# AI504: Programming for Artificial Intelligence

## Week 12: Transformer

Edward Choi
Grad School of AI
edwardchoi@kaist.ac.kr

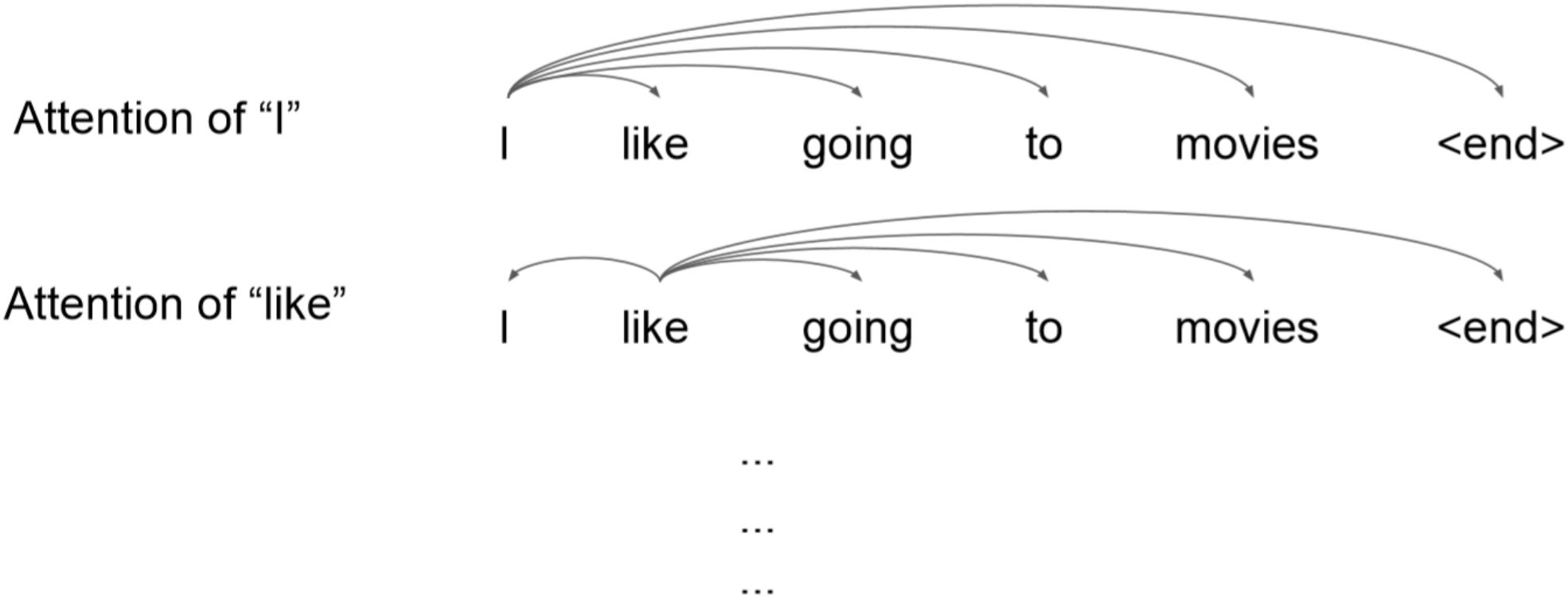
# Index

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

# What is Self-Attention?

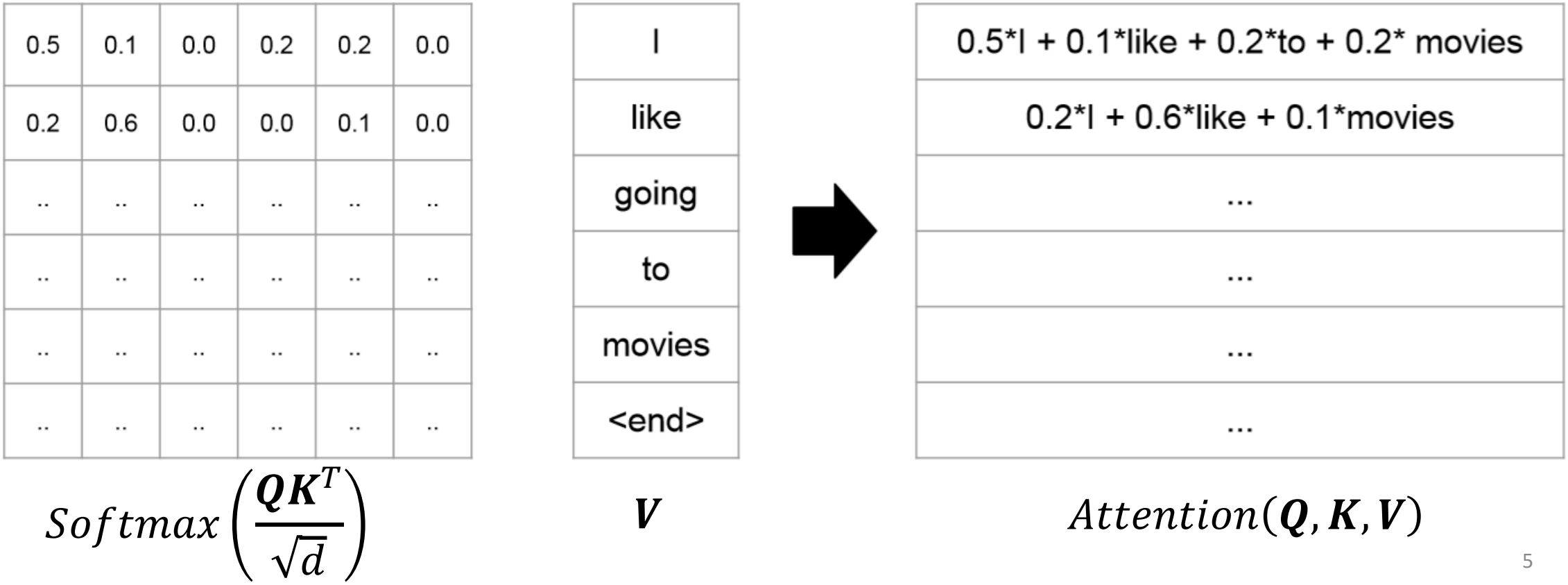
# Attention is All You Need

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



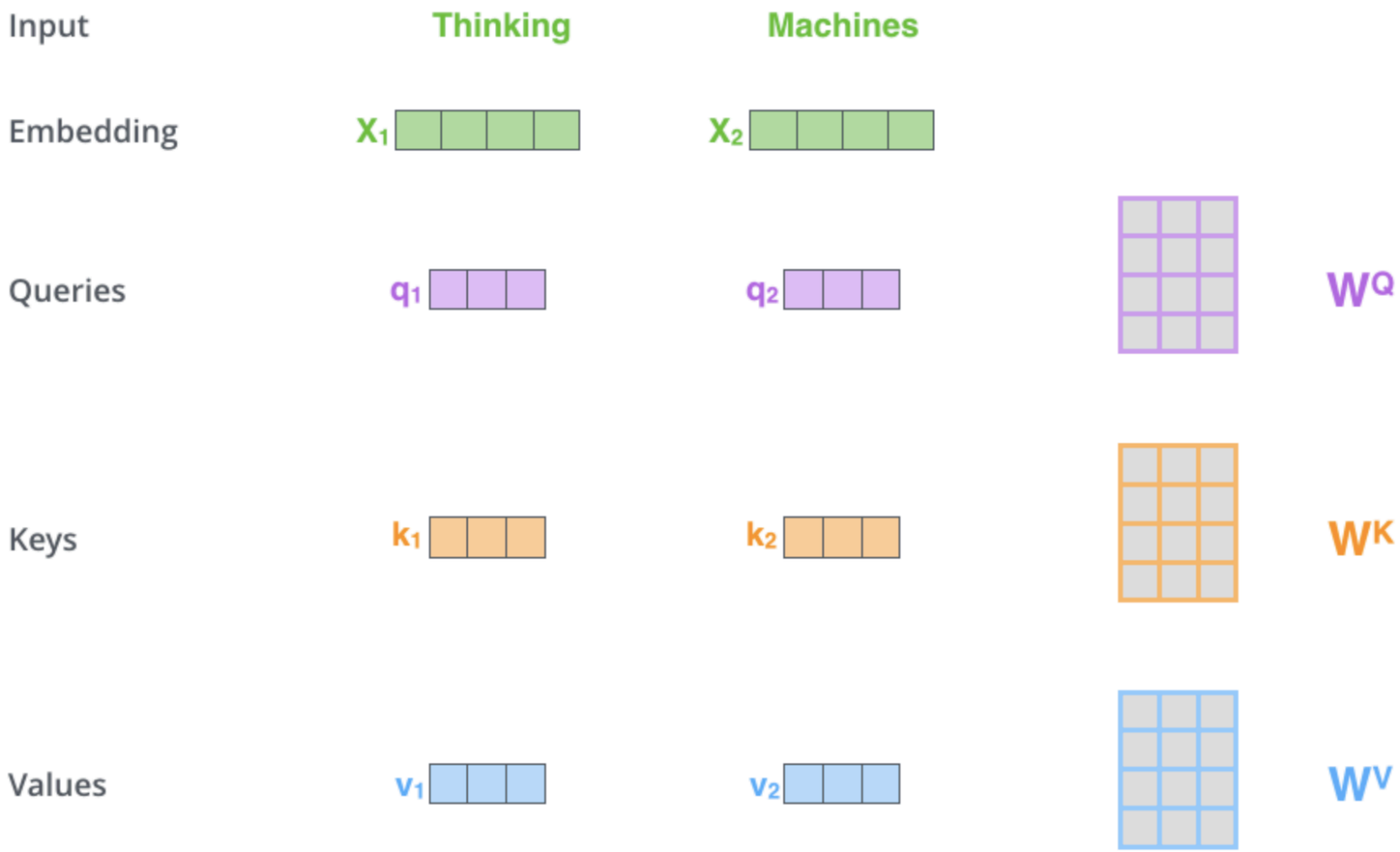
# Self-Attention

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



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

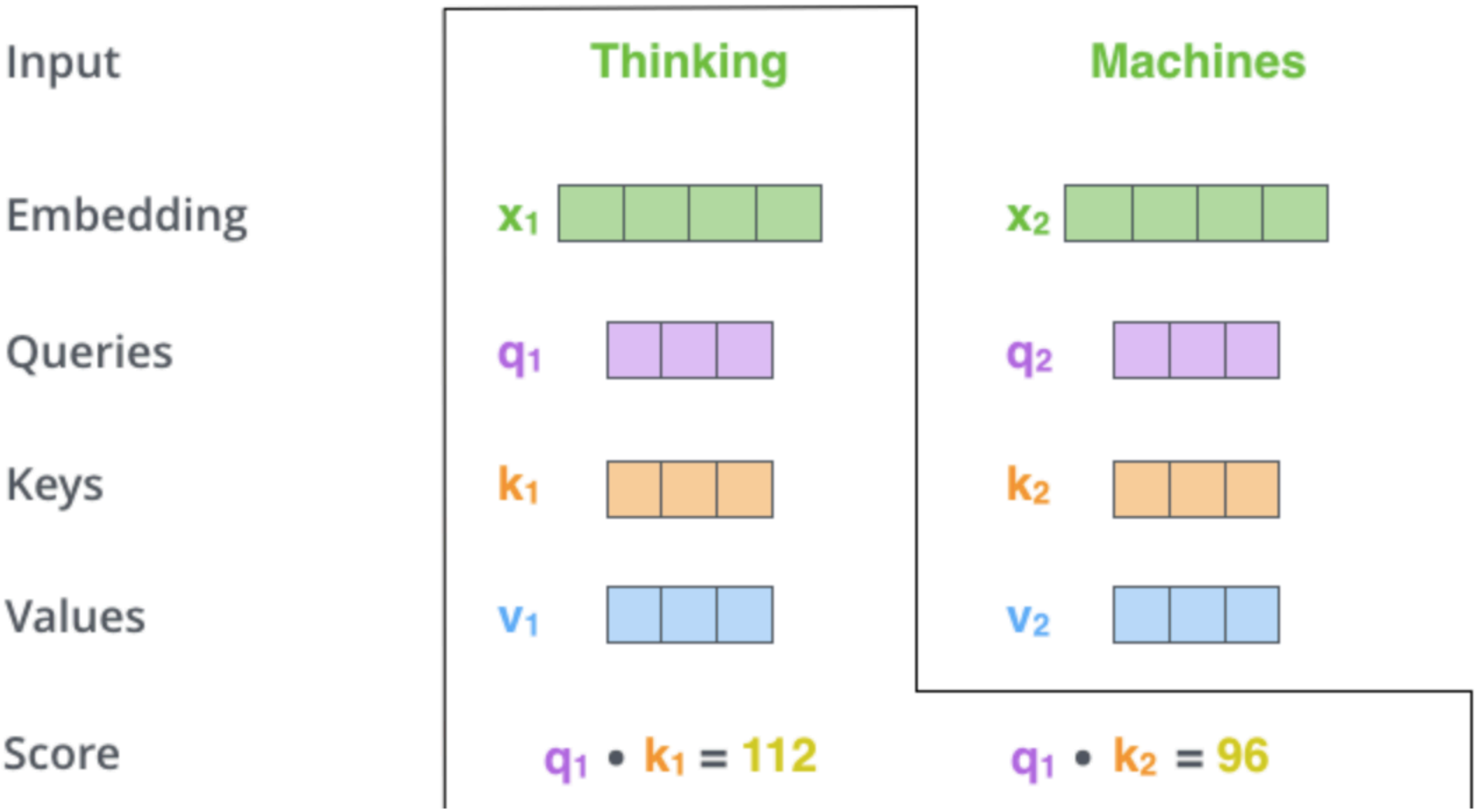
## Generating Queries, Keys, Values




http://jalammar.github.io/illustrated-transformer/

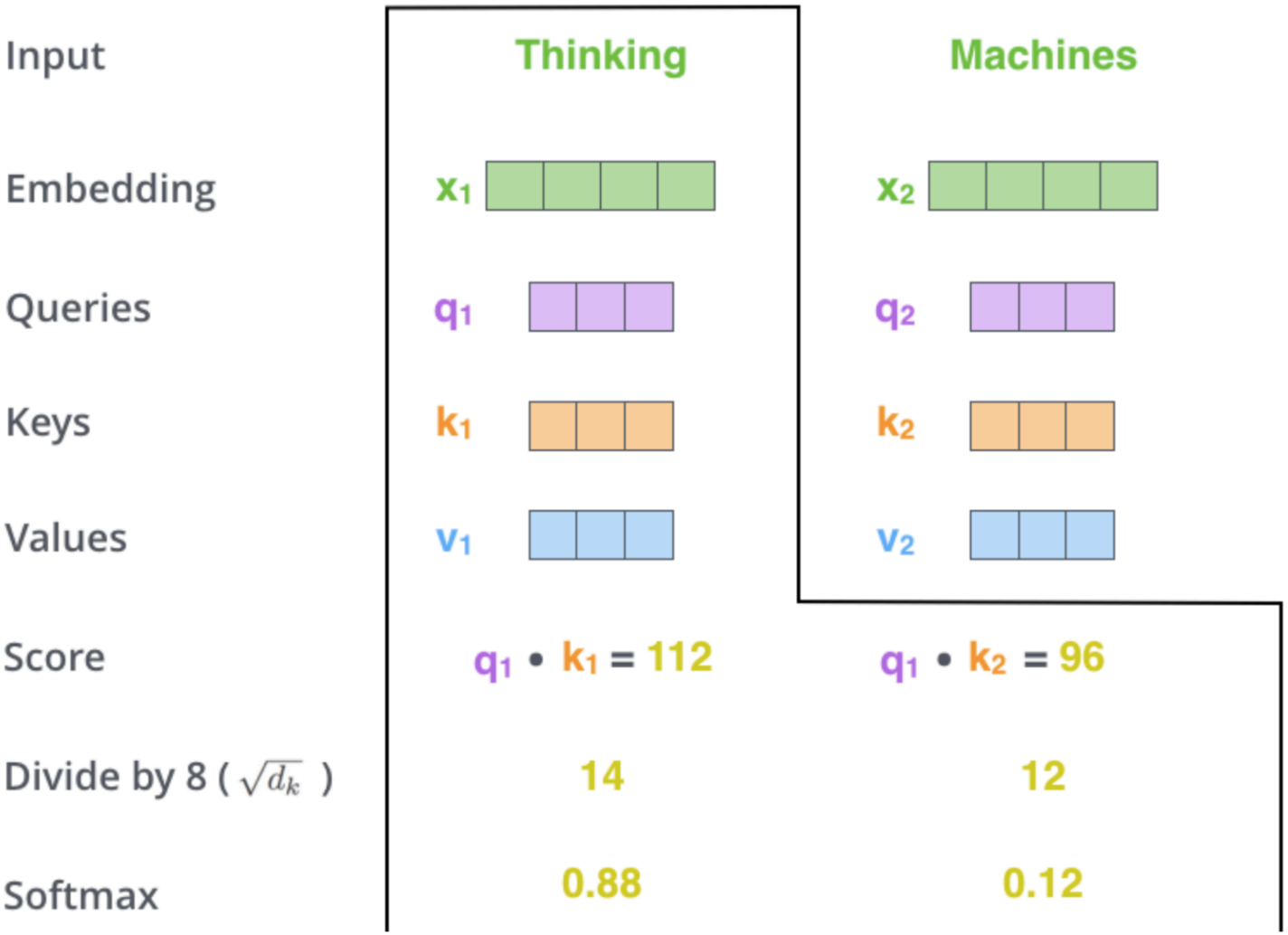
# QKV Operation: $Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

## Attention from “Thinking”



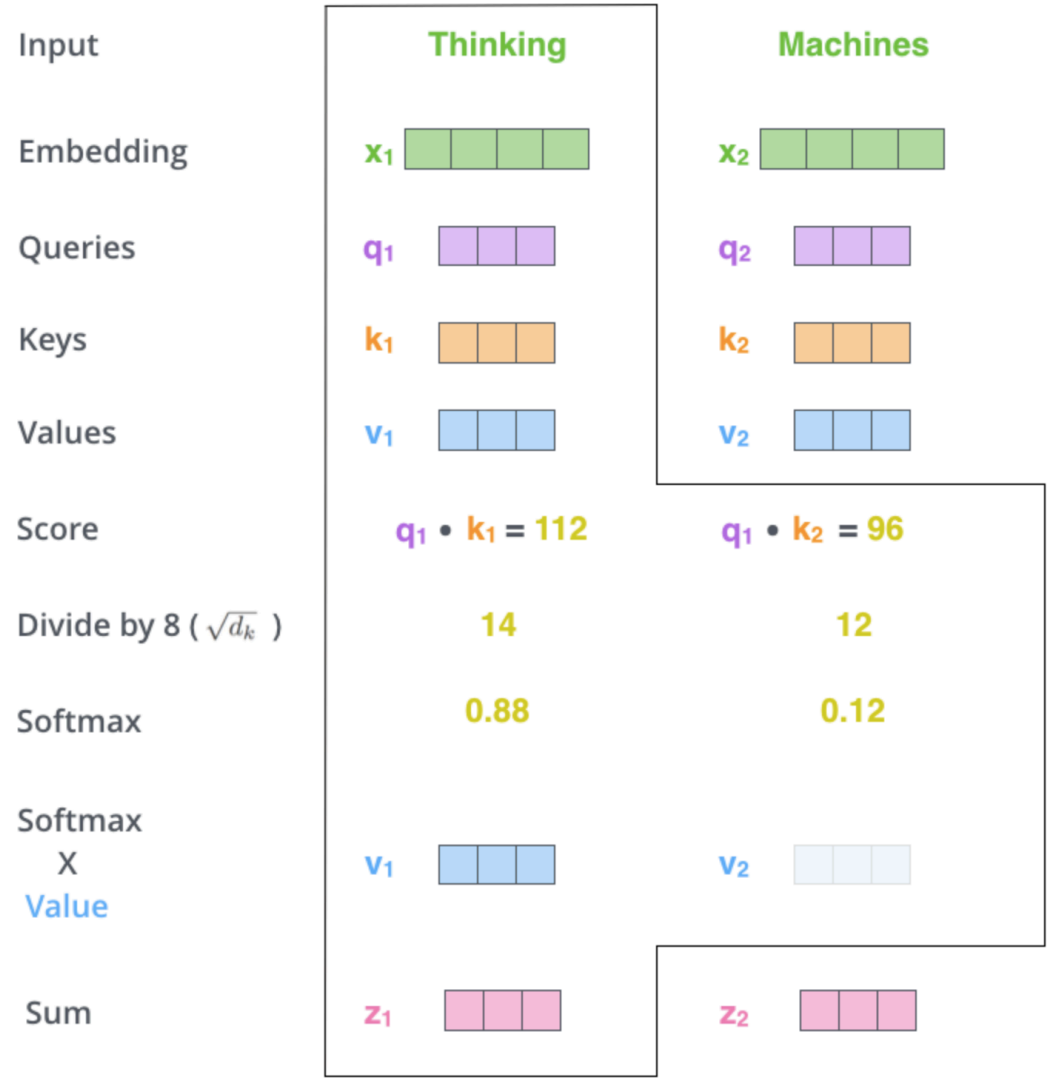
# QKV Operation: $Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

## Attention from “Thinking”



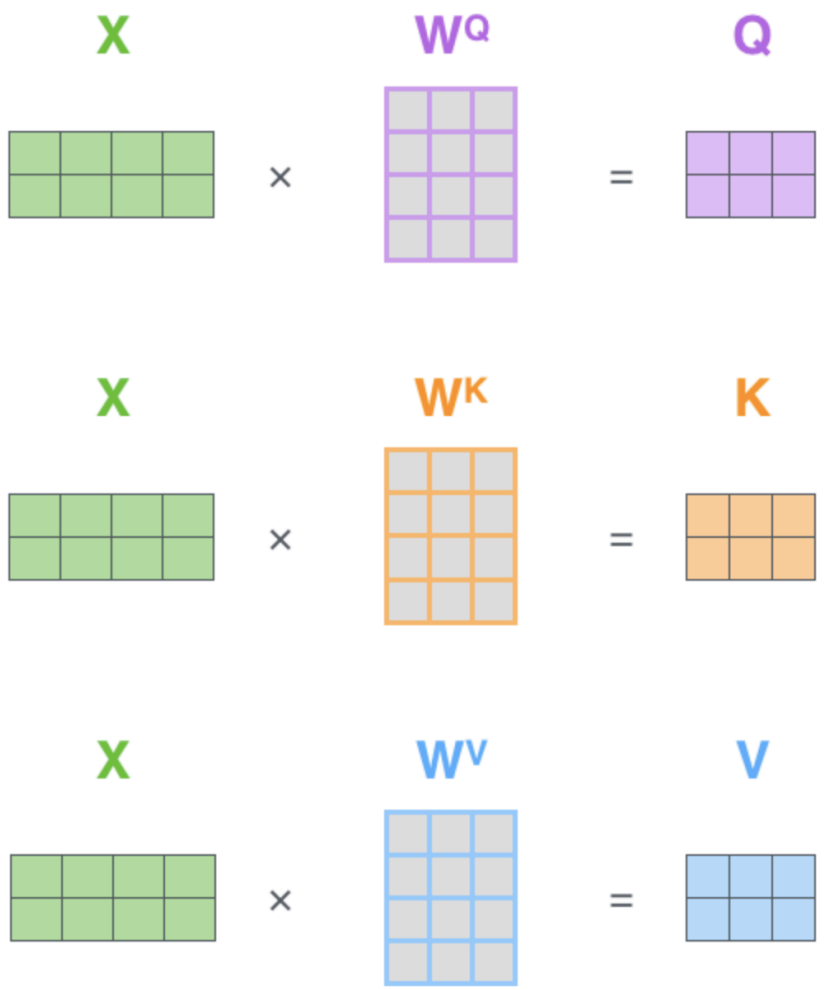
# QKV Operation: $Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

## Attention from “Thinking”



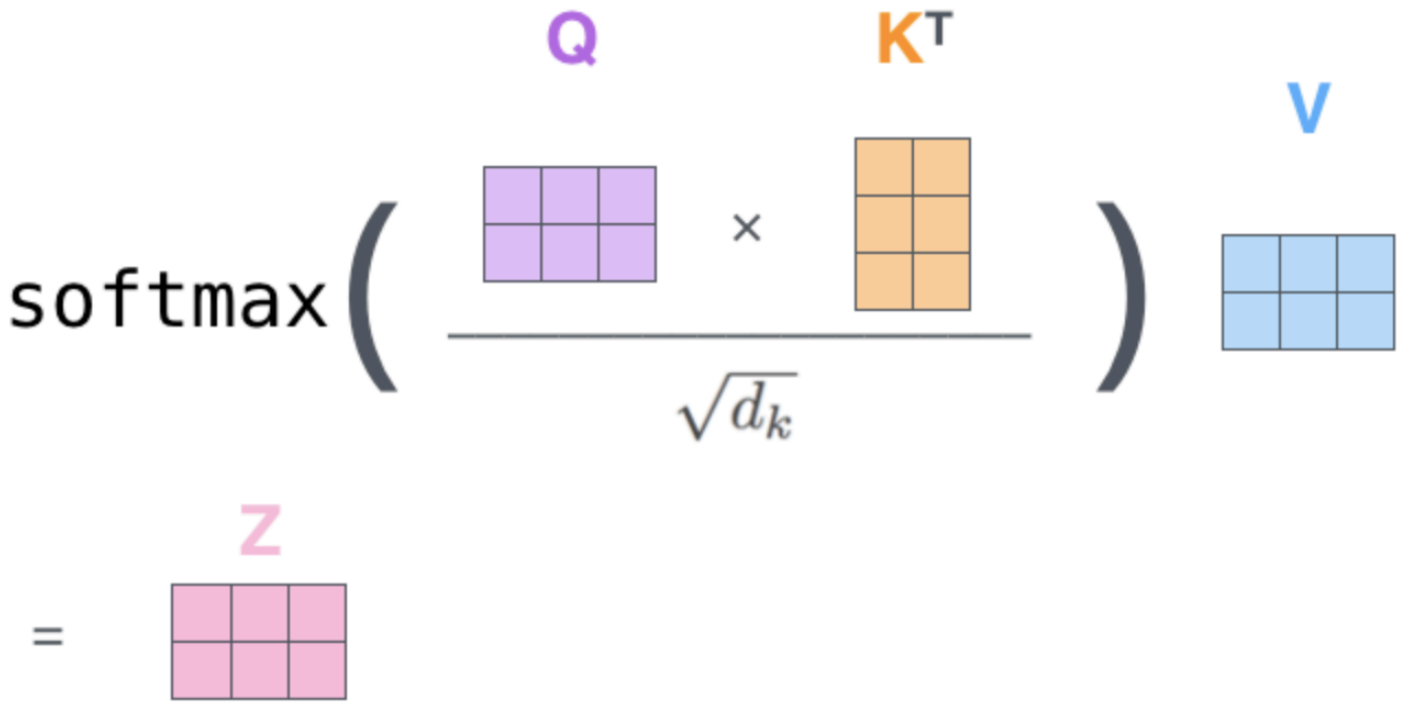
# QKV Operation: $Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

Processing all words at the same time.



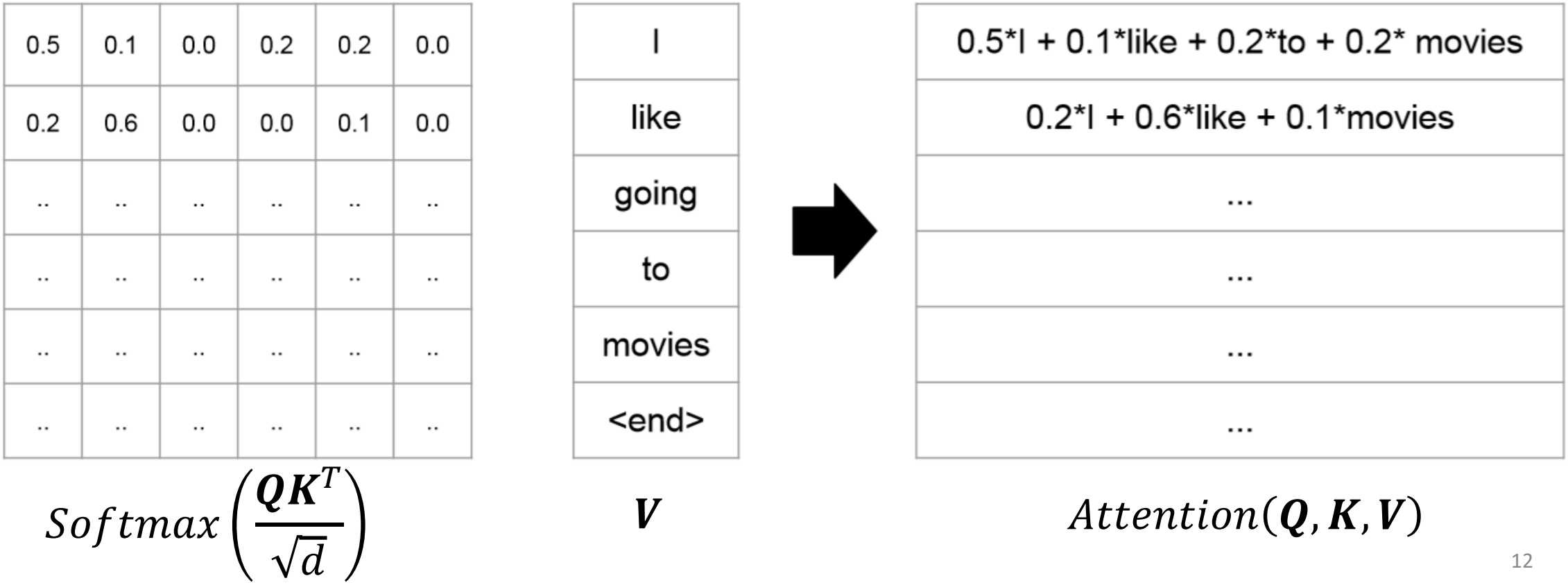
# QKV Operation: $Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

Processing all words at the same time.



# Self-Attention

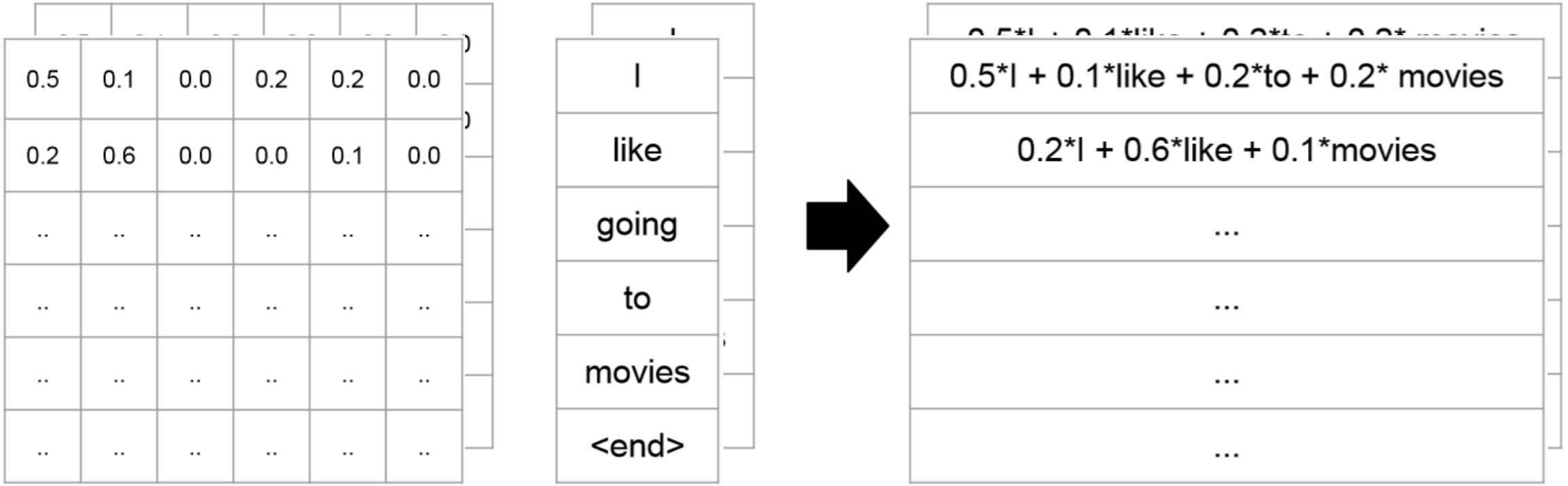
- $Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}$



# What is Multi-head Attention?

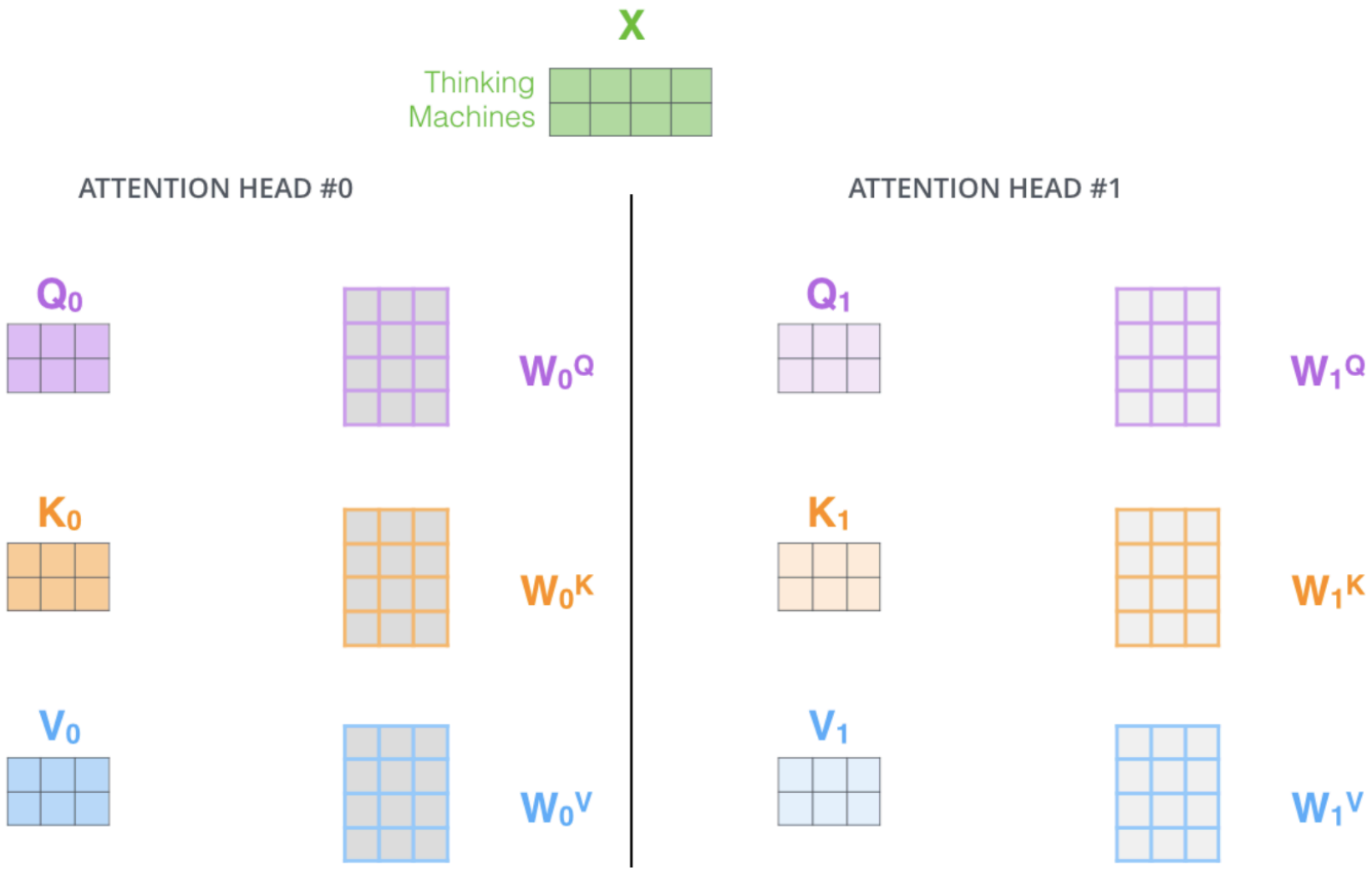
# Multiple Self-Attention

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



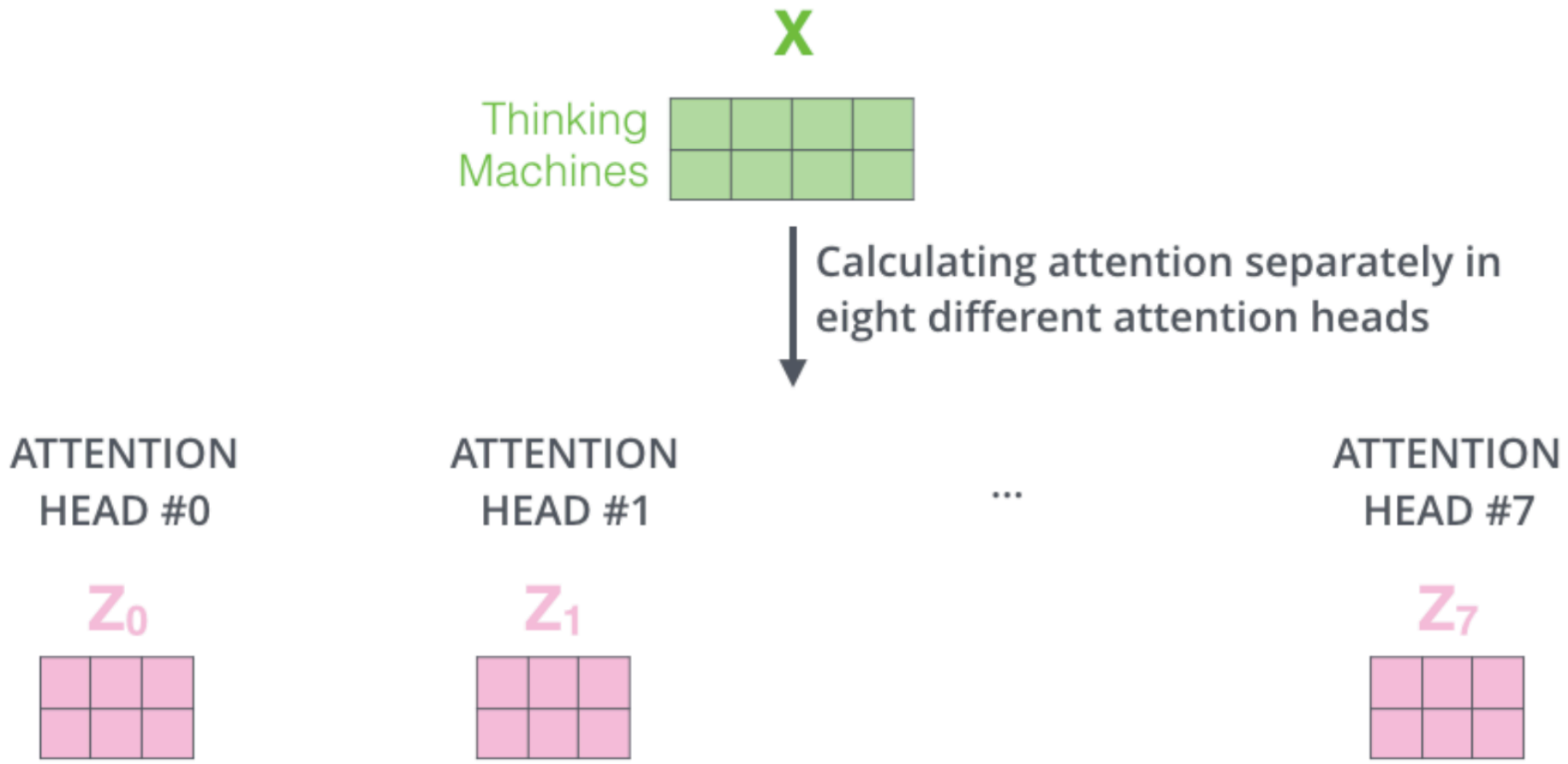
# Multiple Self-Attention

## Multiple QKV matrices



# Multiple Self-Attention

## Multiple (8) Self-Attention Outputs



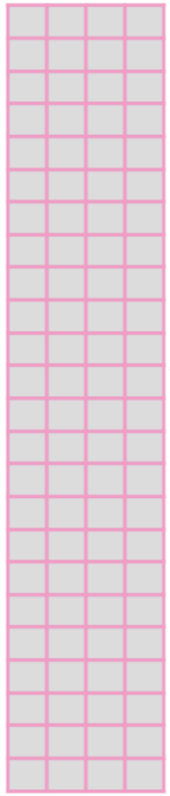
# Multiple Self-Attention

## Merging Multiple Outputs into One

1) Concatenate all the attention heads

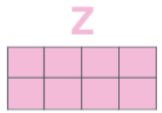

2) Multiply with a weight matrix $W^O$ that was trained jointly with the model

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

=



# Multiple Self-Attention
## From Start to Finish

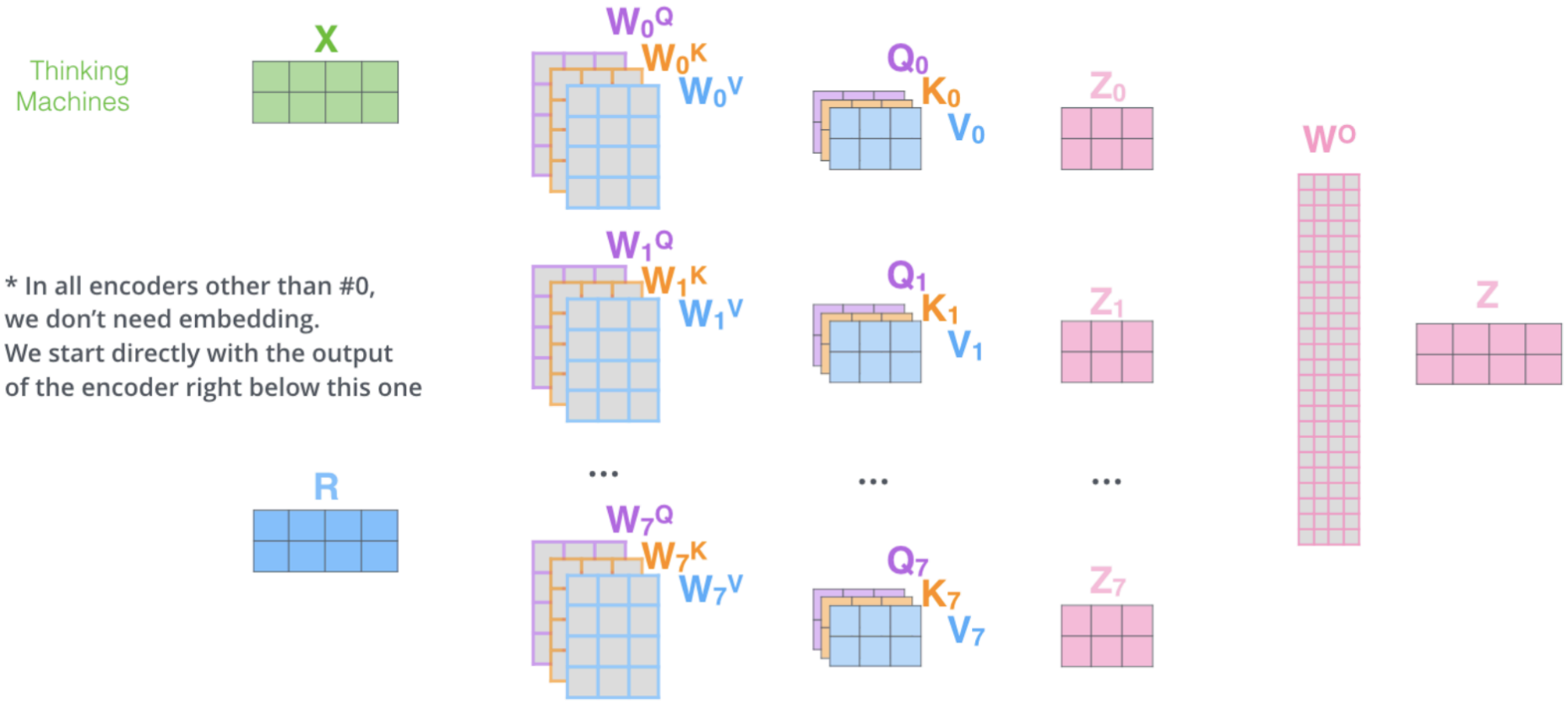
1) This is our input sentence*

2) We embed each word*

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

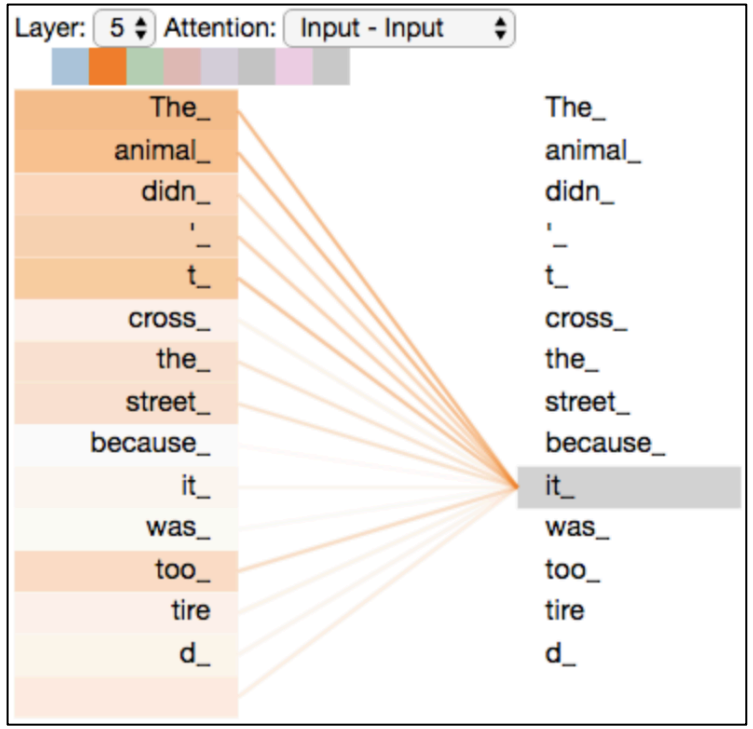
4) Calculate attention using the resulting Q/K/V matrices

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

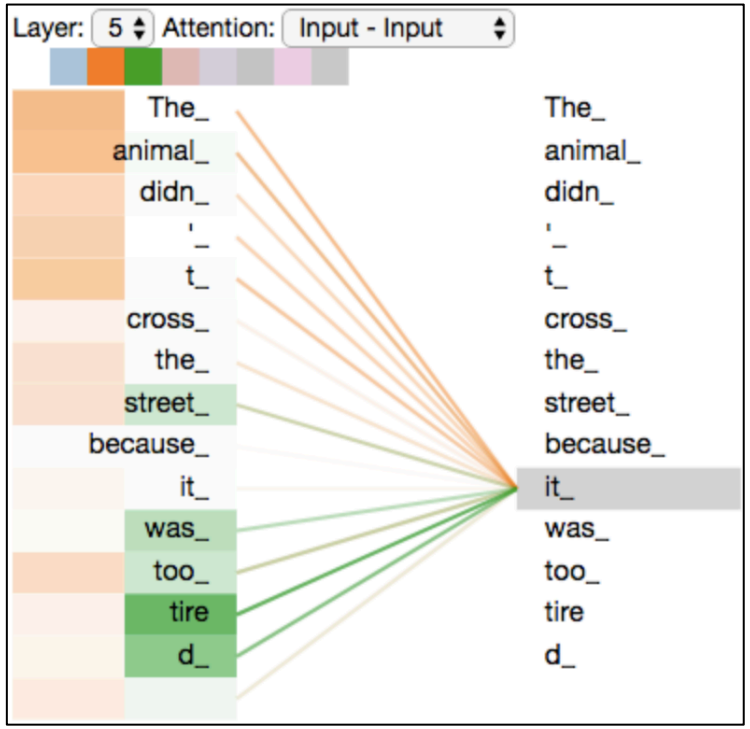



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

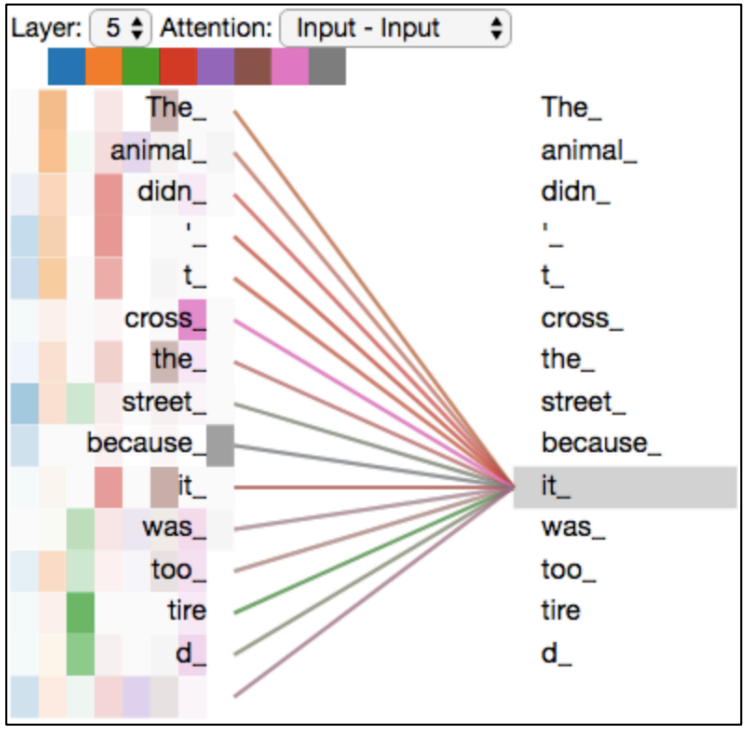
# Visualizing the Attention




Visualizing 1 attention




Visualizing 2 attentions

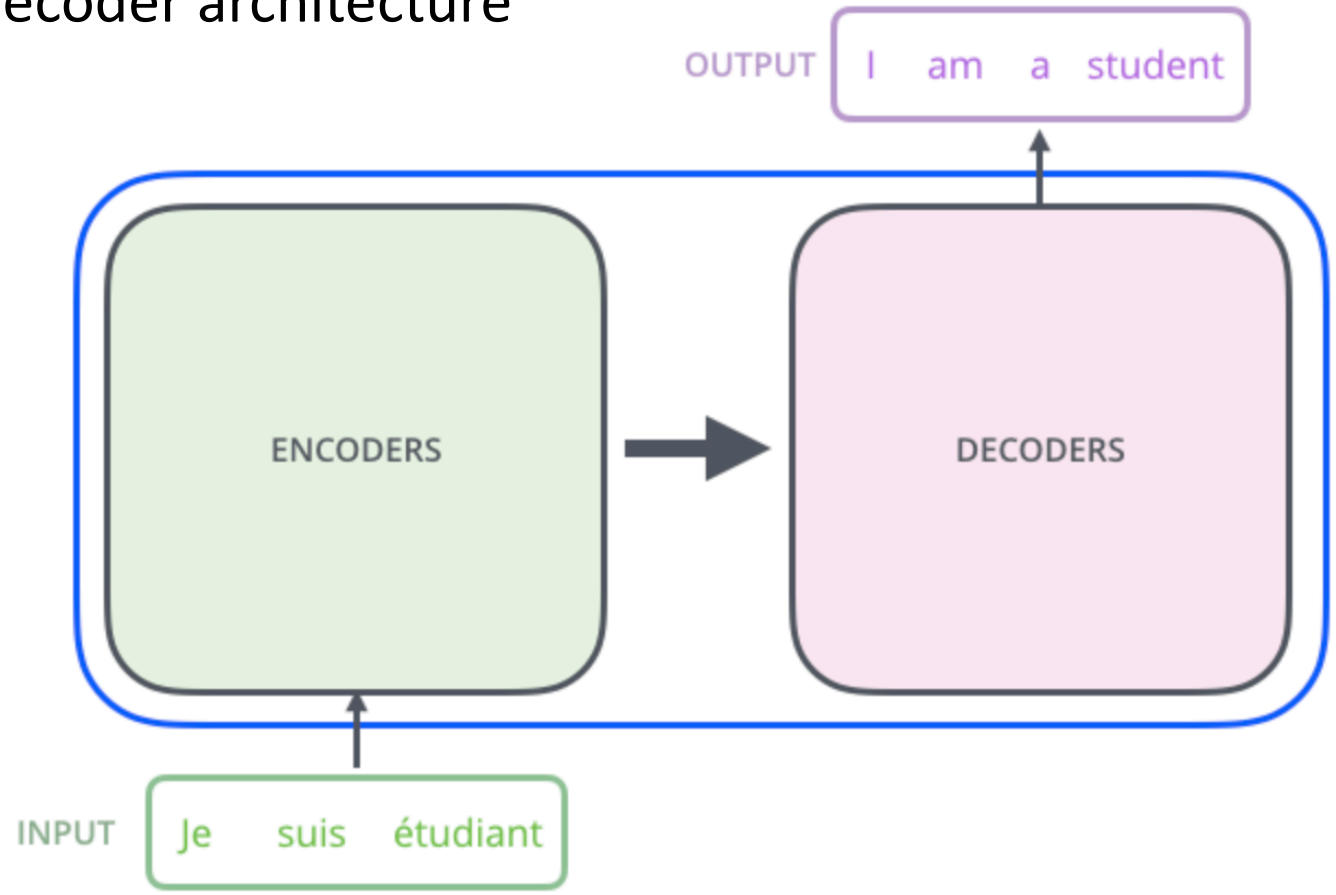



Visualizing 8 attentions

# What is Transformer (Encoder)?

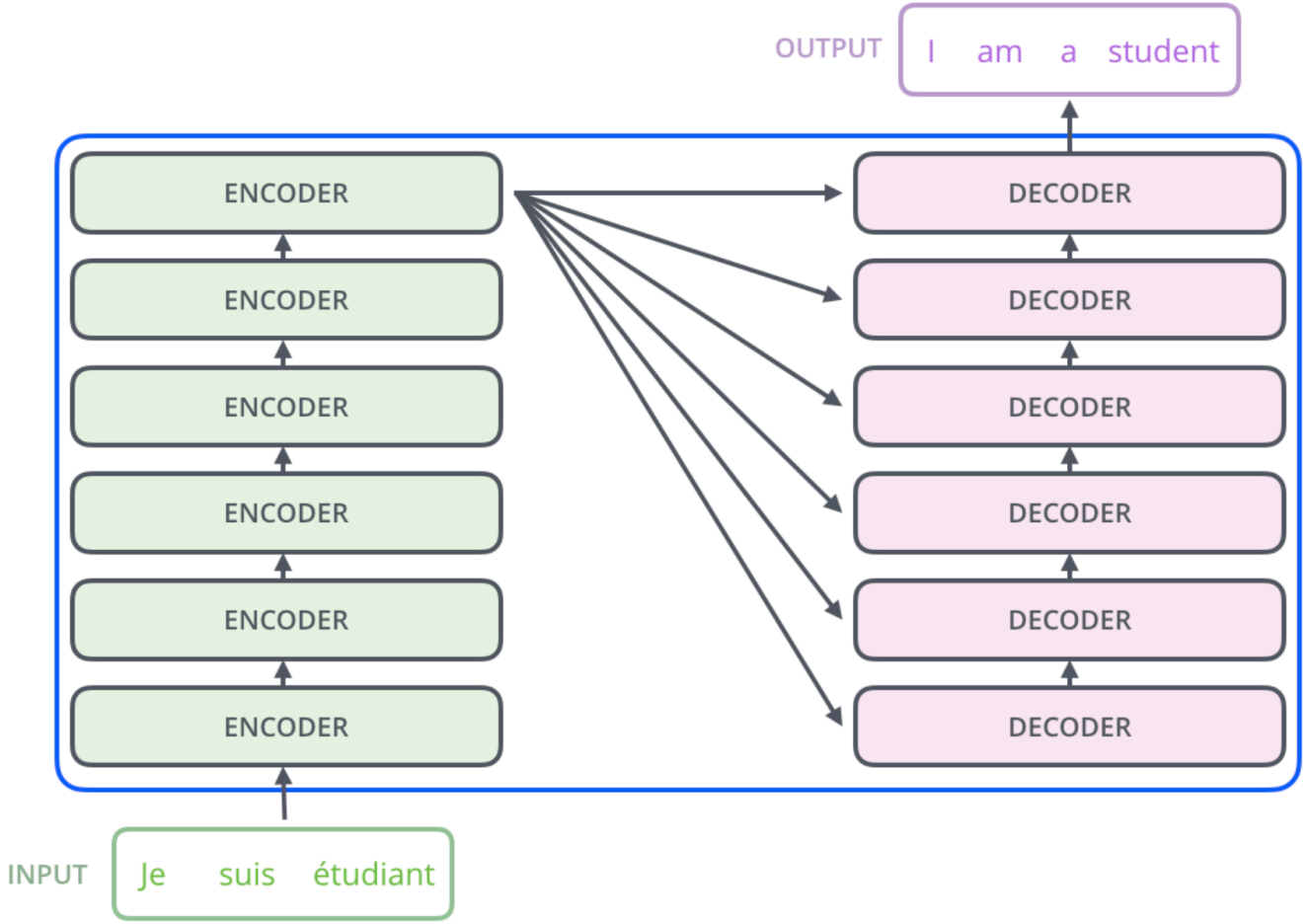
# Transformers

- Originally designed for translation
  - Encoder-Decoder architecture



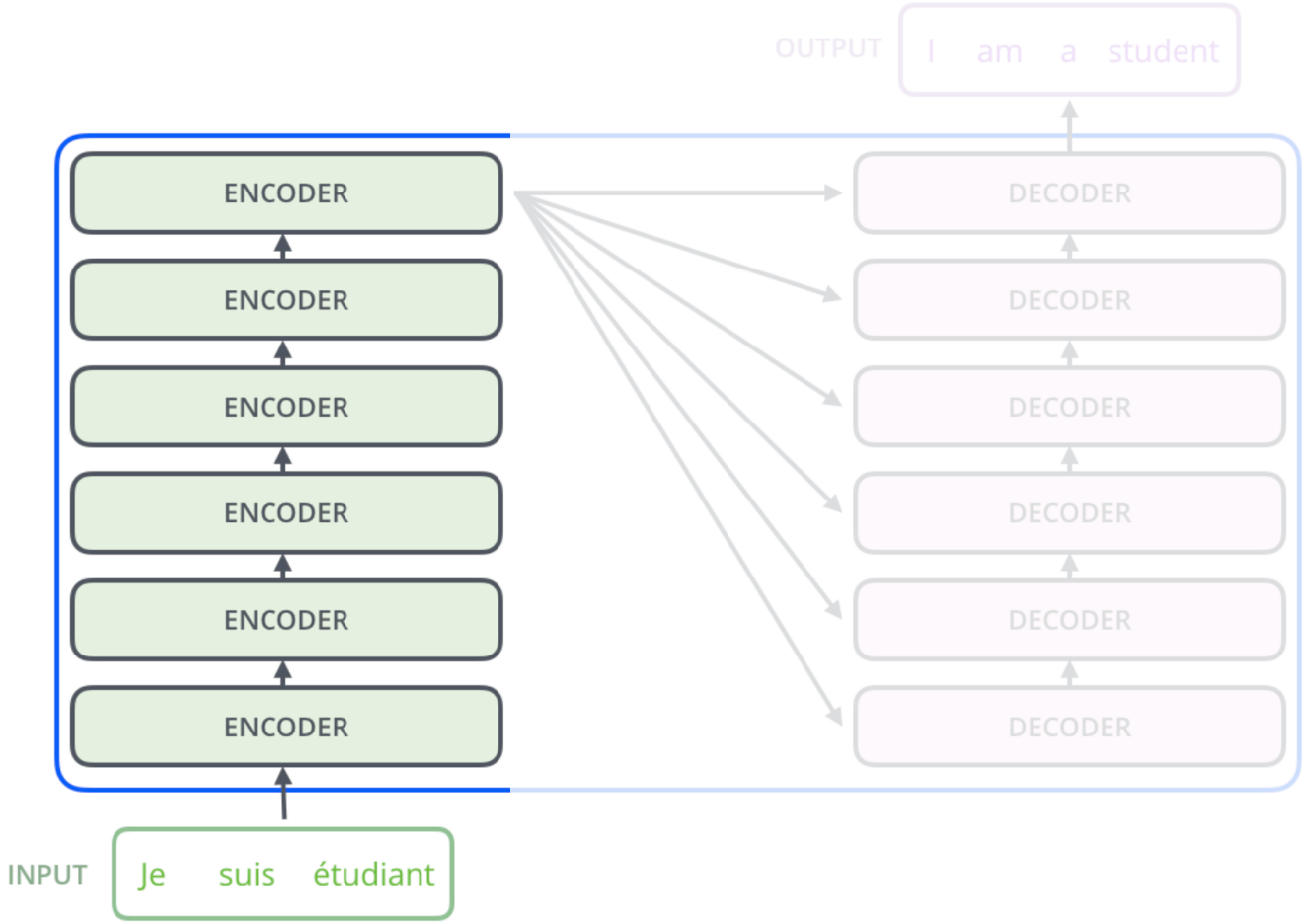
# Transformers

- Encoder and decoder has many “blocks”.



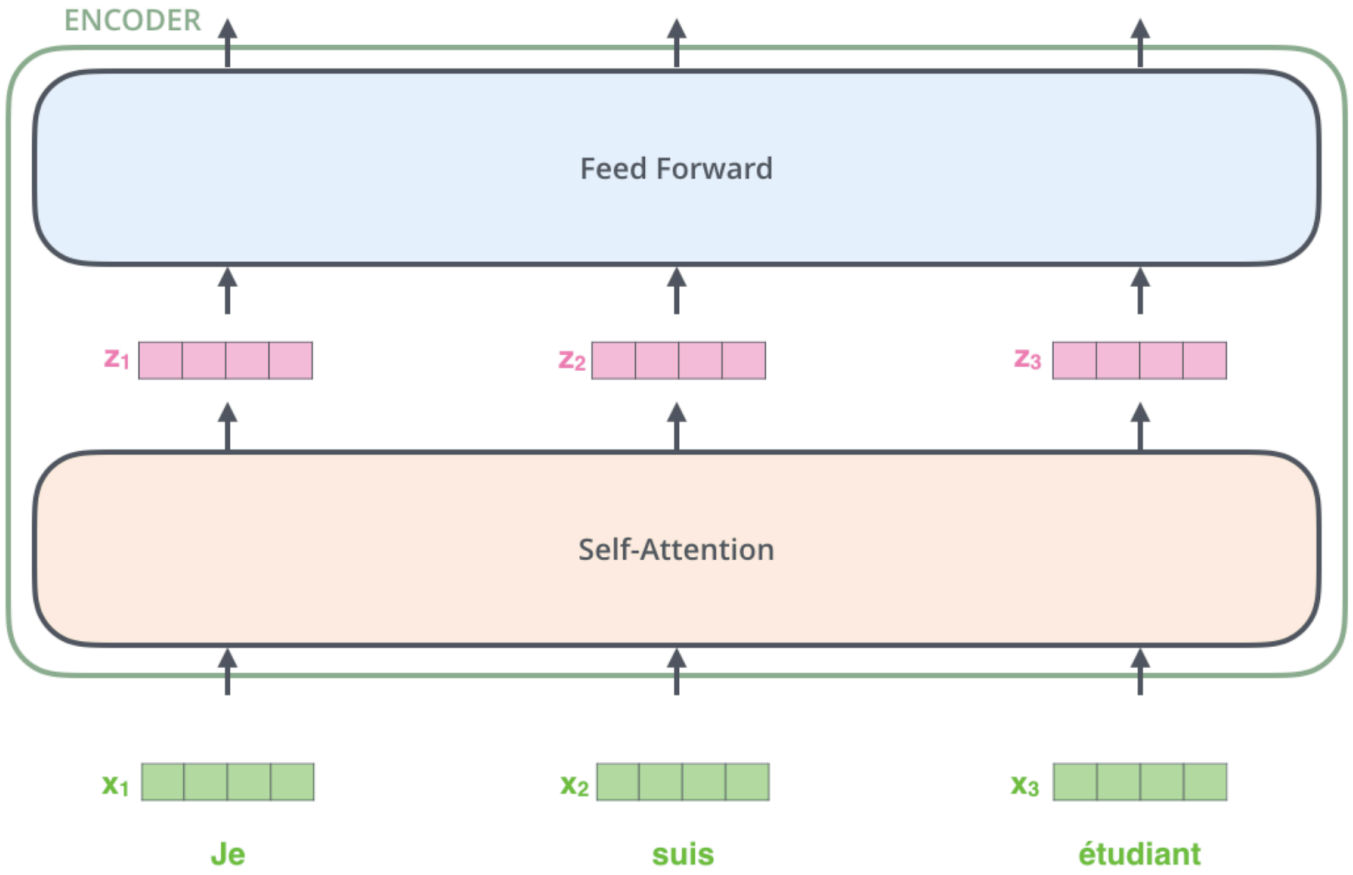
# Transformers

- First focus on the **Encoder**



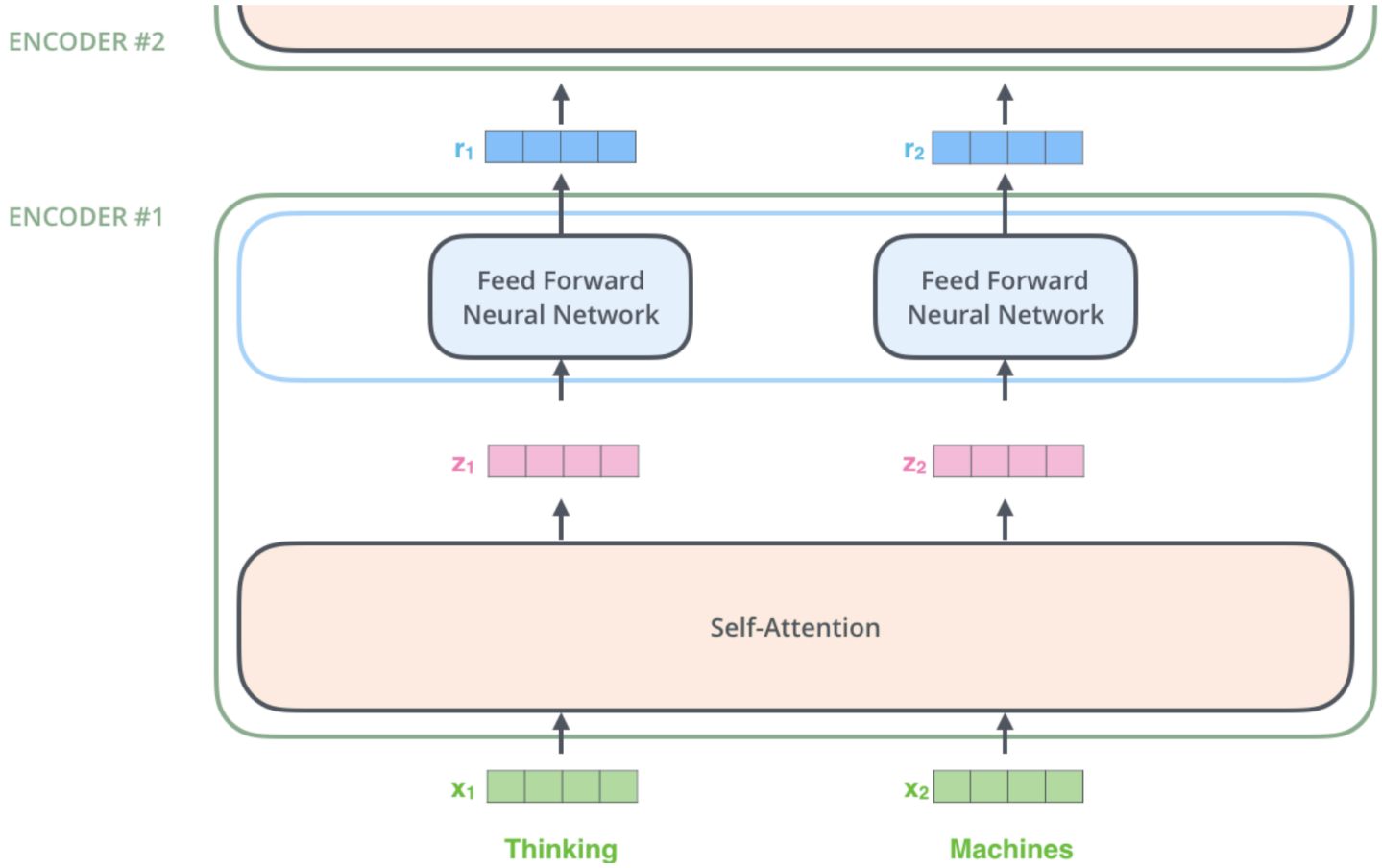
# Encoder, Single Block

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



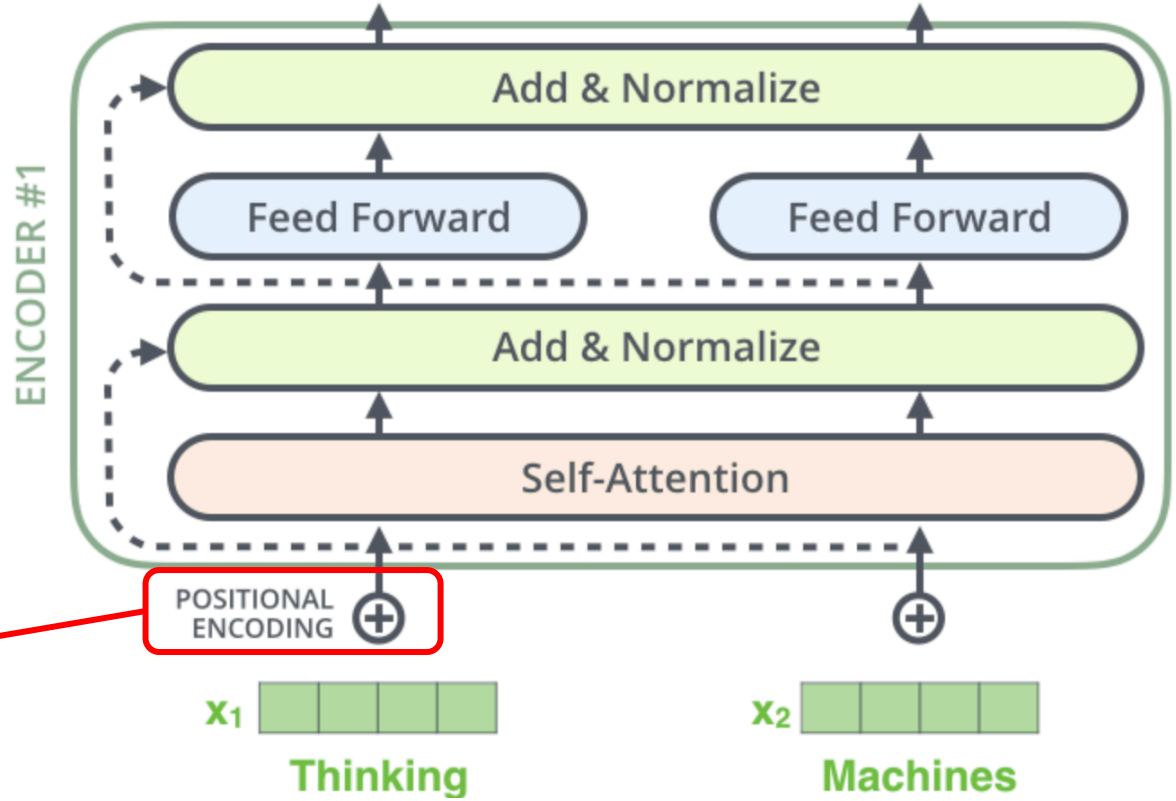
# Encoder, Sequence of Blocks

- Input from previous block → Self-attention → MLP → Output



# Encoder, Residual + Layer Norm

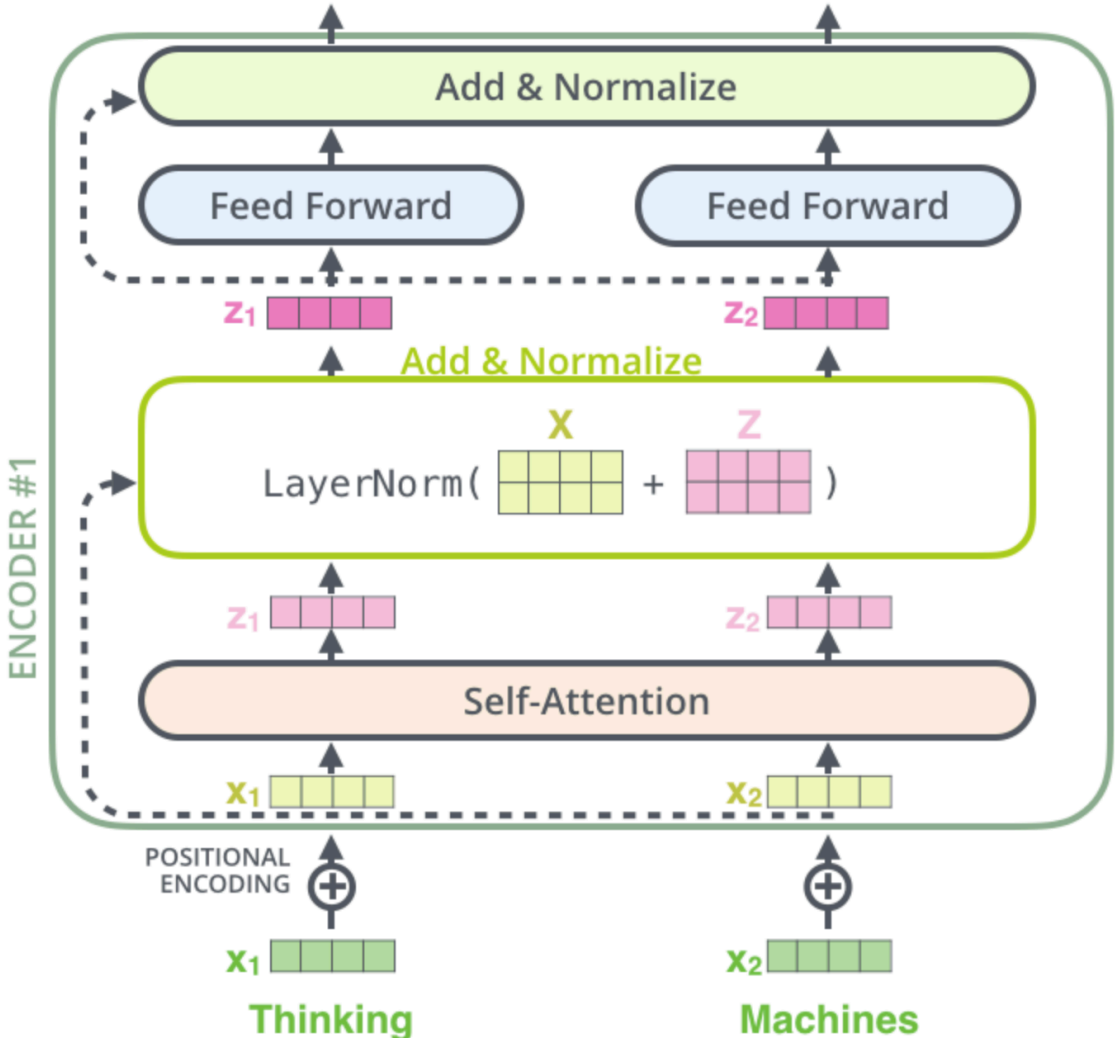
- Residual Connection (from ResNet)
- Layer Normalization (similar but different to BatchNorm)




We will discuss this later

# Encoder, Residual + Layer Norm

- Visualizing the Vectors



# 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 | 0.1 | 0.0 |
| .. | .. | .. | .. | .. | .. |
| .. | .. | .. | .. | .. | .. |
| .. | .. | .. | .. | .. | .. |
| .. | .. | .. | .. | .. | .. |

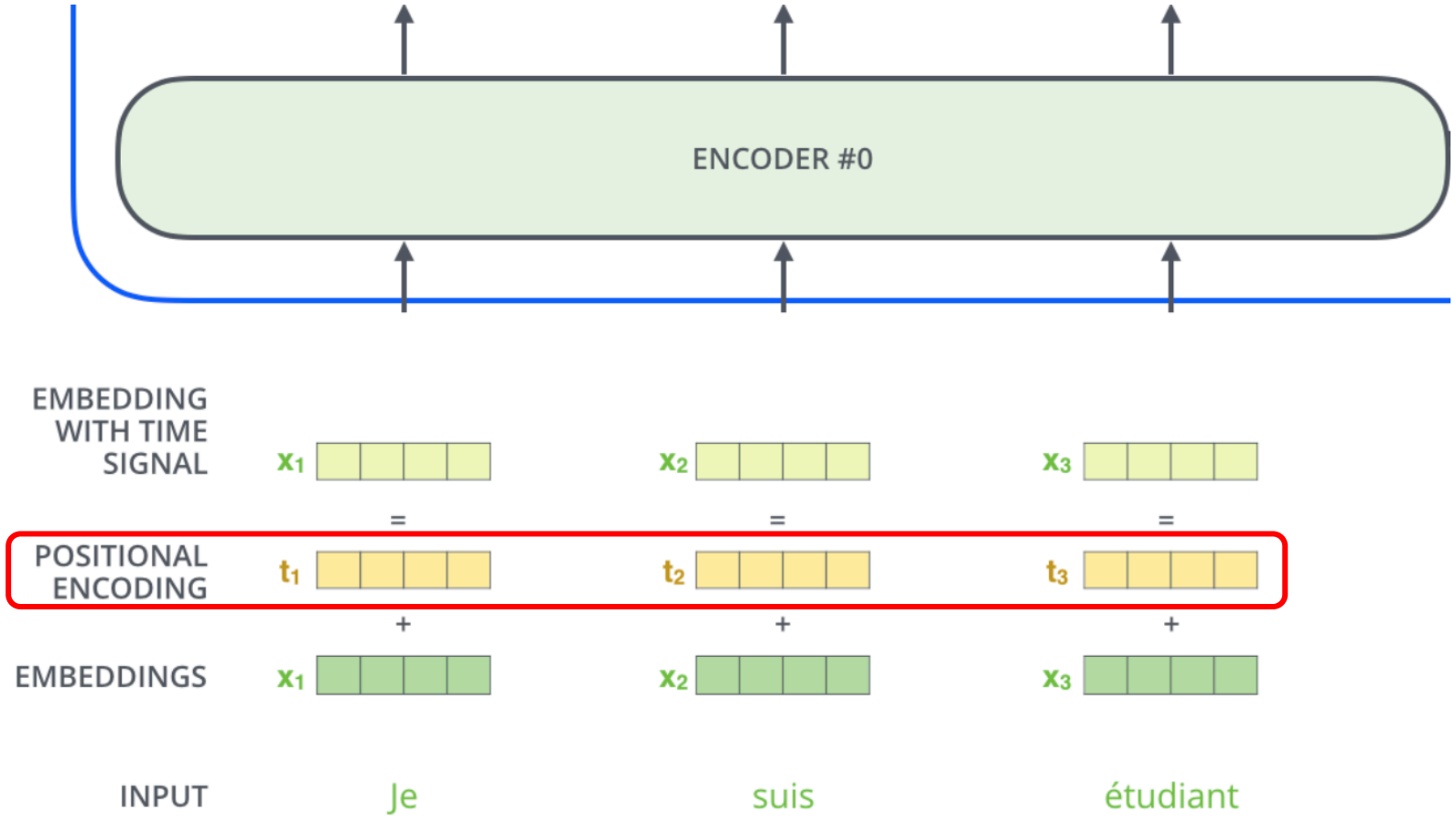
| |
|---|
| I |
| like |
| going |
| to |
| movies |
| <end> |

| |
|---|
| 0.5*I + 0.1*like + 0.2*to + 0.2* movies |
| 0.2*I + 0.6*like + 0.1*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

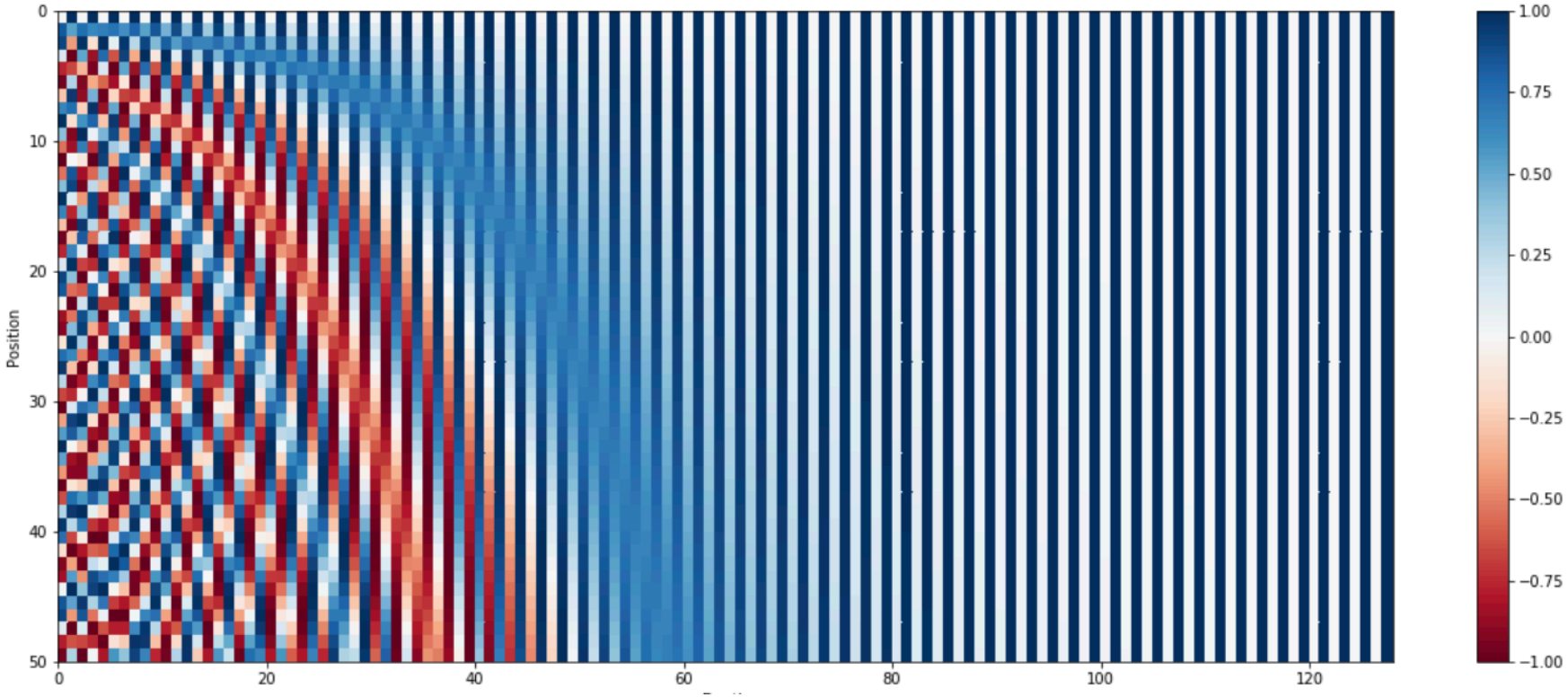
- Use binary values

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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

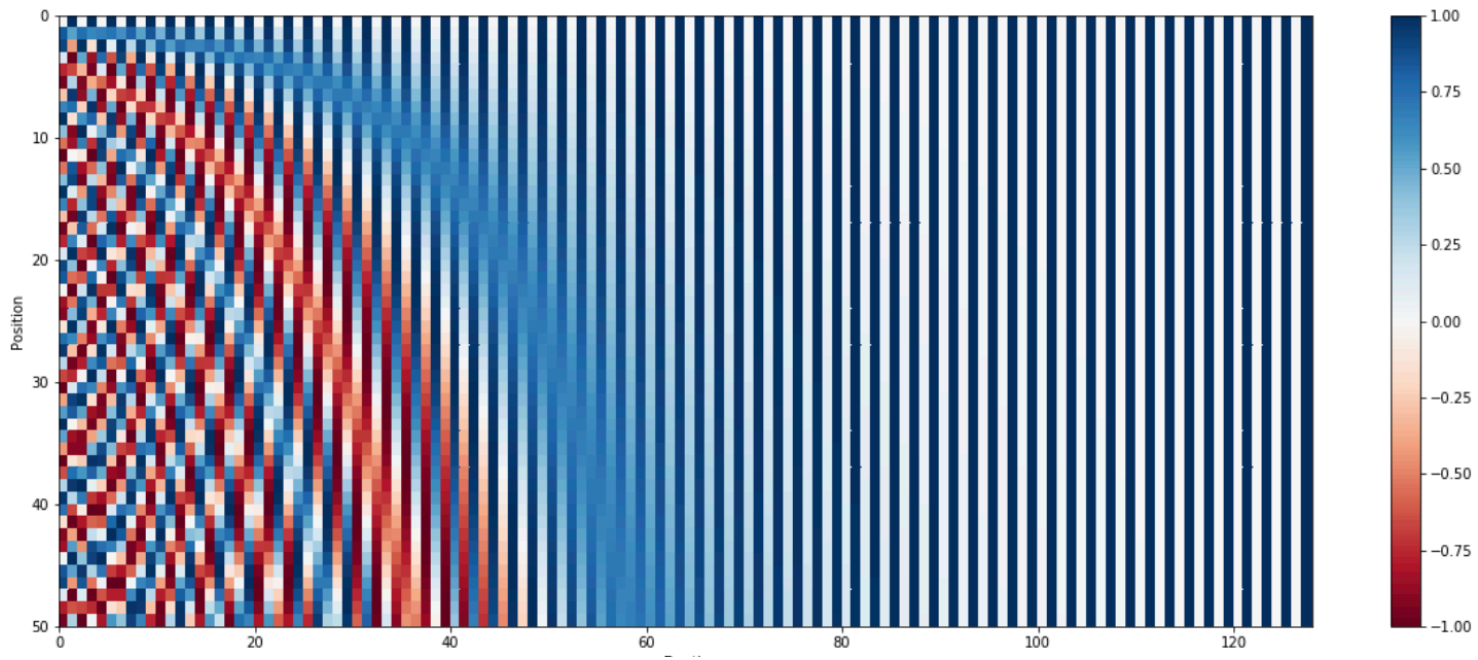


# Positional Encoding

- Use sinusoidal functions

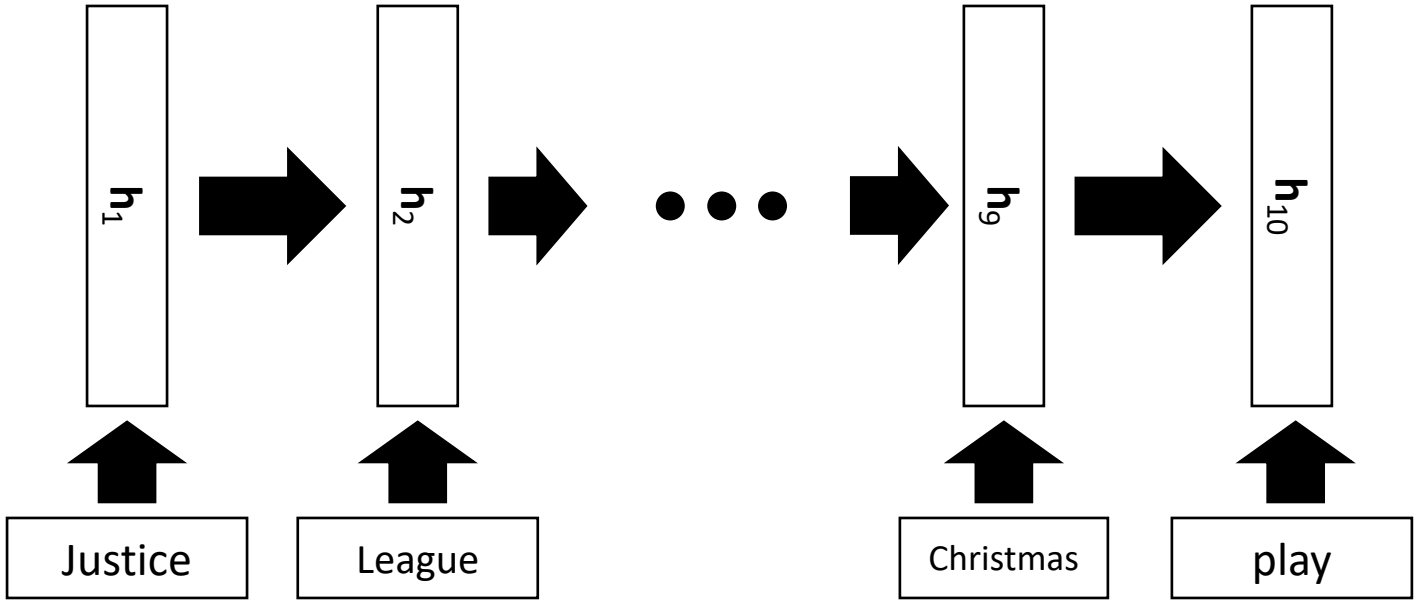
$$\begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}$$

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



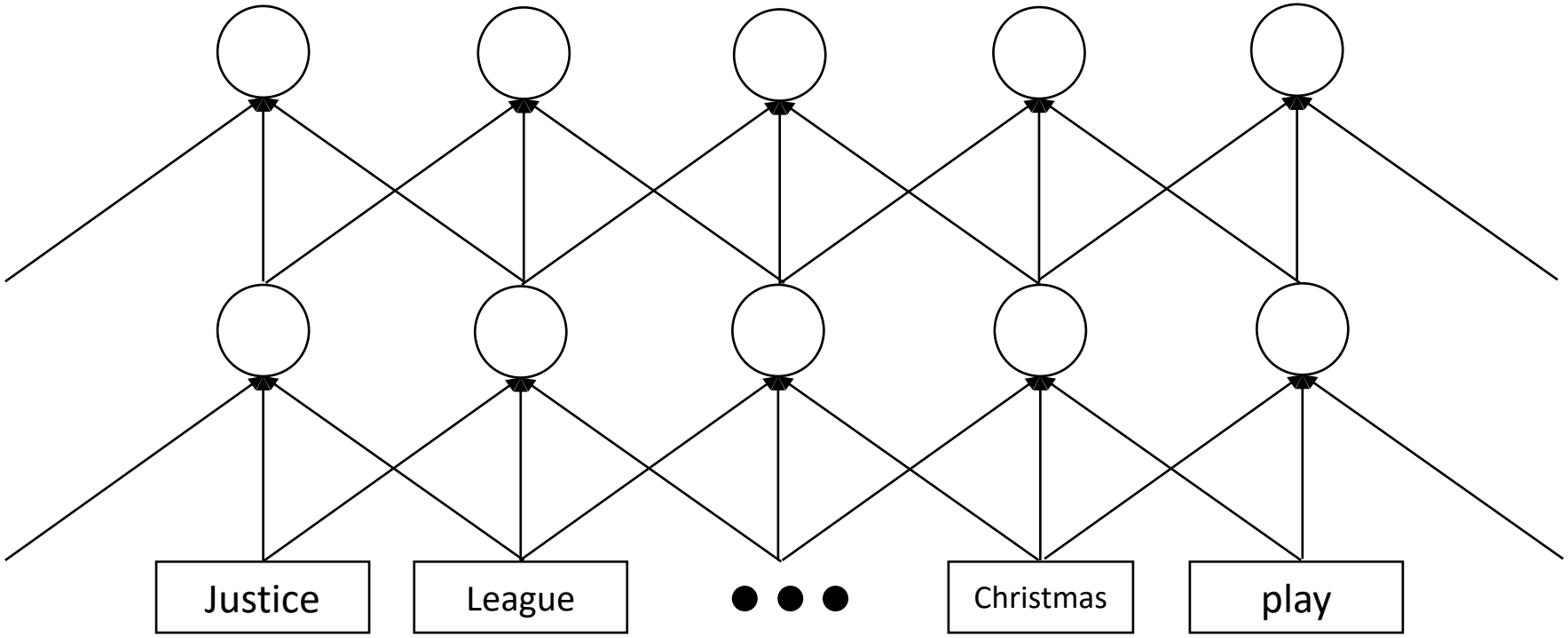
# RNN, 1D Conv, Transformers

- RNN, given a sequence of embeddings



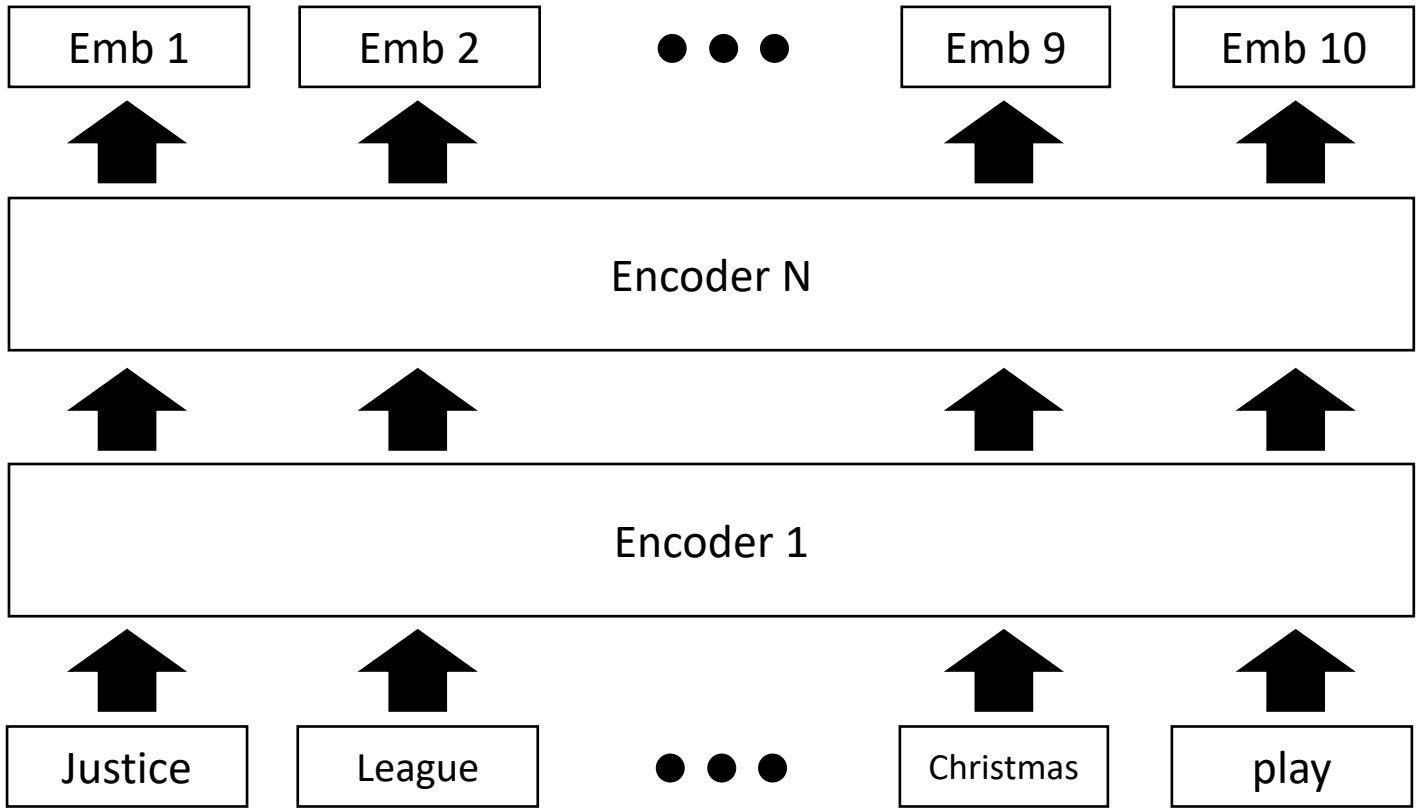
# RNN, 1D Conv, Transformers

- 1-D CNN, given a sequence of embeddings



# RNN, 1D Conv, Transformers

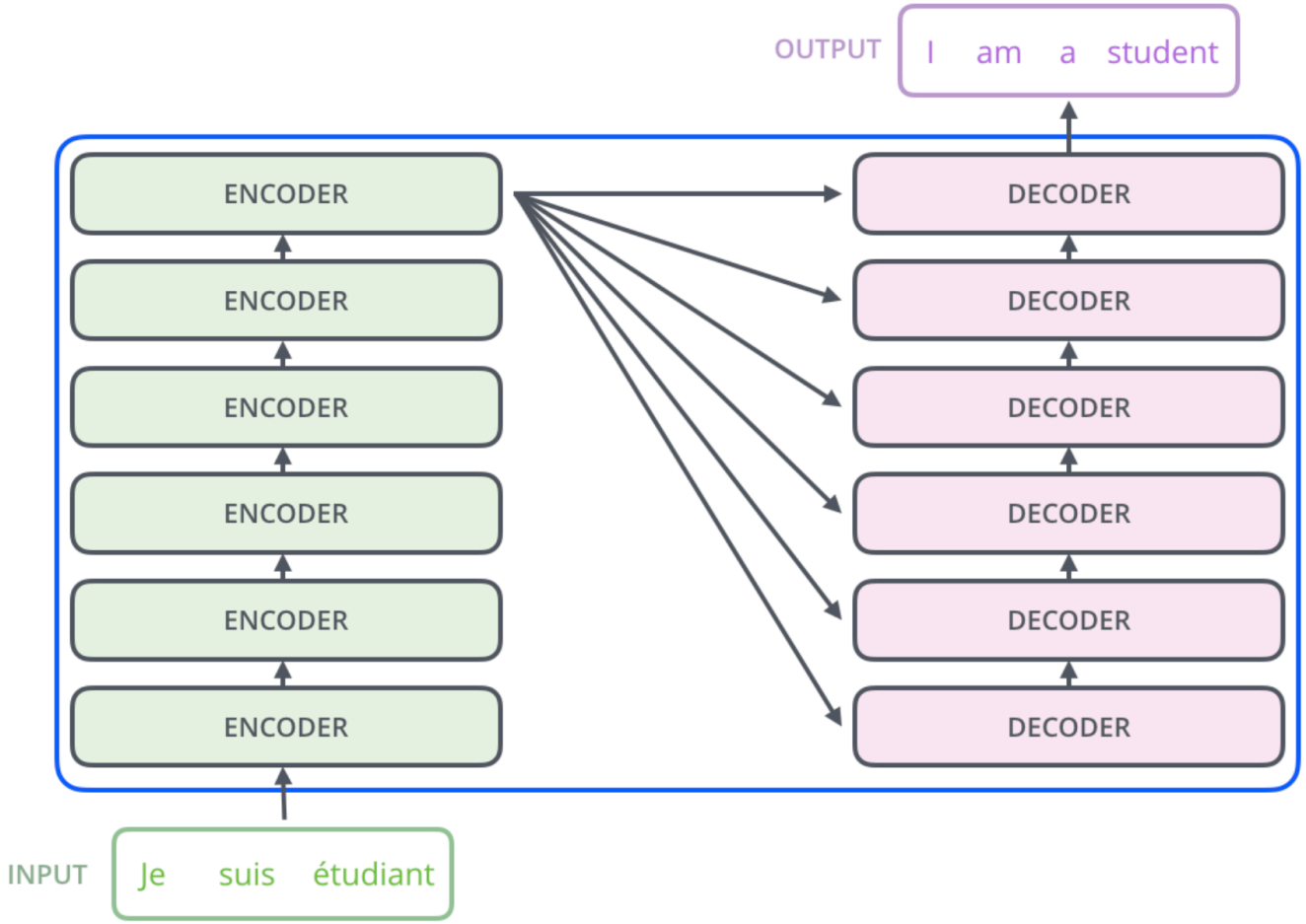
- Transformer, given a sequence of embeddings



# Sequence-to-Sequence

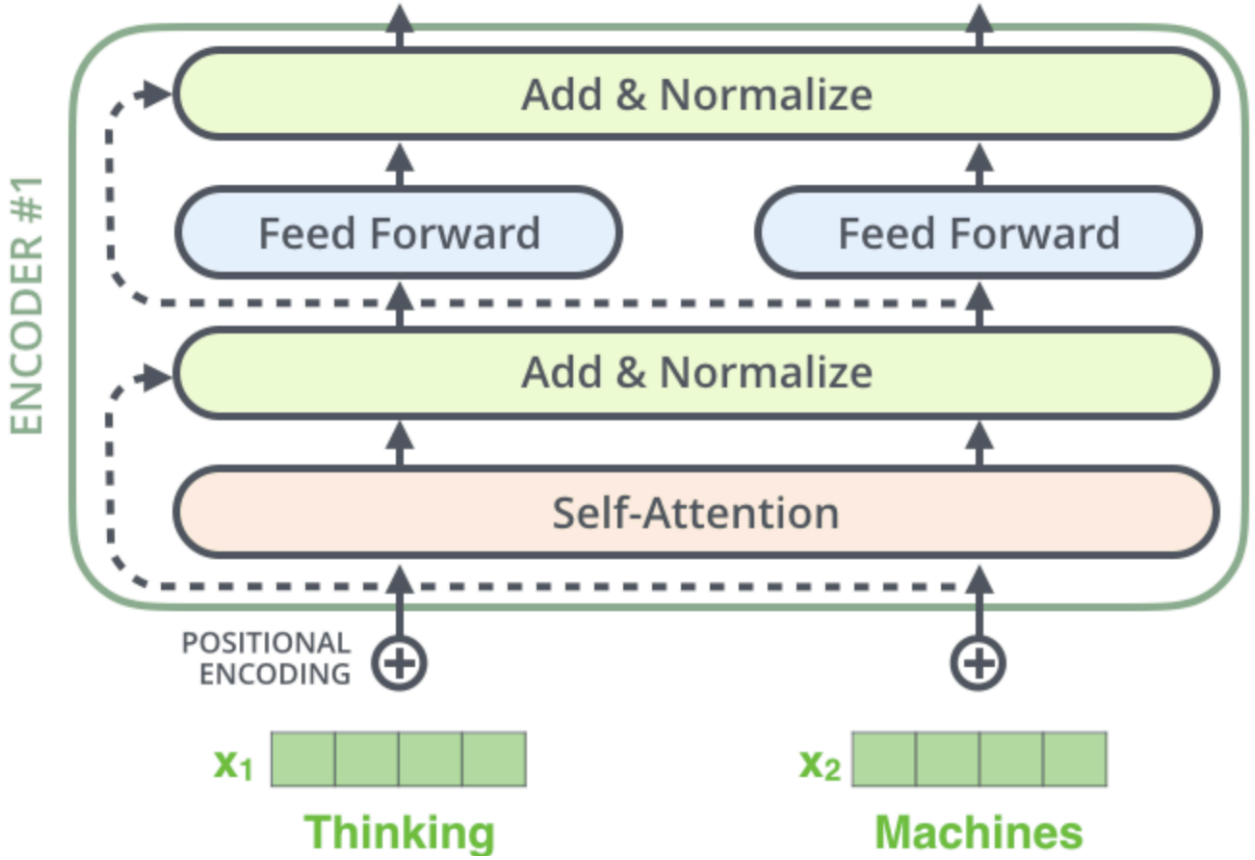
# Transformer

- Encoders and decoders



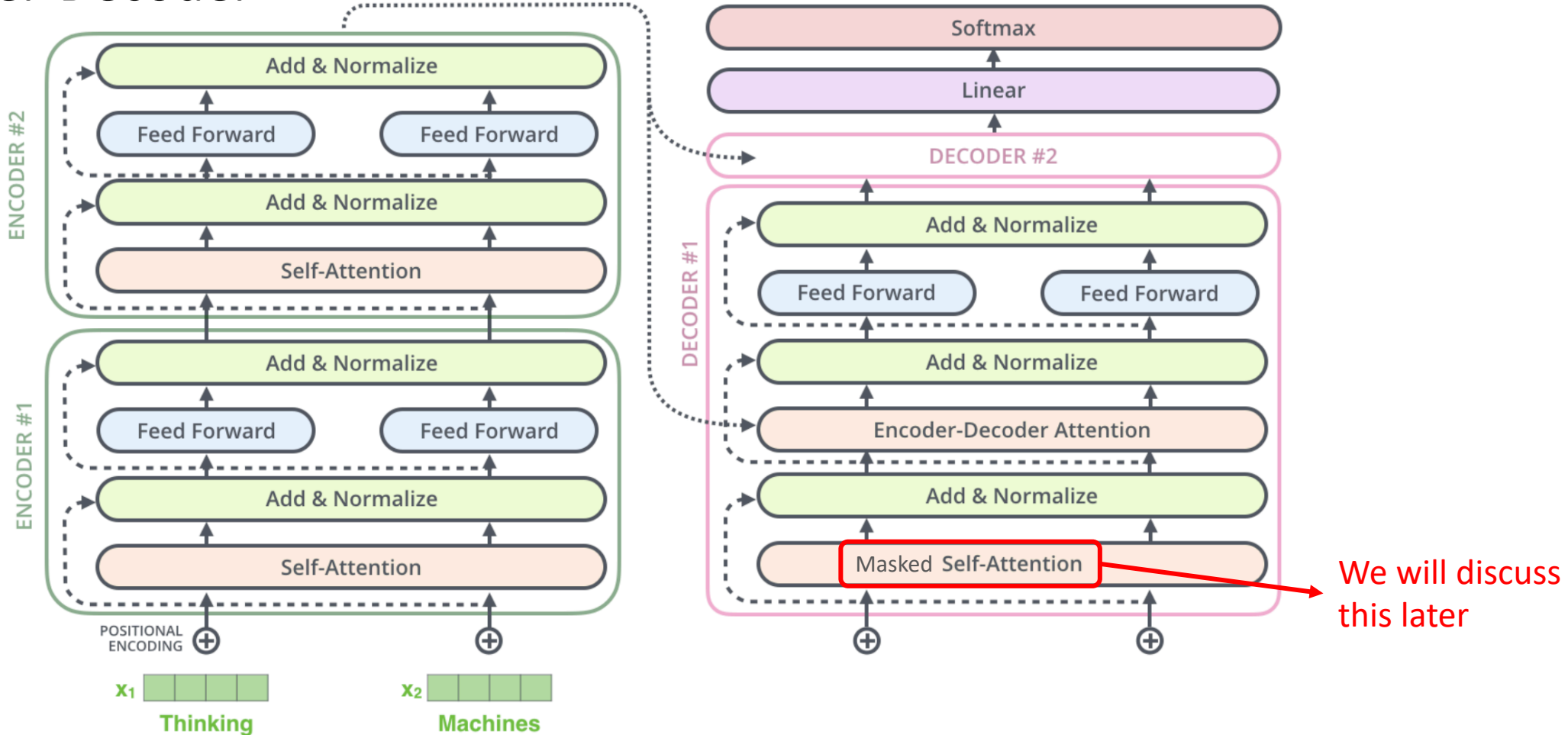
# Transformer

- Single encoder

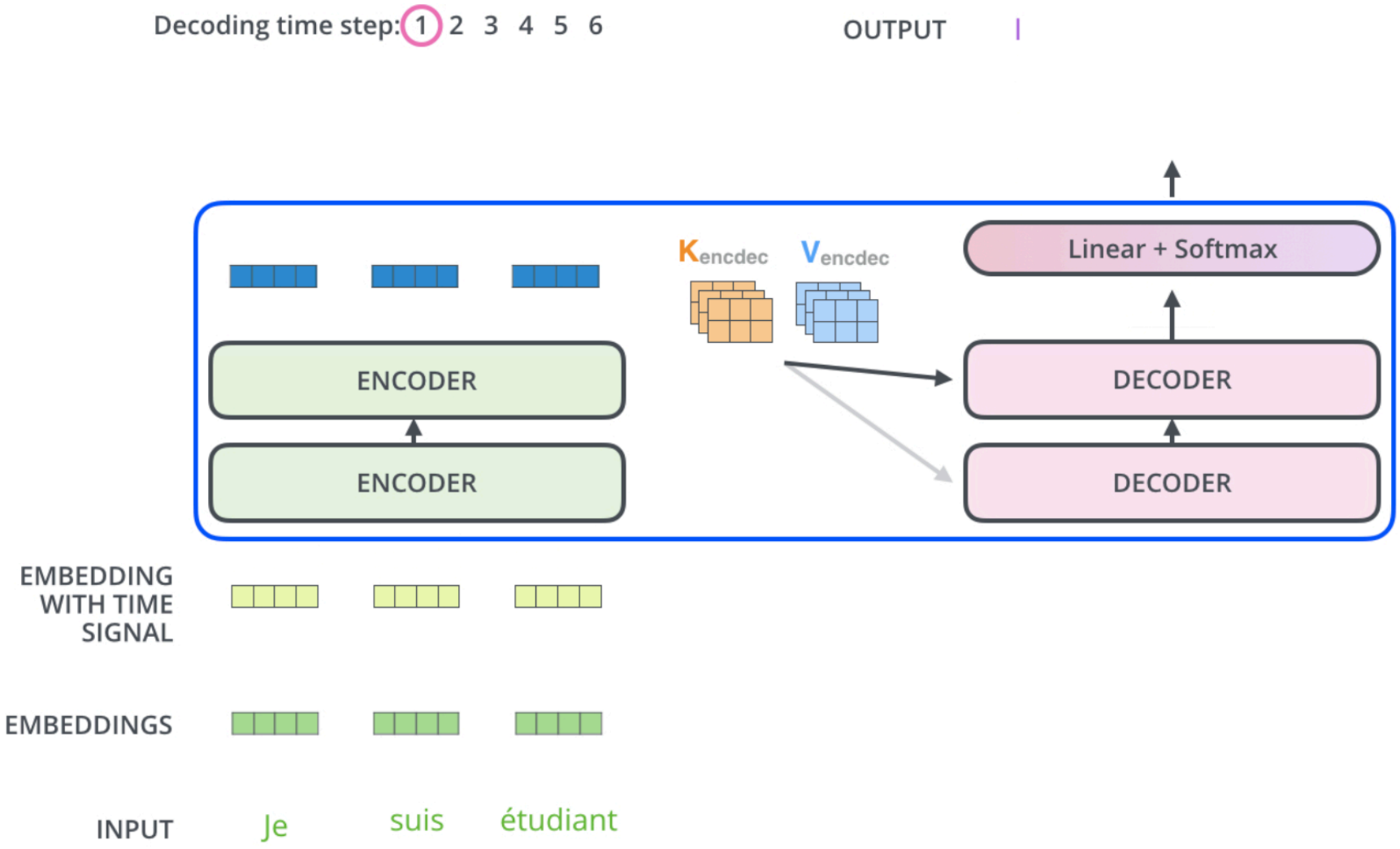


# Transformer

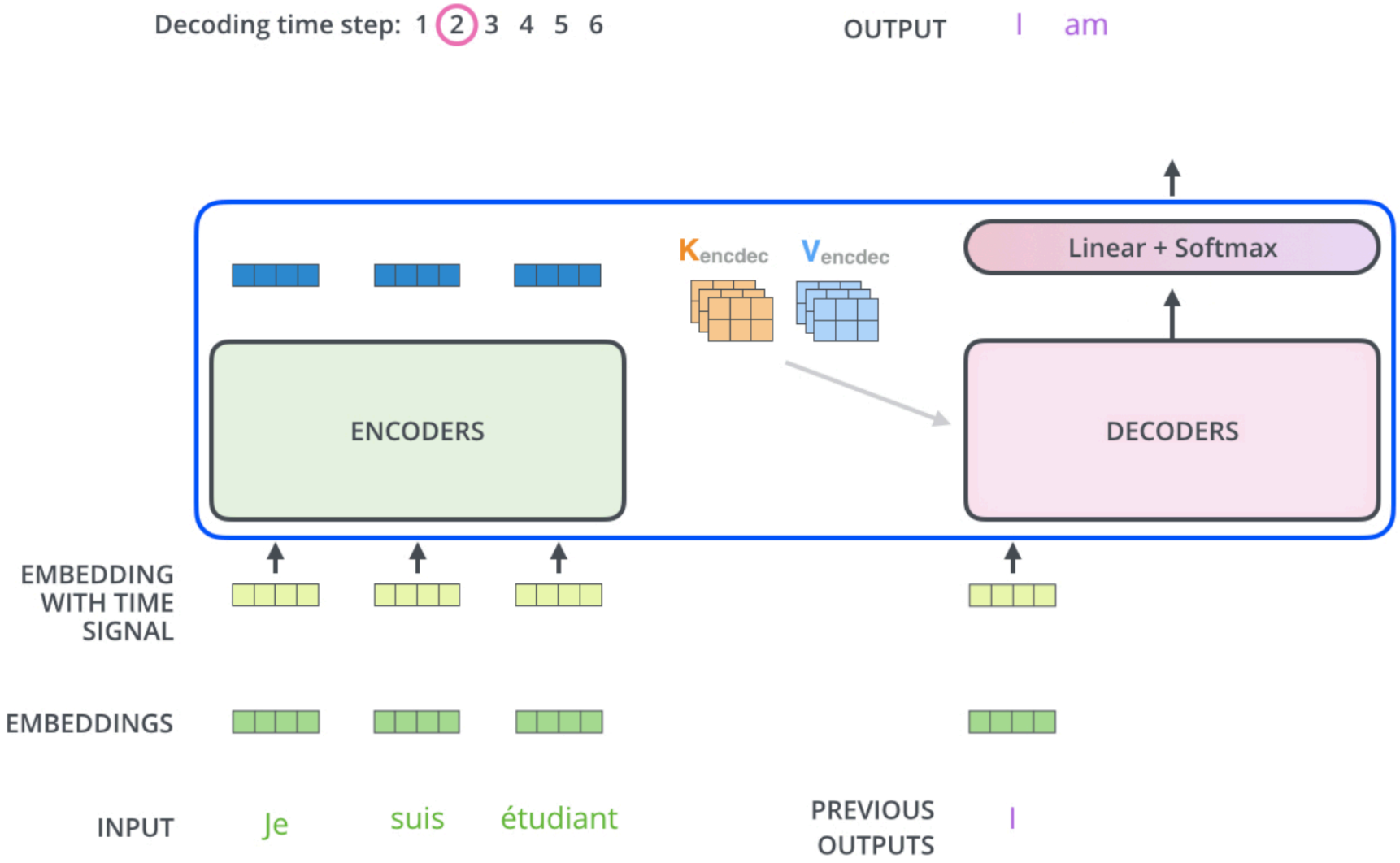
- Encoder-Decoder



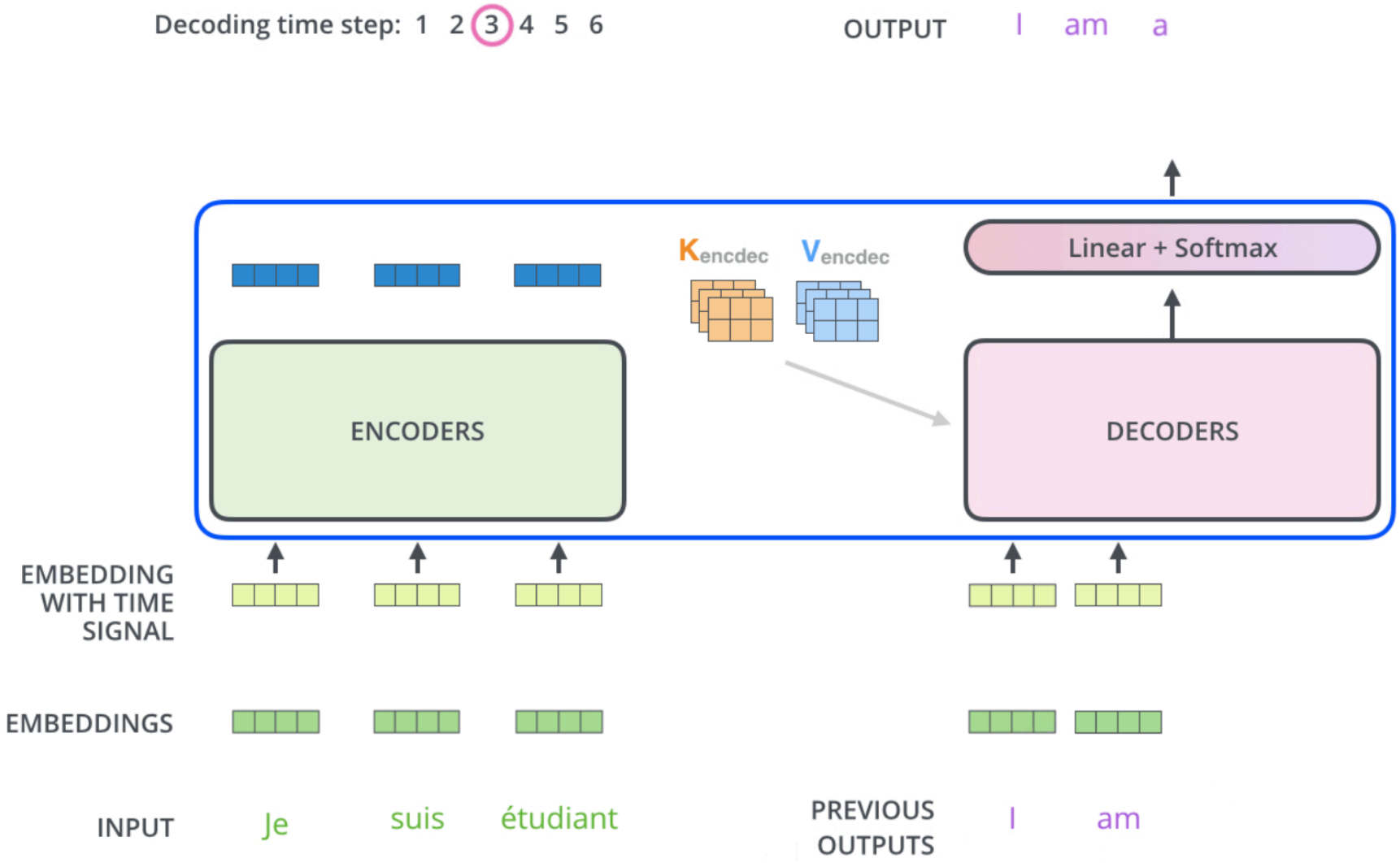
# Decoding Process



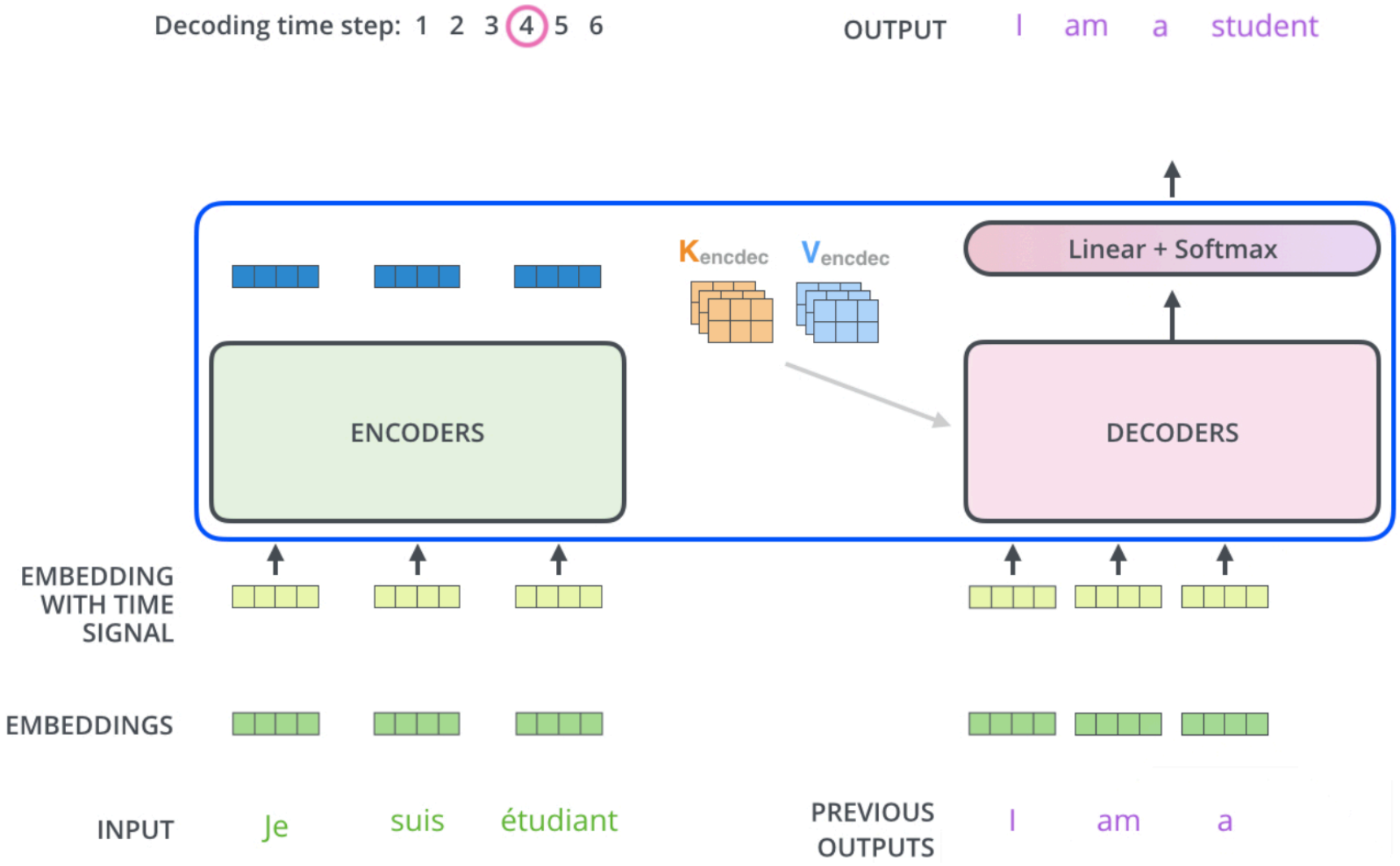
# Decoding Process



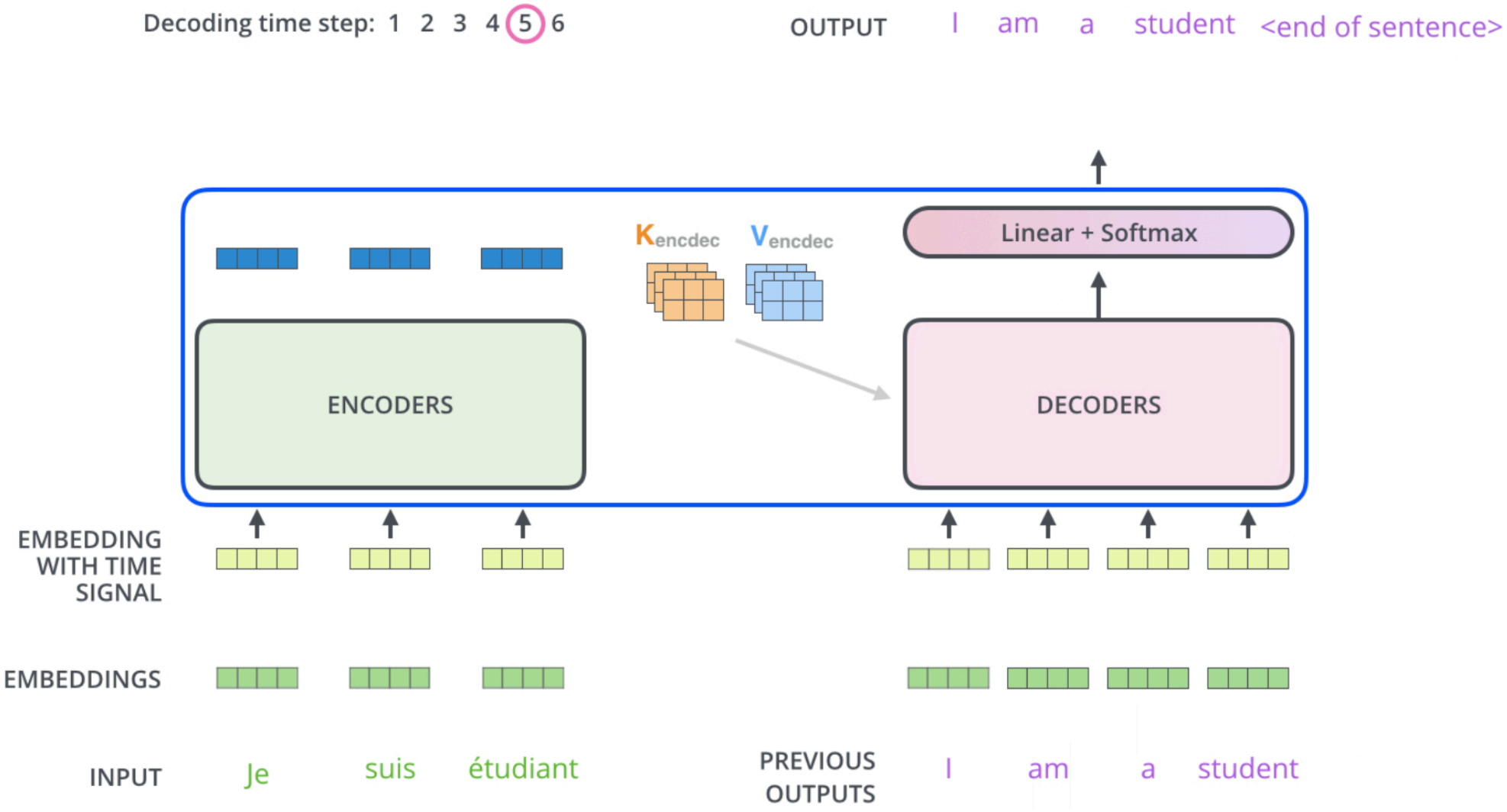
# Decoding Process



# Decoding Process



# Decoding Process



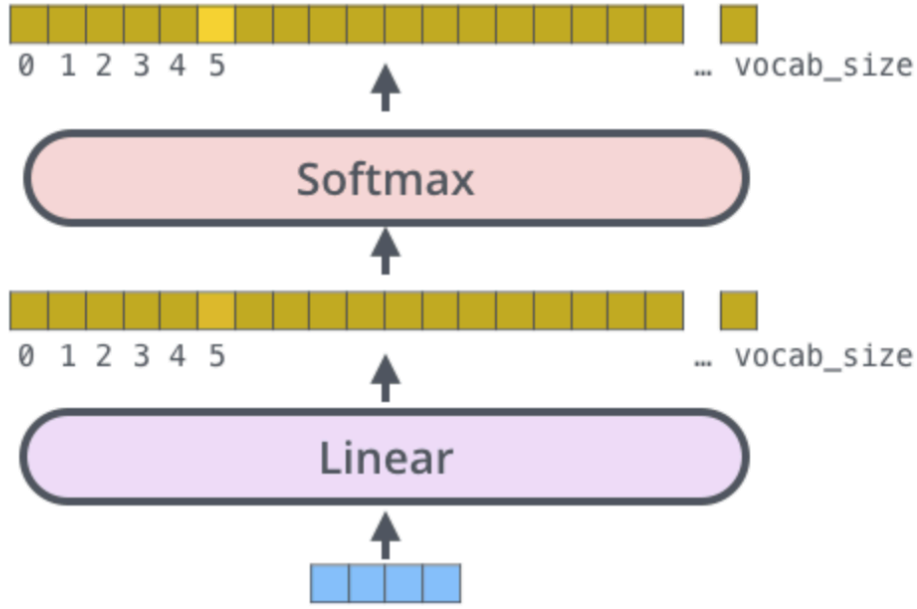
# Prediction Layer

Which word in our vocabulary is associated with this index?

am

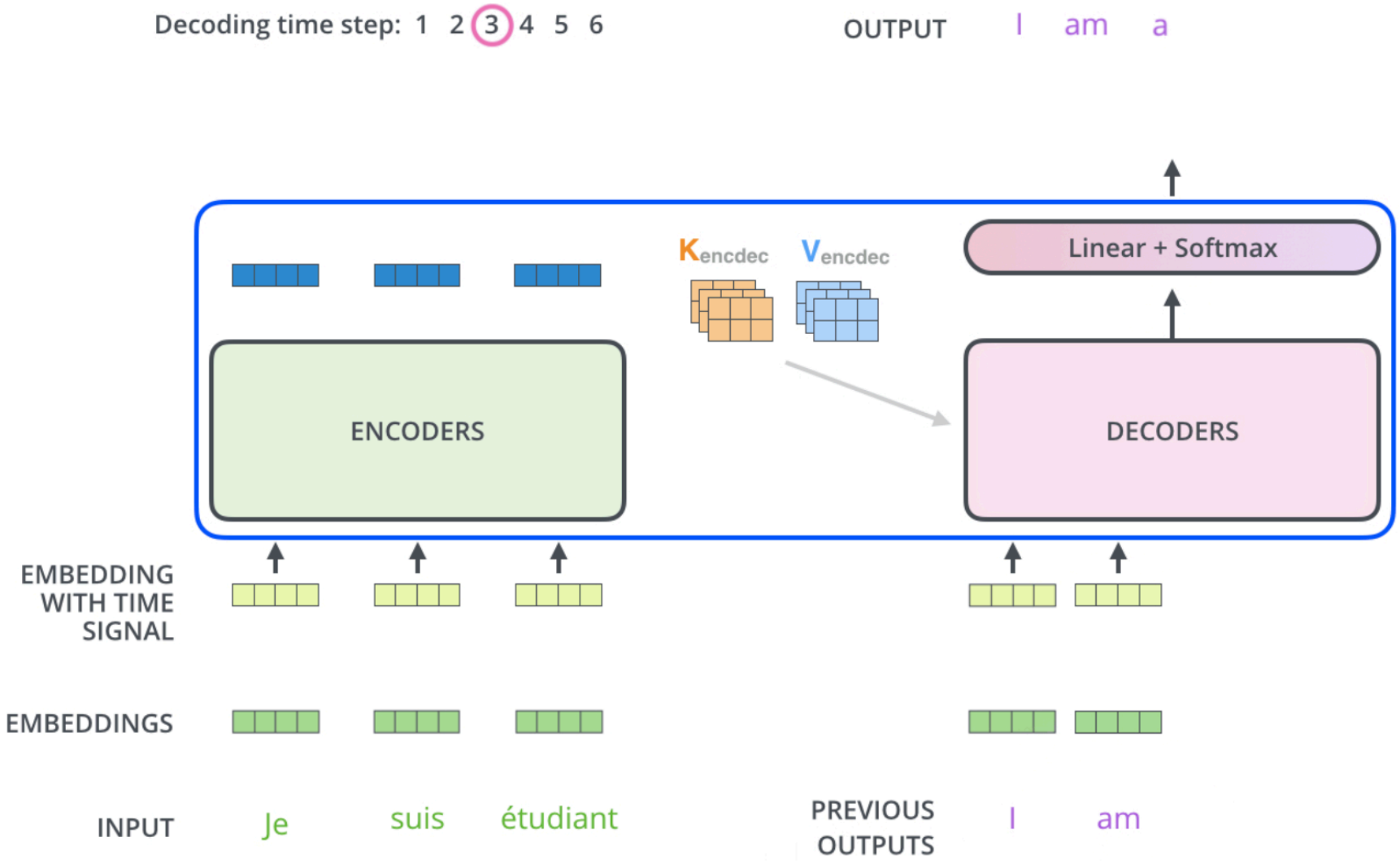
Get the index of the cell with the highest value (**argmax**)

5



# Masked Self-Attention

# Decoding Process



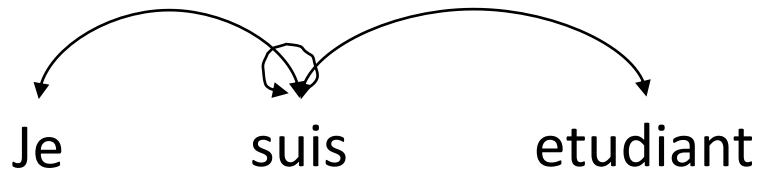
# Decoding Process

- Given:
  - Je, suis, etudiant
  - I, am
- Predict:
  - a

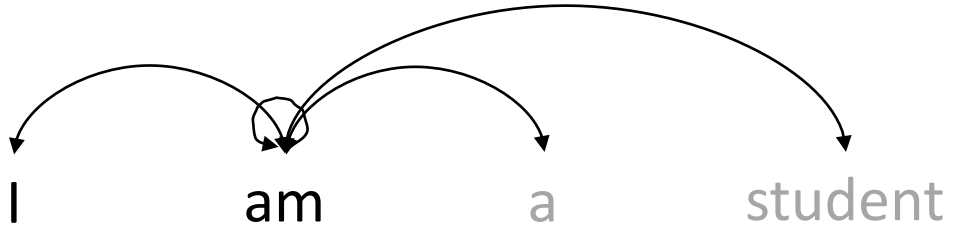
# Decoding Process

- Given:
  - Je, suis, etudiant
  - I, am
- Predict:
  - a

Encoder attention from ”suis”

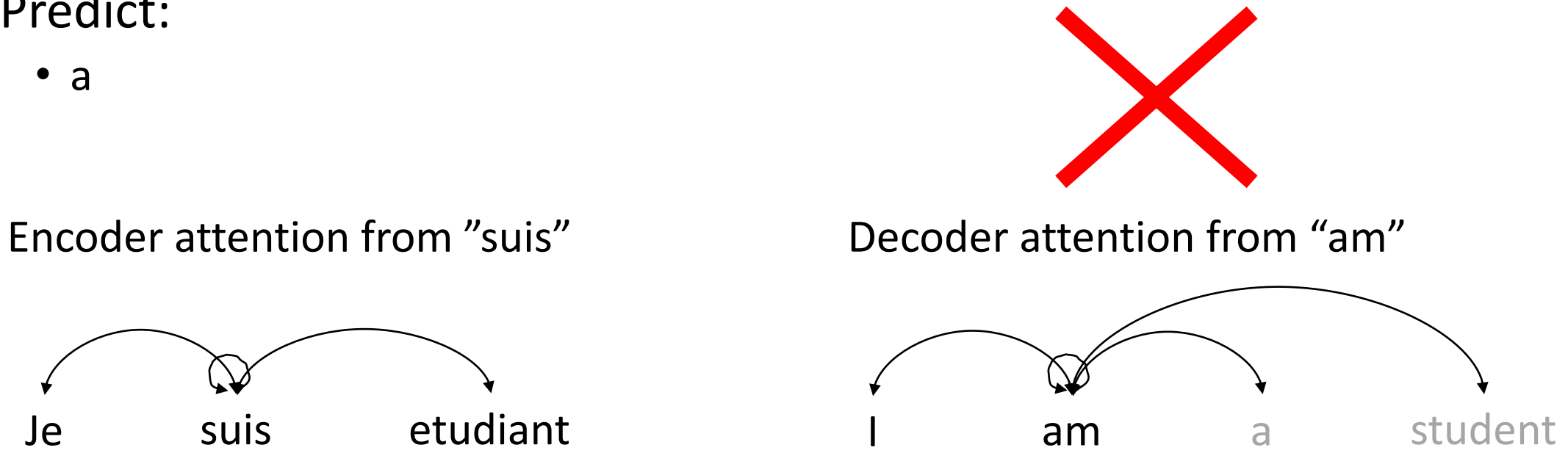



Decoder attention from “am”



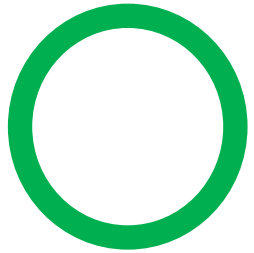
# Decoding Process

- Given:
  - Je, suis, etudiant
  - I, am
- Predict:
  - a

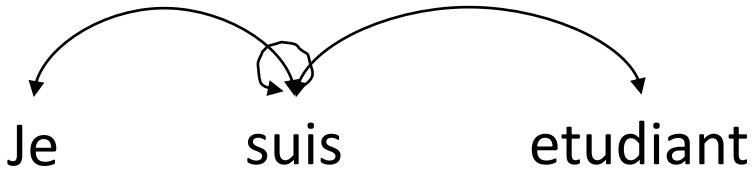


# Decoding Process

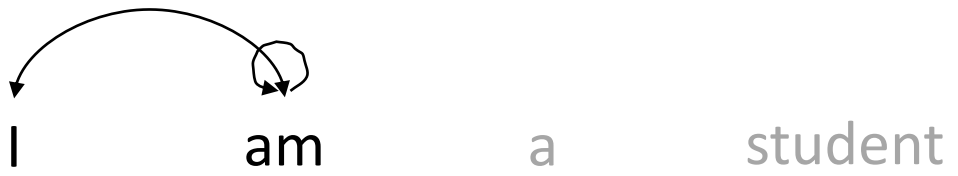
- Given:
  - Je, suis, etudiant
  - I, am
- Predict:
  - a



Encoder attention from ”suis”



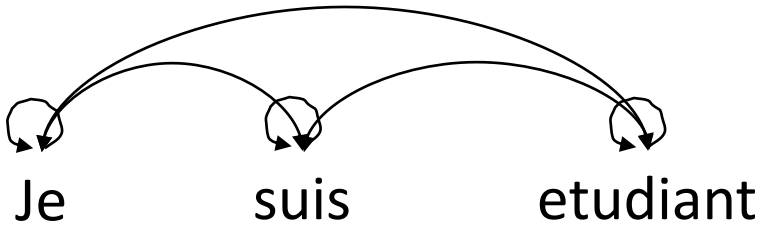

Decoder attention from “am”



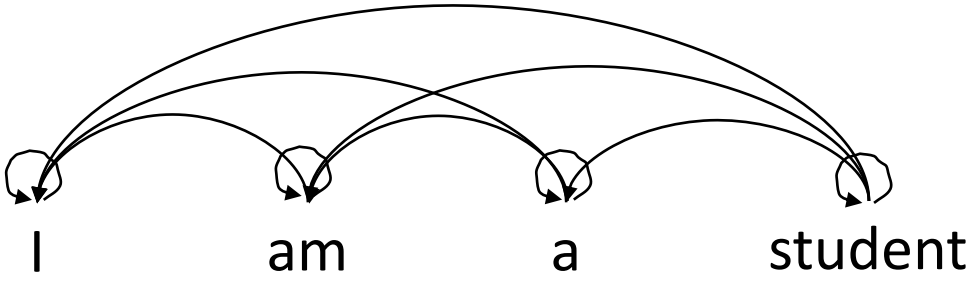
# Decoder Self-Attention

- Encoder
  - Full self-attention
- Decoder
  - Limited (masked) self-attention
  - Can only attend to itself and backwards

Encoder self-attention

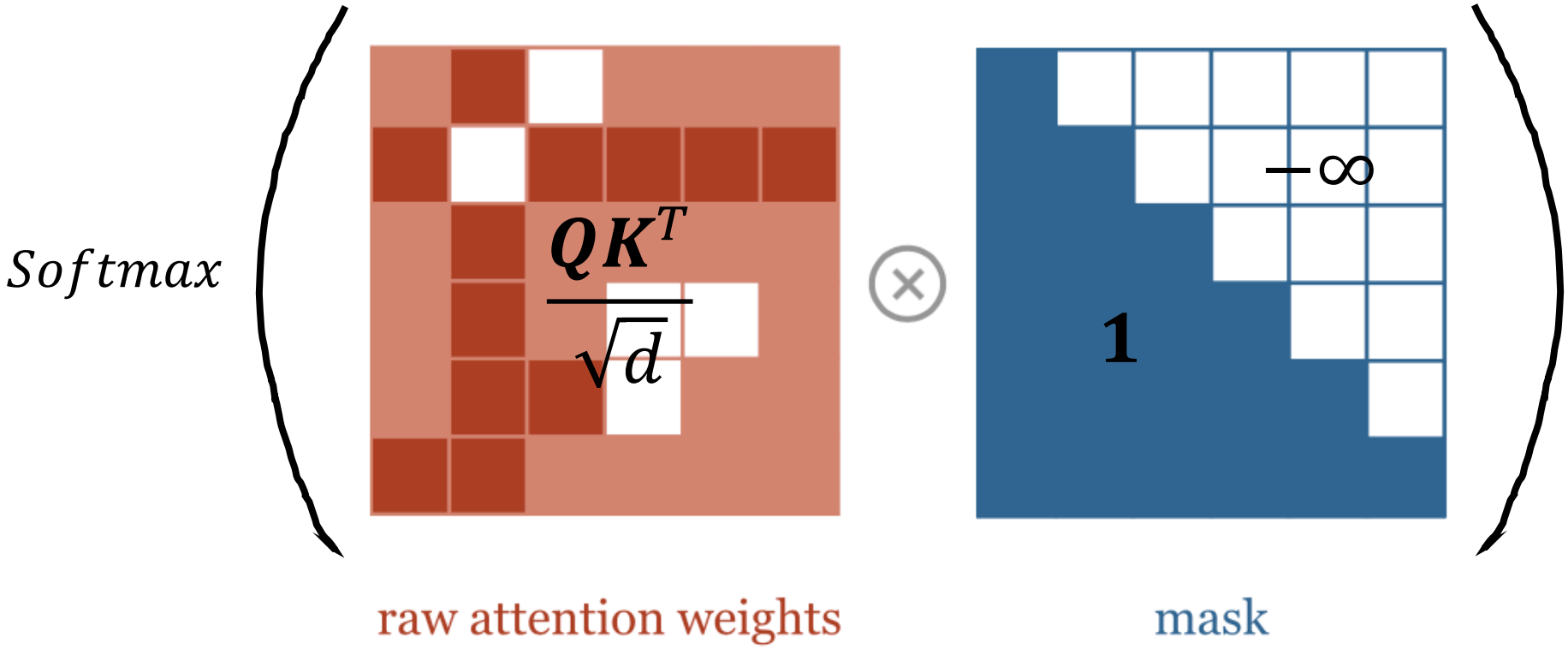



Decoder self-attention



# Masked Self-Attention

- Multiply a triangular matrix to the unnormalized attention.




http://www.peterbloem.nl/blog/transformers

# 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*like |
| 0.2*I + 0.3*like + 0.5*going |
| 0.1*I + 0.2*like + 0.1*going + 0.6*to |
| .. |
| .. |

# AI504: Programming for Artificial Intelligence

## Week 12: Transformer

Edward Choi
Grad School of AI
edwardchoi@kaist.ac.kr