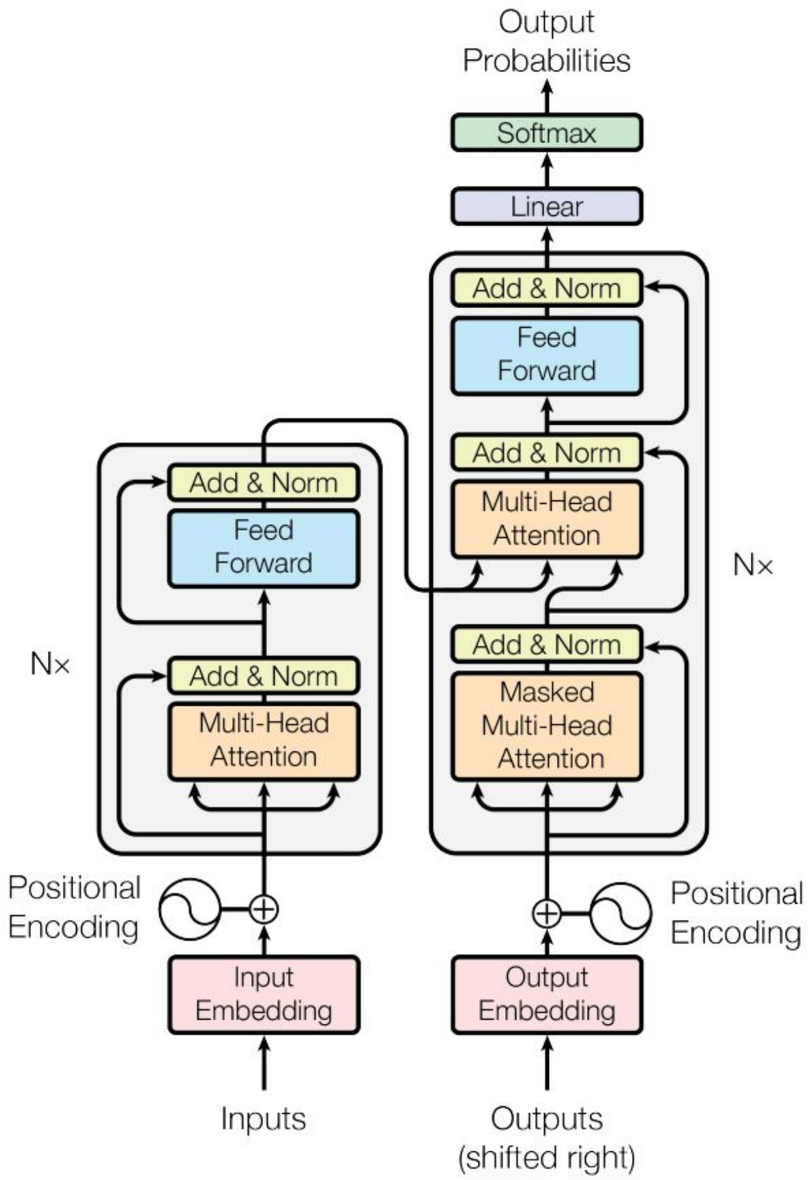
Transformers Intro

CREDITS

1. https://www.youtube.com/watch?v=UPtG_38Oq8o
2. https://www.youtube.com/watch?v=wjZofJX0v4M&t=1224s
3. https://jalammar.github.io/illustrated-transformer/
4. https://www.youtube.com/watch?v=4Bdc55j80l8&t=8s
5. **Dr. Greg Loughnane, Content**

TRANSFORMERS

- This is ChatGPT



TRANSFORMERS



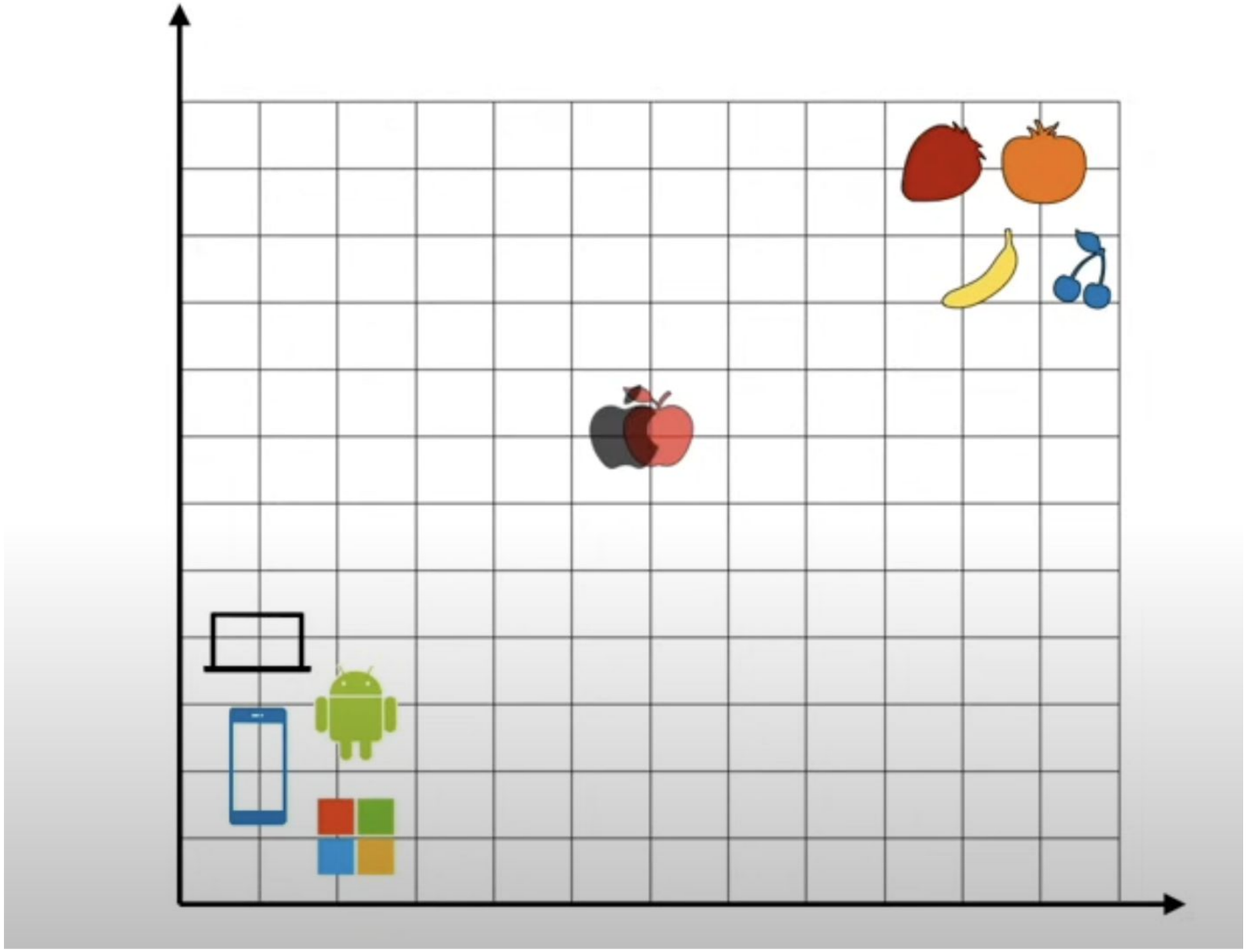


Abstract

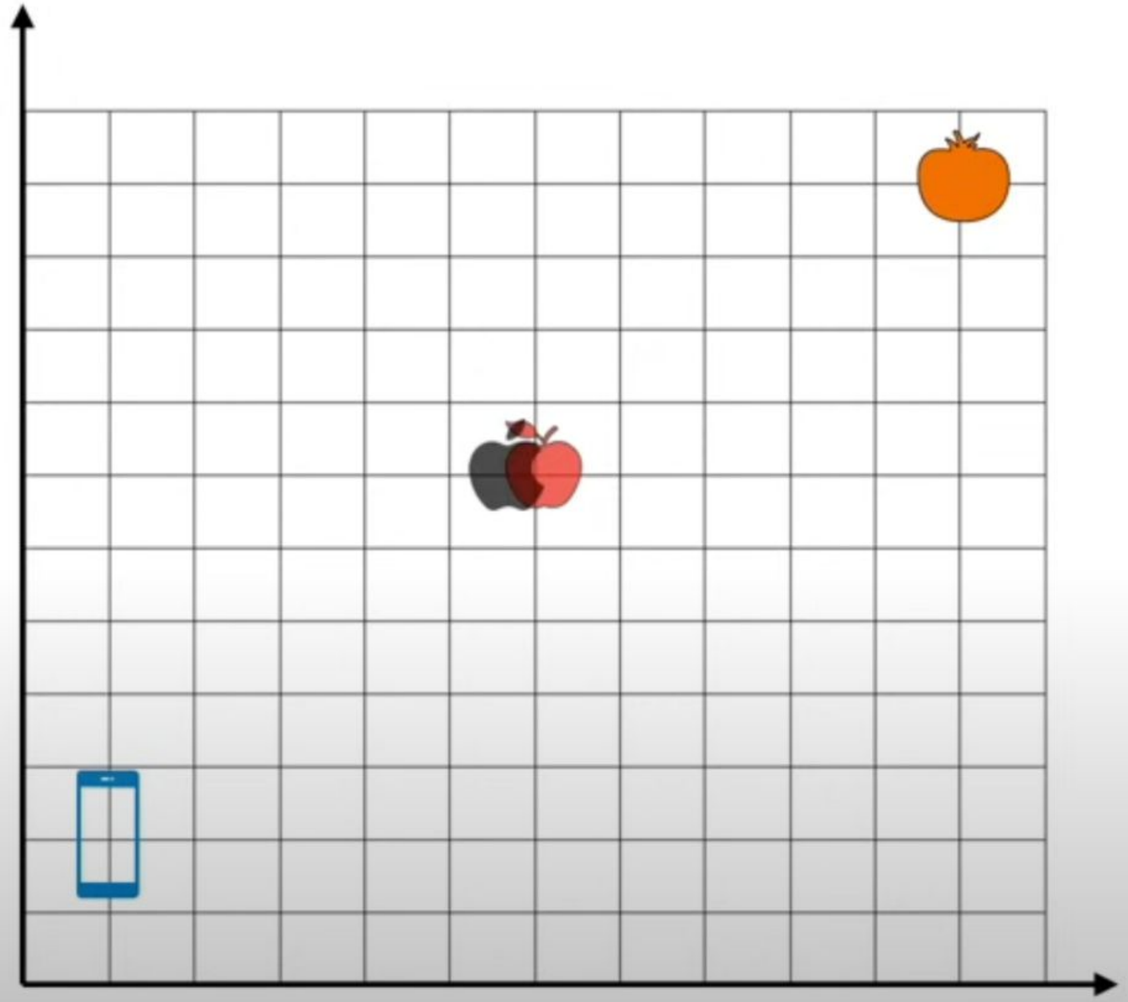
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Embeddings

Embeddings



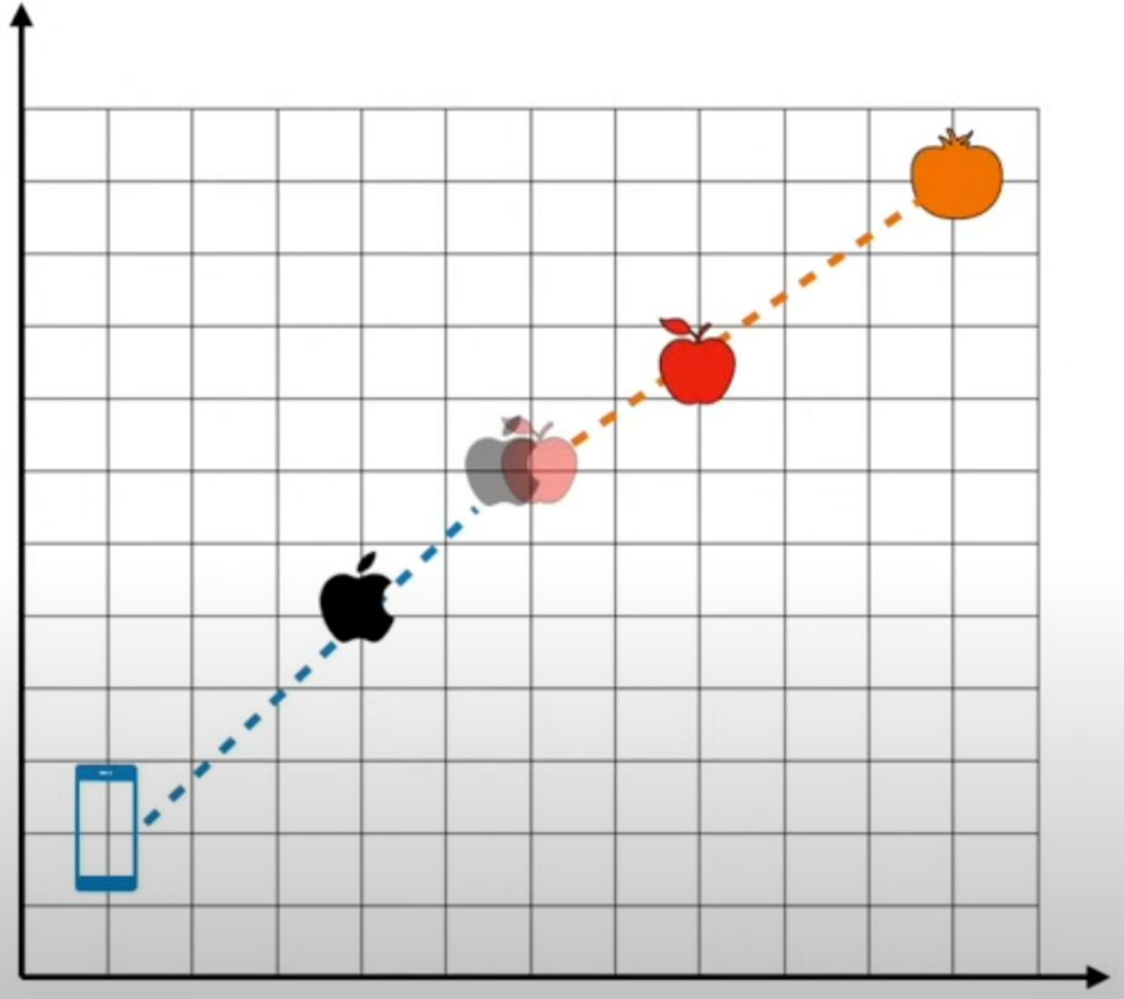
Words pulling words

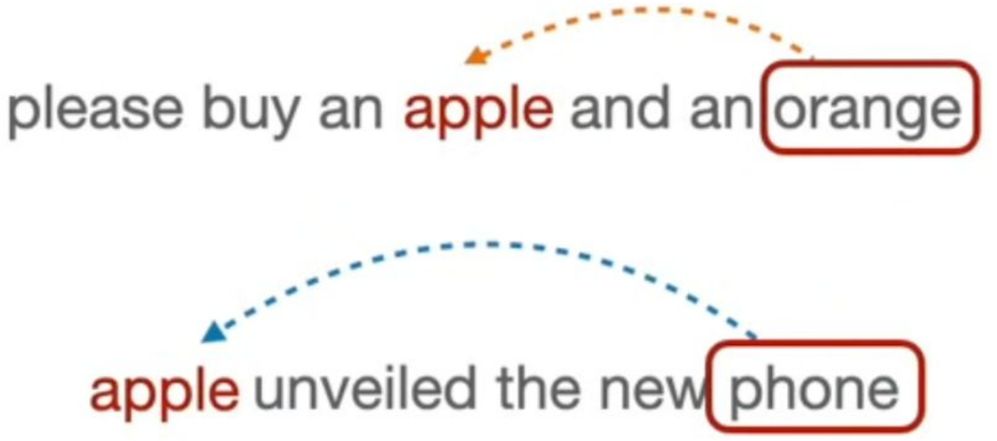


please buy an apple and an orange

apple unveiled the new phone

Words pulling words

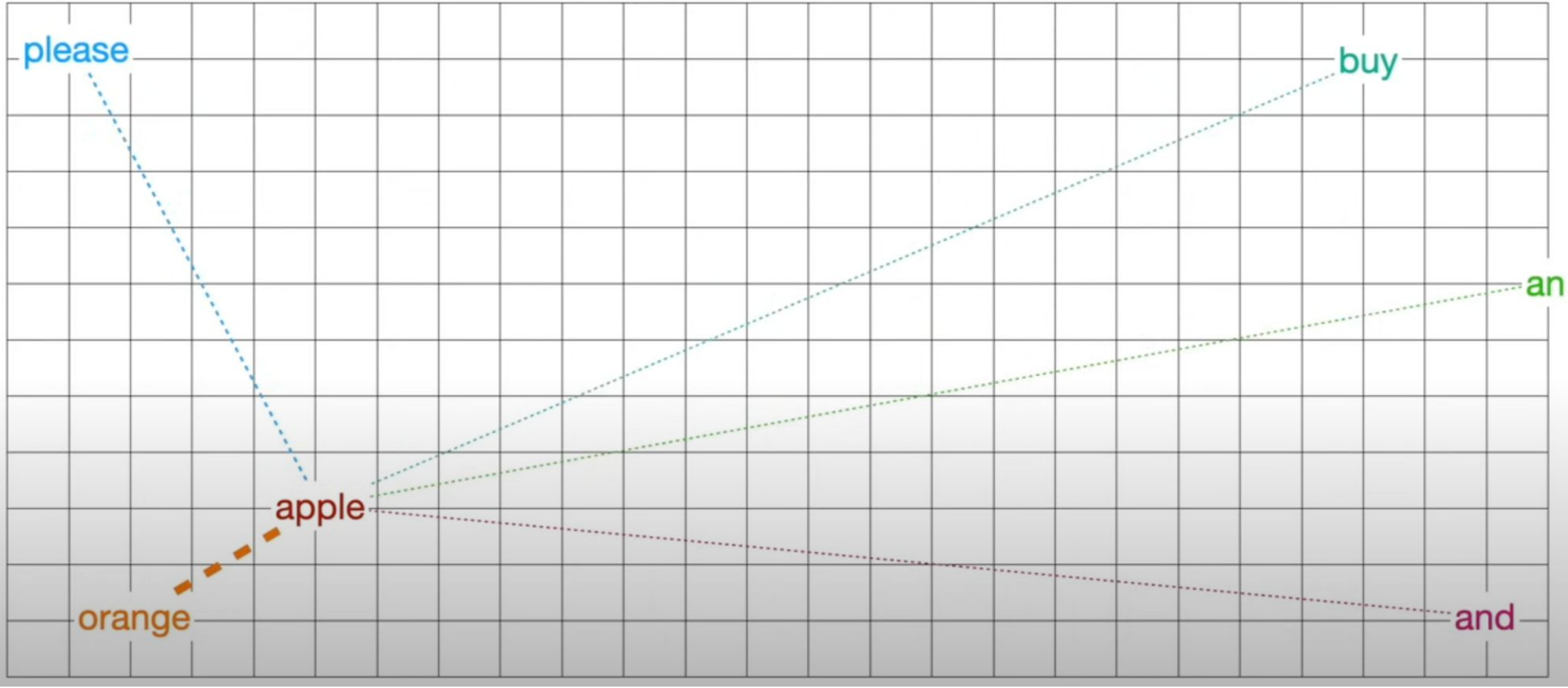




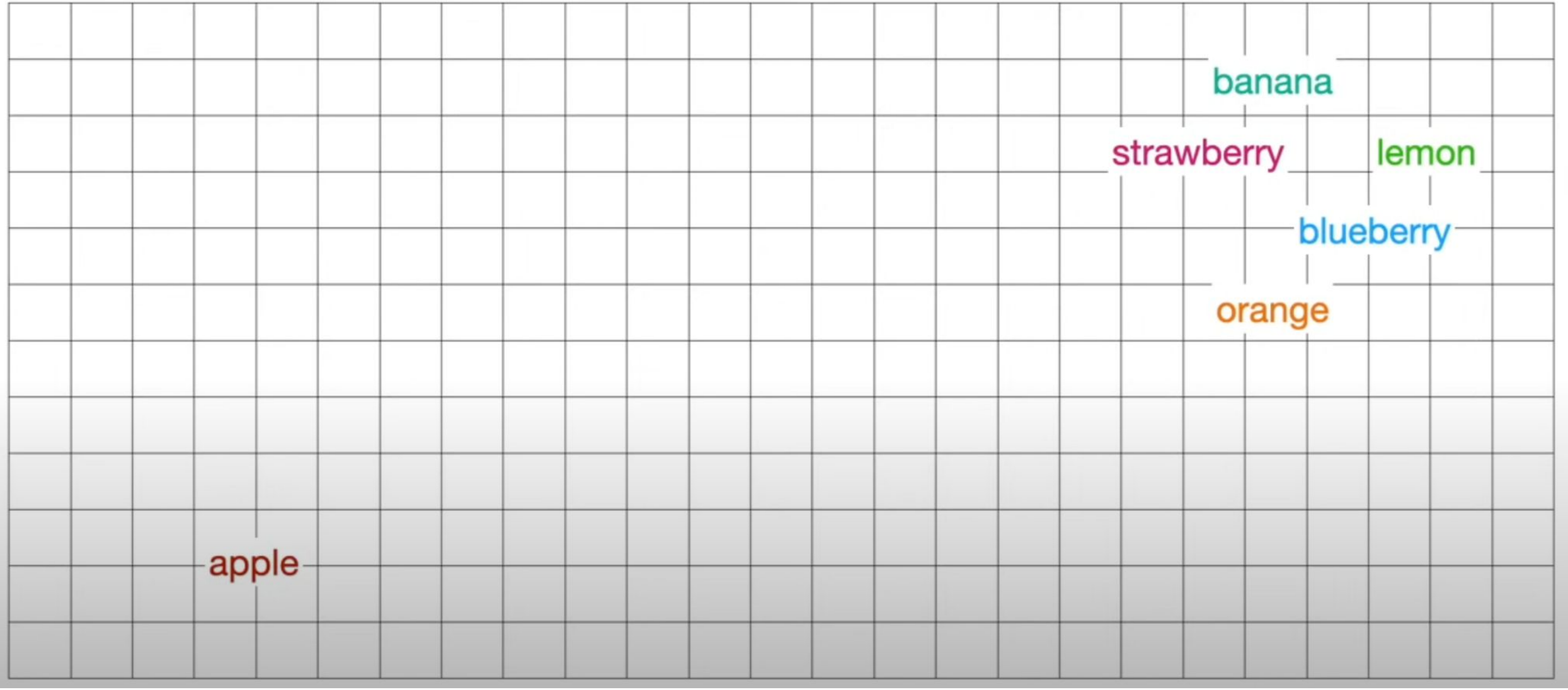
All words pull each other



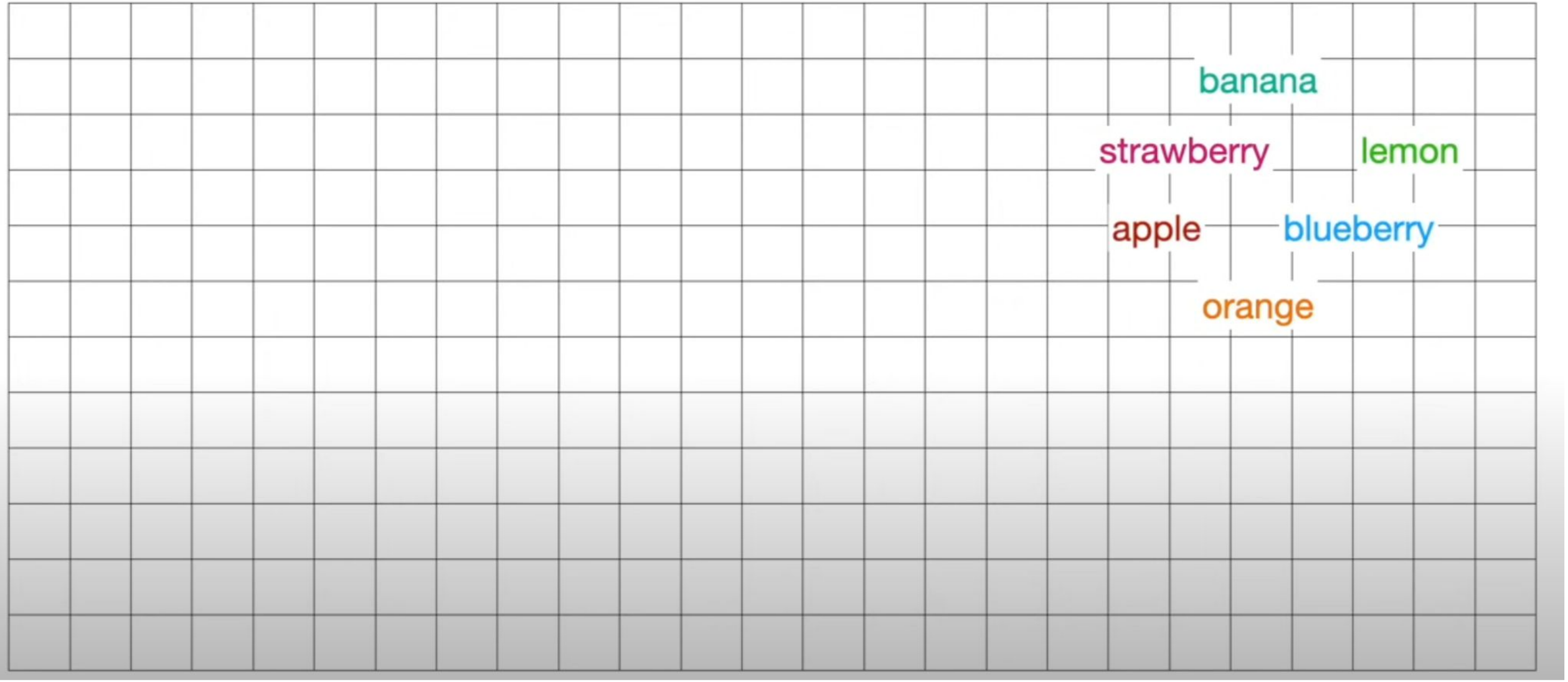
All words pull each other



Context pulls

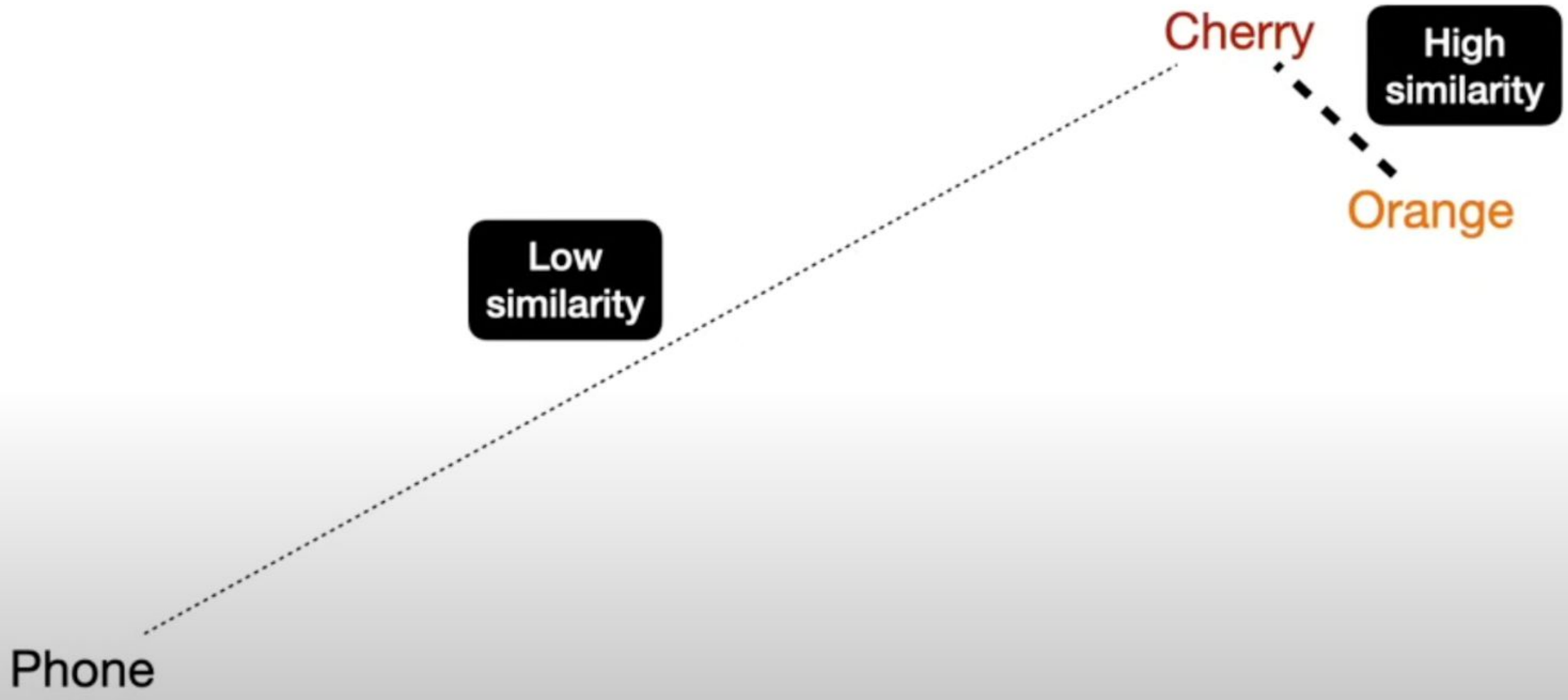


Context pulls

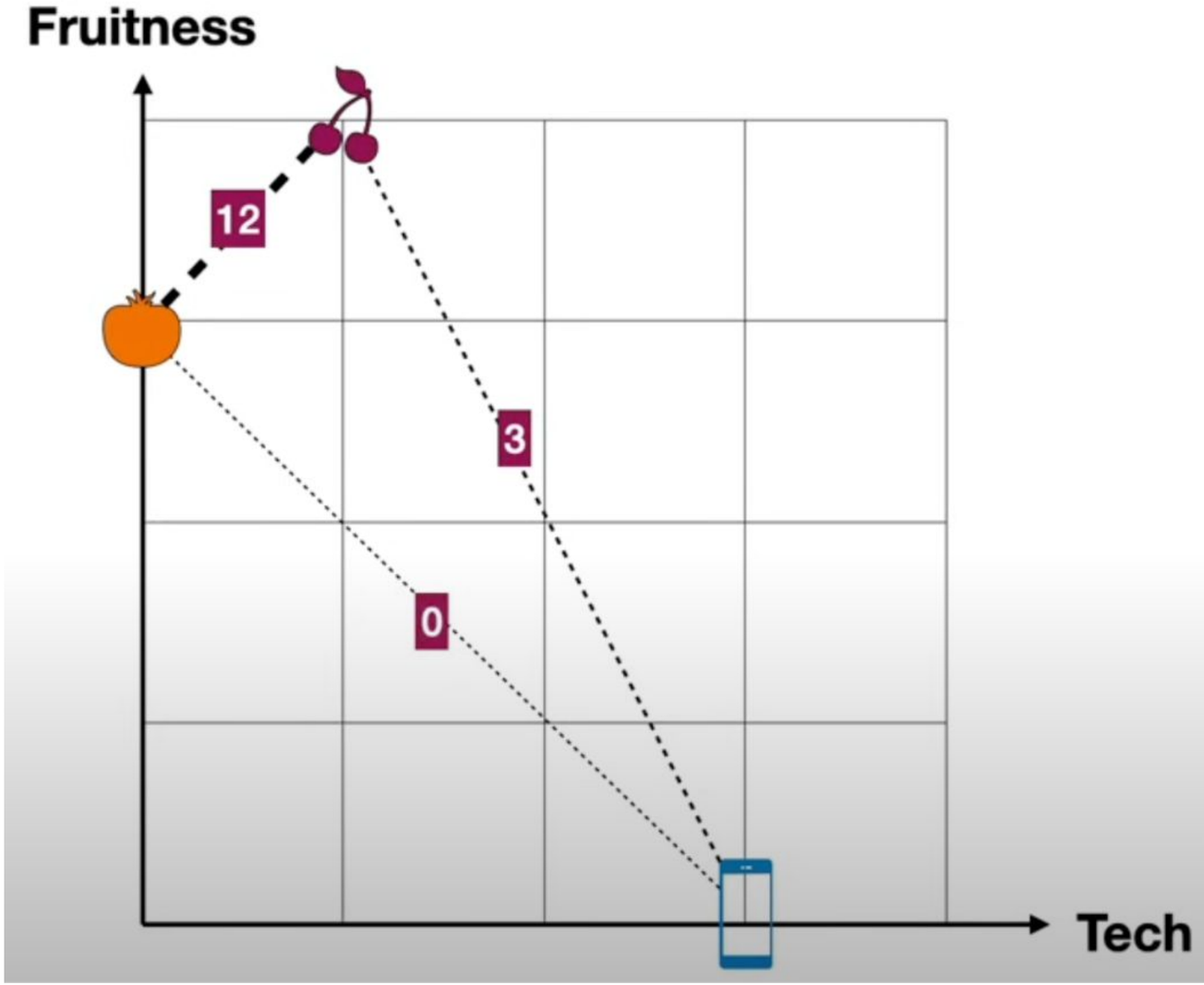


Similarity

Similarity



Measure 1: Dot product



	Tech	Fruitness
Sim	1	4
	0	3

$$1 \cdot 0 + 4 \cdot 3 = 12$$

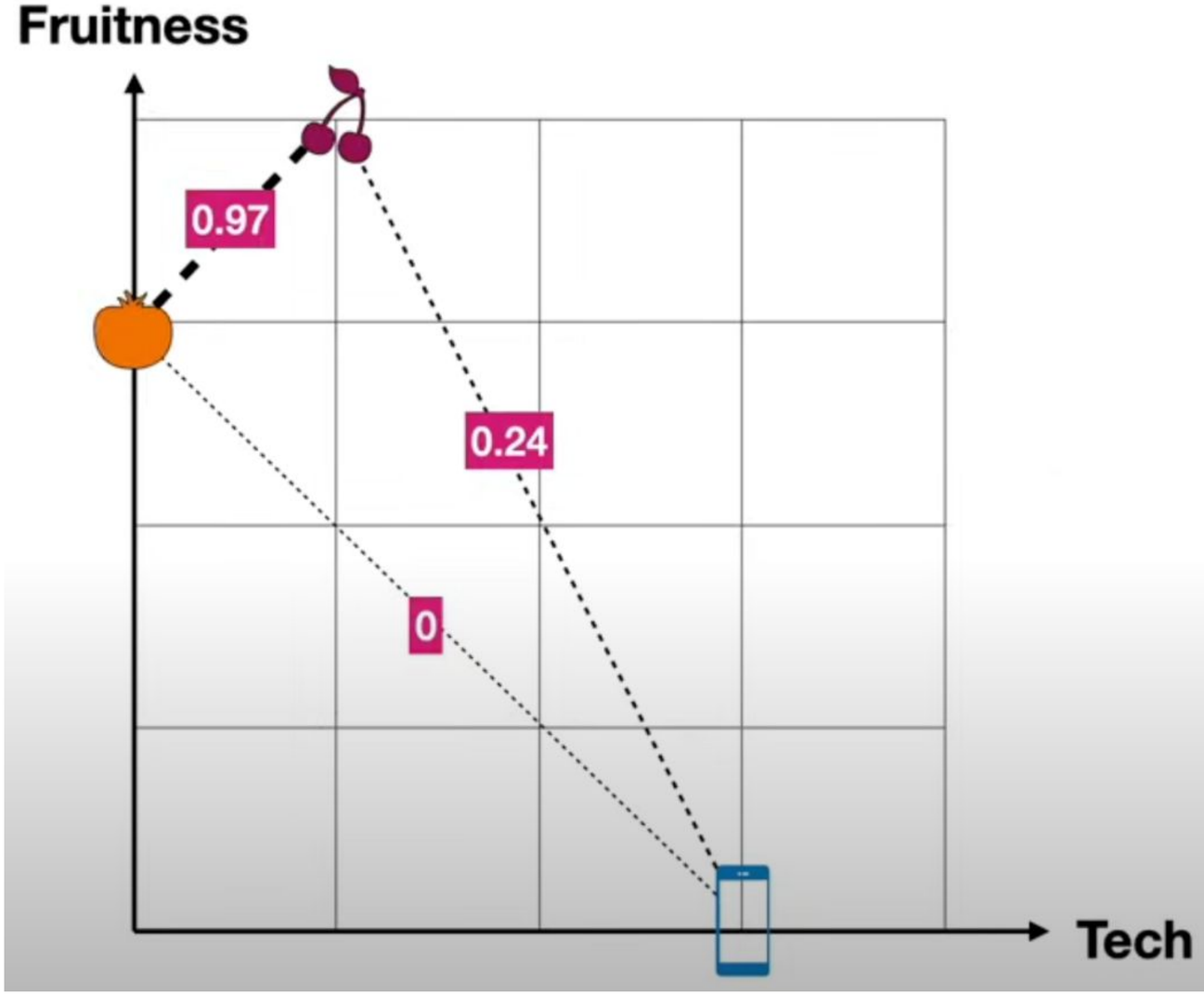
	Tech	Fruitness
Sim	1	4
	3	0

$$1 \cdot 3 + 4 \cdot 0 = 3$$

	Tech	Fruitness
Sim	0	3
	3	0

$$0 \cdot 3 + 3 \cdot 0 = 0$$

Measure 2: Cosine similarity

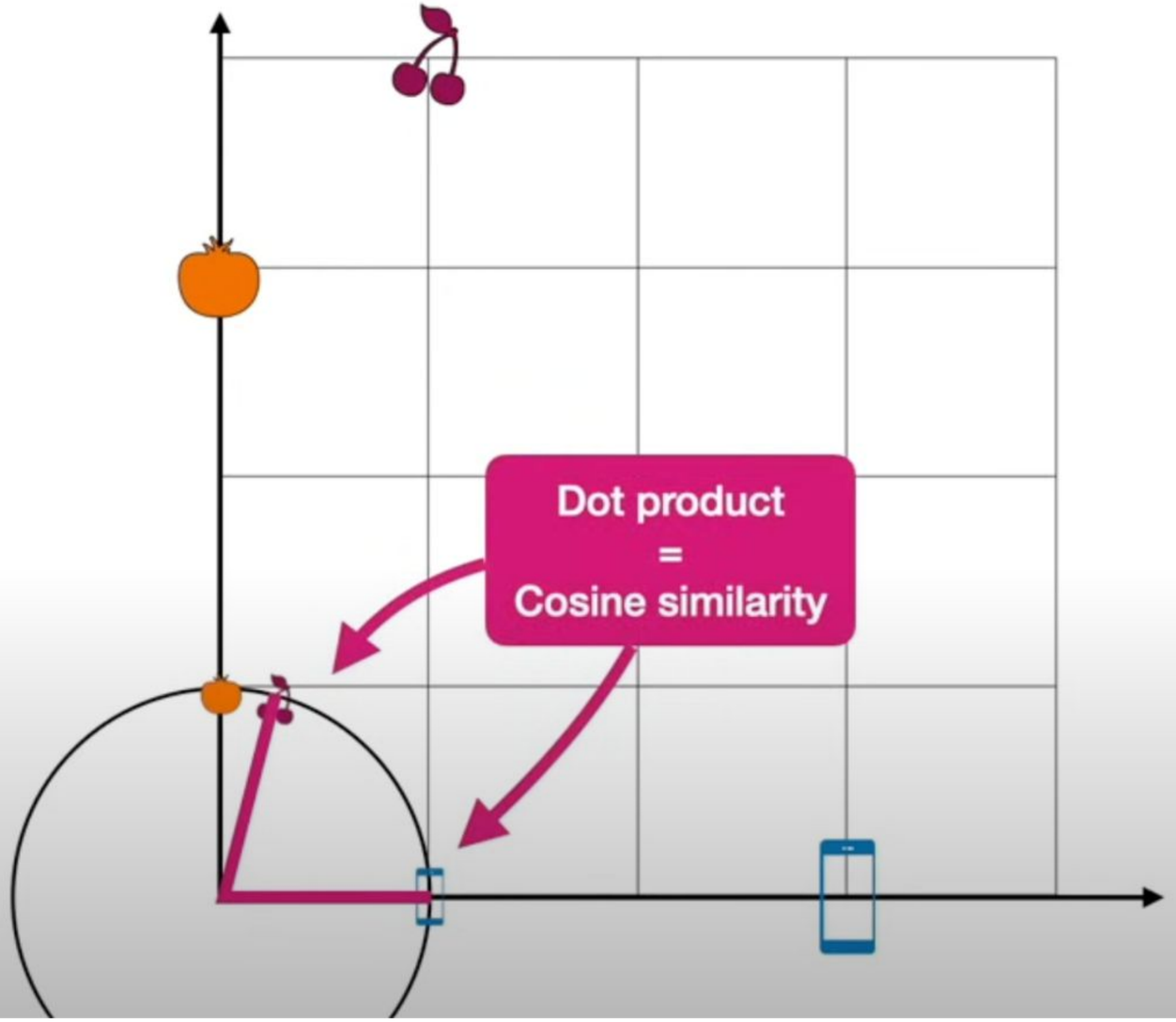


Sim $cos(14°) = 0.97$

Sim $cos(76°) = 0.24$

Sim $cos(90°) = 0$

Dot product and cosine similarity



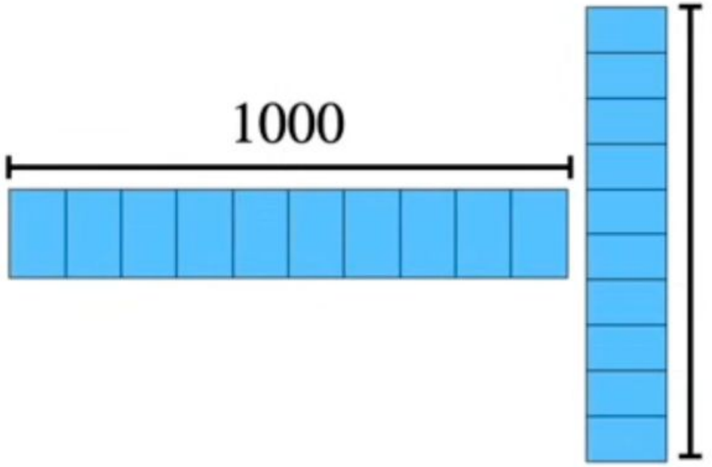
Measure 3: Scaled dot product

Dot product divided by the square root of the length of the vector

Sim

1	4
0	3

$$1 \cdot 0 + 4 \cdot 3 = 12 \longrightarrow \frac{12}{\sqrt{2}} = 8.49$$



Sim

1	4
3	0

$$1 \cdot 3 + 4 \cdot 0 = 3 \longrightarrow \frac{3}{\sqrt{2}} = 2.12$$

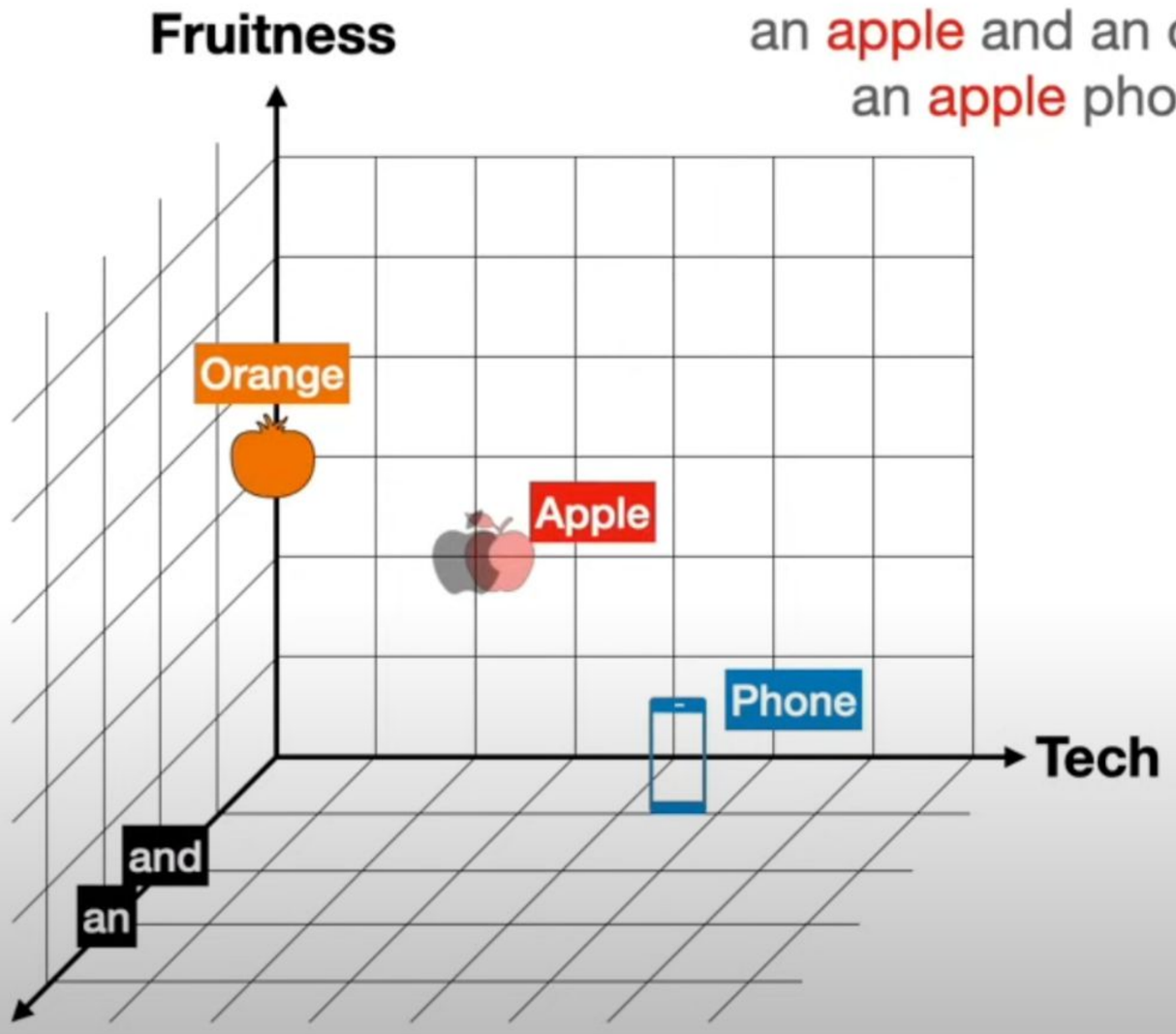
Sim

0	3
3	0

$$0 \cdot 3 + 3 \cdot 0 = 0 \longrightarrow \frac{0}{\sqrt{2}} = 0$$

Attention

Cosine similarity



	Tech	Fruitness	Other
Orange	0	3	0
Phone	4	0	0
Apple	2	2	0
And	0	0	2
An	0	0	3

	Orange	Phone	Apple	And	An
Orange	1	0	0.71	0	0
Phone	0	1	0.71	0	0
Apple	0.71	0.71	1	0	0
And	0	0	0	1	1
An	0	0	0	1	1

Word math

an apple and an orange

	Orange	Apple	And	An
Orange	1	0.71	0	0
Apple	0.71	1	0	0
And	0	0	1	1
An	0	0	1	1

Orange → 1 Orange + 0.71 Apple

Apple → 0.71 Orange + 1 Apple

And → 1 And + 1 An

An → 1 An + 1 And

Word math

an apple phone

	Phone	Apple	An
Phone	1	0.71	0
Apple	0.71	1	0
An	0	0	1

Phone $\longrightarrow$ 1 Phone + 0.71 Apple

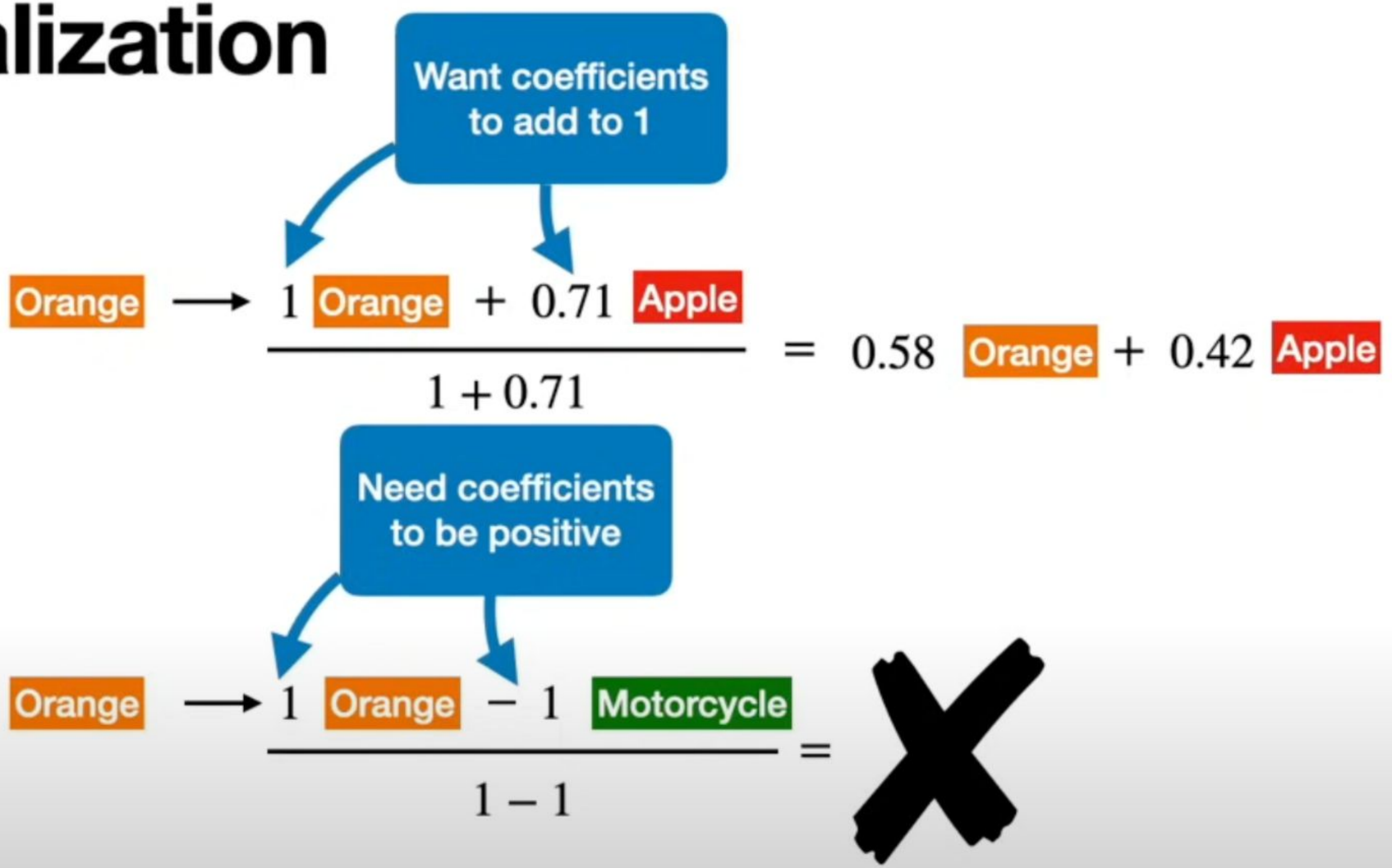
Apple $\longrightarrow$ 0.71 Phone + 1 Apple

An $\longrightarrow$ 1 An

Normalization

Want coefficients to add to 1

Need coefficients to be positive



Normalization

Want coefficients to add to 1

Orange $\longrightarrow \dfrac{1\ \text{Orange} + 0.71\ \text{Apple}}{1 + 0.71} = 0.58\ \text{Orange} + 0.42\ \text{Apple}$

Need coefficients to be positive



Orange $\longrightarrow \dfrac{1\ \text{Orange} - 1\ \text{Motorcycle}}{1 - 1} =$ 

Solution? $x \longrightarrow e^x$

Softmax

$$x \longrightarrow e^{x}$$

$$\text{Orange} \longrightarrow \frac{e^{1}\,\text{Orange} + e^{0.71}\,\text{Apple}}{e^{1} + e^{0.71}} = 0.57\,\text{Orange} + 0.43\,\text{Apple}$$



$$\text{Orange} \longrightarrow \frac{e^{1}\,\text{Orange} + e^{-1}\,\text{Motorcycle}}{e^{1} + e^{-1}} = 0.88\,\text{Orange} + 0.12\,\text{Motorcycle}$$

an apple and an orange

	Orange	Apple	And	An
Orange	1	0.71	0	0
Apple	0.71	1	0	0
And	0	0	1	1
An	0	0	1	1

Orange → 1 Orange + 0.71 Apple

Apple → 0.71 Orange + 1 Apple

And → 1 And + 1 An

An → 1 An + 1 And

an apple phone

	Phone	Apple	An
Phone	1	0.71	0
Apple	0.71	1	0
An	0	0	1

Phone → 1 Phone + 0.71 Apple

Apple → 0.71 Phone + 1 Apple

An → 1 An

an apple and an orange

	Orange	Apple	And	An
Orange	1	0.71	0	0
Apple	0.71	1	0	0
And	0	0	1	1
An	0	0	1	1

$$\text{Orange} \rightarrow 0.57\,\text{Orange} + 0.43\,\text{Apple}$$

$$\text{Apple} \rightarrow 0.43\,\text{Orange} + 0.57\,\text{Apple}$$

$$\text{And} \rightarrow 0.5\,\text{And} + 0.5\,\text{An}$$

$$\text{An} \rightarrow 0.5\,\text{An} + 0.5\,\text{And}$$

an apple phone

	Phone	Apple	An
Phone	1	0.71	0
Apple	0.71	1	0
An	0	0	1

$$\text{Phone} \rightarrow 0.57\,\text{Phone} + 0.43\,\text{Apple}$$

$$\text{Apple} \rightarrow 0.43\,\text{Phone} + 0.57\,\text{Apple}$$

$$\text{An} \rightarrow 1\,\text{An}$$

an apple and an orange

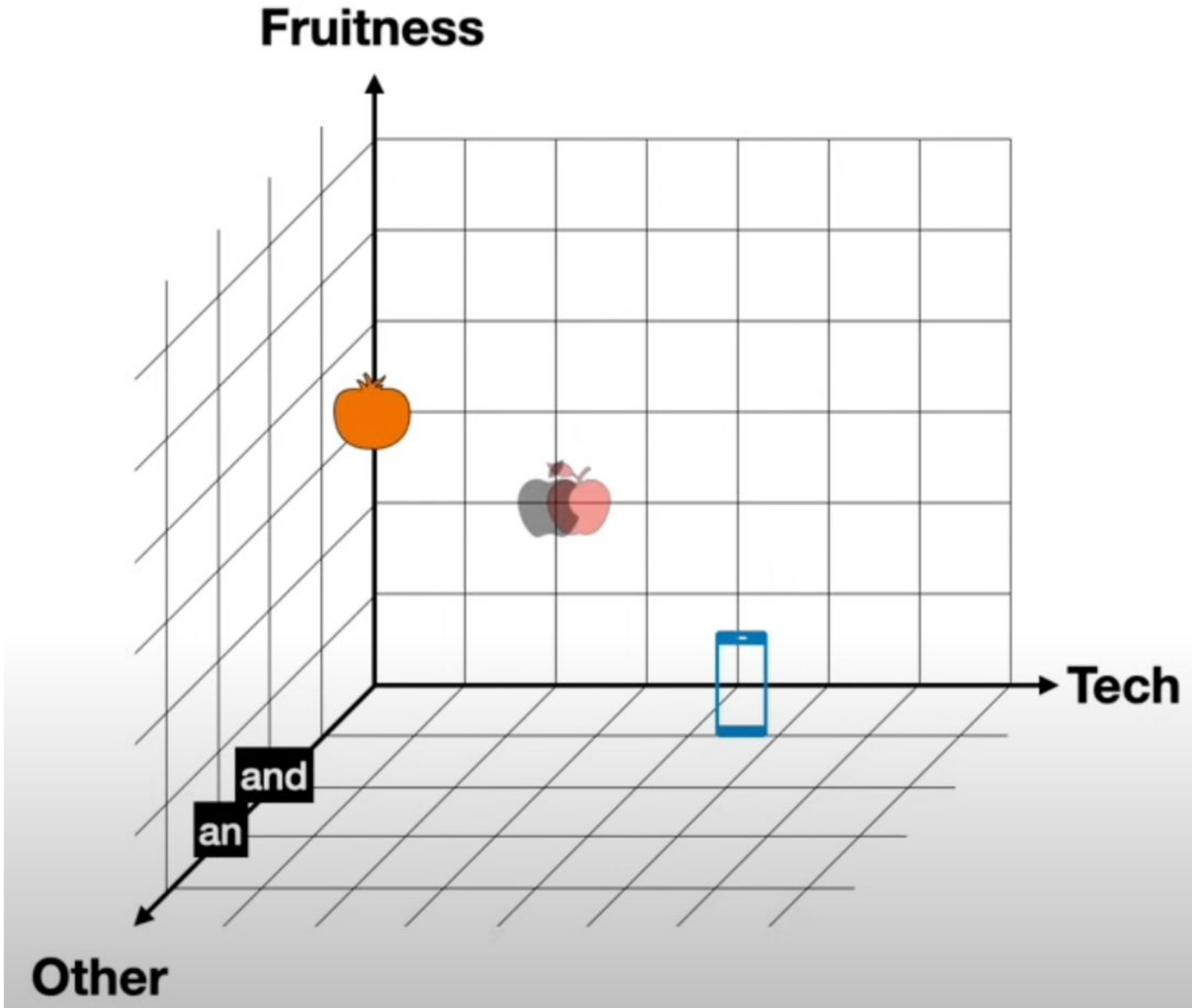
	Orange	Apple	And	An
Orange	1	0.71	0	0

$$\frac{e^{1}\,\text{Orange} + e^{0.71}\,\text{Apple} + e^{0}\,\text{And} + e^{0}\,\text{An}}{e^{1} + e^{0.71} + e^{0} + e^{0}}$$

Ok, I'm kinda lying to you…

$$\text{Orange} \longrightarrow 0.57\,\text{Orange} + 0.43\,\text{Apple}$$

$$\text{Orange} \longrightarrow 0.4\,\text{Orange} + 0.3\,\text{Apple} + 0.15\,\text{And} + 0.15\,\text{An}$$

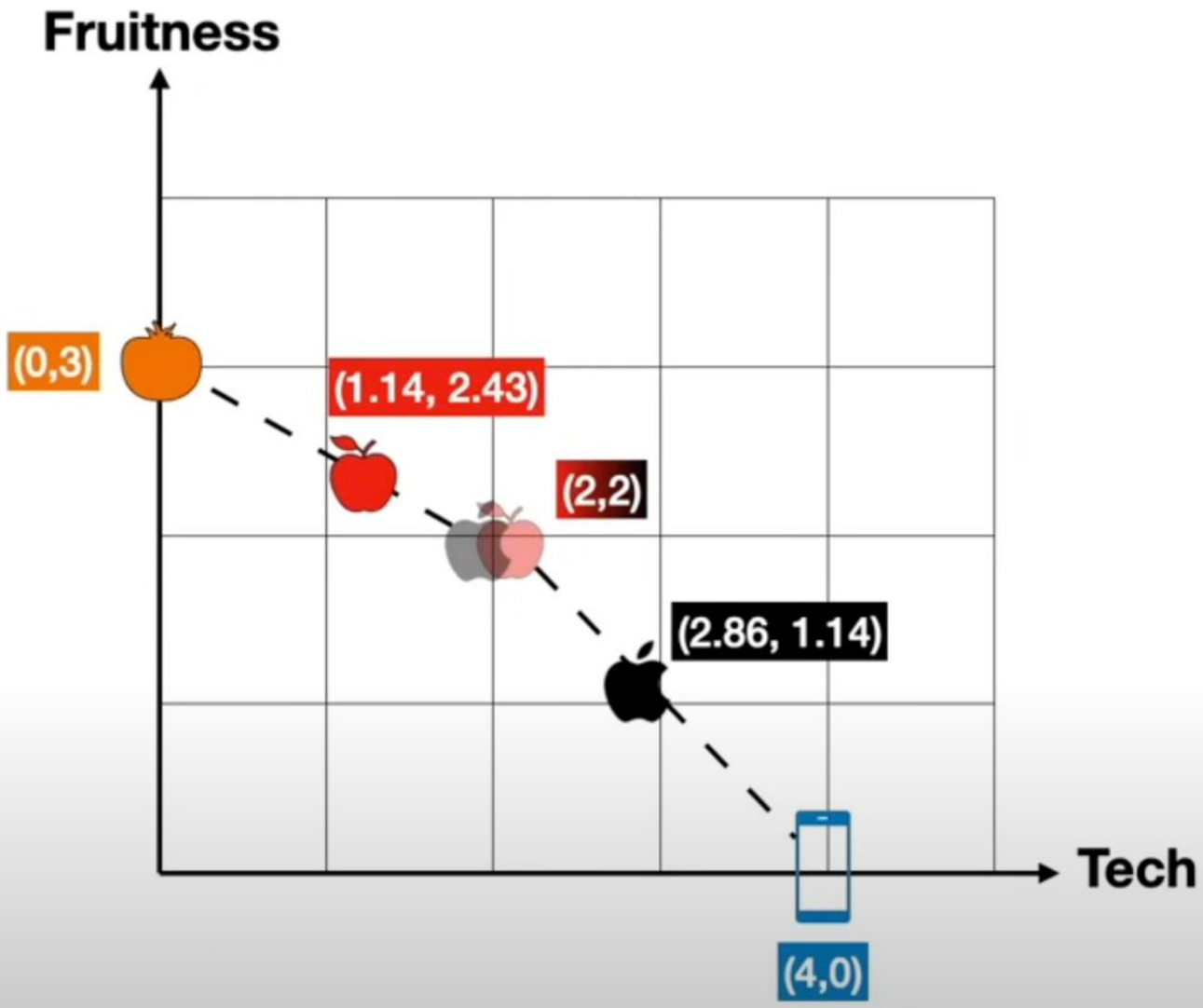


an apple and an orange

Apple → 0.43 Orange + 0.57 Apple

an apple phone

Apple → 0.43 Phone + 0.57 Apple



an apple and an orange

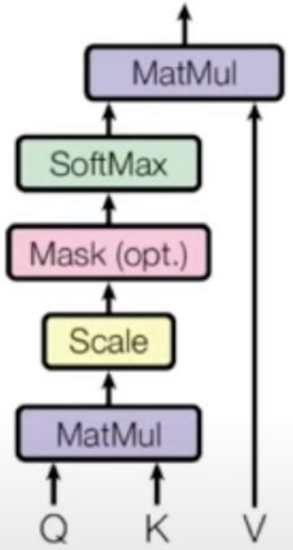
Apple → 0.43 Orange + 0.57 Apple

an apple phone

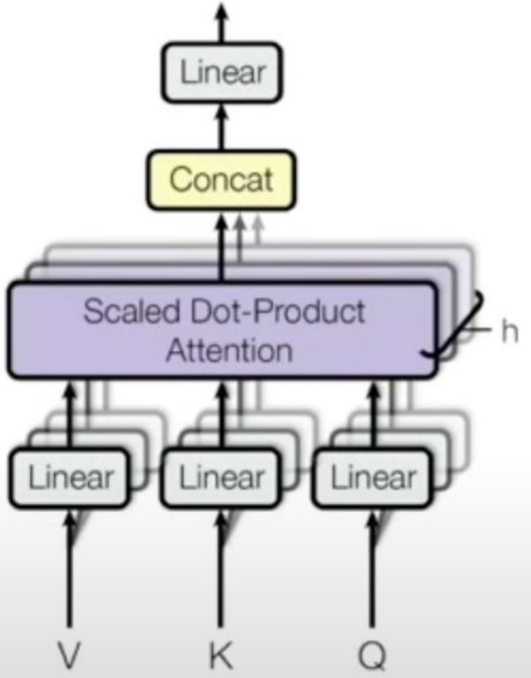
Apple → 0.43 Phone + 0.57 Apple

Attention

Scaled Dot-Product Attention



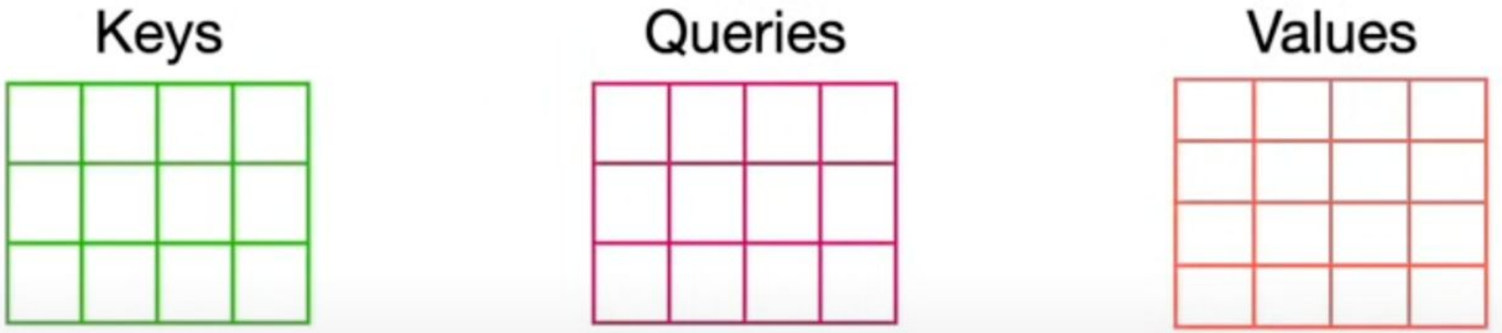
Multi-Head Attention



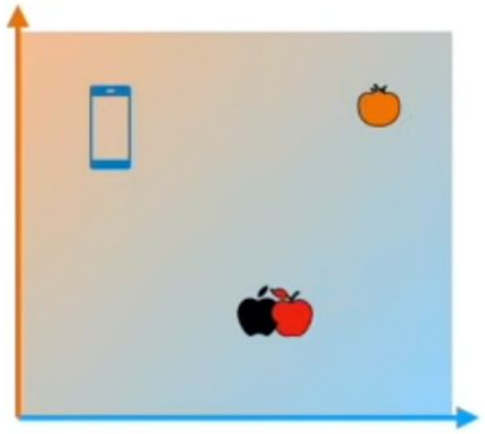
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

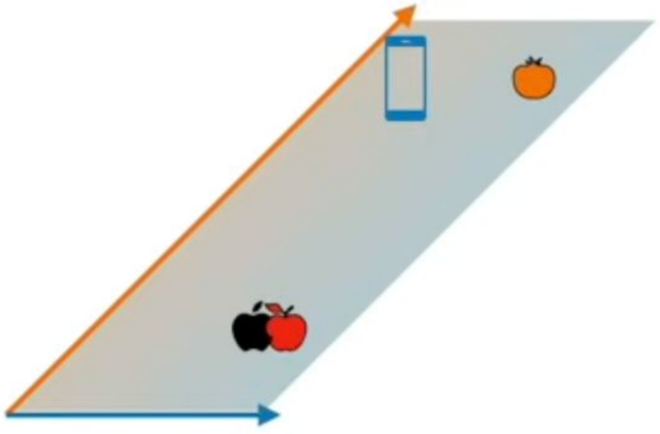
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

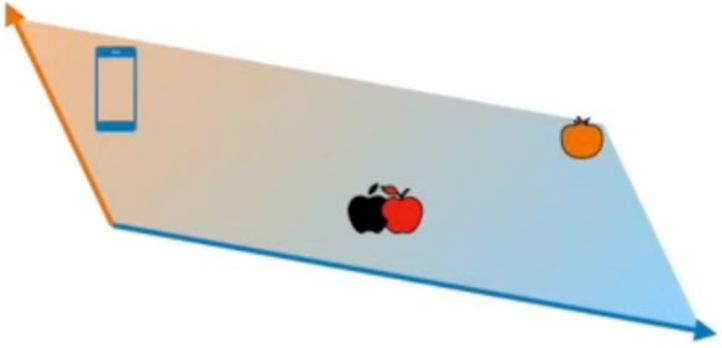
The Key, Query, and Value matrices



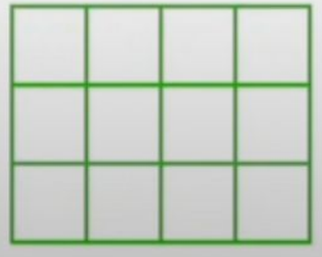
Get new embeddings from existing ones







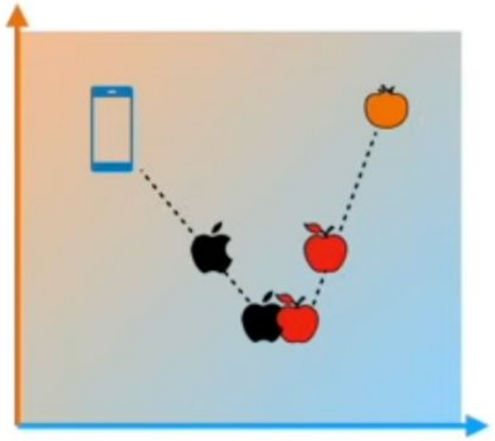
Keys



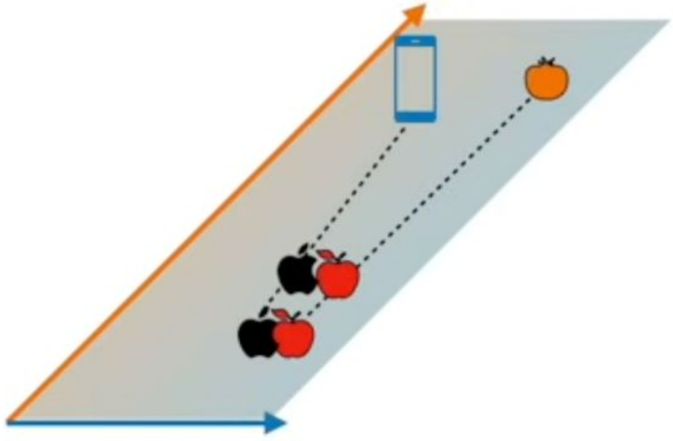
Queries



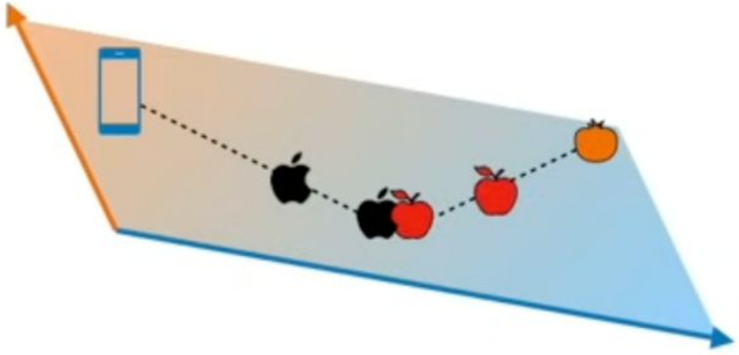
Get new embeddings from existing ones

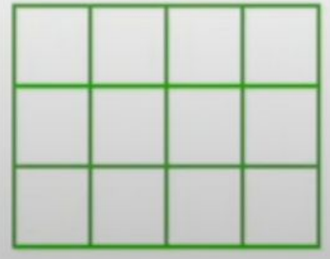


Okay



Bad



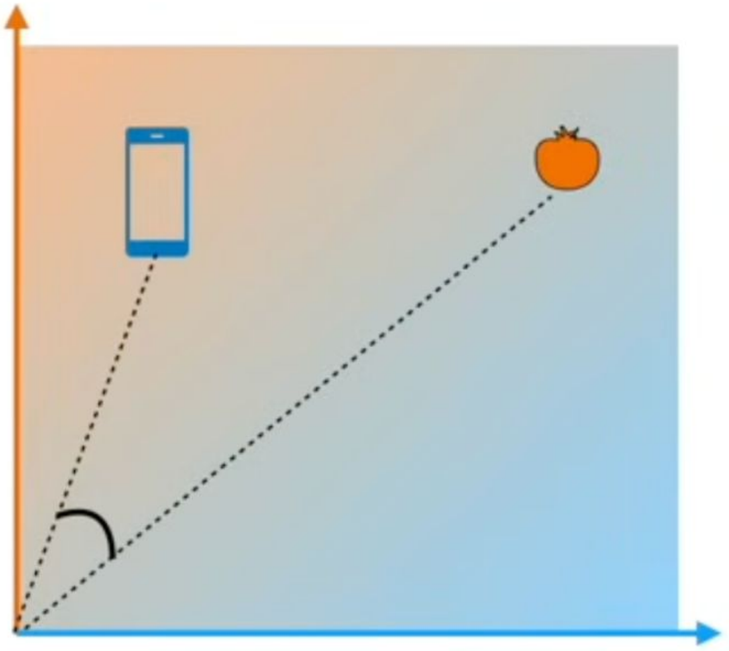




Similarity

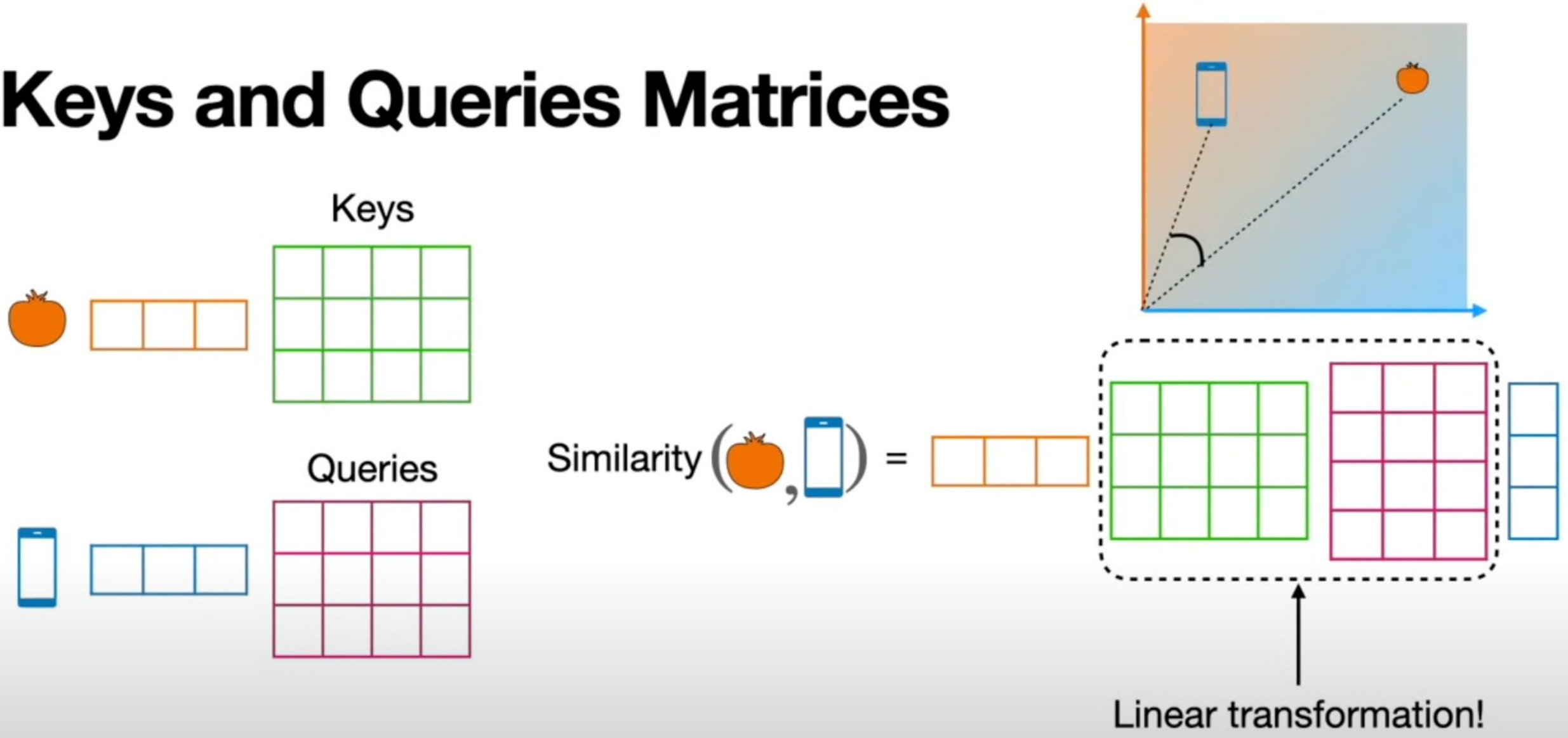




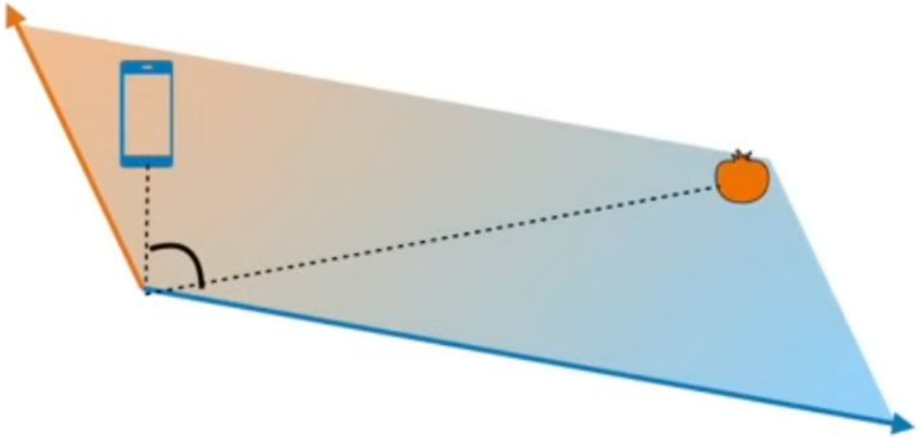


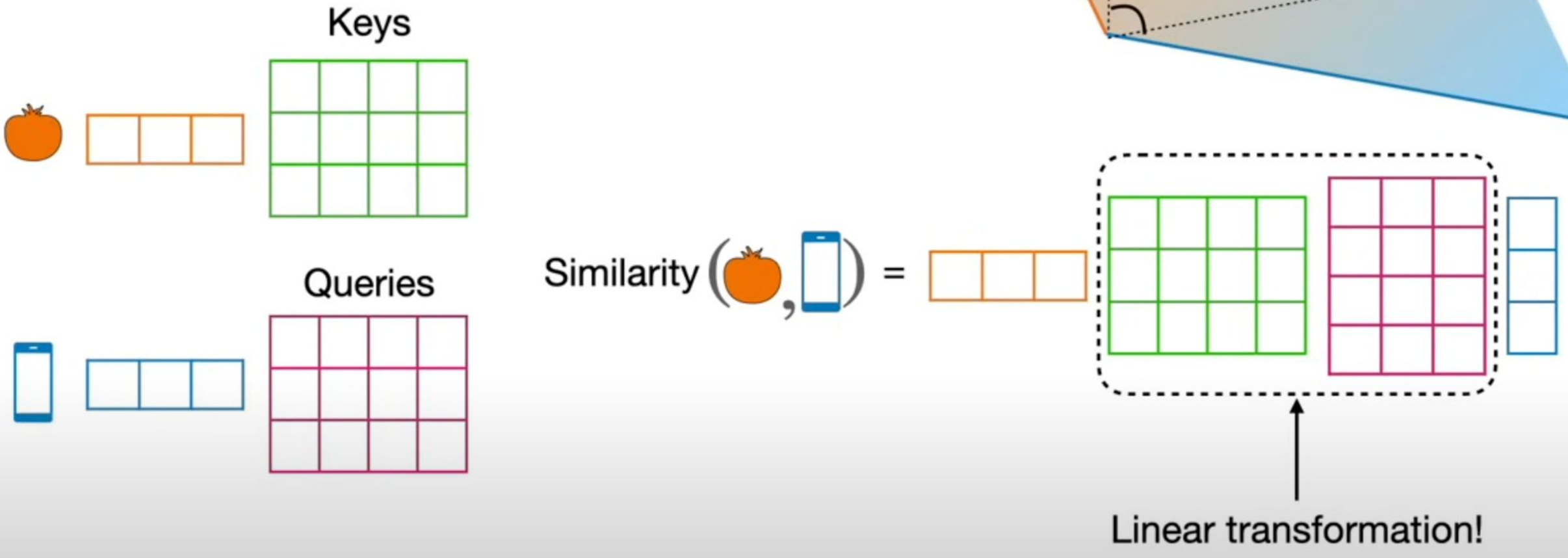


Keys and Queries Matrices

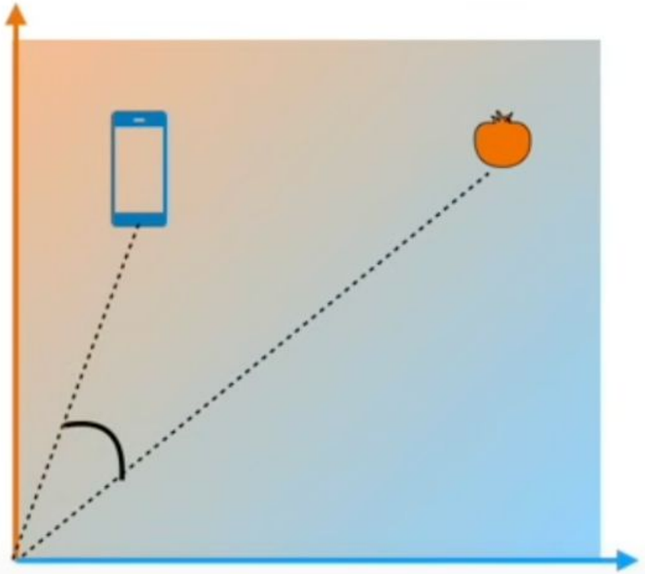


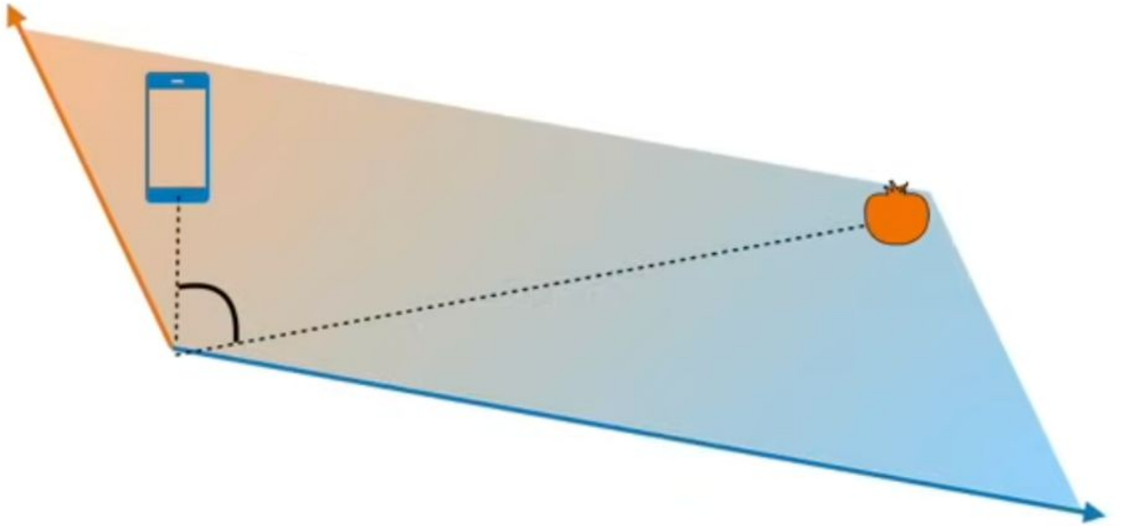
Keys and Queries Matrices



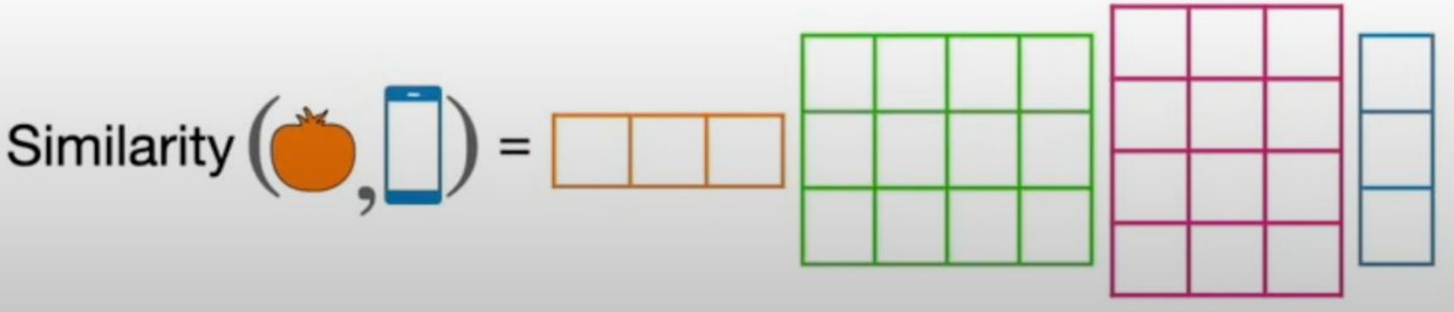


Similarity on a transformed embedding

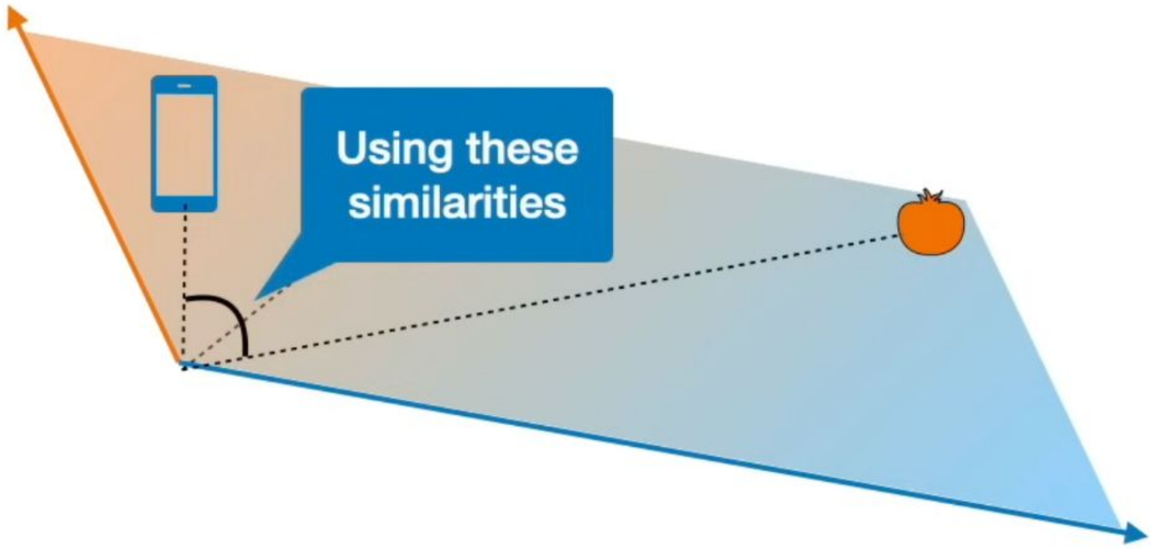






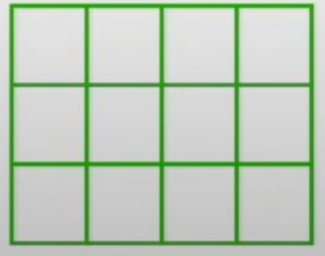


Values matrix



Best embedding for finding similarities

Keys



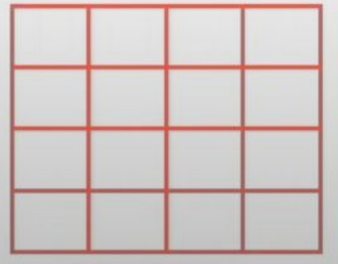
Queries



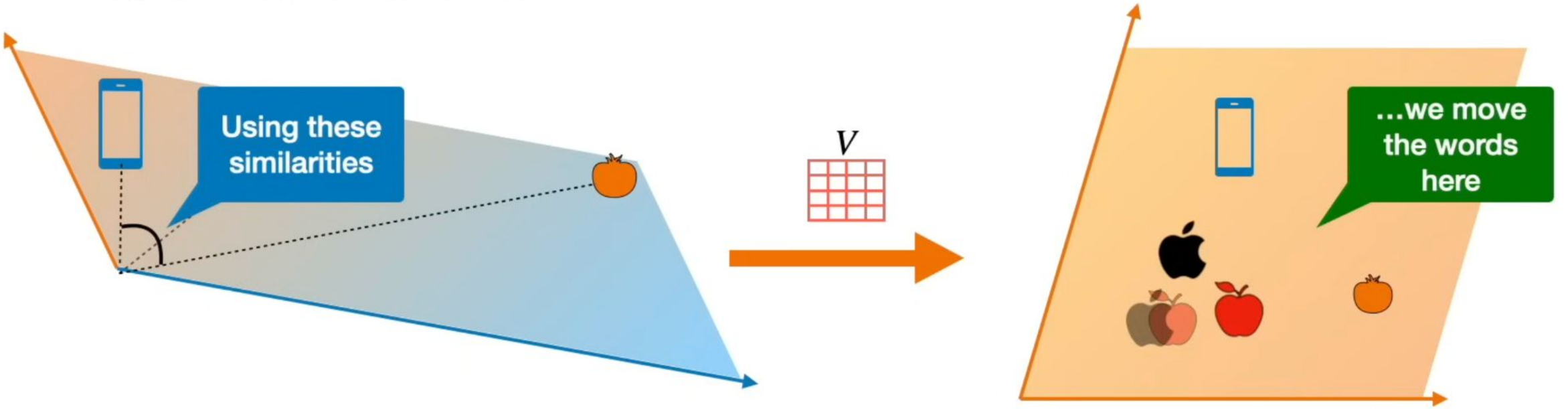


Best embedding for finding the next word

Values



Values matrix



Best embedding for finding similarities

Best embedding for finding the next word

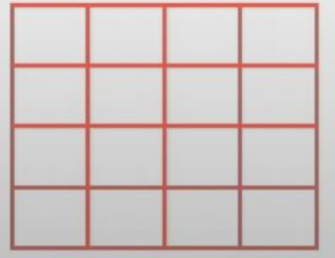
Keys

Queries

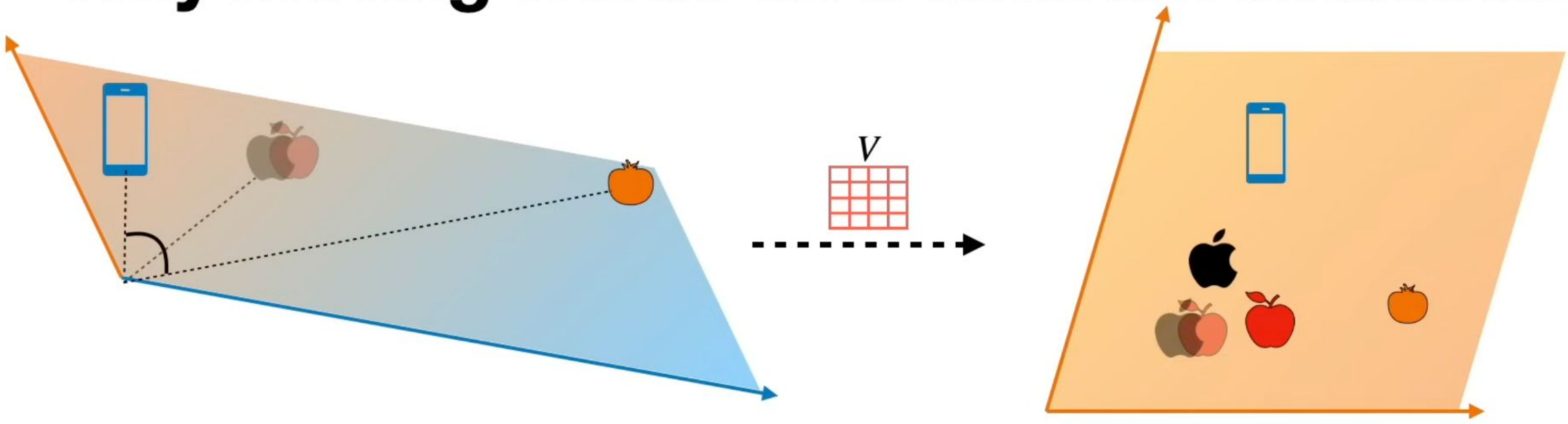
Values







Why moving words on a different embedding?



Best embedding for finding similarities

This embedding(s) know features of the words

- Color
- Size
- Fruitness
- Technology

Best embedding for finding the next word

This embedding knows when two words could appear in the same context

I want to buy a _____

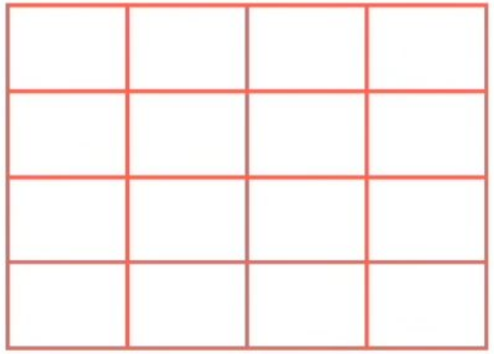
- car
- apple
- phone

Value matrix

an apple and an orange

	Orange	Apple	And	An
Orange	0.4	0.3	0.15	0.15
Apple	0.3	0.4	0.15	0.15
And	0.15	0.15	0.5	0.5
An	0.15	0.15	0.5	0.5

Value matrix



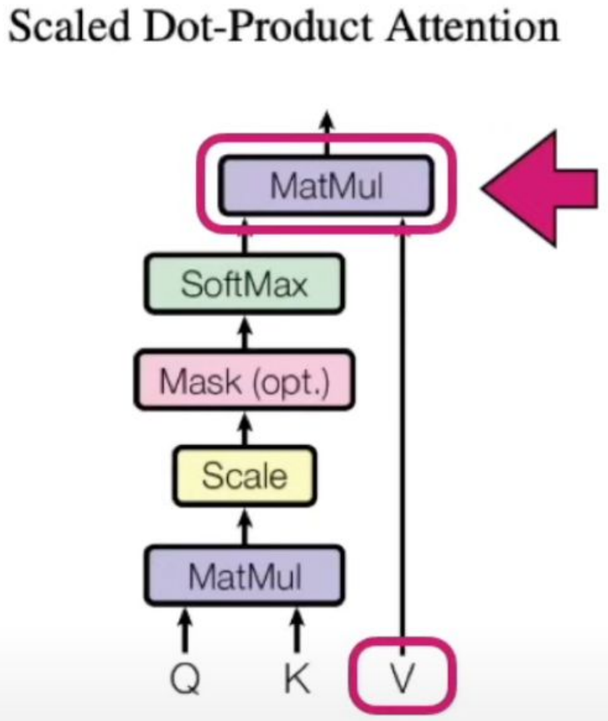
=

	Orange	Apple	And	An
Orange	v_{11}	v_{12}	v_{13}	v_{14}
Apple	v_{21}	v_{22}	v_{23}	v_{24}
And	v_{31}	v_{32}	v_{33}	v_{34}
An	v_{41}	v_{42}	v_{43}	v_{44}

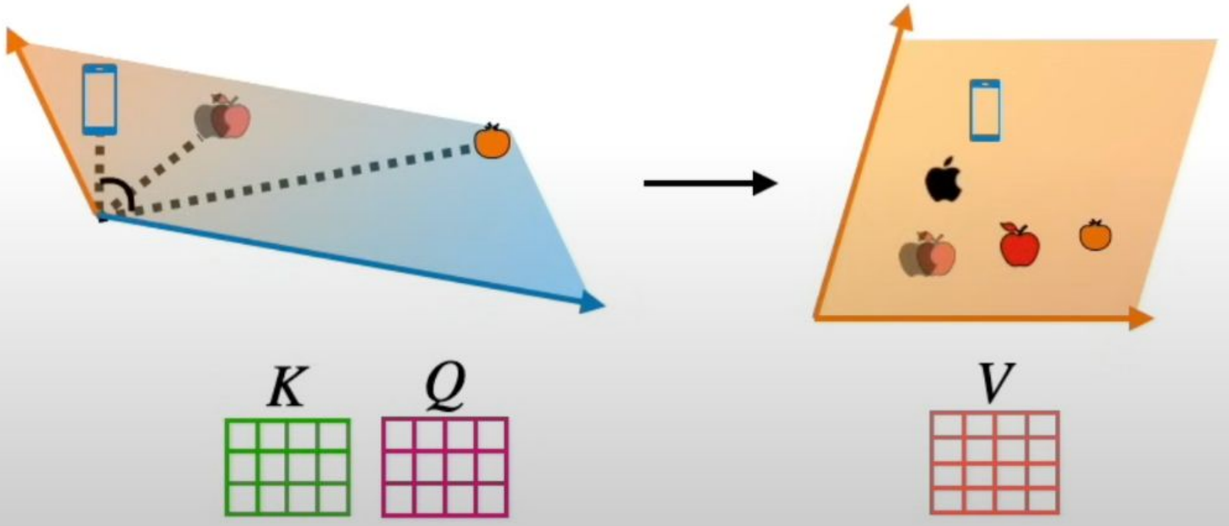
apple $\longrightarrow$ $0.3 \cdot$ orange
$+0.4 \cdot$ apple
$+0.15 \cdot$ and
$+0.15 \cdot$ an

apple $\longrightarrow$ $v_{21} \cdot$ orange
$+v_{22} \cdot$ apple
$+v_{23} \cdot$ and
$+v_{24} \cdot$ an

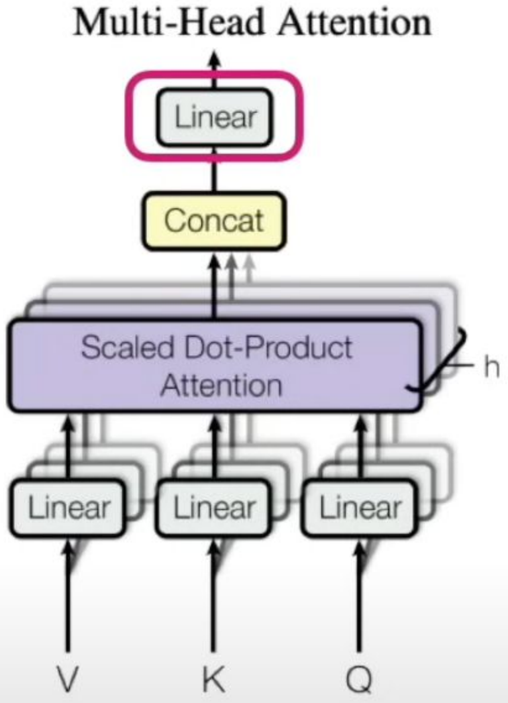
Self-attention



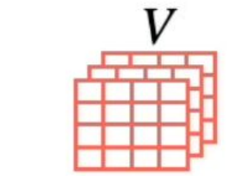
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}V)$$

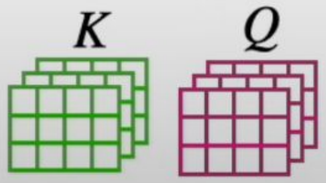


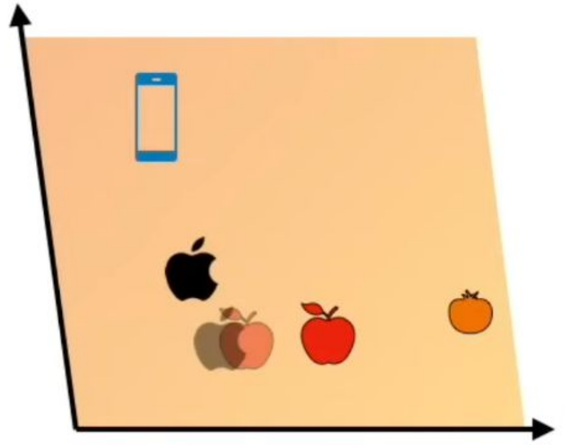
Multi-head attention

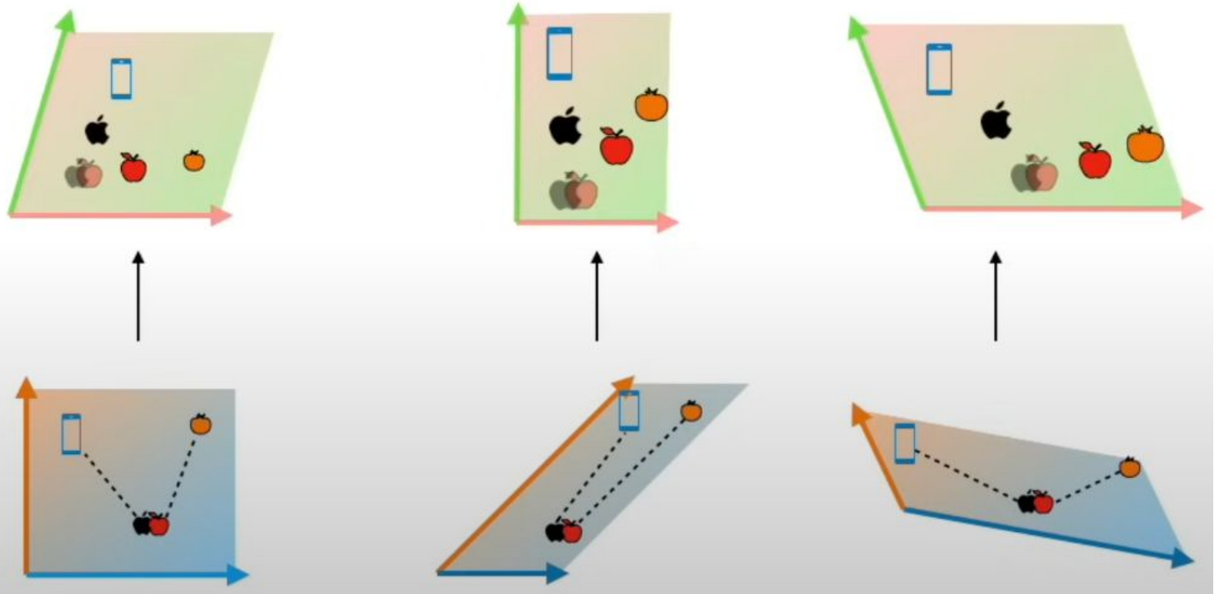


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

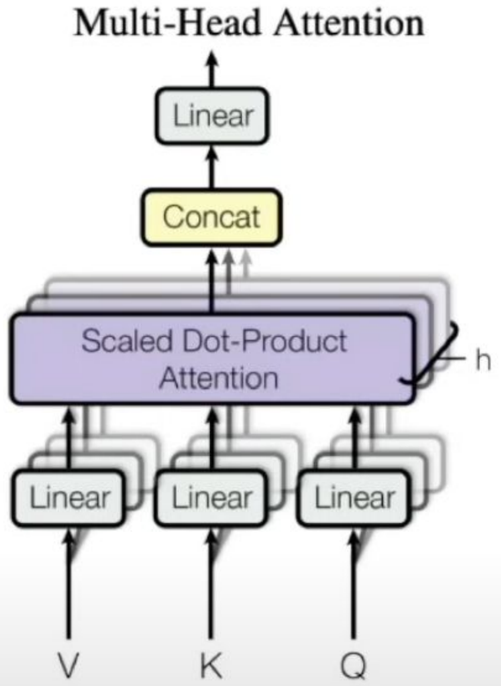




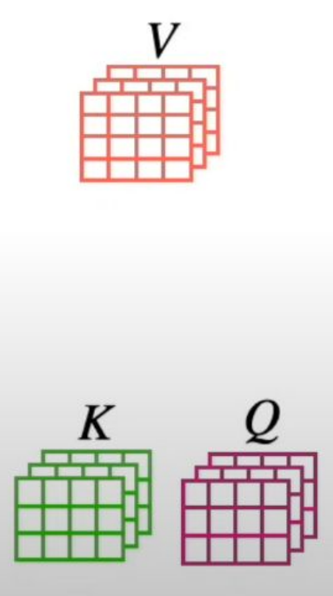


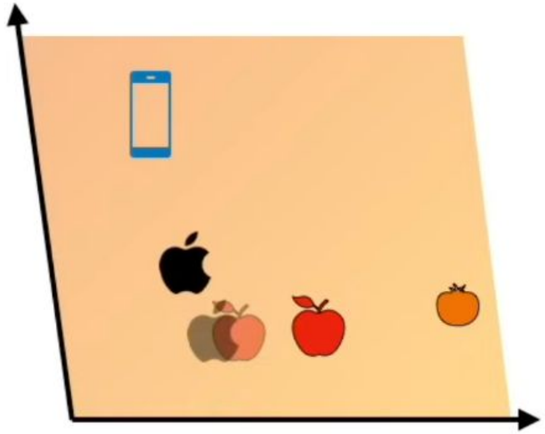


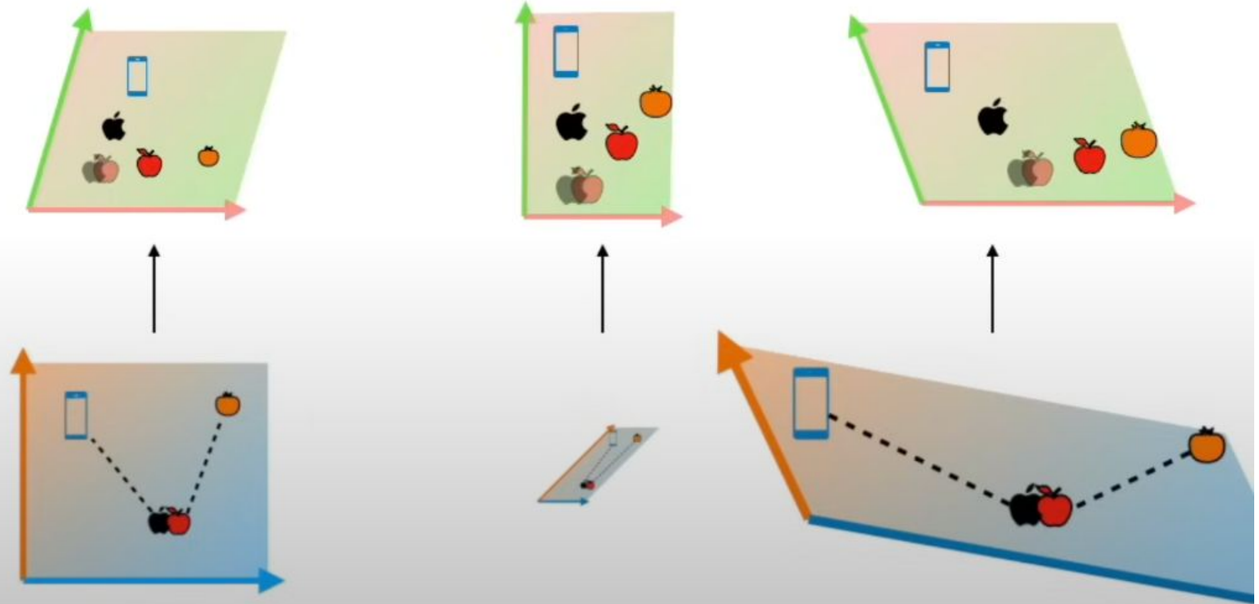
Multi-head attention



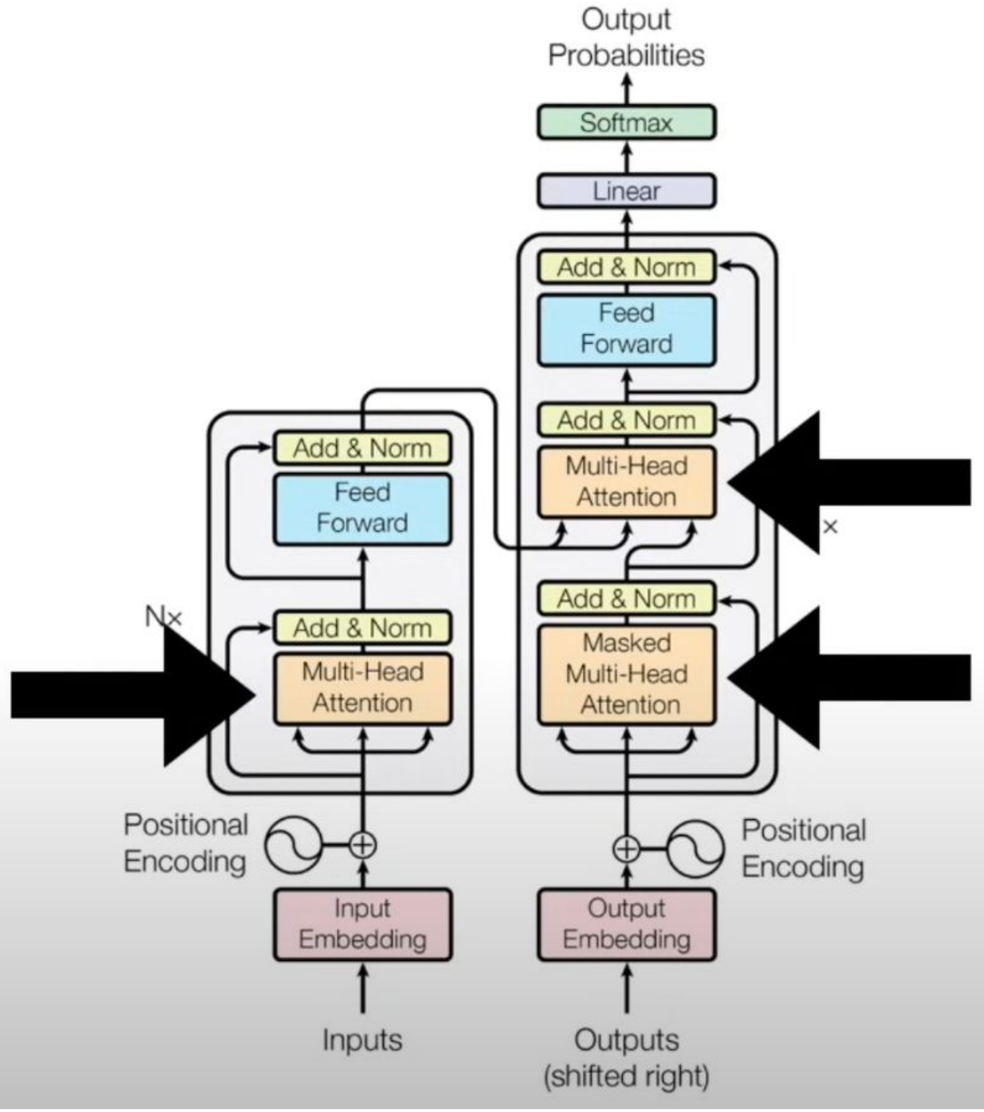
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$





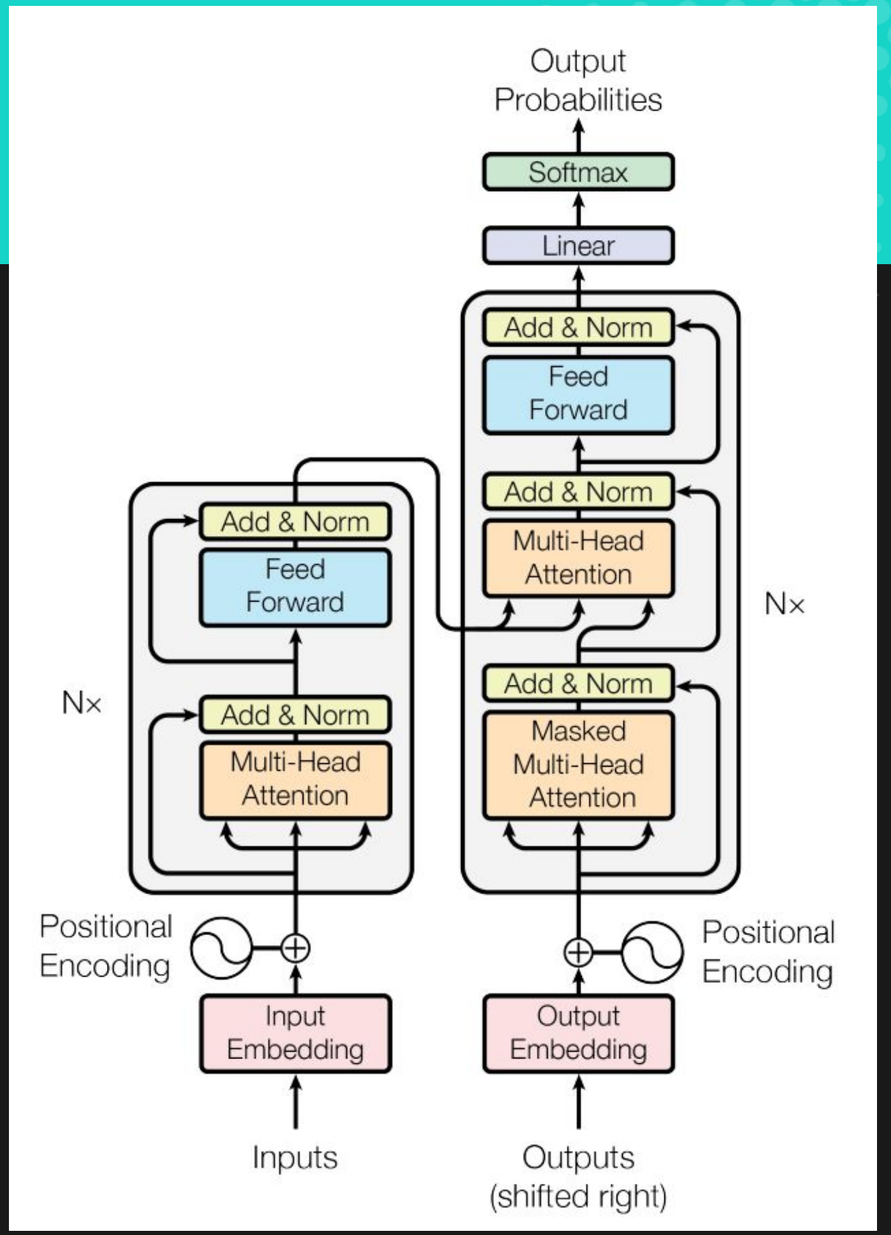


Weights get trained with the transformer model

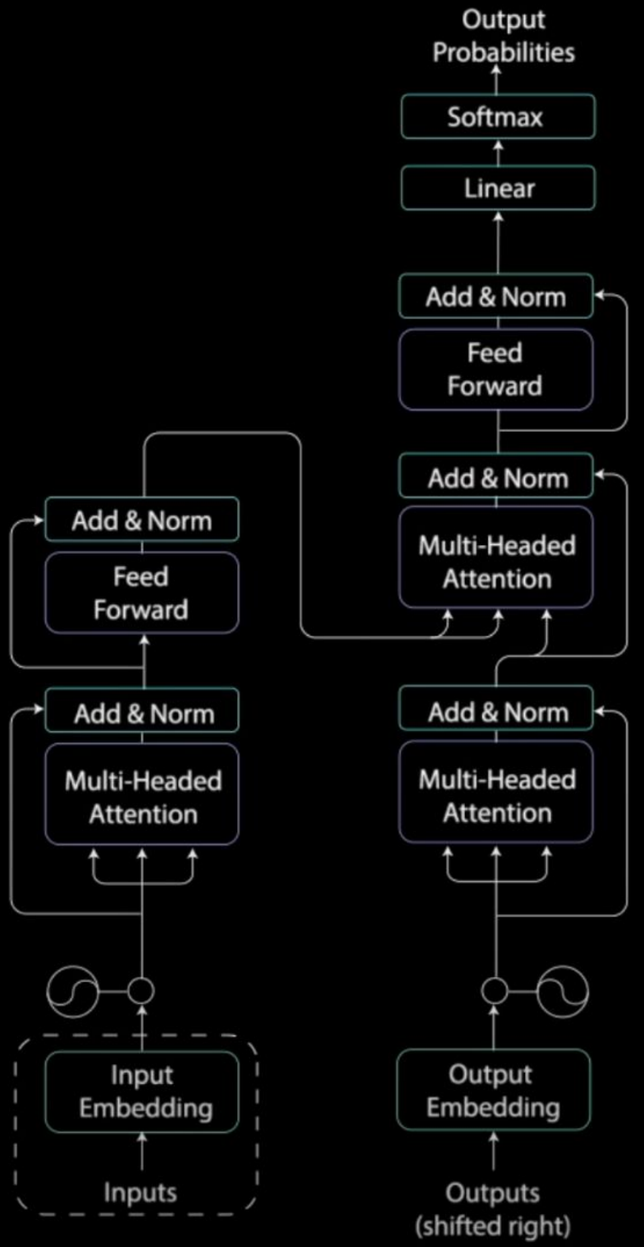


TRANSFORMER

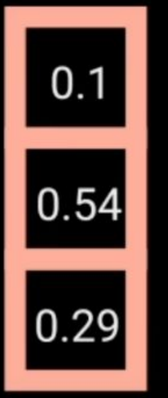


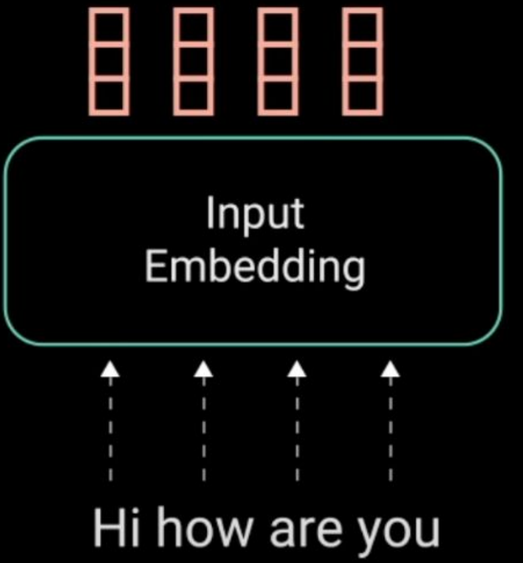


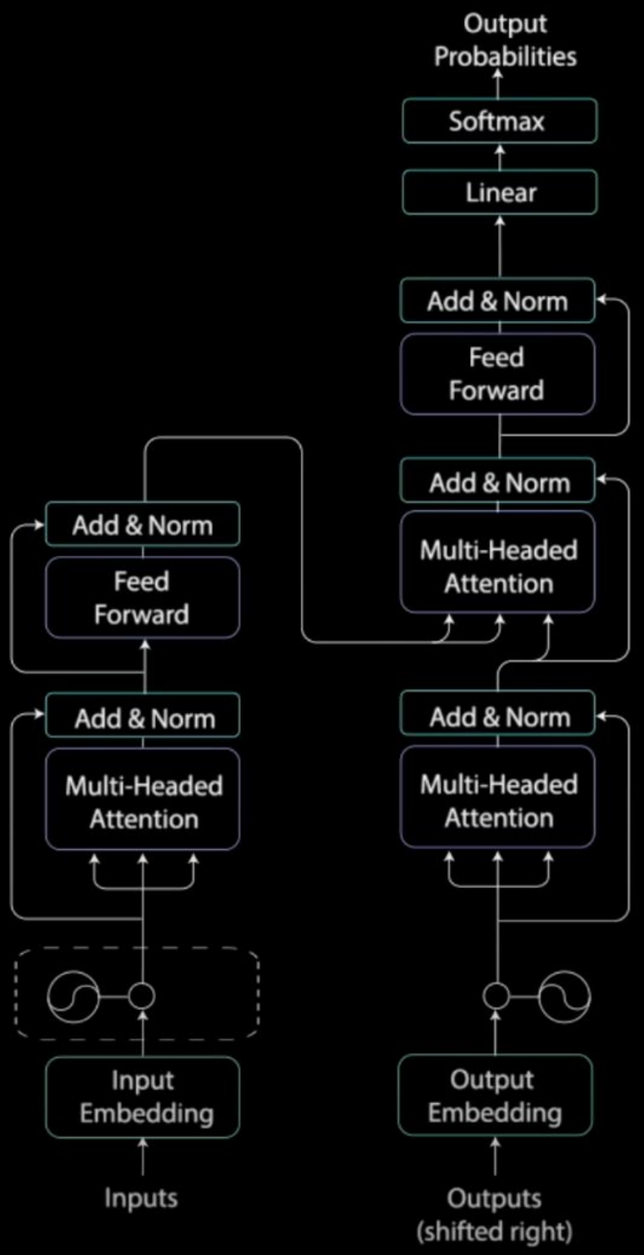
1. Input Embedding



Hi =







2. Positional Encoding

Positional Input Embeddings

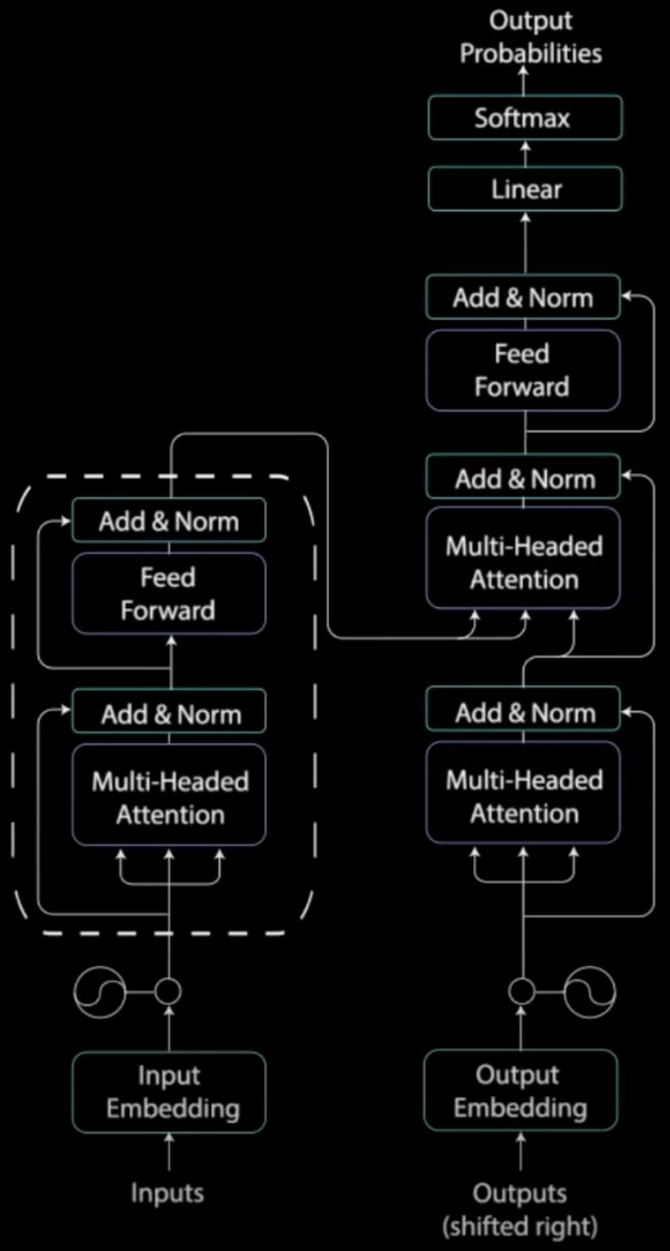
Positional Encoding

Time Step 1 2 3 4

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/dmodel}})$$

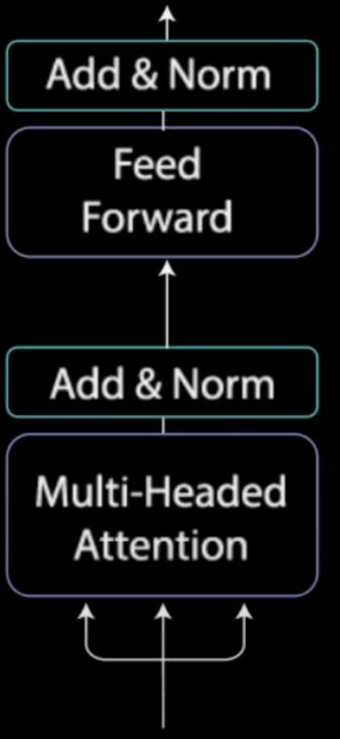
$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/dmodel}})$$

3 - 4. Encoder Layer



Encoder Input Representation



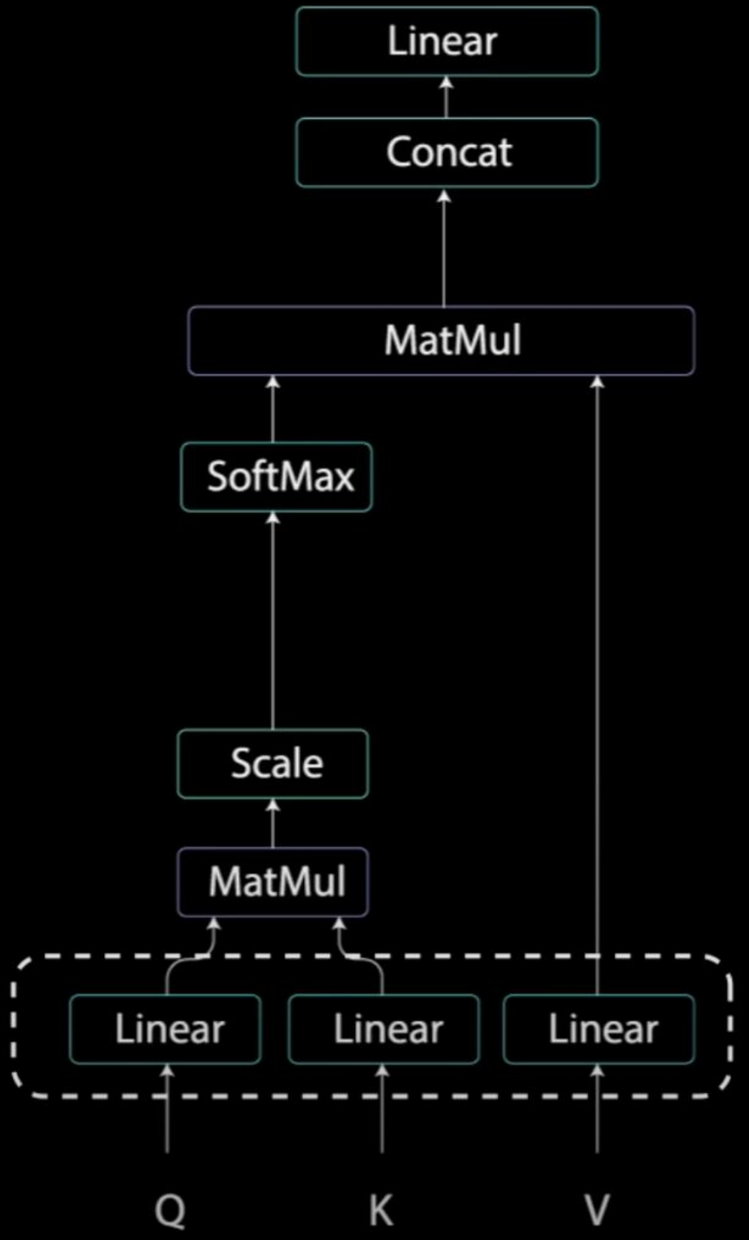


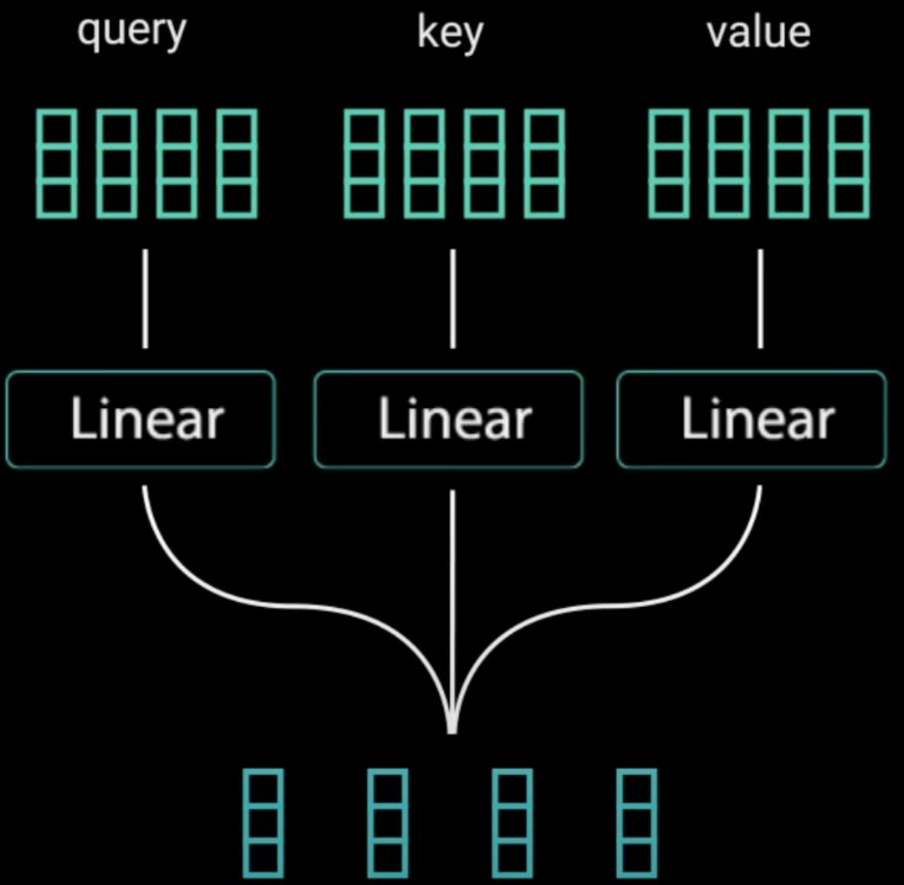
Positional Input Embedding



3. Multi-headed Attention

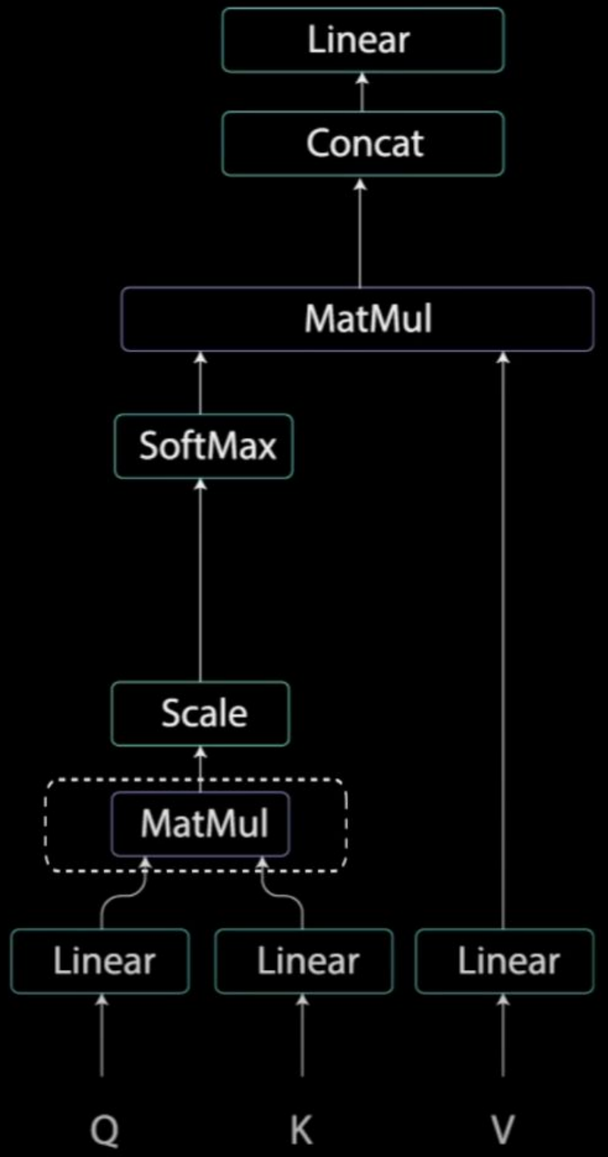
3.1. Self-Attention





3. Multi-headed Attention

3.1. Self-Attention



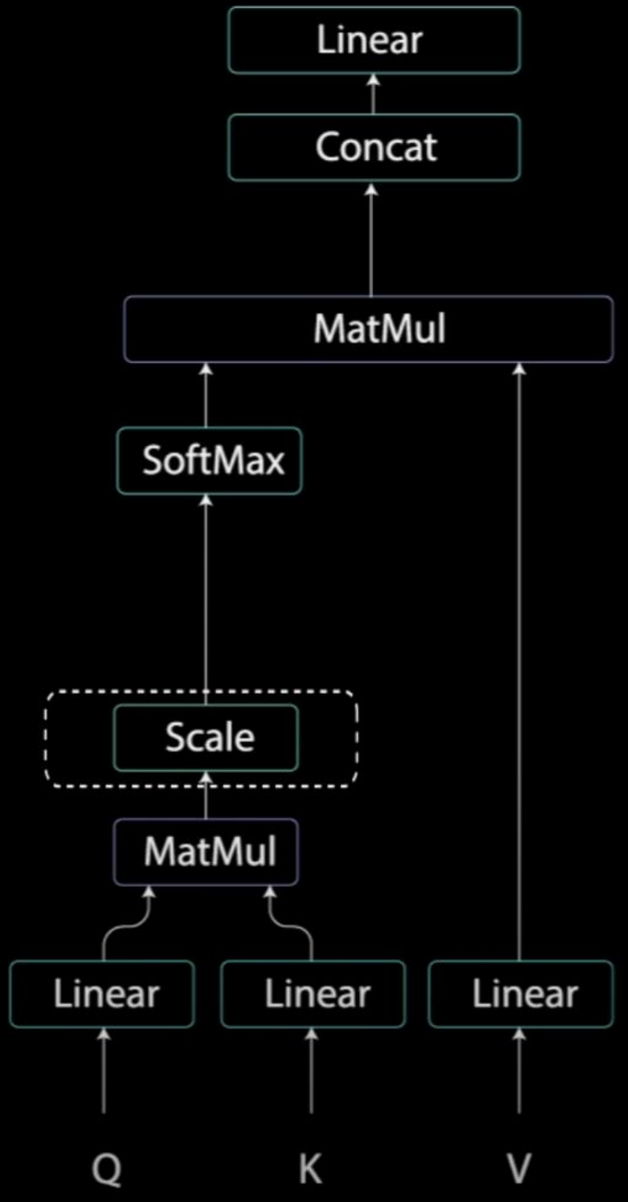


value



3. Multi-headed Attention

3.1. Self-Attention

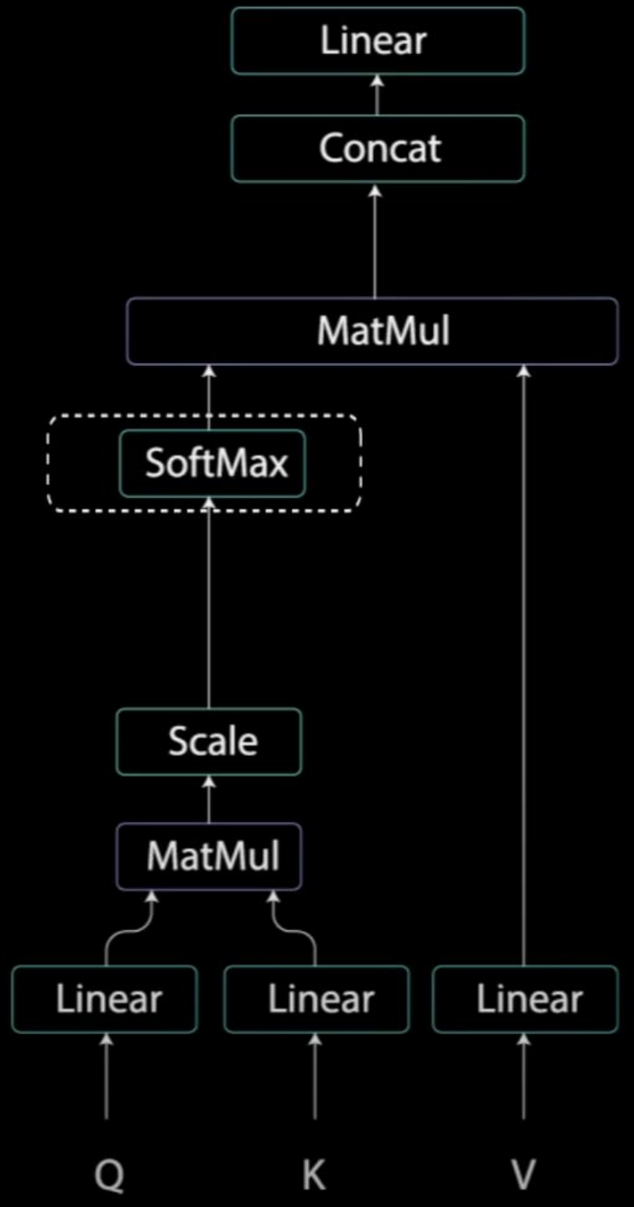


$$\frac{\square}{\sqrt{d_k}} = \square$$

Scaled Scores

3. Multi-headed Attention

3.1. Self-Attention



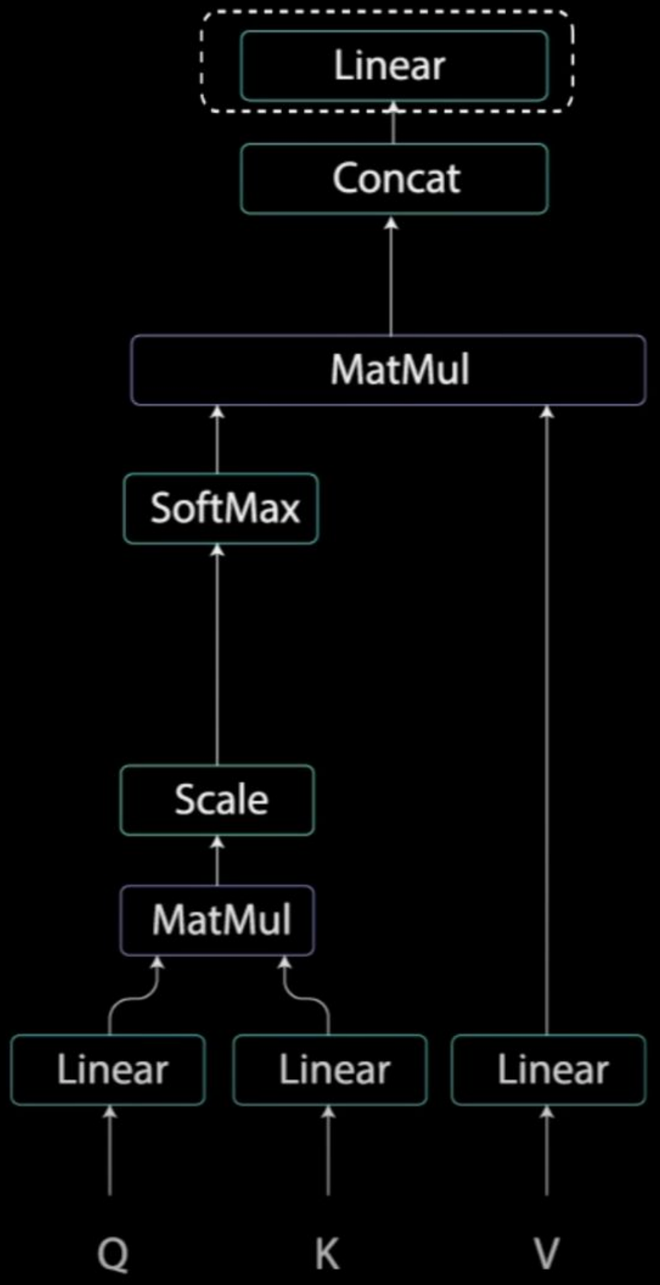
Softmax(▦) =

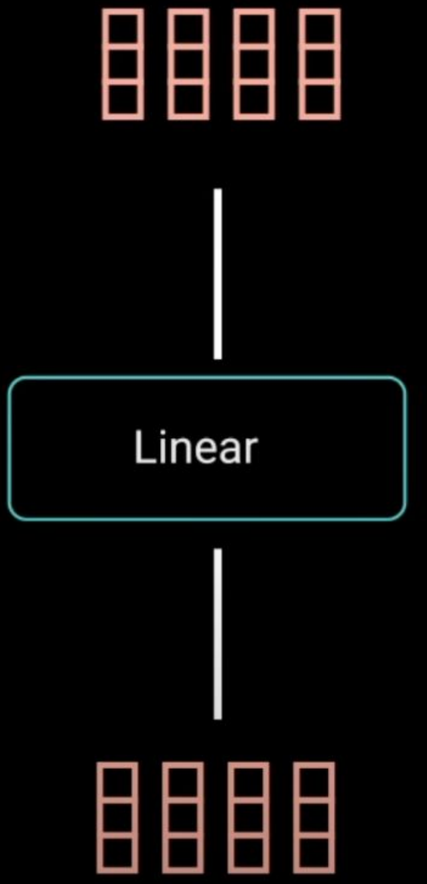
	Hi	how	are	you
Hi	0.7	0.1	0.1	0.1
how	0.1	0.6	0.2	0.1
are	0.1	0.3	0.6	0.1
you	0.1	0.3	0.3	0.3

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

3. Multi-headed Attention

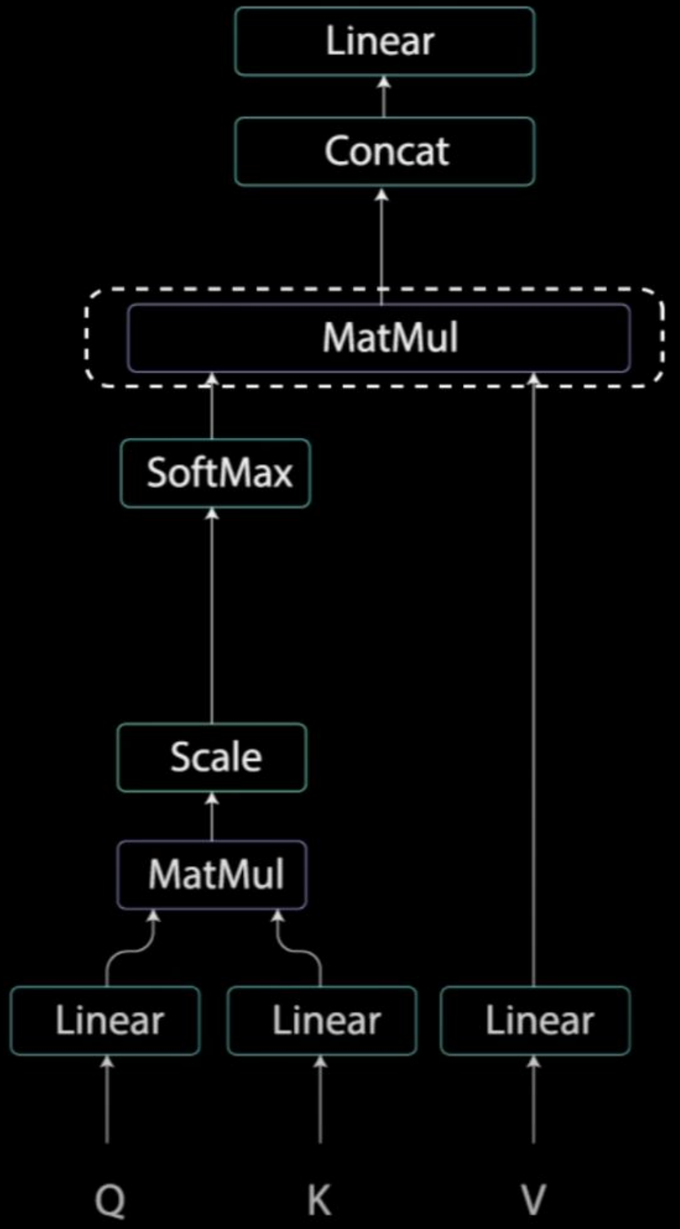
3.1. Self-Attention





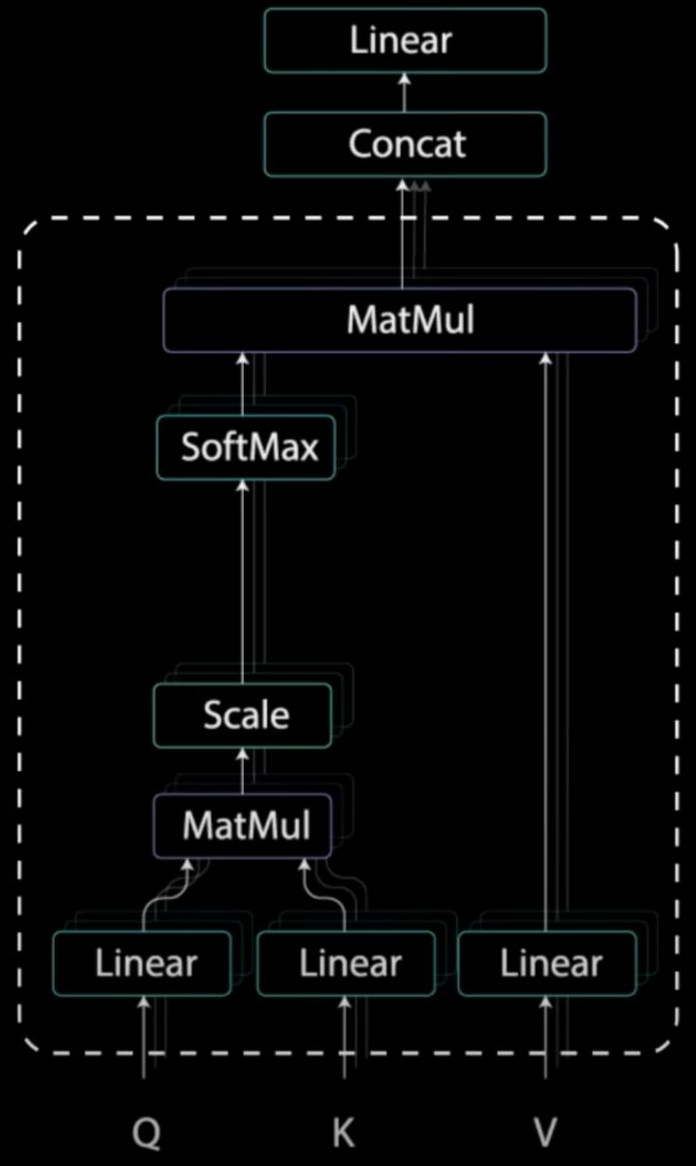
3. Multi-headed Attention

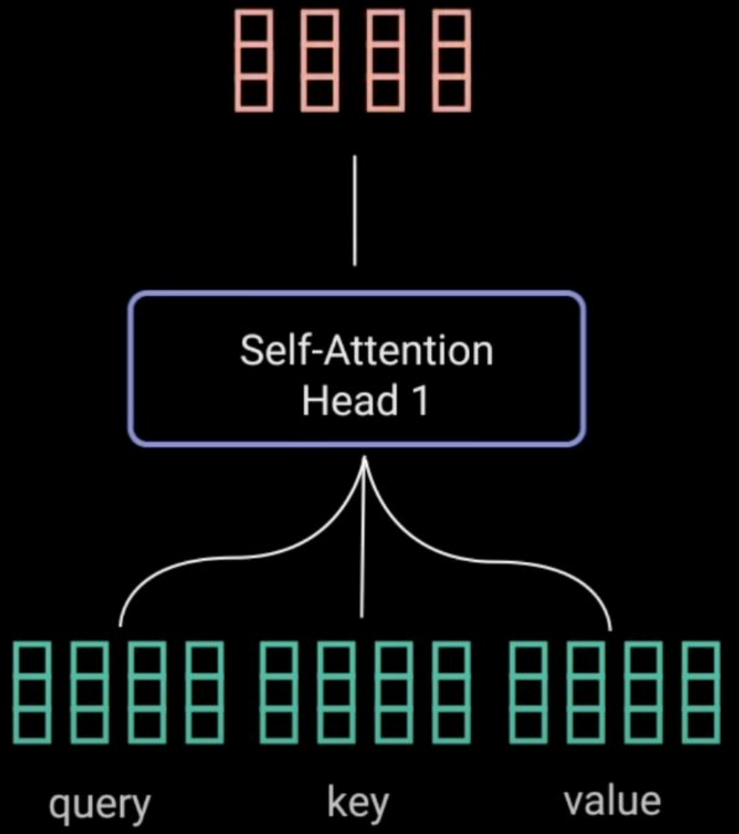
3.1. Self-Attention

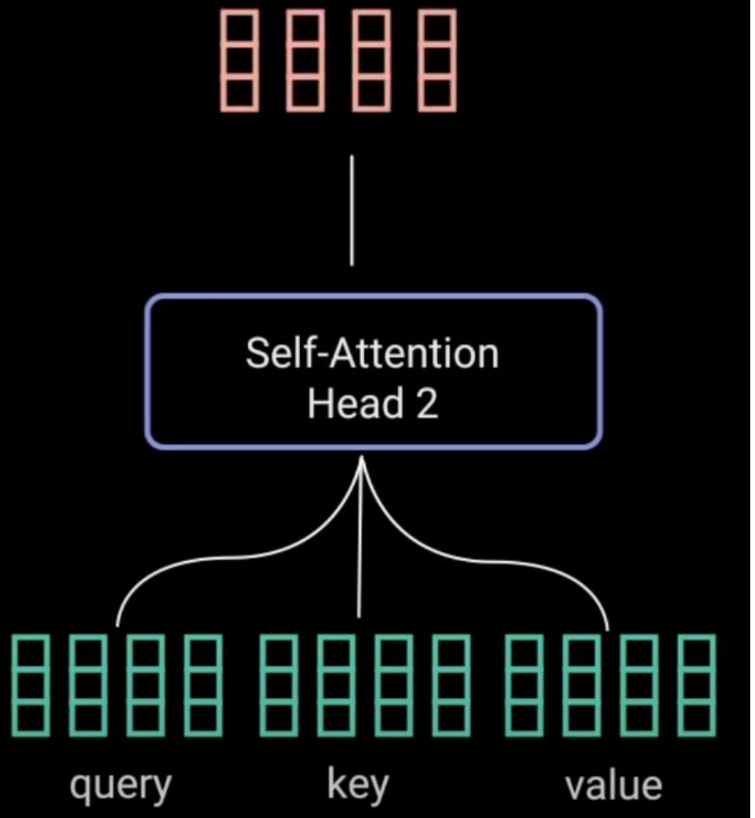




3. Multi-headed Attention

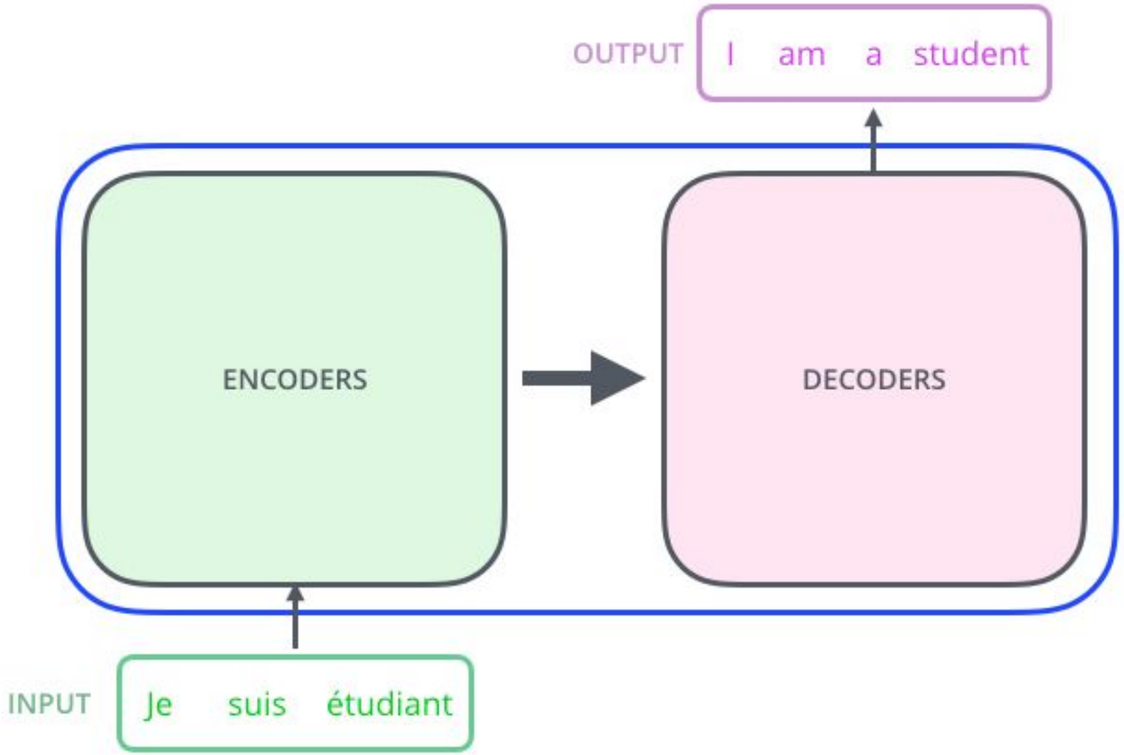


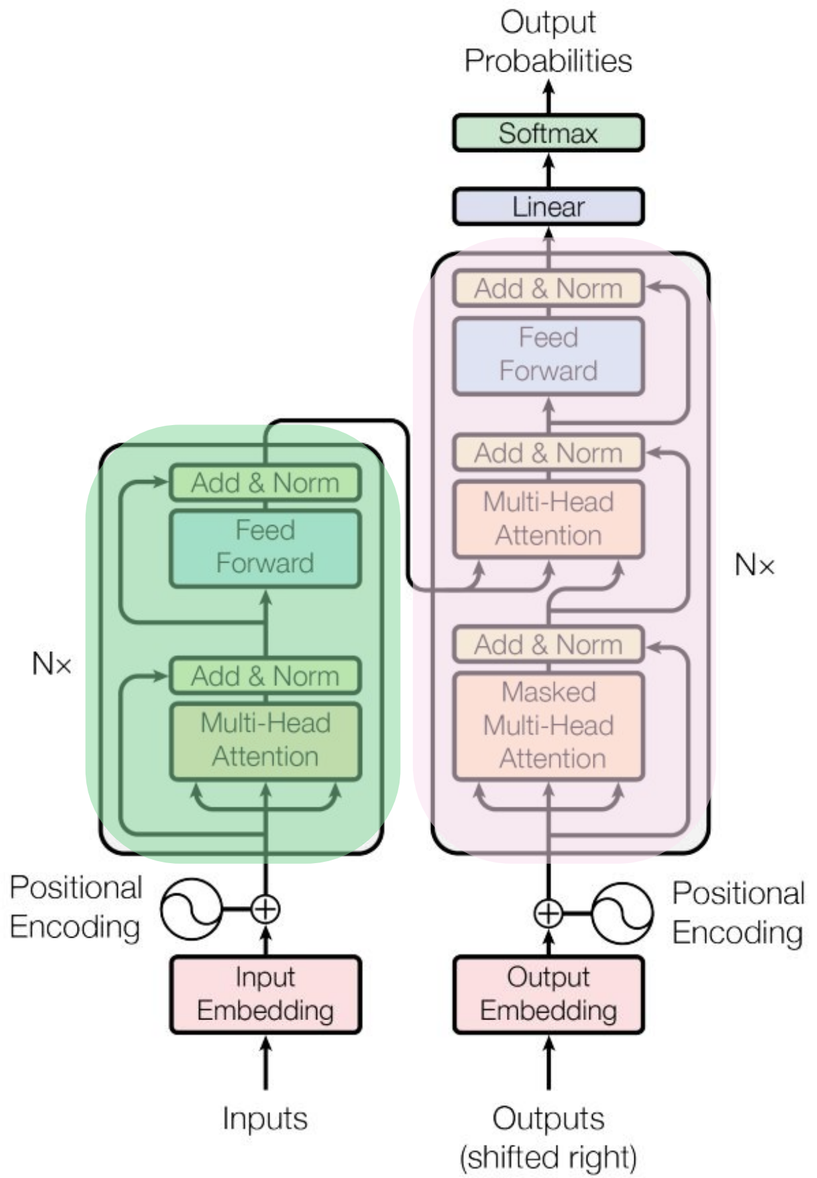




N = 2

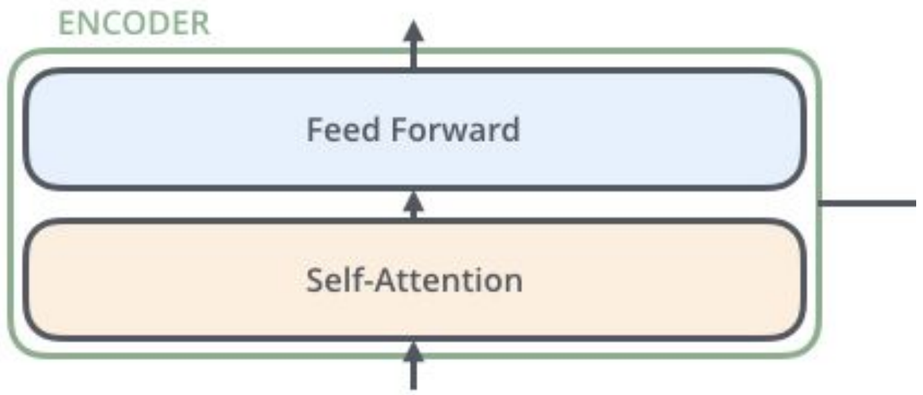
ARCHITECTURE





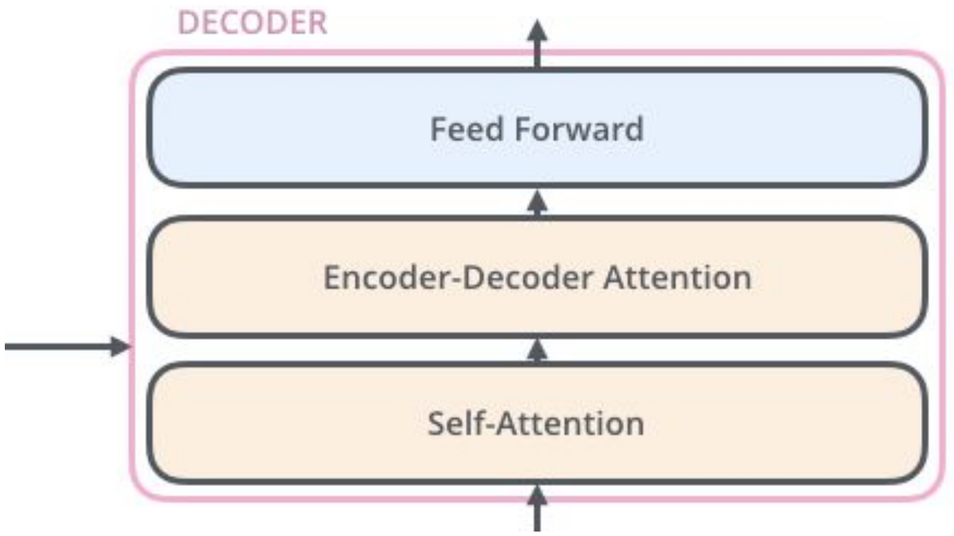
ENCODER STACKS

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.



DECODER STACKS

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

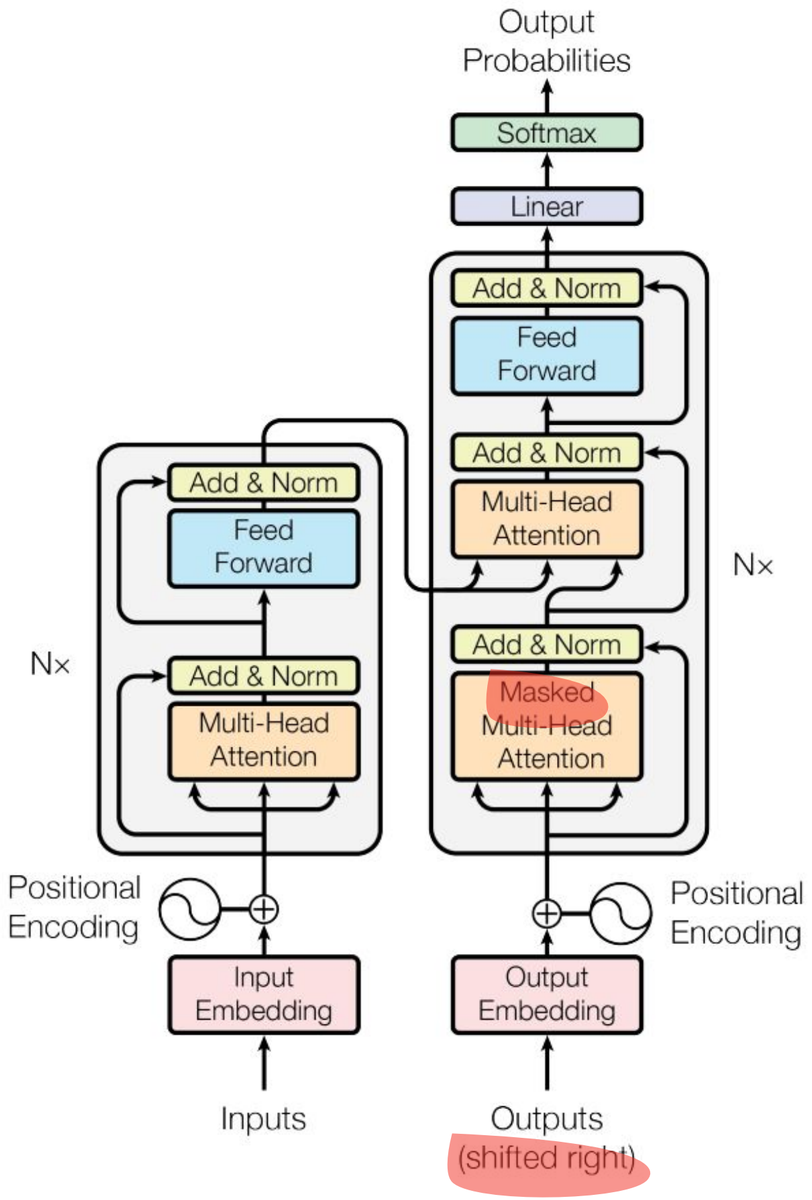


TRANSFORMERS

Offset + Masking

=

Next word prediction based only on previous words



DECODER STACKS

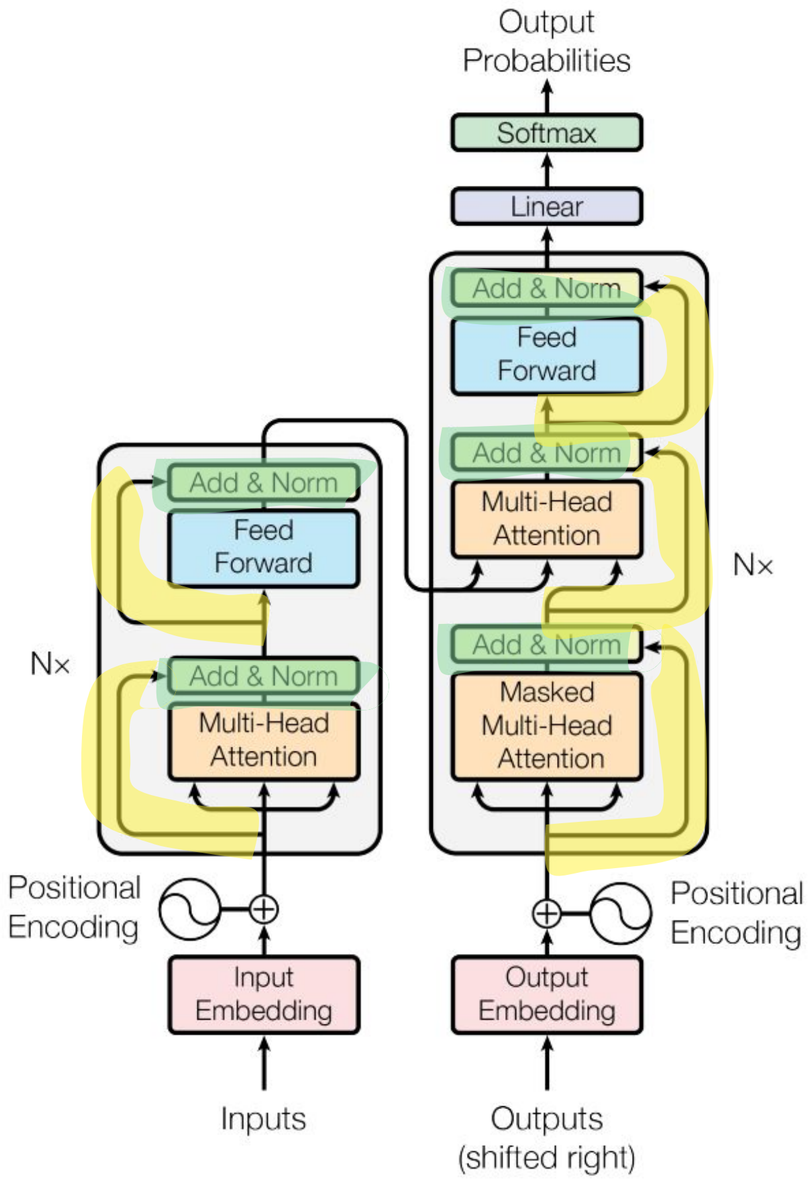
Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

TRANSFORMERS

Layer Normalization

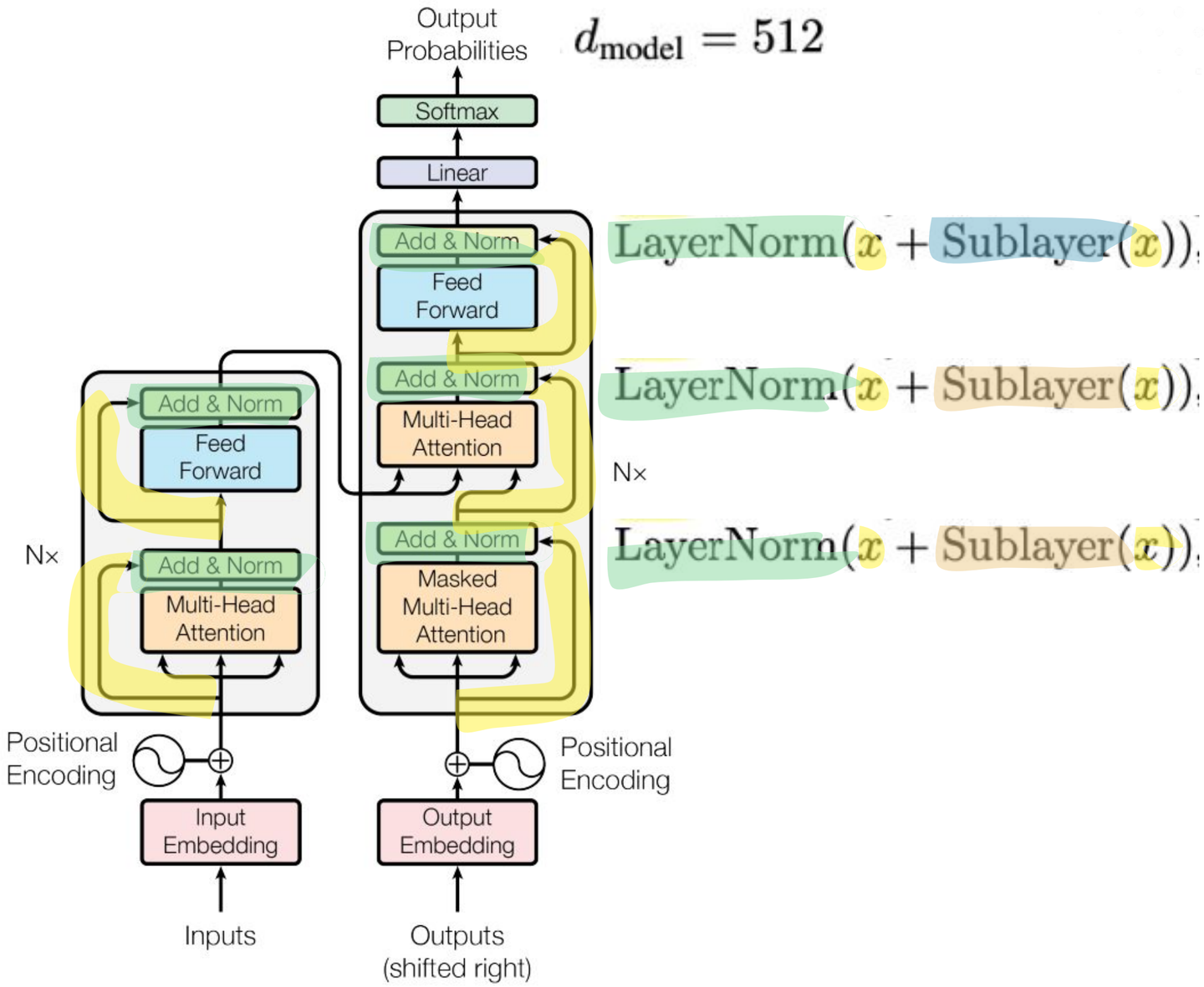
Residual Connection



TRANSFORMERS

Layer Normalization

Residual Connection



$d_{\text{model}} = 512$

$\text{LayerNorm}(x + \text{Sublayer}(x))$

$\text{LayerNorm}(x + \text{Sublayer}(x))$

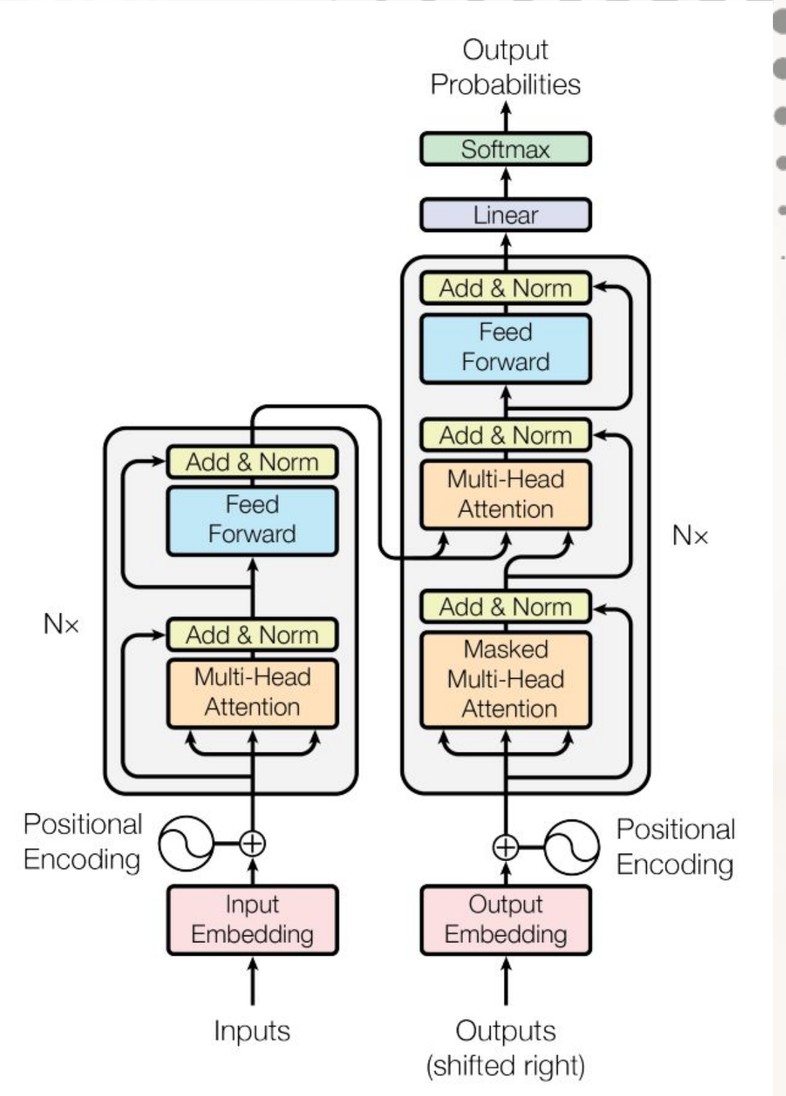
$\text{LayerNorm}(x + \text{Sublayer}(x))$

Recap:

- **Stacks** of encoders and decoders

- Encoder --> two sub-layers
  - **Self-Attention**
  - **Feed Forward**
    - **Residual** connection around each
    - Followed by L**ayer Norm**alization

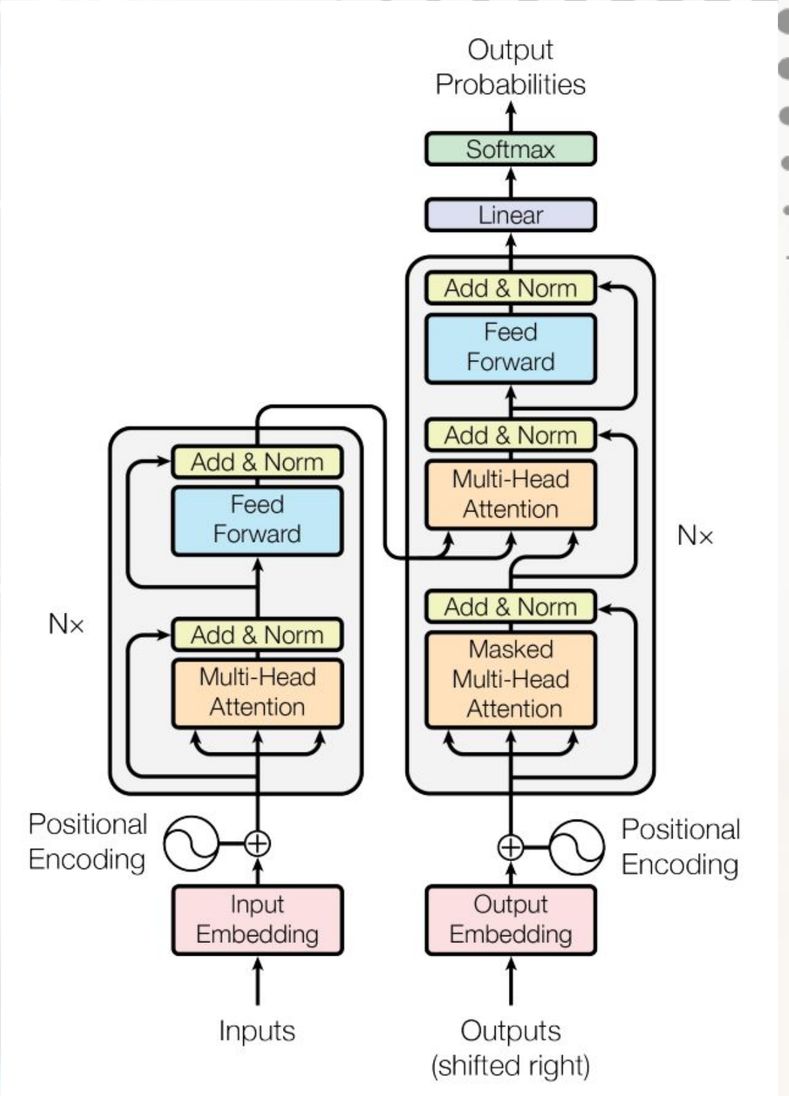
- Decoder -> same sub-layers



Still haven't covered:

- **I/O Embeddings**
- **Positional Encoding**
- **Output (Linear --> Softmax)**

These are (mostly) more classic ideas, but we will come back...

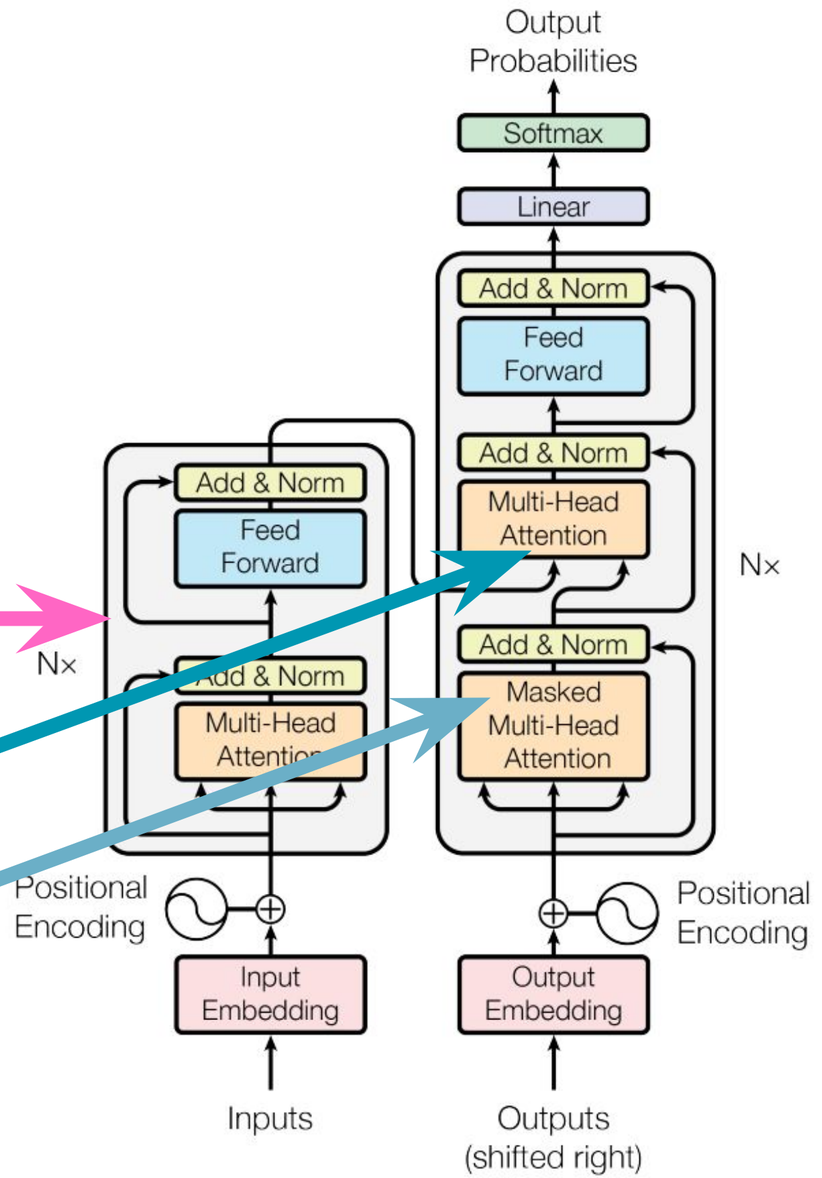




FOR NOW, LET’S ZOOM OUT

TRANSFORMER

- **Encoder**
  - Focused on **understanding** the **input**
- **Decoder**
  - Uses understanding from encoder
  - **Generates** one word at a time (**mask**)



ENCODER-DECODER = BART

- **Transformer Type**
  - Sequence to sequence
- **Model**-like
  - BART/T5-like
- **Focus**
  - Generative tasks that require an input
  - e.g. translation, summarization

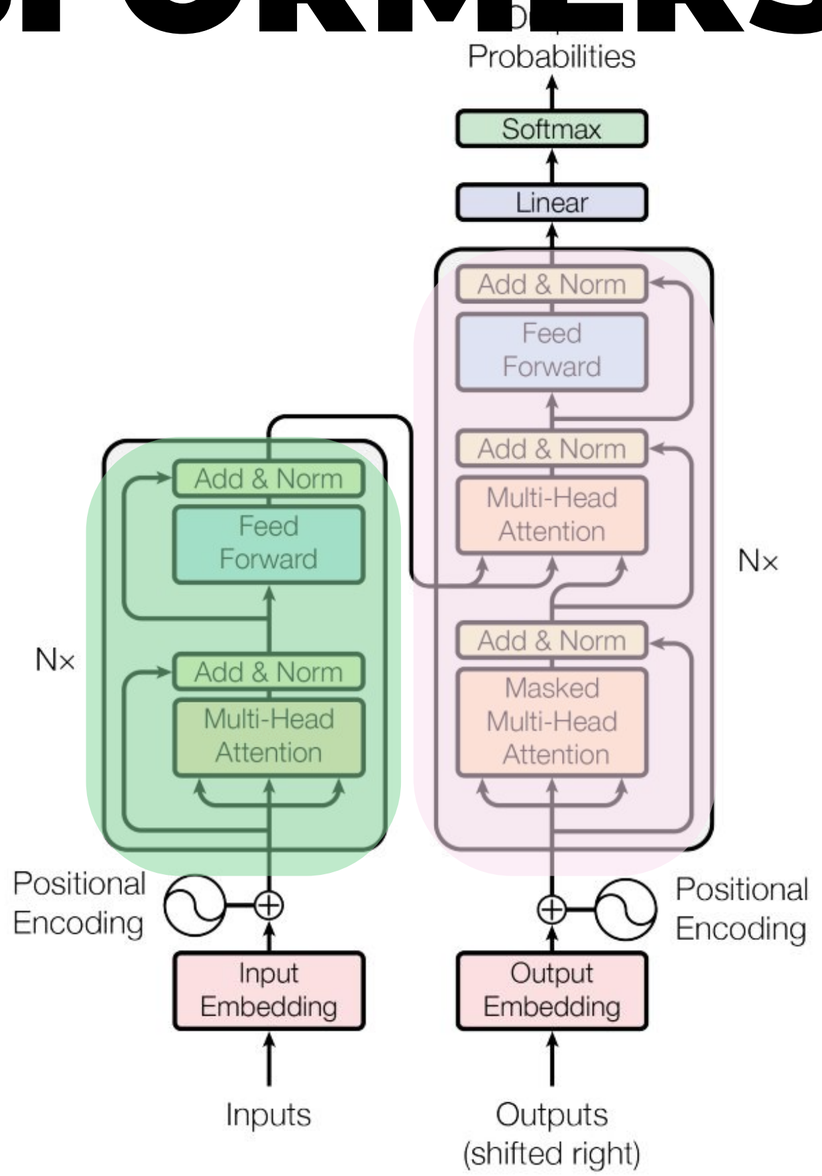
BART

- **Bidirectional AutoRegressive Transformer (BART)**

- Encoder-decoder (seq2seq) model
  - Bidirectional (BERT-like) encoder
  - Autoregressive (GPT-like) decoder

TRANSFORMERS

- BART-style



ENCODER = BERT

- **Transformer Type**
  - Auto-encoding
- **Model**-like
  - BERT-like
- **Focus**
  - Understanding of the input
  - e.g., question answering, sentiment analysis

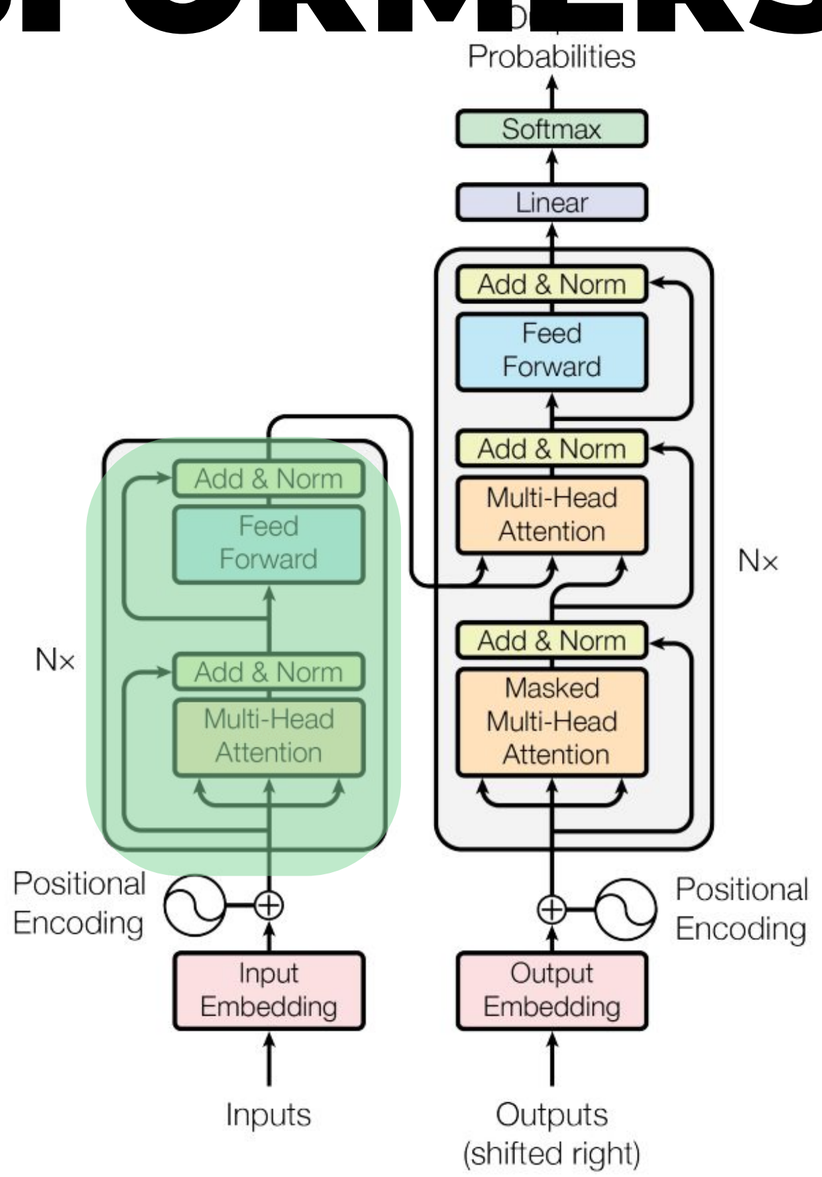
BERT

- **Bidirectional Encoder Representations from Transformers (BERT)**

- Bidirectional (BERT-like) encoder

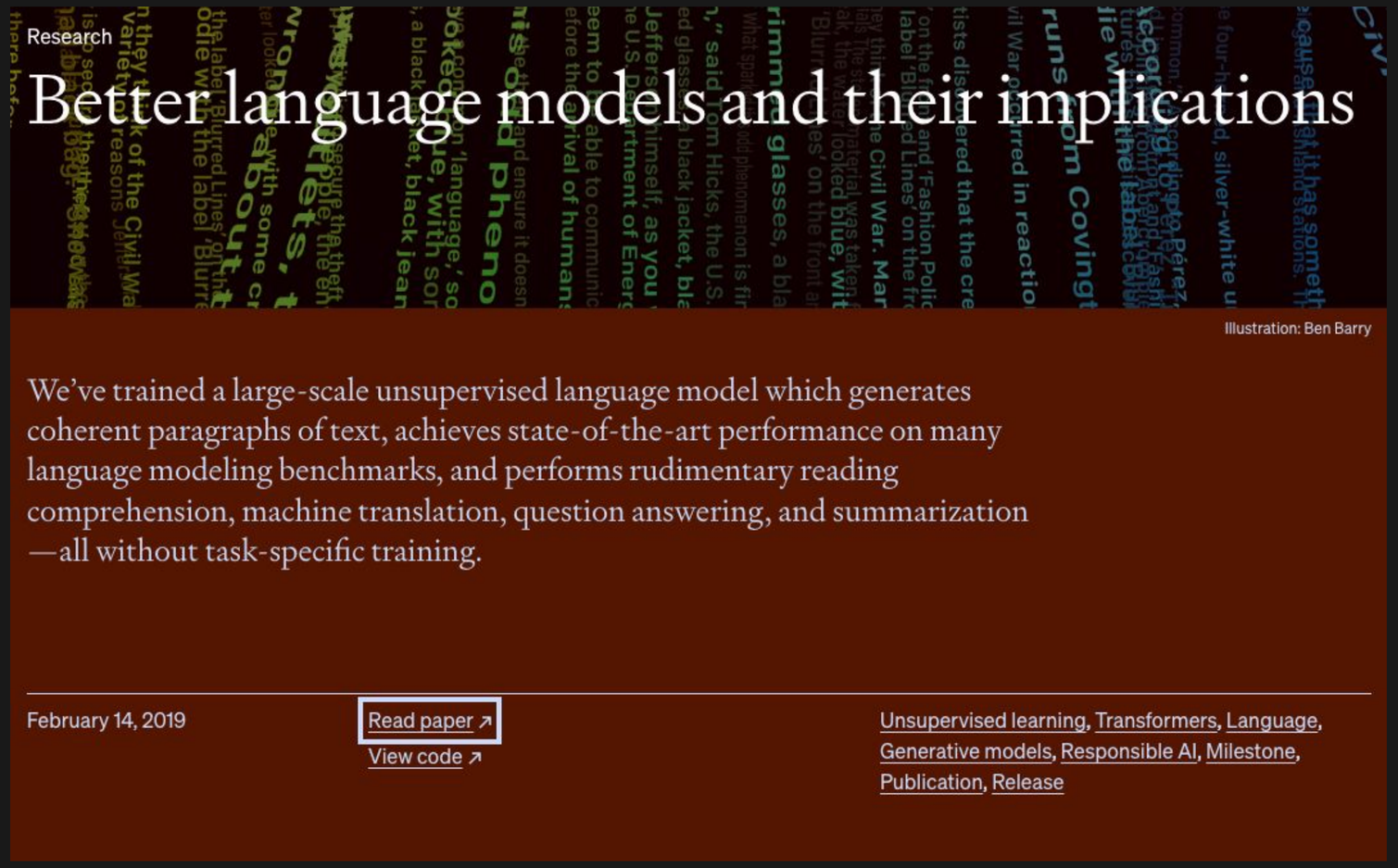
TRANSFORMERS

- BERT-style



DECODER = GPT

- **Transformer Type**
  - Auto-regressive
- **Model**-like
  - GPT-like
- **Focus**
  - Generative tasks
  - e.g., chat bot



GPT

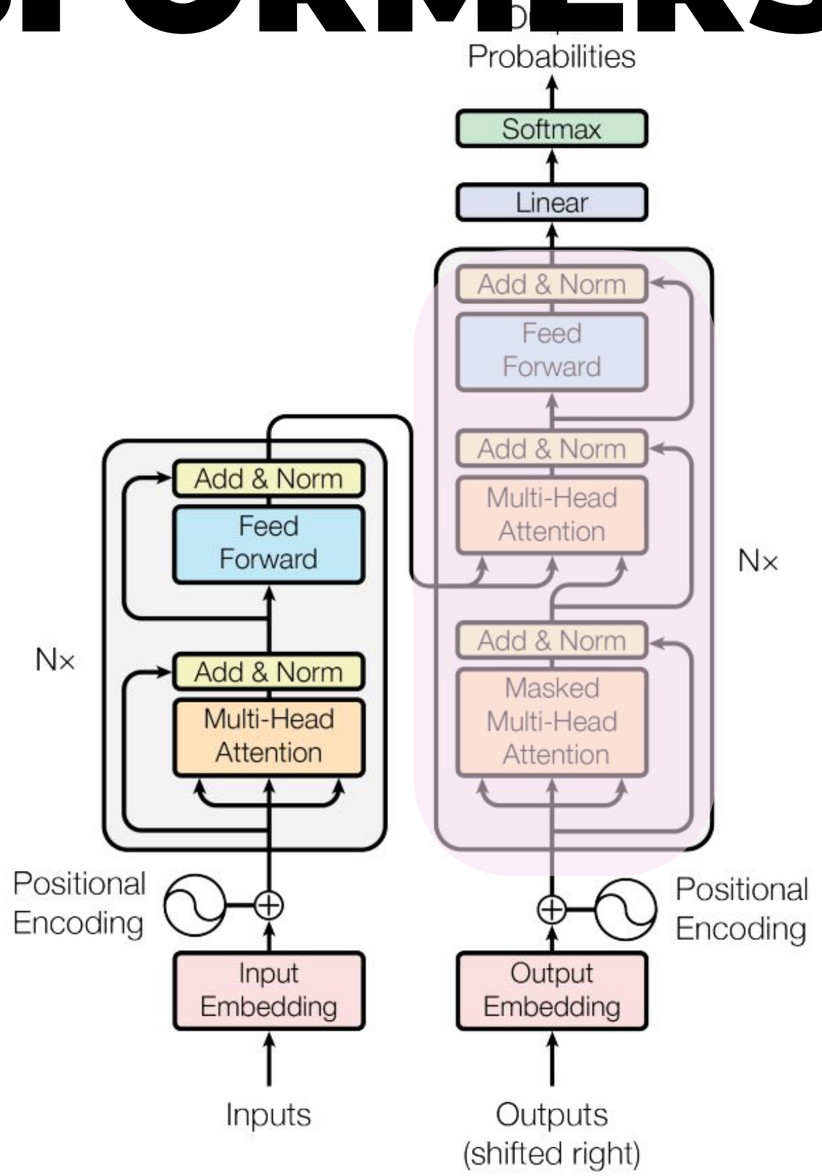
- **Generative Pre-Trained Transformer (GPT)**

- Autoregressive (GPT-like) decoder

- 12 Decoder blocks

TRANSFORMERS

- GPT-Style





ATTENTION

ATTENTION

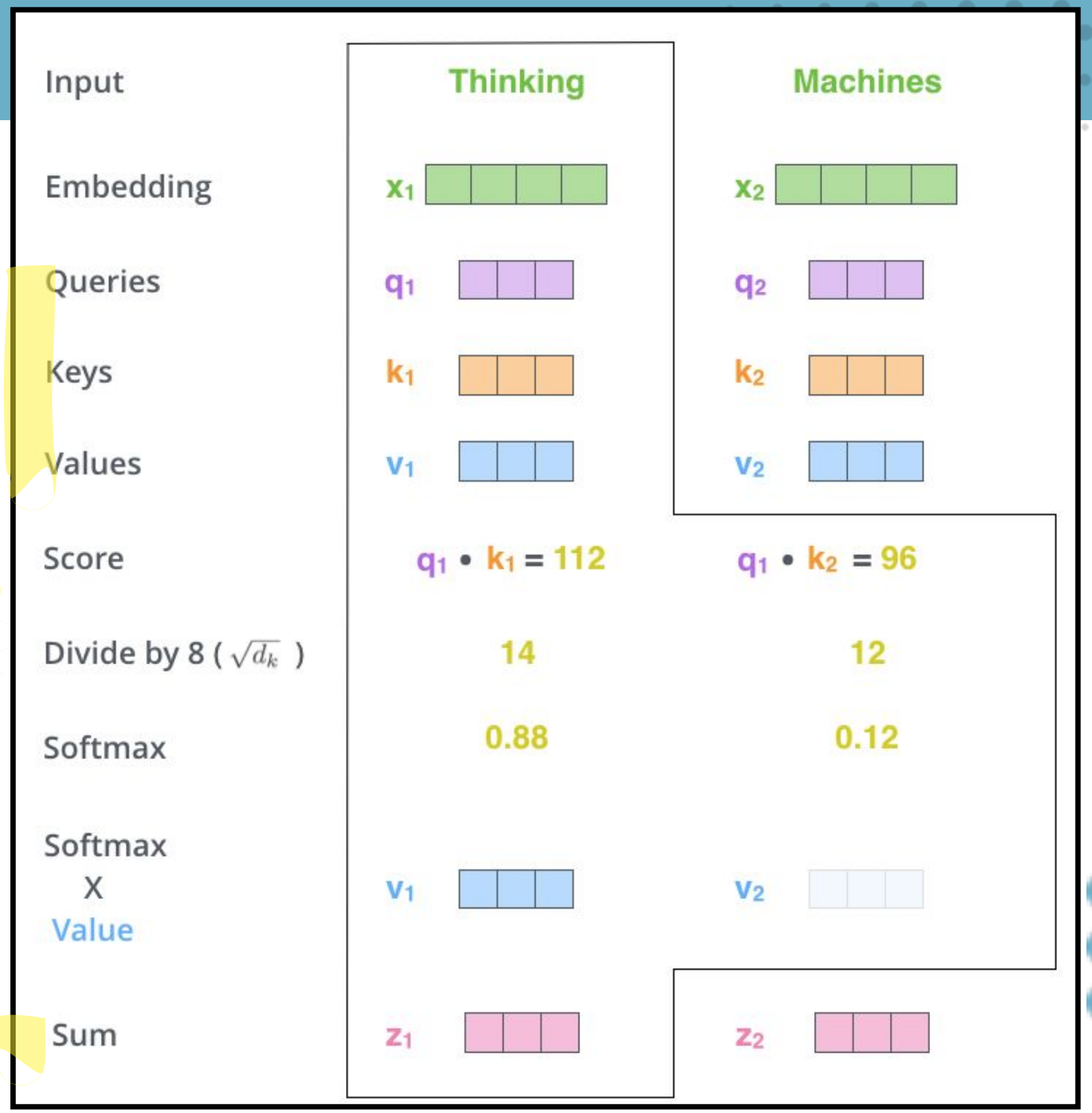
3.2 Attention

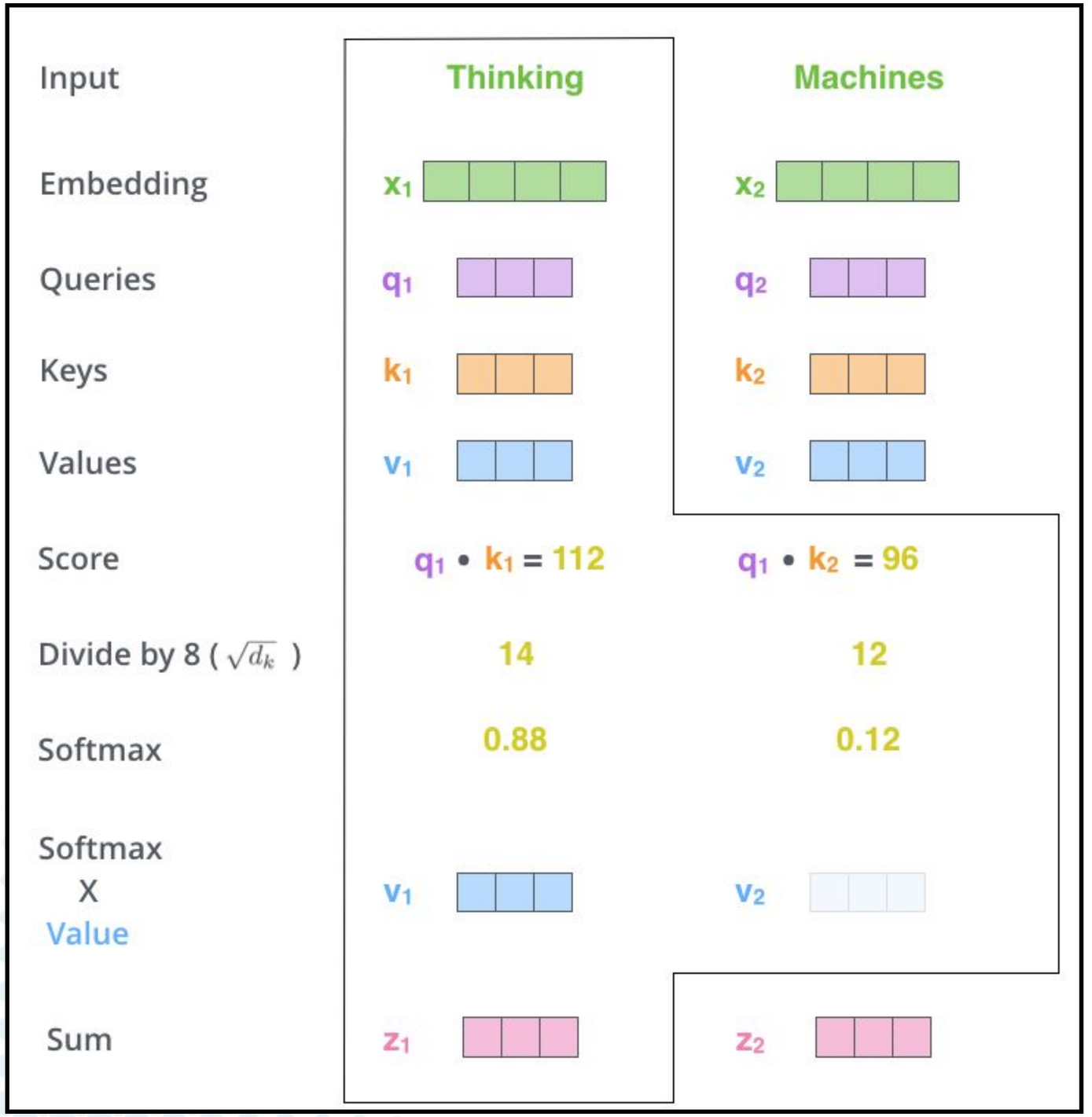
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

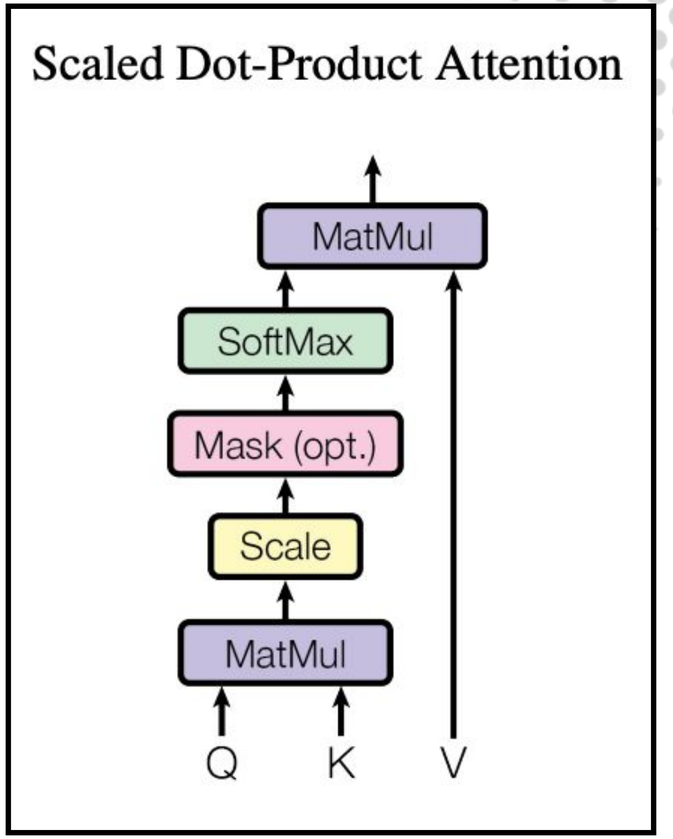
ATTENTION

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.



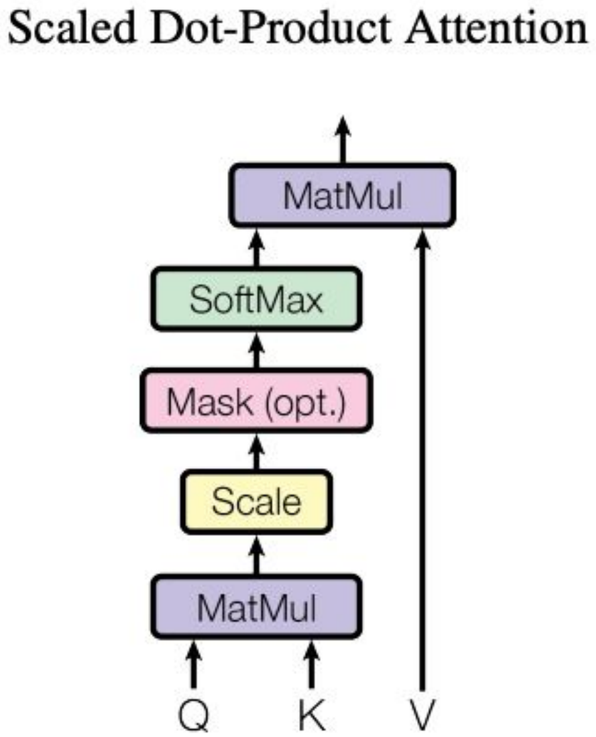




$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

BUT WE HAVE MORE THAN 1 WORD...

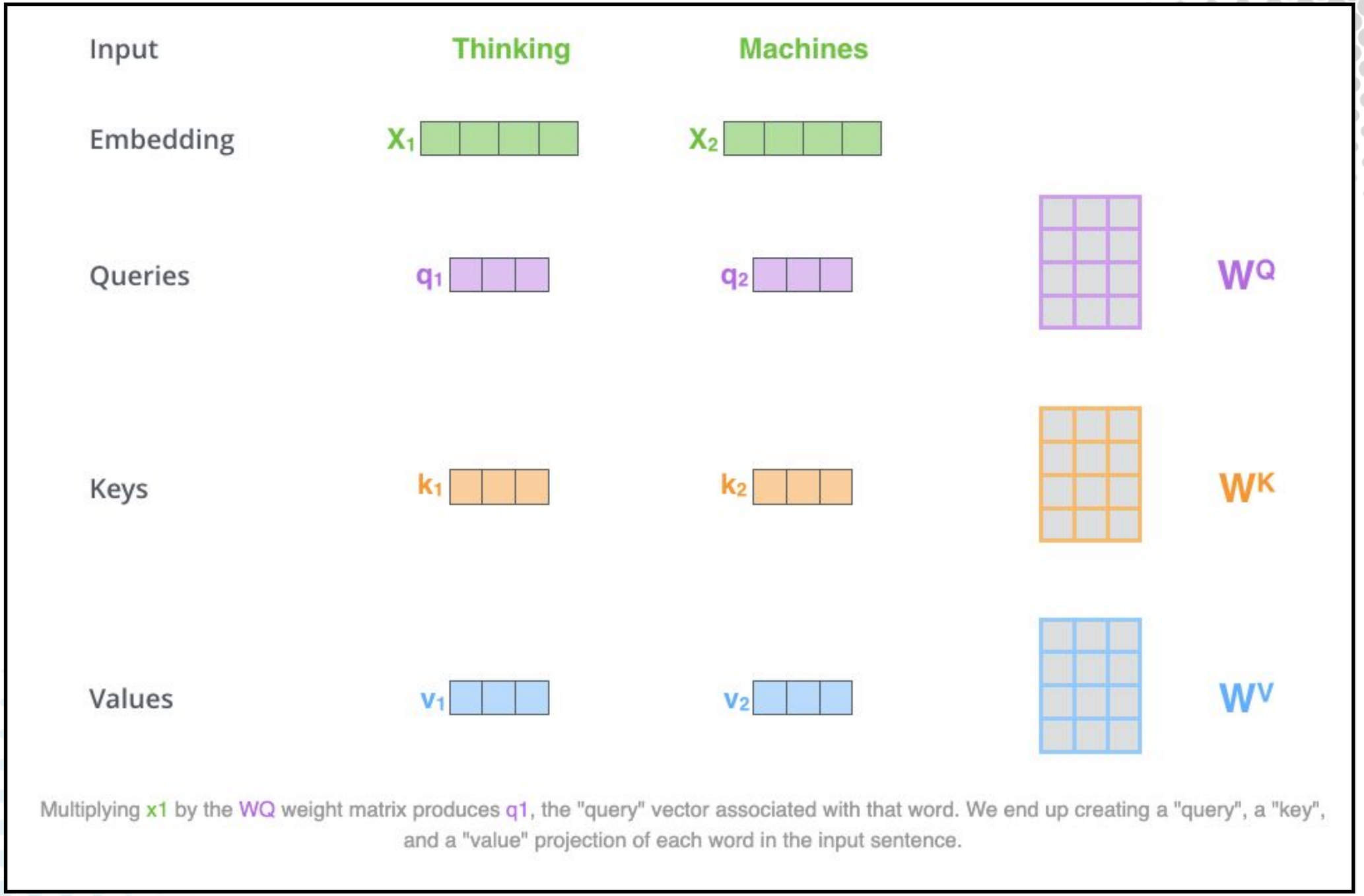
SCALED DOT-PRODUCT ATTENTION



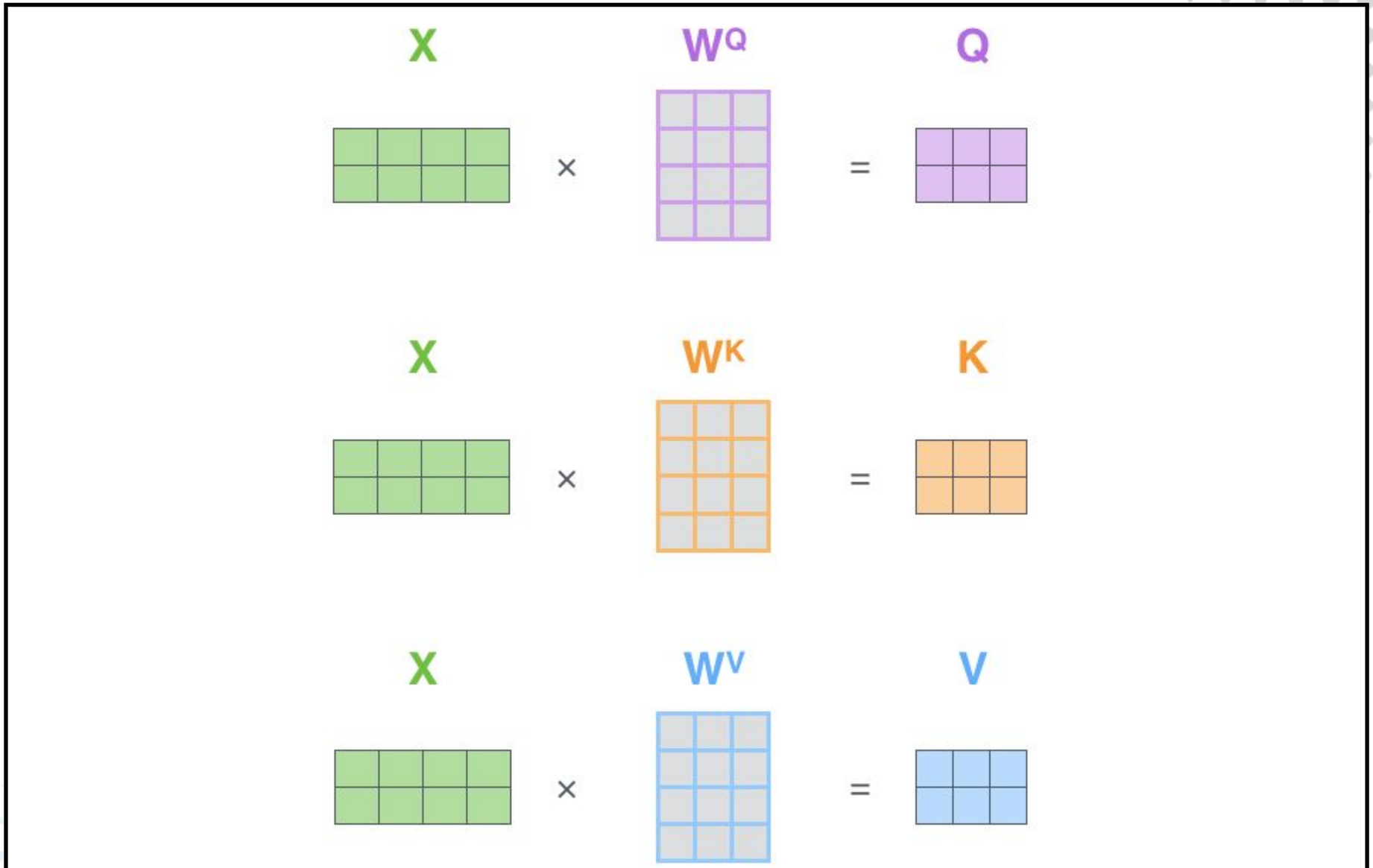
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

In practice, we **compute the attention function** on a set of queries simultaneously, packed together into a matrix Q.

(The keys and values are also packed together into matrices K and V)



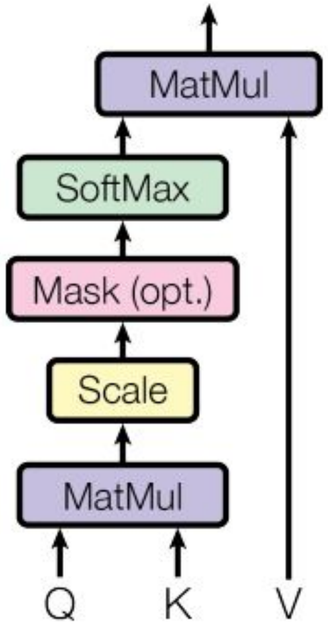
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

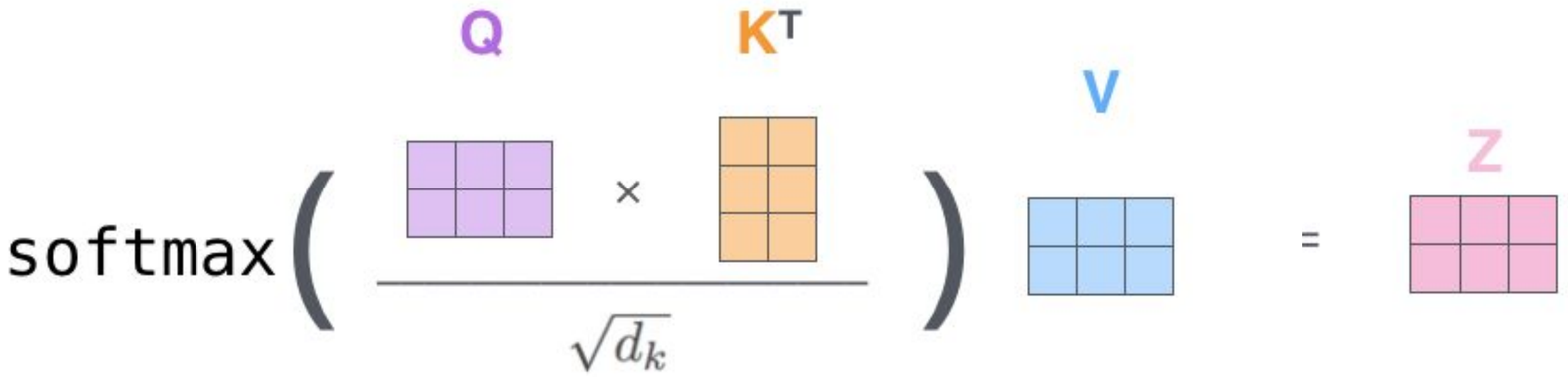


Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

MATRIX FORM

Scaled Dot-Product Attention

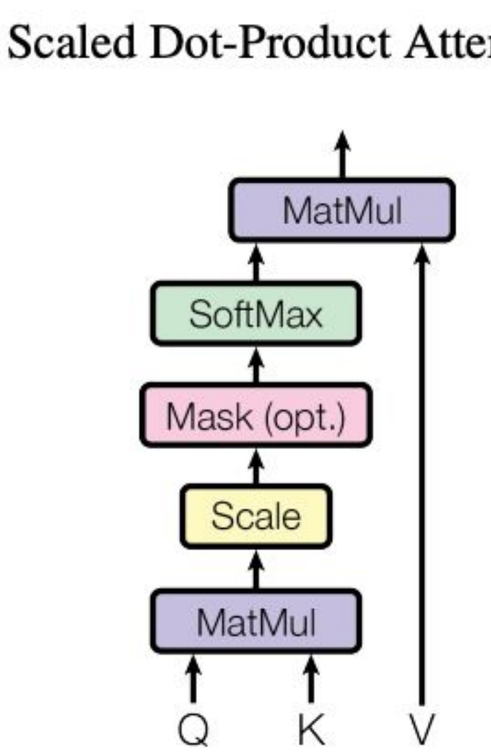


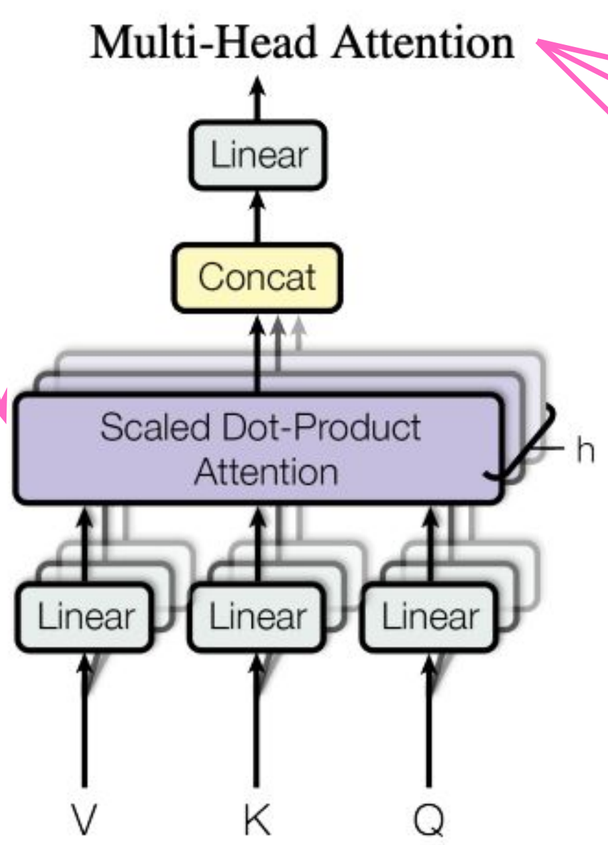


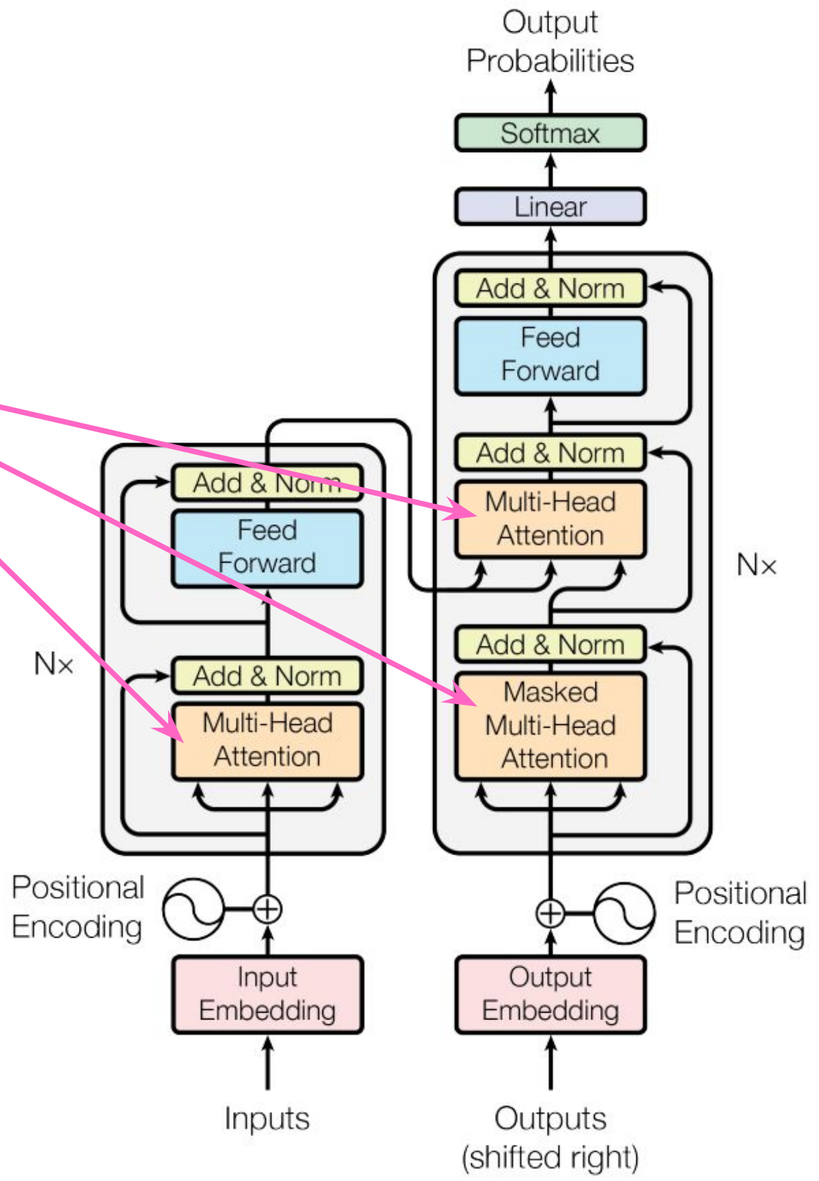
The self-attention calculation in matrix form

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

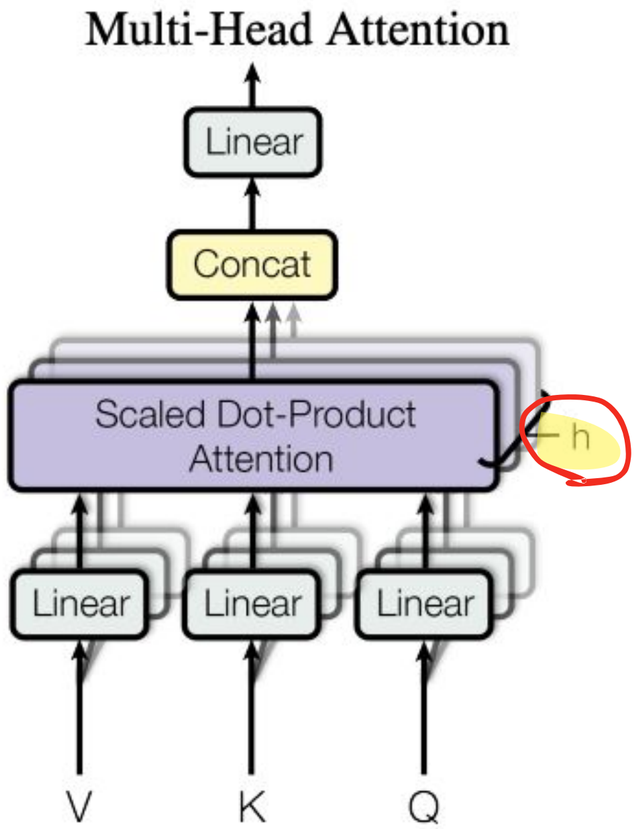
ATTENTION

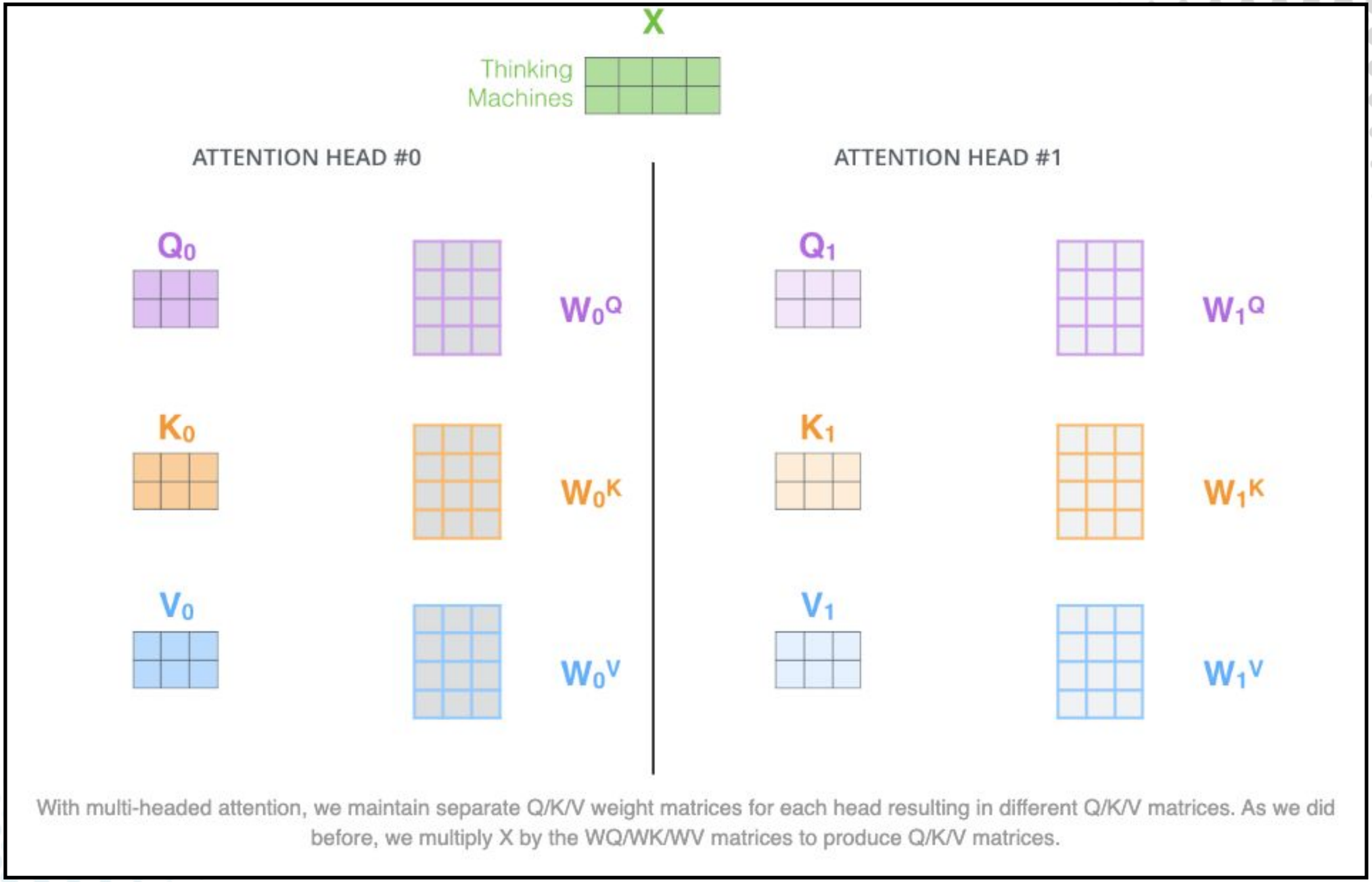

Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V

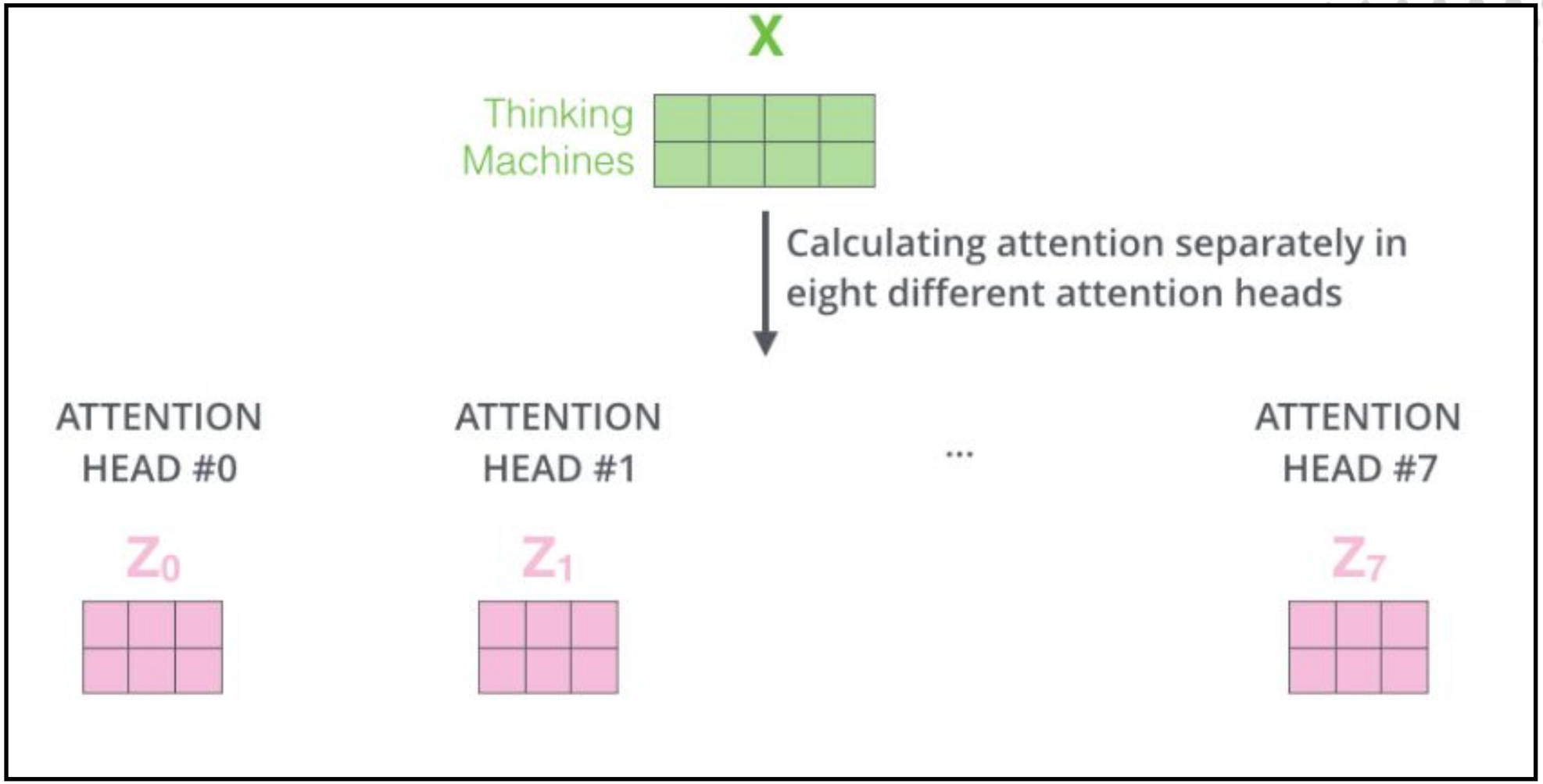

Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q


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

“MULTI-HEAD” ATTENTION




X
Thinking
Machines
ATTENTION HEAD #0
ATTENTION HEAD #1
Q_0
W_0^Q
K_0
W_0^K
V_0
W_0^V
Q_1
W_1^Q
K_1
W_1^K
V_1
W_1^V
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.


X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7

1) Concatenate all the attention heads



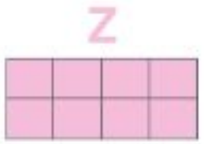
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

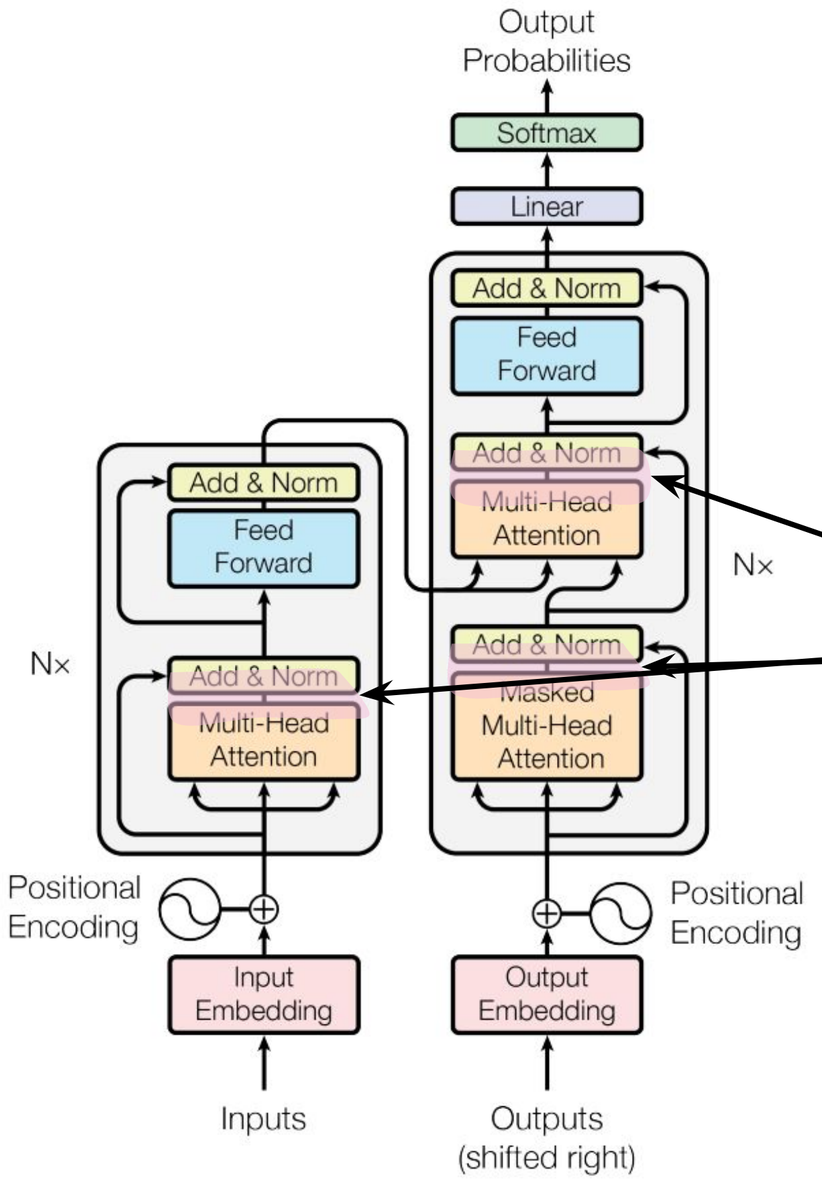


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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.

TRANSFORMERS

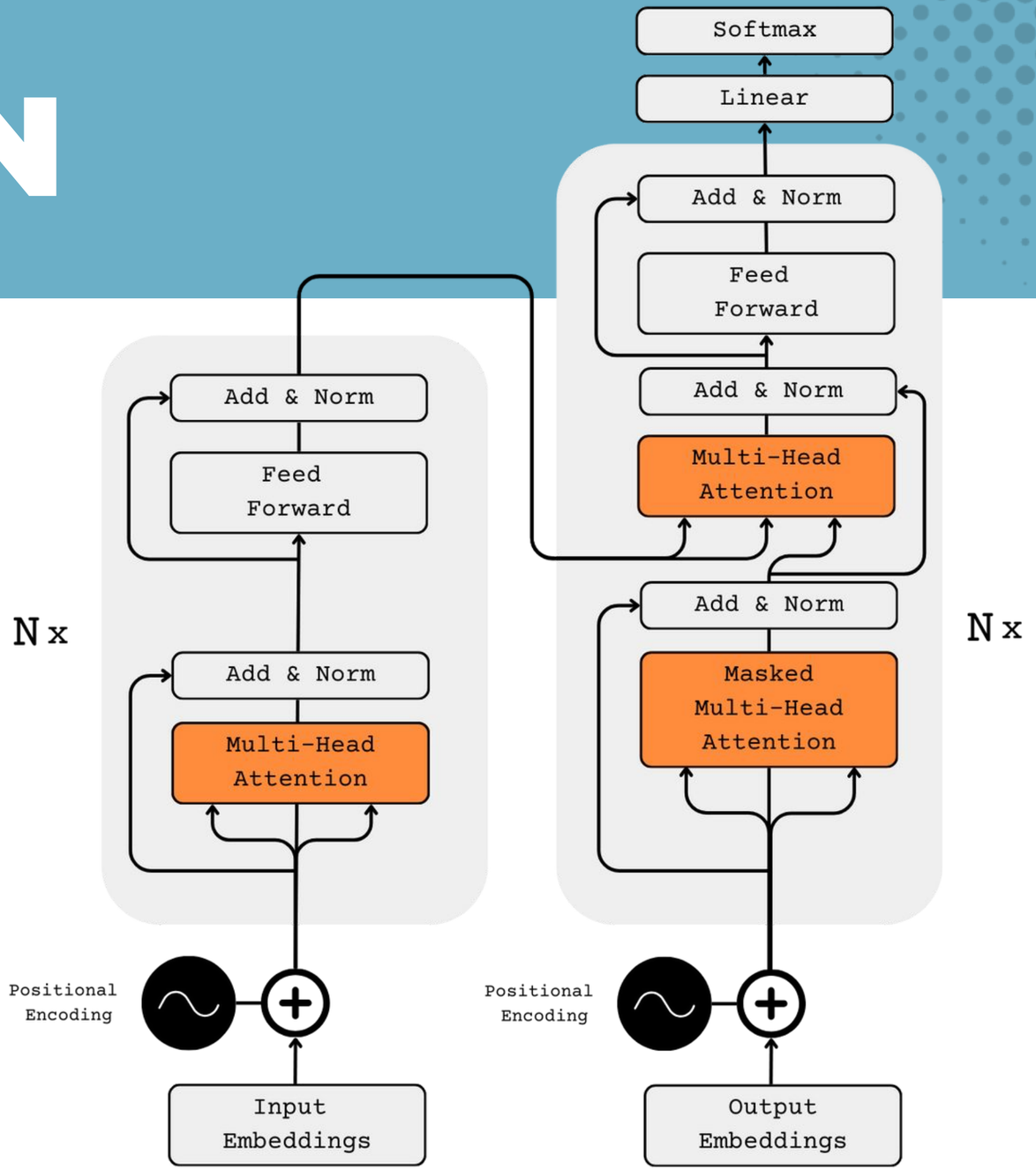


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Z

ATTENTION


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N x
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N x
Positional Encoding
Positional Encoding
Input Embeddings
Output Embeddings



CODING ATTENTION!

Presented by
Chris Alexiuk, LLM Wizard 🪄

3.2.3 Applications of Attention in our Model

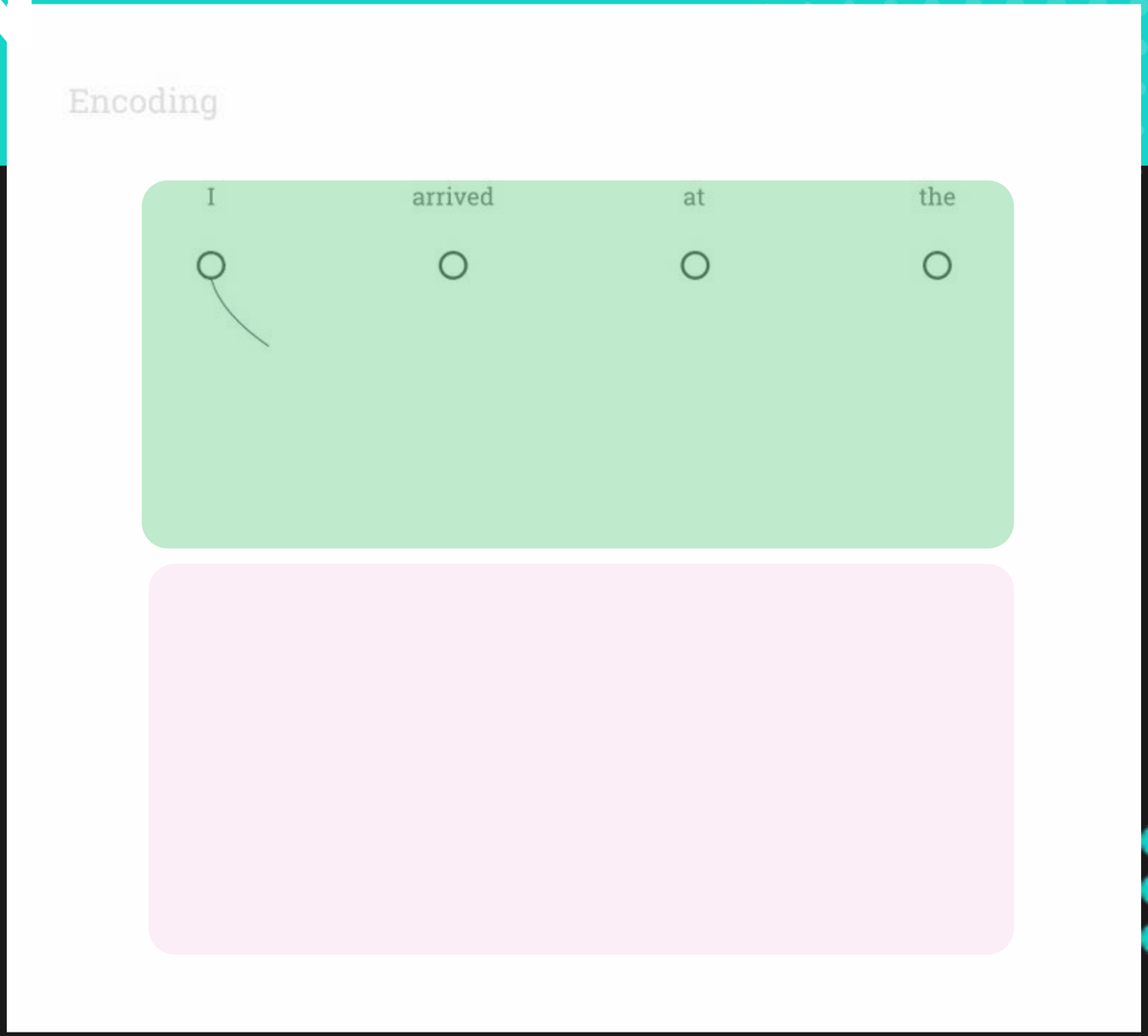
The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

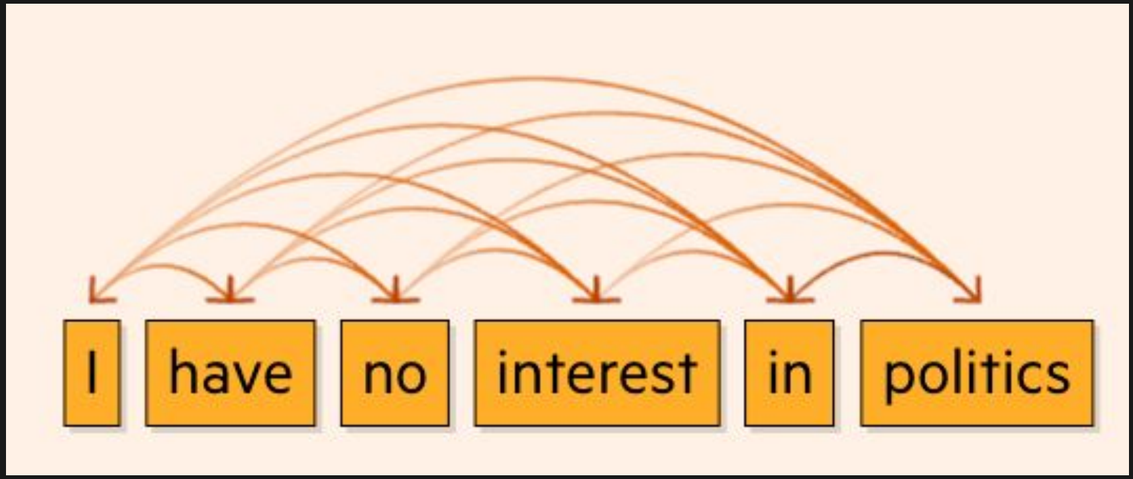
LET’S ZOOM OUT AGAIN FOR A SECOND

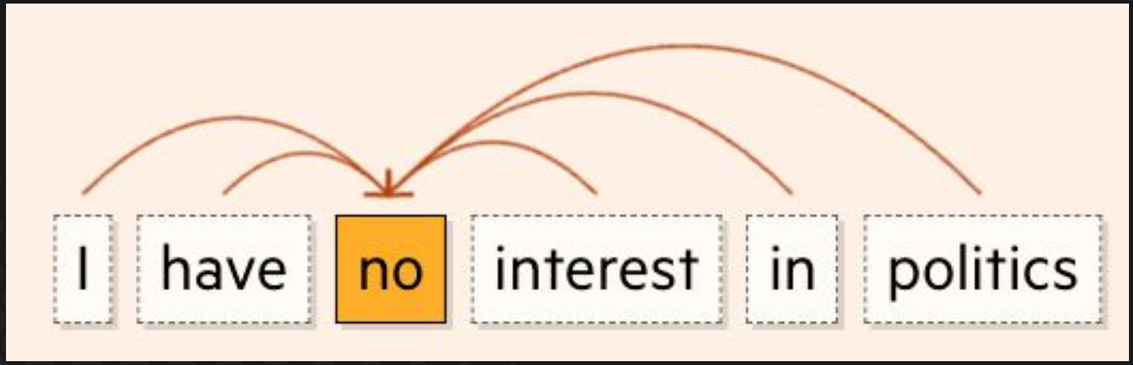
SELF-ATTENTION

- Applying Transformer to machine translation



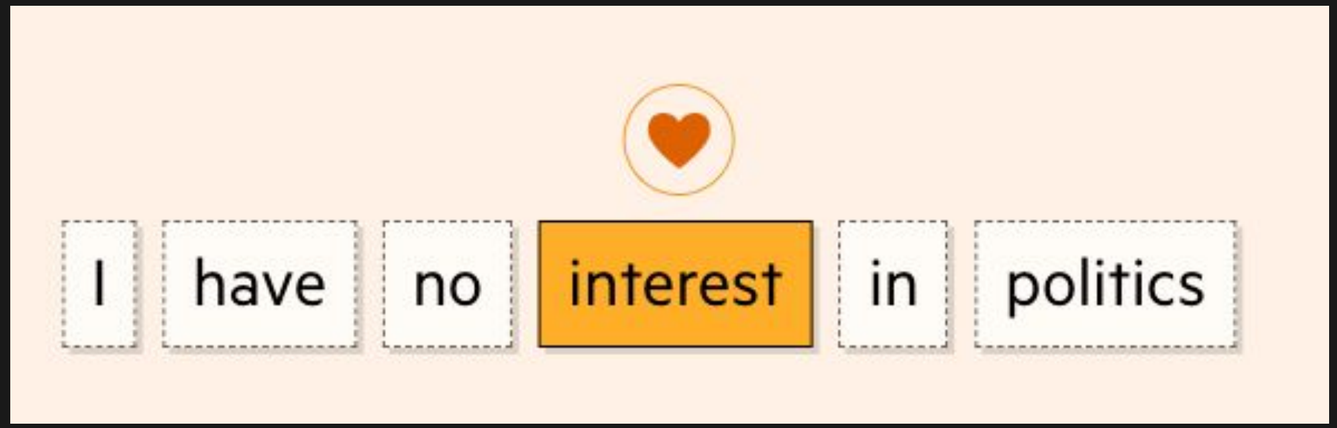
SELF-ATTENTION

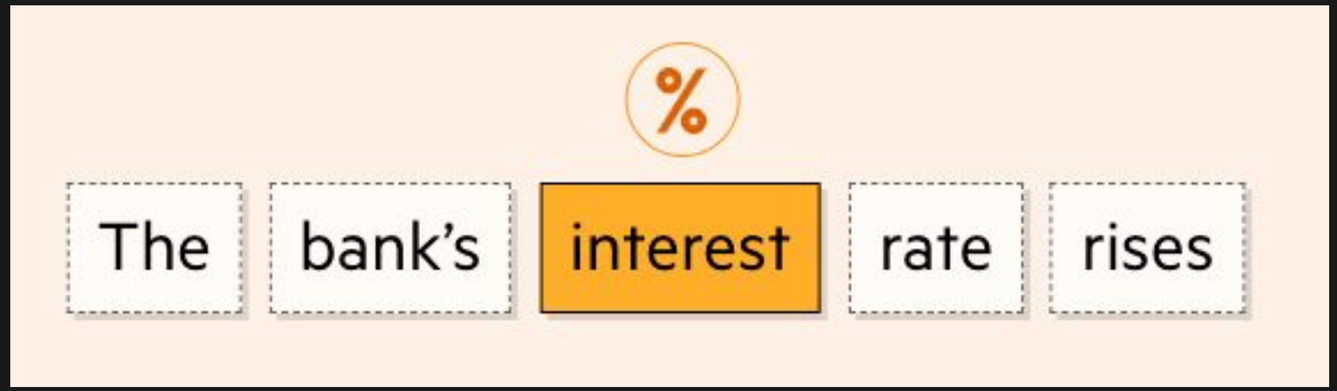




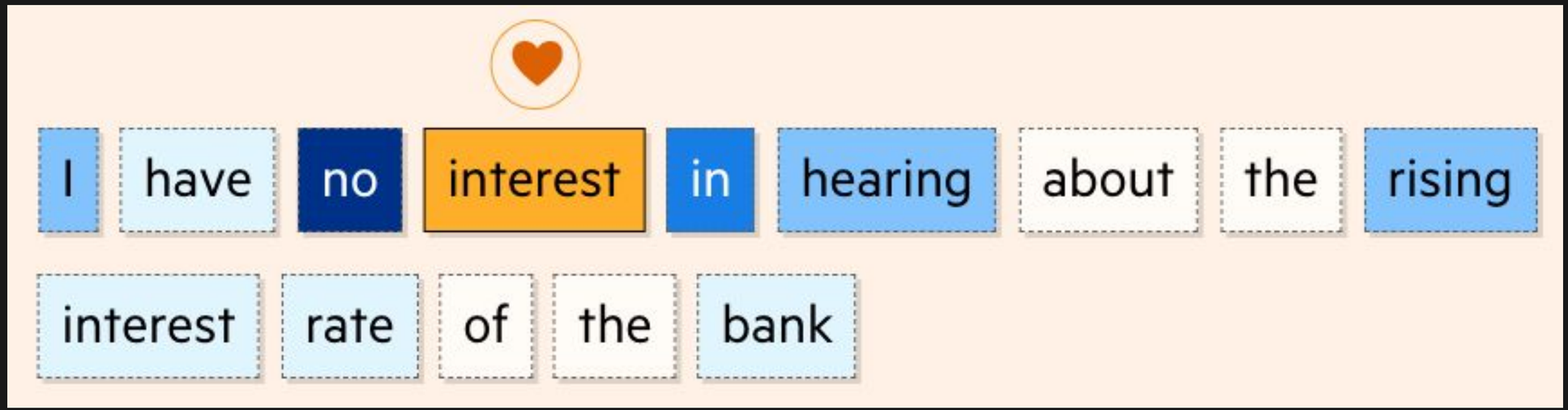
Self-attention looks at each **token** in a body of text and decides which others are most important to understanding its meaning.

SELF-ATTENTION



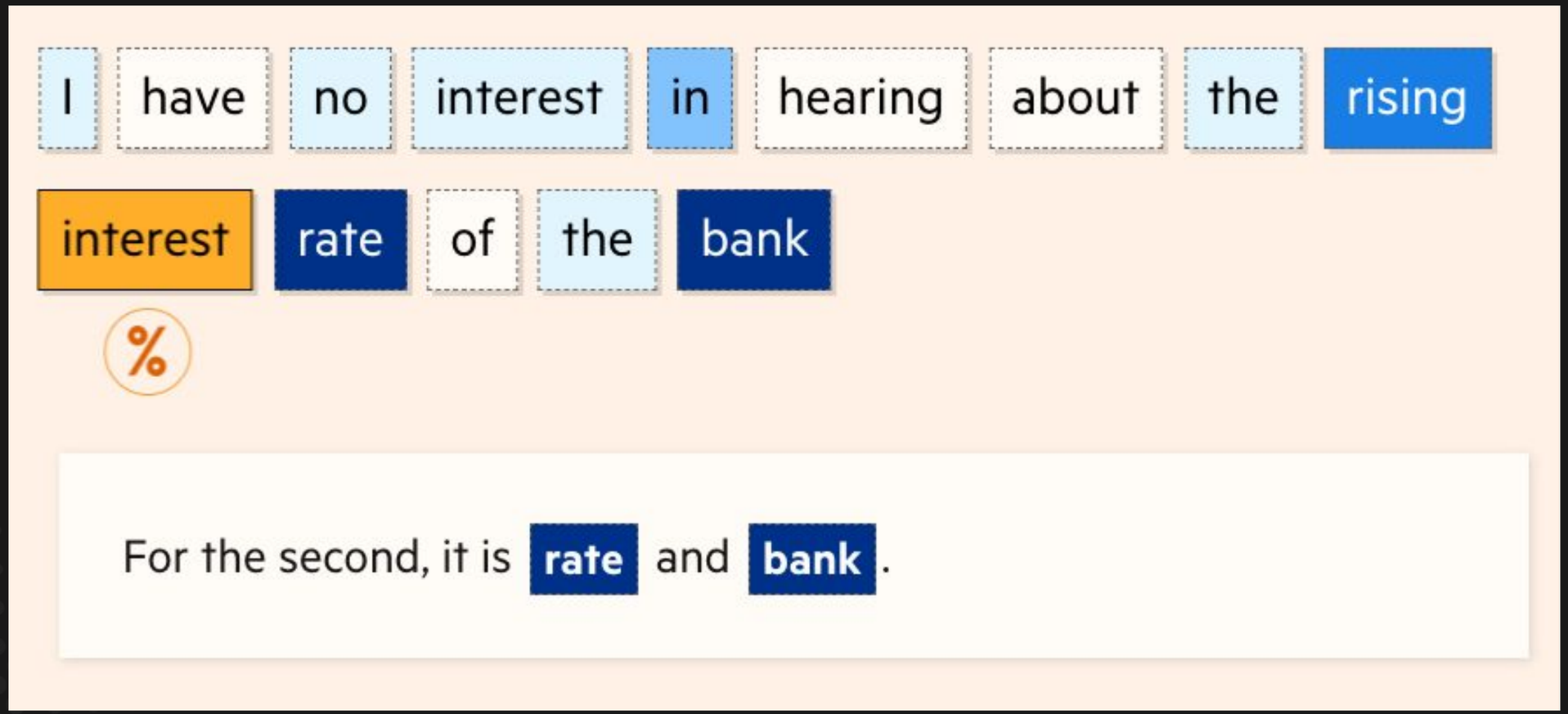


SELF-ATTENTION



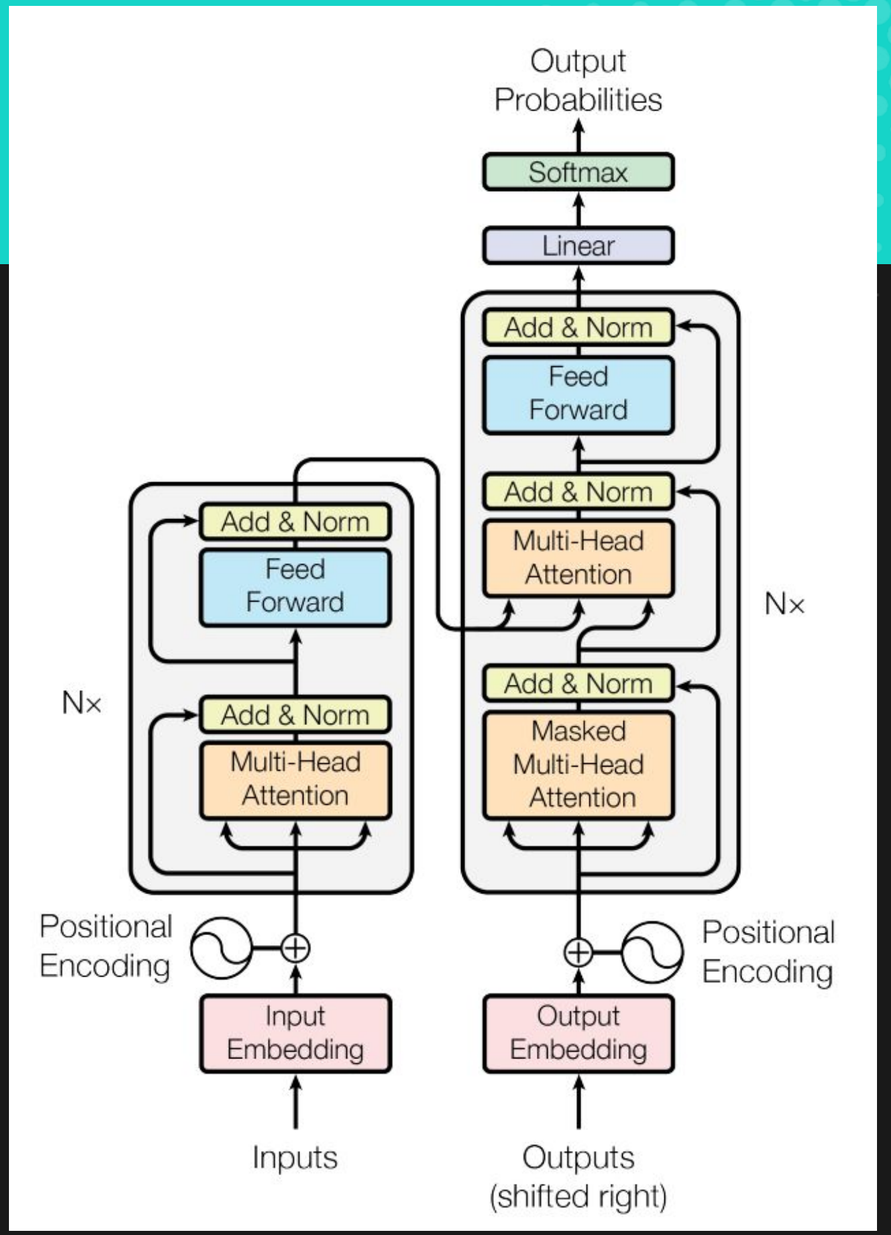
For the first use of interest, it is **no** and **in** that are most attended.

SELF-ATTENTION

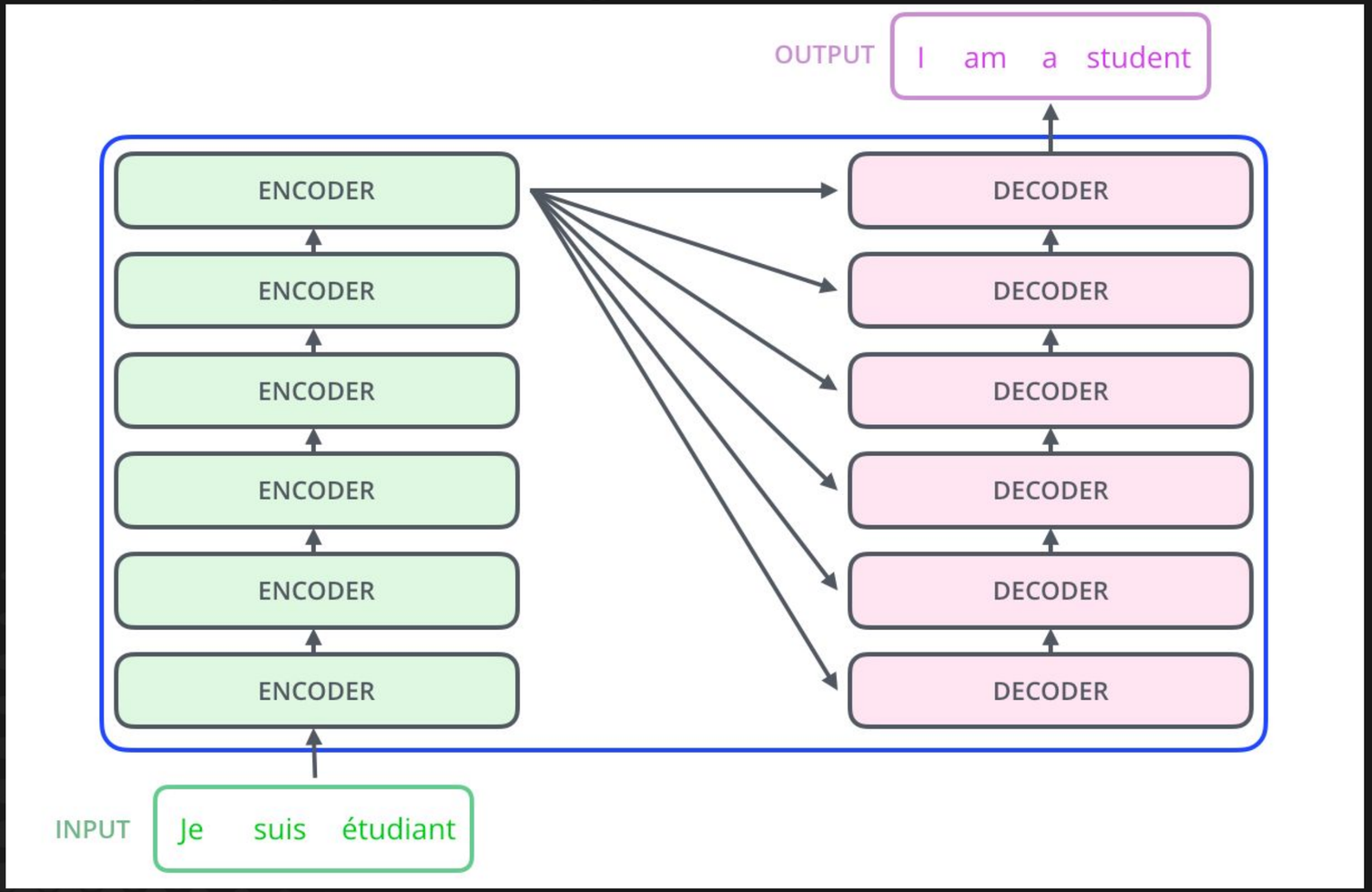


TRANSFORMER





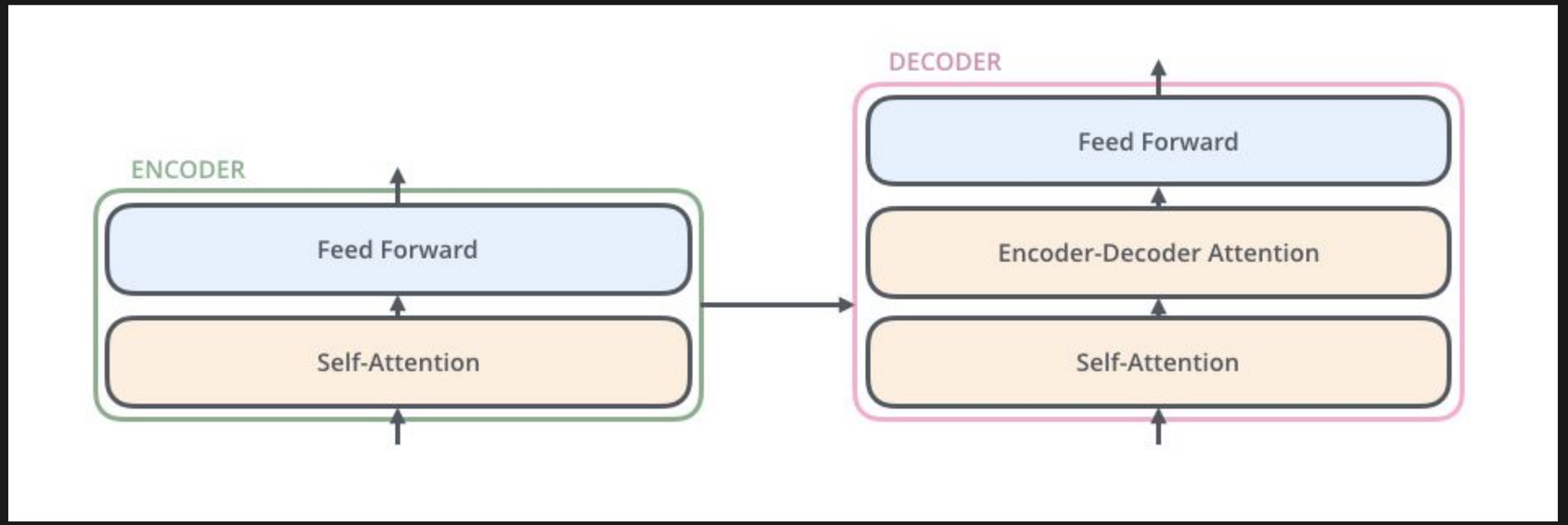
STACKS OF ENCODERS & DECODERS





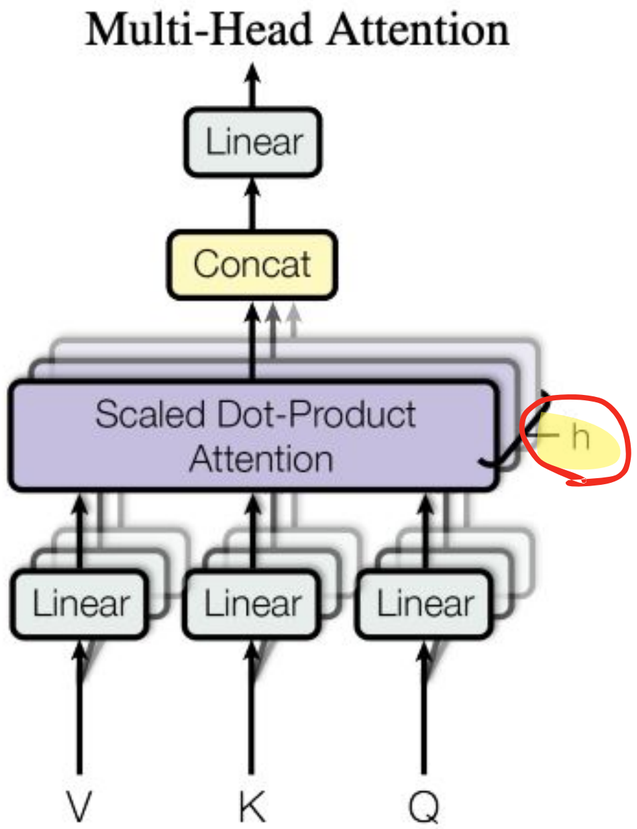
WITHIN EACH ENCODER & DECODER

- **Self-Attention**
  - Helps encoder look at other words in input sentence as it encodes a specific word

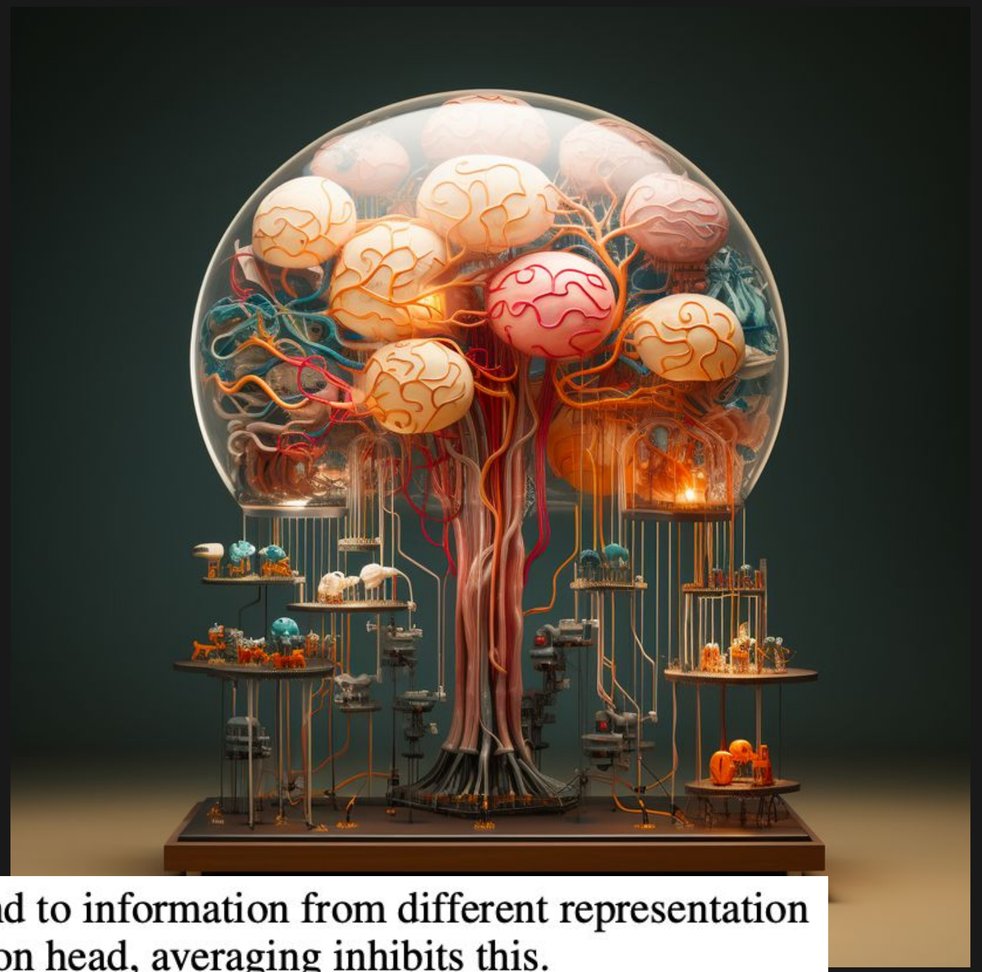


- **Encoder-Decoder Attention**
  - Helps decoder focus on relevant parts of input

“MULTI-HEAD” ATTENTION



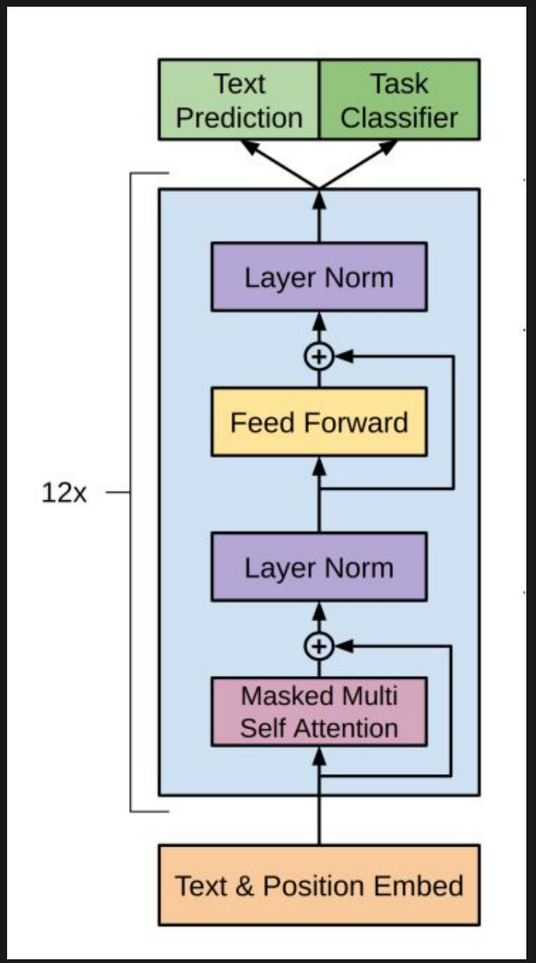
ATTENTION HEADS



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.

ATTENTION HEADS

- Many **blocks** (12)
- Many attention **heads**



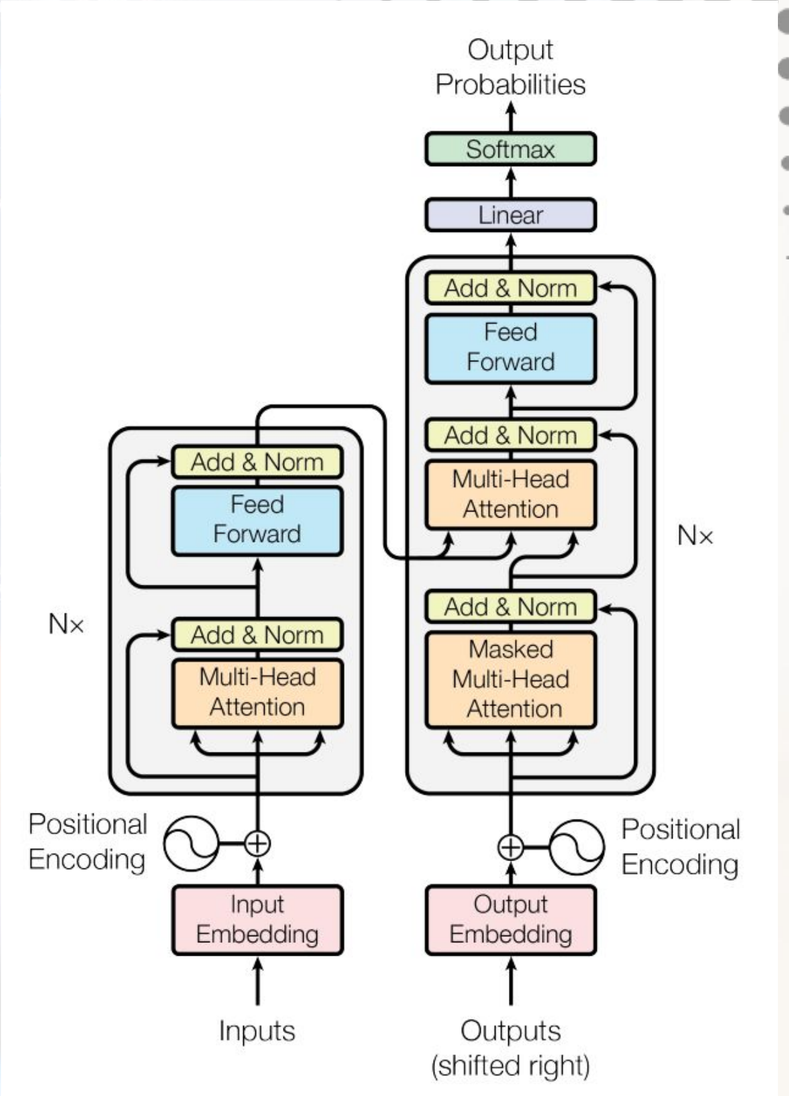


LET’S RETURN...

Still haven’t covered:

- **I/O Embeddings**
- **Positional Encoding**
- Output (Linear --> Softmax)

These are (mostly) more classic ideas, but we will come back...



KEY VOCABULARY

Tokenization

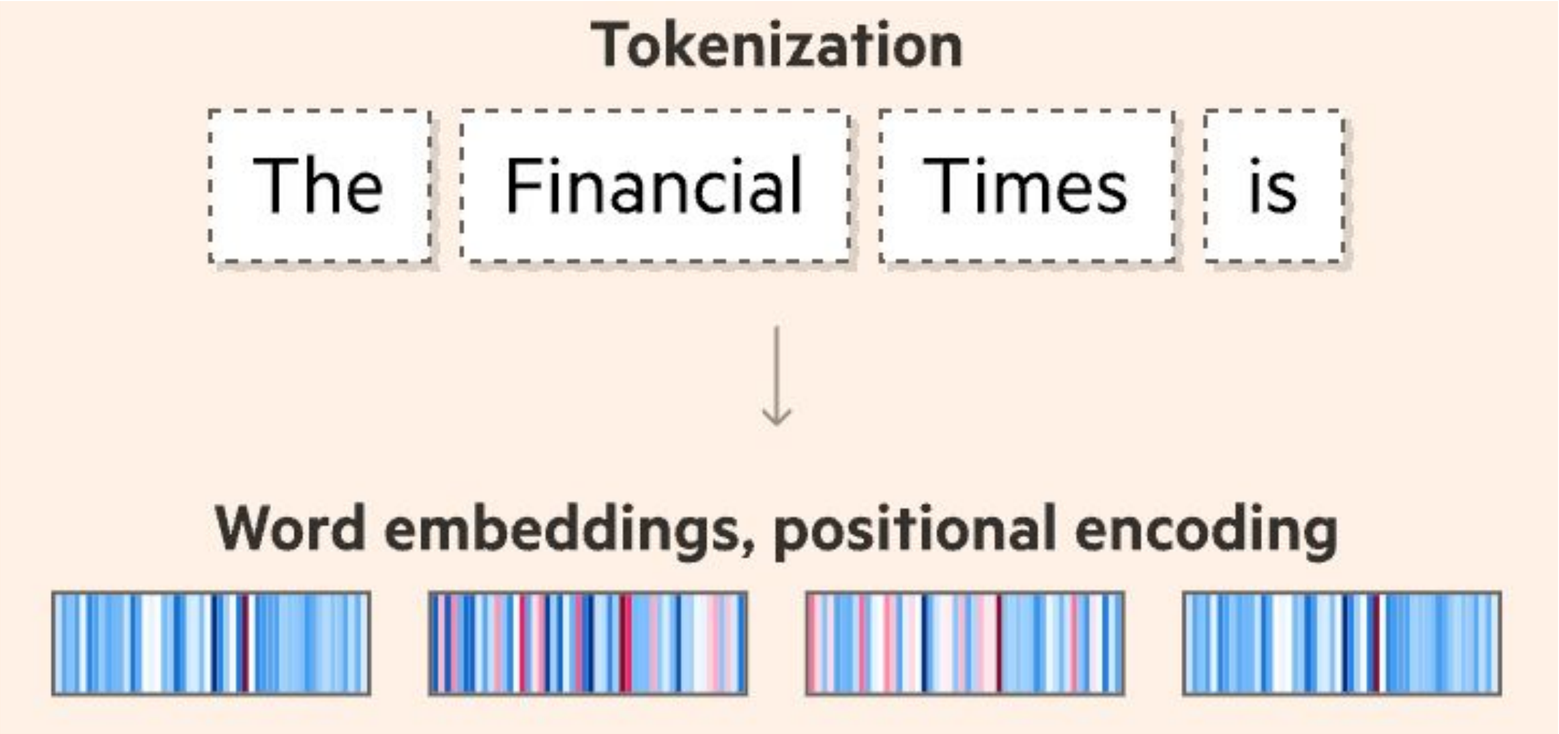
- Chop words

Embeddings

- Words --> numbers

(Positional) Encoding

- Words within sentence



KEY VOCABULARY

Tokenization

- Chop words

Embeddings

- Words --> numbers

(Positional) Encoding

- Words within sentence



TOKENIZATION

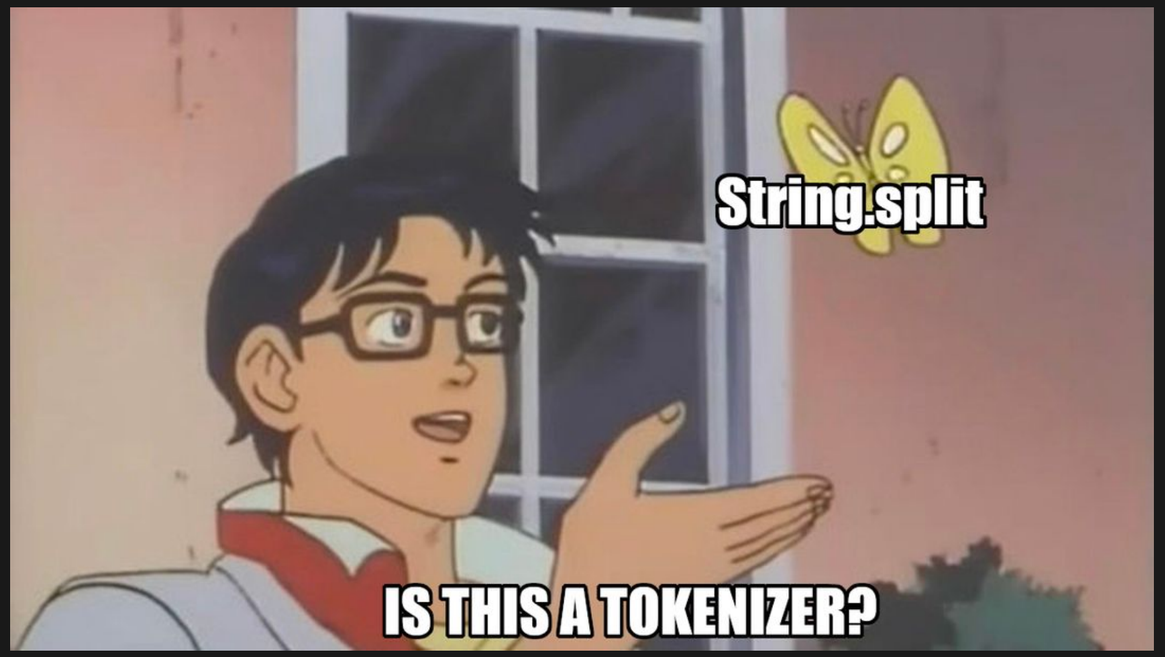
- **Chopping text into pieces (tokens)**
- Words, usually subwords

For *O'Neill*, which of the following is the desired tokenization?

neill		
oneill		
o'neill		
o'	neill	
o	neill	?

And for *aren't*, is it:

aren't		
arent		
are	n't	
aren	t	?



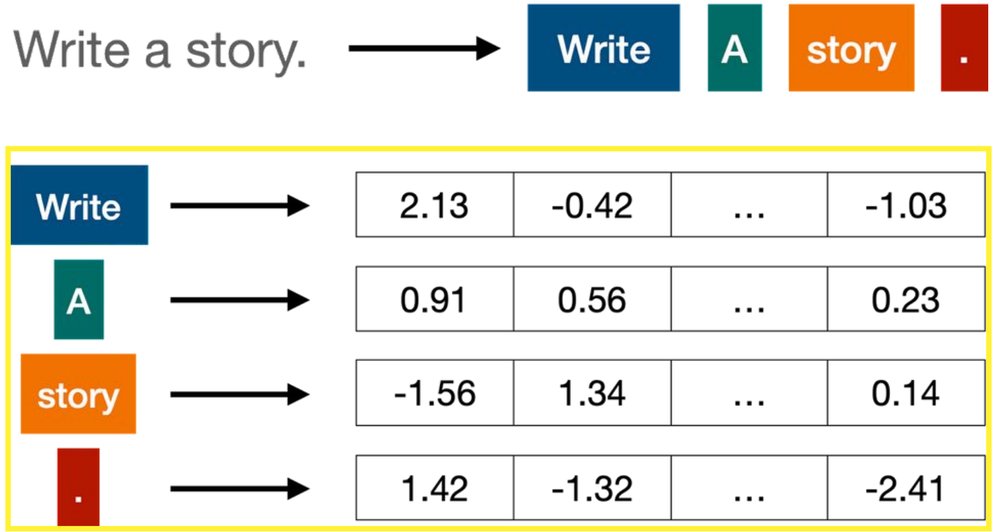
Input: Friends, Romans, Countrymen, lend me your ears;

Output:							
	Friends	Romans	Countrymen	lend	me	your	ears

KEY VOCABULARY

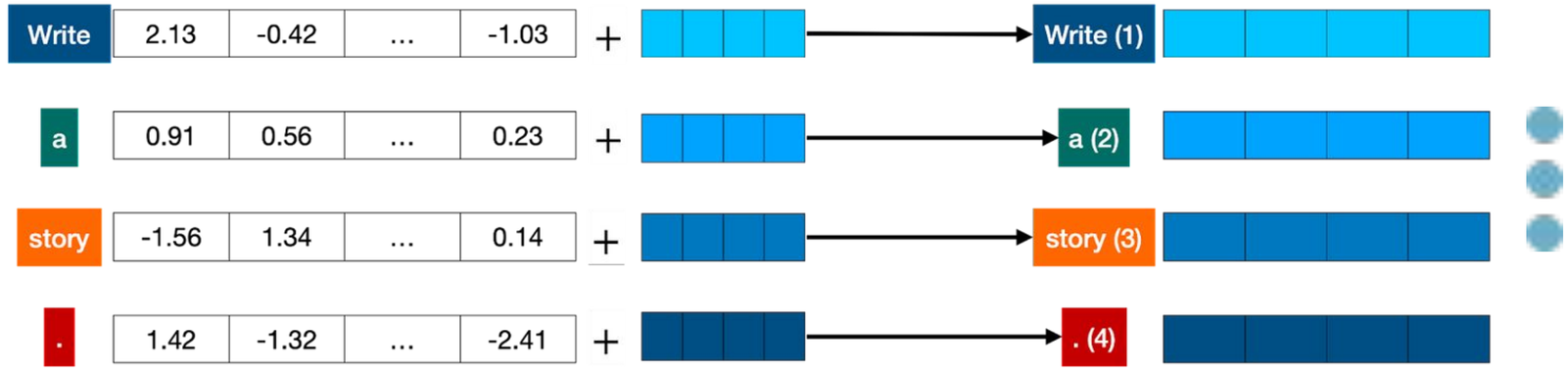
Tokenization

- Chop words



Embeddings

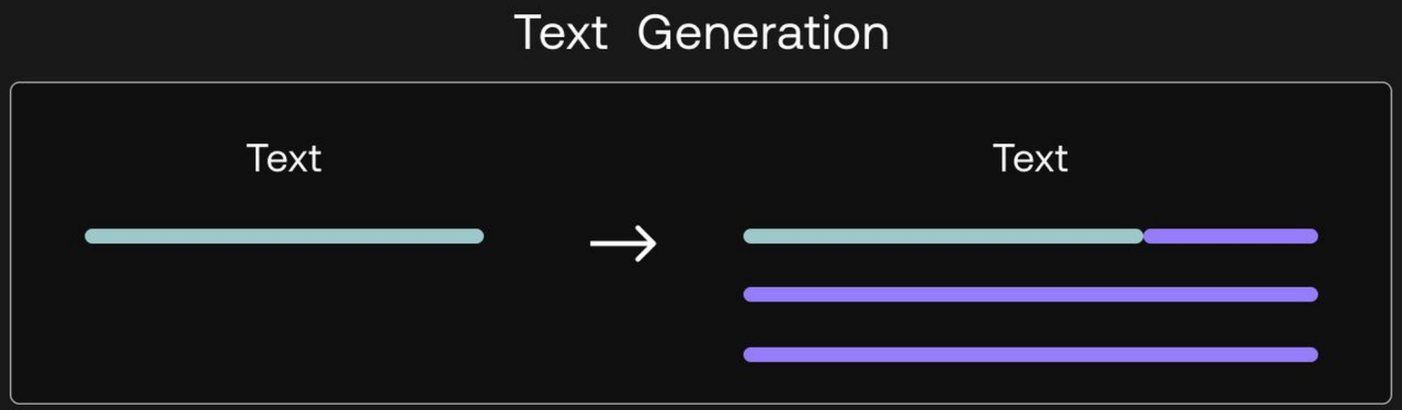
- Words --> numbers



(Positional) Encoding

- Words within sentence

EMBEDDINGS



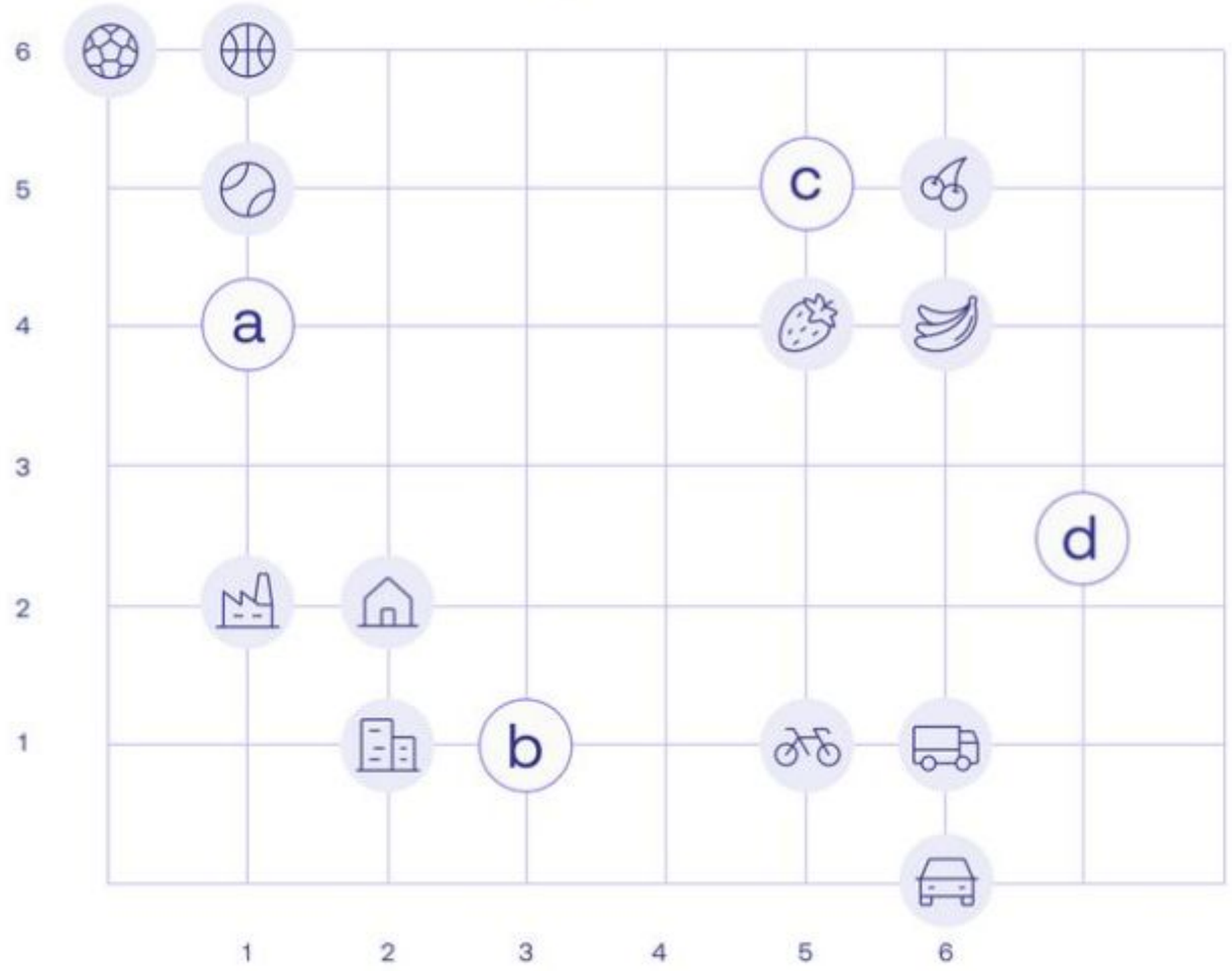


https://txt.cohere.com/text-embeddings/

QUIZ!

Embeddings Quiz 1:

Where would you put the word “apple”?

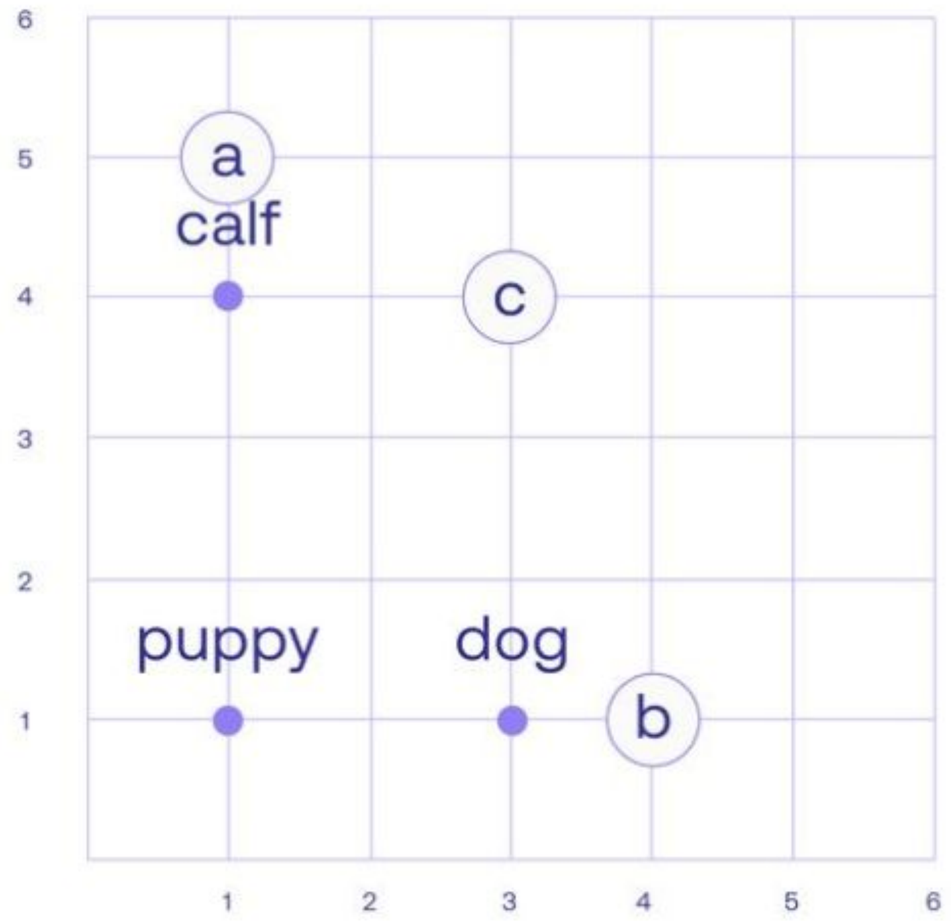


https://txt.cohere.com/sentence-word-embeddings/

QUIZ!

Embeddings Quiz 2:

Where would you put the word “cow”?



https://txt.cohere.com/sentence-word-embeddings/

Word embeddings

Many more columns

Word	Numbers	
Apple	5	5
Soccer	0	6
House	2	2
Car	6	0

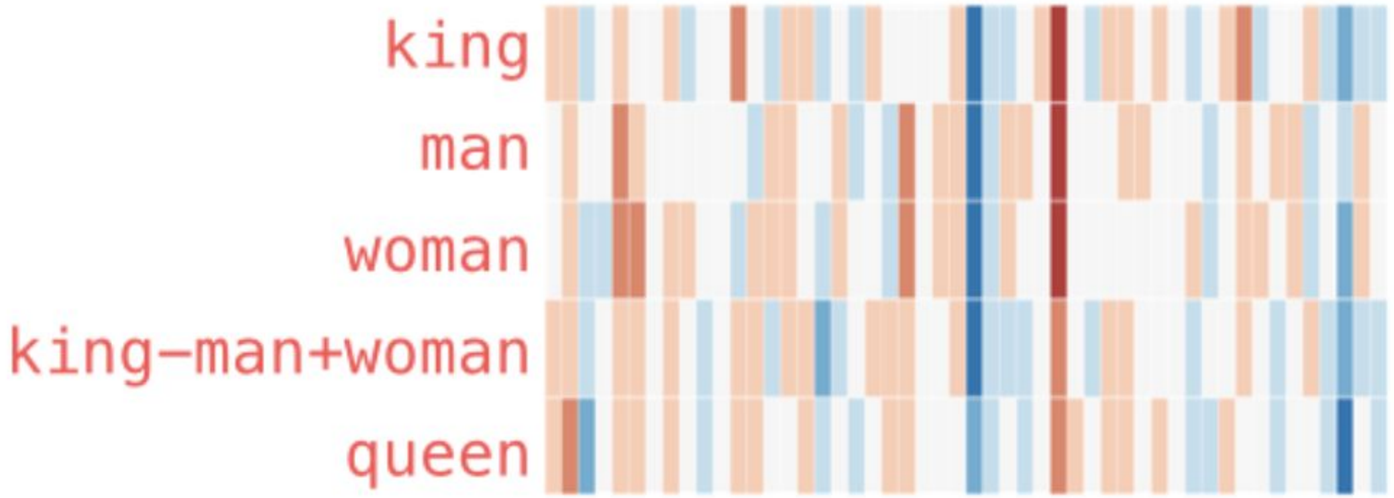
Word	Numbers			
A	-0.82	-0.32	...	-0.23
Aardvark	0.419	1.28	...	-0.06
...			...	
Zygote	-0.74	-1.02	...	1.35

4096

https://txt.cohere.com/sentence-word-embeddings/

WORD2VEC

king - man + woman ~= queen



KEY VOCABULARY

Tokenization

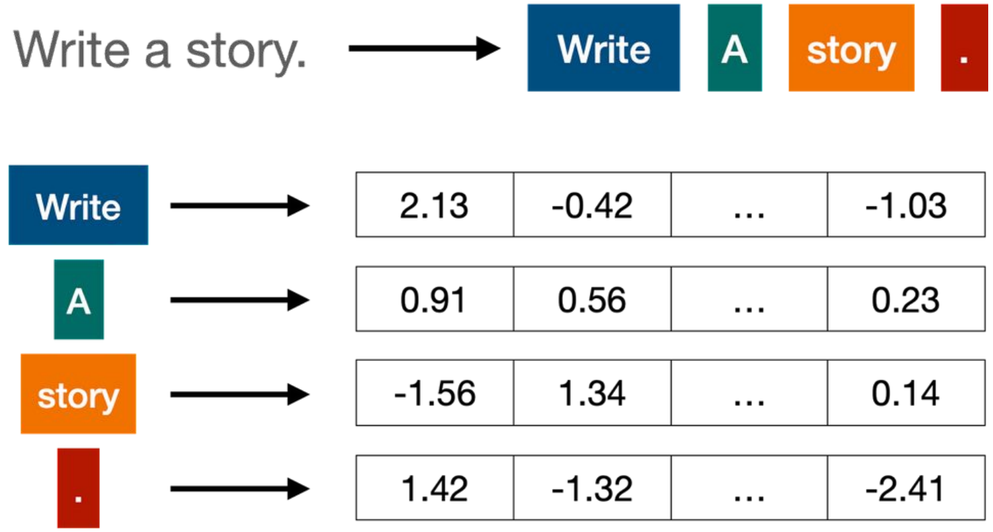
- Chop words

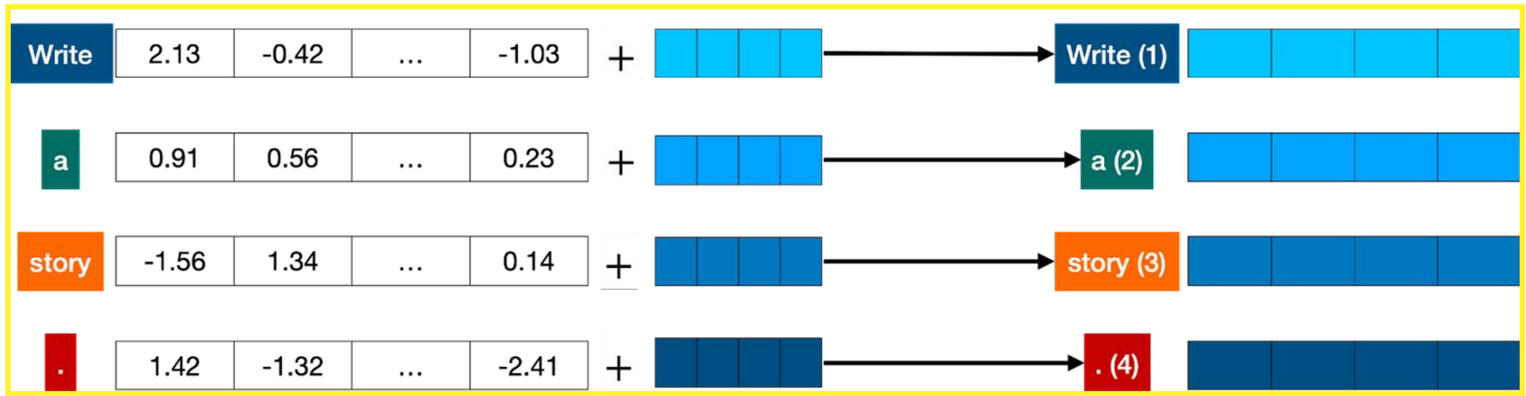
Embeddings

- Words --> numbers

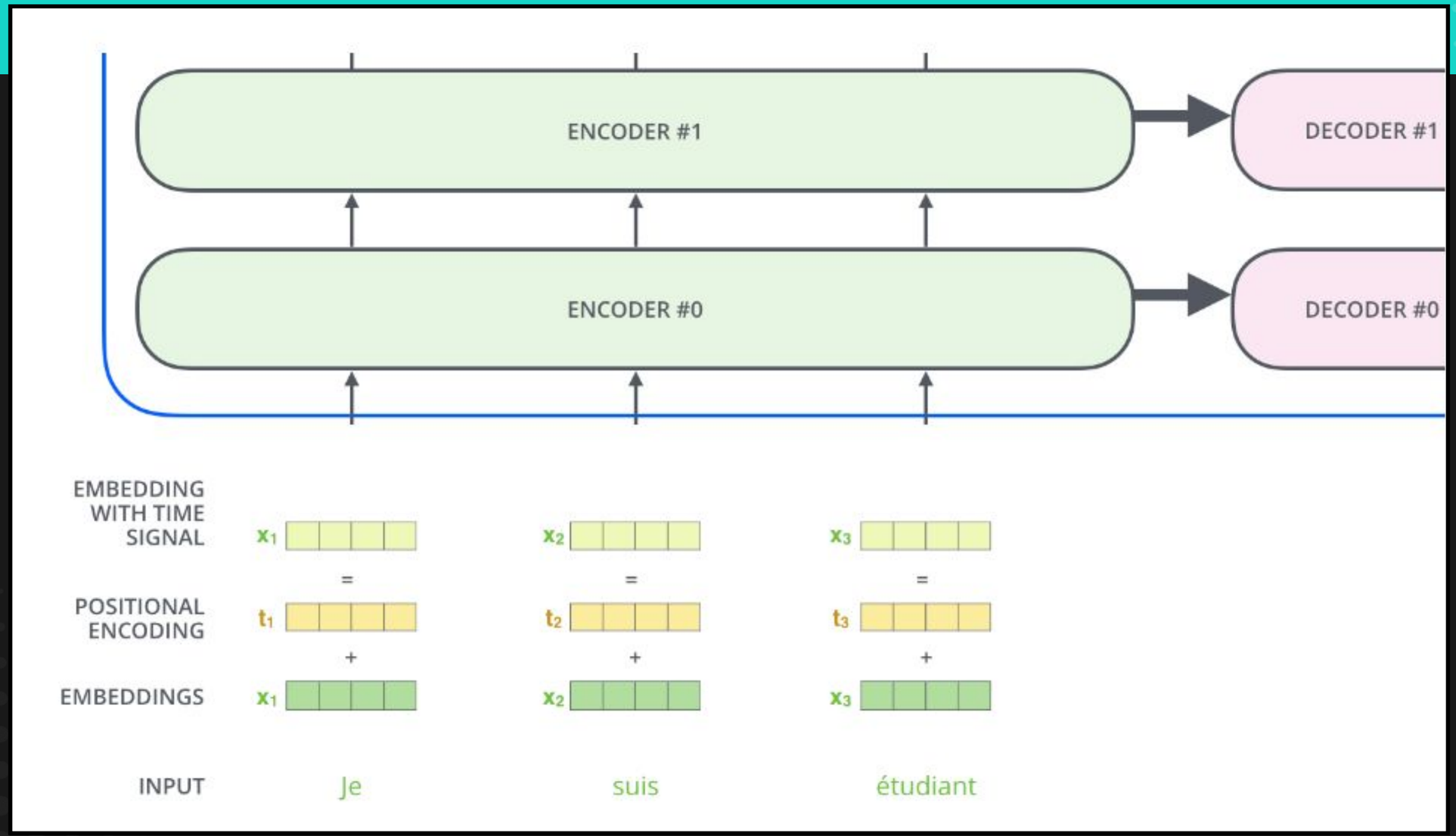
(Positional) Encoding

- Words within sentence



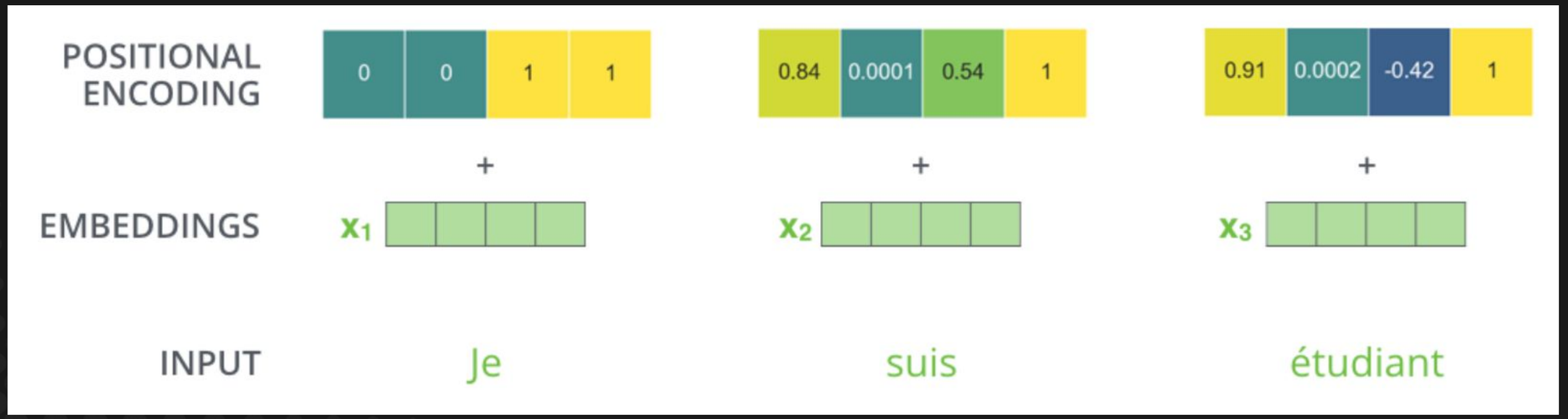


(POSITIONAL) ENCODING



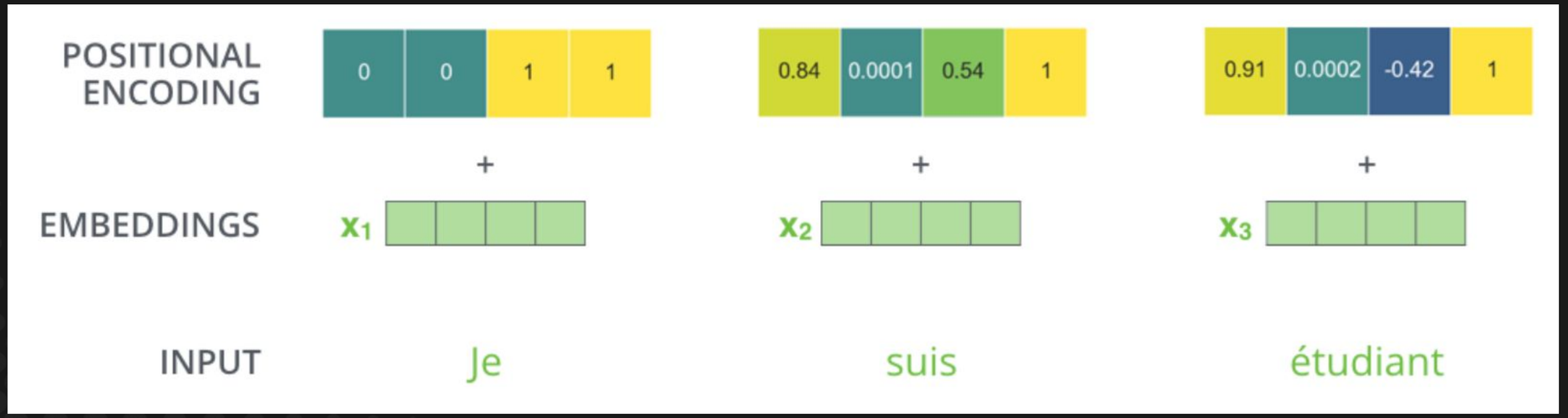
(POSITIONAL) ENCODING

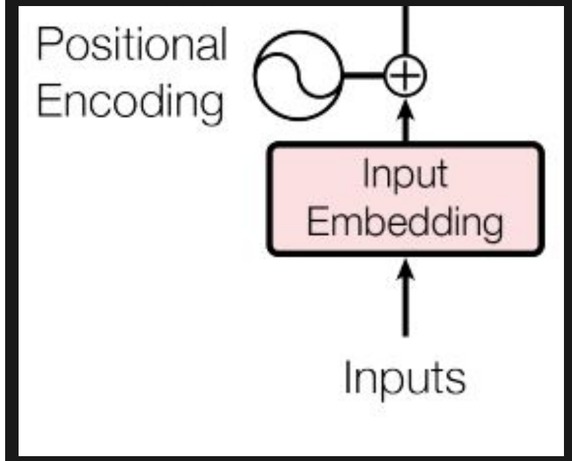
- **Order of the words (tokens)**
- Add meaningful vector to each token embedding



(POSITIONAL) ENCODING

- **Order of the words (tokens)**
- Add meaningful vector to each token embedding





KEY VOCABULARY

Tokenization

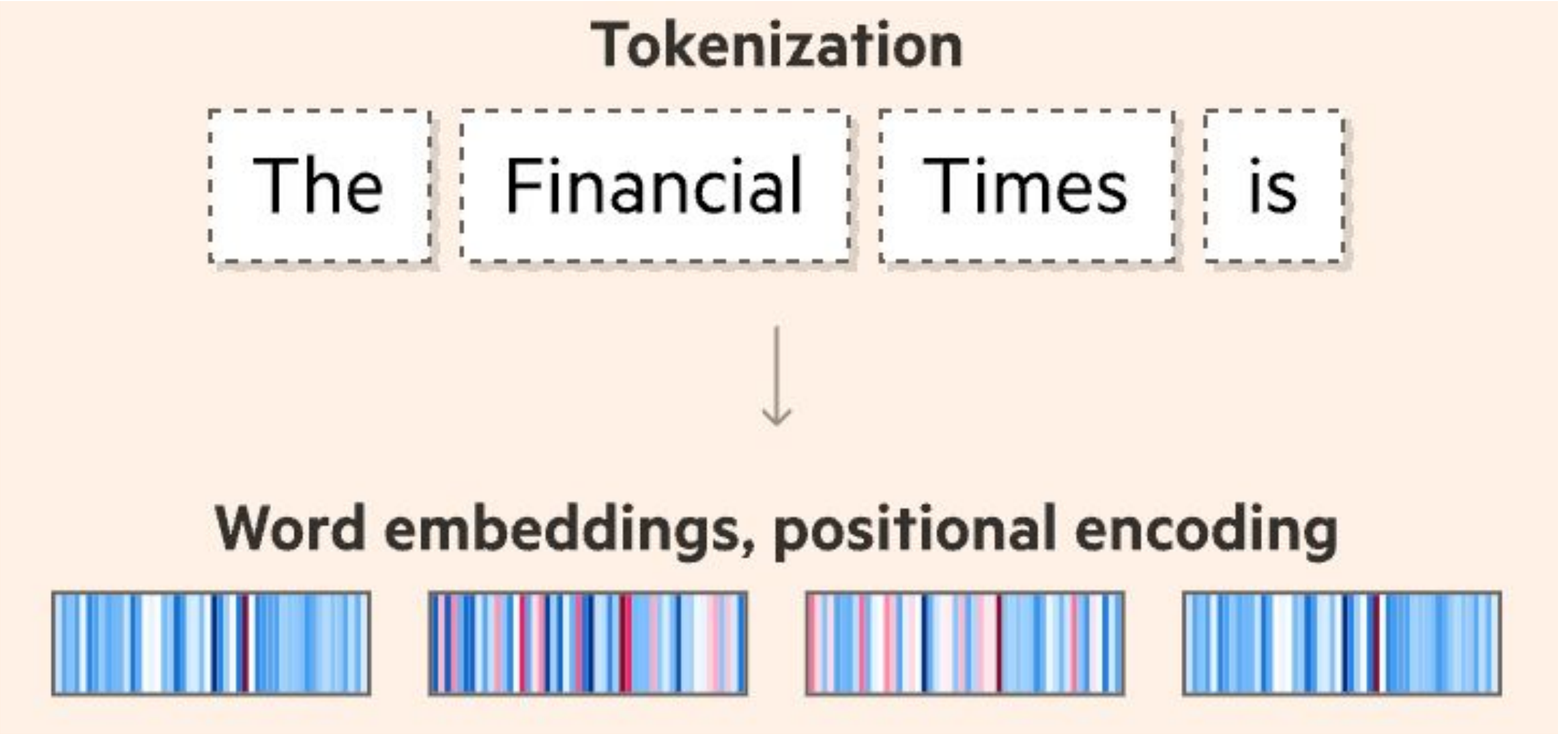
- Chop words

Embeddings

- Words --> numbers

(Positional) Encoding

- Words within sentence

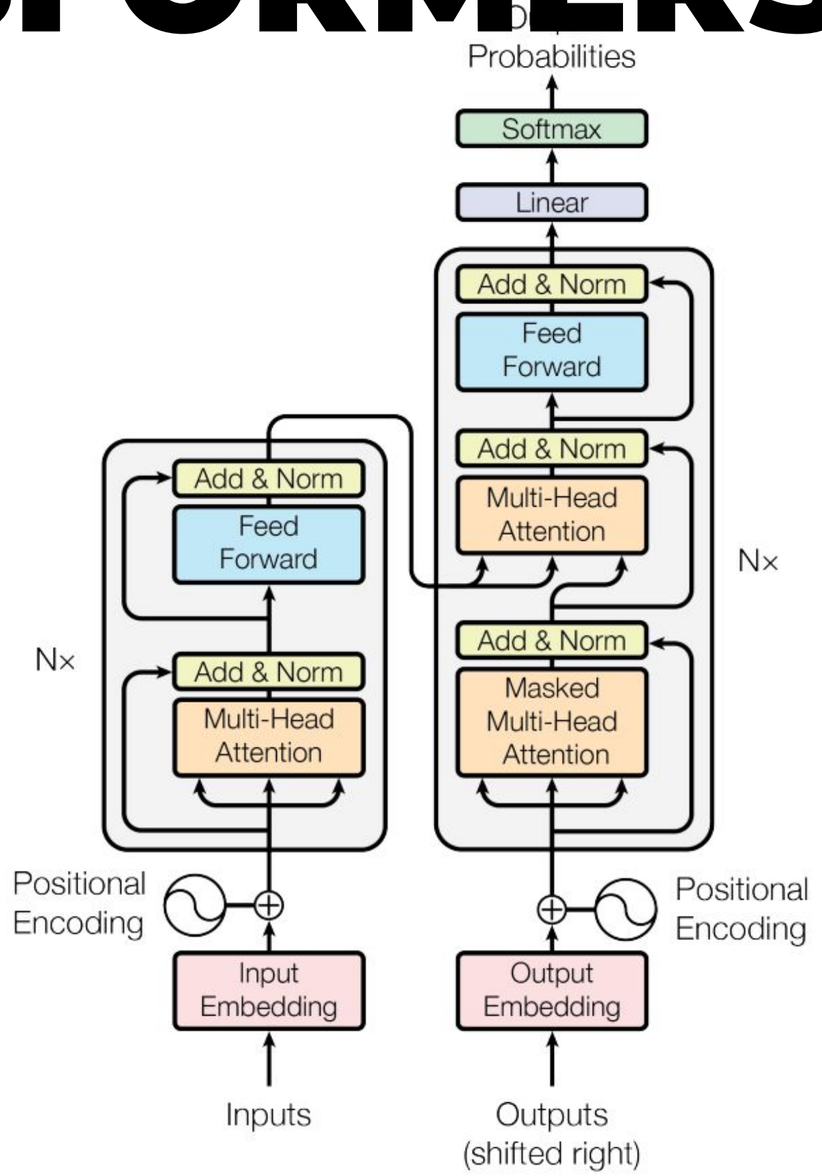


THE TRANSFORMER

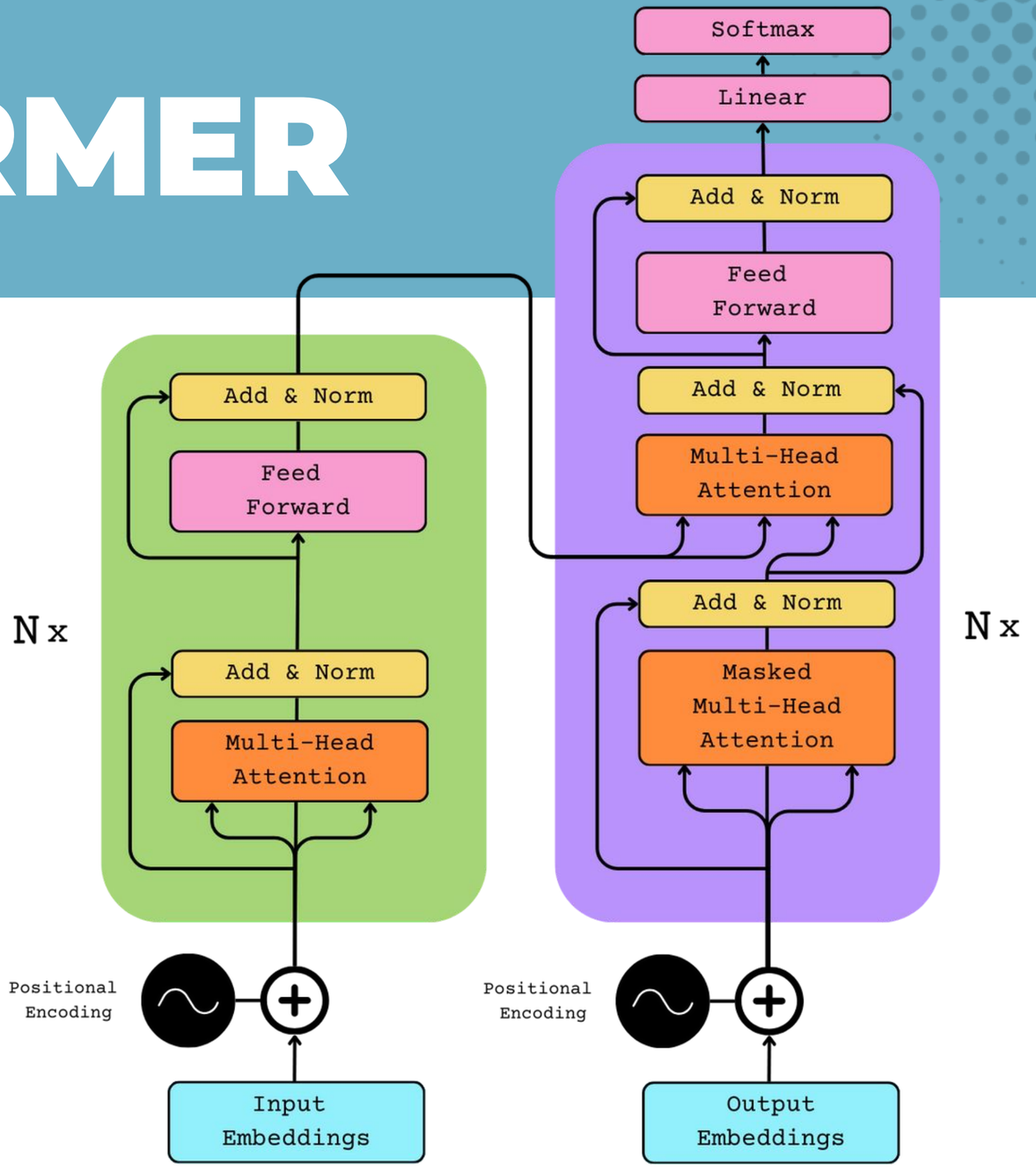
A BRIEF INTRODUCTION

TRANSFORMERS

- ~~This is ChatGPT~~



TRANSFORMER

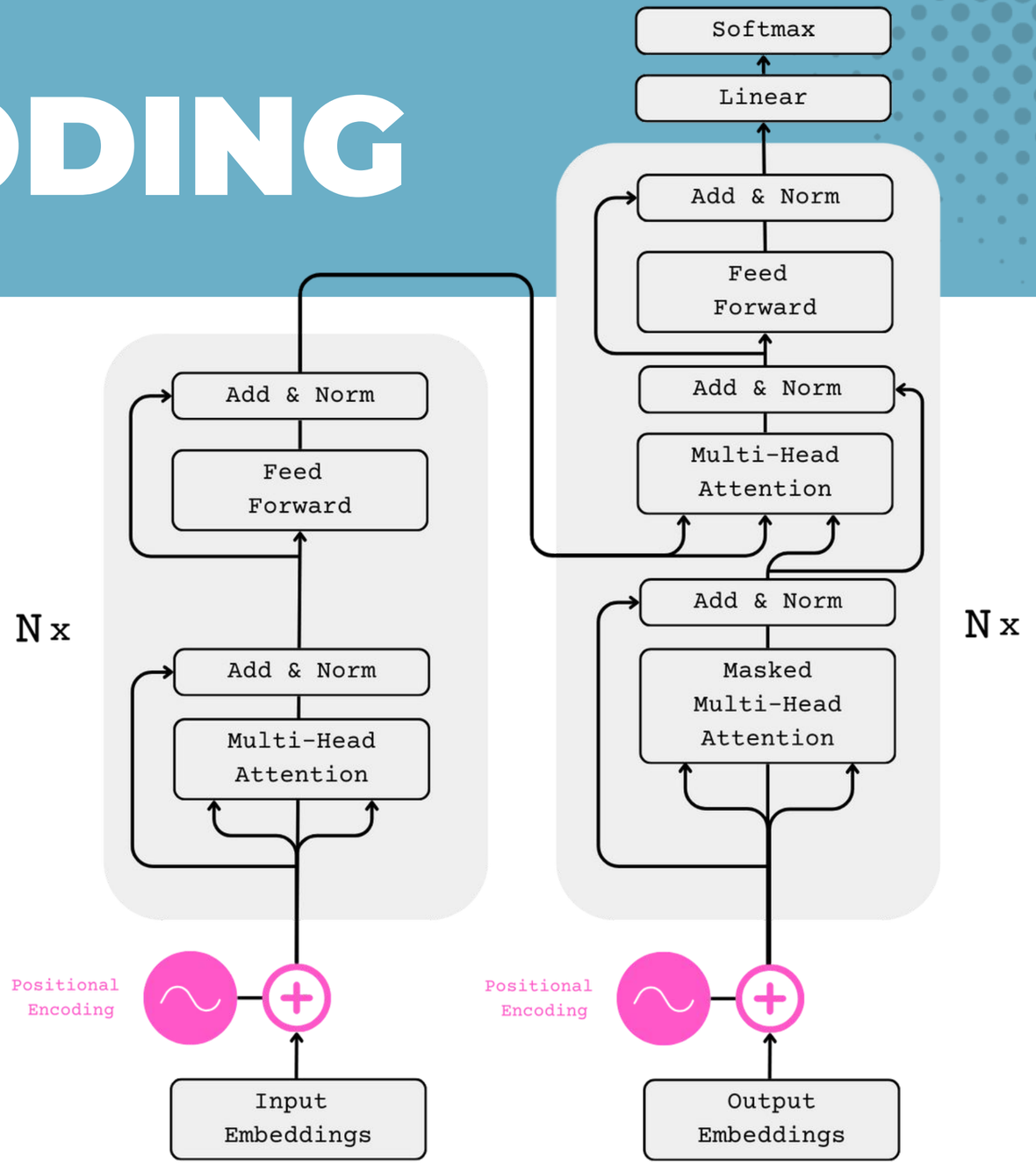


POS. ENCODING

Impart information about

where each token is **in the sequence**

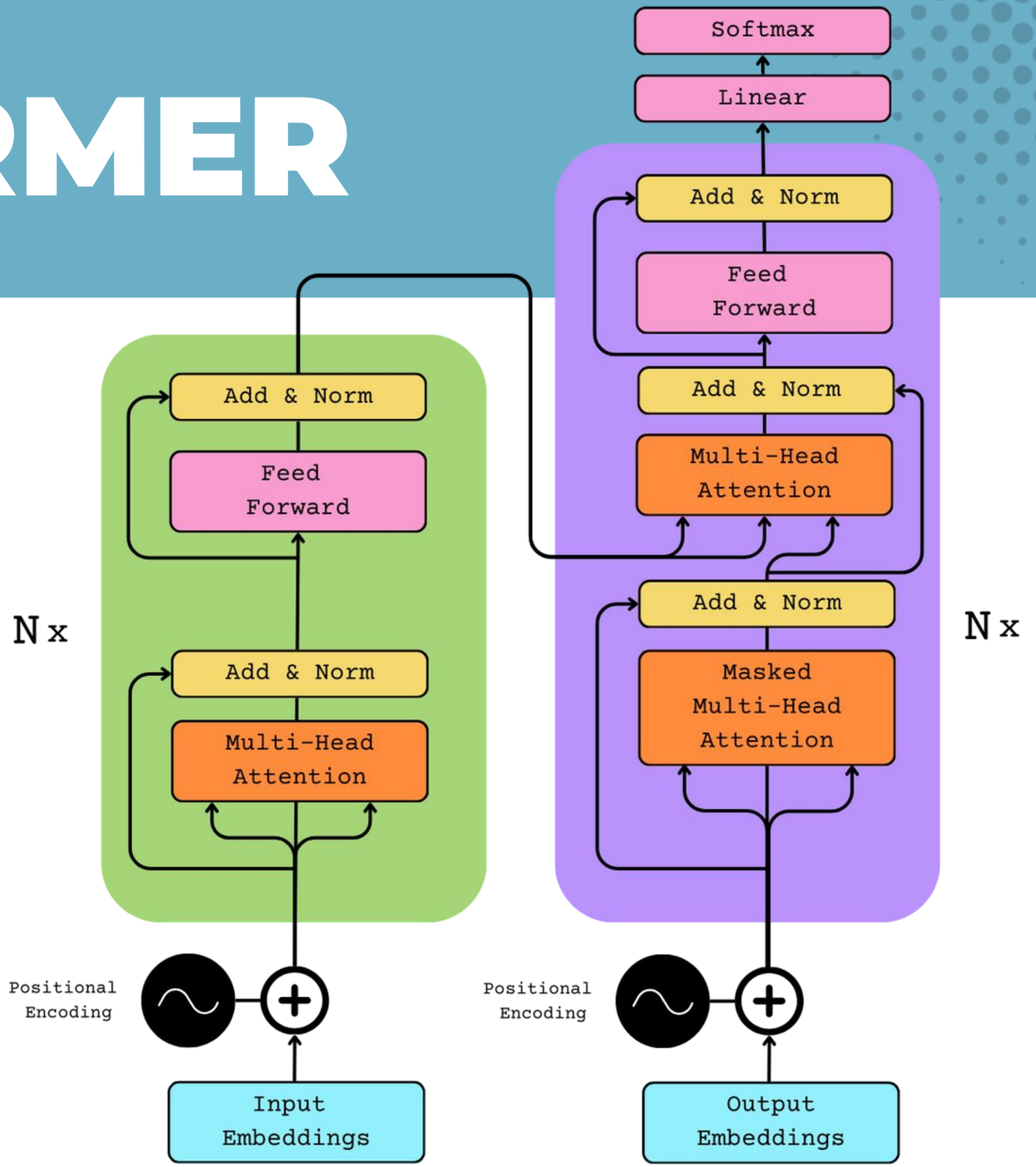
WITHOUT **recurrence** or **convolutions**



SETTING UP EMBEDDINGS, POSITIONAL ENCODING

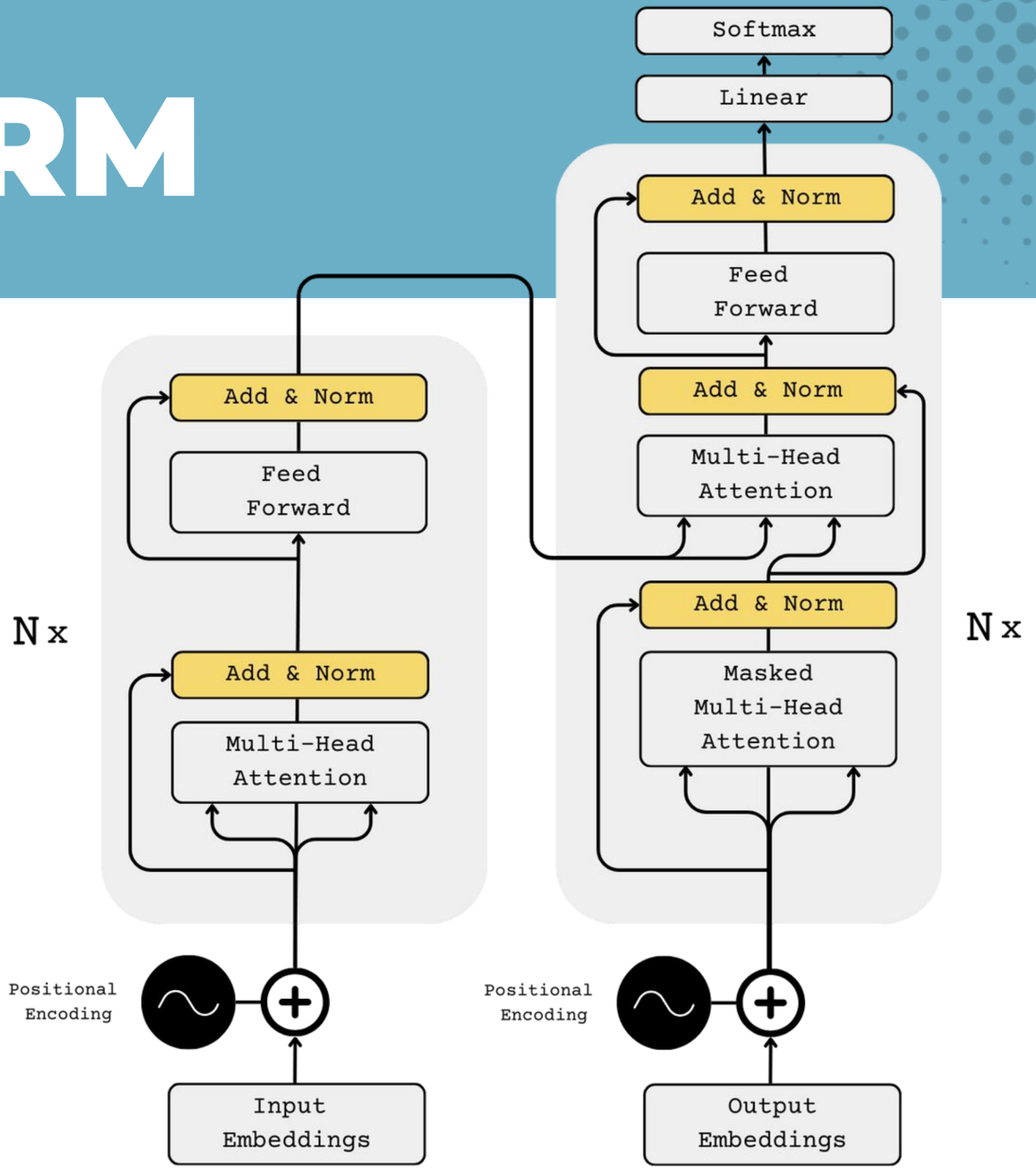
Presented by
Chris Alexiuk, LLM Wizard 🪄

TRANSFORMER



ADD & NORM

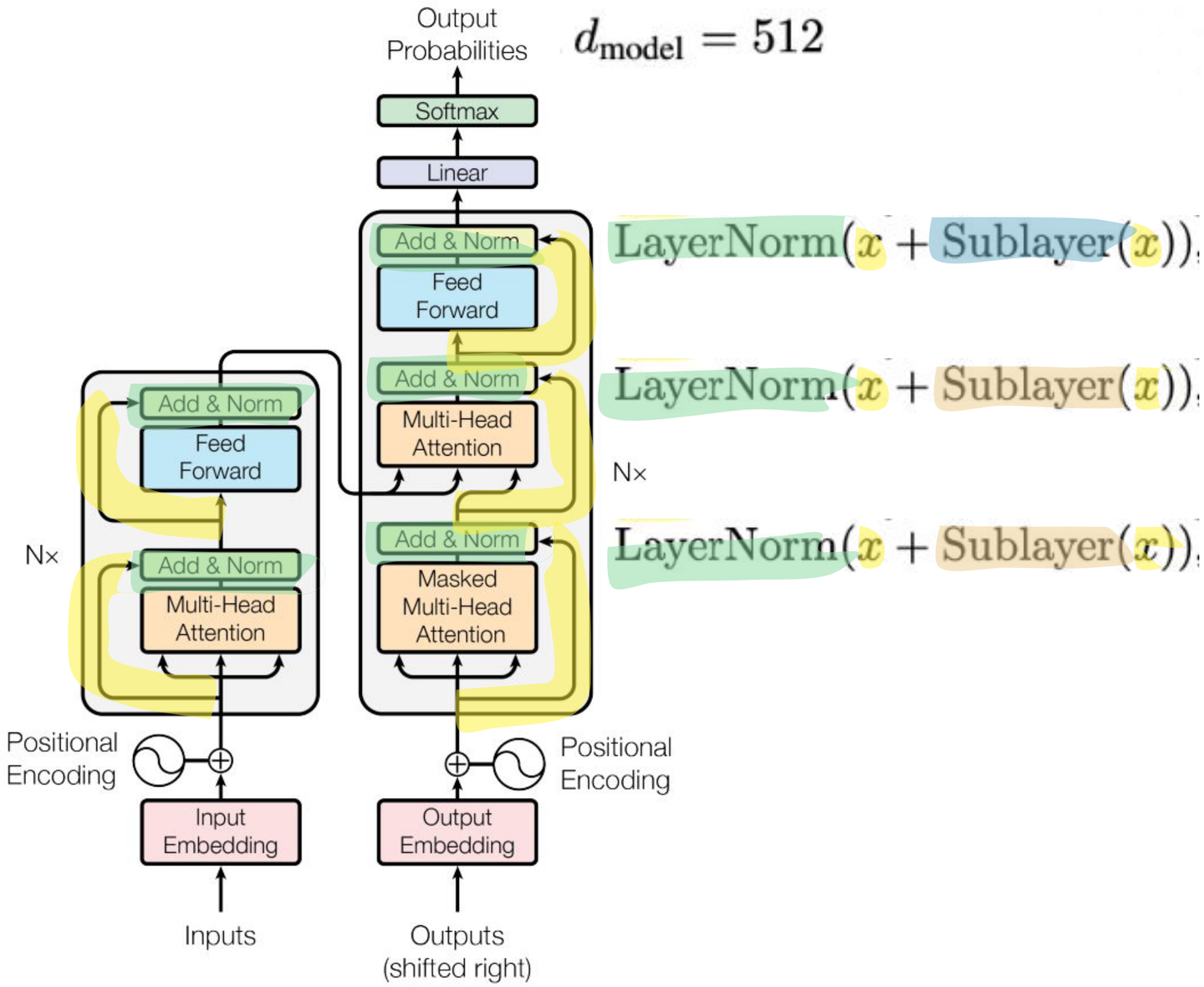
The basic idea is that it **makes training the model a bit easier**, and allows the model to **generalize** a bit better.



TRANSFORMERS

Layer Normalization

Residual Connection



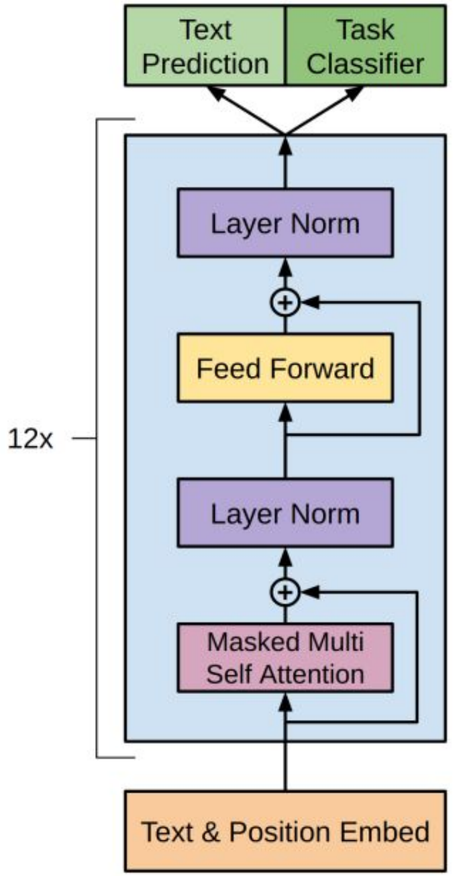
$d_{\text{model}} = 512$

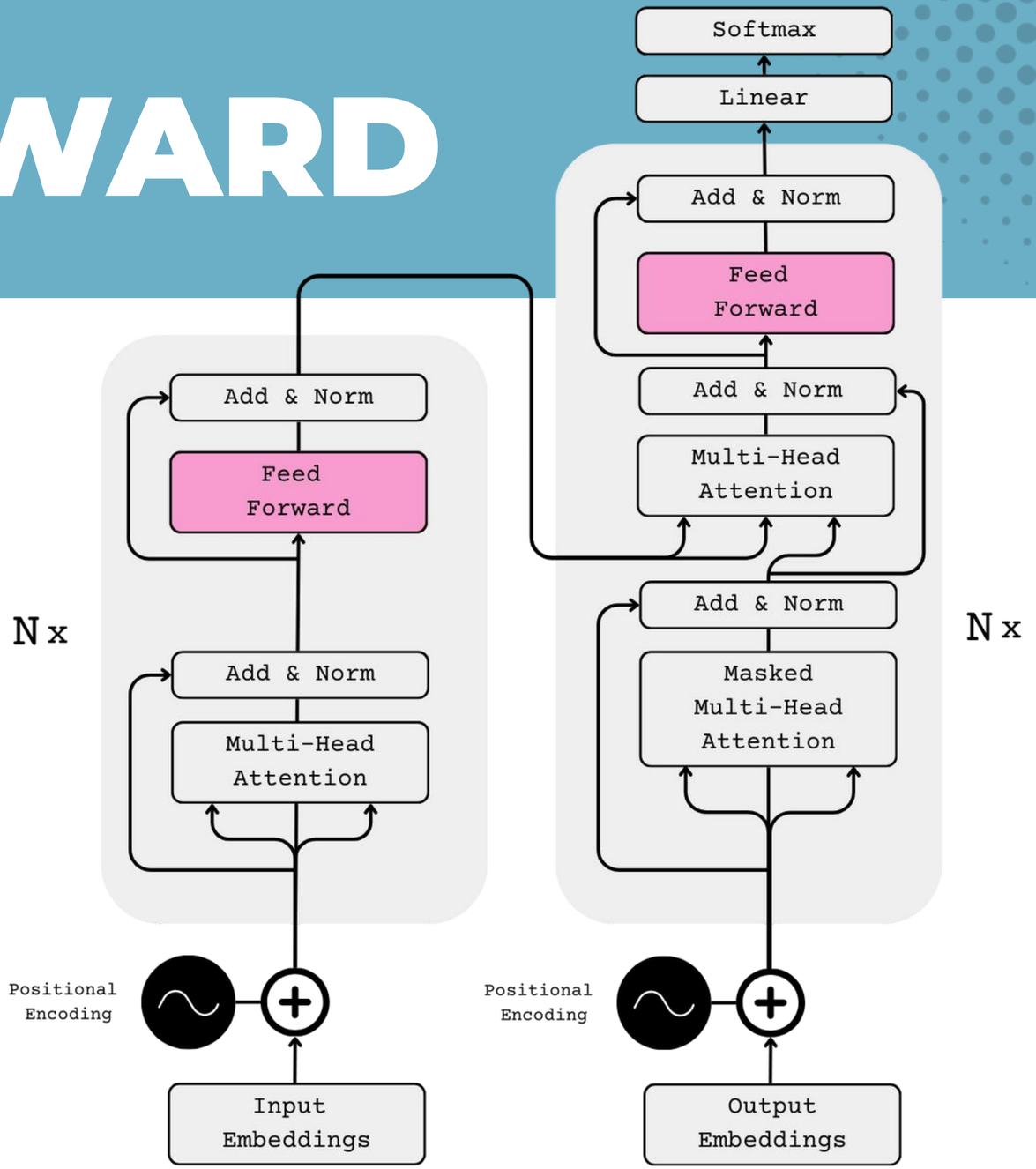
$\text{LayerNorm}(x + \text{Sublayer}(x))$

$\text{LayerNorm}(x + \text{Sublayer}(x))$

$\text{LayerNorm}(x + \text{Sublayer}(x))$

FEED FORWARD





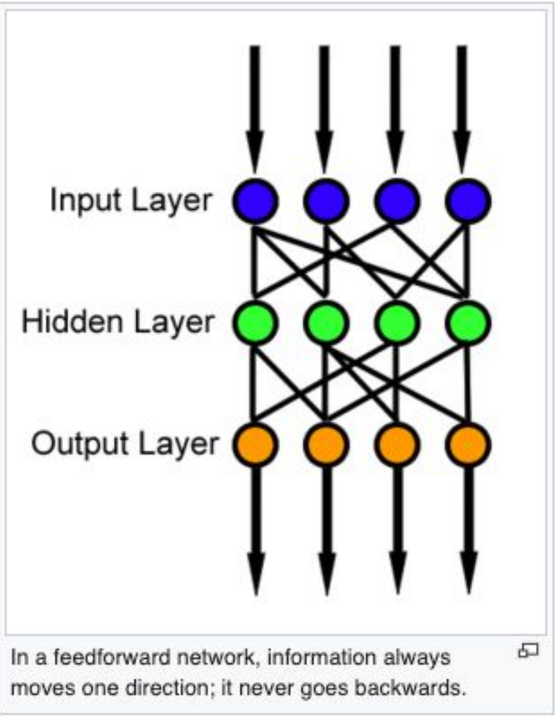
FEED FORWARD

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

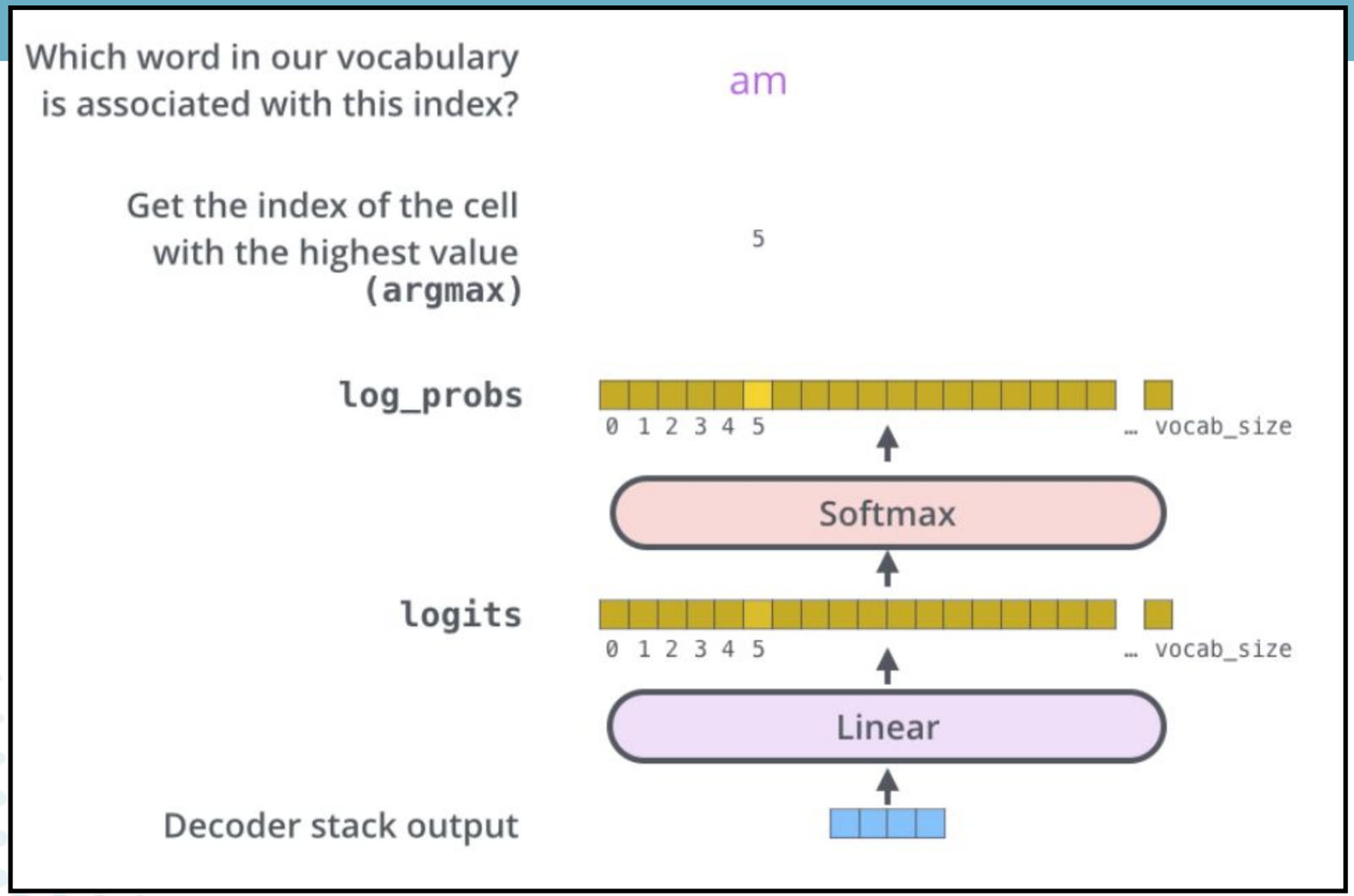


In a feedforward network, information always moves one direction; it never goes backwards.

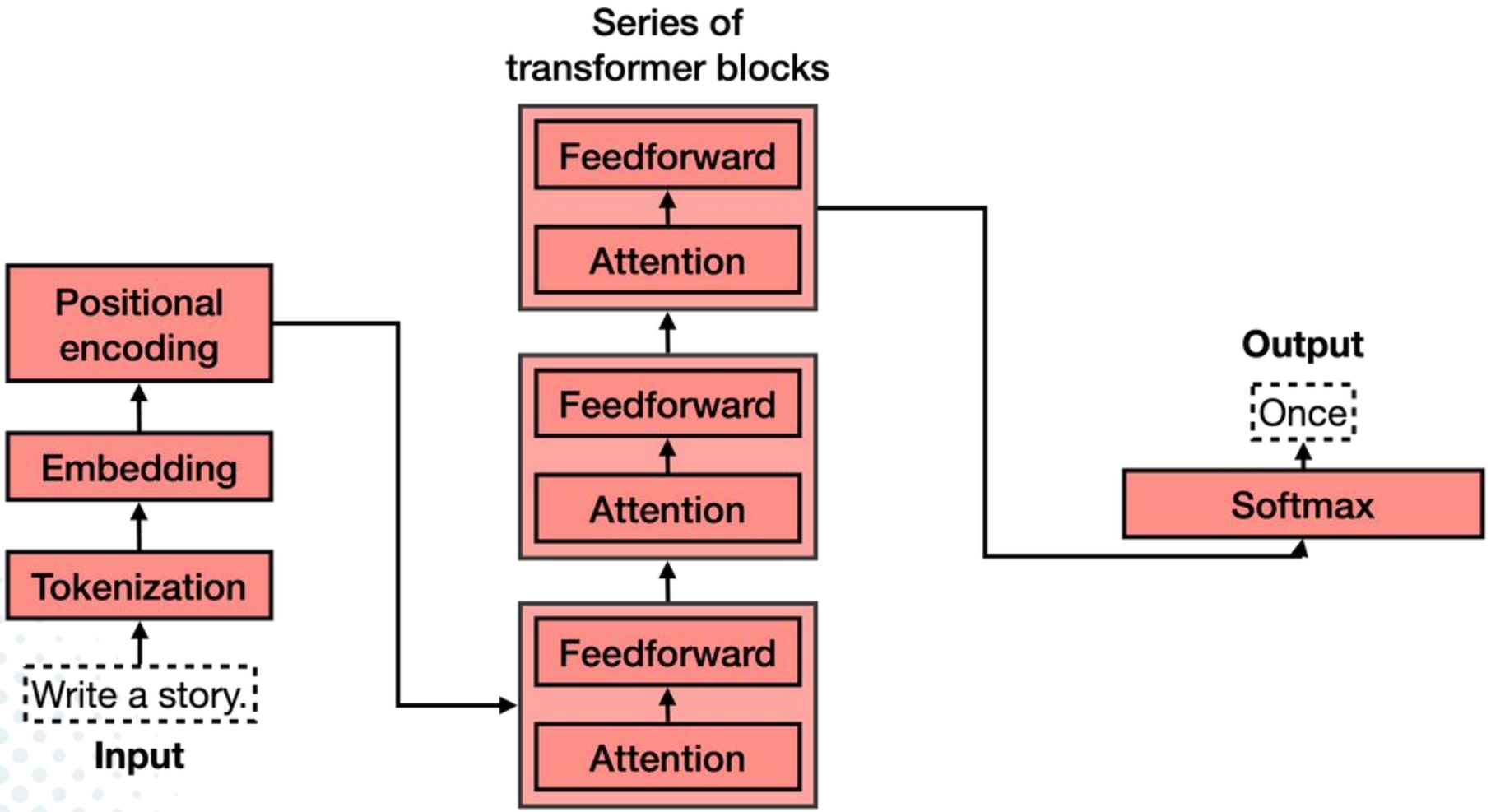
ADD & LAYER NORM, RESIDUAL CONNECTION, AND FEED FORWARD NN

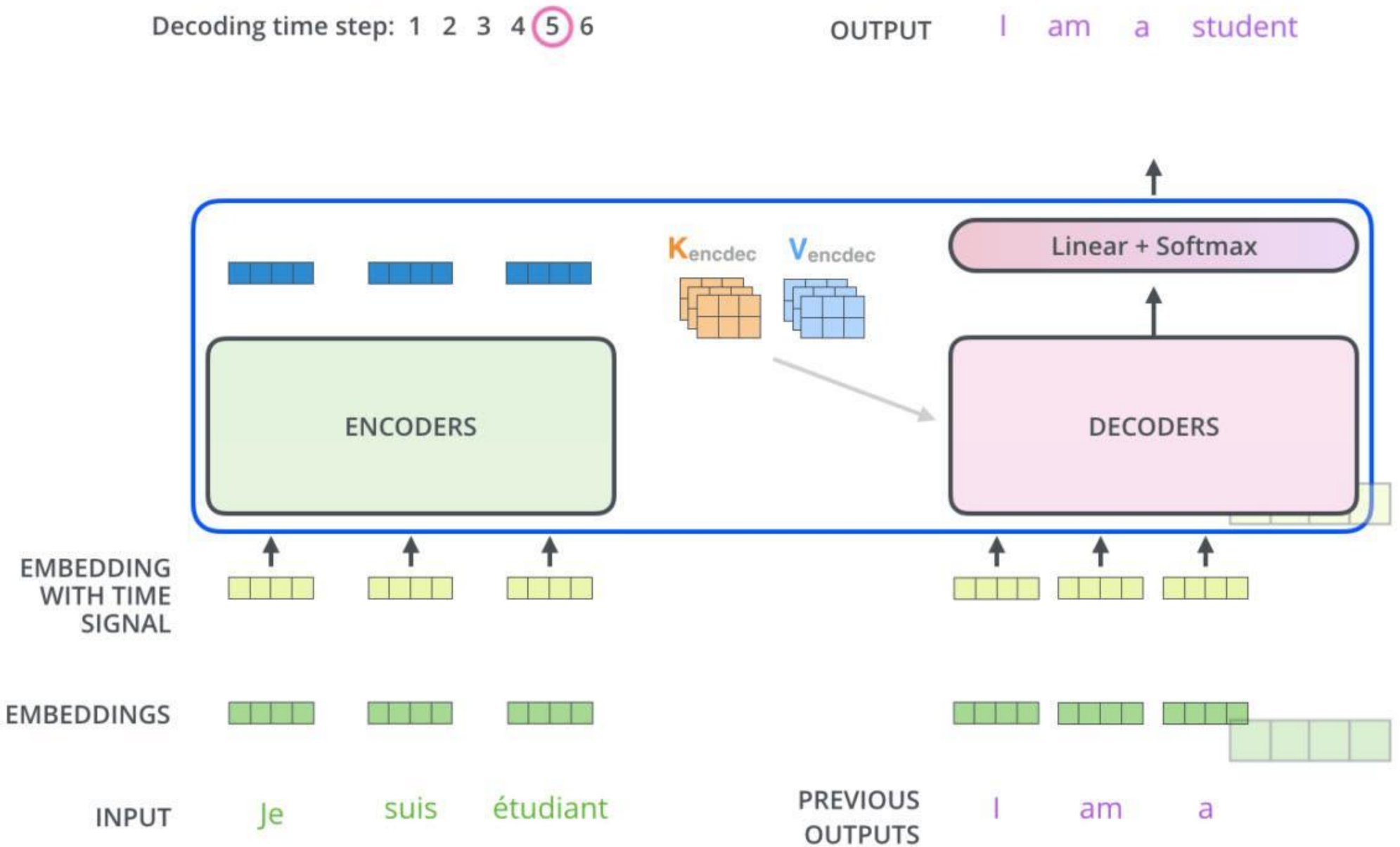
Presented by
Chris Alexiuk, LLM Wizard 🪄

LINEAR PROJECTION



GPT-STYLE EXAMPLE




Decoding time step: 1 2 3 4 5 6
OUTPUT I am a student
Linear + Softmax
K encdec
V encdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT Je suis étudiant
PREVIOUS OUTPUTS I am a

COMBINING IT ALL TOGETHER!

Presented by
Chris Alexiuk, LLM Wizard 🪄



TRAINING

BART

- **Bidirectional AutoRegressive Transformer (BART)**

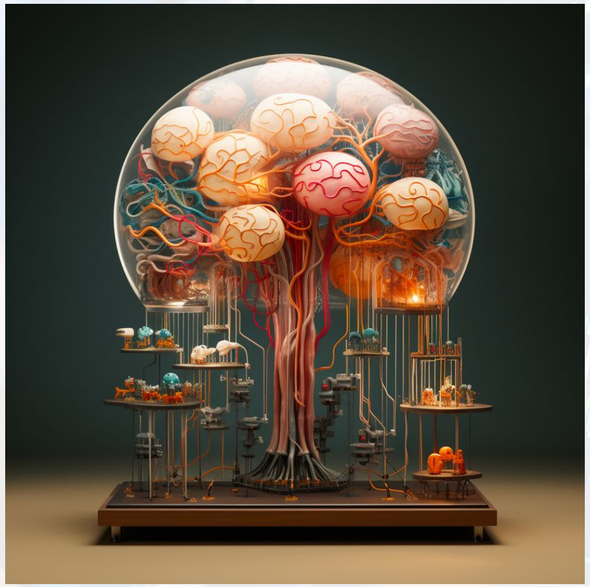
- Encoder-decoder (seq2seq) model
  - Bidirectional (BERT-like) encoder
  - Autoregressive (GPT-like) decoder

TRAINING (FROM THE PAPER!)

- 5.1 **Training Data** and Batching
- 5.2 **Hardware** and Schedule (NVIDIA P100)
- 5.3 **Optimizer** (Adam)
- 5.4 **Regularization**
  - Residual Dropout
  - Label Smoothing

TRAINING FOR TRANSLATION

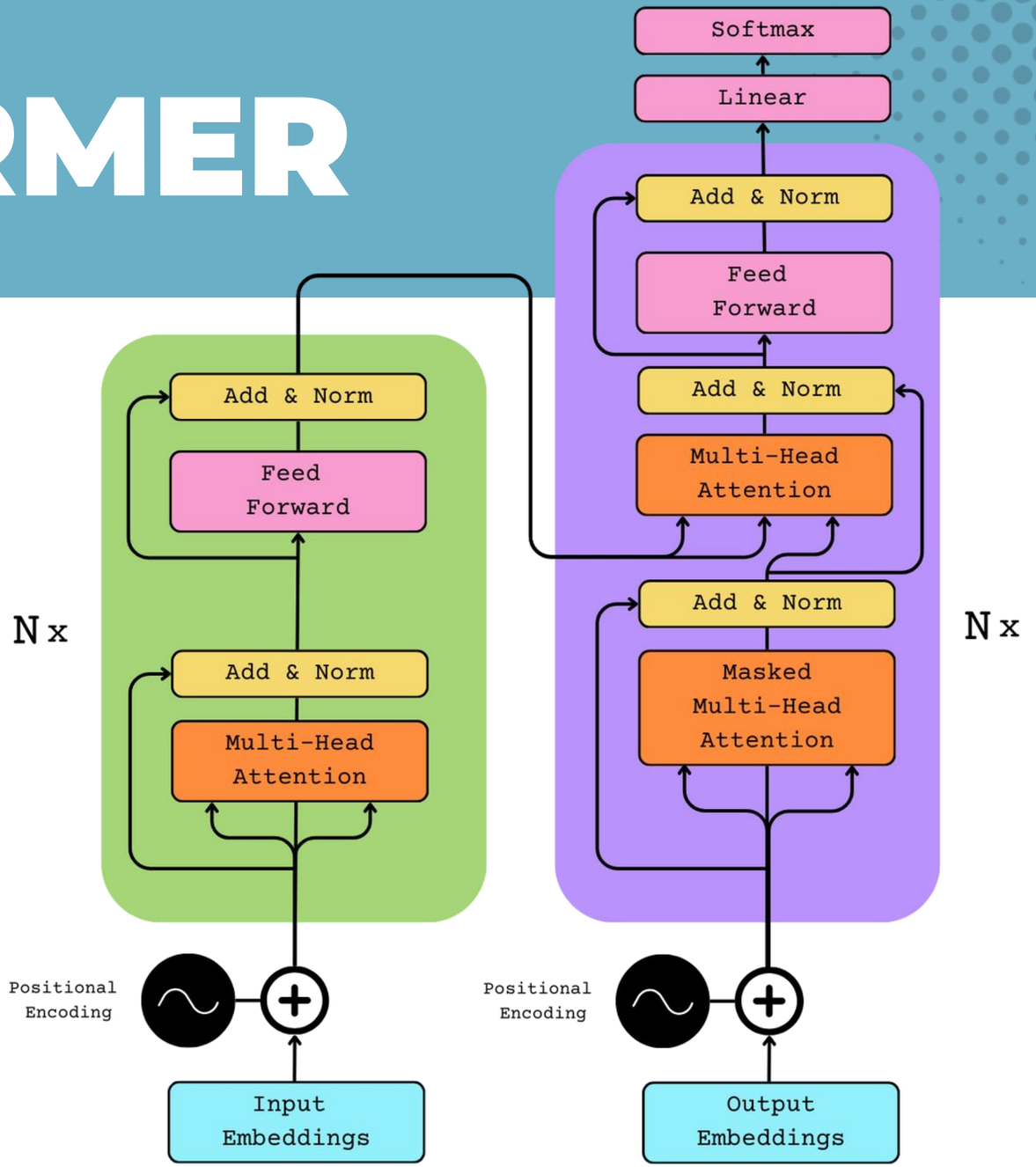
- Set up **BilingualDataset** from PyTorch
- Build **Tokenizer**
- Training Loop
  - Configure **hardware**
  - Create **directory** for saving model weights based on config.
  - Get data loaders, tokenizers, and model. **Move model** to device.
  - **Initialize** Adam **optimizer**
  - Set up **initial training parameters**



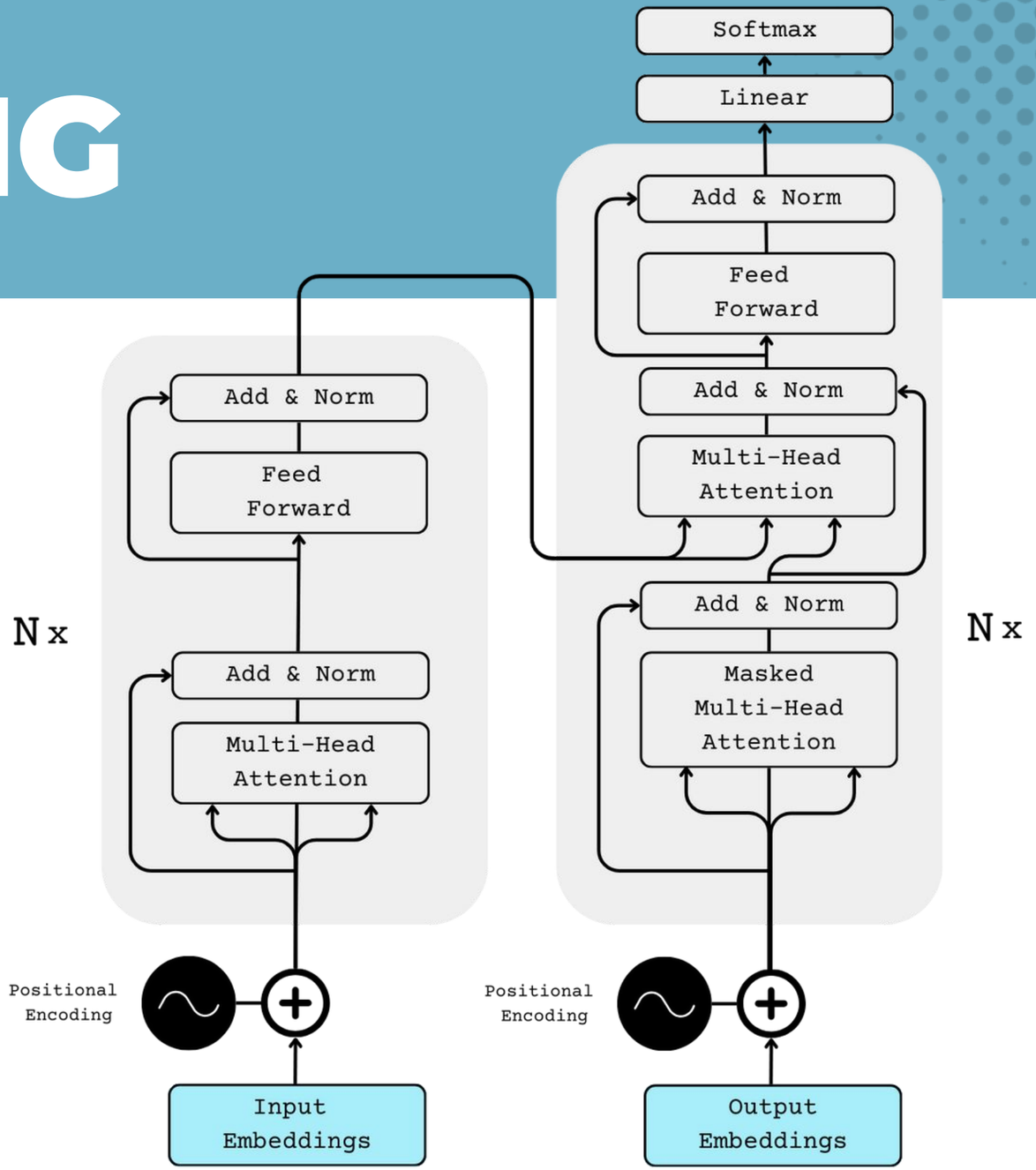
TRAINING THE BRAIN!

Presented by
Chris Alexiuk, LLM Wizard 🪄

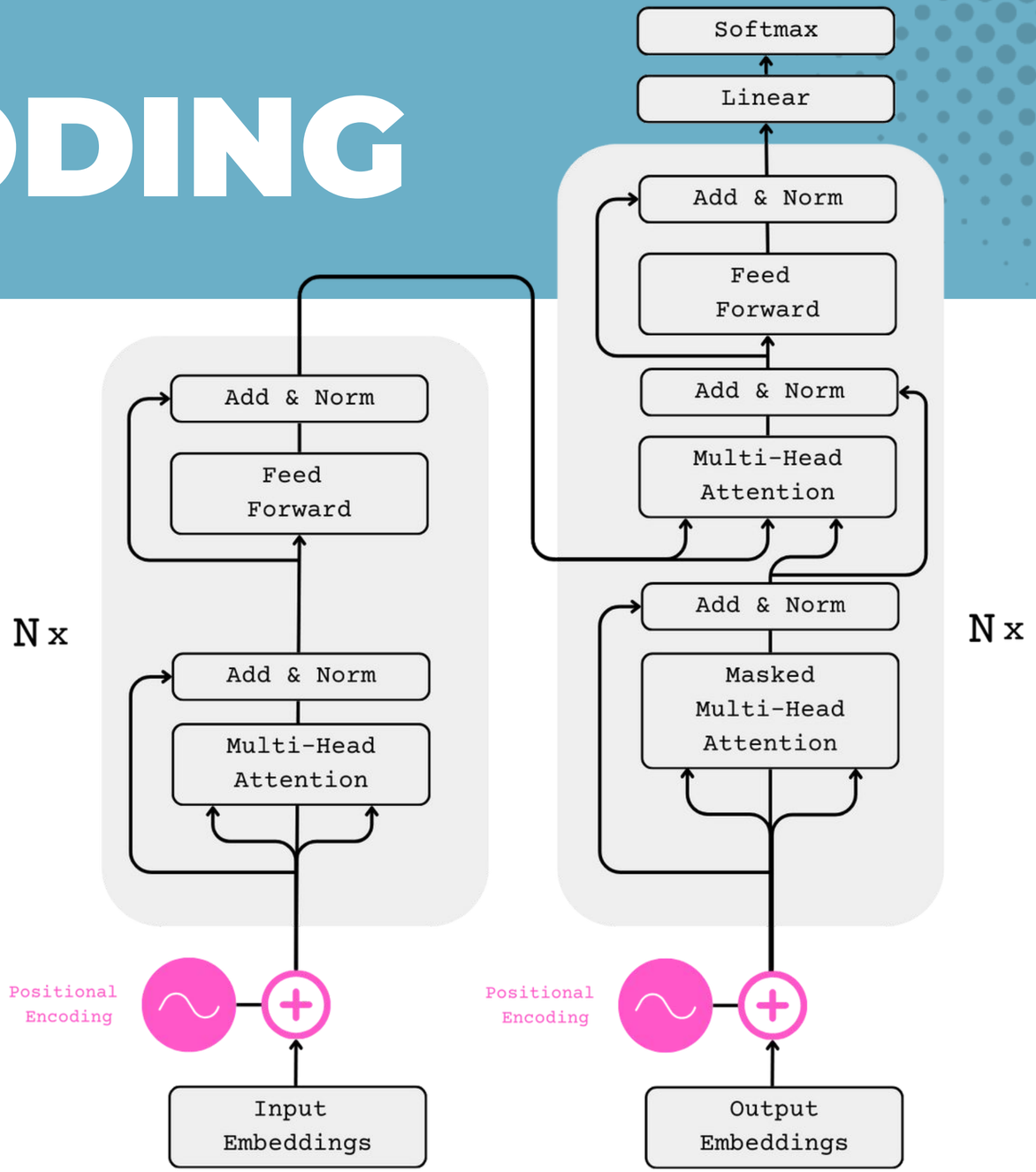
TRANSFORMER



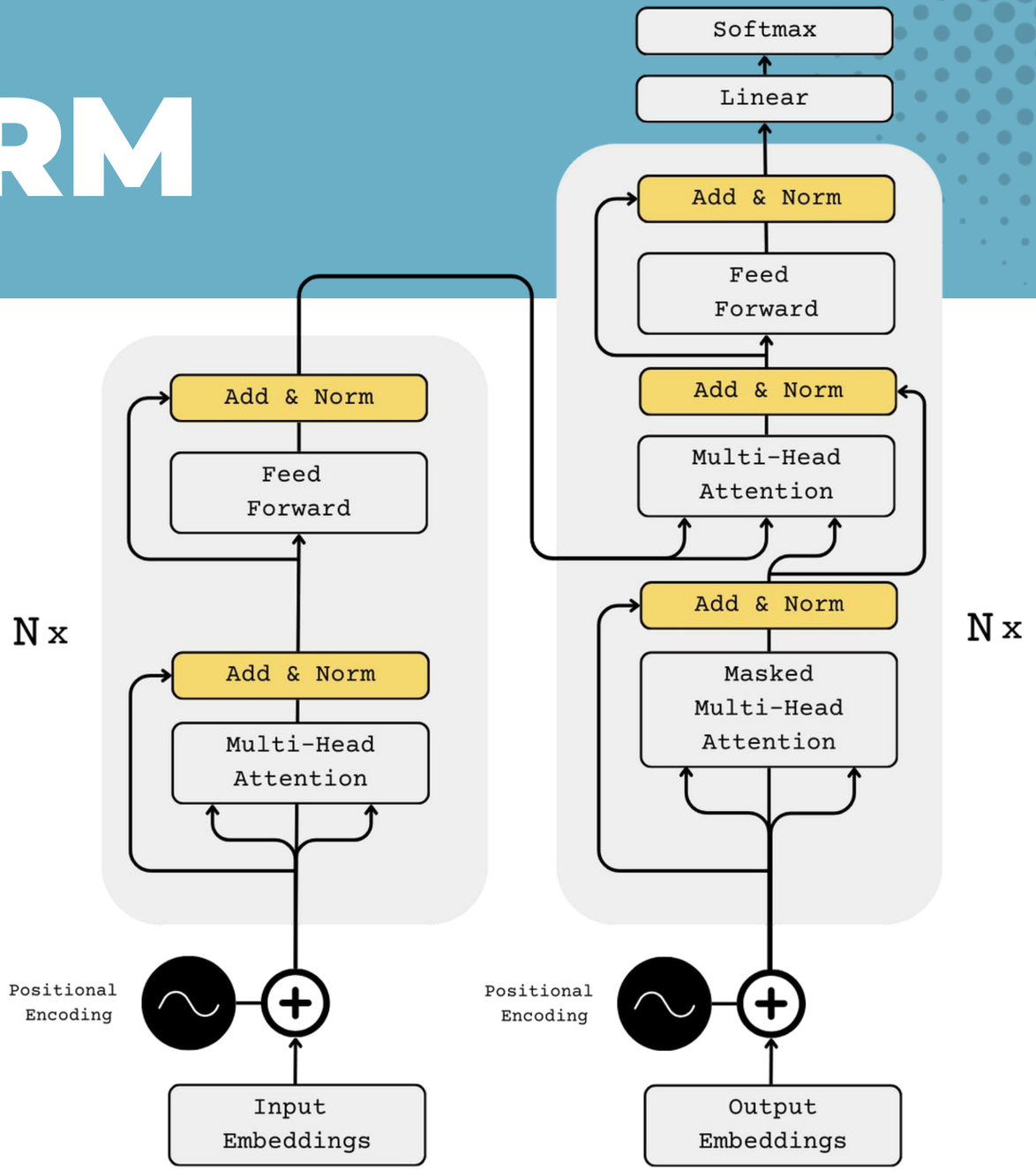
EMBEDDING


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N x
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N x
Positional Encoding
Positional Encoding
Input Embeddings
Output Embeddings

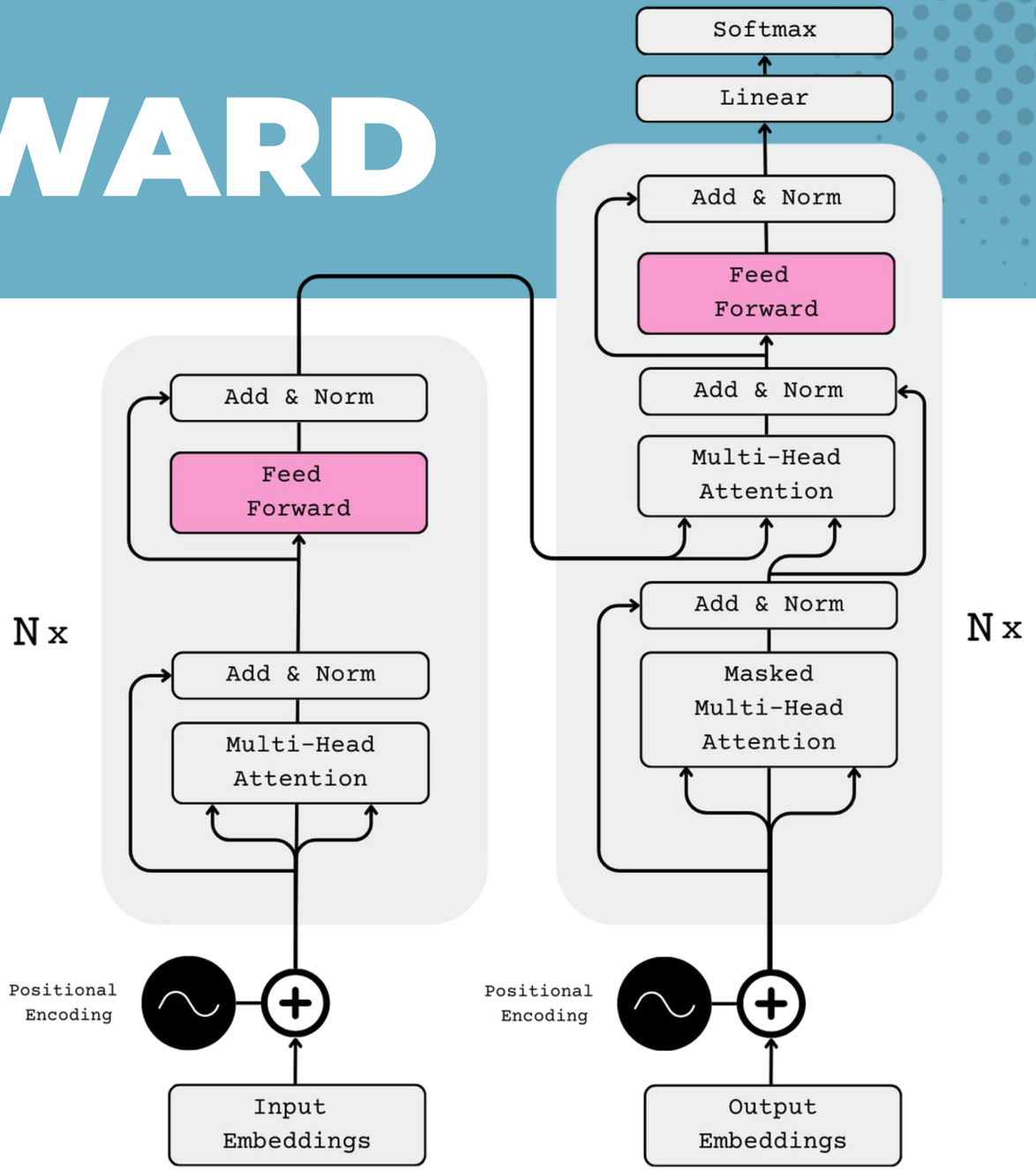
POS. ENCODING



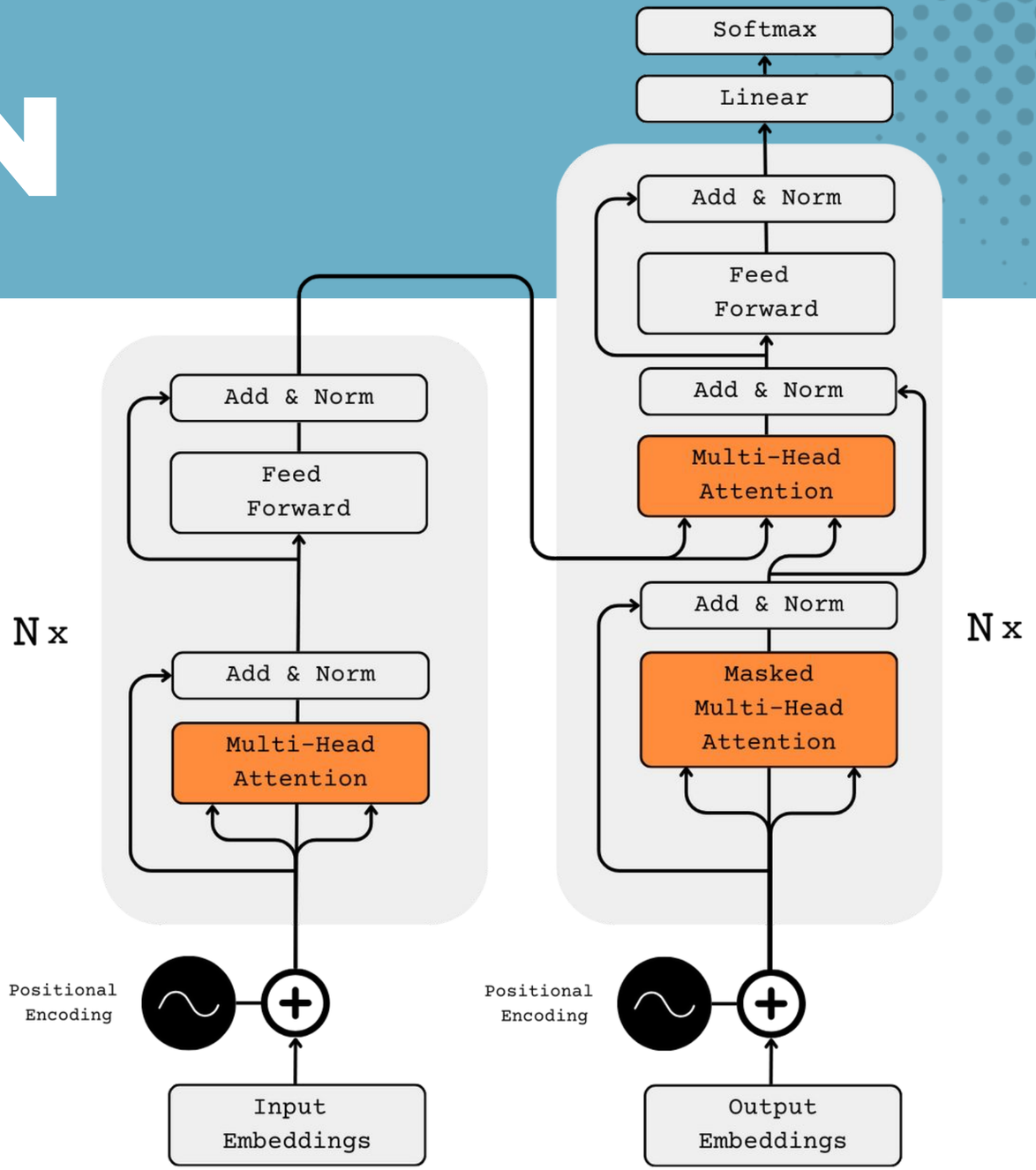
ADD & NORM


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N x
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N x
Positional Encoding
Positional Encoding
Input Embeddings
Output Embeddings

FEED FORWARD



ATTENTION


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N x
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N x
Positional Encoding
Positional Encoding
Input Embeddings
Output Embeddings

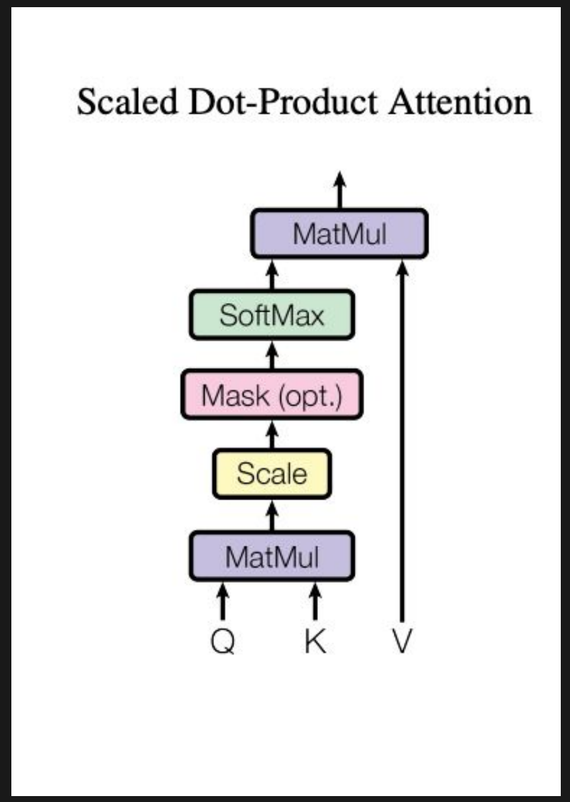
STEP-BY-STEP

- Tokens
- Embeddings
- Positional encoding
- Encoder
  - Multi-Head Attention
  - Add + Norm
  - Feed Forward
  - Add + Norm
- Decoder
  - Masked Muti-Head Attention
  - Add + Norm
  - Multi-Head Attention
  - Add + Norm
  - Feed Forward
  - Add + Norm
- Linear Projection

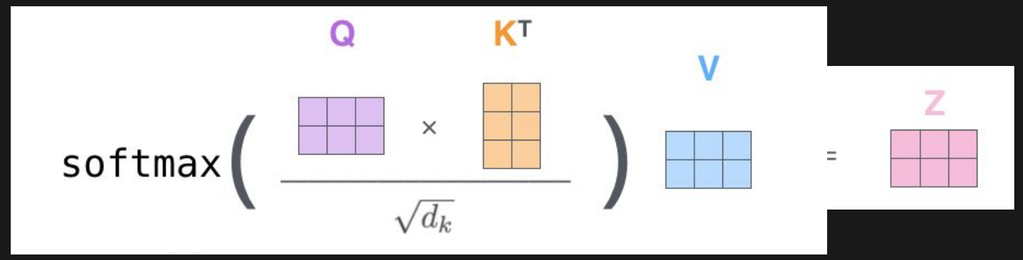
WITHIN ATTENTION

- Tokens
- Embeddings

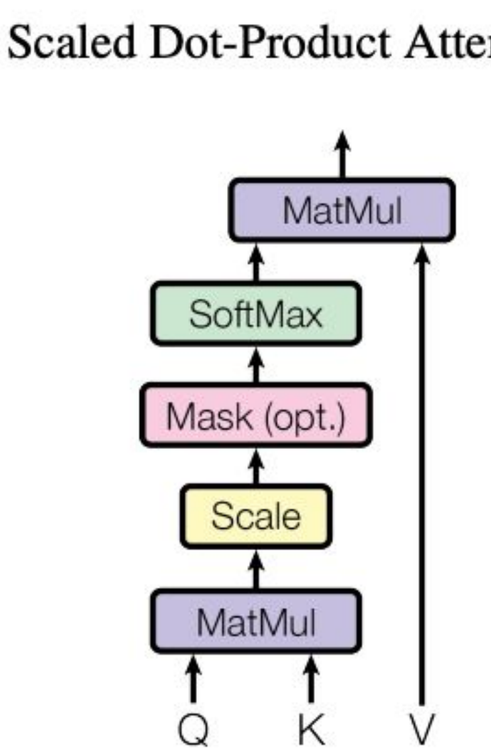
- Query, Keys, Values
  - Assemble matrices

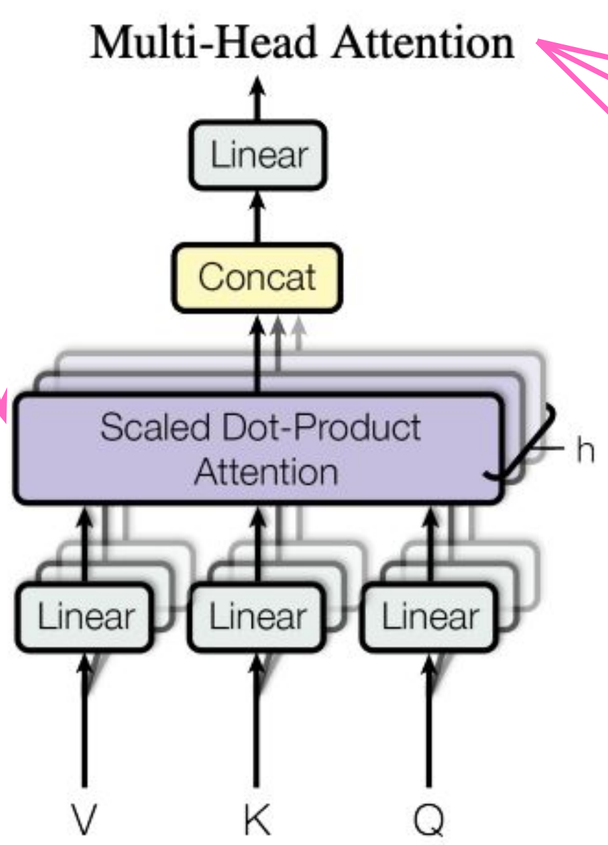


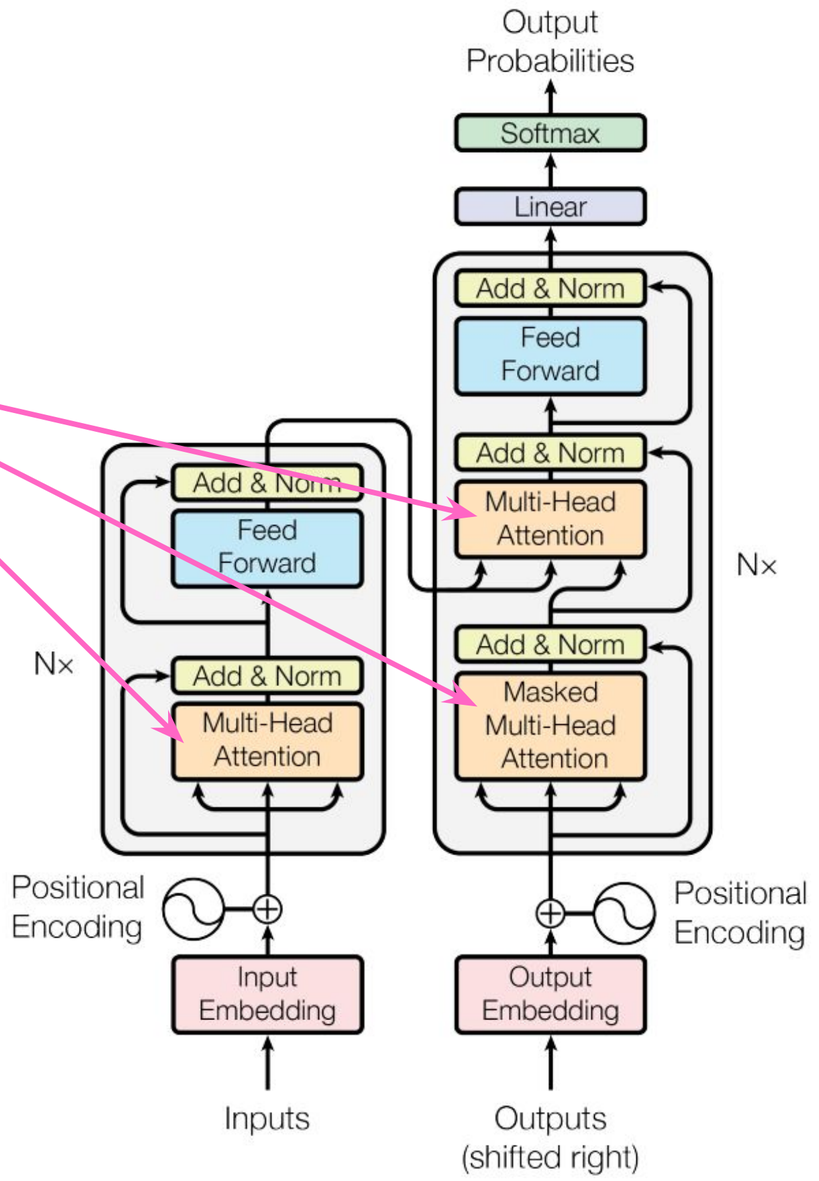
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



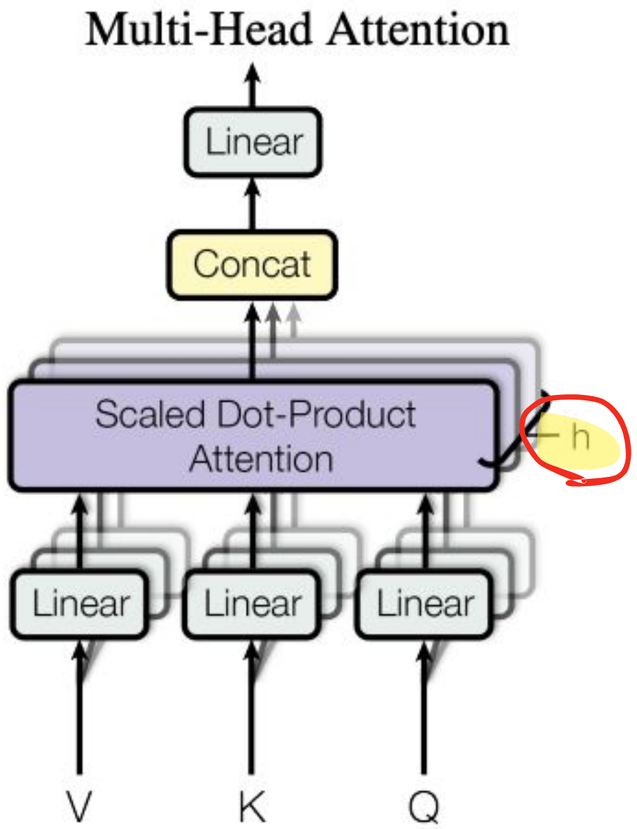
ATTENTION


Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V

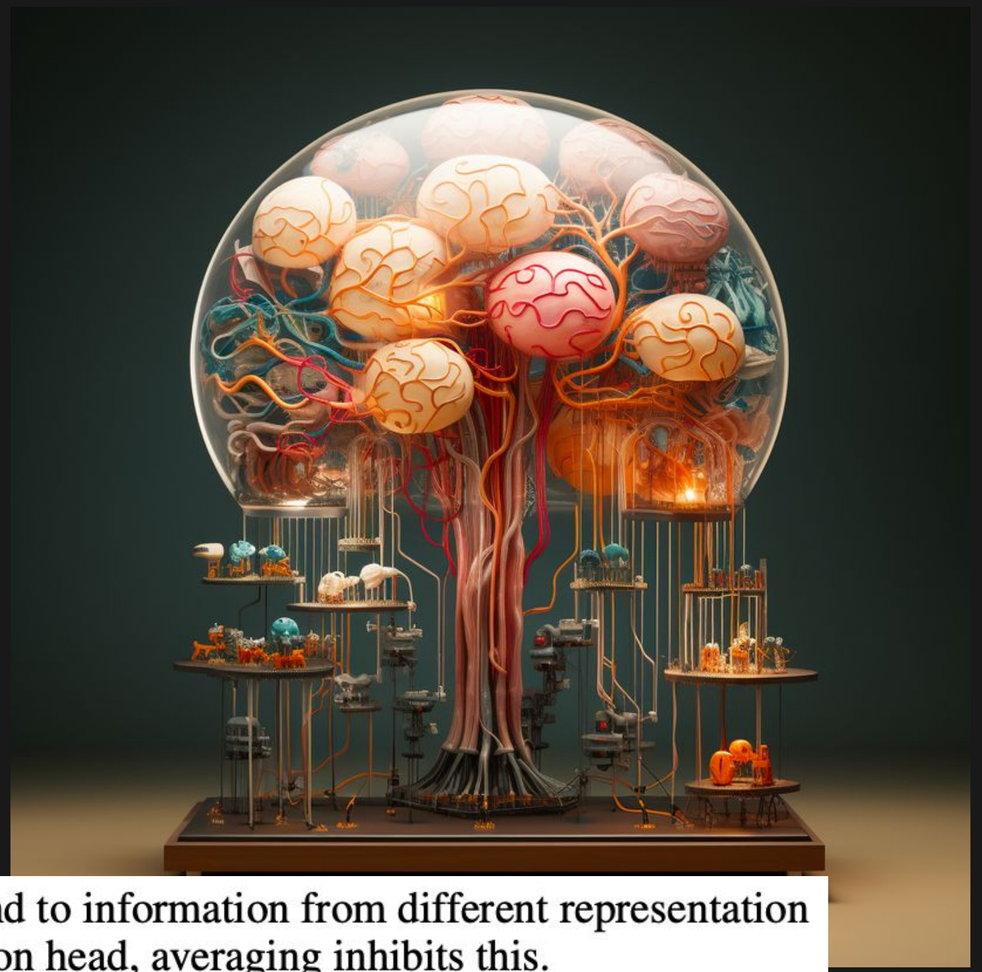

Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q


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

“MULTI-HEAD” ATTENTION

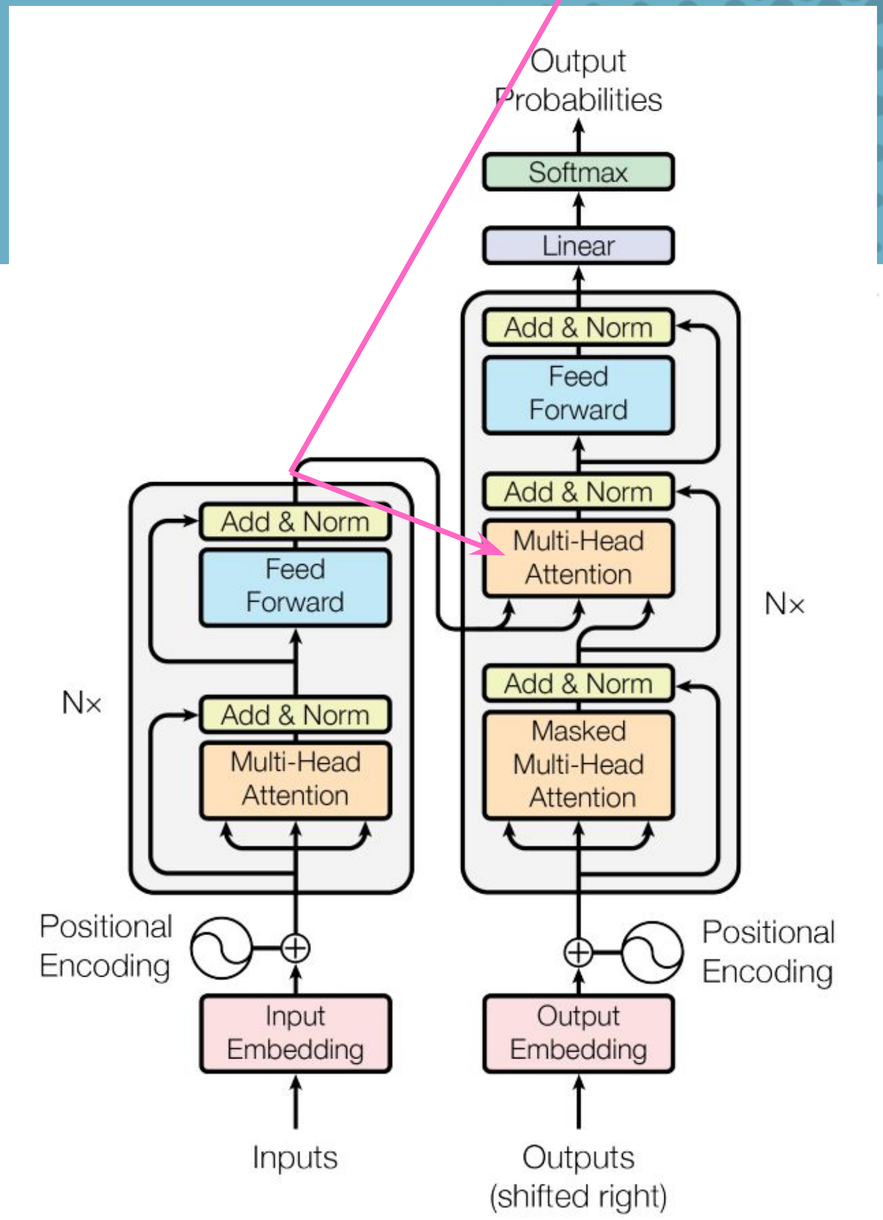


ATTENTION HEADS

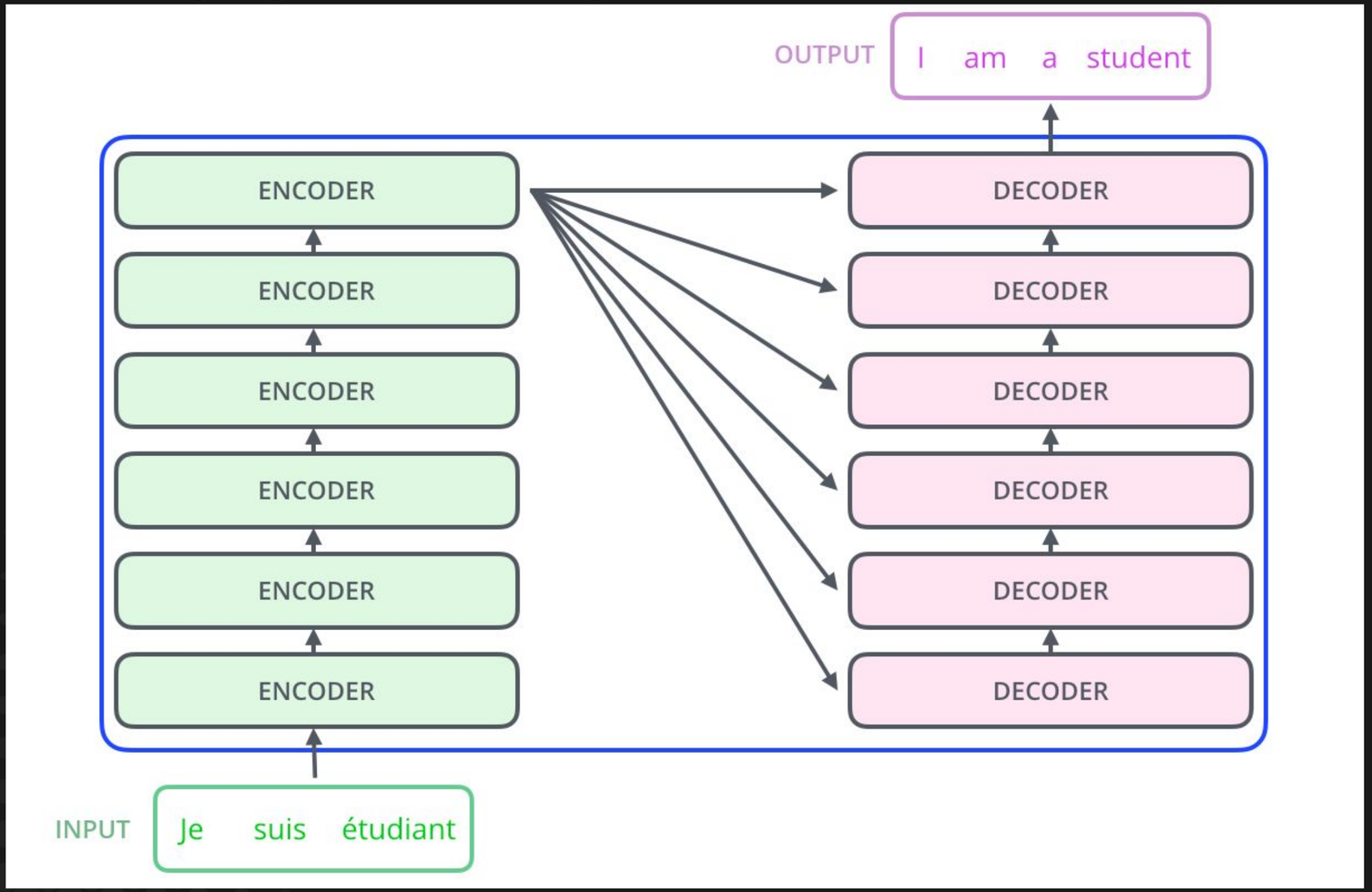


Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

TO DEC BLOCKS



STACKS OF ENCODERS & DECODERS





CONCLUSIONS

- **Transformer** = sequence transduction engine **based entirely on attention**
  - No convolution, recurrence!
  - BART (enc-dec), BERT (enc), and GPT (dec)
  - Only decoders are generative!
  - Multiple blocks --> each decoder block takes same encoder output

- Multi-headed attention allows us to focus on different pieces of the puzzle

TRANSFORMERS

- This is (not just)

  ChatGPT

