

AI504: Programming for Artificial Intelligence

Week 11: Transformer

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

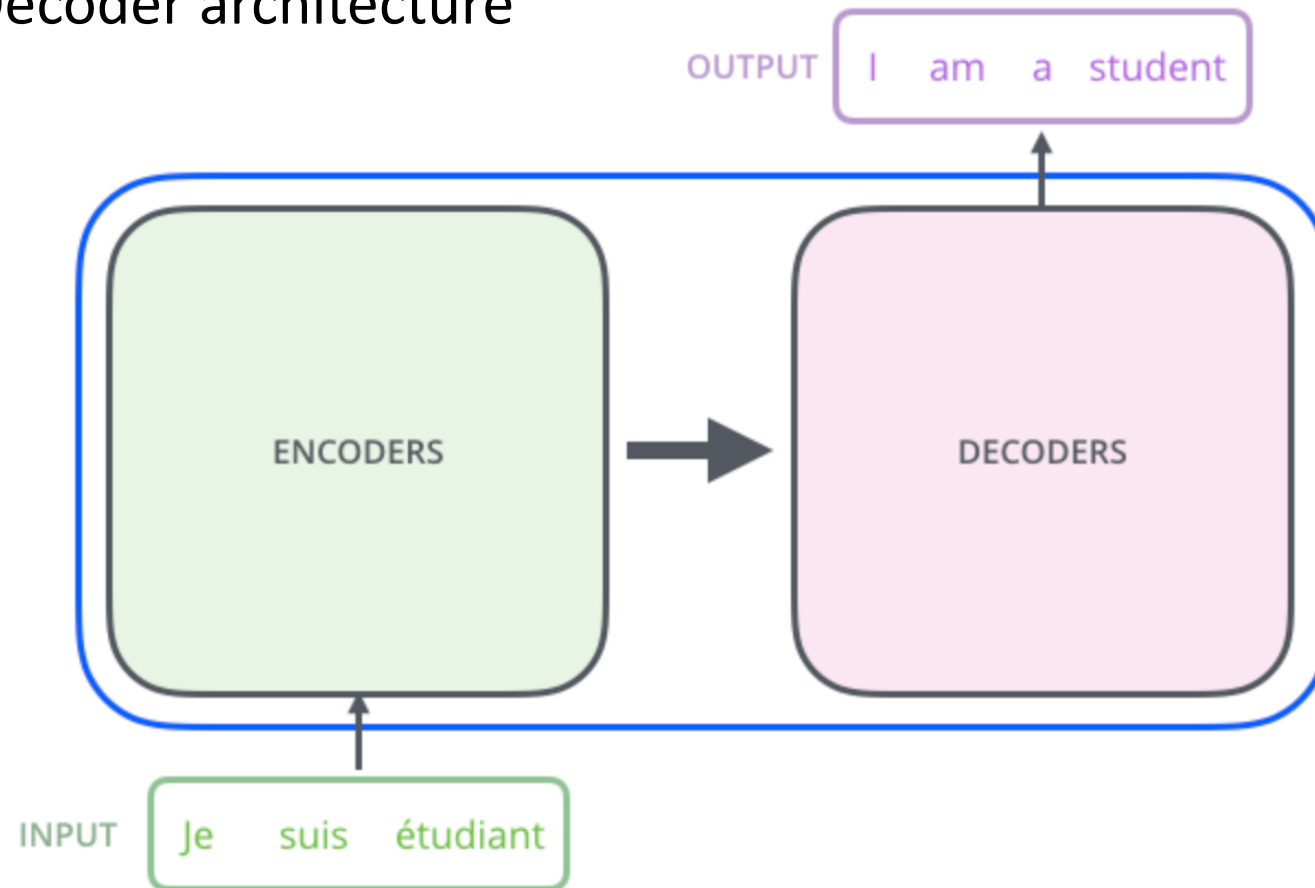
Index

- What is Transformer?
- What is Self-Attention?
 - What is Multi-head Attention?
- Positional Encoding
 - RNN, 1-D CNN, Transformers
- Seq-to-seq with Transformers
 - Masked Self-Attention

What is Transformer?

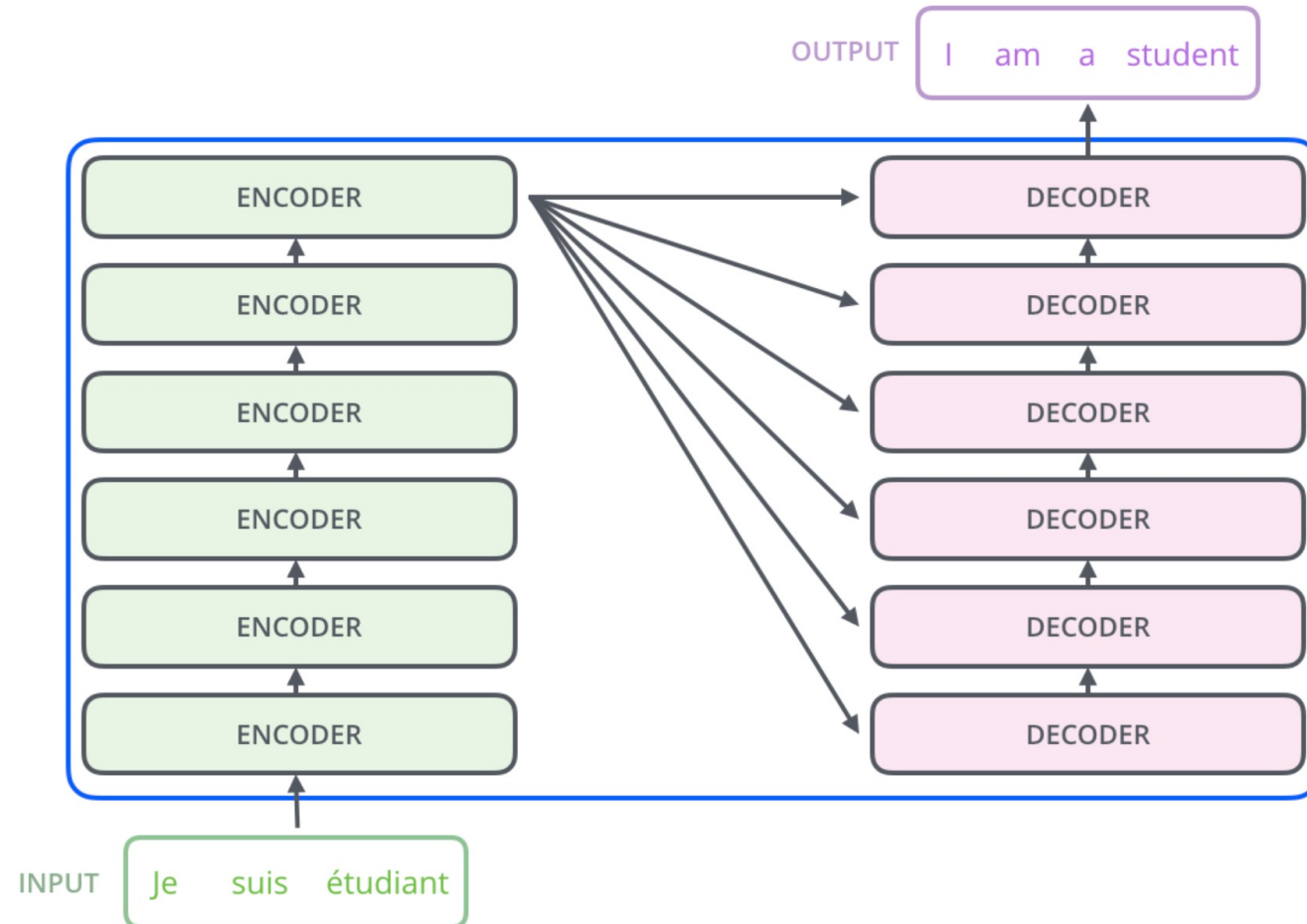
Transformers

- Originally designed for translation
 - Encoder-Decoder architecture



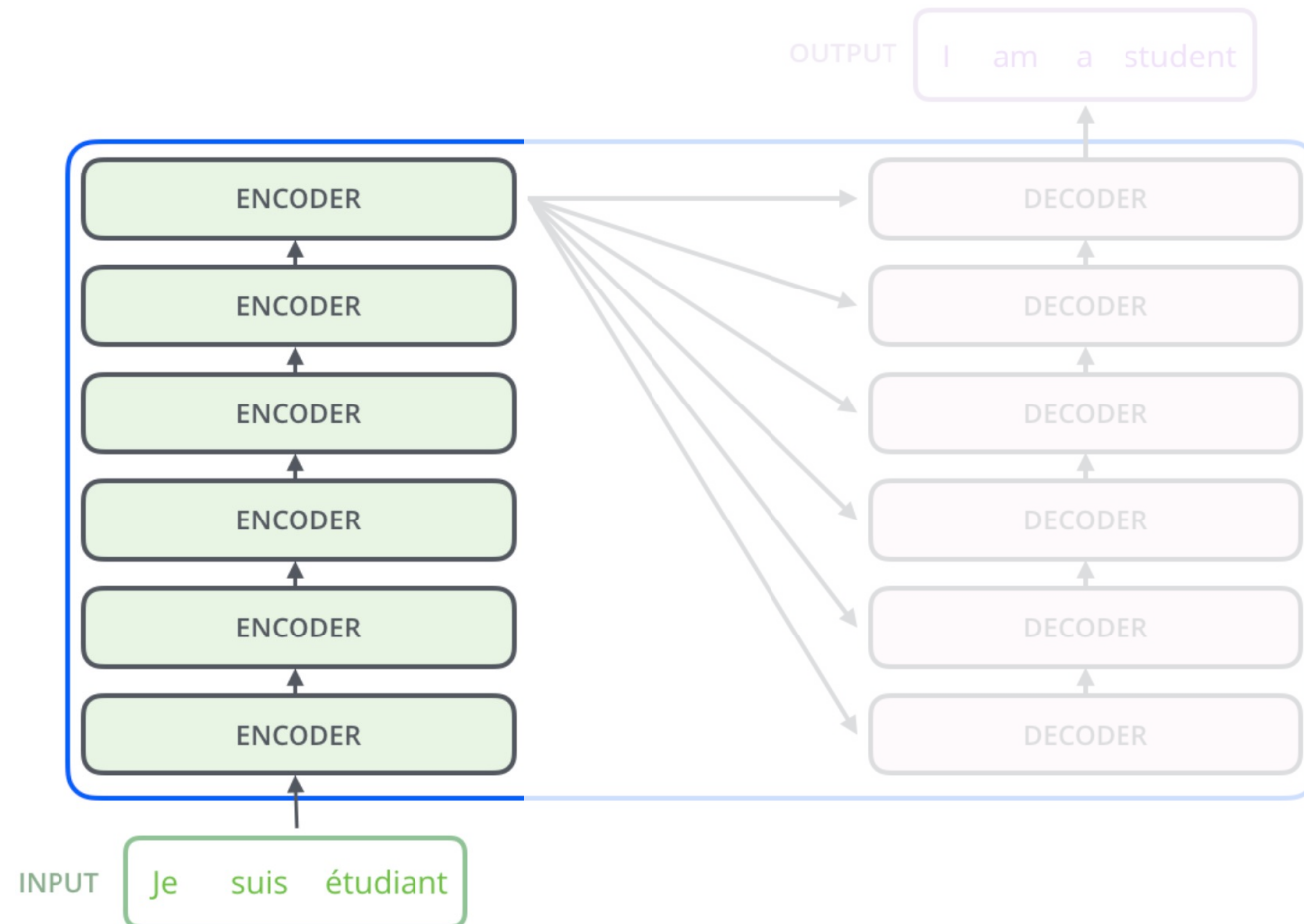
Transformers

- Encoder and decoder has many “blocks”.



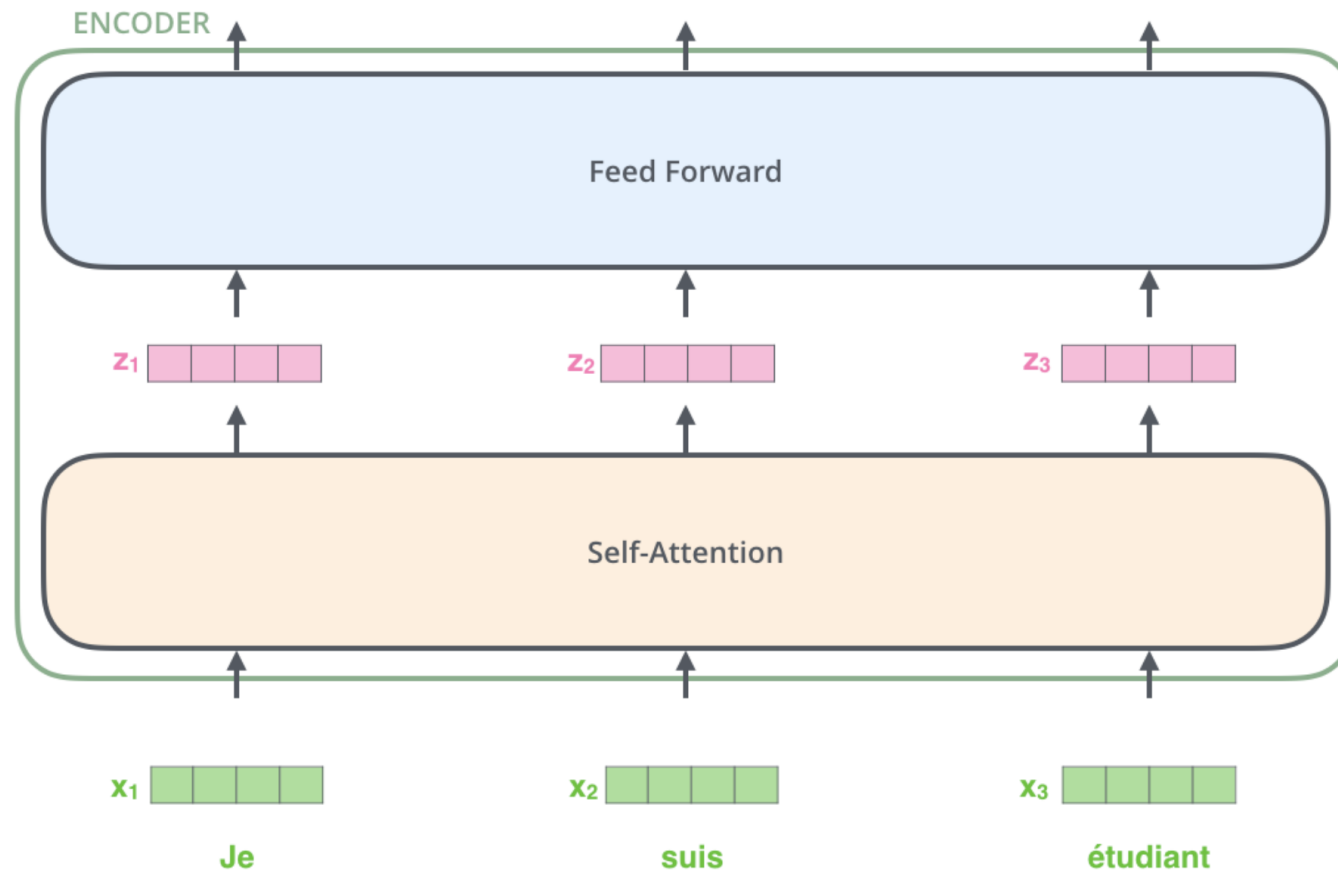
Transformers

- First focus on the **Encoder**



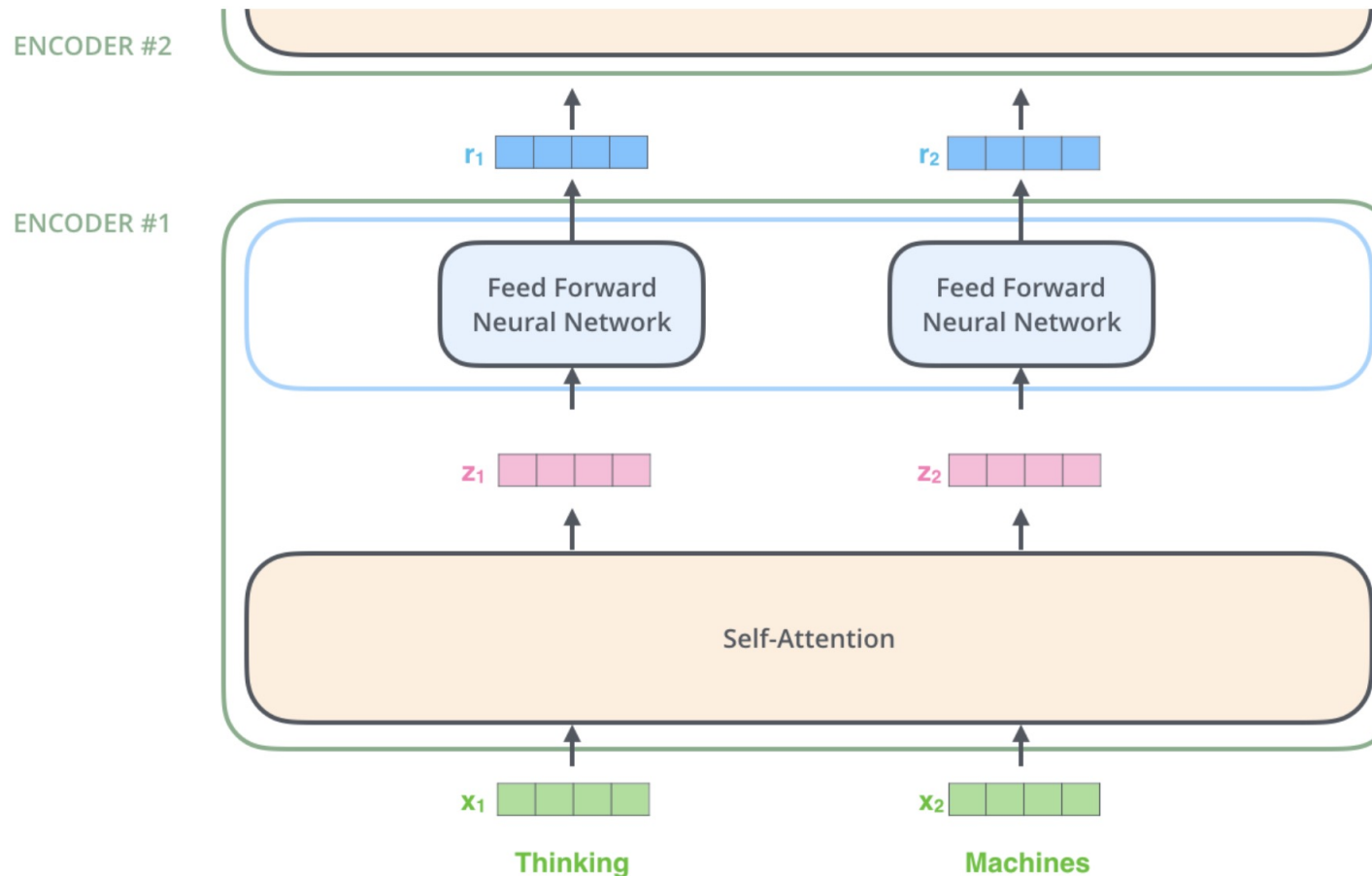
Encoder, Single Block

- Input (words) → Self-attention (QKV Op) → MLP → Output



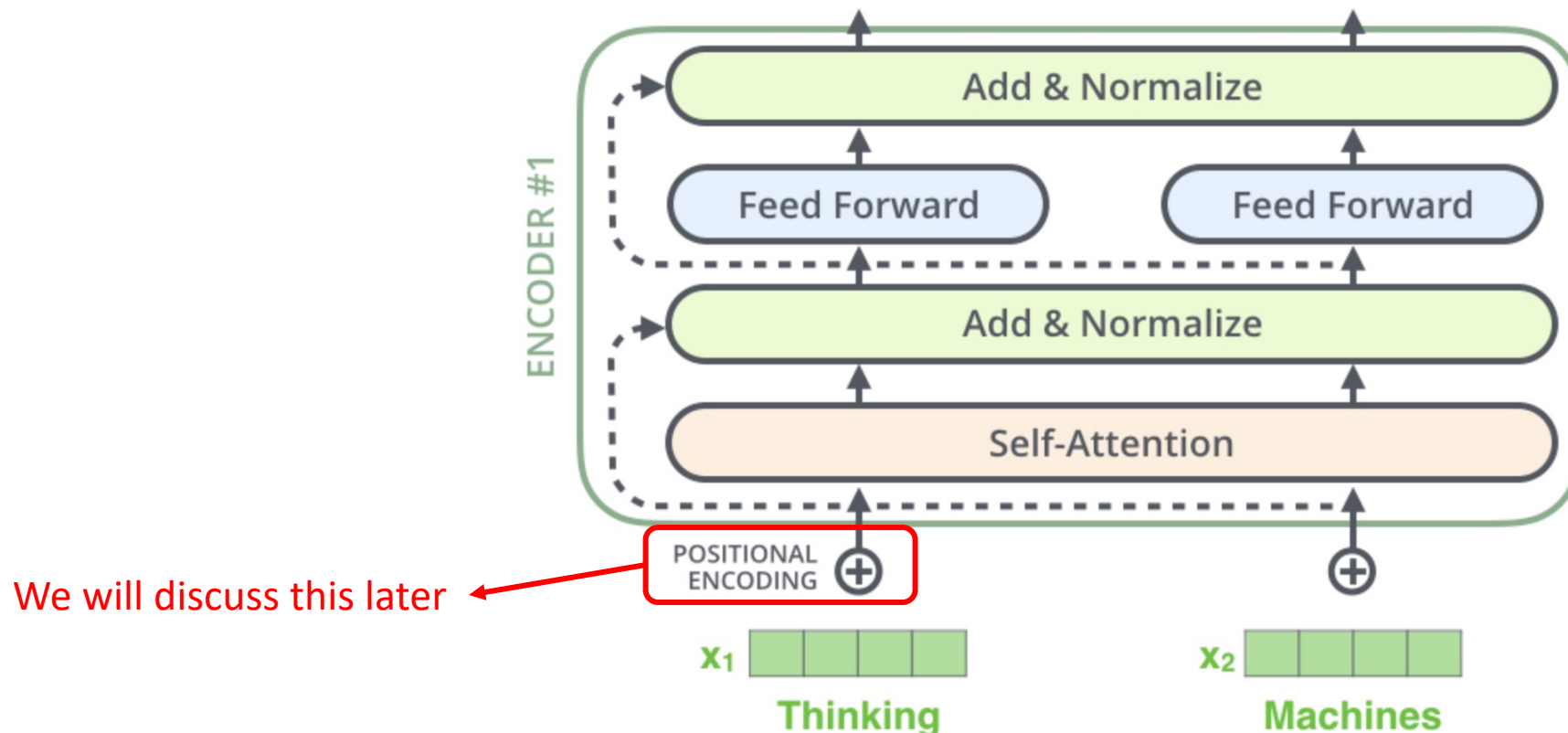
Encoder, Sequence of Blocks

- Input from previous block $\rightarrow$ Self-attention $\rightarrow$ MLP $\rightarrow$ Output



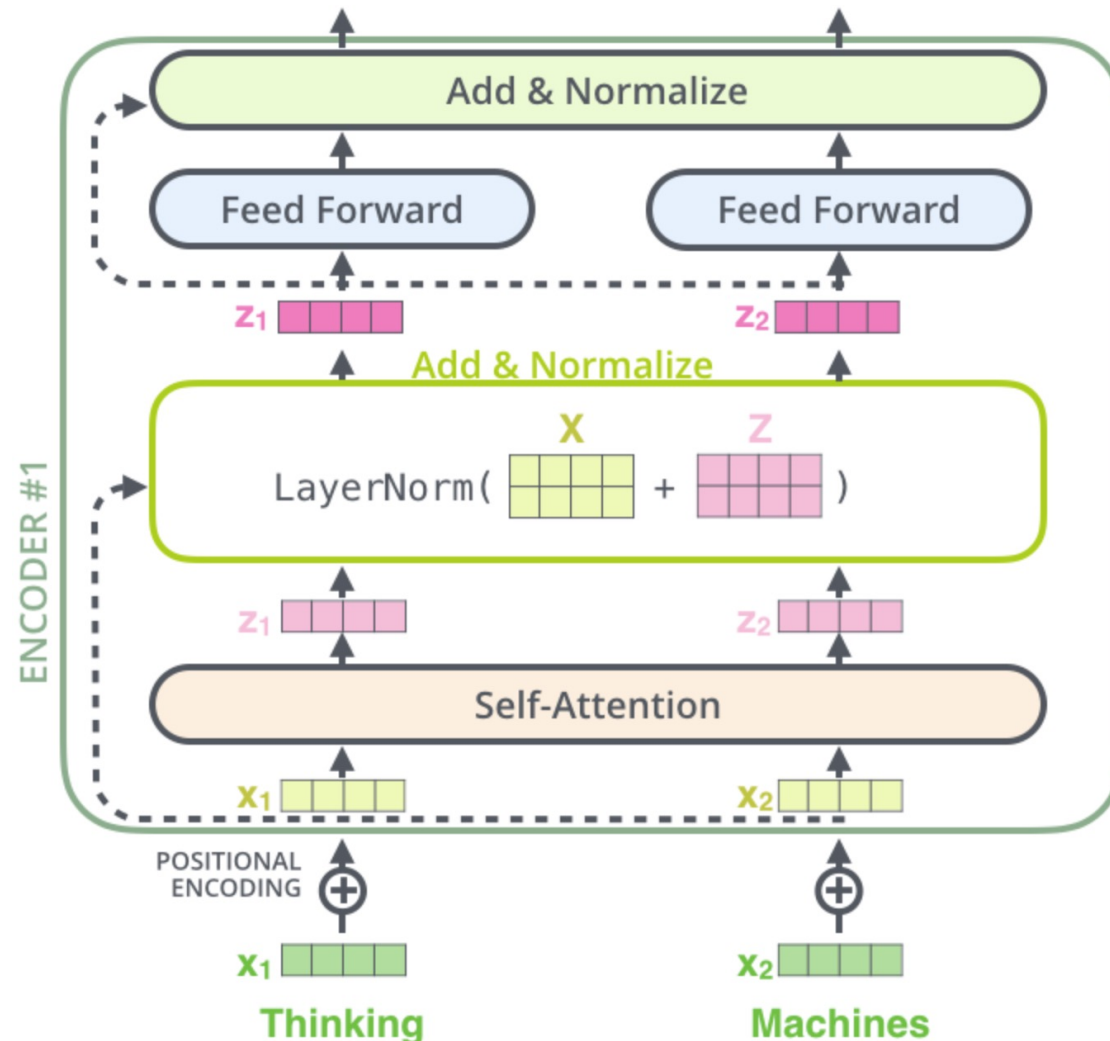
Encoder, Residual + Layer Norm

- Residual Connection (from ResNet)
- Layer Normalization (similar but different to BatchNorm)



Encoder, Residual + Layer Norm

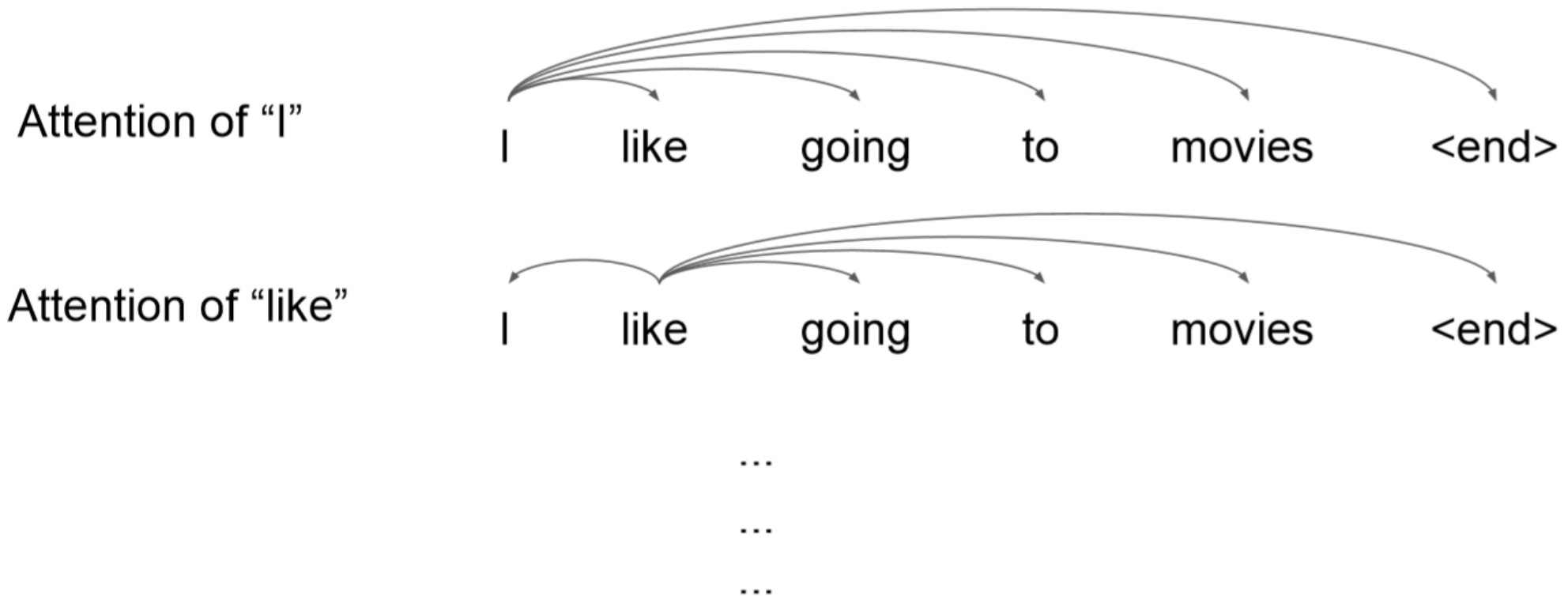
- Visualizing the Vectors



What is Self-Attention?

Attention is All You Need

- Vaswani et al. 2017
- Let's use only attentions to handle sequences.



Self-Attention

- $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right) \mathbf{V}$

0.5	0.1	0.0	0.2	0.2	0.0
0.2	0.6	0.0	0.0	0.2	0.0
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

$$Softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)$$

I
like
going
to
movies
<end>

$\mathbf{V}$

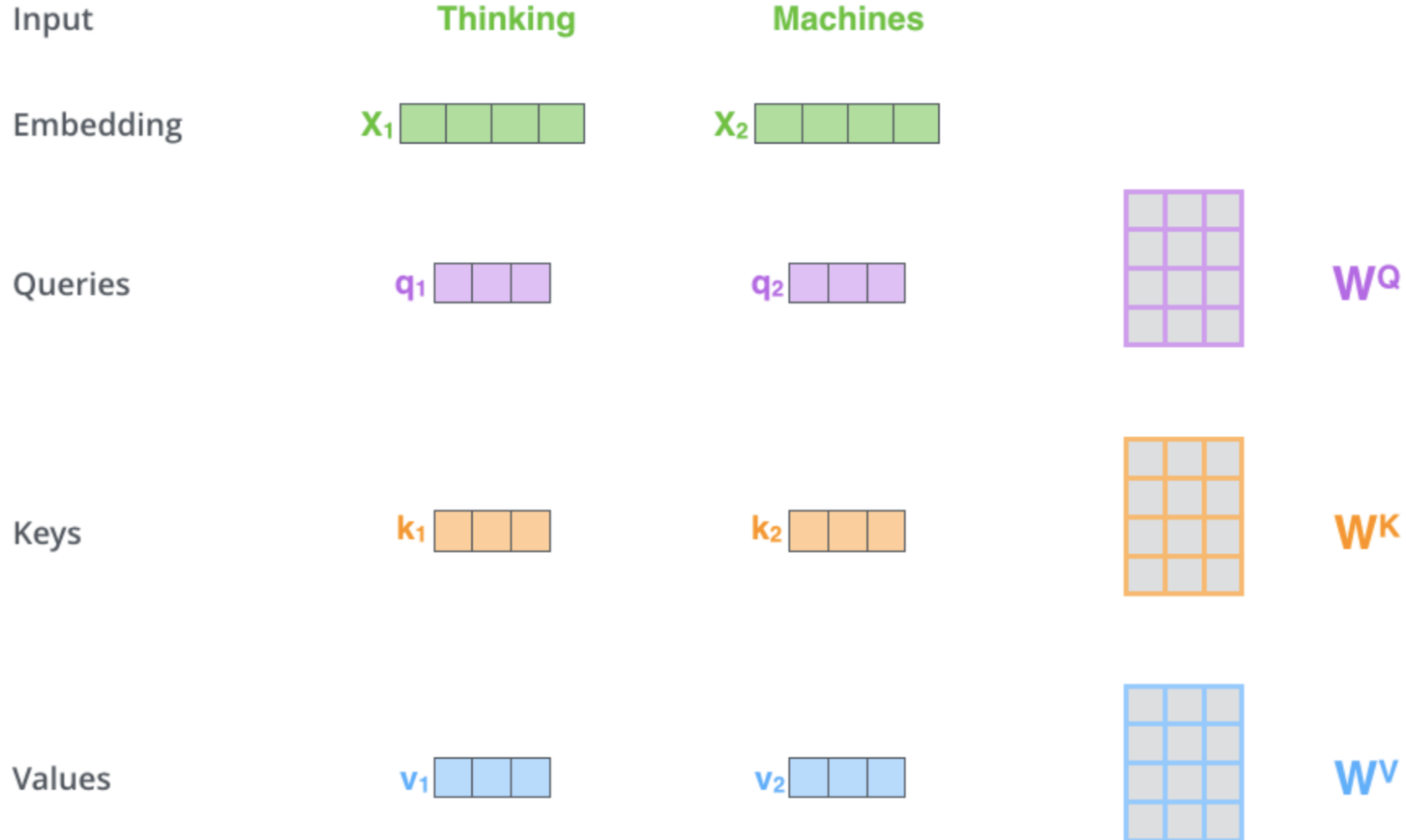


0.5*I + 0.1*like + 0.2*to + 0.2* movies
0.2*I + 0.6*like + 0.2*movies
...
...
...
...

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

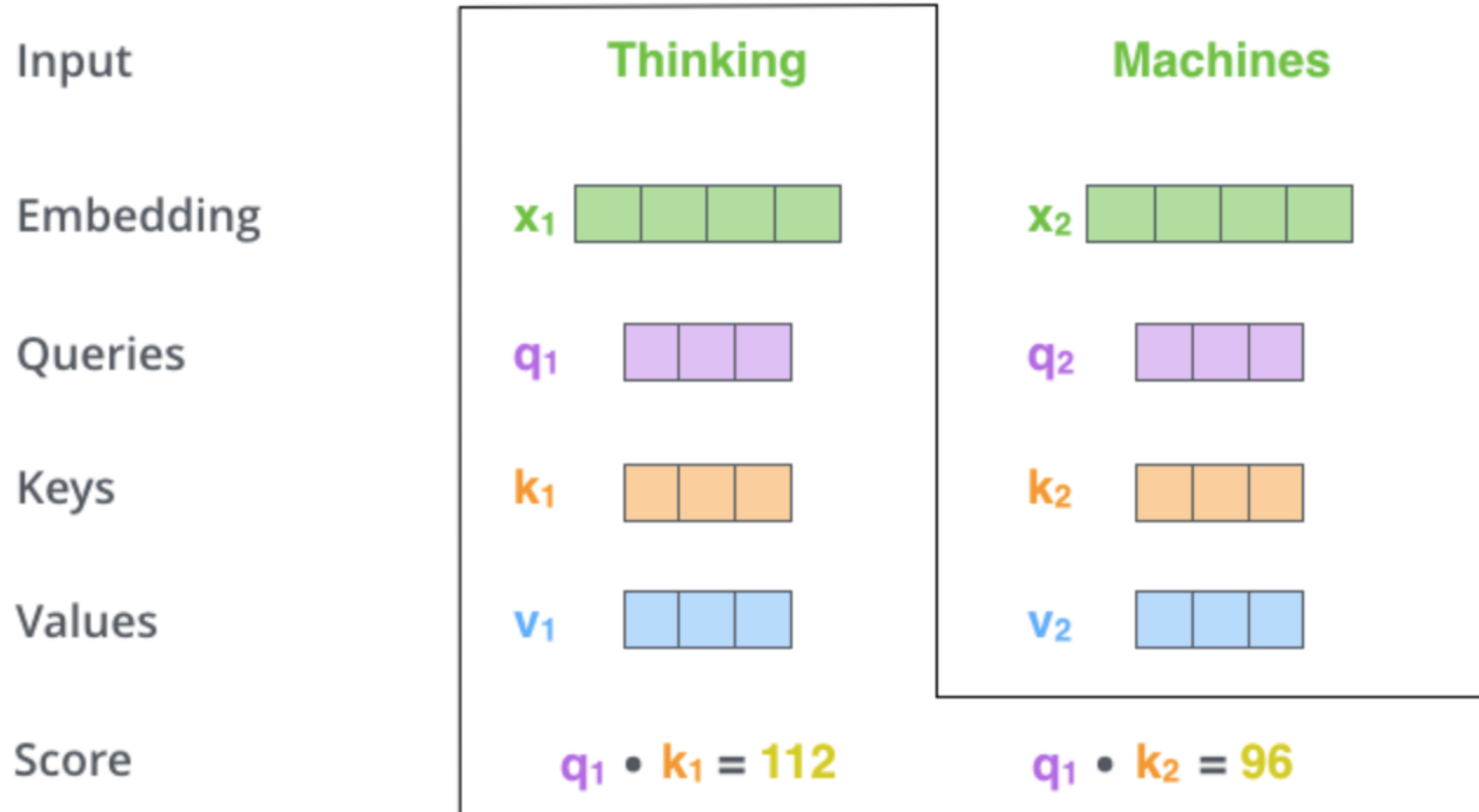
QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Generating Queries, Keys, Values



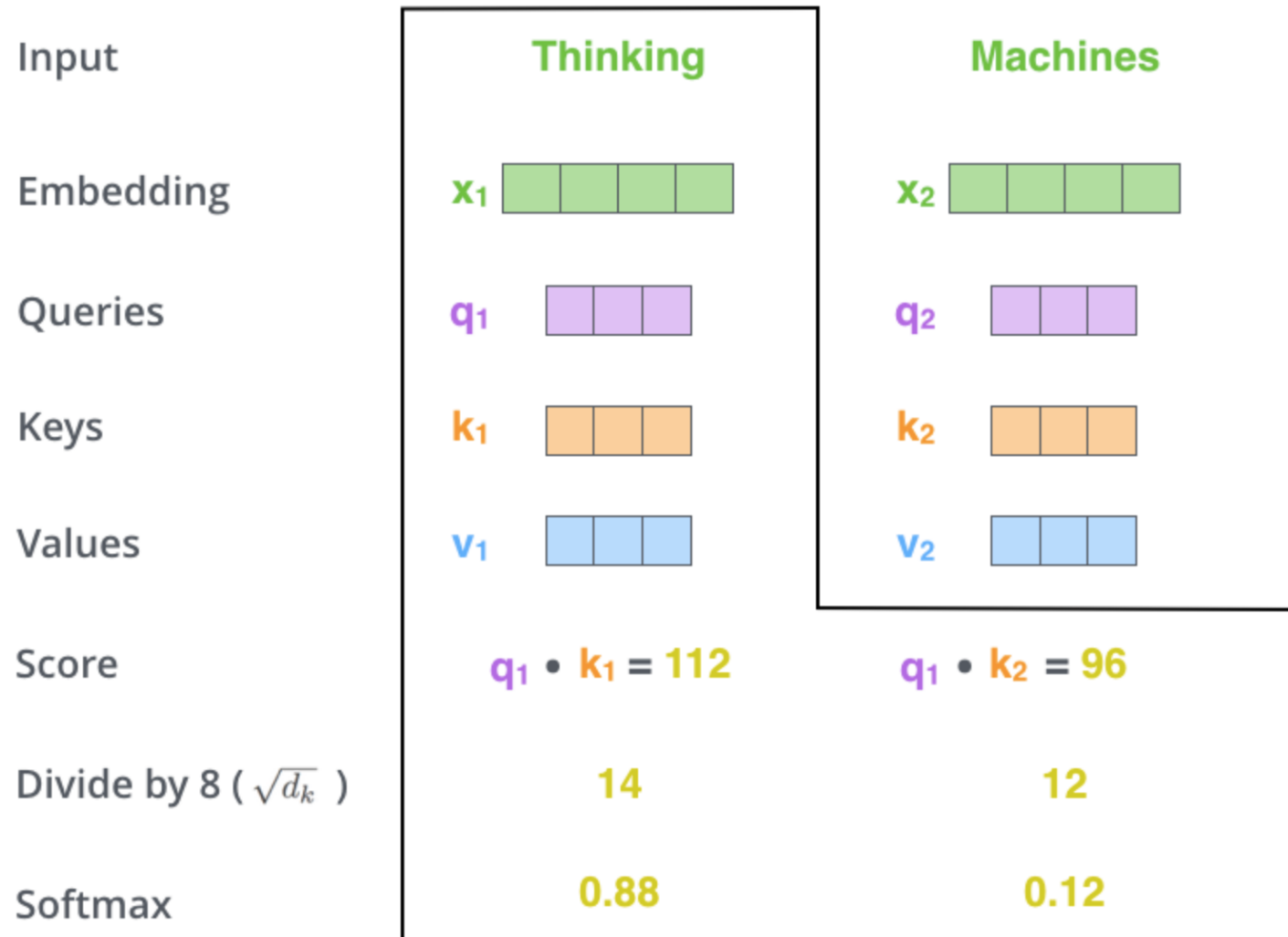
QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Attention from “Thinking”



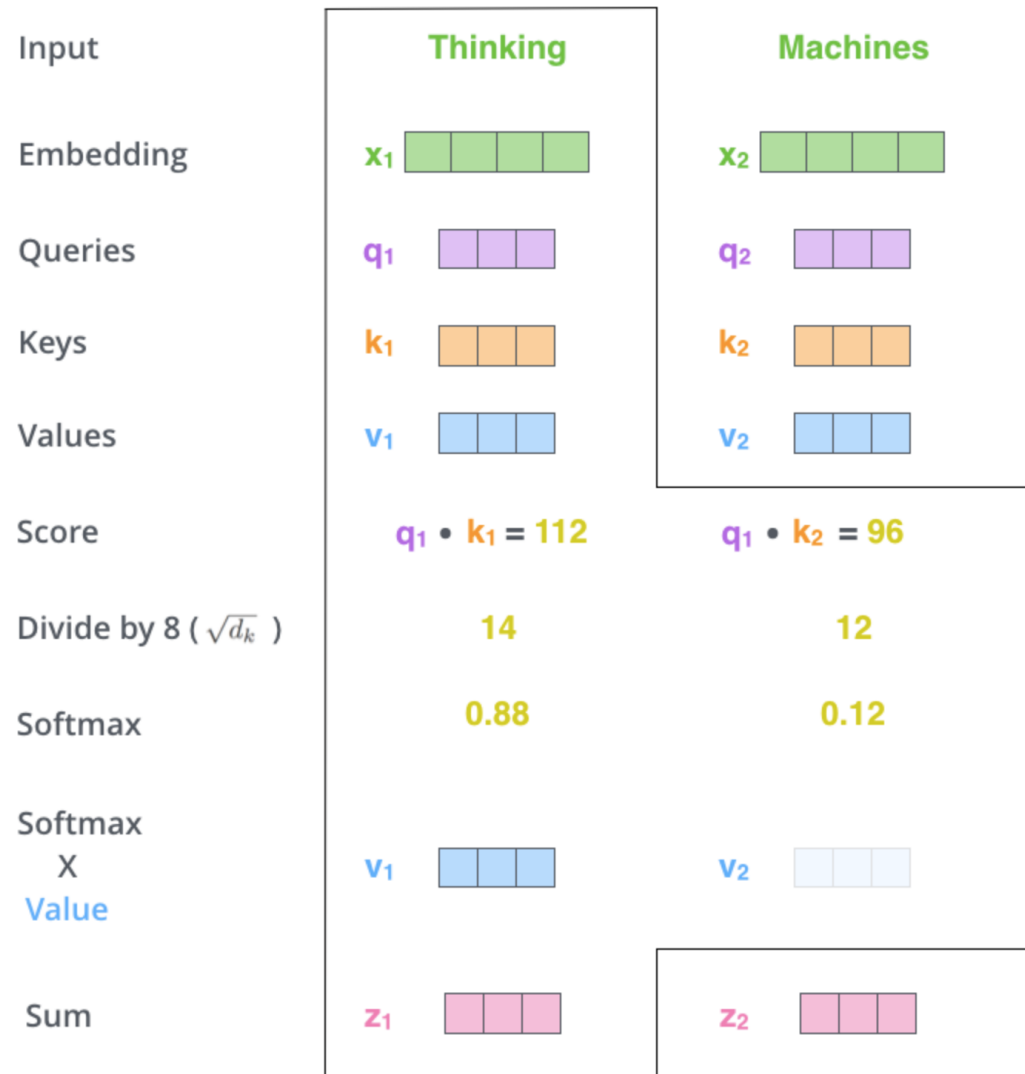
QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Attention from “Thinking”



QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Attention from “Thinking”



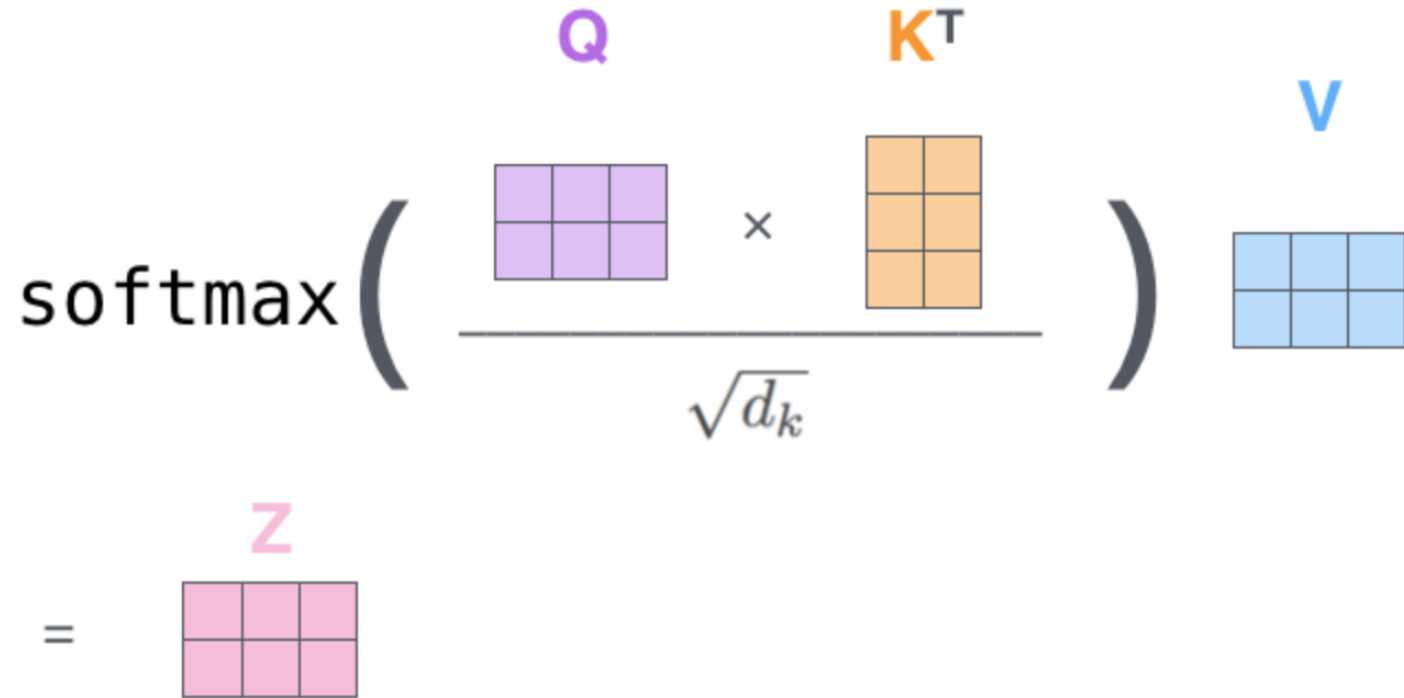
QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Processing all words at the same time.



QKV Operation: $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

Processing all words at the same time.



Self-Attention

- $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}$

0.5	0.1	0.0	0.2	0.2	0.0
0.2	0.6	0.0	0.0	0.2	0.0
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

$$Softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)$$

I
like
going
to
movies
<end>

$\mathbf{V}$



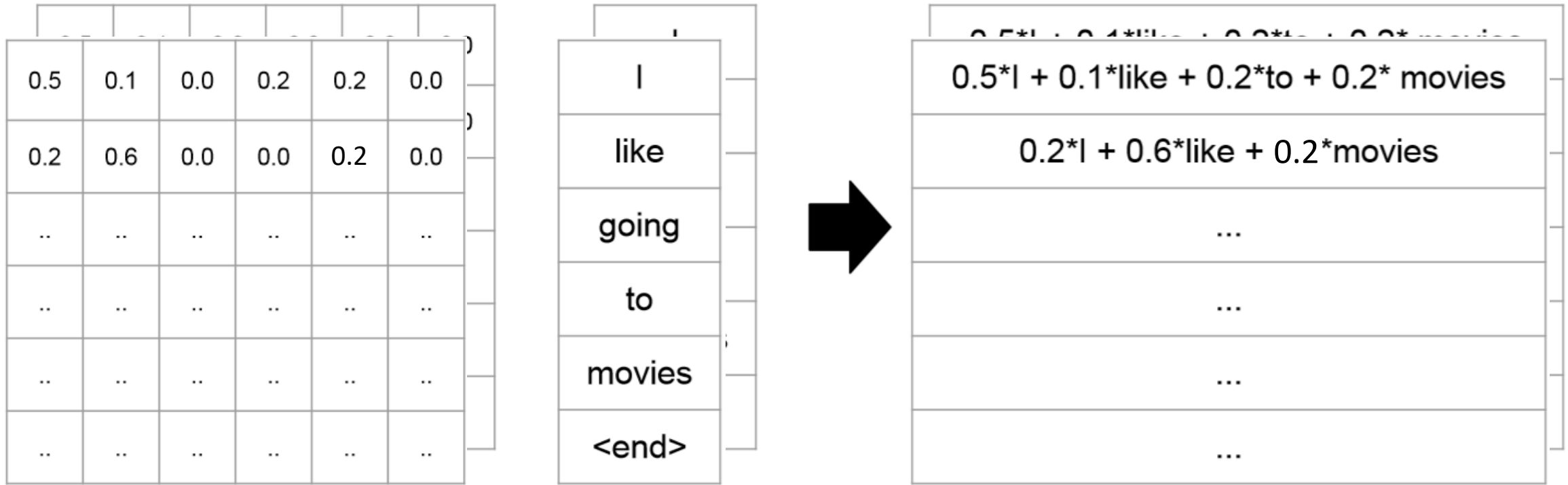
0.5*I + 0.1*like + 0.2*to + 0.2* movies
0.2*I + 0.6*like + 0.2*movies
...
...
...
...

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

What is Multi-head Attention?

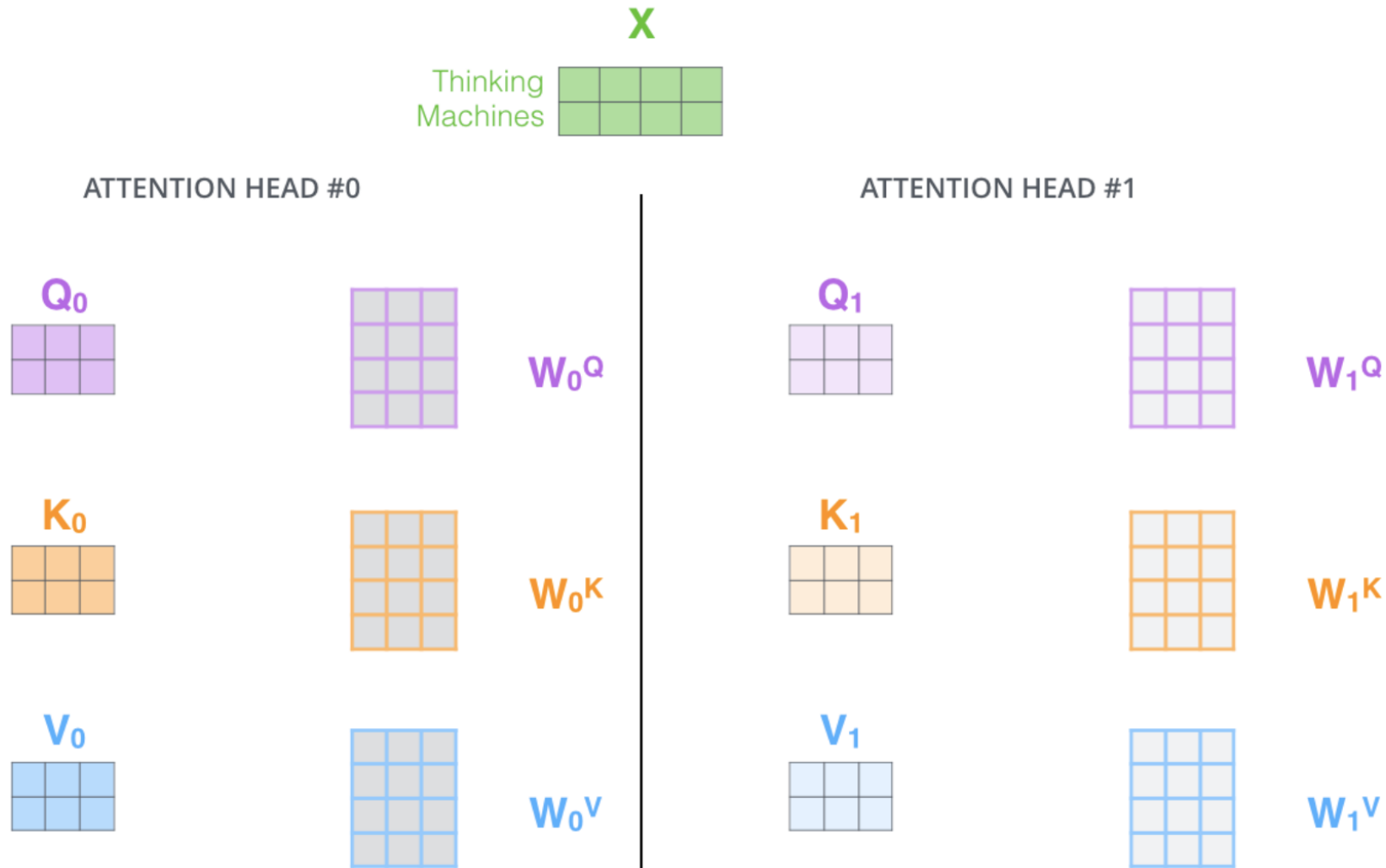
Multiple Self-Attention

- What if we used 2 attention maps instead of 1?



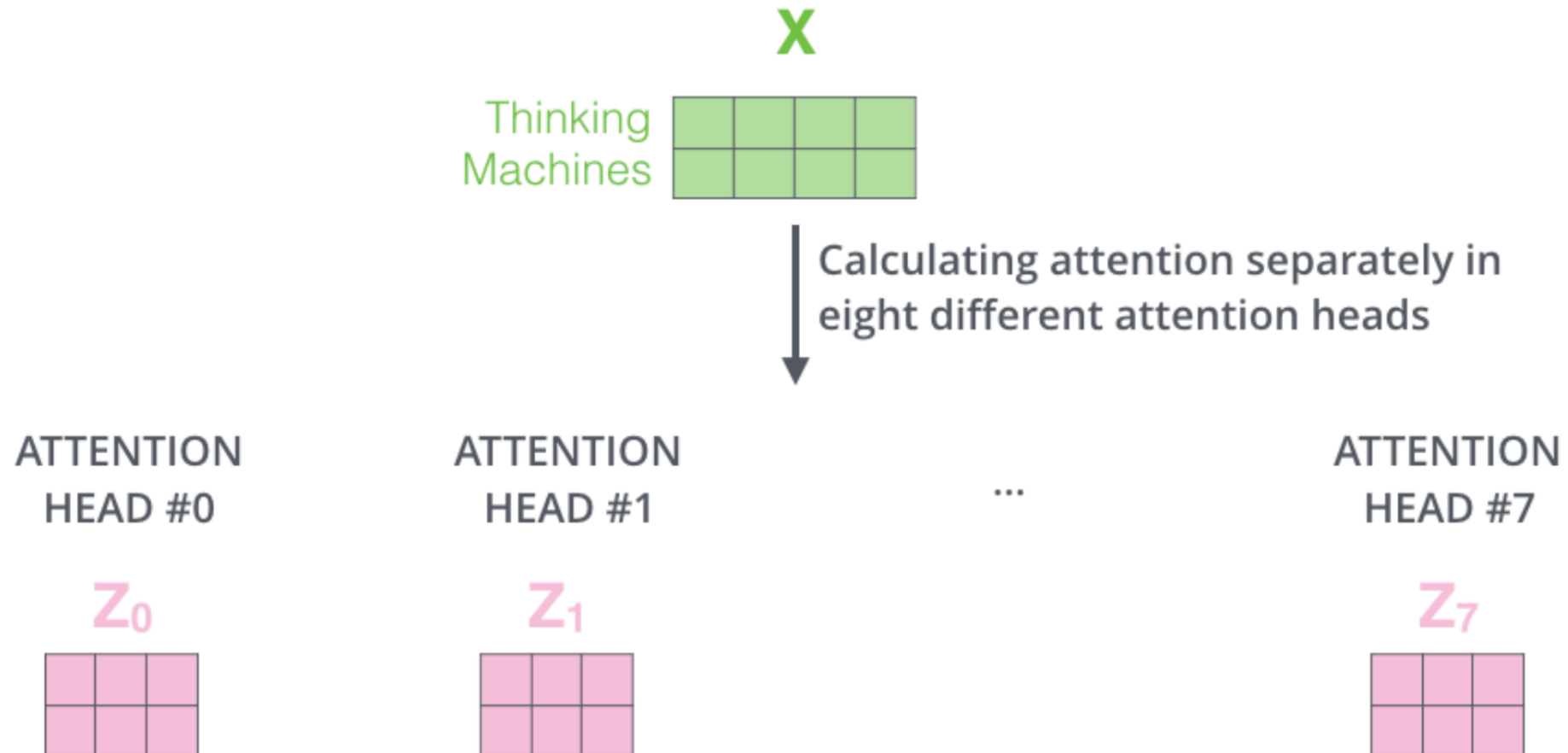
Multiple Self-Attention

Multiple QKV matrices



Multiple Self-Attention

Multiple (8) Self-Attention Outputs



Multiple Self-Attention

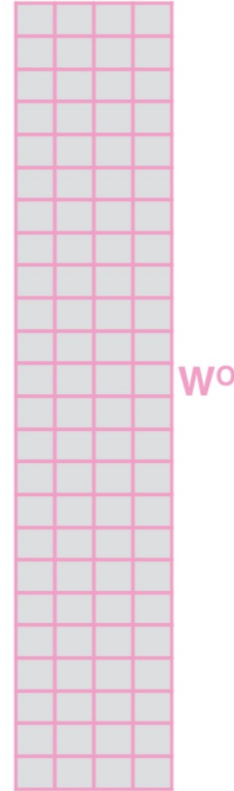
Merging Multiple Outputs into One

1) Concatenate all the attention heads



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

$\times$



W^O

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

$=$



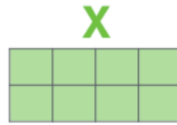
Multiple Self-Attention

From Start to Finish

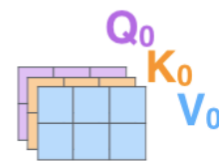
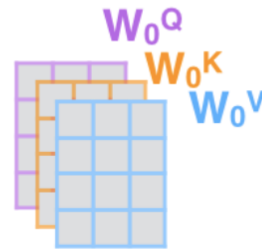
1) This is our input sentence*

Thinking
Machines

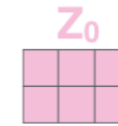
2) We embed each word*



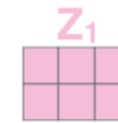
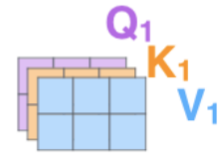
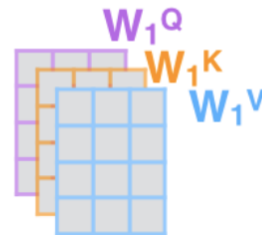
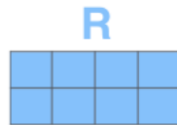
3) Split into 8 heads. We multiply X or R with weight matrices



5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



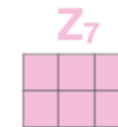
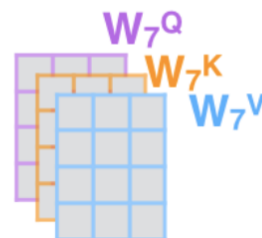
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



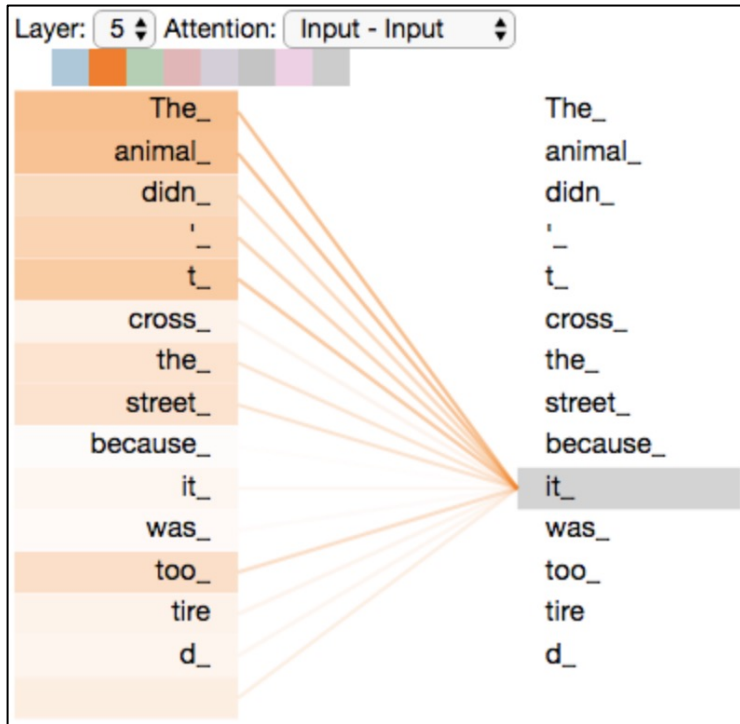
...

...

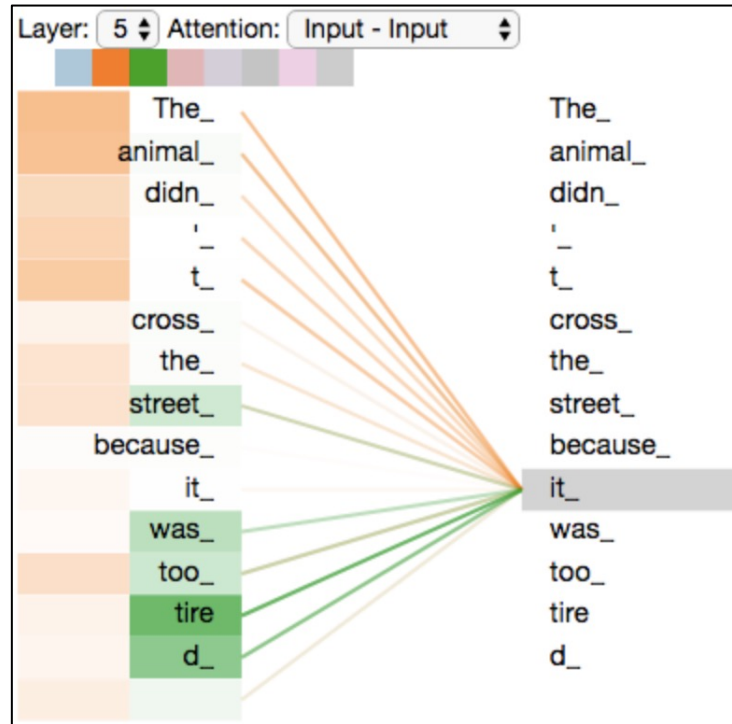
...



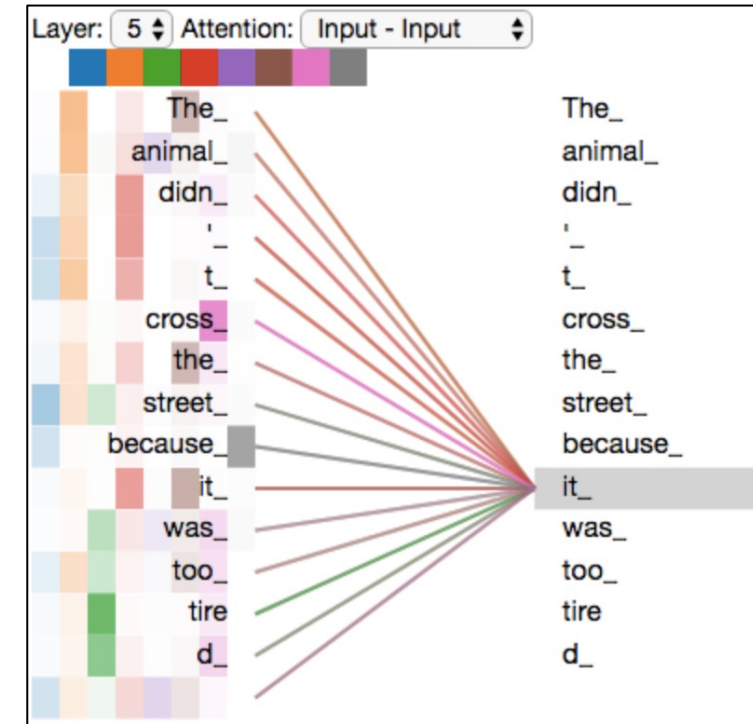
Visualizing the Attention



Visualizing 1 attention



Visualizing 2 attentions



Visualizing 8 attentions

Positional Encoding

Self-Attention = Set Encoding

- Transformer Encoder is inherently a set encoder.

0.5	0.1	0.0	0.2	0.2	0.0
0.2	0.6	0.0	0.0	0.2	0.0
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

I
like
going
to
movies
<end>



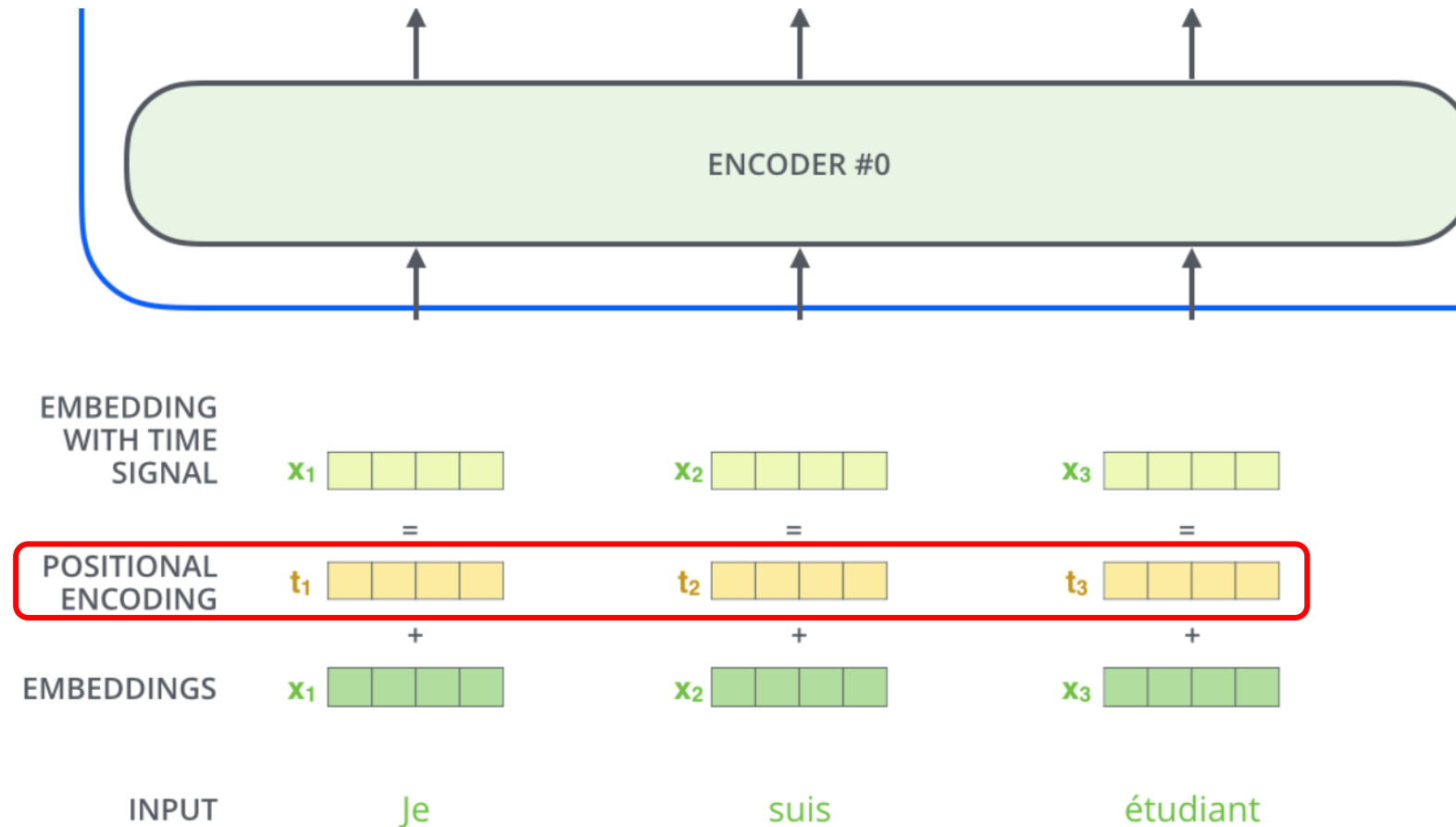
$0.5*I + 0.1*like + 0.2*to + 0.2* movies$
$0.2*I + 0.6*like + 0.2*movies$
...
...
...
...



No concept of order.

Positional Encoding

- We need to **inject** order information.



Position Encoding Requirement

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

Naïve Examples

- Training sample

Sentence	Dark	horses	are	faster	than	white	horses
Pos 1	1	2	3	4	5	6	7
Pos 2	0.14	0.28	0.42	0.56	0.70	0.84	1.00

- Test sample

Sentence	Dark	horses	might	be	faster	than	white	horses
Pos 1	1	2	3	4	5	6	7	8
Pos 2	0.12	0.25	0.37	0.50	0.62	0.75	0.87	1.00

Discrete Example

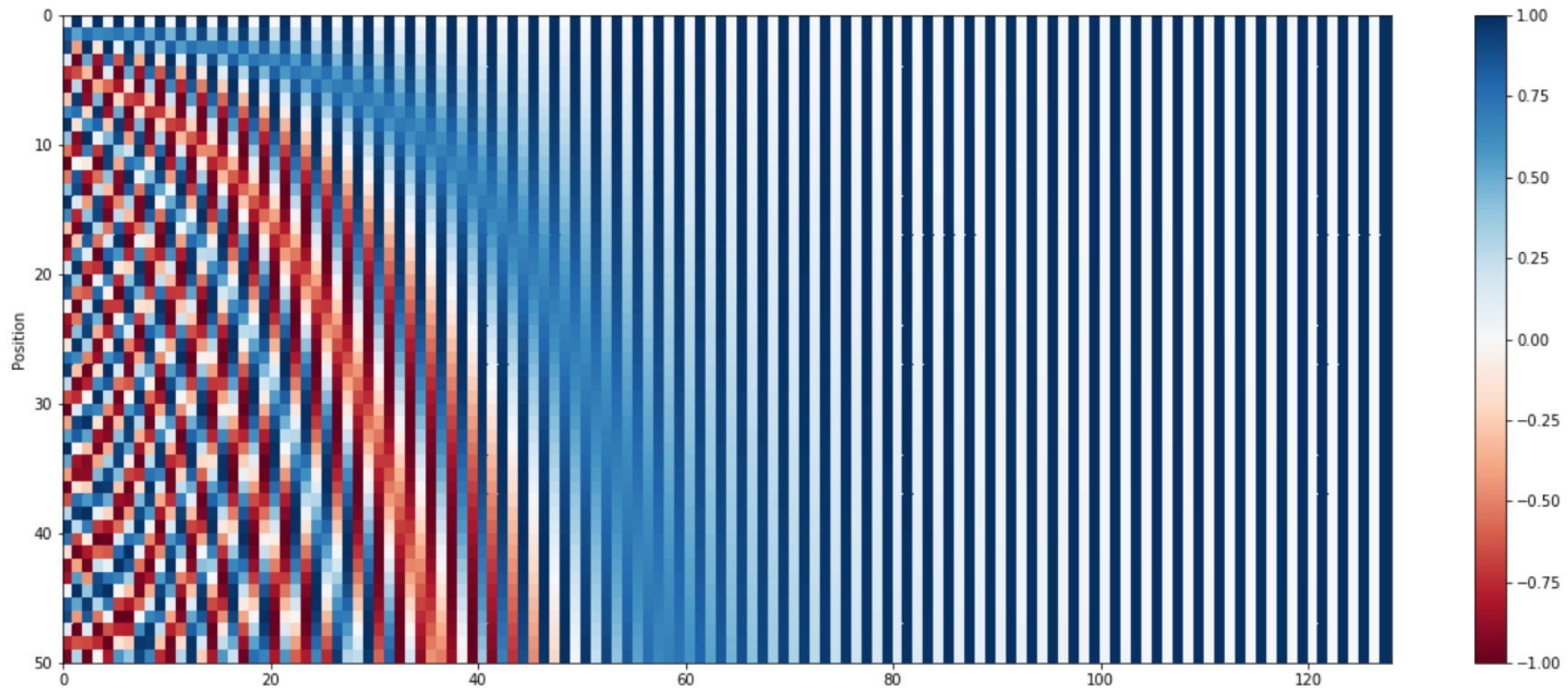
- Use binary values

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	2 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

- But binary values waste space

Continuous Encoding

- Use sinusoidal functions

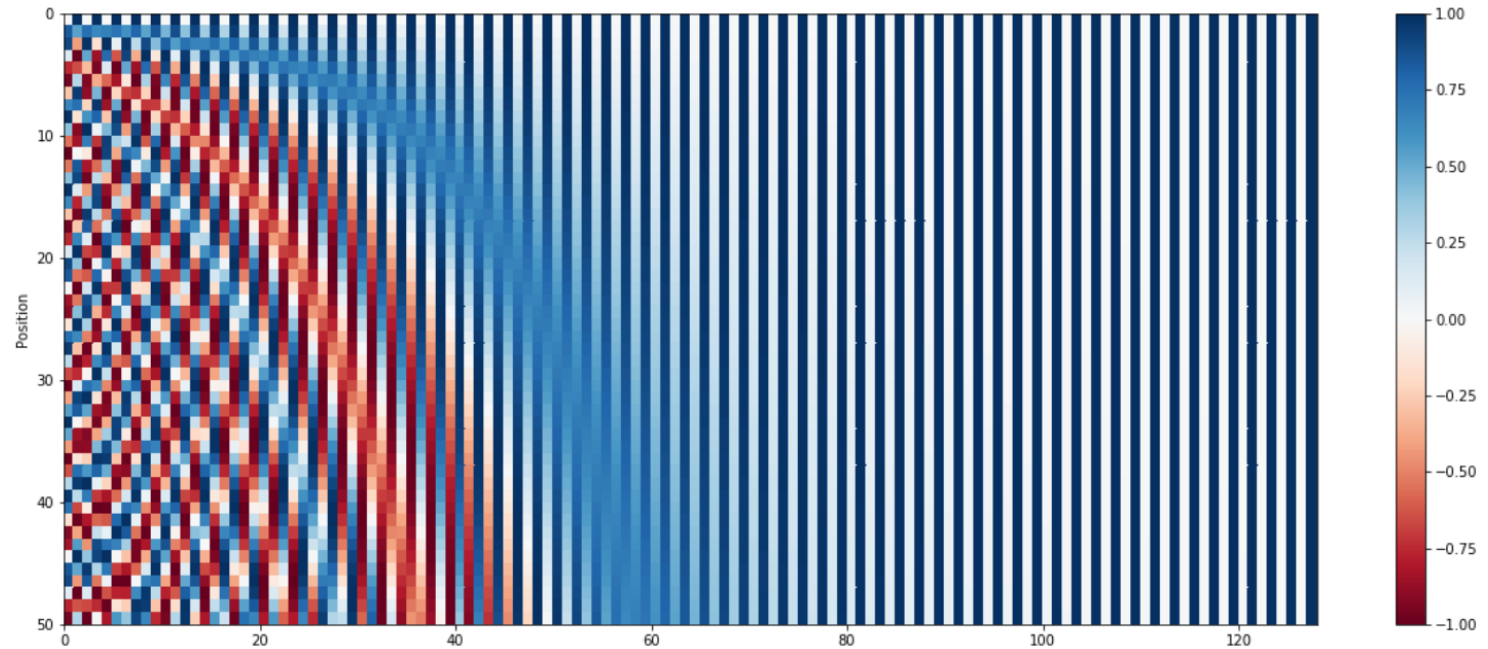


Positional Encoding

- Use sinusoidal functions

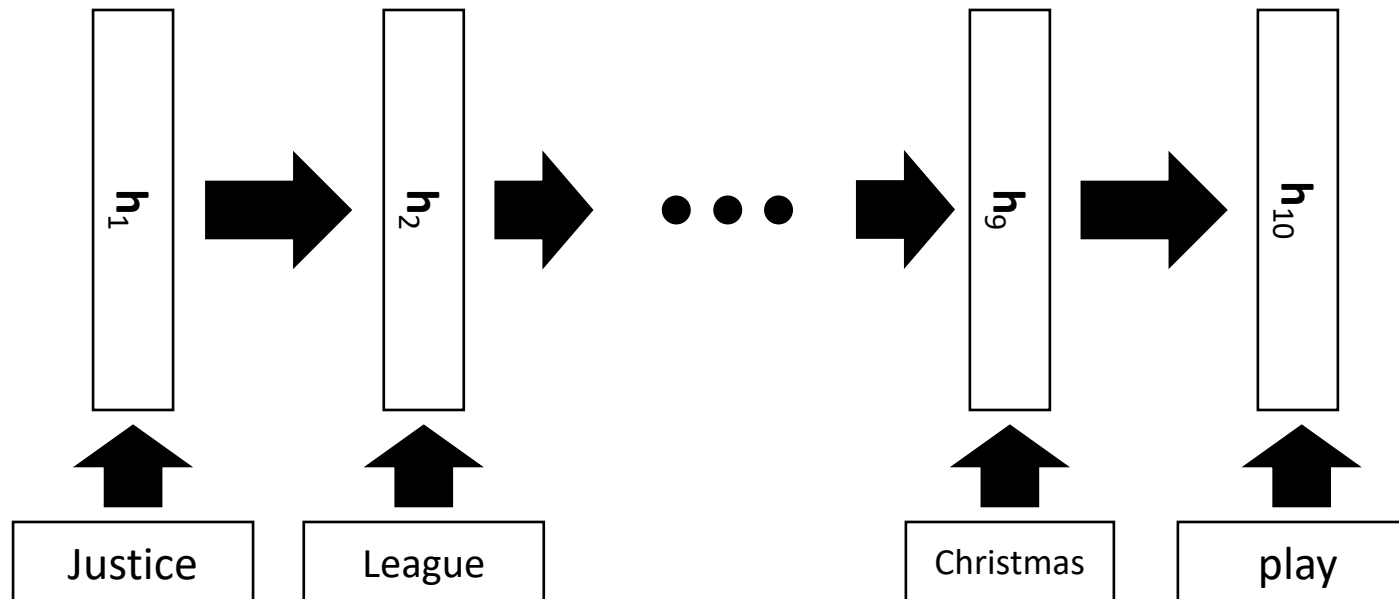
$$\begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$



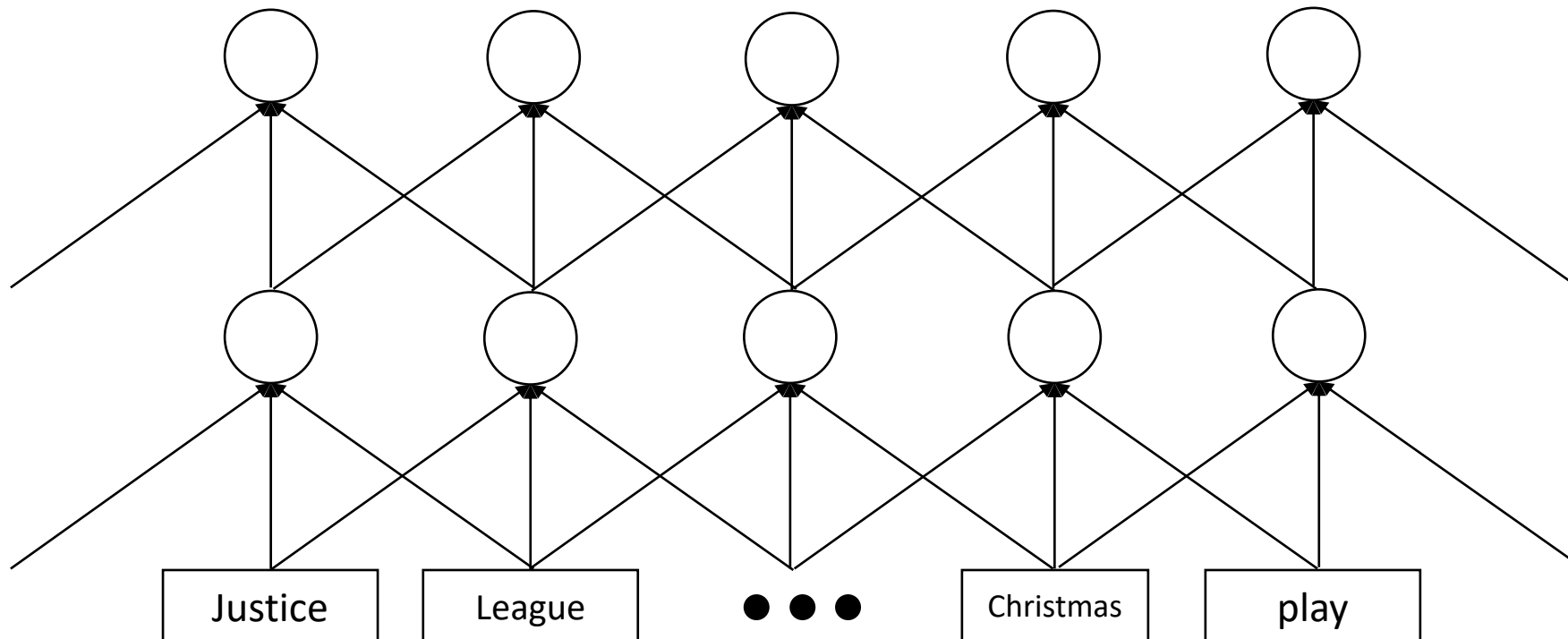
RNN, 1D Conv, Transformers

- RNN, given a sequence of embeddings



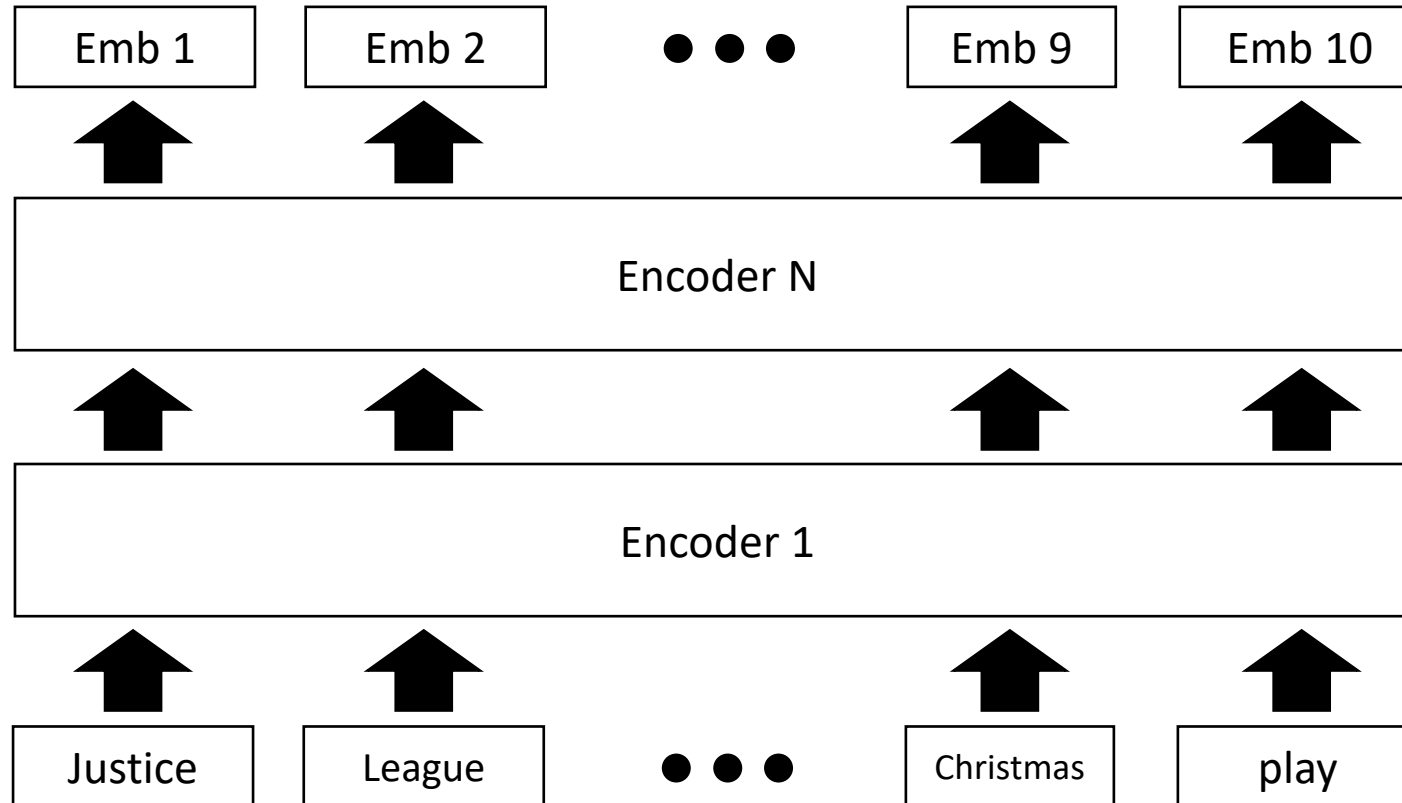
RNN, 1D Conv, Transformers

- 1-D CNN, given a sequence of embeddings



RNN, 1D Conv, Transformers

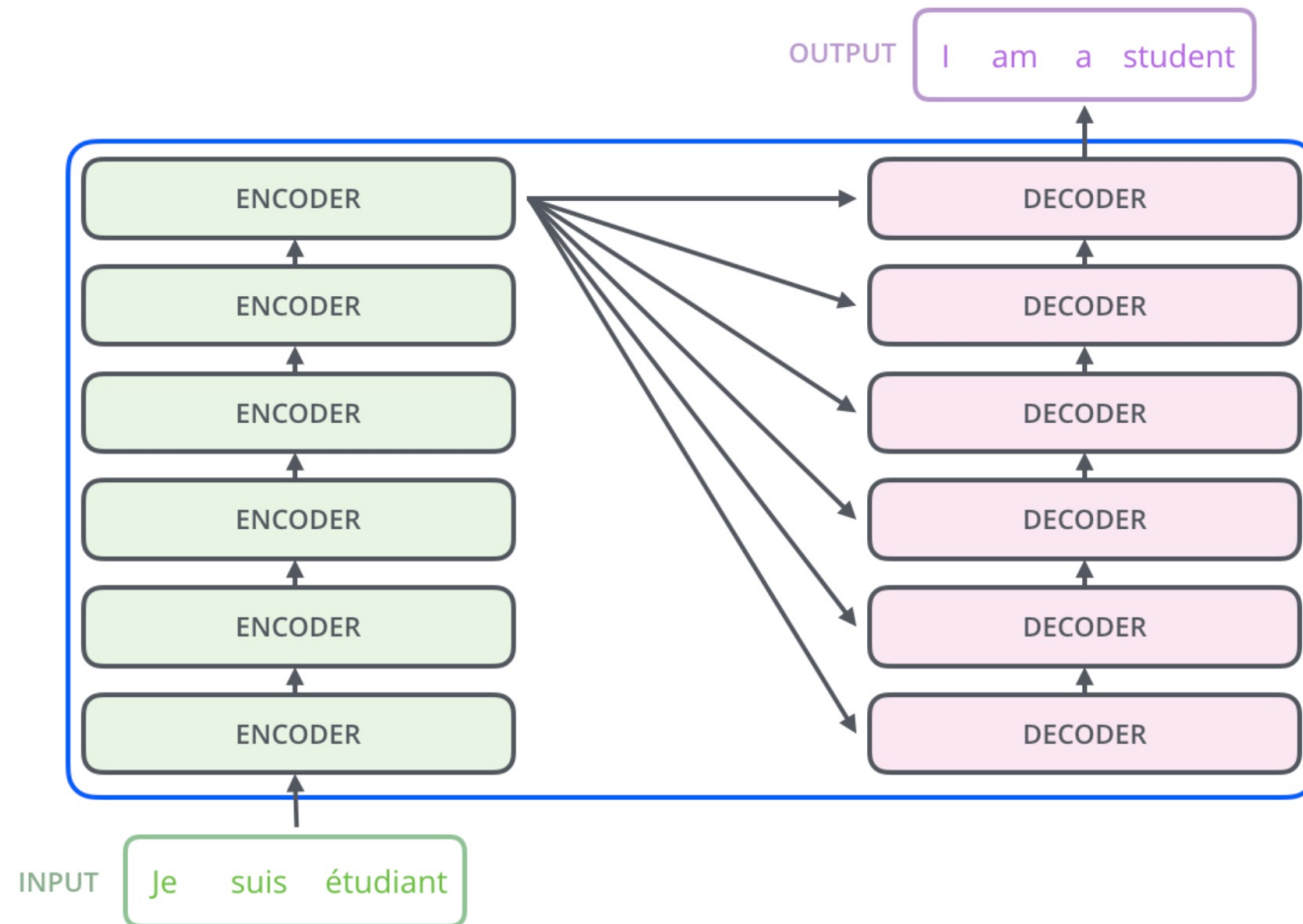
- Transformer, given a sequence of embeddings



Sequence-to-Sequence

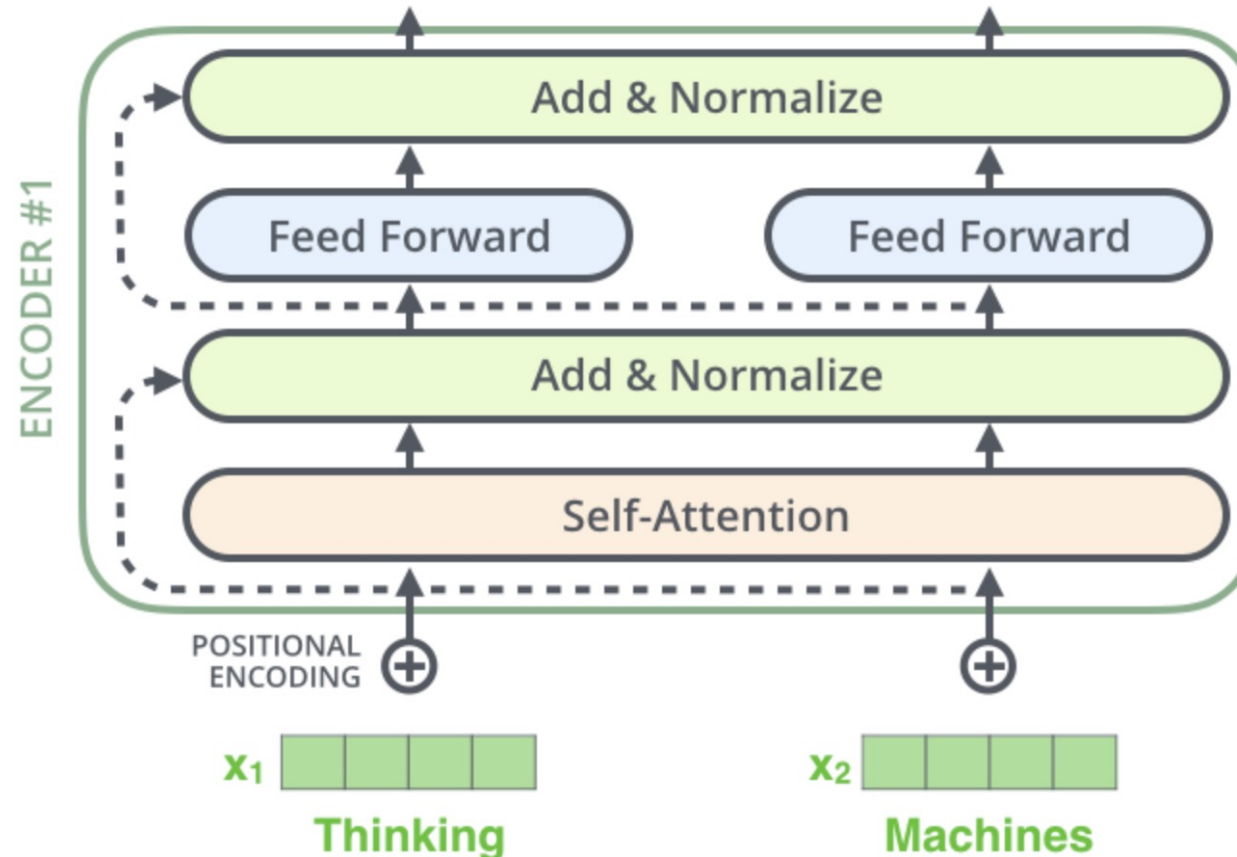
Transformer

- Encoders and decoders



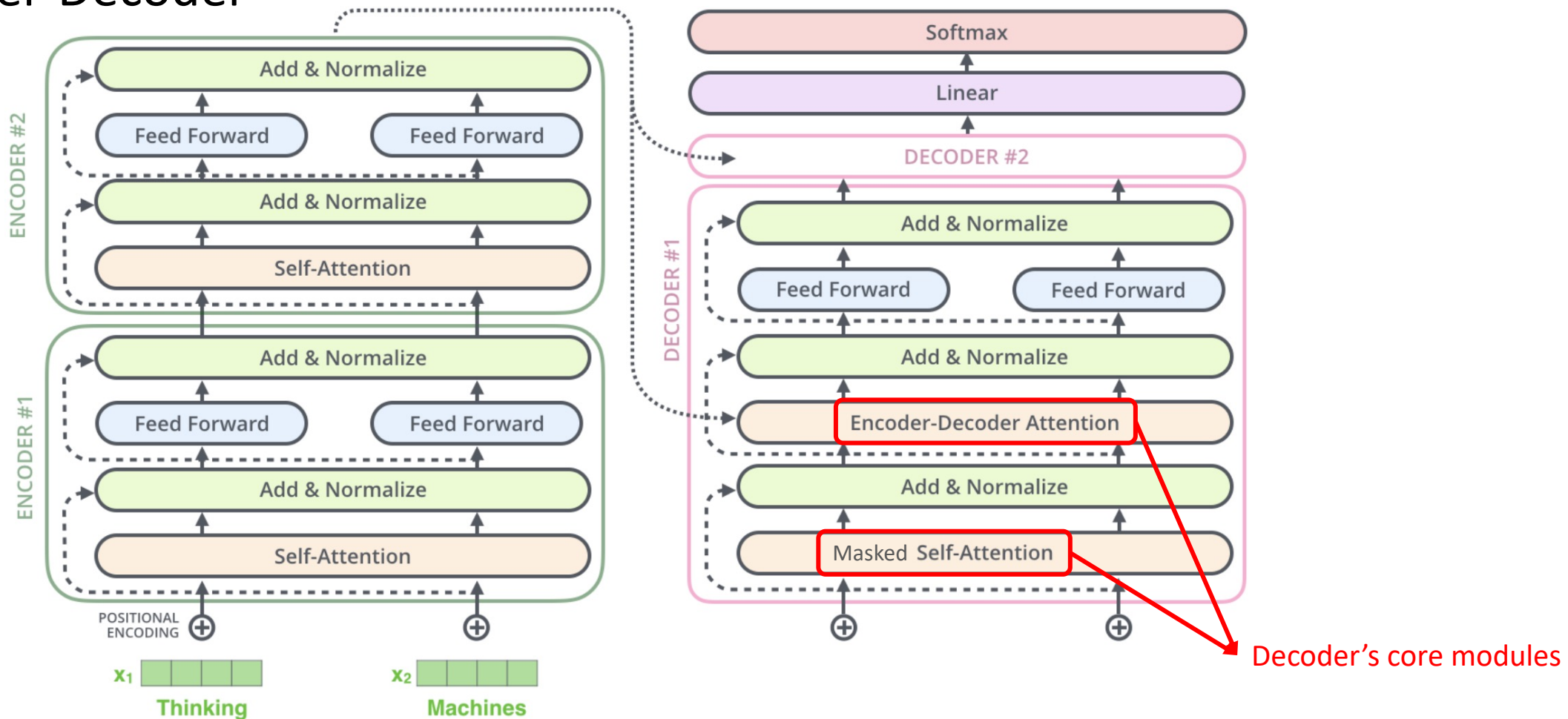
Transformer

- Single encoder



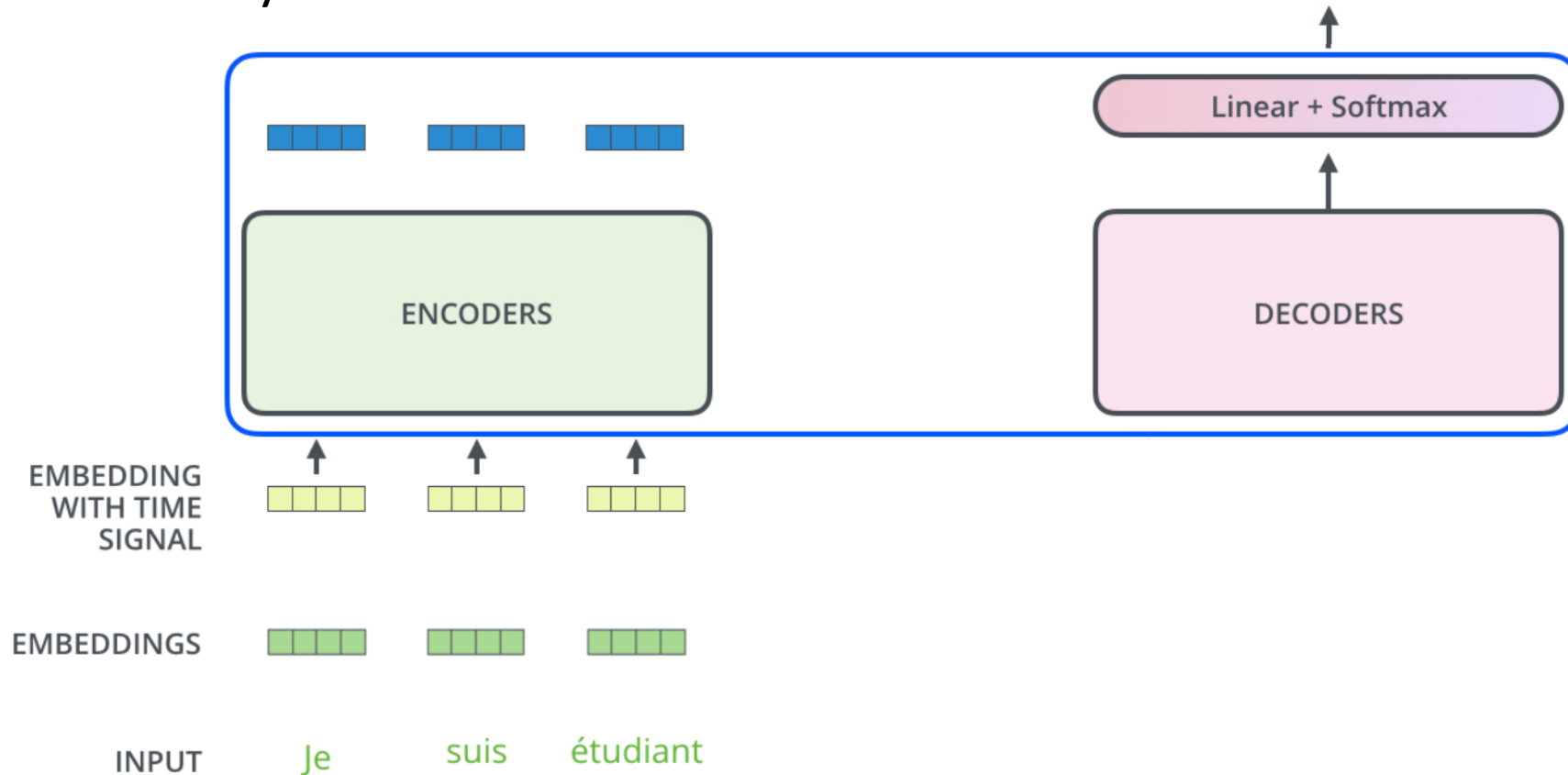
Transformer

- Encoder-Decoder

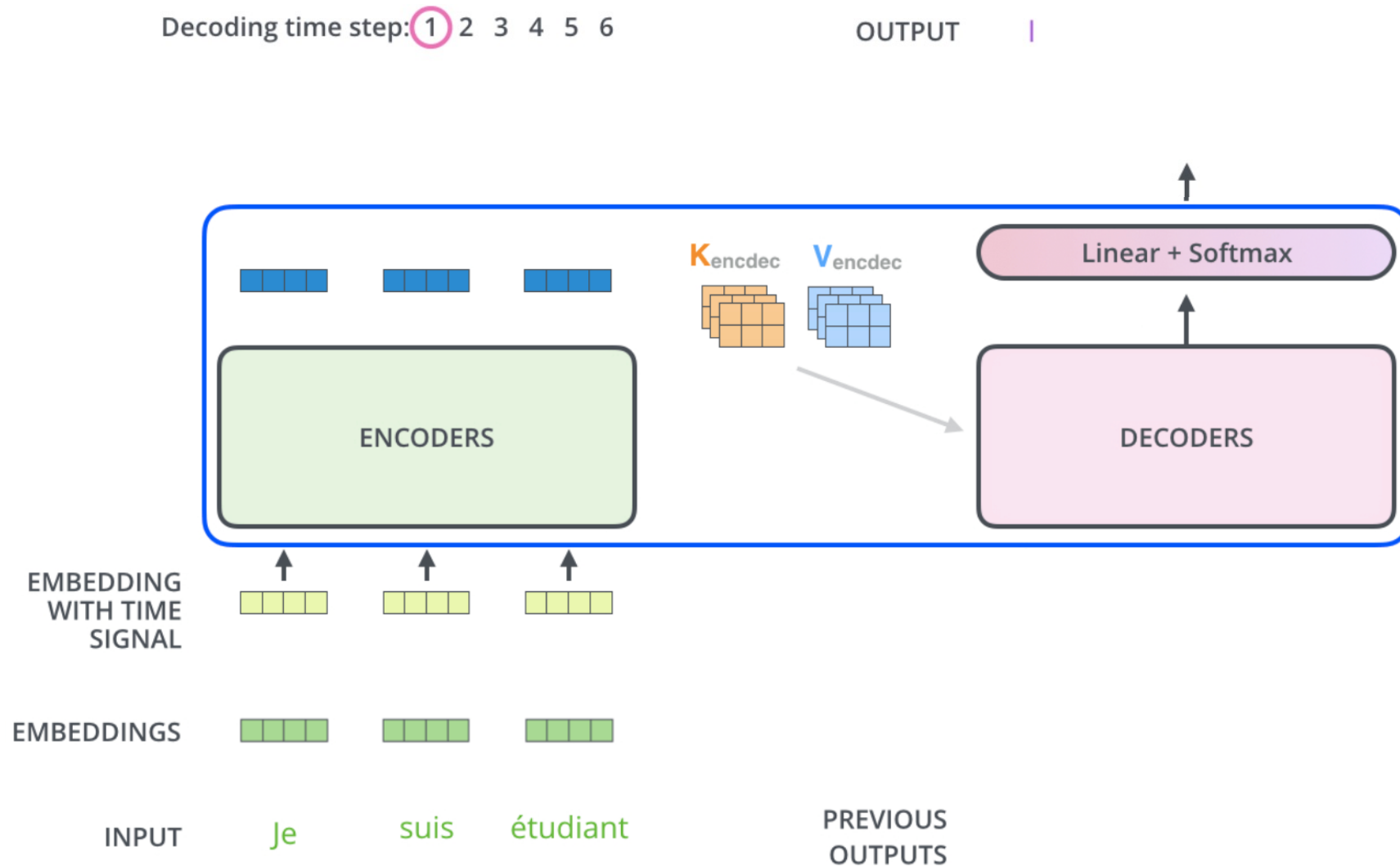


Decoding Process

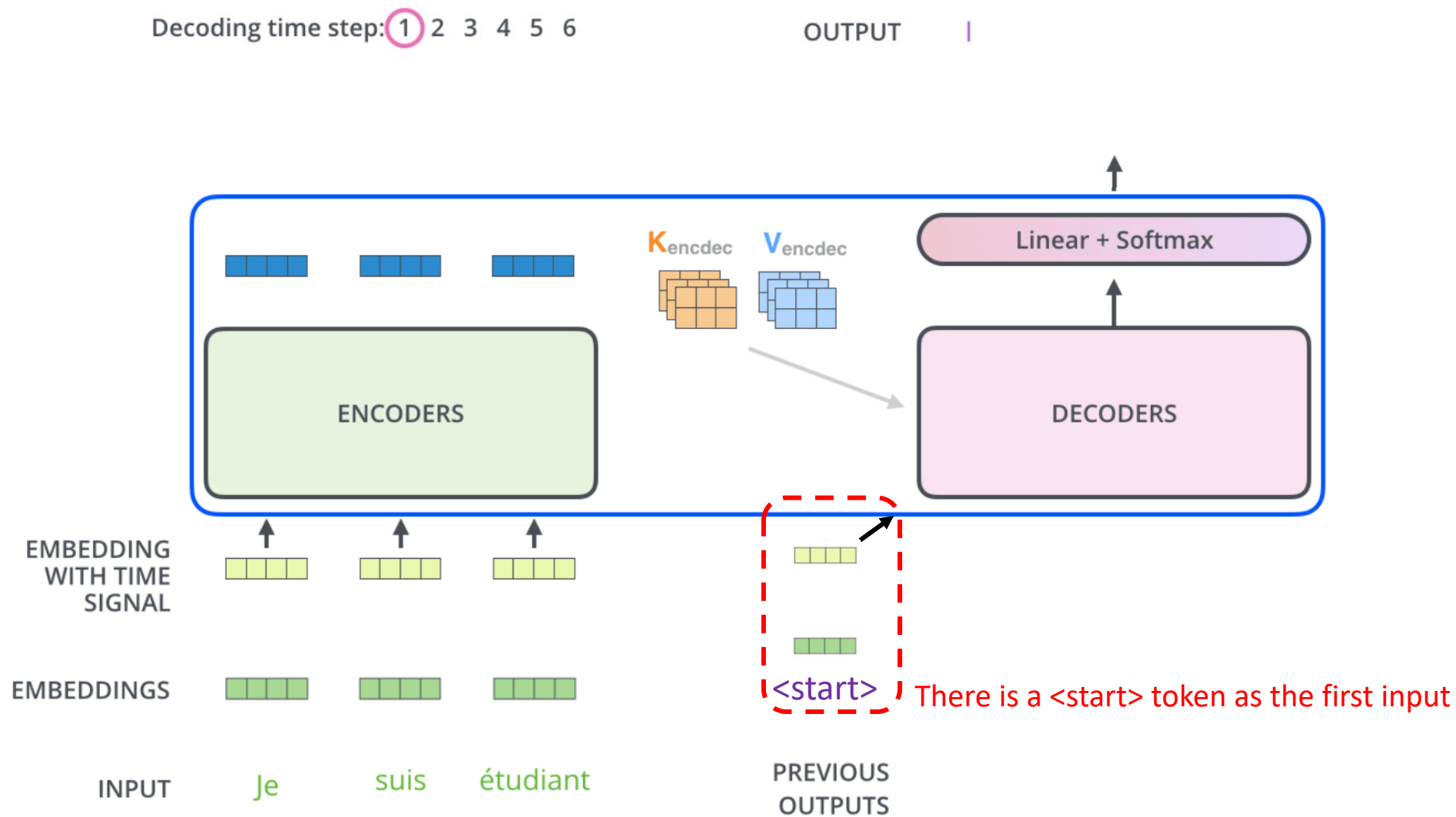
- Feed French sentence into Encoder
 - Decoder ready to decode



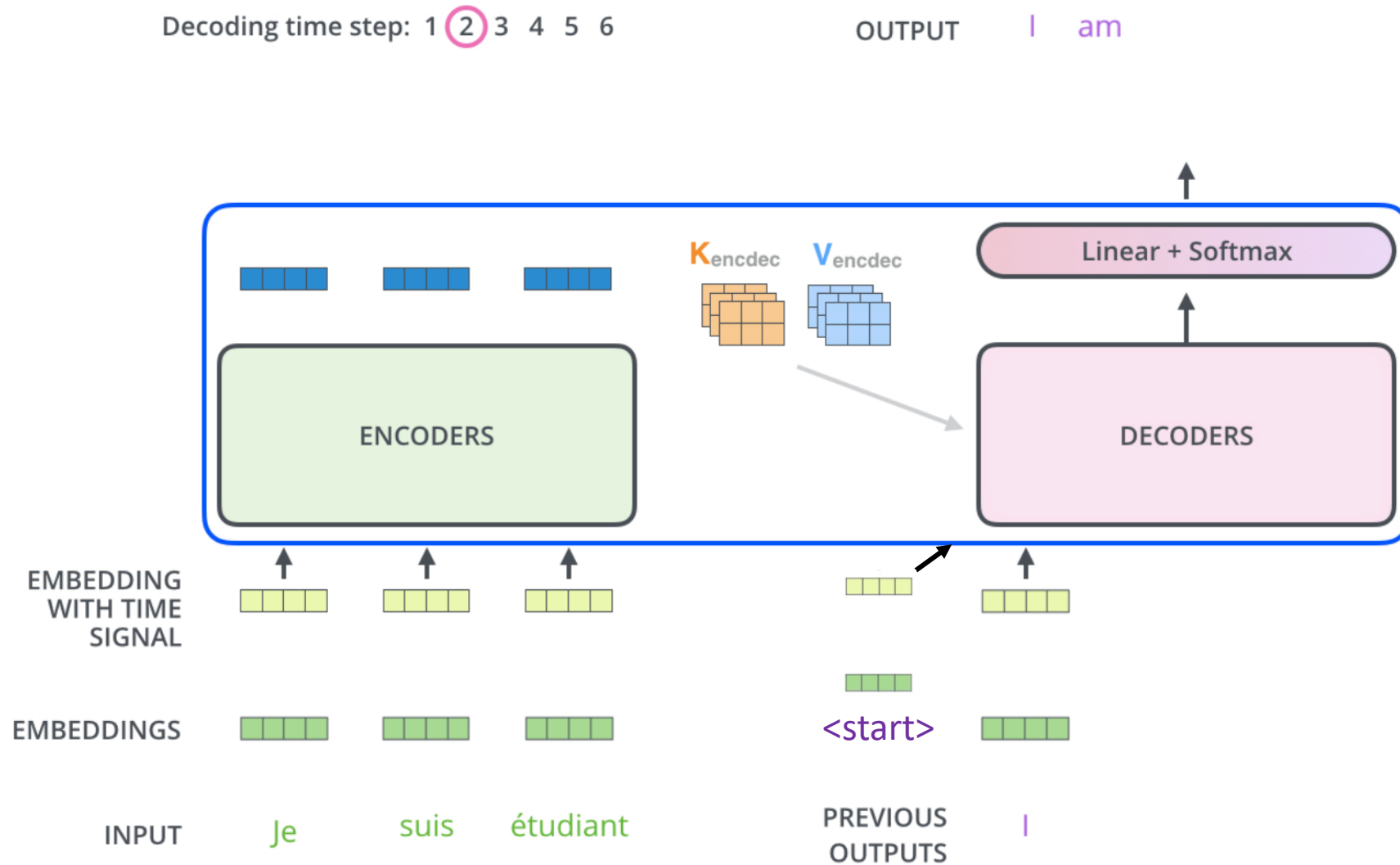
Decoding Process



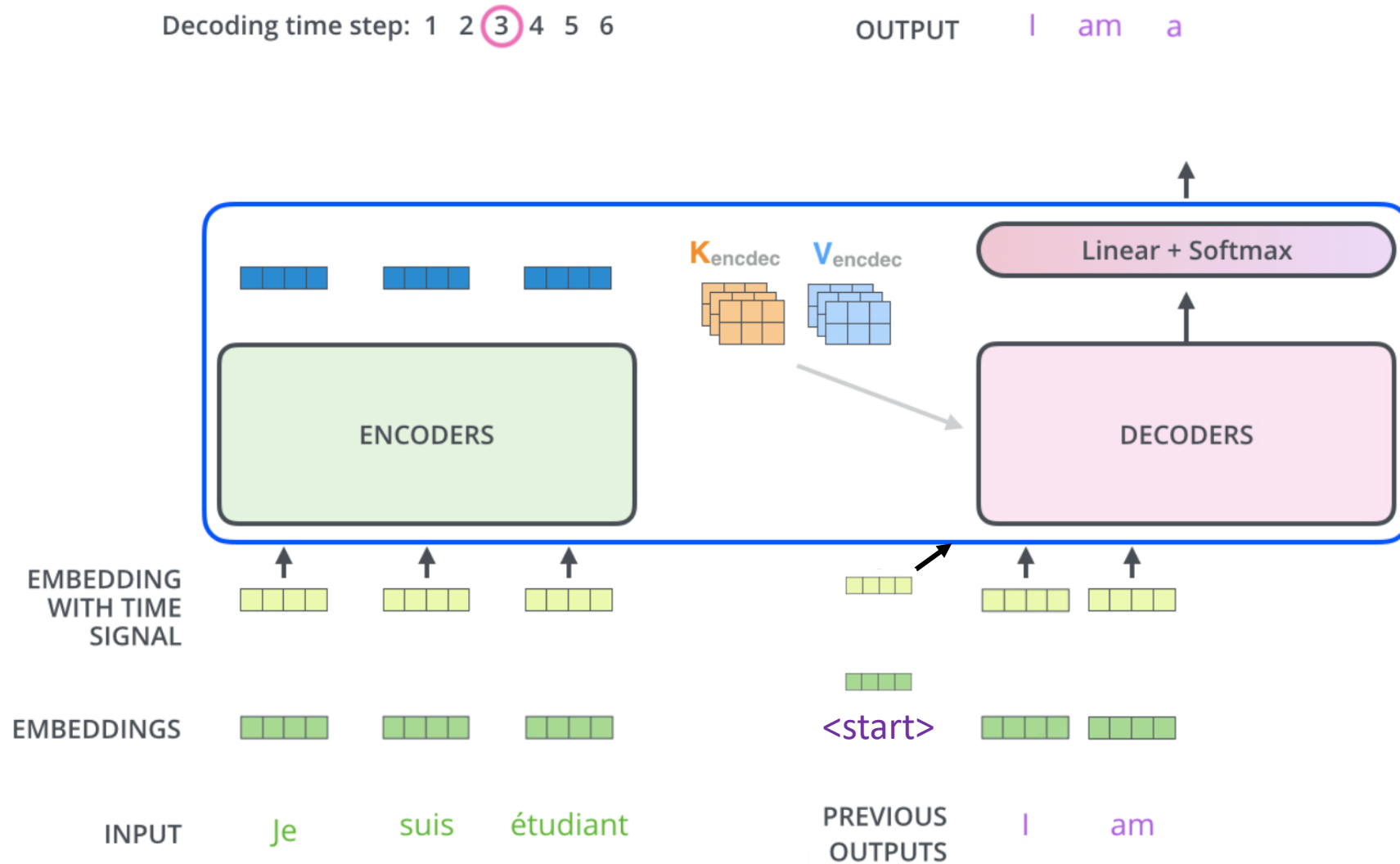
Decoding 예제



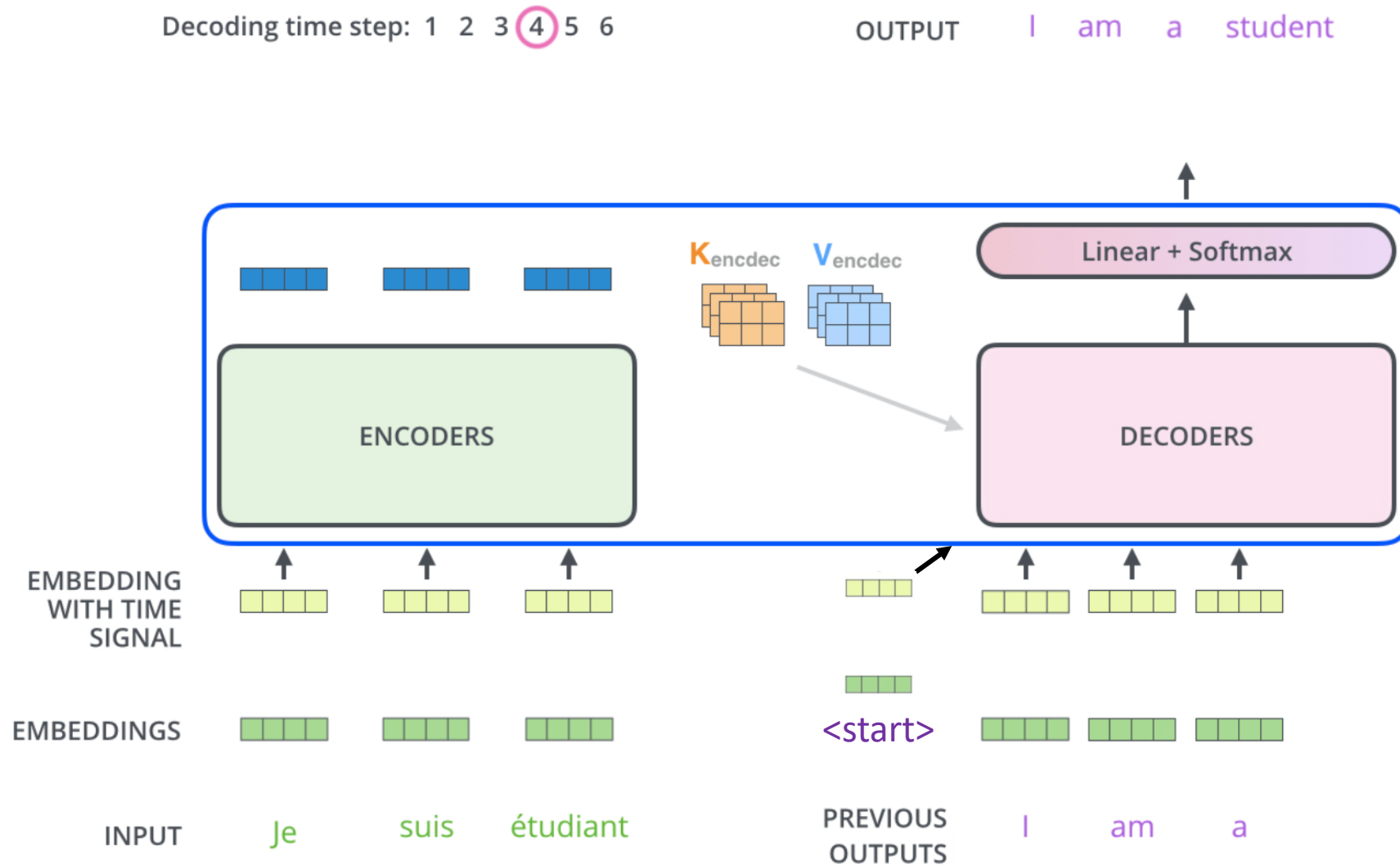
Decoding Process



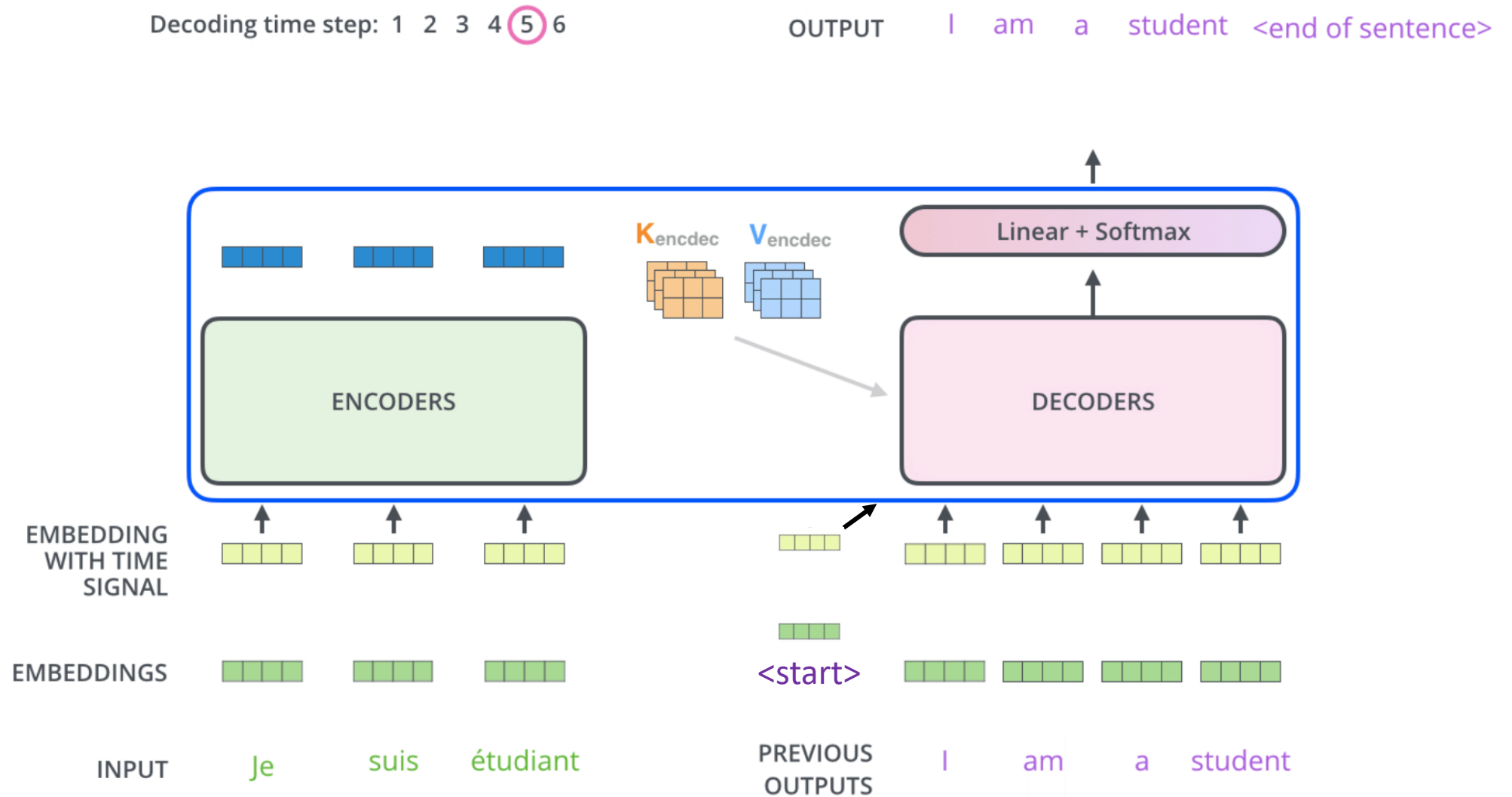
Decoding Process



Decoding Process



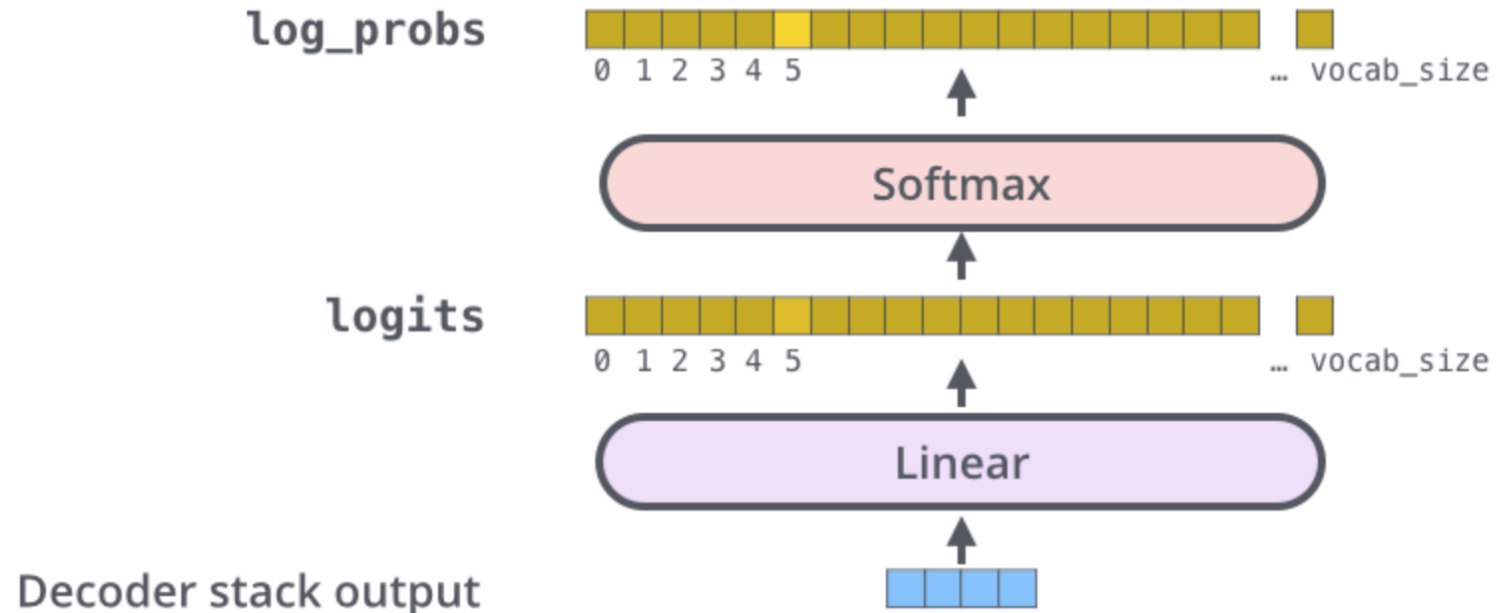
Decoding Process



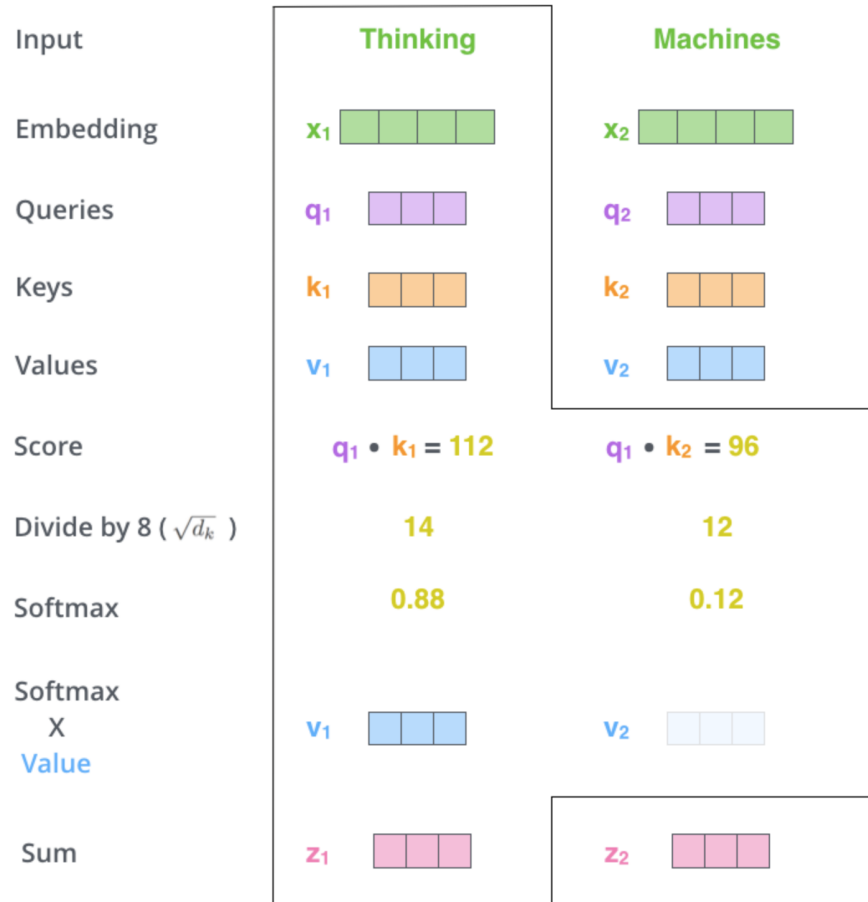
Prediction Layer

Which word in our vocabulary
is associated with this index?

Get the index of the cell
with the highest value
(**argmax**)

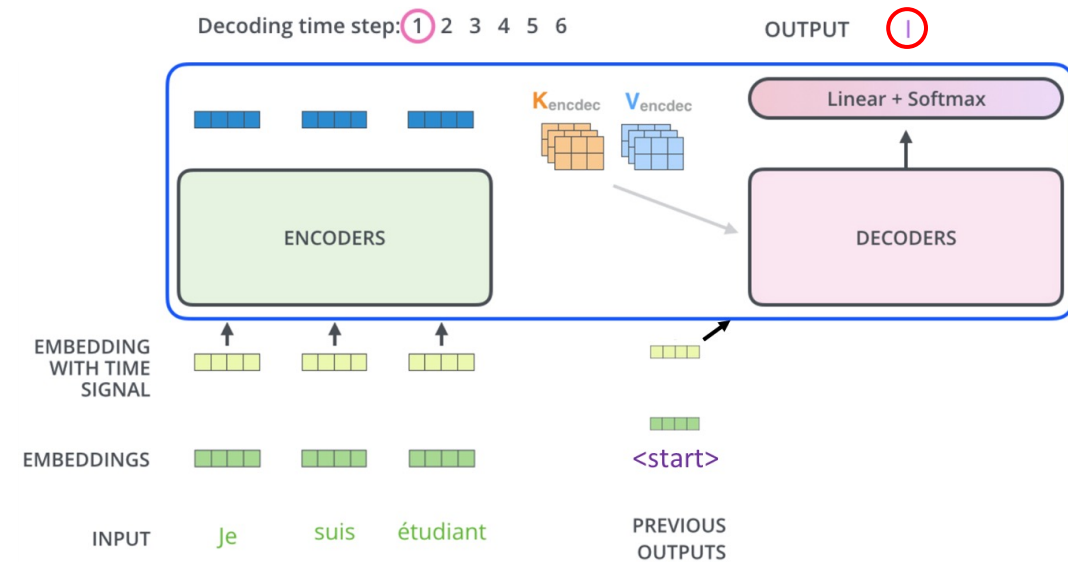


Encoder-Decoder Attention



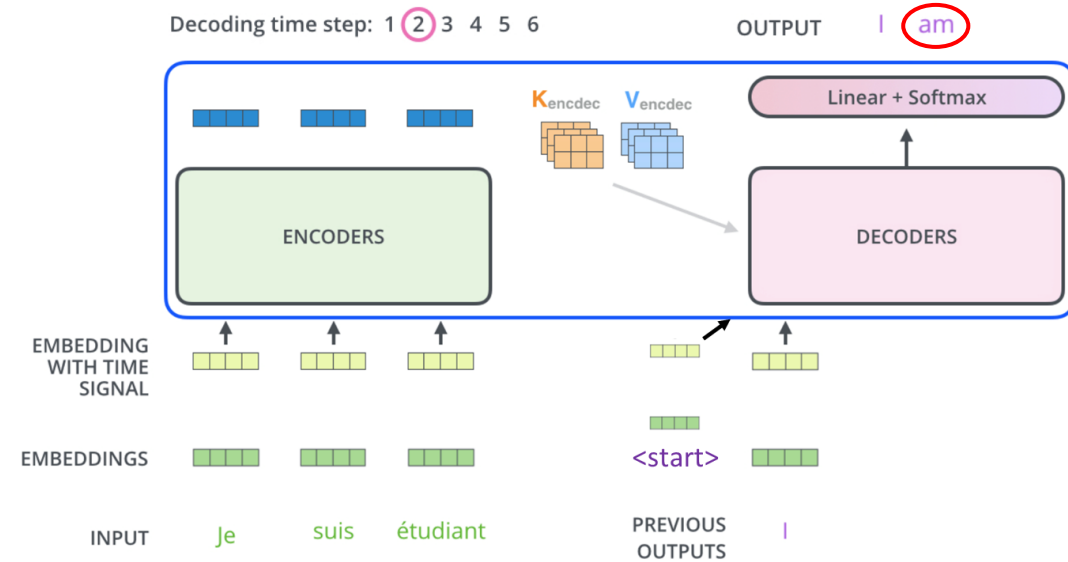
- Encoder's self-attention recap
- At "Thinking"'s turn
 - Generate queries with "Thinking"
 - Generate keys and values with all tokens
 - Calculate scores based on the current token (query) and all tokens (keys)
 - Sum values proportional to the scores

Encoder-Decoder Attention



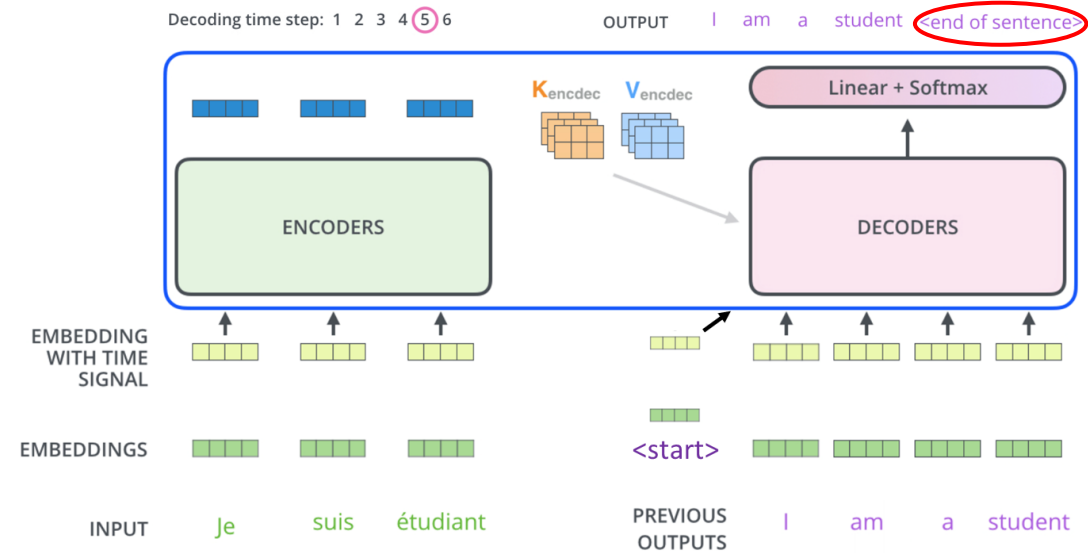
- Predicting the first token “I” in Decoder
- It is <start>’s turn, so generate query using <start>
- Generate keys and values with all three outputs from the Encoder
- Calculate scores based on query and keys
- Sum values proportional to the scores
- Use the summed value to predict “I”

Encoder-Decoder Attention



- Predicting the second token “am” in Decoder
- It is “I”’s turn, so generate query using “I”
- Generate keys and values with all three outputs from the Encoder
- Calculate scores based on query and keys
- Sum values proportional to the scores
- Use the summed value to predict “am”

Encoder-Decoder Attention



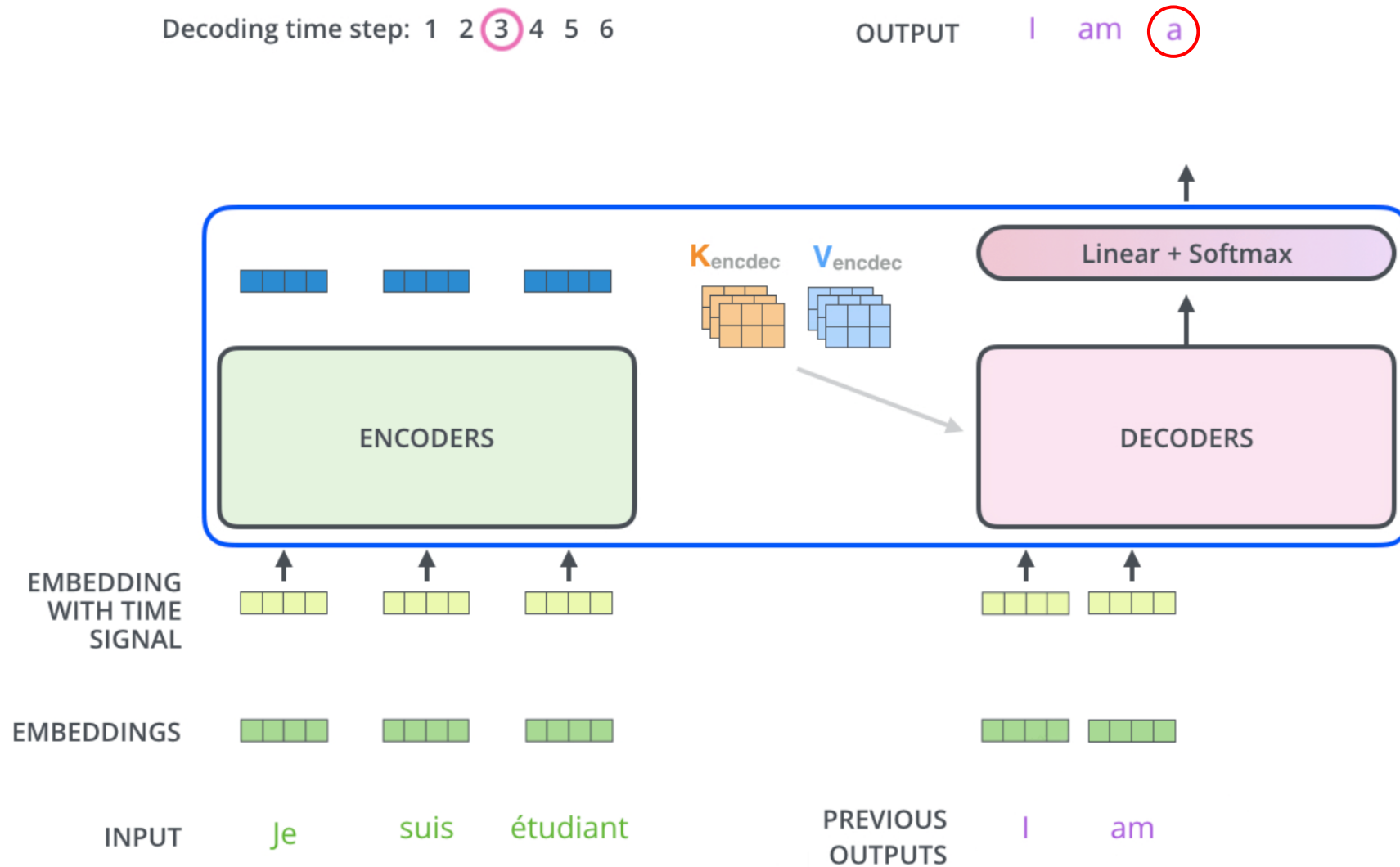
- Predicting the final token <EoS> in Decoder
- It is “student”’s turn, so generate query using “student”
- Generate keys and values with all three outputs from the Encoder
- Calculate scores based on query and keys
- Sum values proportional to the scores
- Use the summed value to predict <EoS>

Decoding during Training & Testing

- Encoder-Decoder attention handles one query at a time?
 - Not really!
 - If we knew the input to Decoder (i.e. ground-truth English sentence), we can process all queries at once
- During training, we know GT English sentence
 - Can generate all Decoder output tokens at once
- During testing, we **don't** know GT English sentence
 - We generate one token at a time
 - Because we don't know the total length of the output sentence
 - Use the previous output token as the next input
 - Autoregressive generation

Masked Self-Attention

Decoding: Predicting “a”



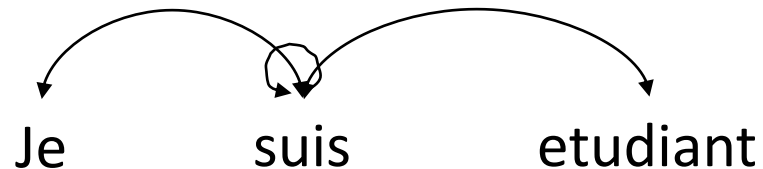
Decoding: Predicting “a”

- Currently known tokens:
 - Je, suis, etudiant
 - I, am
- Token to predict:
 - a

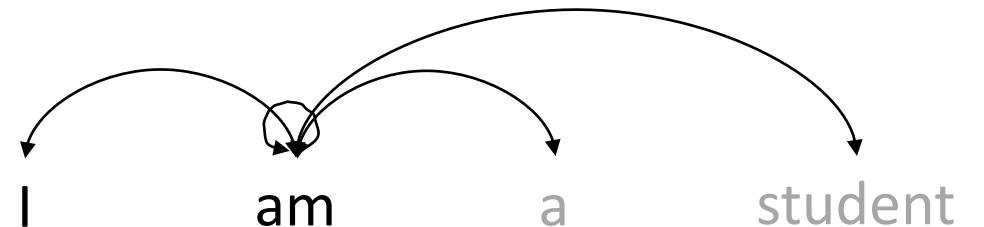
Decoding: Predicting “a”

- Currently known tokens:
 - Je, suis, etudiant
 - I, am
- Token to predict:
 - a

Encoder attention from “suis”



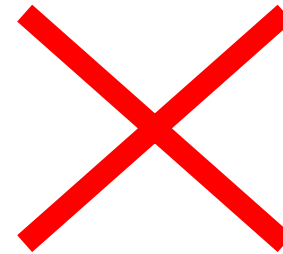
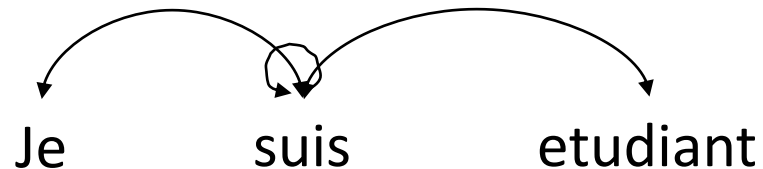
Decoder attention from “am”



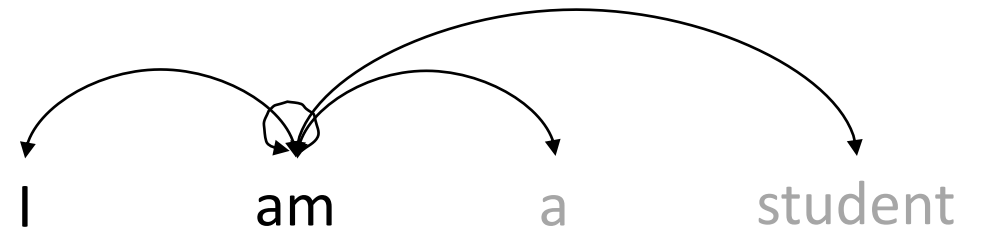
Decoding: Predicting “a”

- Currently known tokens:
 - Je, suis, etudiant
 - I, am
- Token to predict:
 - a

Encoder attention from “suis”

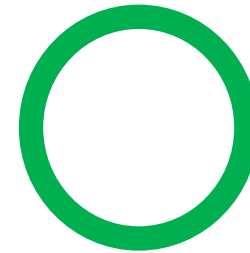


Decoder attention from “am”

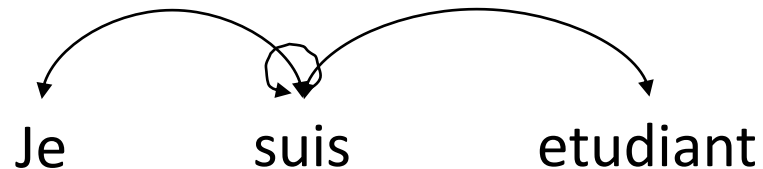


Decoding: Predicting “a”

- Currently known tokens:
 - Je, suis, etudiant
 - I, am
- Token to predict:
 - a



Encoder attention from “suis”



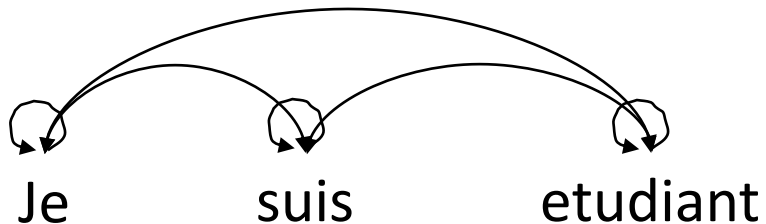
Decoder attention from “am”



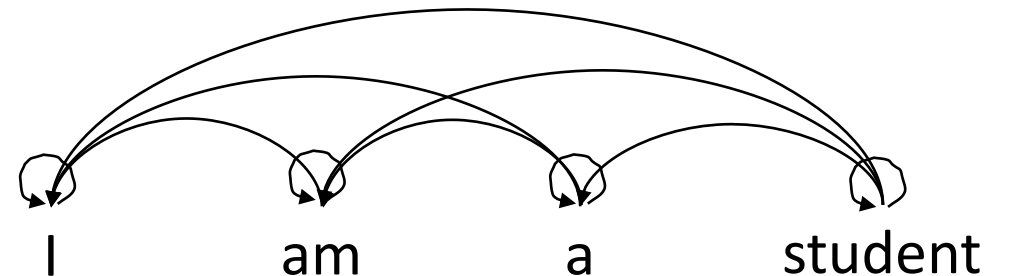
Decoder Self-Attention

- Encoder
 - Full (bi-directional) self-attention
- Decoder
 - Limited (masked, or uni-directional) self-attention
 - Can only attend to itself and backwards

Encoder self-attention

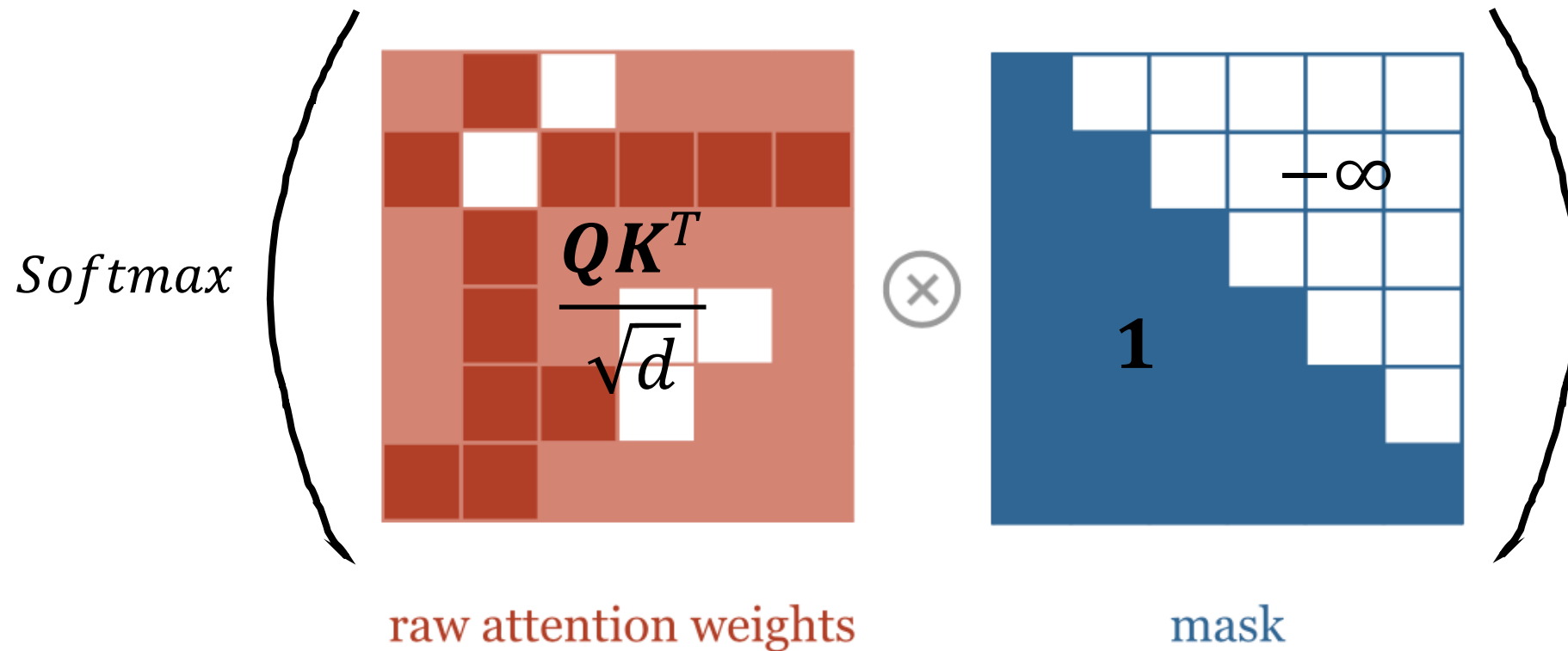


Decoder self-attention



Masked Self-Attention

- Multiply a triangular matrix to the unnormalized attention.



Masked Self-Attention

- Masked QKV Operation

1.0	0	0	0	0	0
0.3	0.7	0	0	0	0
0.2	0.3	0.5	0	0	0
0.1	0.2	0.1	0.6	0	0
..	..	..	..	..	0
..	..	..	..	..	..

I
like
going
to
movies
.



$1.0 * I$
$0.3 * I + 0.7 * \text{like}$
$0.2 * I + 0.3 * \text{like} + 0.5 * \text{going}$
$0.1 * I + 0.2 * \text{like} + 0.1 * \text{going} + 0.6 * \text{to}$
..
..

AI504: Programming for Artificial Intelligence

Week 11: Transformer

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr