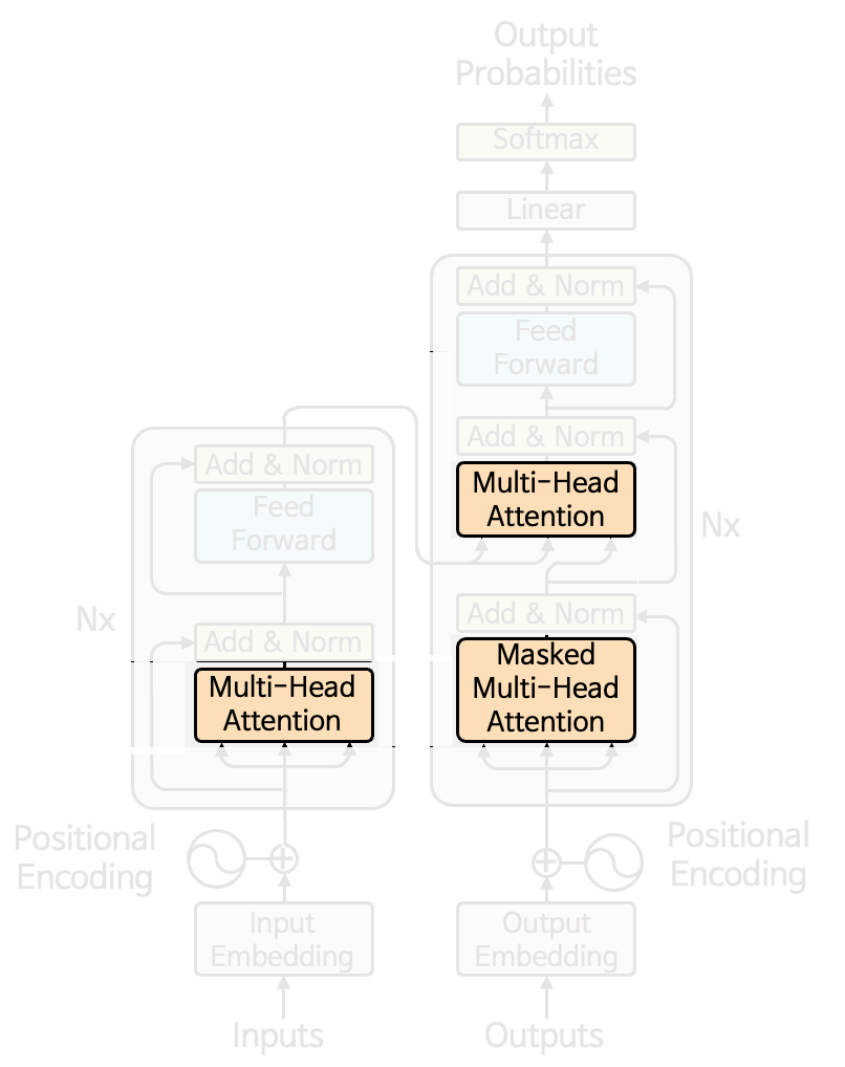
We can see that the same kinds of blocks are **repeatedly stacked**.



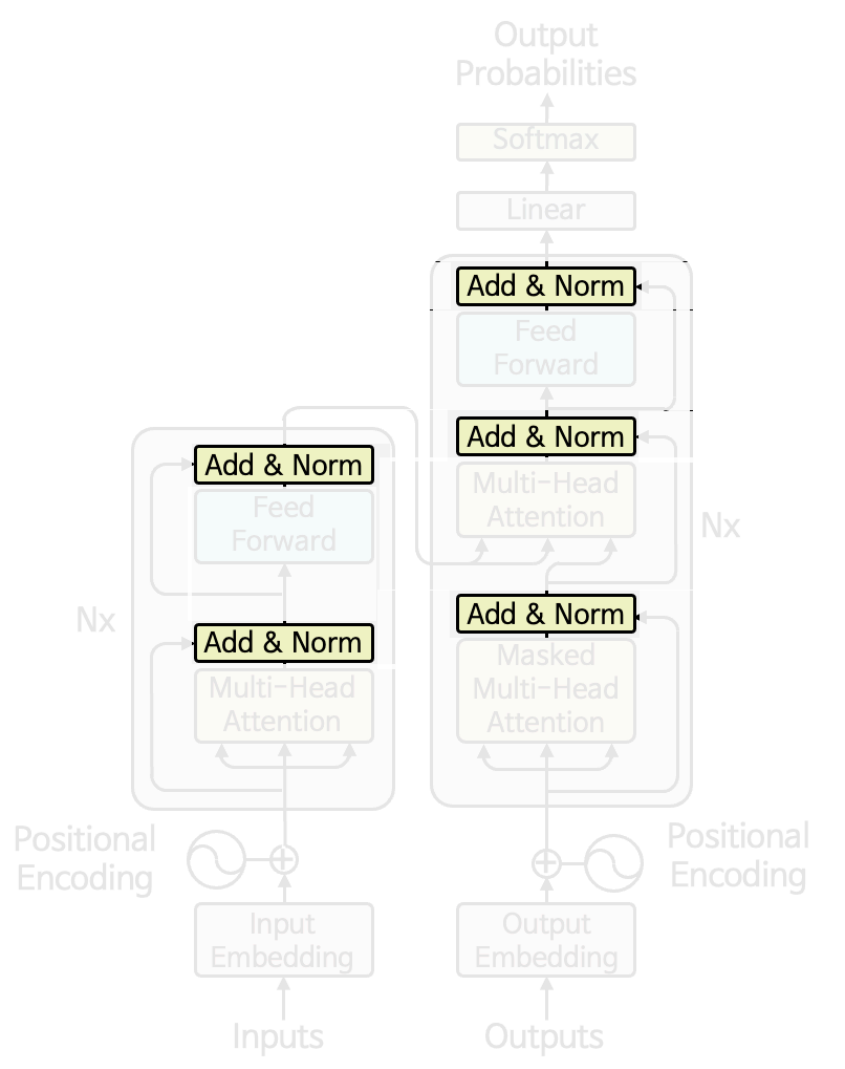
Embedding layer



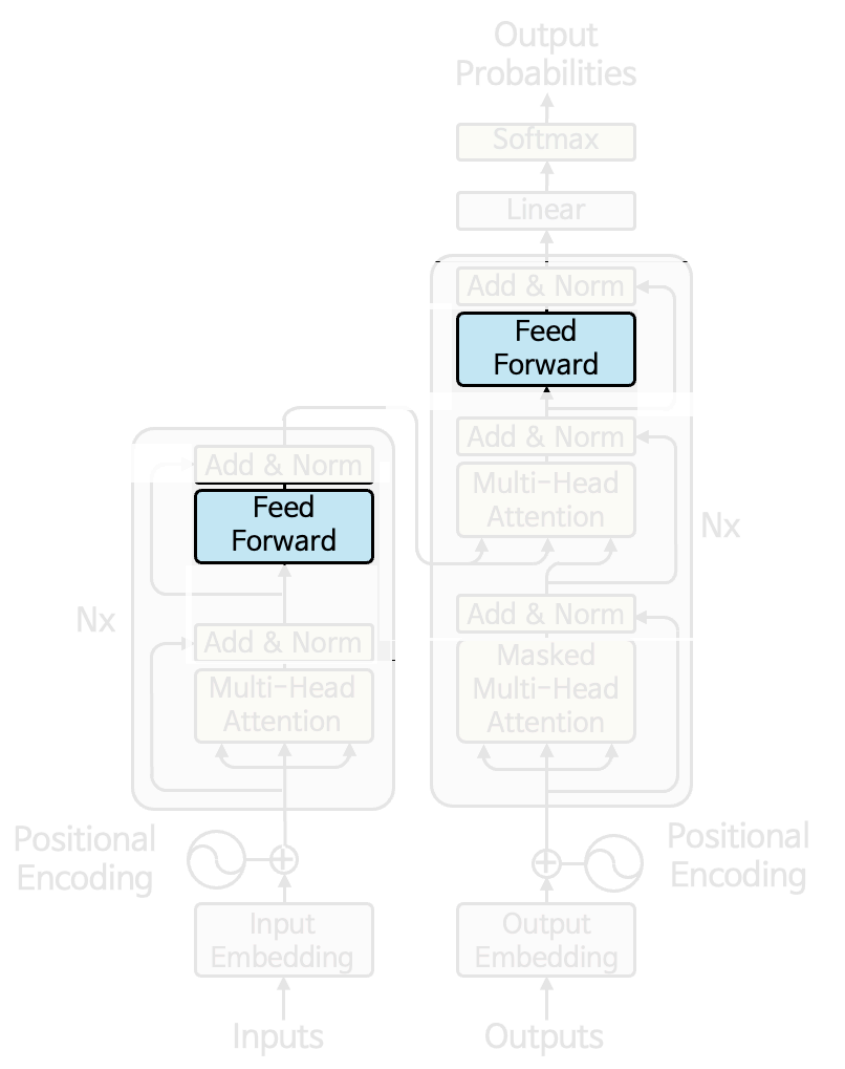
Multi-head attention



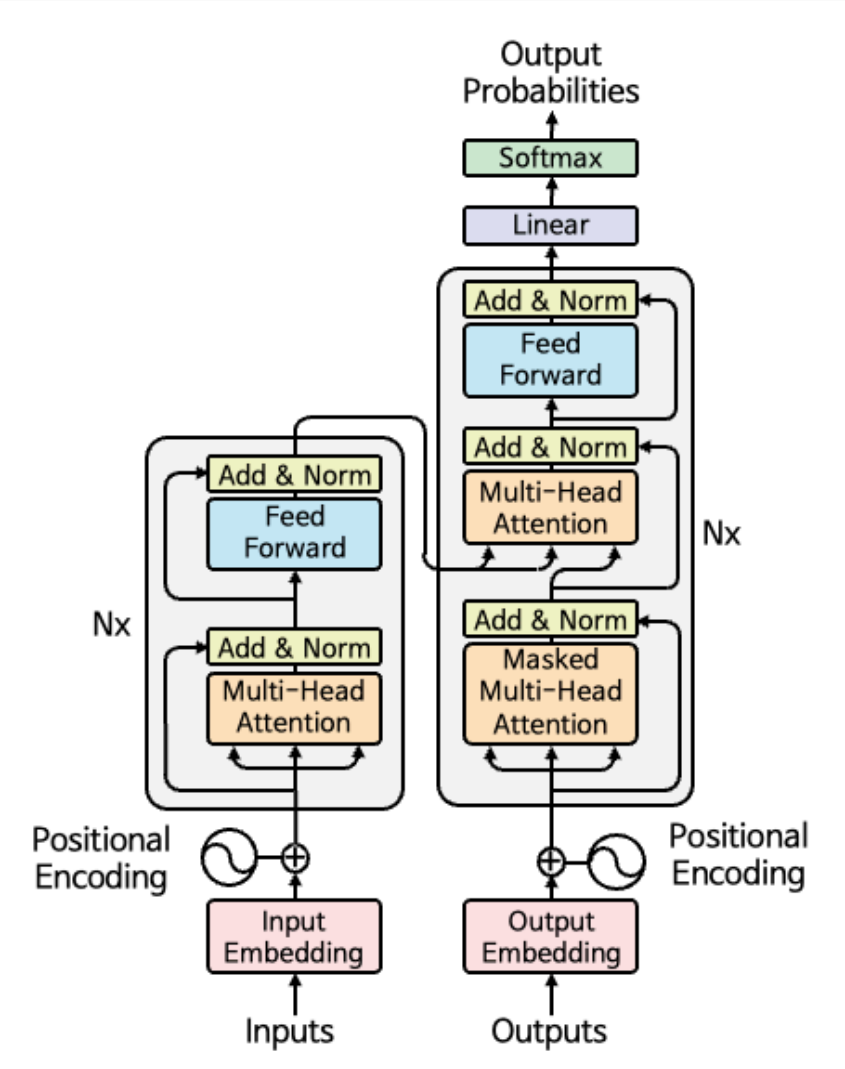
Add & Norm layer



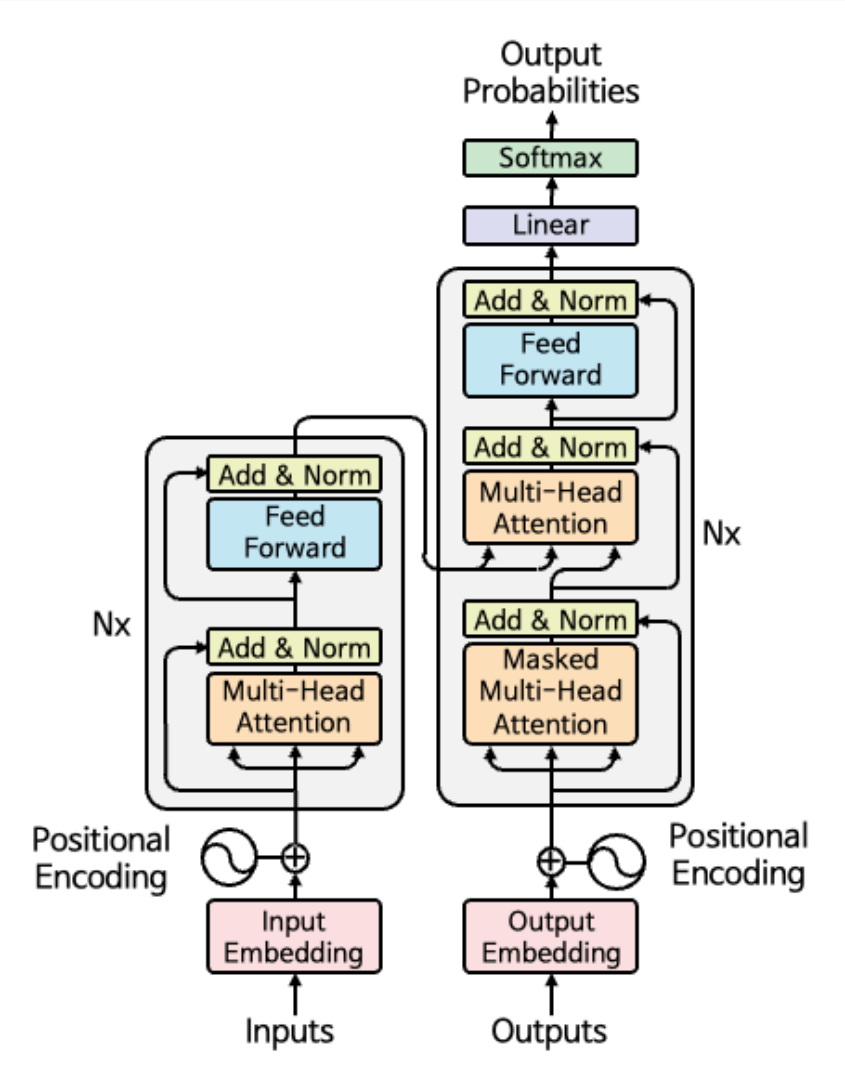
and also the Feed-Forward layer.



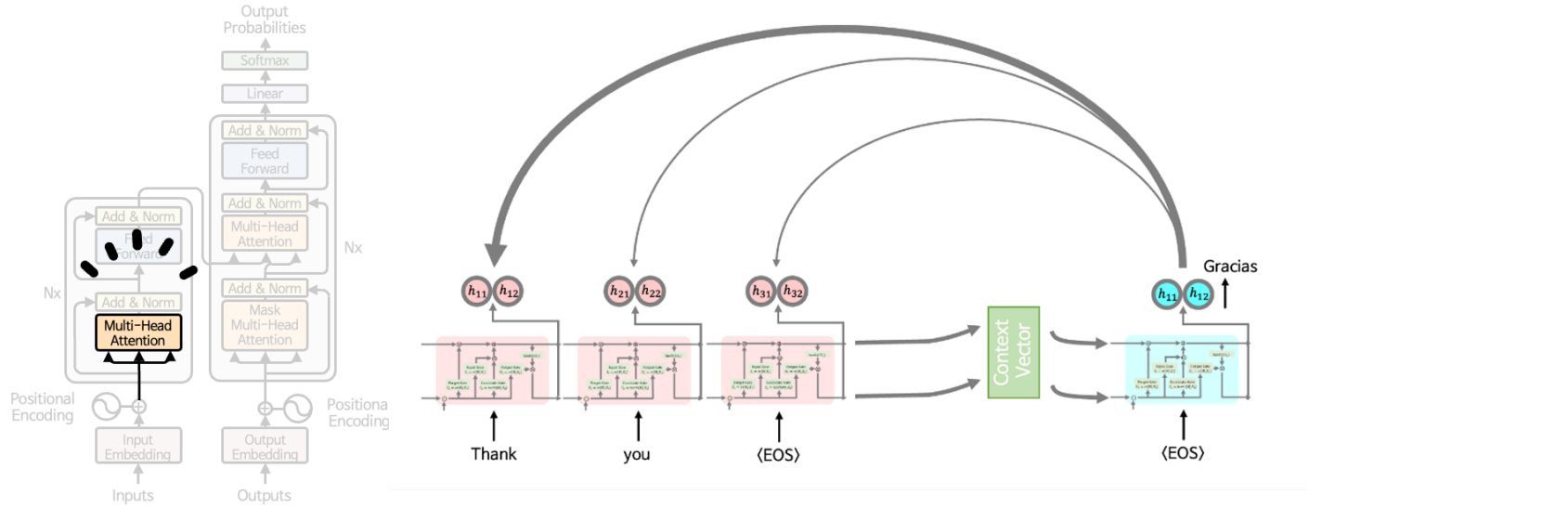
So, it’s actually made up of a few components that are repeatedly stacked.



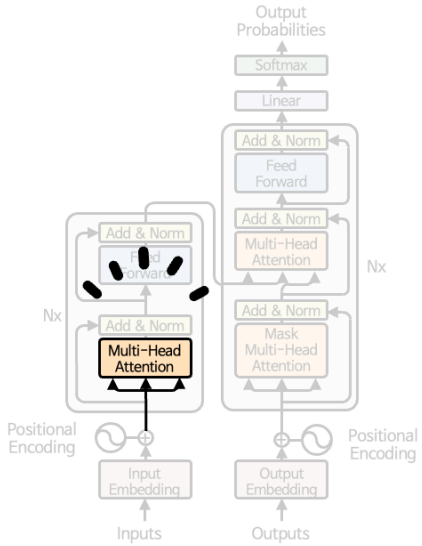
Which of these do you think is the most important feature of the Transformer model?

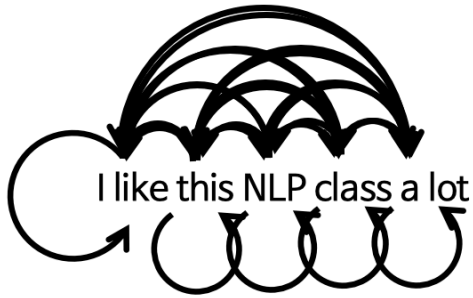


The Transformer’s multi-head attention is (1) different from the attention mechanism used in traditional seq2seq models.

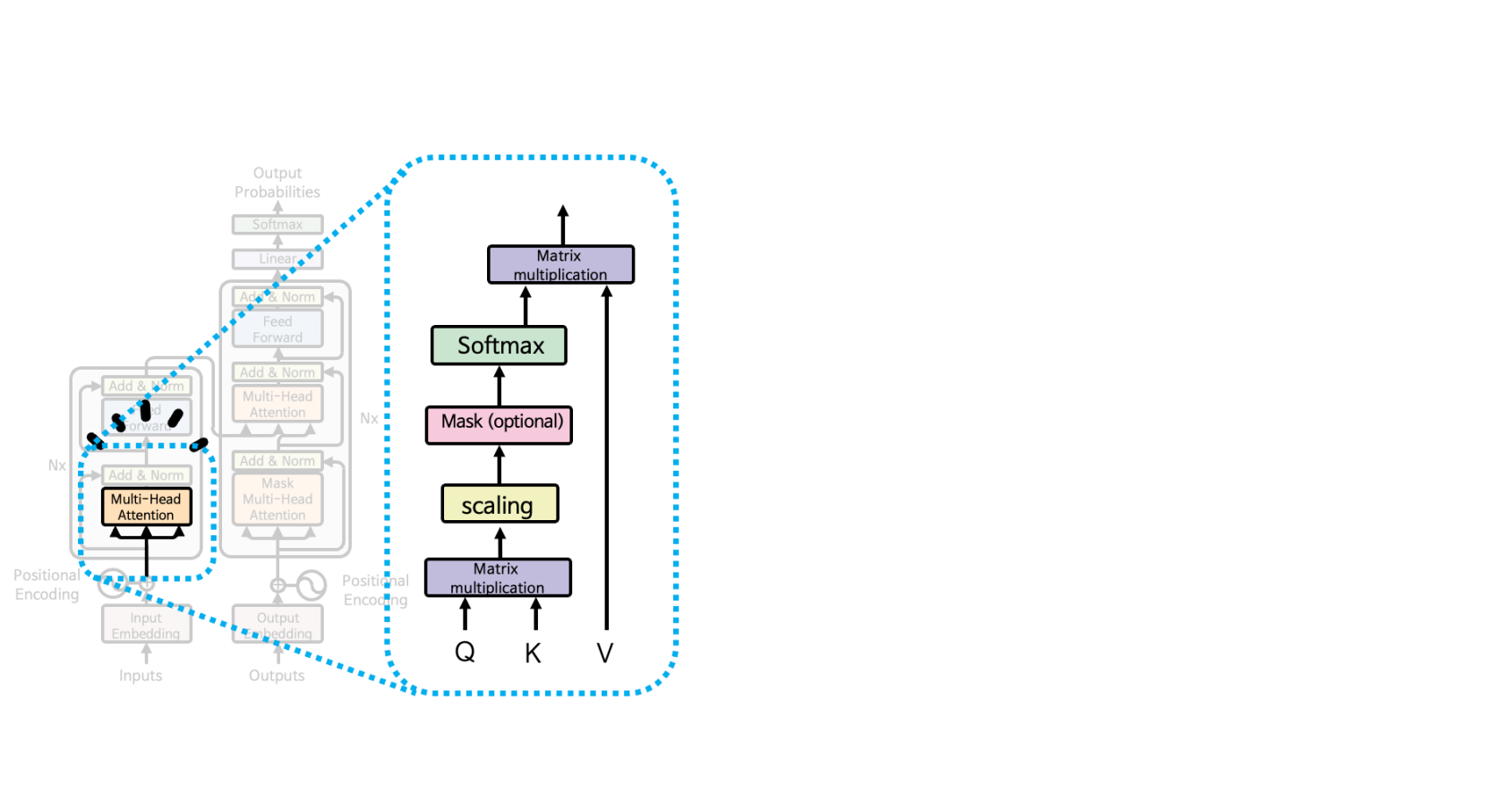


The Transformer’s attention (2) captures the relationships between words within the same input sentence.

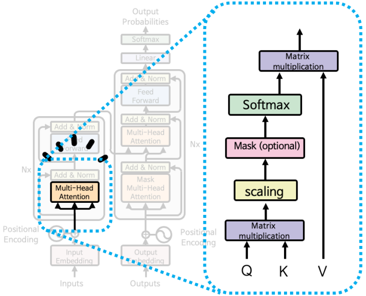




The structure of the multi-head attention mechanism used for self-attention looks like this:



We make three copies of the input + positional encoding matrix.

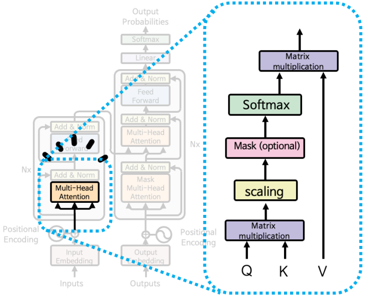


-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

We make three copies of the input + positional encoding matrix.



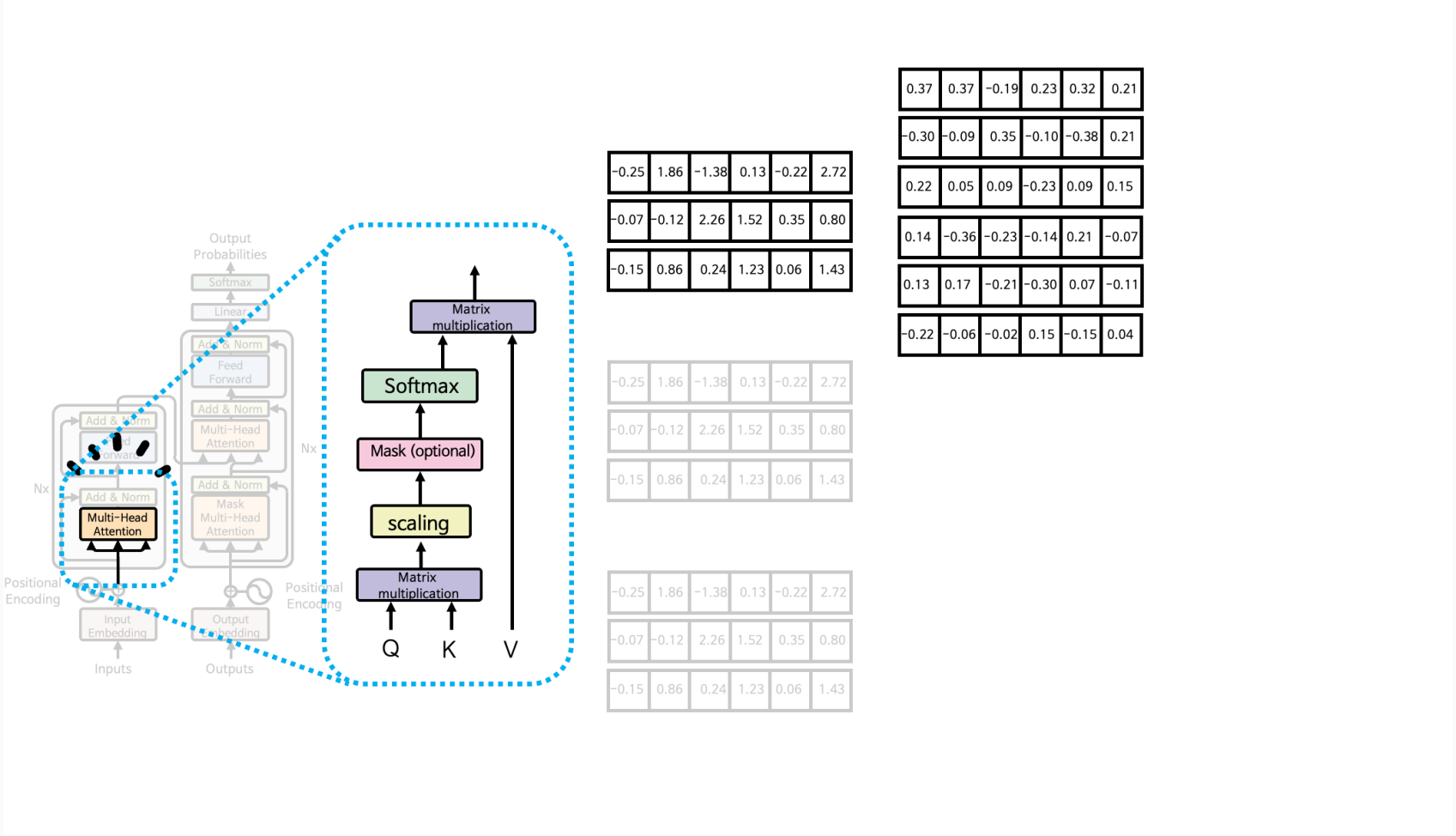
-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

This is done to create Query (Q), Key (K), and Value (V) matrices, each representing a different projection of the same input for the attention mechanism.

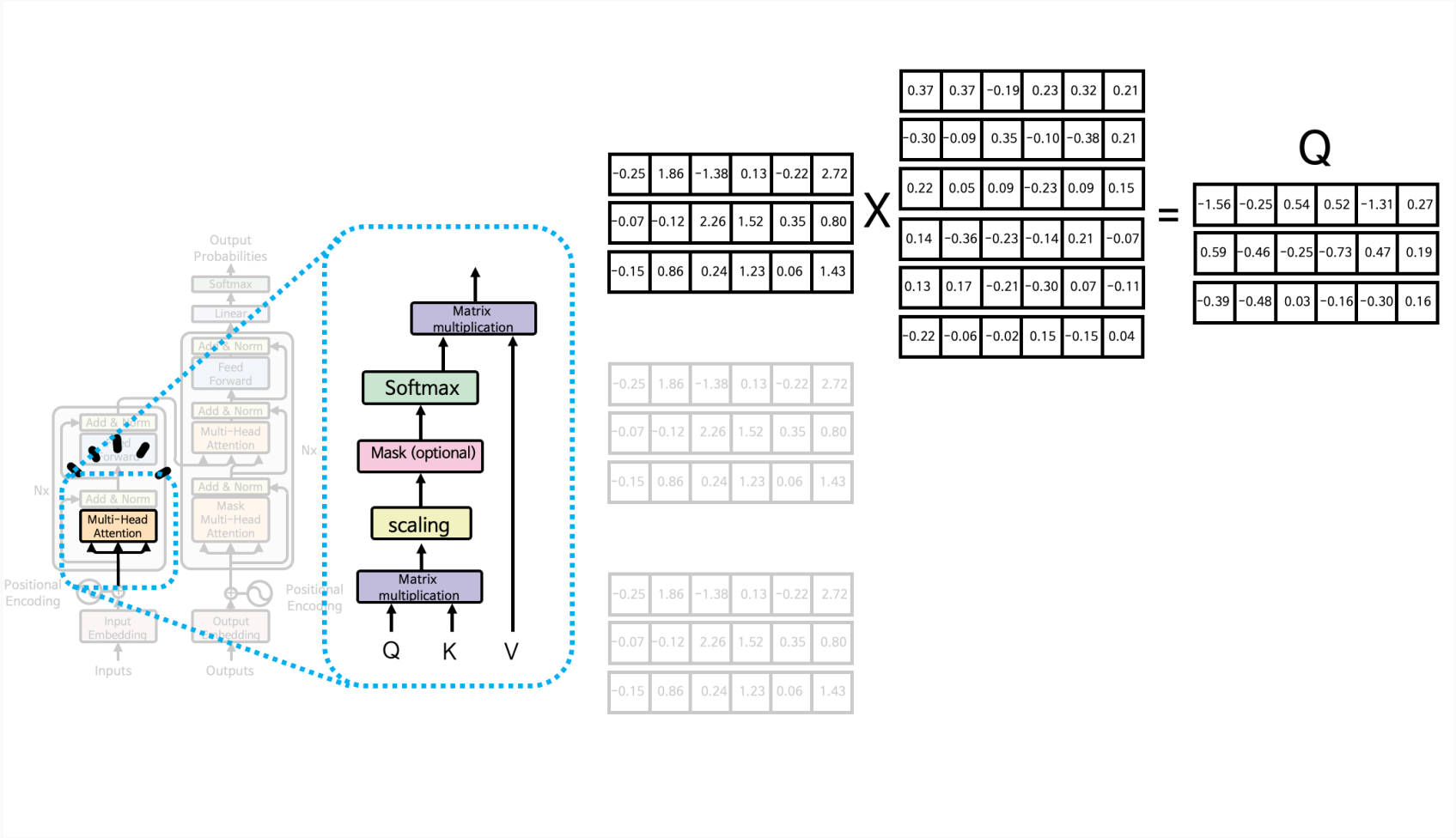
To obtain the Q matrix, we create the following 6×6 weight matrix (randomly initialized).



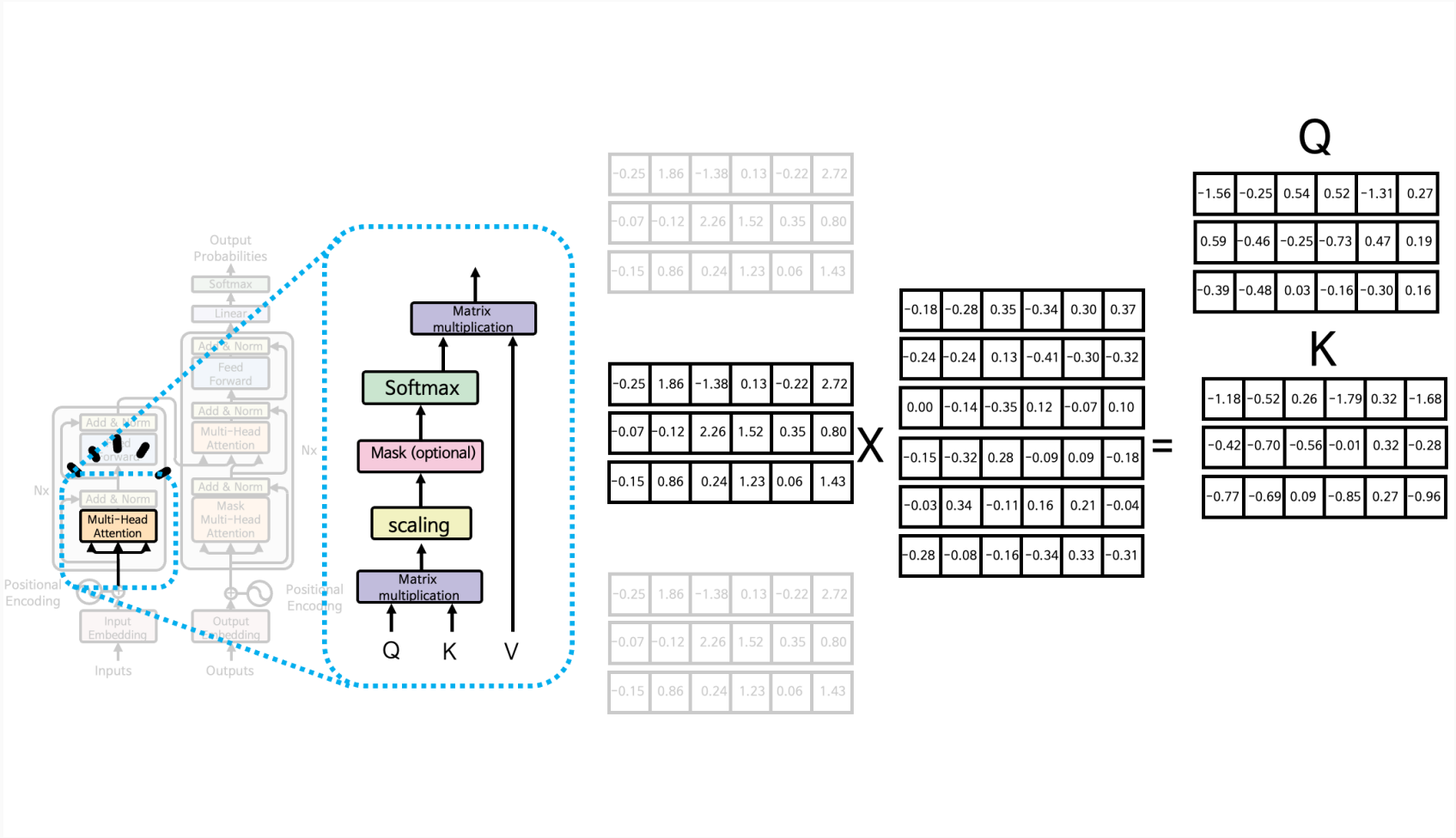
-0.25	1.86	-1.38	0.13	-0.22	2.72
-0.07	-0.12	2.26	1.52	0.35	0.80
-0.15	0.86	0.24	1.23	0.06	1.43

0.37	0.37	-0.19	0.23	0.32	0.21
-0.30	-0.09	0.35	-0.10	-0.38	0.21
0.22	0.05	0.09	-0.23	0.09	0.15
0.14	-0.36	-0.23	-0.14	0.21	-0.07
0.13	0.17	-0.21	-0.30	0.07	-0.11
-0.22	-0.06	-0.02	0.15	-0.15	0.04

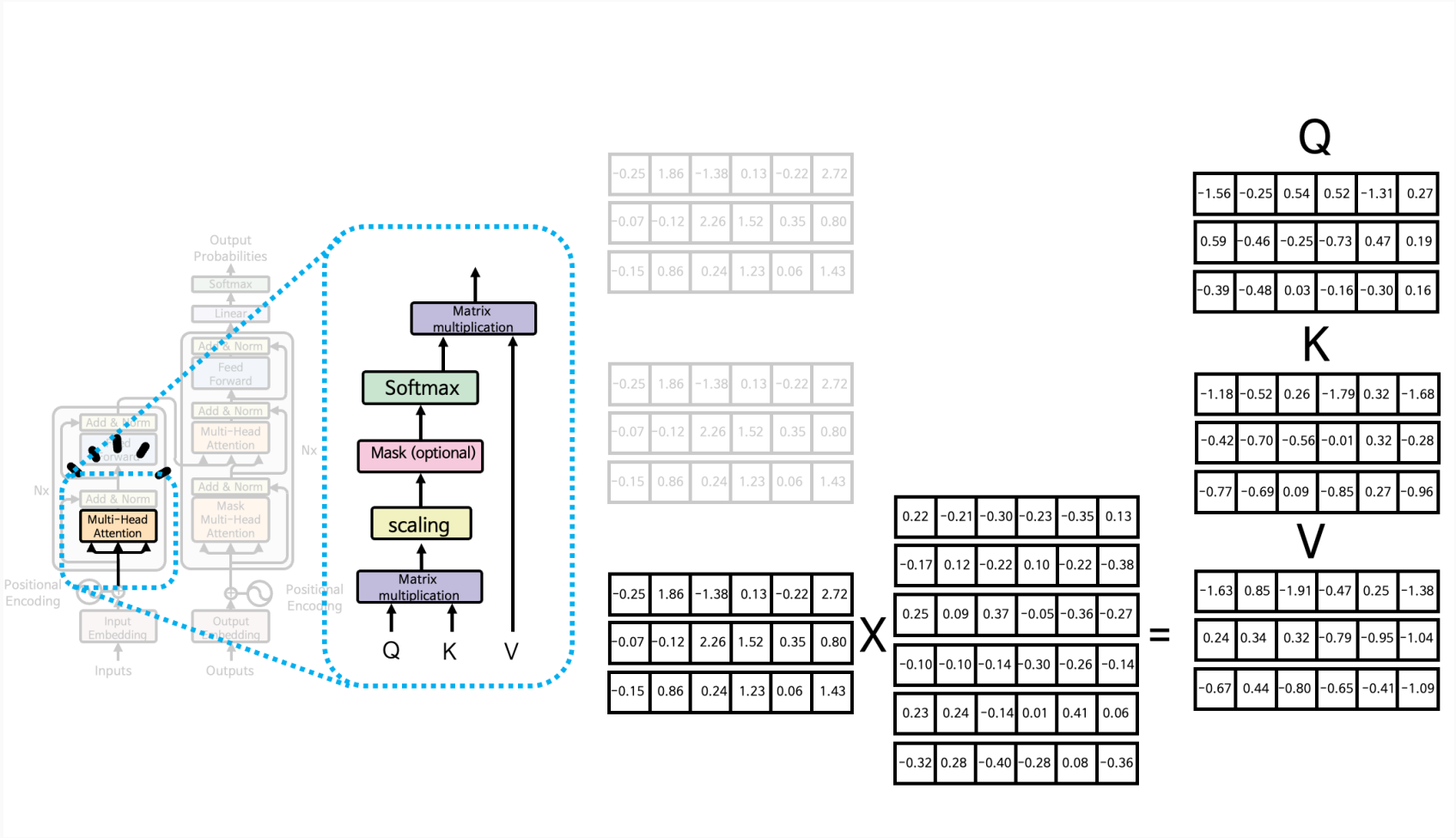
And then we perform matrix multiplication to obtain the Q (Query) matrix.



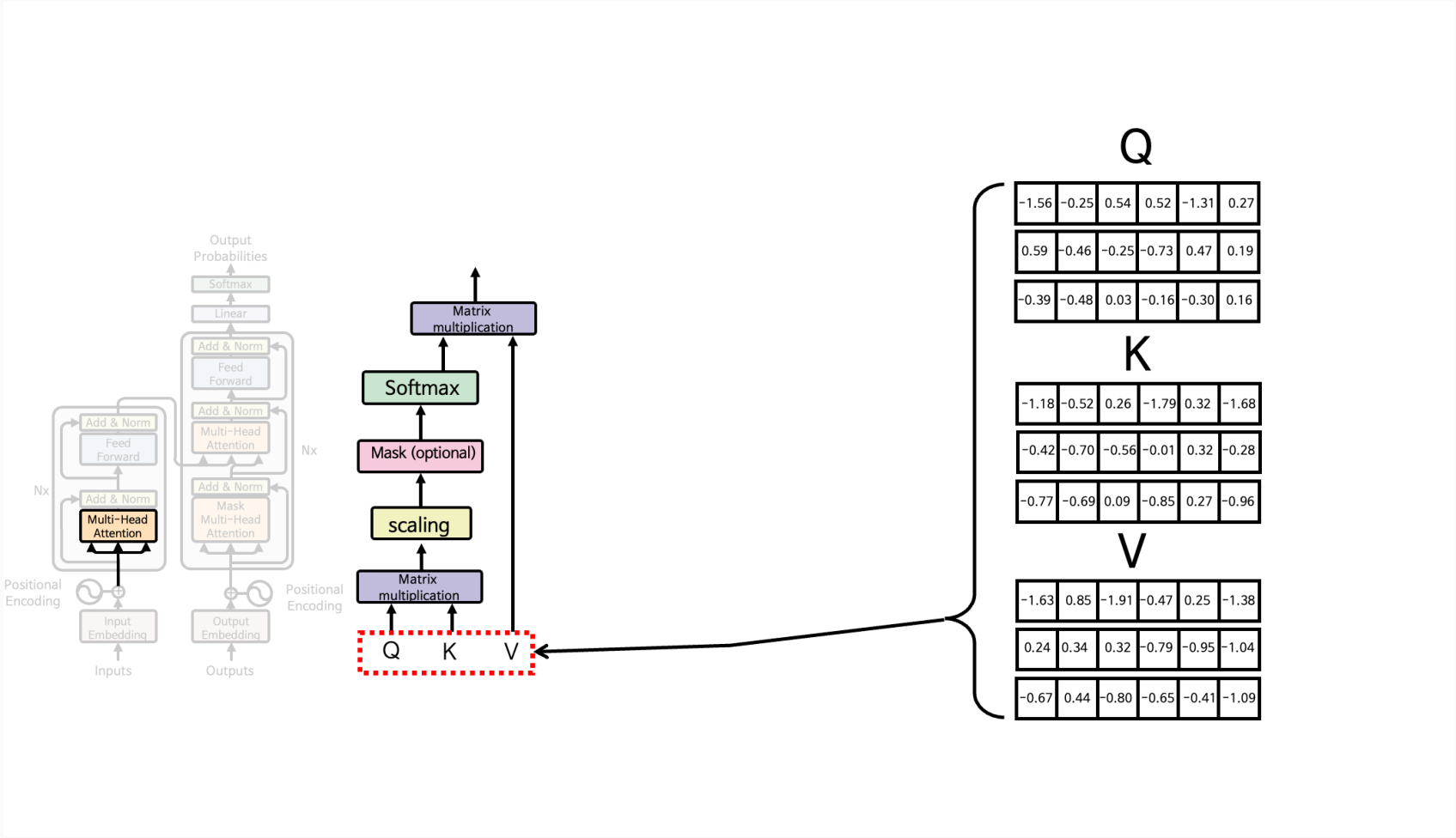
To compute the K (Key) matrix, we create another 6×6 weight matrix (randomly initialized) and multiply it with the input.



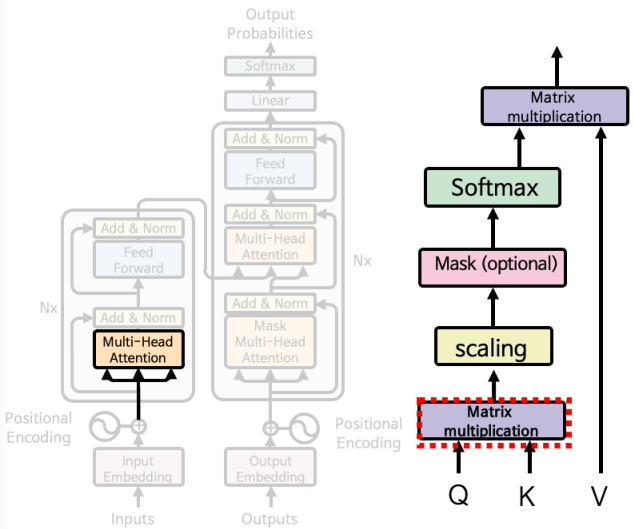
Using the same process, we perform another matrix multiplication to obtain the V (Value) matrix.



Now, we have the three inputs of the multi-head attention layer: Q (Query), K (Key), and V (Value).



Next, we perform the matrix multiplication between Q and K. The formula for this operation is as follows:



$$Q \cdot K^T$$

Q

-1.56	-0.25	0.54	0.52	-1.31	0.27
0.59	-0.46	-0.25	-0.73	0.47	0.19
-0.39	-0.48	0.03	-0.16	-0.30	0.16

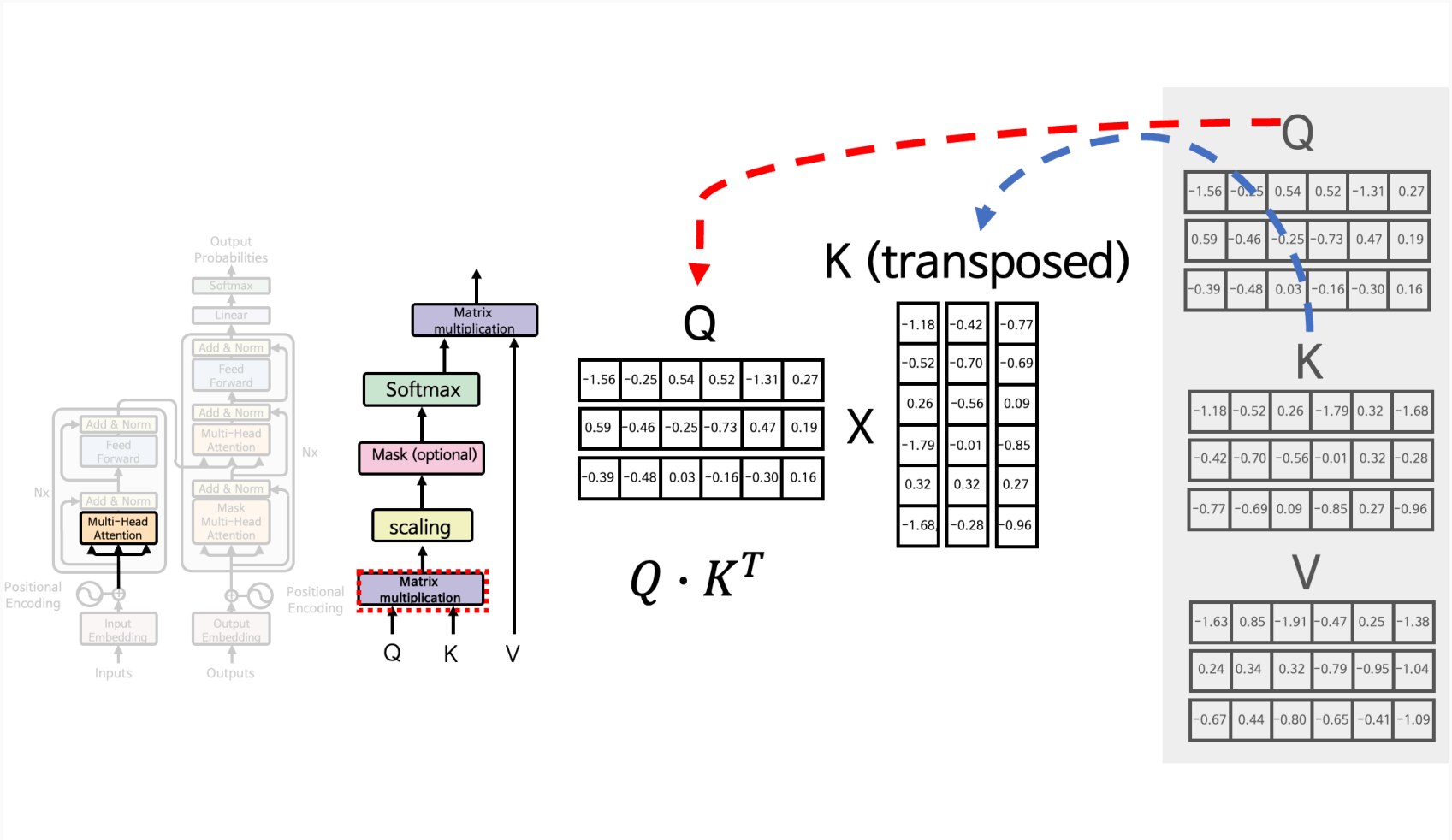
K

-1.18	-0.52	0.26	-1.79	0.32	-1.68
-0.42	-0.70	-0.56	-0.01	0.32	-0.28
-0.77	-0.69	0.09	-0.85	0.27	-0.96

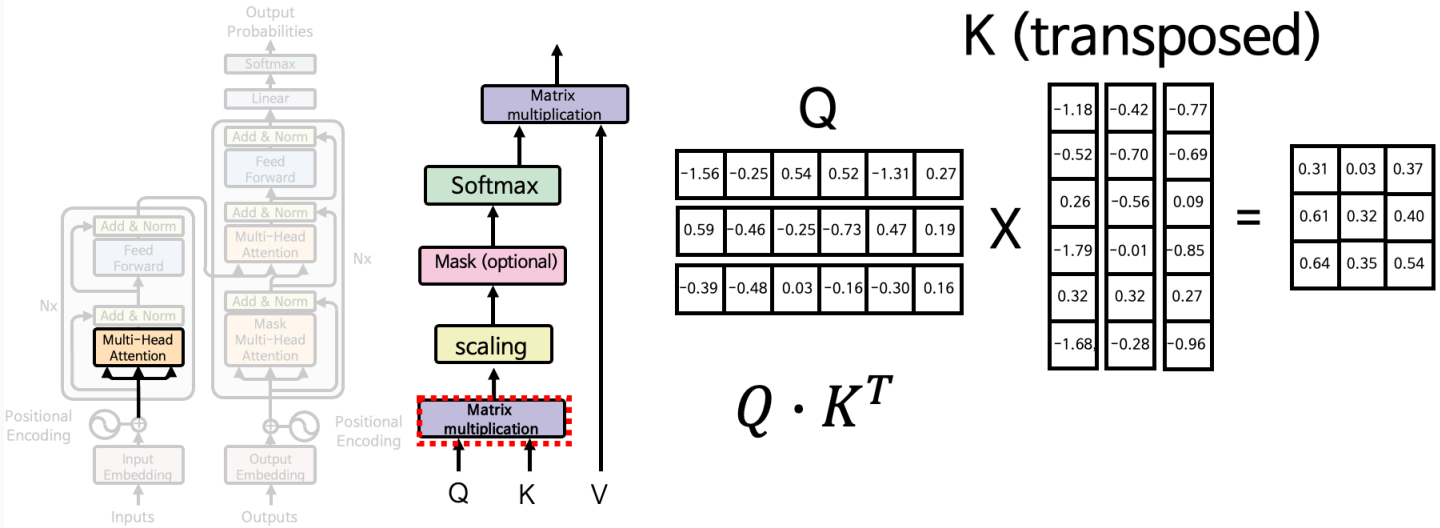
V

-1.63	0.85	-1.91	-0.47	0.25	-1.38
0.24	0.34	0.32	-0.79	-0.95	-1.04
-0.67	0.44	-0.80	-0.65	-0.41	-1.09

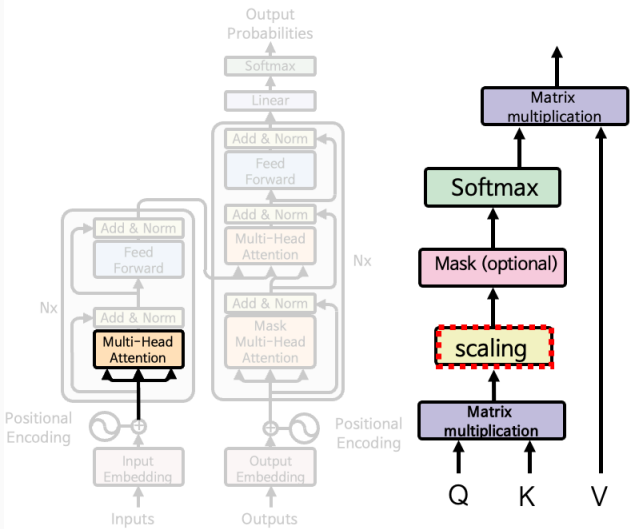
When we plug in the matrix values and calculate,



we can obtain the result as follows:

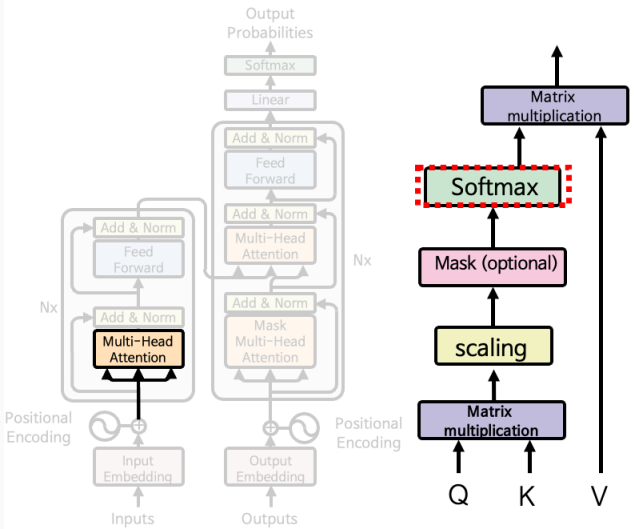


Next, we apply *scaling*, which divides the matrix by $\sqrt{6}$, since in this example the value of d_{model} is 6.



$$Q \cdot K^T = \begin{bmatrix} 0.31 & 0.03 & 0.37 \\ 0.61 & 0.32 & 0.40 \\ 0.64 & 0.35 & 0.54 \end{bmatrix} \div \sqrt{6} = \begin{bmatrix} 0.13 & 0.01 & 0.15 \\ 0.25 & 0.13 & 0.16 \\ 0.26 & 0.14 & 0.22 \end{bmatrix}$$

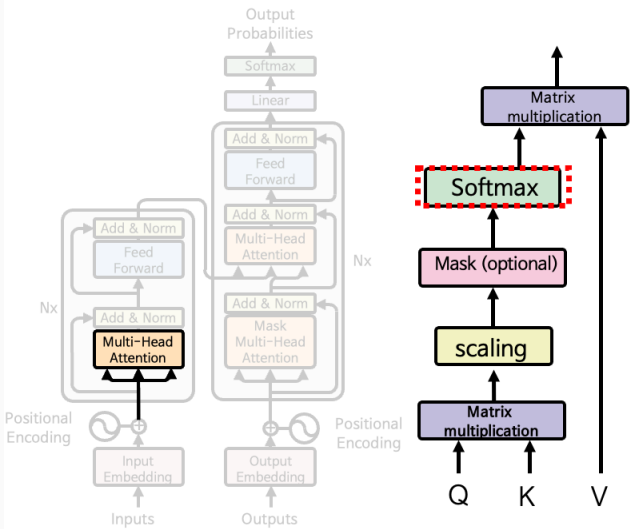
The next softmax layer converts the matrix values into probabilities.



$$softmax\left(\begin{bmatrix}0.13 & 0.01 & 0.15\\ 0.25 & 0.13 & 0.16\\ 0.26 & 0.14 & 0.22\end{bmatrix}\right) = \begin{bmatrix}0.34 & 0.31 & 0.35\\ 0.36 & 0.32 & 0.33\\ 0.35 & 0.31 & 0.34\end{bmatrix}$$

$$Q \cdot K^T = \begin{bmatrix}0.31 & 0.03 & 0.37\\ 0.61 & 0.32 & 0.40\\ 0.64 & 0.35 & 0.54\end{bmatrix} \div \sqrt{6} = \begin{bmatrix}0.13 & 0.01 & 0.15\\ 0.25 & 0.13 & 0.16\\ 0.26 & 0.14 & 0.22\end{bmatrix}$$

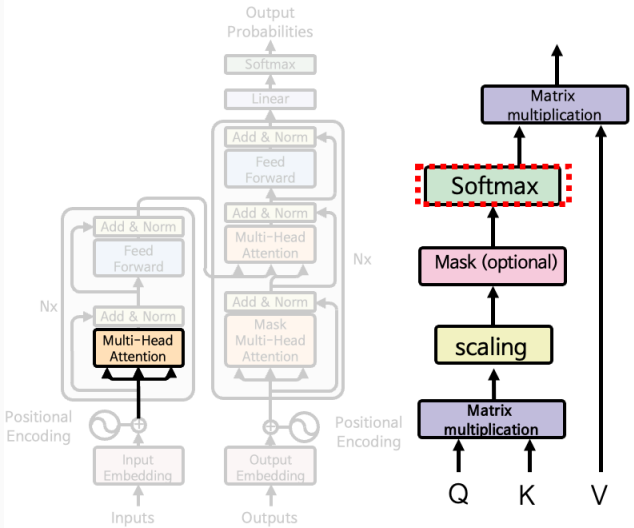
This 3×3 matrix represents the self-attention weights, which shows how each word in the input is related to every other.



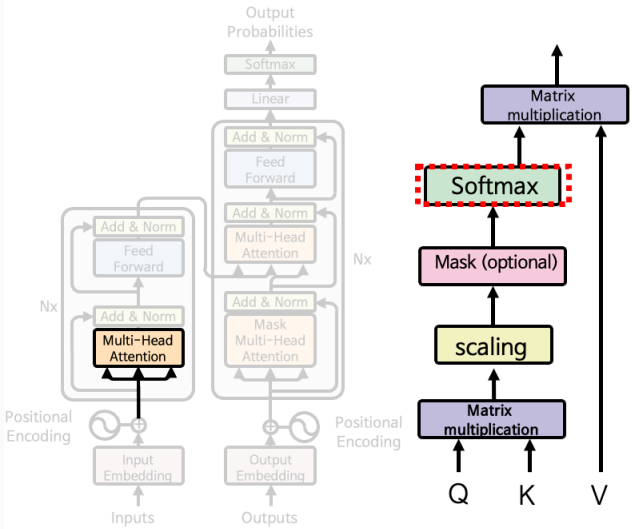
$$softmax\left(\begin{bmatrix} 0.13 & 0.01 & 0.15 \\ 0.25 & 0.13 & 0.16 \\ 0.26 & 0.14 & 0.22 \end{bmatrix}\right) =$$

	how	are	you
how	0.34	0.31	0.35
are	0.36	0.32	0.33
you	0.35	0.31	0.34

Word pairs with higher relevance receive higher attention values, while those with lower relevance receive smaller values. To be specific...

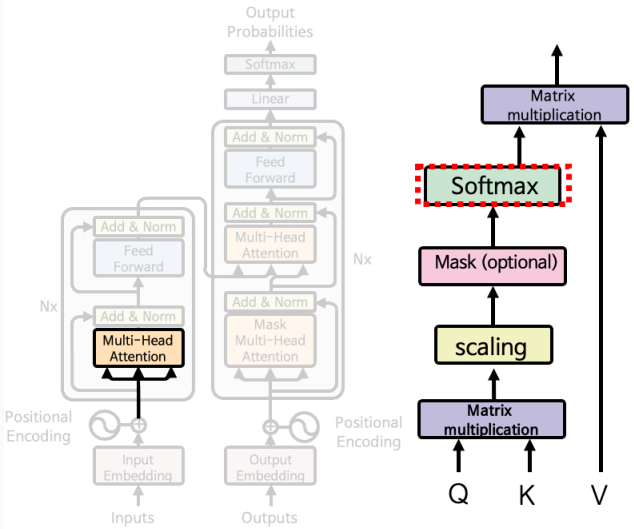


	how	are	you
how	0.34	0.31	0.35
are	0.36	0.32	0.33
you	0.35	0.31	0.34



	how	are	you
how	0.34	0.31	0.35
are	0.36	0.32	0.33
you	0.35	0.31	0.34

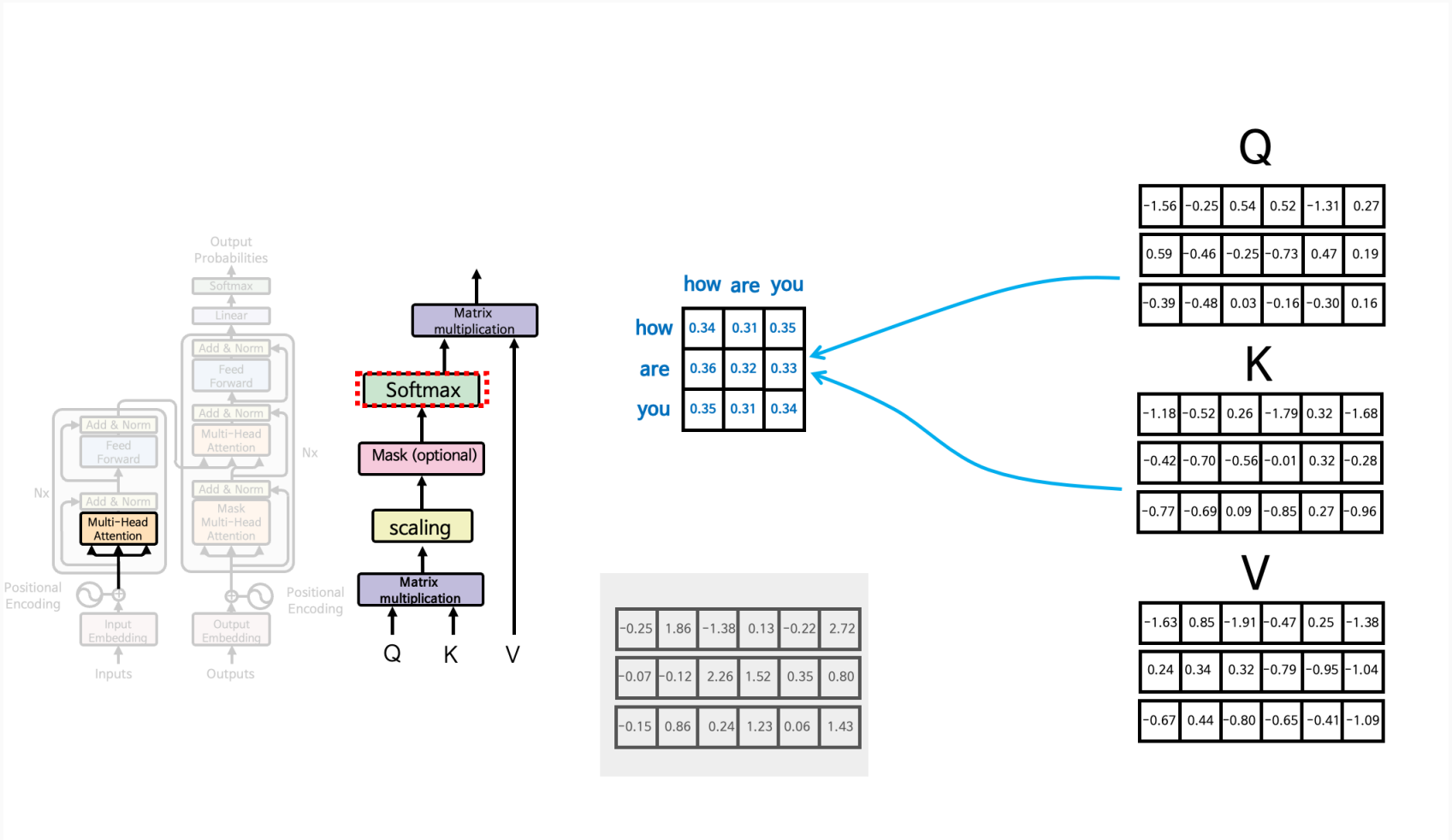
- Each *row*: how much a query word attends to other words (where it sends attention)



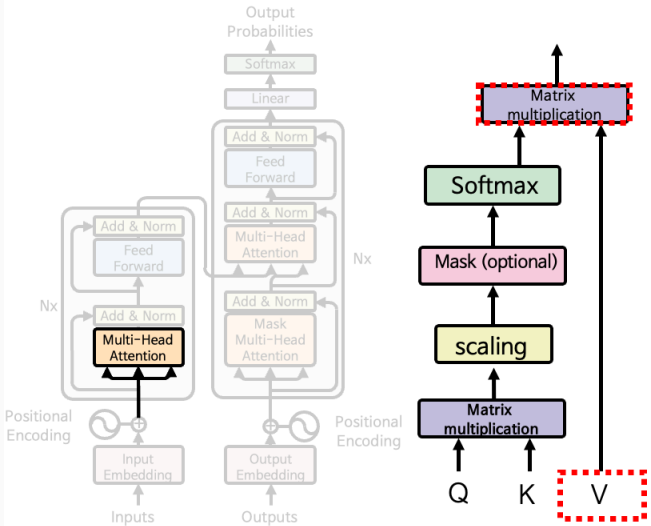
	how	are	you
how	0.34	0.31	0.35
are	0.36	0.32	0.33
you	0.35	0.31	0.34

- Each *row*: how much a query word attends to other words (where it sends attention)
- Each *column*: how much a key words is attended to by other words (where it receives attention)

This self-attention mechanism is the core structure.



So, what about the final matrix multiplication?



	how	are	you
how	0.34	0.31	0.35
are	0.36	0.32	0.33
you	0.35	0.31	0.34

Q

-1.56	-0.25	0.54	0.52	-1.31	0.27
0.59	-0.46	-0.25	-0.73	0.47	0.19
-0.39	-0.48	0.03	-0.16	-0.30	0.16

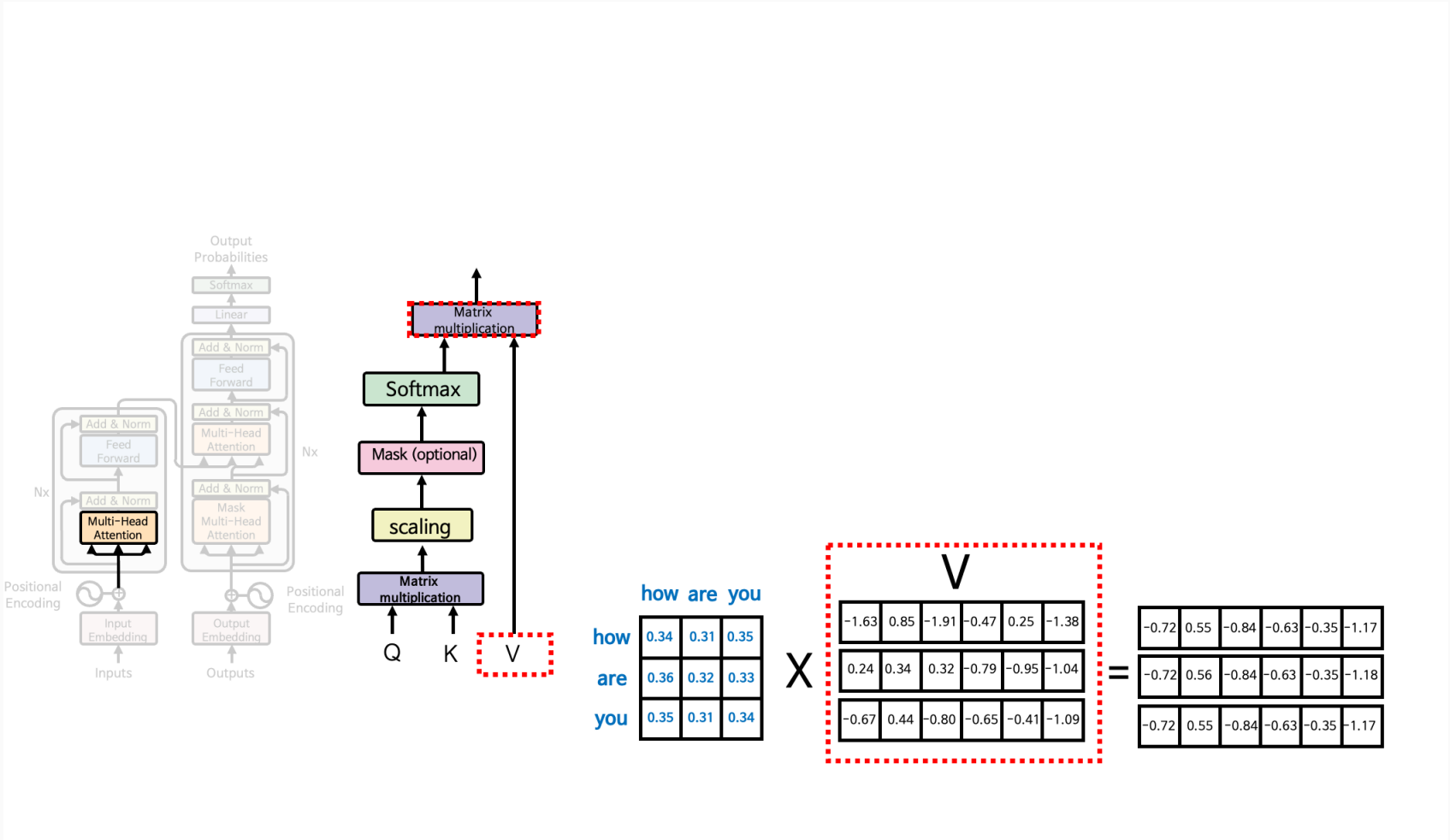
K

-1.18	-0.52	0.26	-1.79	0.32	-1.68
-0.42	-0.70	-0.56	-0.01	0.32	-0.28
-0.77	-0.69	0.09	-0.85	0.27	-0.96

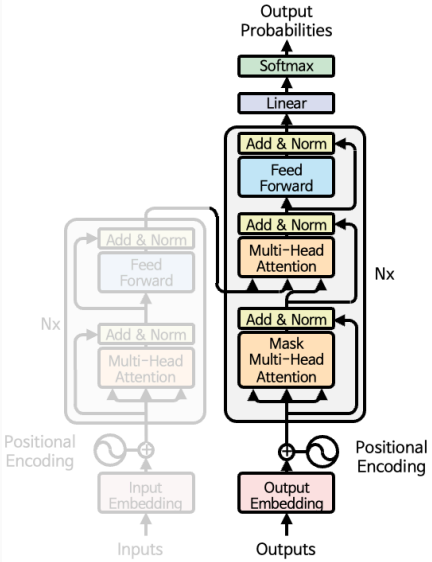
V

-1.63	0.85	-1.91	-0.47	0.25	-1.38
0.24	0.34	0.32	-0.79	-0.95	-1.04
-0.67	0.44	-0.80	-0.65	-0.41	-1.09

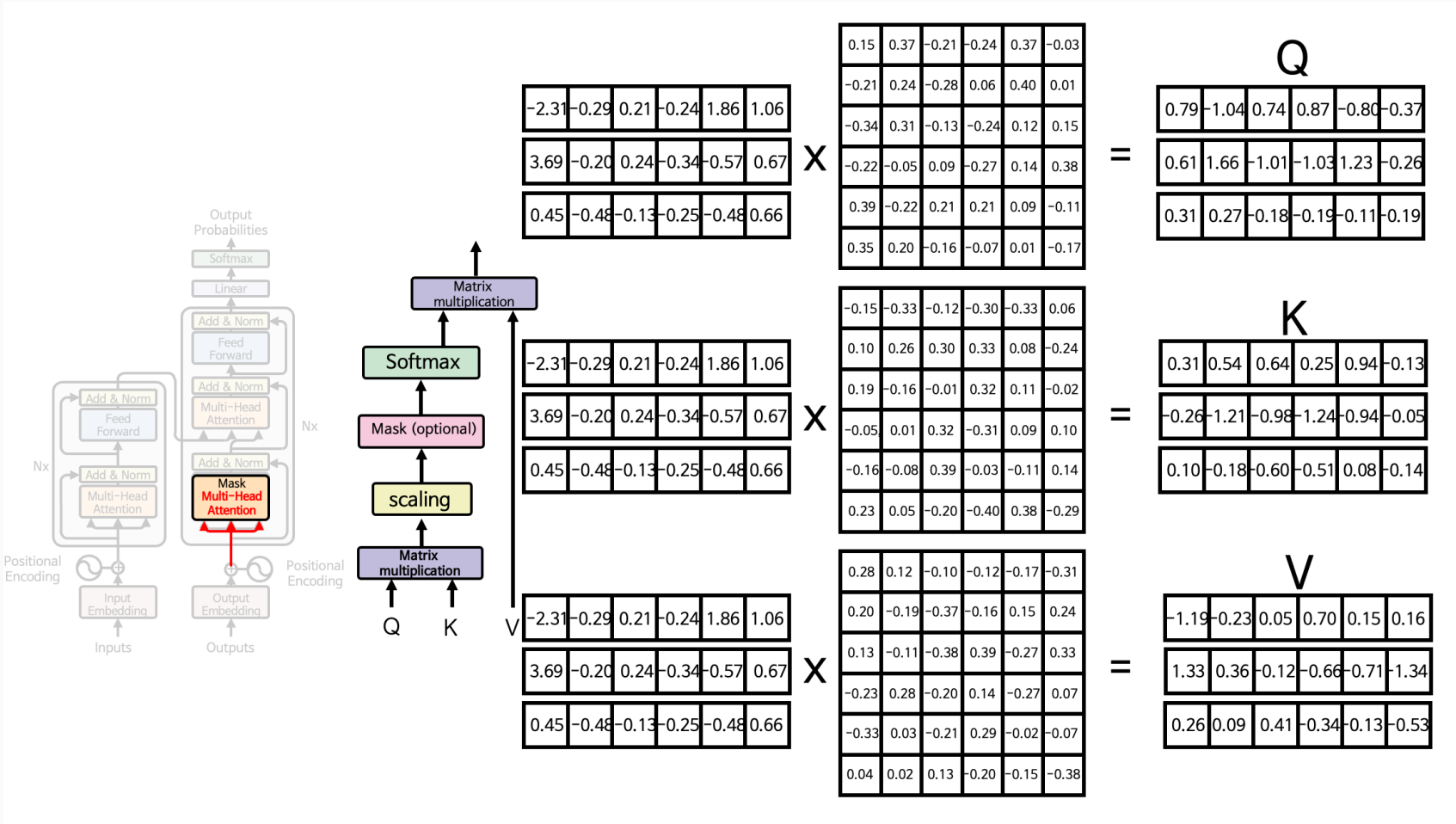
We multiply this by the V matrix to obtain the final output — a self-attended embedding that combines (1) input, (2) positional, and (3) attention information.



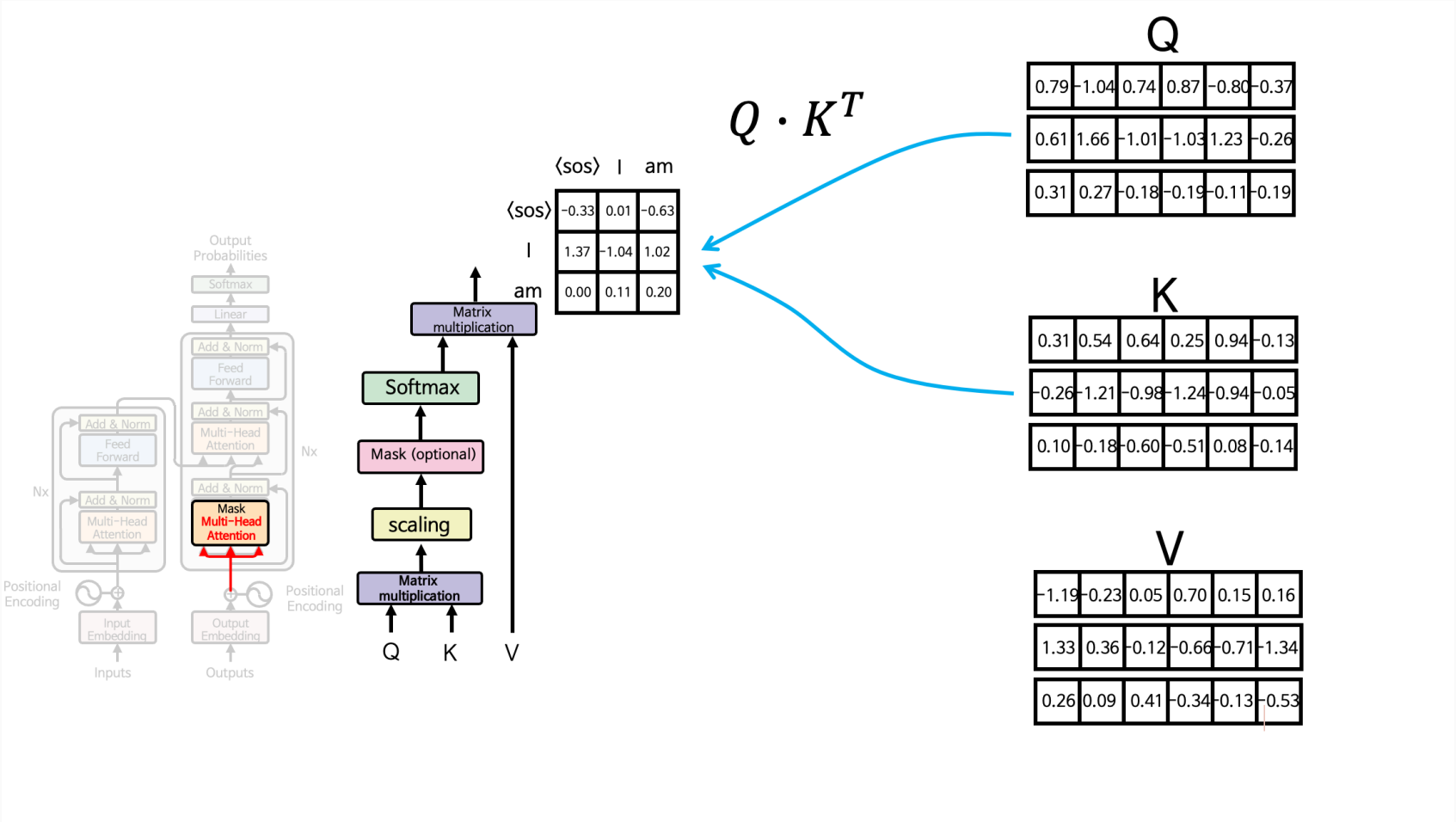
Focus on to the **Decoder**.



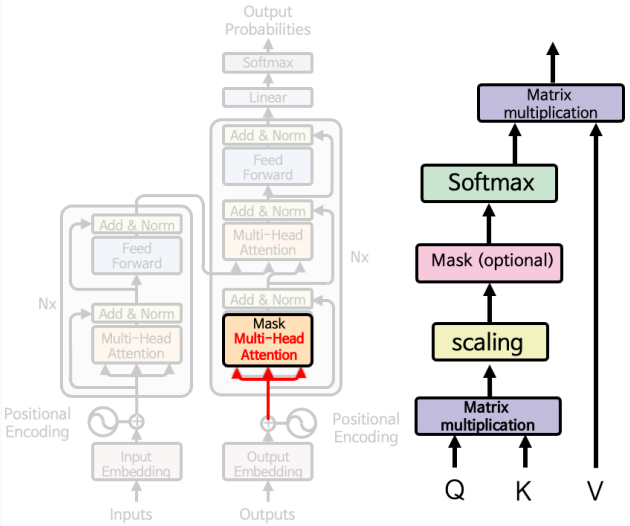
The **Masked Multi-Head Attention** operation in the decoder is almost the same as in the encoder.



We multiply the Q and K matrices to create the attention matrix.



We then scale the matrix by dividing it by $\sqrt{6}$, just as before, so that the range of values changes accordingly.

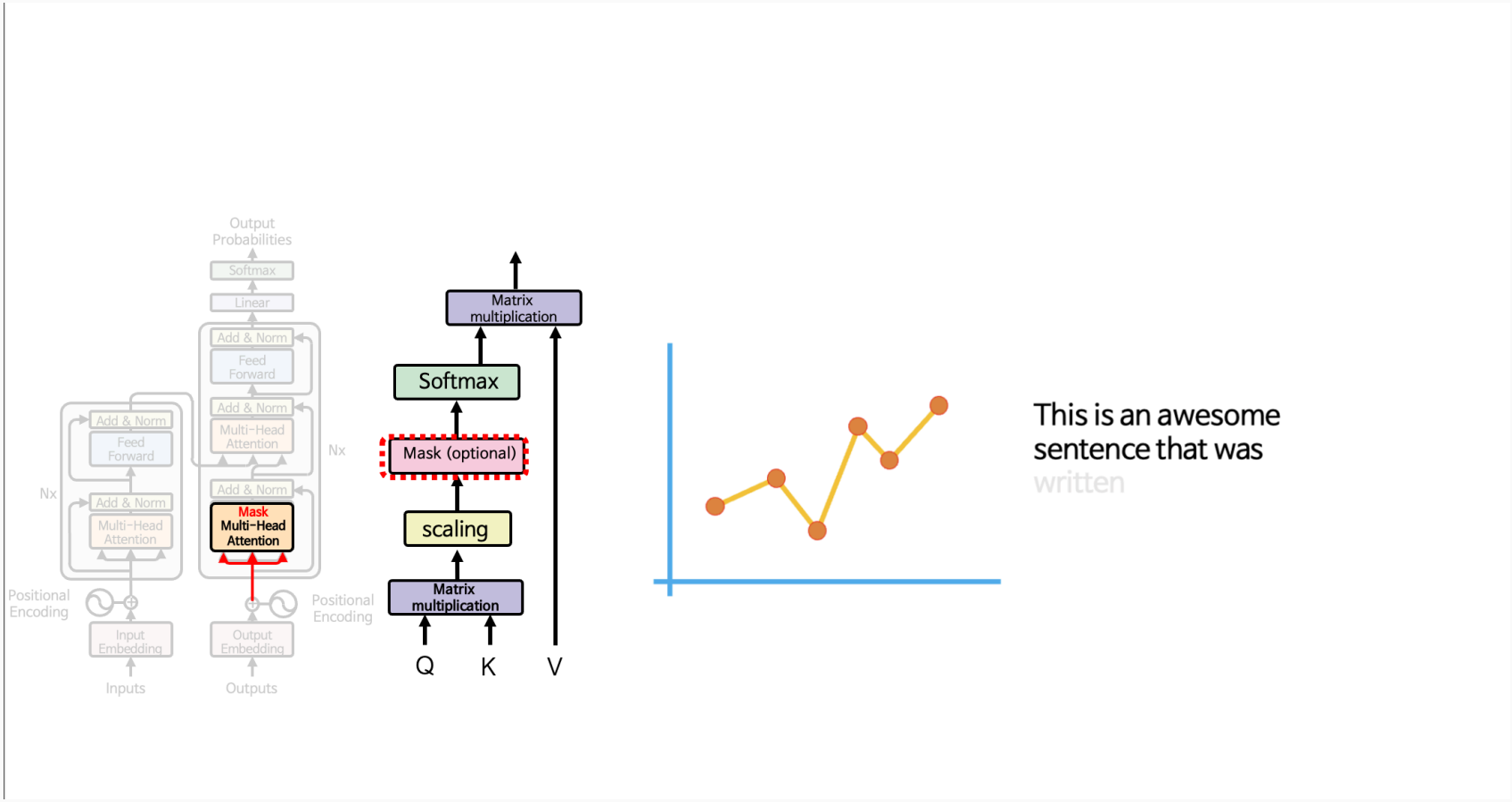


	⟨sos⟩	I	am
⟨sos⟩	-0.33	0.01	-0.63
I	1.37	-1.04	1.02
am	0.00	0.11	0.20

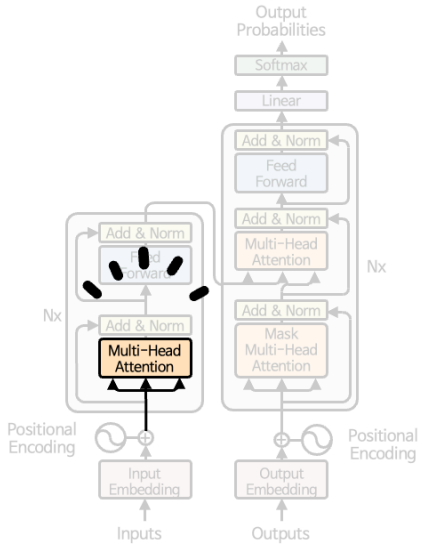
$\div \sqrt{6} =$

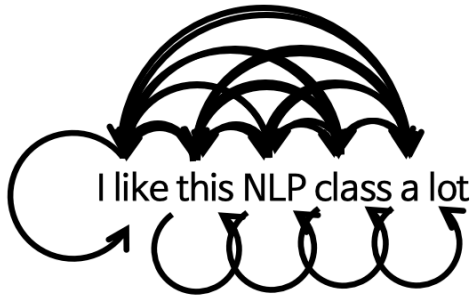
-0.13	0.00	-0.26
0.56	-0.43	0.42
0.00	0.04	0.08

The goal of the transformer *decoder* is to generate the output word sequence, one token at a time (recall: language modeling).

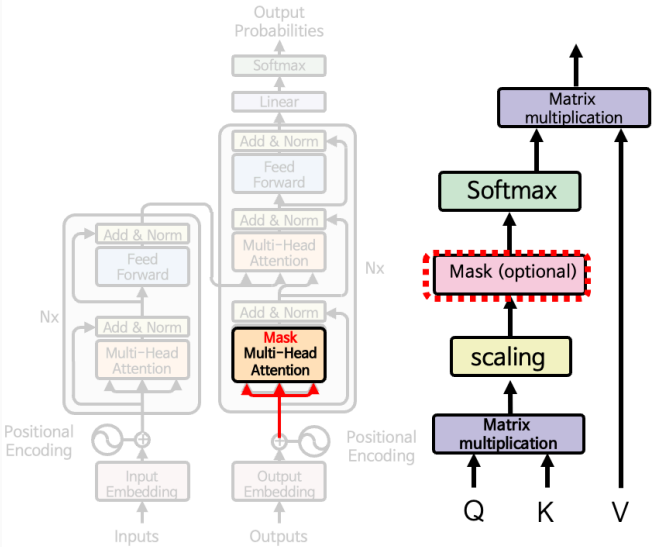


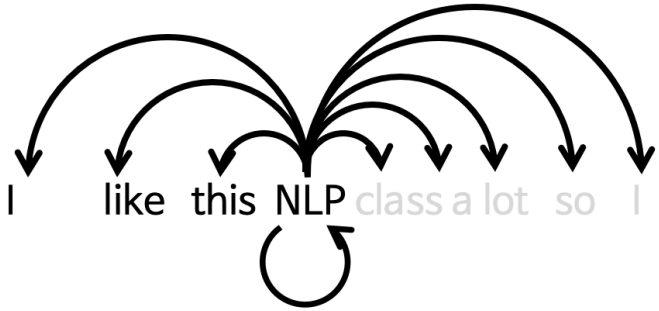
While the encoder needs to consider all tokens in the input sentence to understand the full meaning,



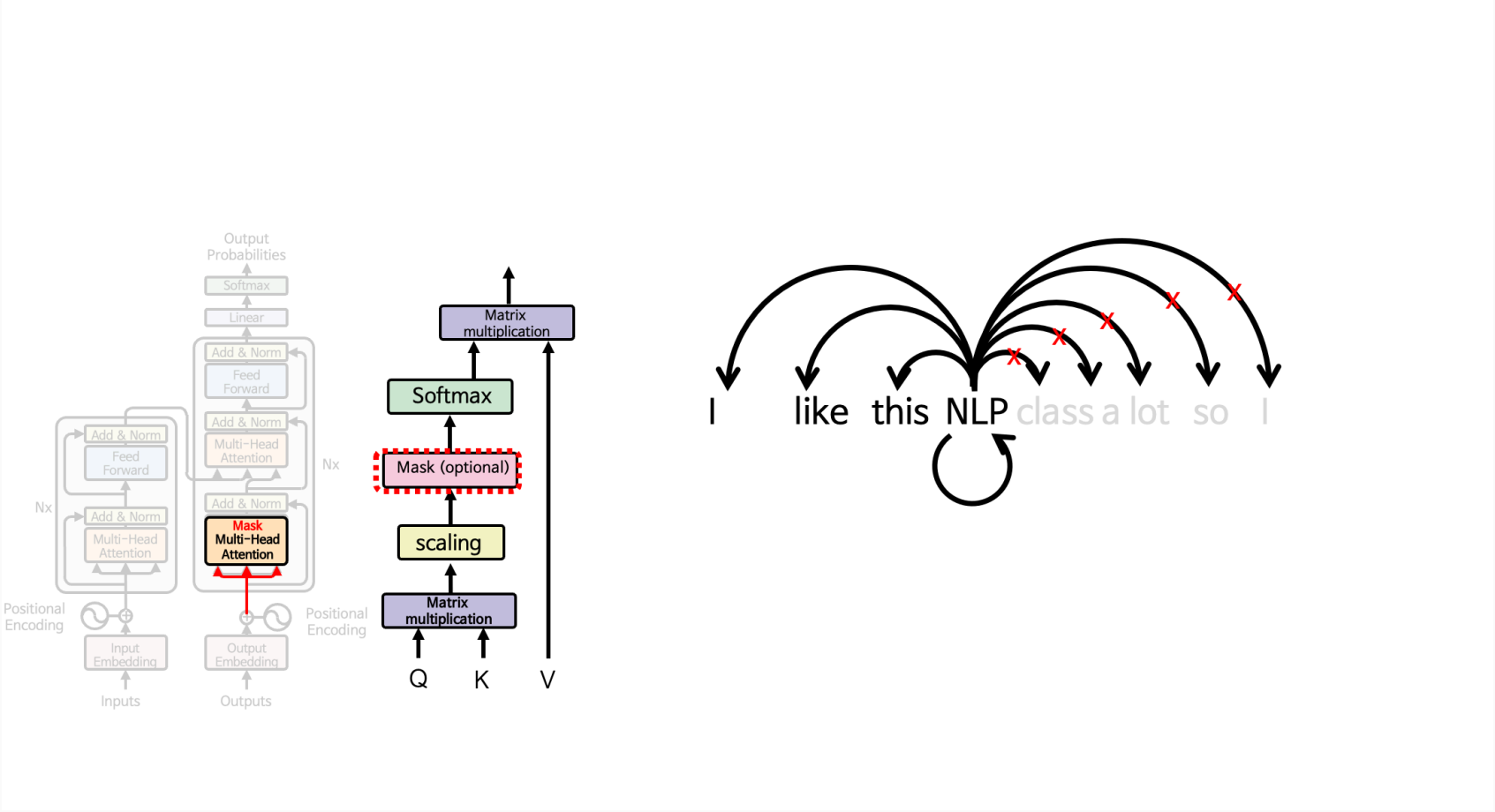


the decoder generates output **one word at a time**.

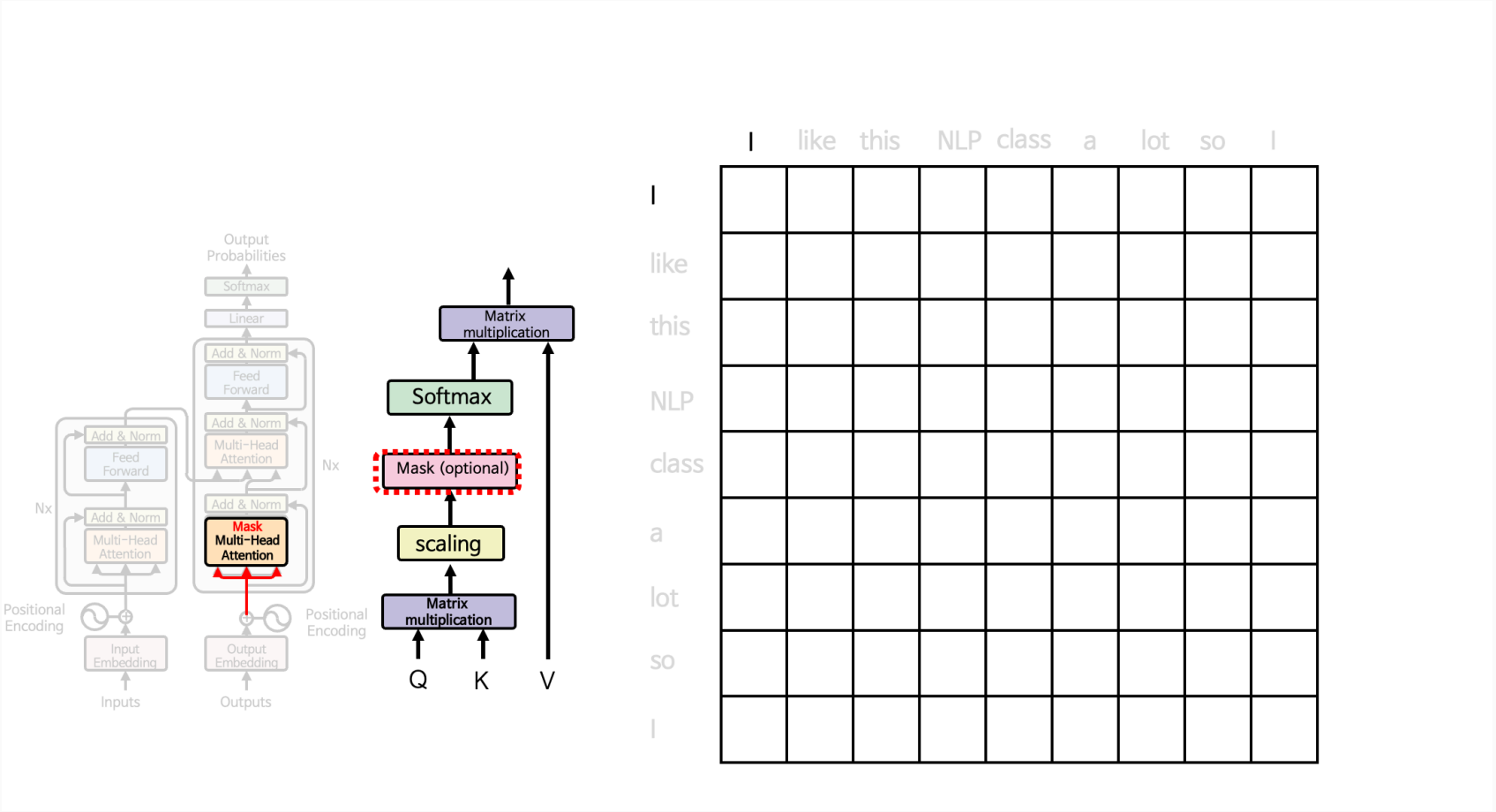




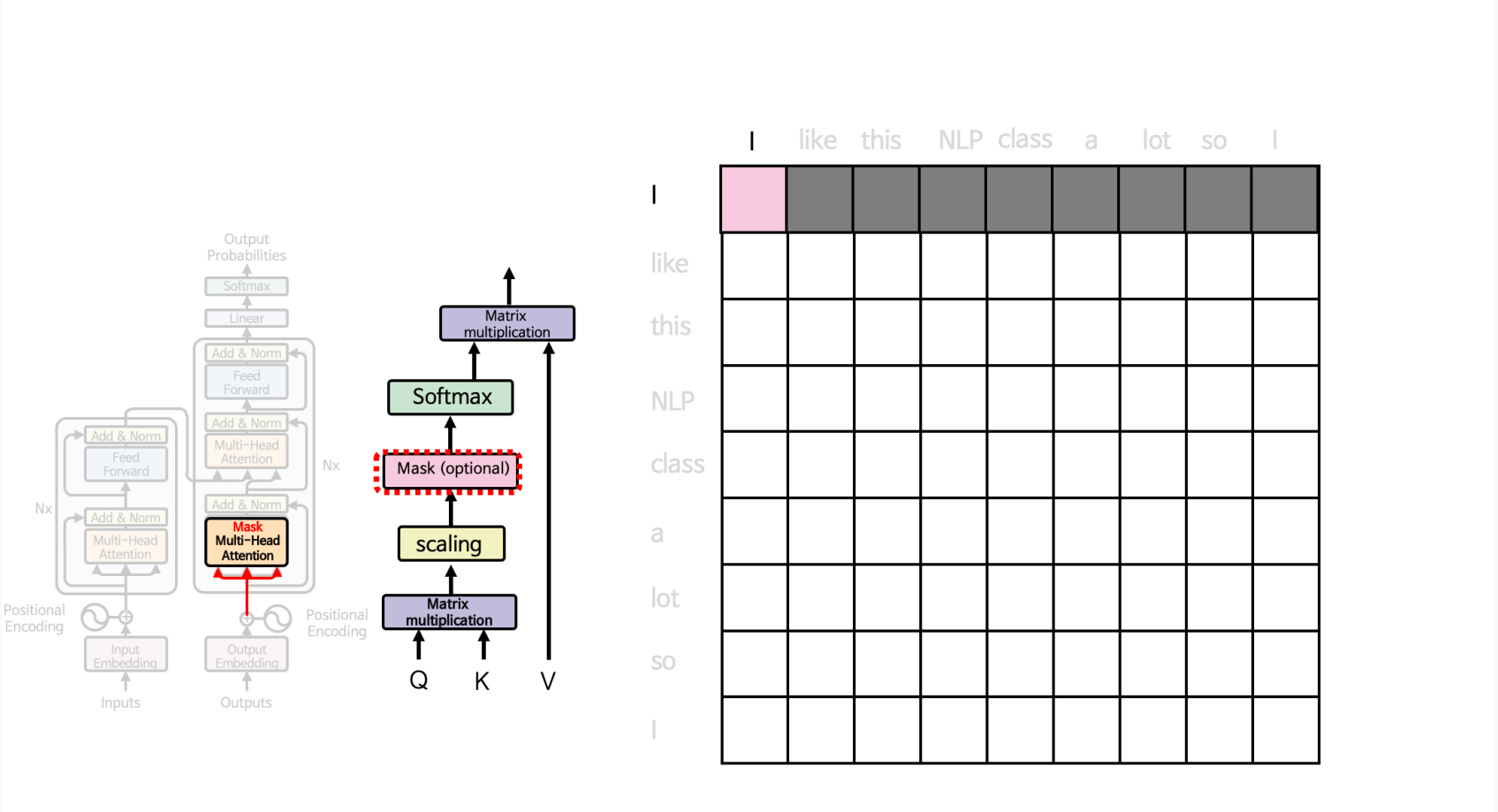
Therefore, it’s natural that the decoder should NOT attend to words that haven’t been generated yet.

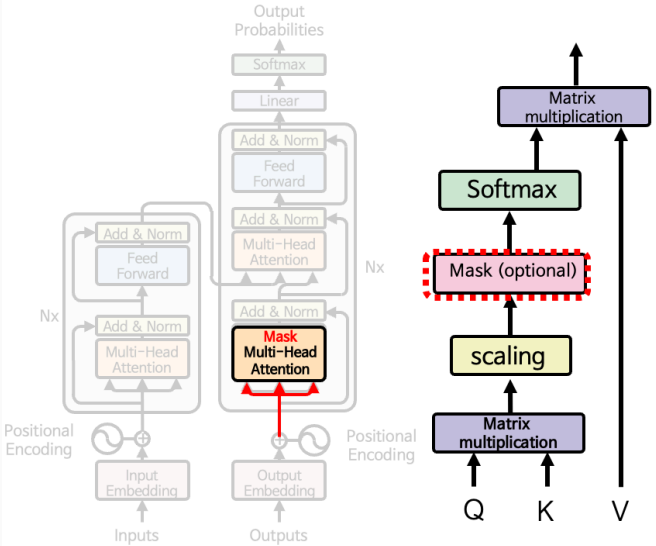


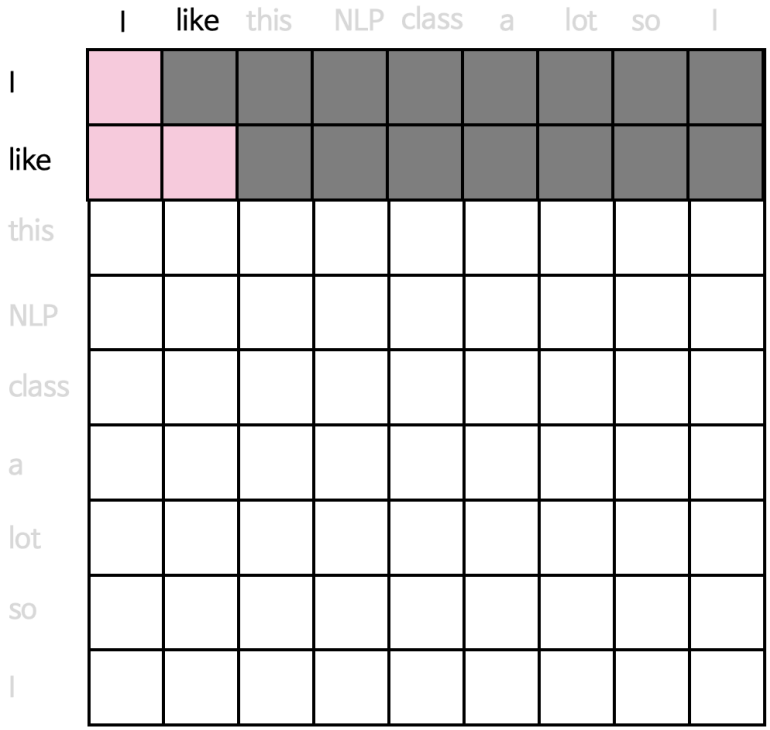
To reflect this characteristic, the decoder applies a masking mechanism during training.

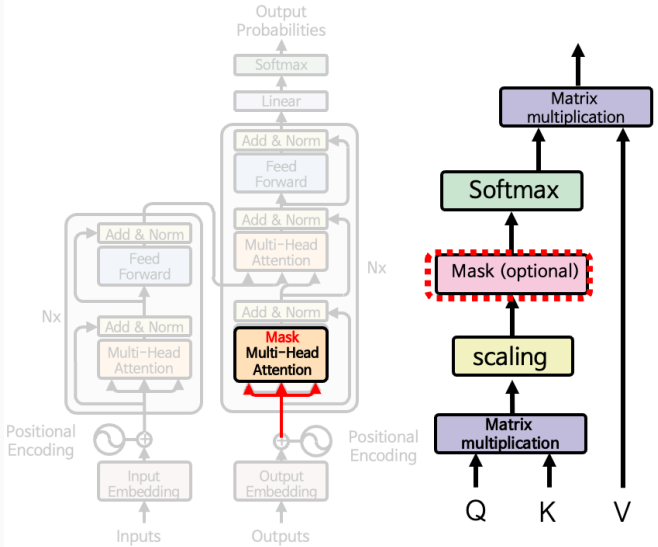


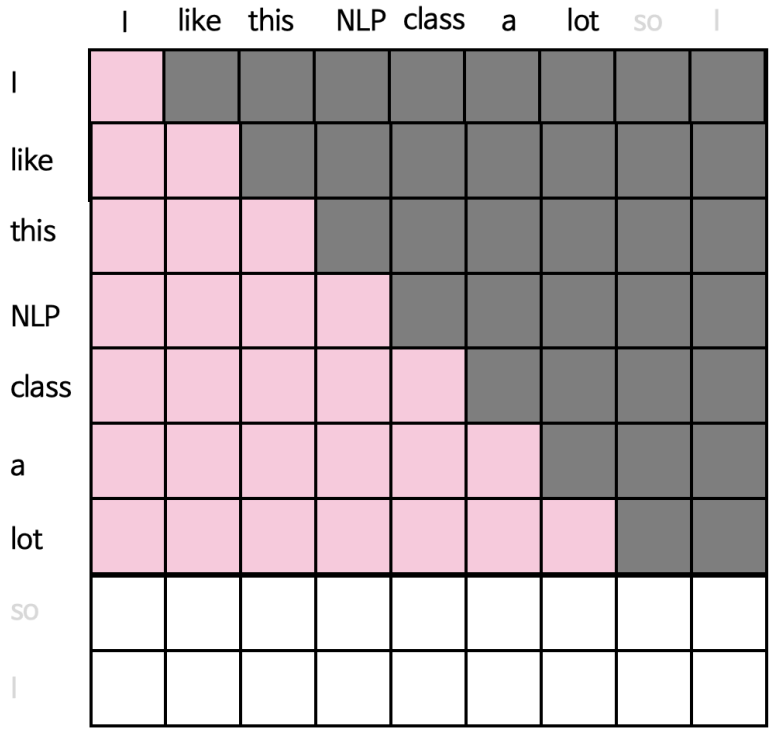
The key idea is to hide future tokens so they do not affect the current prediction.

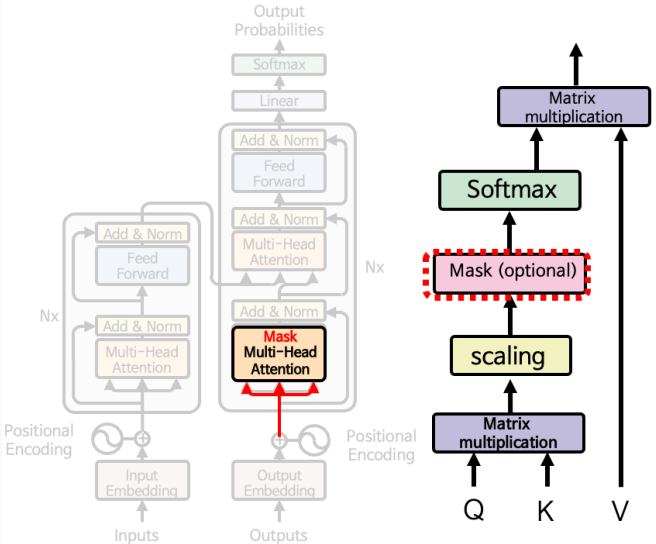


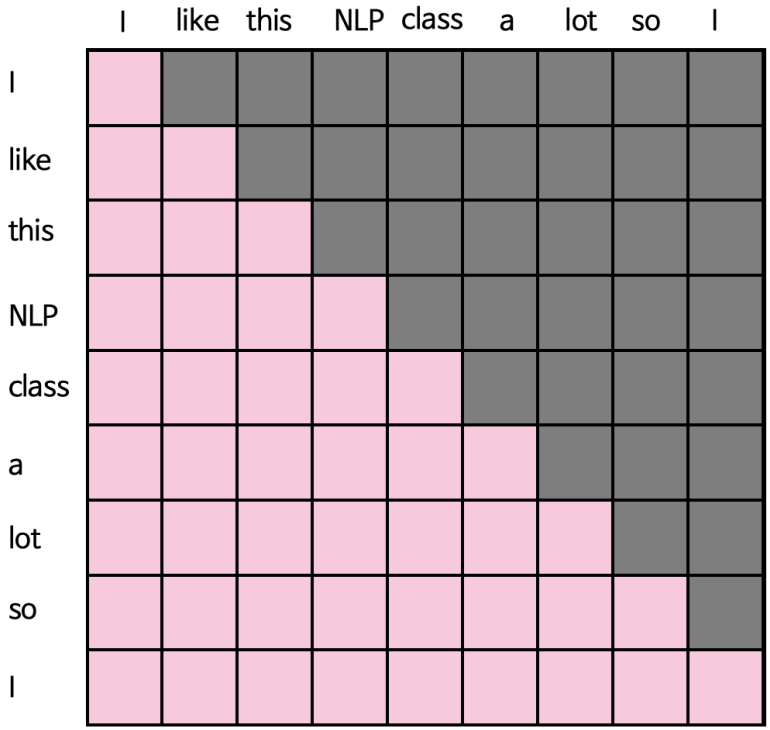

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Mask Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs
Matrix multiplication
Softmax
Mask (optional)
scaling
Matrix multiplication
Q
K
V


I
like
this
NLP
class
a
lot
so
I
I
like
this
NLP
class
a
lot
so
I

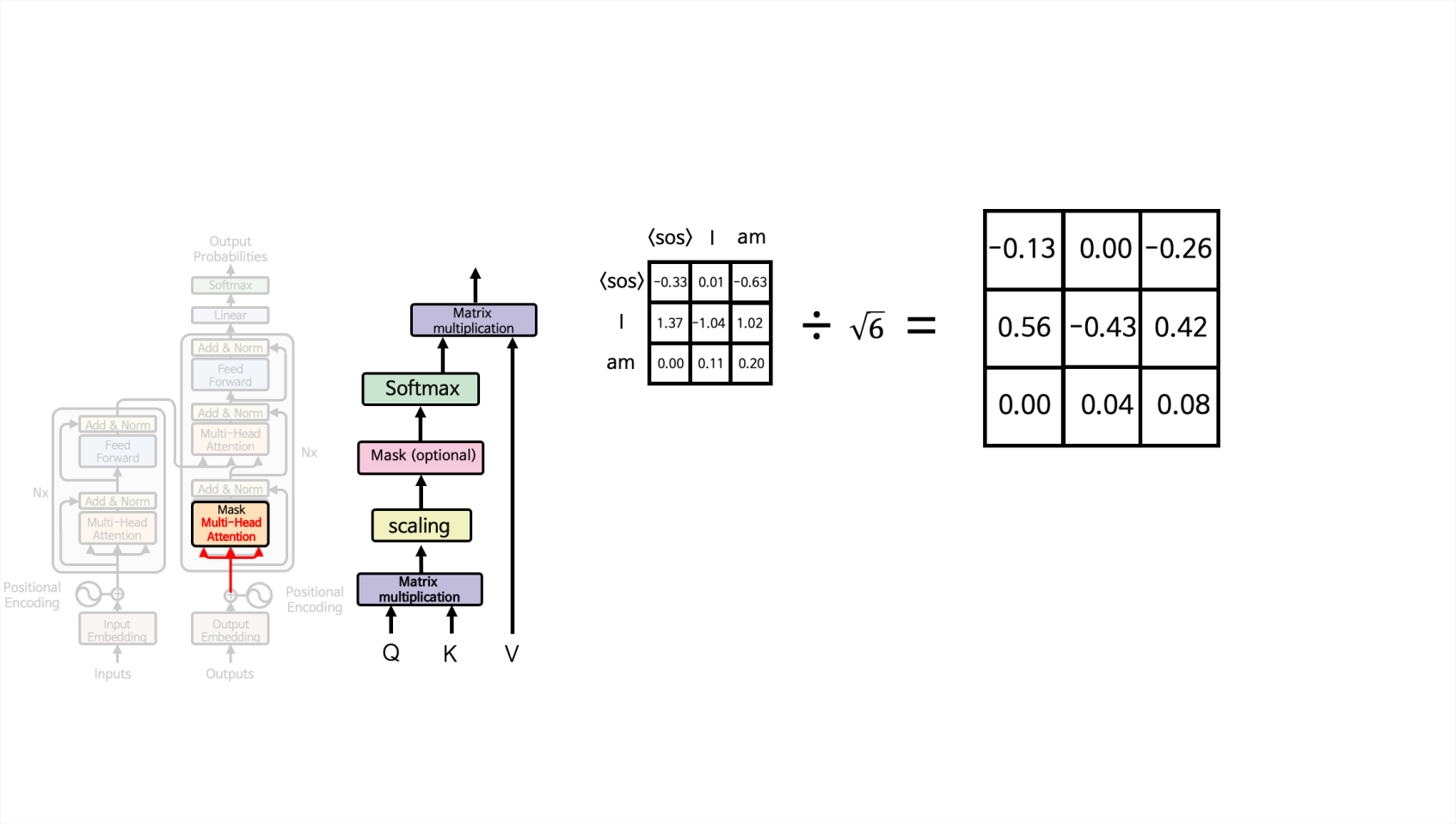

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Mask Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs
Matrix multiplication
Softmax
Mask (optional)
scaling
Matrix multiplication
Q
K
V


I
like
this
NLP
class
a
lot
so
I

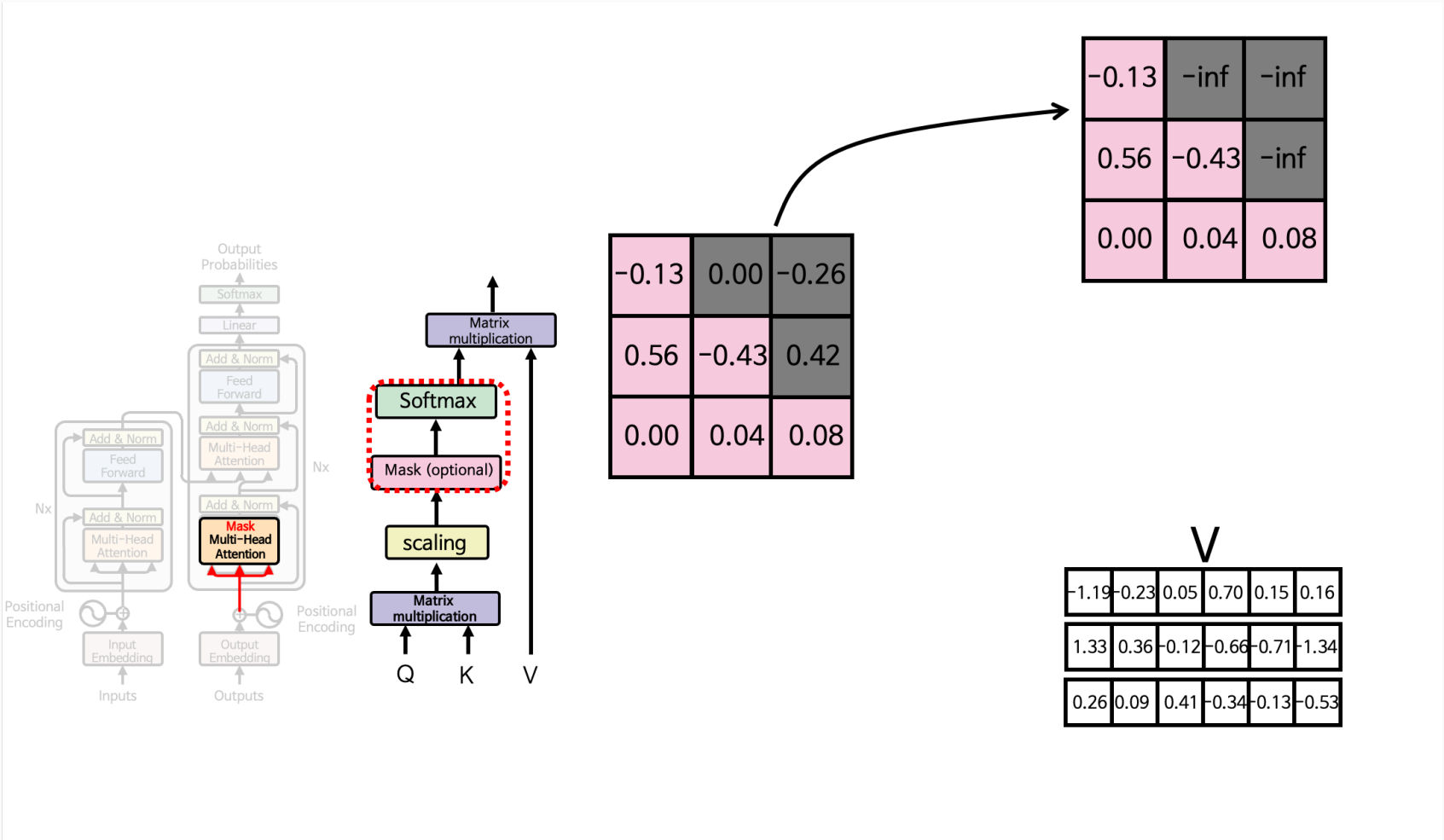

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Mask Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs
Matrix multiplication
Softmax
Mask (optional)
scaling
Matrix multiplication
Q
K
V


I
like
this
NLP
class
a
lot
so
I

When this masking algorithm is applied to the attention matrix, we get:

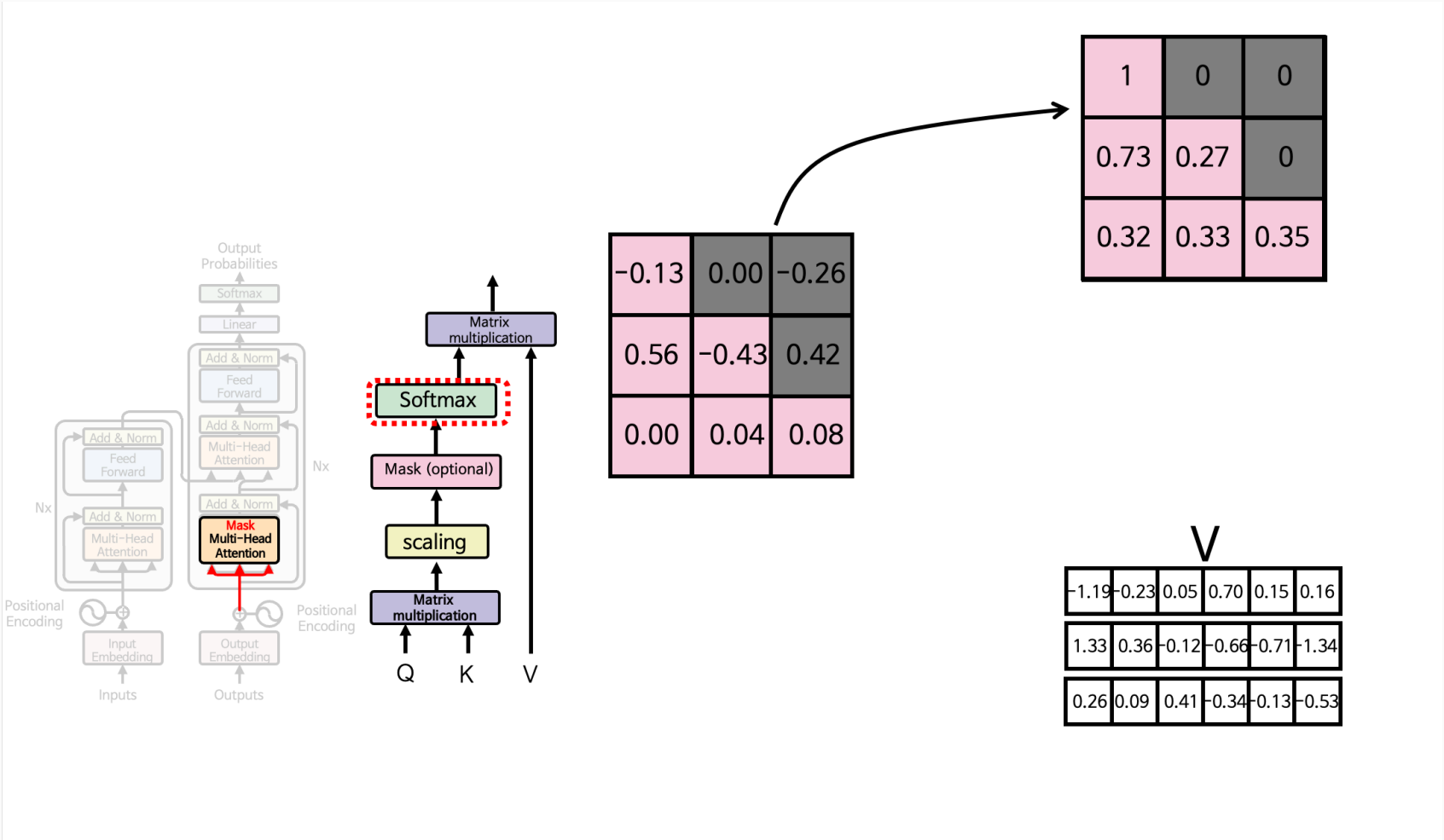


We add *–inf* to the masked positions because, after passing through the softmax layer, *–inf* becomes 0, effectively eliminating attention to those positions.

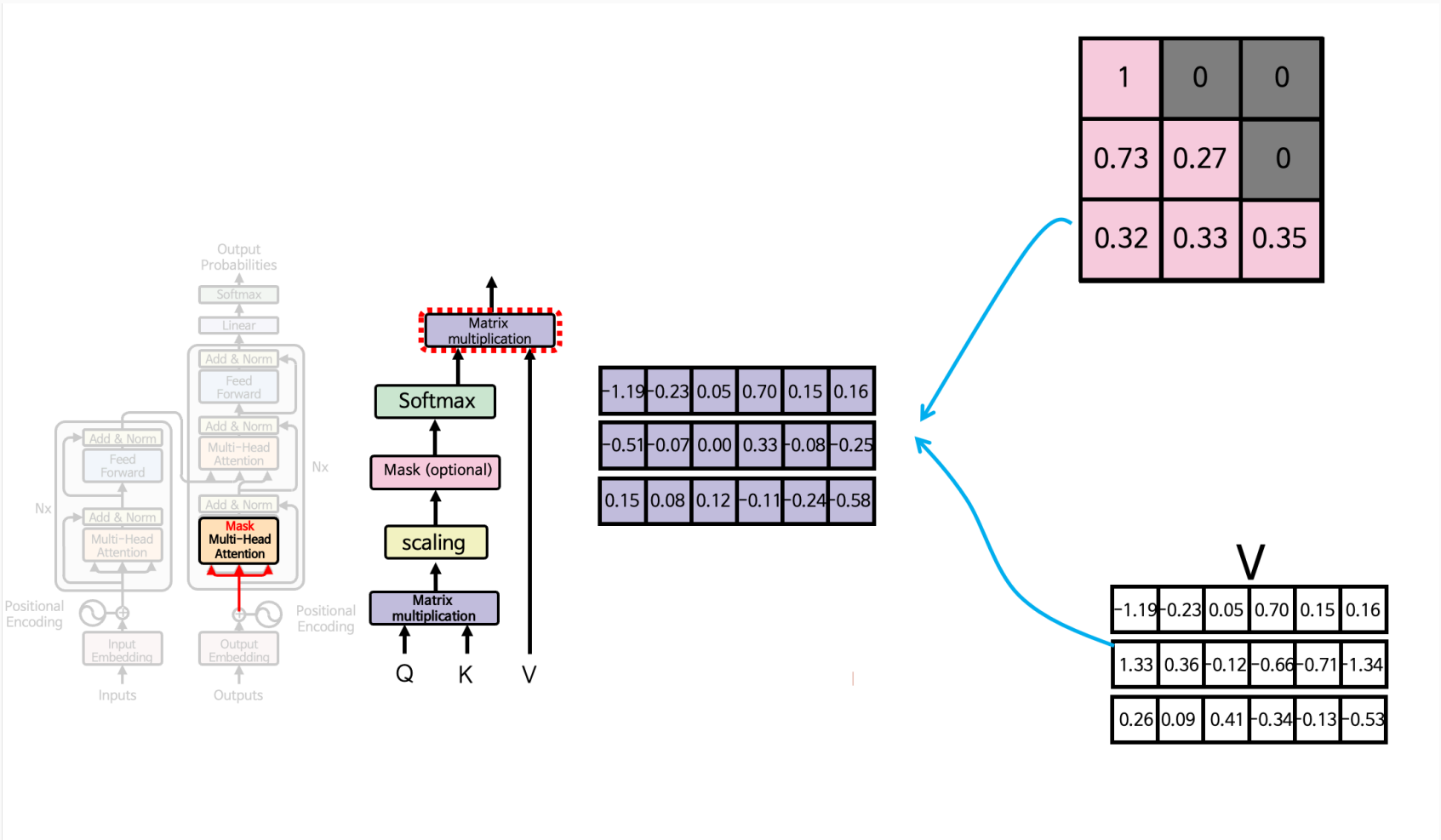


[1] https://www.youtube.com/shorts/SrJN_hpiuAs

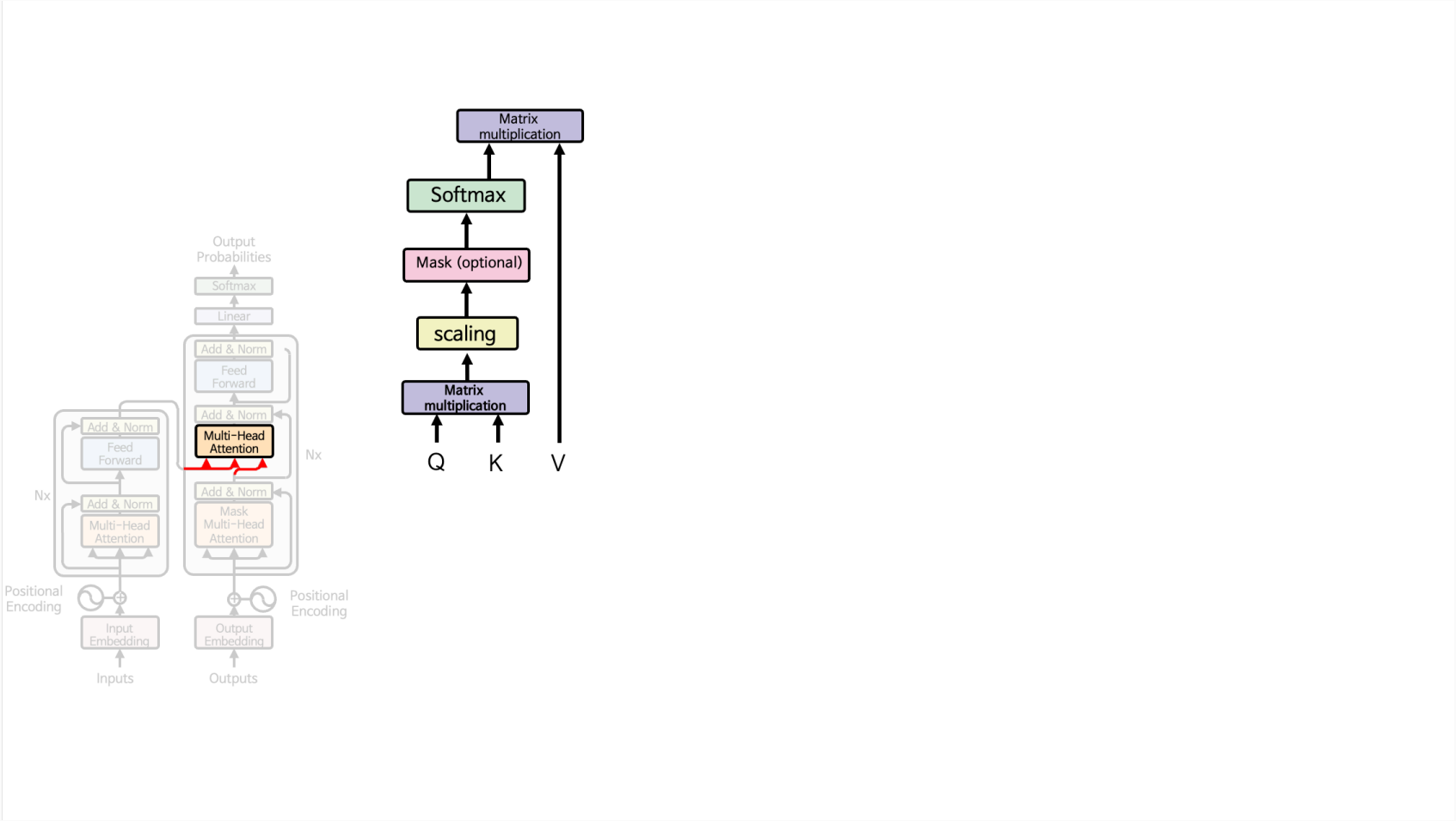
Feeding this matrix into the softmax layer gives us:



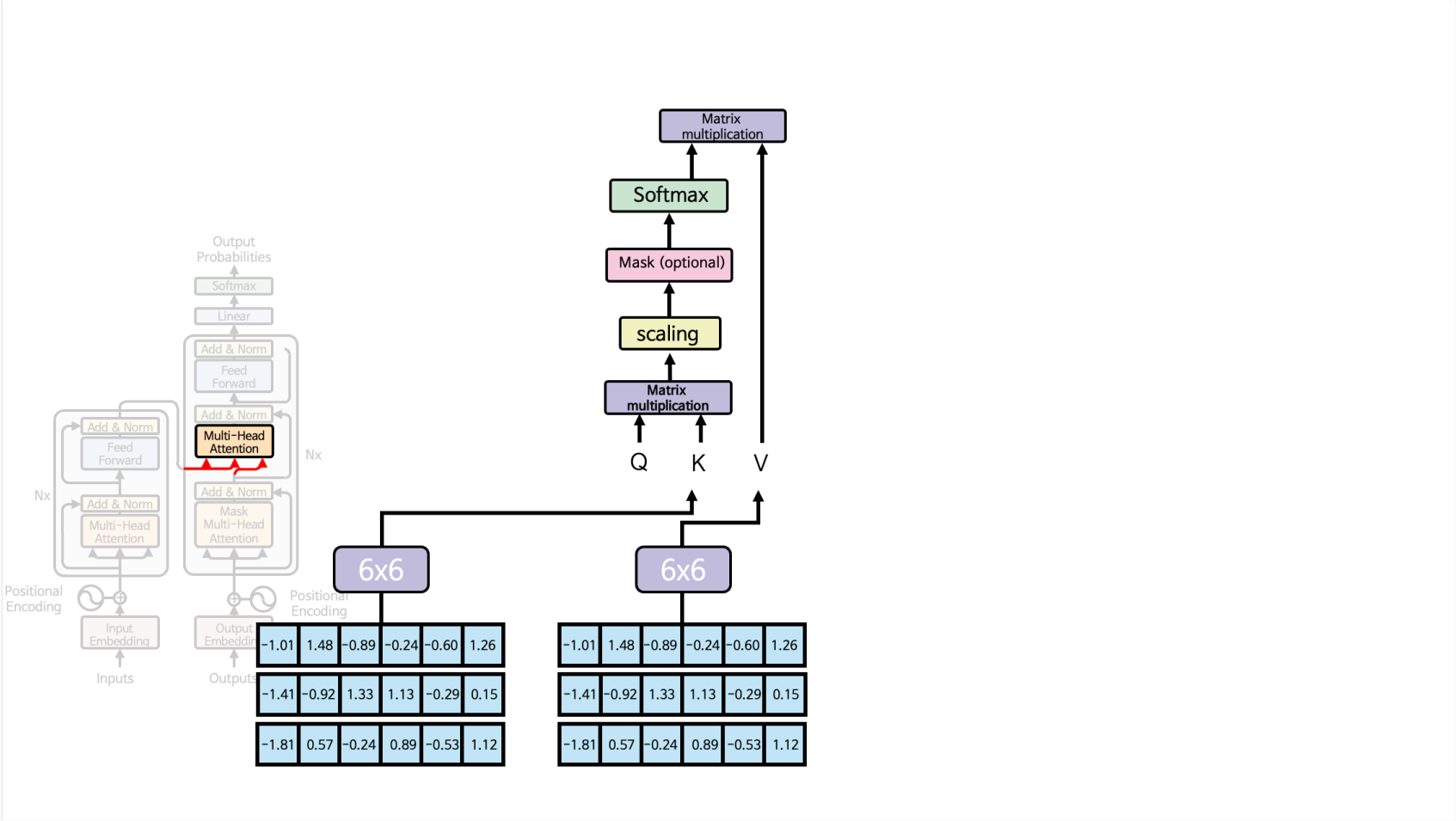
Then we multiply the two matrices as follows:



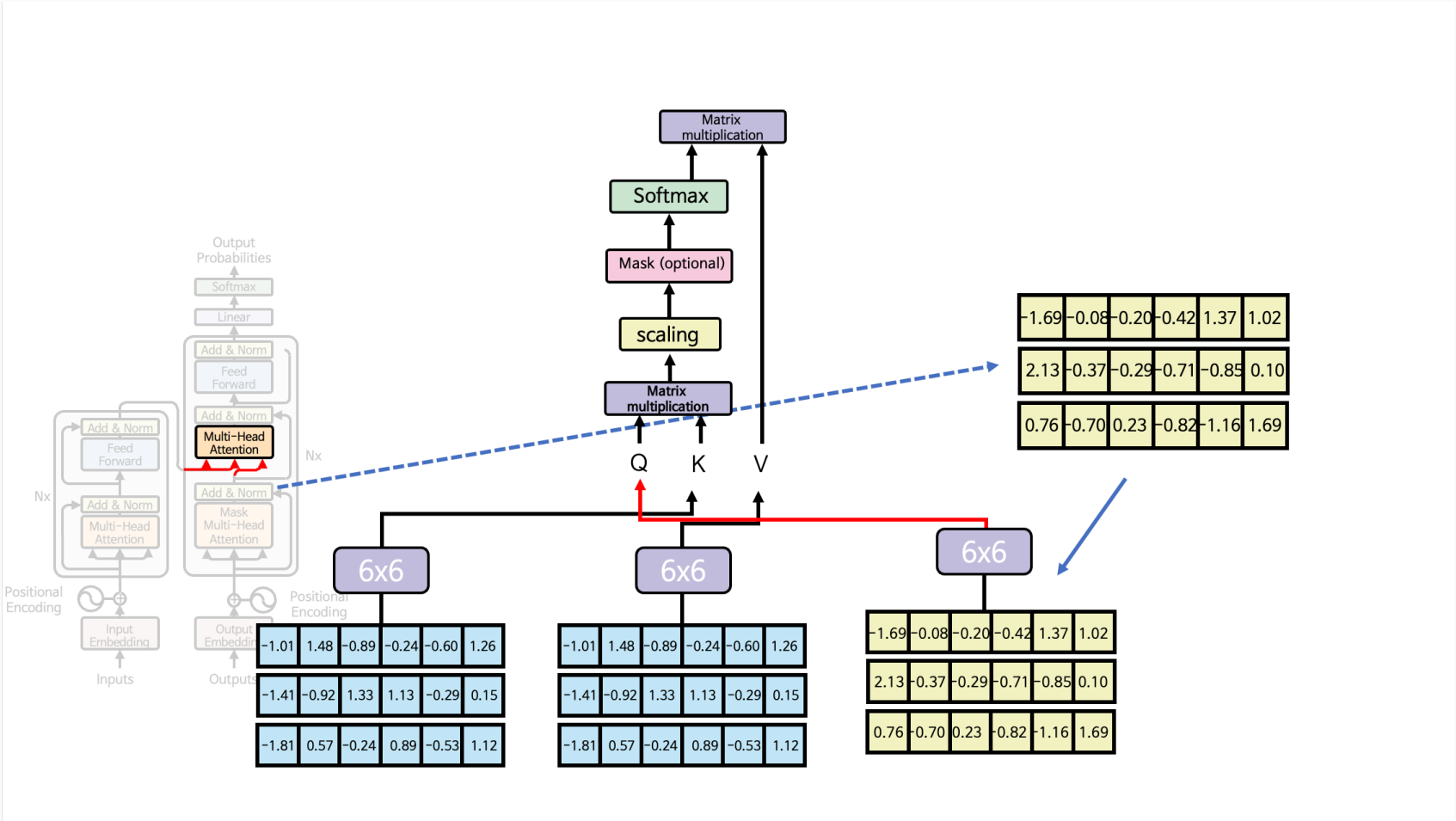
The decoder's **second** multi-head attention operates the same way as the encoder's, except for the inputs.



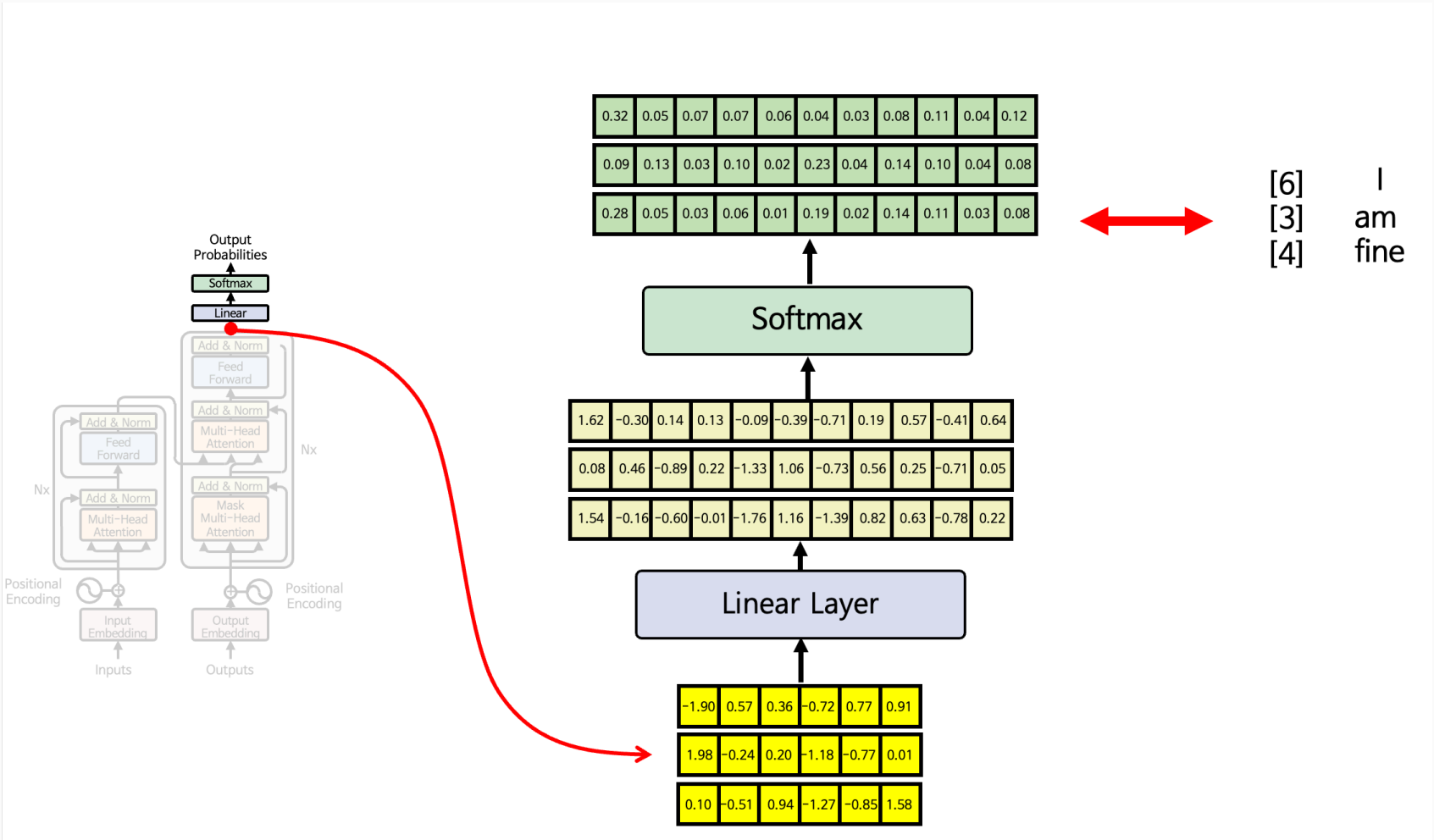
Here, the values of K and V are derived from the **encoder's final output**, multiplied by a 6×6 matrix.

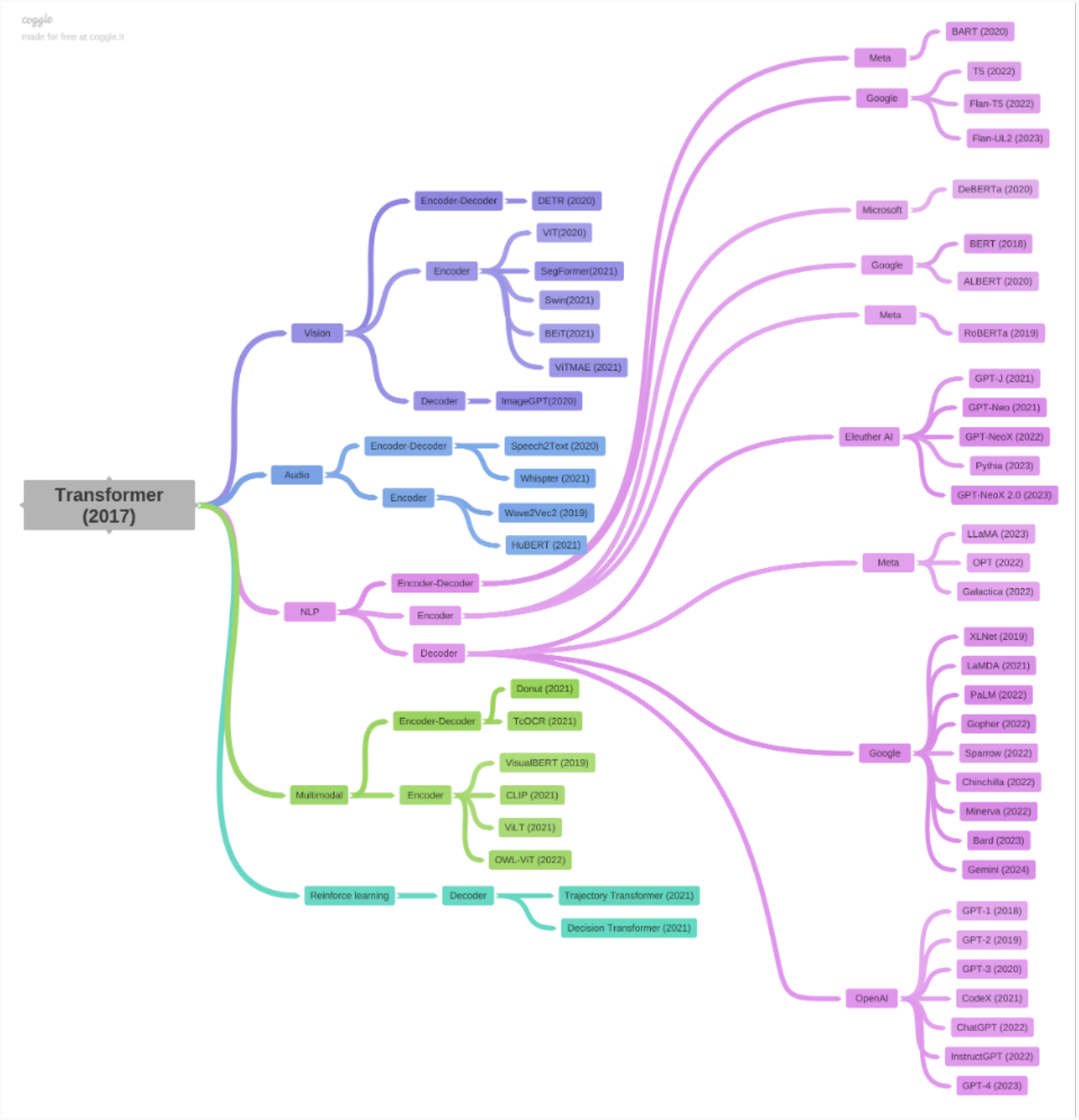


The value of Q comes from the output of the decoder's previous Add & Norm layer. In other words, the decoder determines which parts of the encoder's output (K, V) to attend to, based on the context it has generated so far (Q)-similar to attention in RNNs.



Using a loss function (e.g., cross-entropy) and backpropagation, the model updates all weight parameters across every layer — this is the learning process of the Transformer.




coggle
made for free at coggle.it
Transformer (2017)
Vision
Encoder-Decoder
DETR (2020)
Encoder
ViT(2020)
SegFormer(2021)
Swin(2021)
BEiT(2021)
ViTMAE (2021)
Decoder
ImageGPT(2020)
Audio
Encoder-Decoder
Speech2Text (2020)
Whispter (2021)
Encoder
Wave2Vec2 (2019)
HuBERT (2021)
NLP
Encoder-Decoder
Encoder
Decoder
Multimodal
Encoder-Decoder
Donut (2021)
TcOCR (2021)
Encoder
VisualBERT (2019)
CLIP (2021)
ViLT (2021)
OWL-ViT (2022)
Reinforce learning
Decoder
Trajectory Transformer (2021)
Decision Transformer (2021)
Meta
BART (2020)
Google
T5 (2022)
Flan-T5 (2022)
Flan-UL2 (2023)
Microsoft
DeBERTa (2020)
Google
BERT (2018)
ALBERT (2020)
Meta
RoBERTa (2019)
Eleuther AI
GPT-J (2021)
GPT-Neo (2021)
GPT-NeoX (2022)
Pythia (2023)
GPT-NeoX 2.0 (2023)
Meta
LLaMA (2023)
OPT (2022)
Galactica (2022)
Google
XLNet (2019)
LaMDA (2021)
PaLM (2022)
Gopher (2022)
Sparrow (2022)
Chinchilla (2022)
Minerva (2022)
Bard (2023)
Gemini (2024)
OpenAI
GPT-1 (2018)
GPT-2 (2019)
GPT-3 (2020)
CodeX (2021)
ChatGPT (2022)
InstructGPT (2022)
GPT-4 (2023)

Different transformers

Transformer types

- **Encoder-only models** use only the encoder stack to **understand text**.
  - Bidirectional attention (see both left and right context)
  - Tasks: text classification, NER, POS tagging
  - Examples: **BERT**, RoBERTa, ALBERT

Transformer types

- **Decoder-only models** use only the decoder stack to **generate text**.
  - Left-to-right (causal) attention — cannot see future tokens
  - Tasks: text generation, dialogue, code completion
  - Examples: **GPT**, LLaMA, Gemini, Claude

Transformer types

- **Encoder–Decoder models** combine both for **sequence-to-sequence tasks**.
  - Encoder encodes the input; decoder generates the output
  - Tasks: translation, summarization
  - Examples: **T5**, BART, mBART

LLMs in 2025

Sharing Thoughts

- Take a few minutes to share your thoughts or reflections on today’s session.

Sharing Thoughts

- Take a few minutes to share your thoughts or reflections on today’s session.
- Contribute one or two points to the **shared slide deck** (click here, or a link in the course website) - this could be:

Sharing Thoughts

- Take a few minutes to share your thoughts or reflections on today’s session.
- Contribute one or two points to the **shared slide deck** (click here, or a link in the course website) - this could be:
    - Something new you learned about LLMs or Transformers

Sharing Thoughts

- Take a few minutes to share your thoughts or reflections on today’s session.
- Contribute one or two points to the **shared slide deck** (click here, or a link in the course website) - this could be:
  - Something new you learned about LLMs or Transformers
  - A question or idea you’d like to explore further

Sharing Thoughts

- Take a few minutes to share your thoughts or reflections on today’s session.
- Contribute one or two points to the **shared slide deck** (click here, or a link in the course website) - this could be:
  - Something new you learned about LLMs or Transformers
  - A question or idea you’d like to explore further
  - An observation about how these models relate to real-world NLP tasks

Wrap up

Wrap-up

- We explored recent advances in large language models (LLMs) as of 2025.

Wrap-up

- We explored recent advances in large language models (LLMs) as of 2025.
- In the lab session, we will further experiment with these models using *Ollama*, an open-source platform for running LLMs locally.