

# Deep Learning Basics

## Lecture 8: Transformer

**최성준** (고려대학교 인공지능학과)

**WARNING:** 본 교육 콘텐츠의 지식재산권은 재단법인 네이버커넥트에 귀속됩니다. **본 콘텐츠를 어떠한 경로로든 외부로 유출 및 수정하는 행위를 엄격히 금합니다.** 다만, 비영리적 교육 및 연구활동에 한정되어 사용할 수 있으나 재단의 허락을 받아야 합니다. 이를 위반하는 경우, 관련 법률에 따라 책임을 질 수 있습니다.

# Sequential Model

- What makes sequential modeling a hard problem to handle?



Original sequence



Trimmed sequence



Omitted sequence

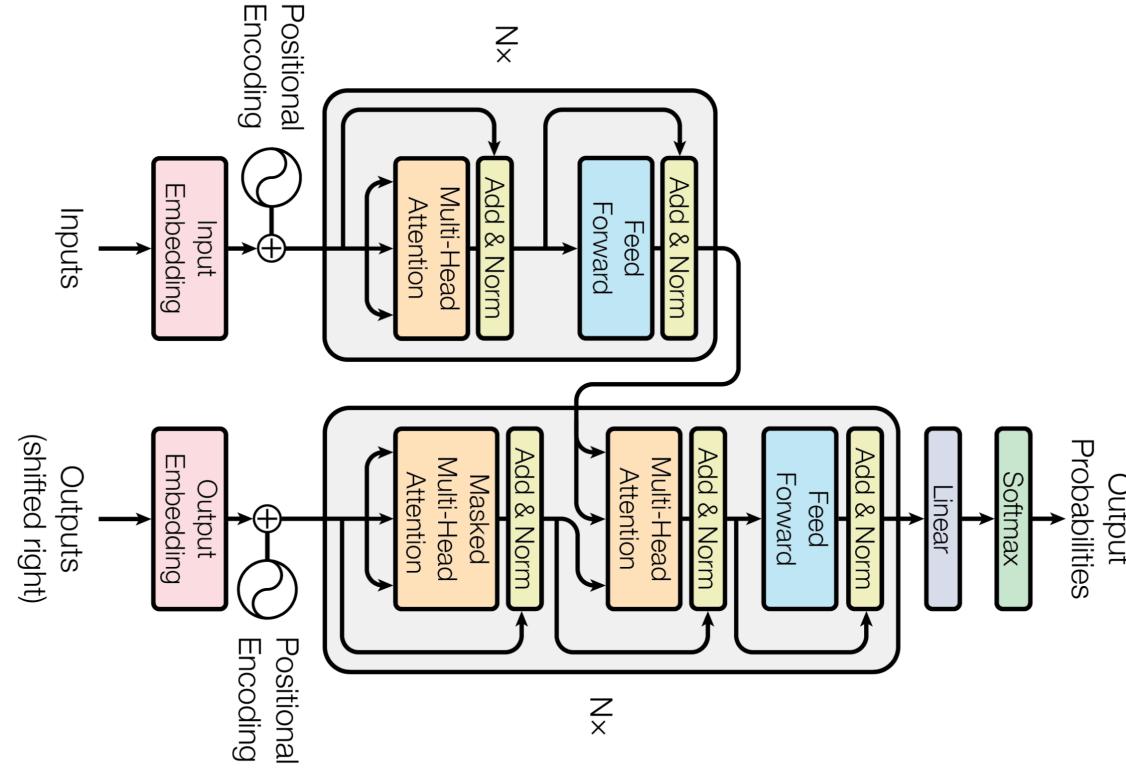


Permuted sequence

# Transformer

Attention is All You Need, NIPS, 2017

# Transformer



- Transformer is the first sequence transduction model based entirely on attention.

<https://arxiv.org/abs/1706.03762>

# Transformer

- From a bird's-eye view, this is what the **Transformer** does for machine translation tasks.



Jay Alammar

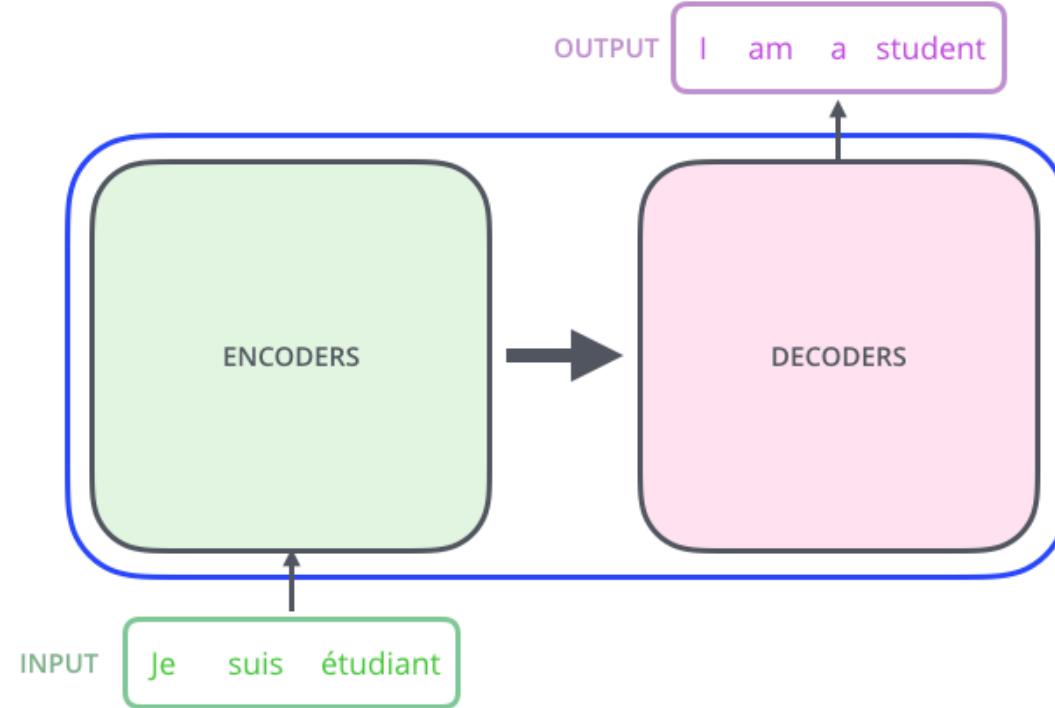
Visualizing machine learning one concept at a time.

@JayAlammar on Twitter. [YouTube Channel](#)

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

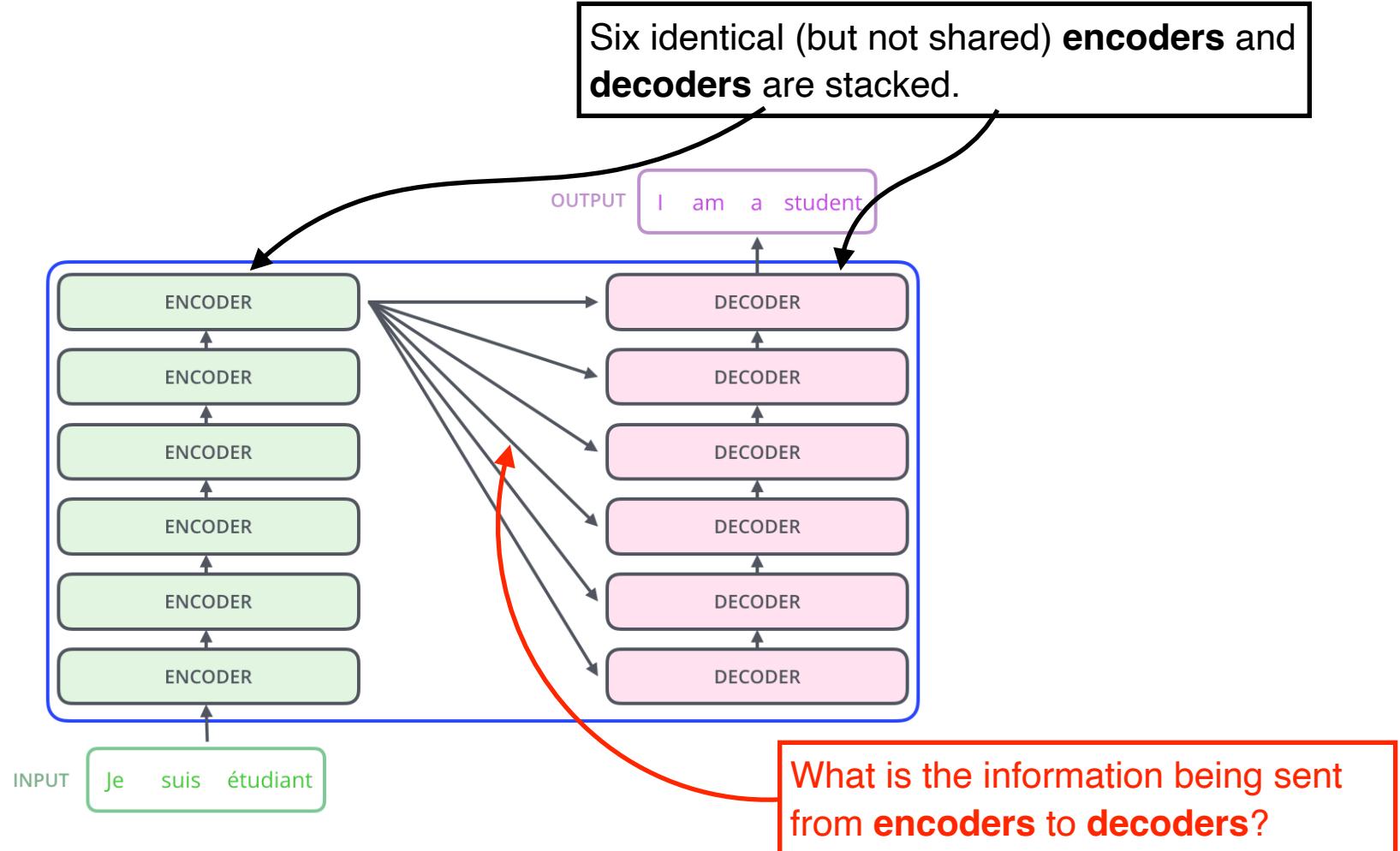
# Transformer

---

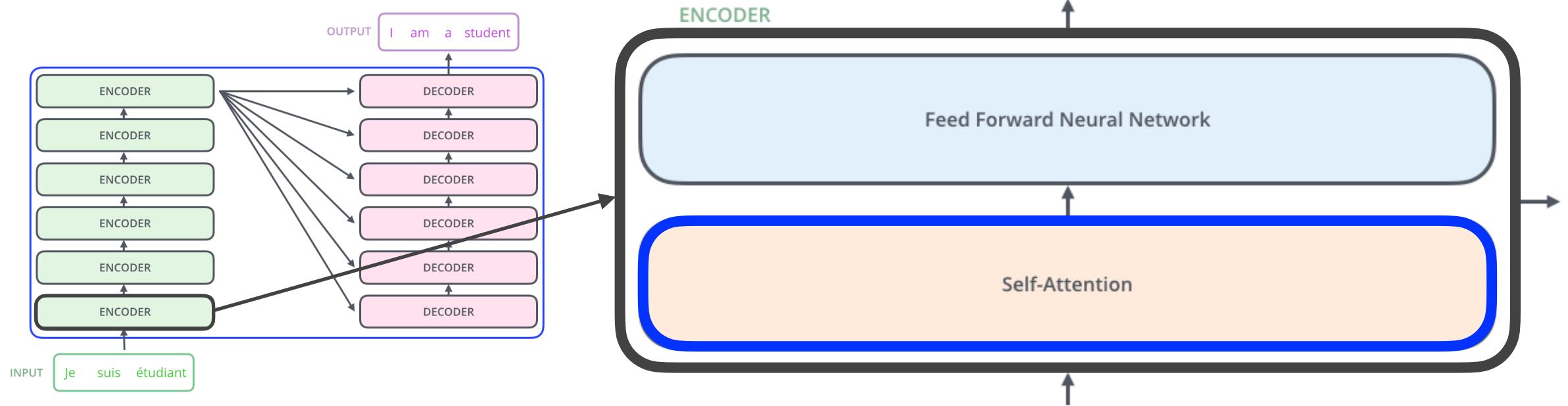


- If we glide down a little bit, this is what the **Transformer** does.

# Transformer



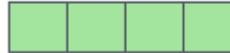
# Transformer



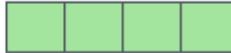
- The **Self-Attention** in both encoder and decoder is the cornerstone of Transformer.

# Transformer

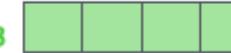
---

$x_1$  

Je

$x_2$  

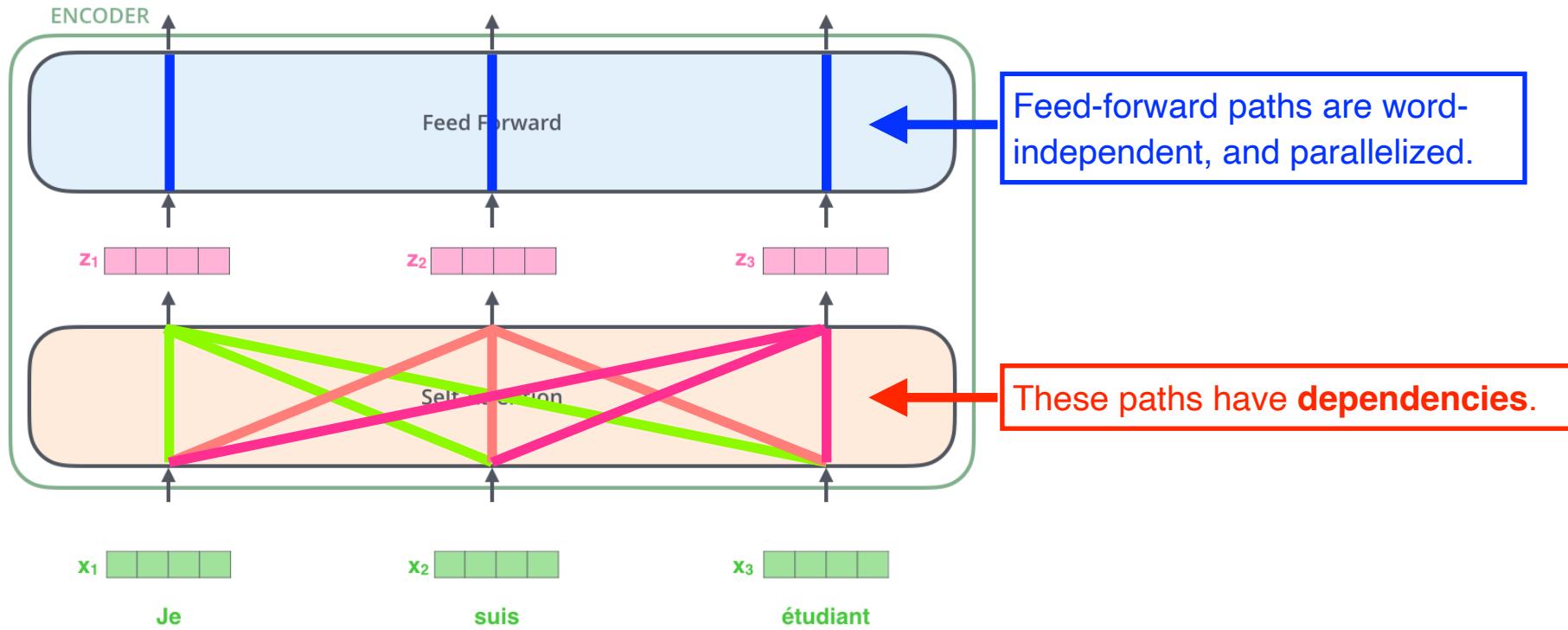
suis

$x_3$  

étudiant

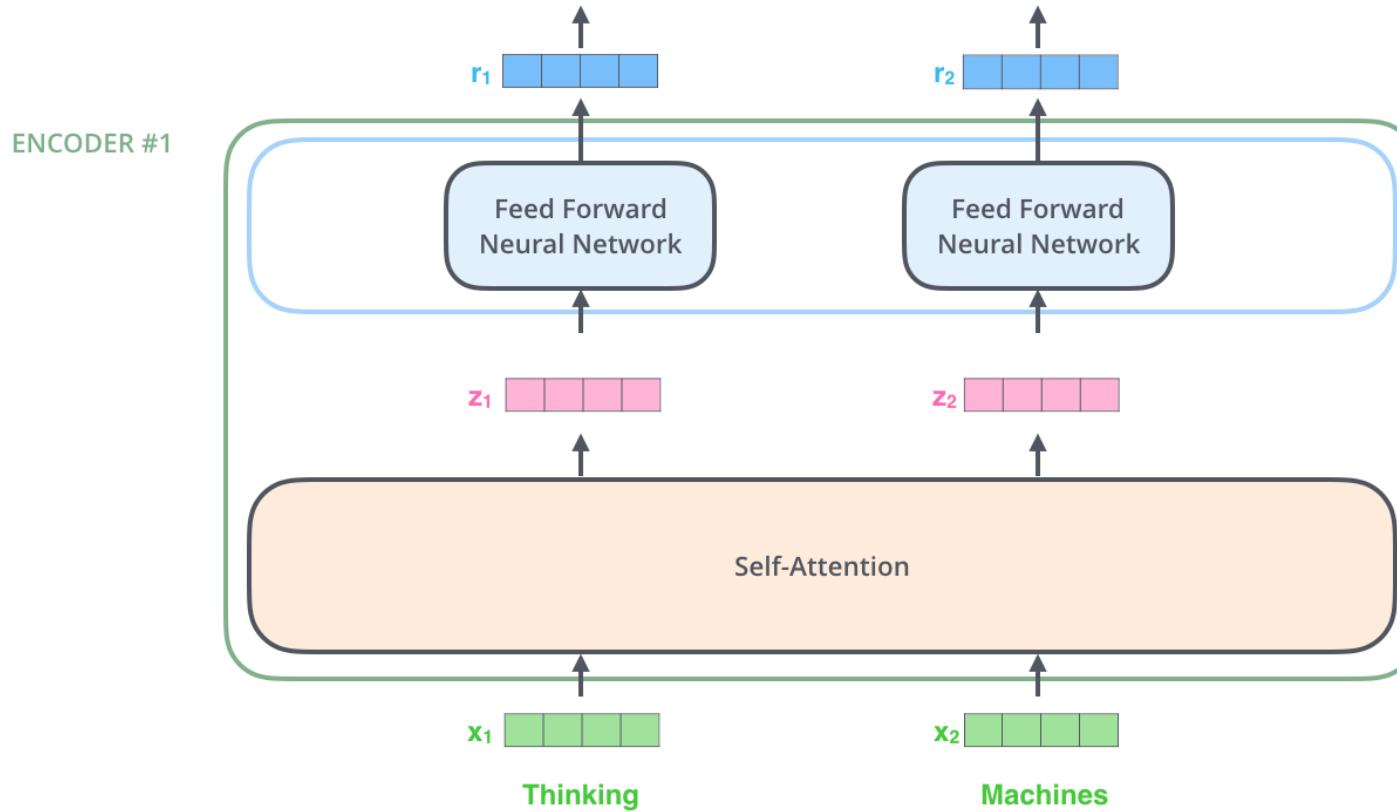
- First, we represent each word with some embedding vectors.

# Transformer



- Then, Transformer encodes each word to feature vectors with **Self-Attention**.

# Transformer

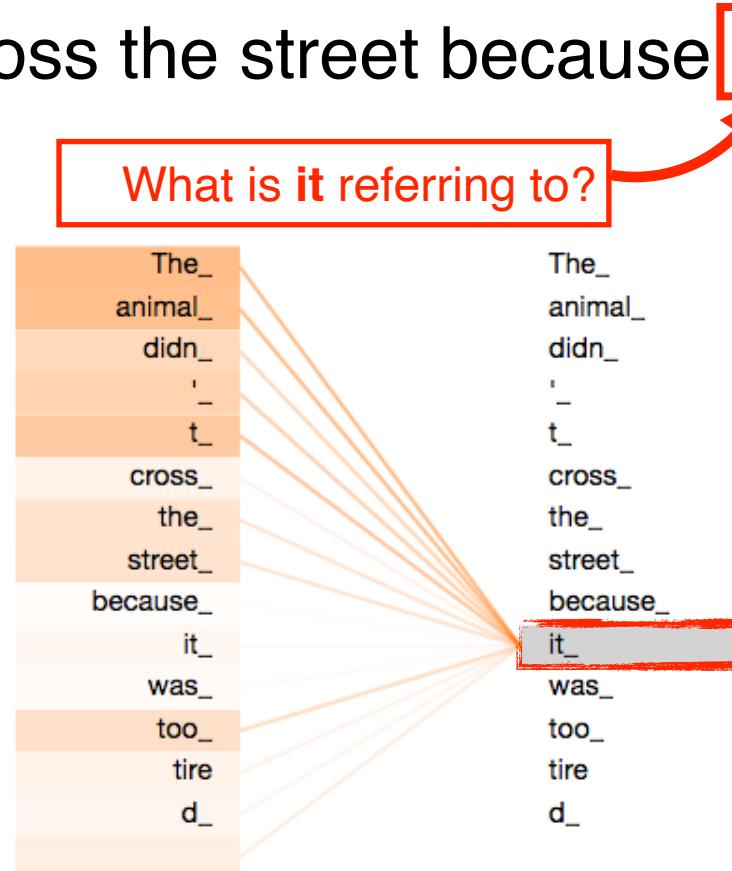


- Suppose we are encoding two words:
  - Thinking and Machines.**

# Transformer

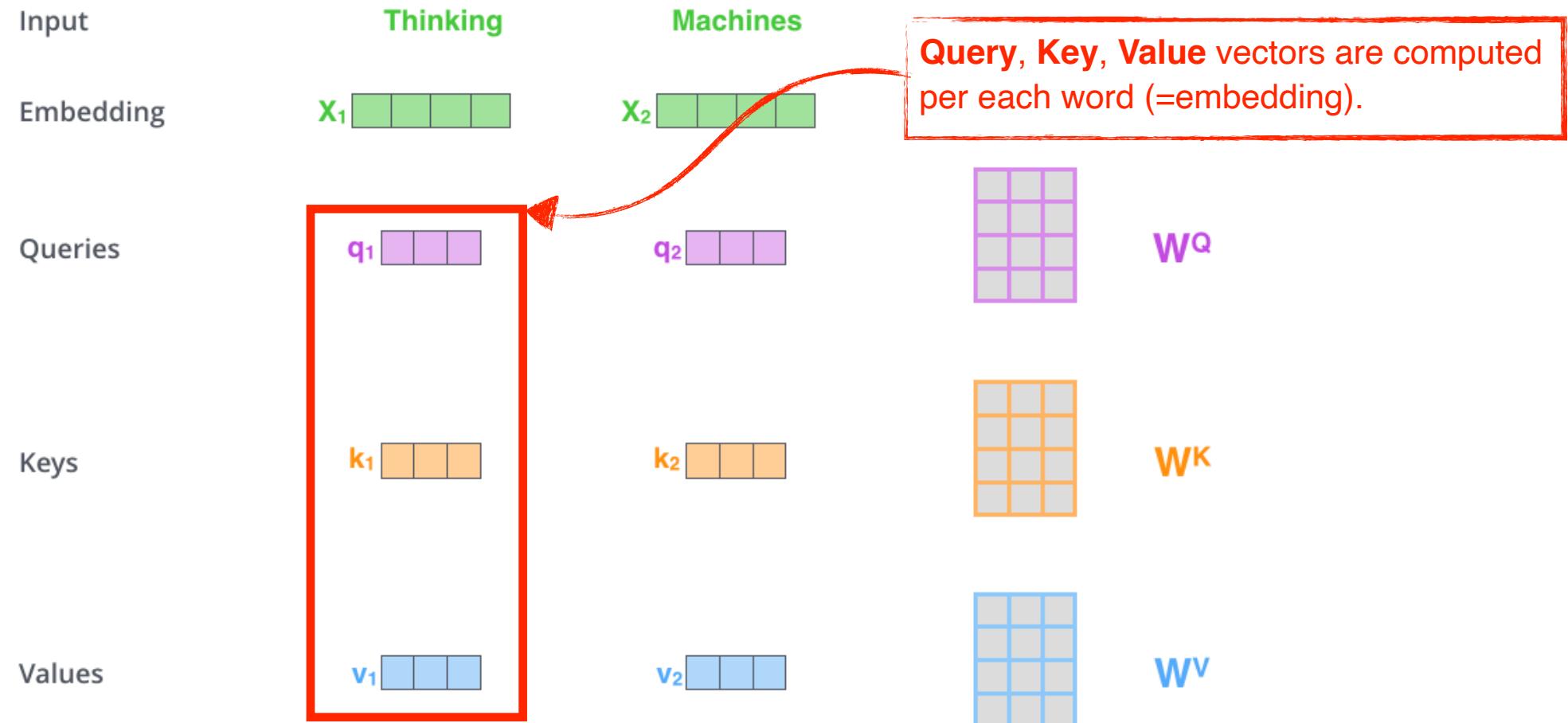
- **Self-Attention at a high level**

- The animal didn't cross the street because **it** was too tired.

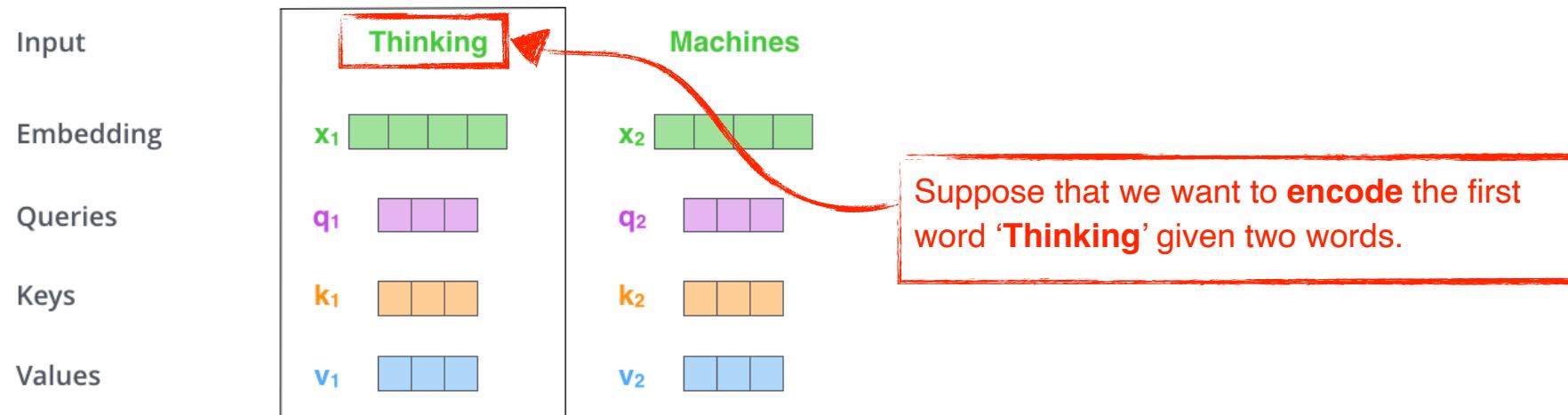


<http://jalammar.github.io/illustrated-transformer/>

# Transformer

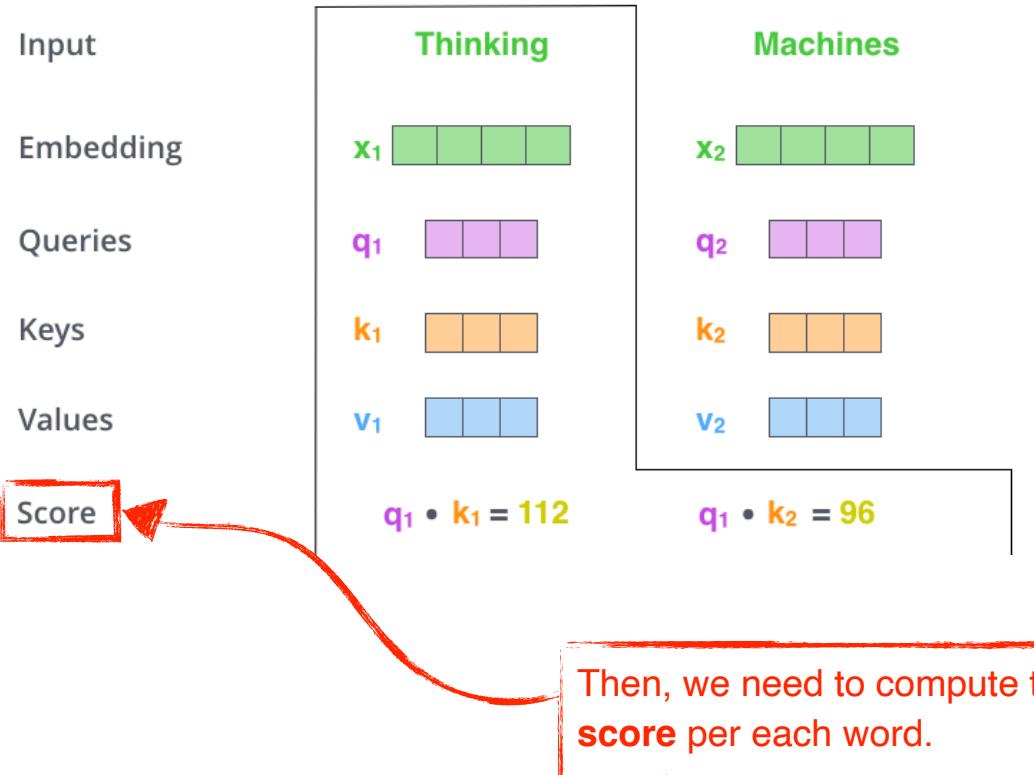


# Transformer



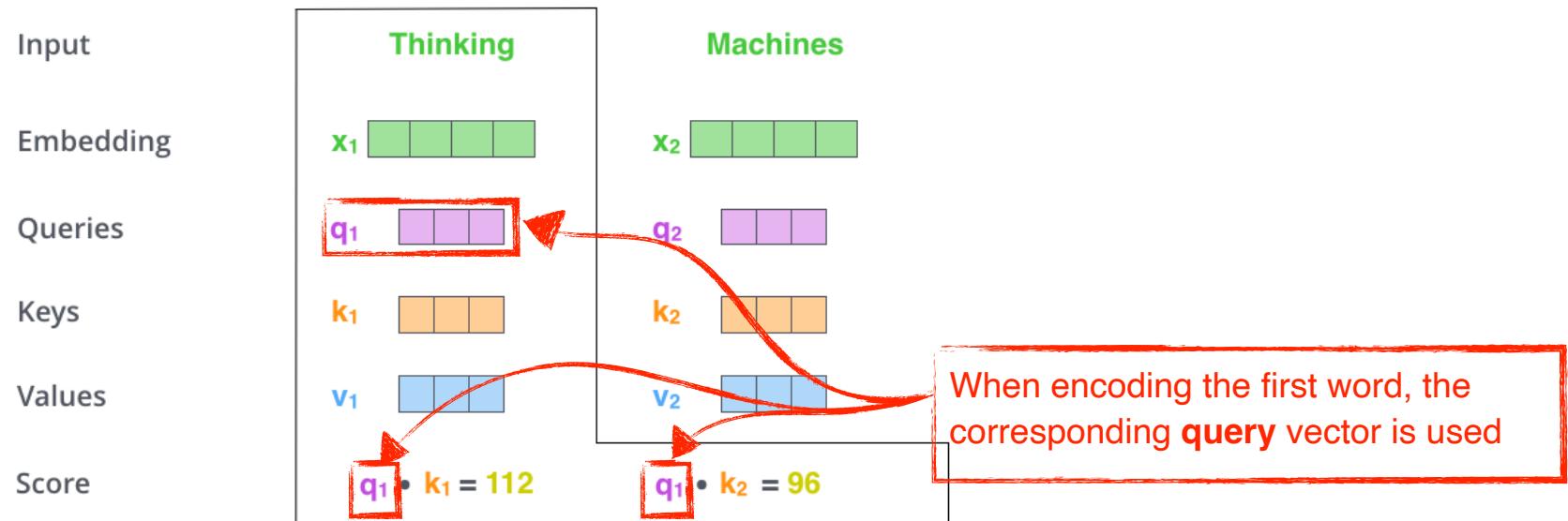
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



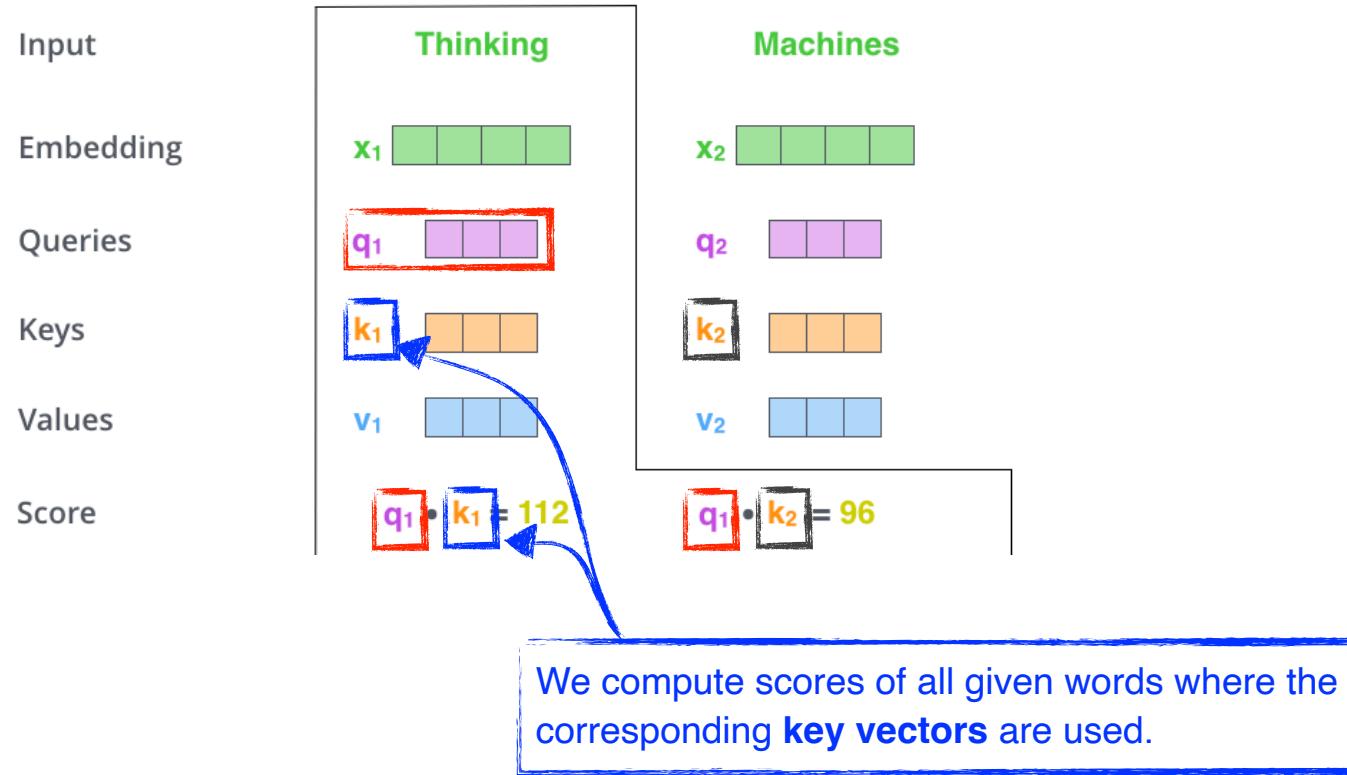
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



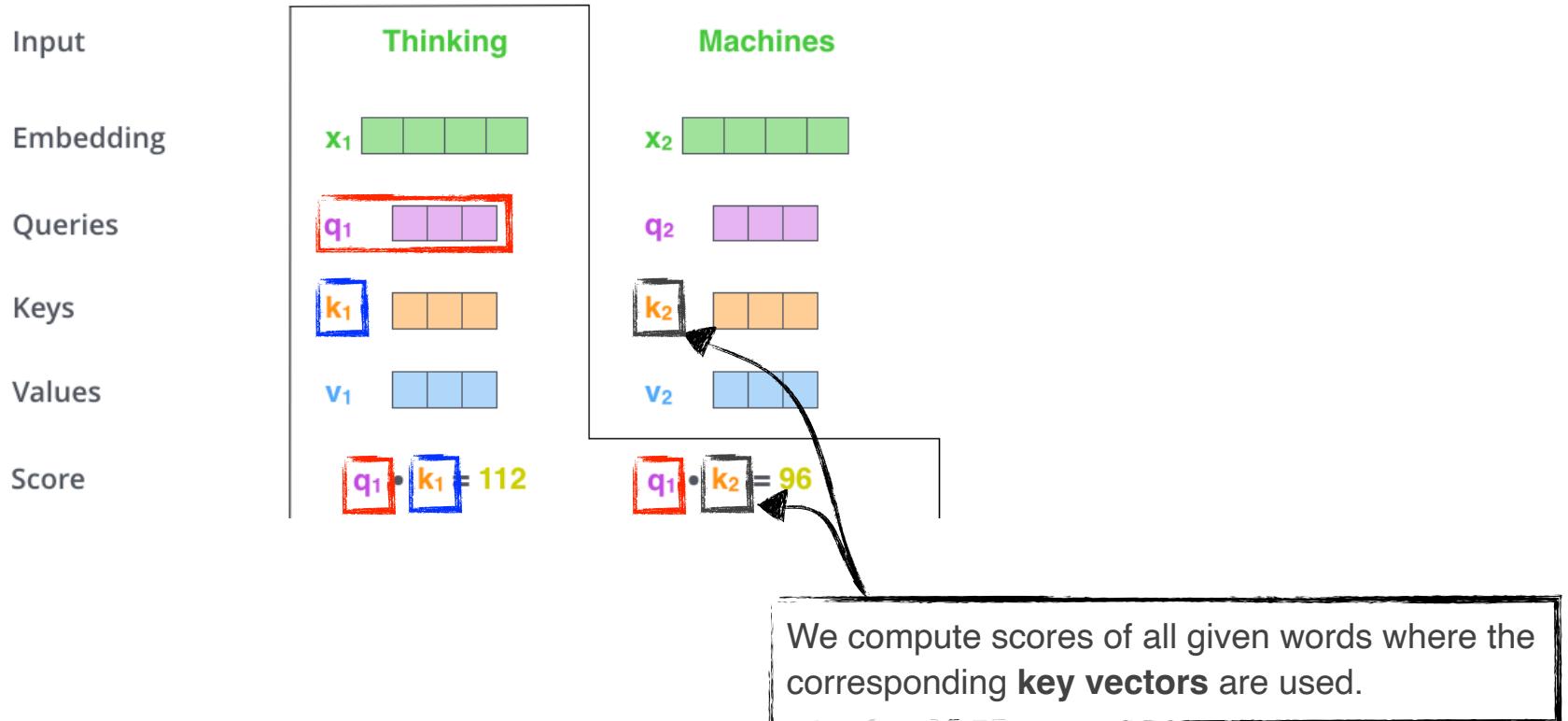
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



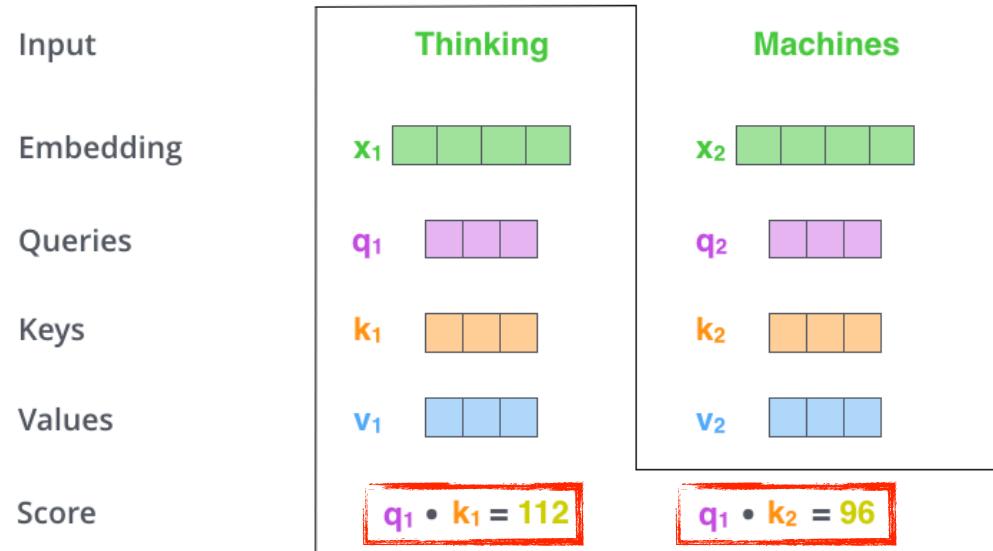
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



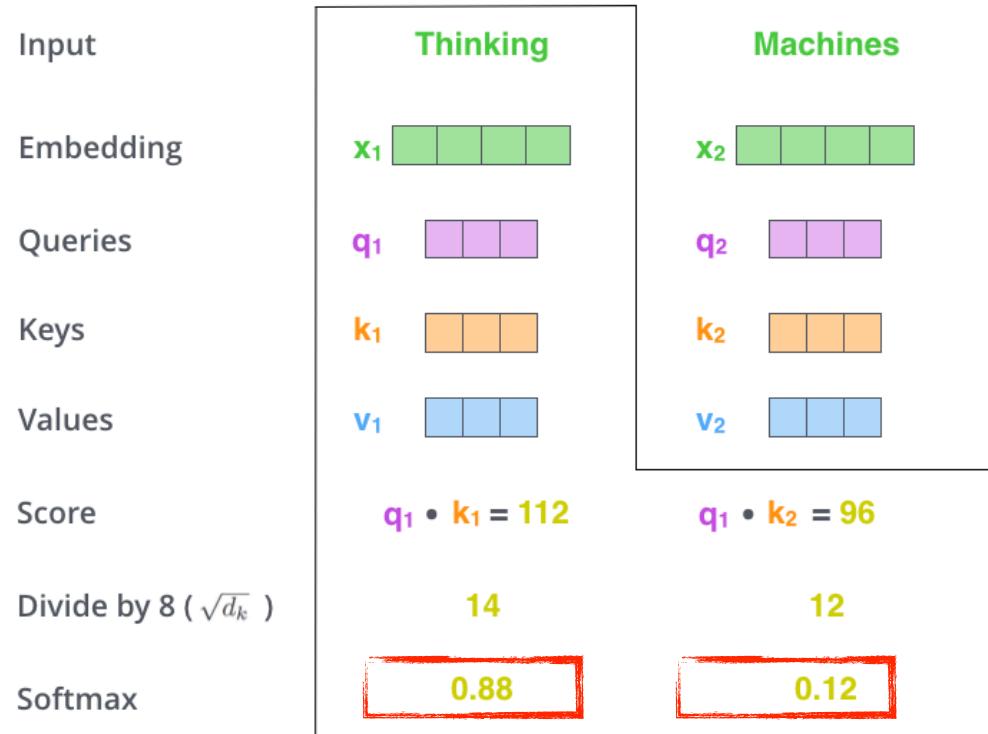
- Suppose we are encoding the first word: '**Thinking**' given '**Thinking**' and '**Machines**'.

# Transformer



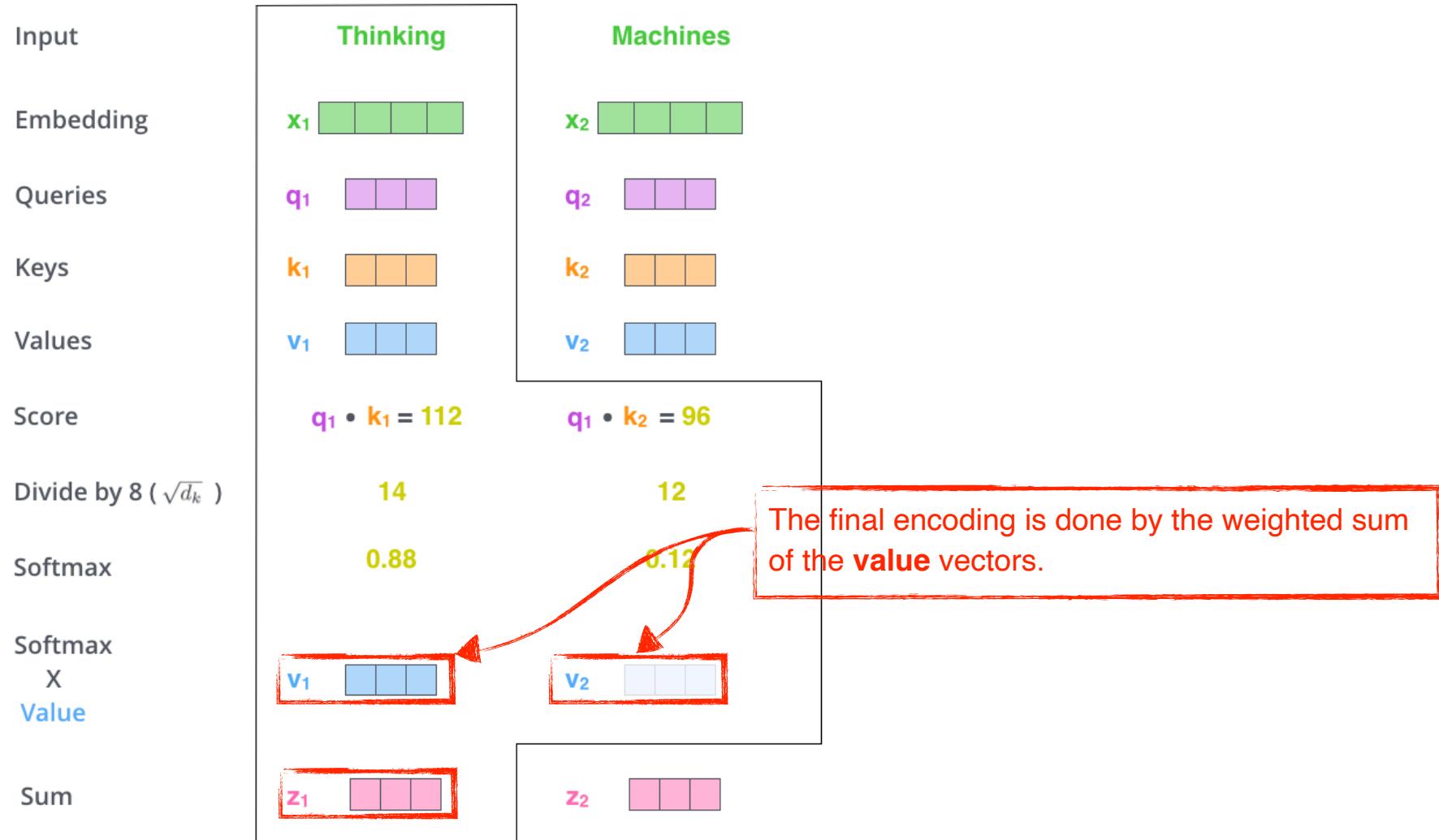
- We compute the **scores** of each word with respect to 'Thinking'.

# Transformer



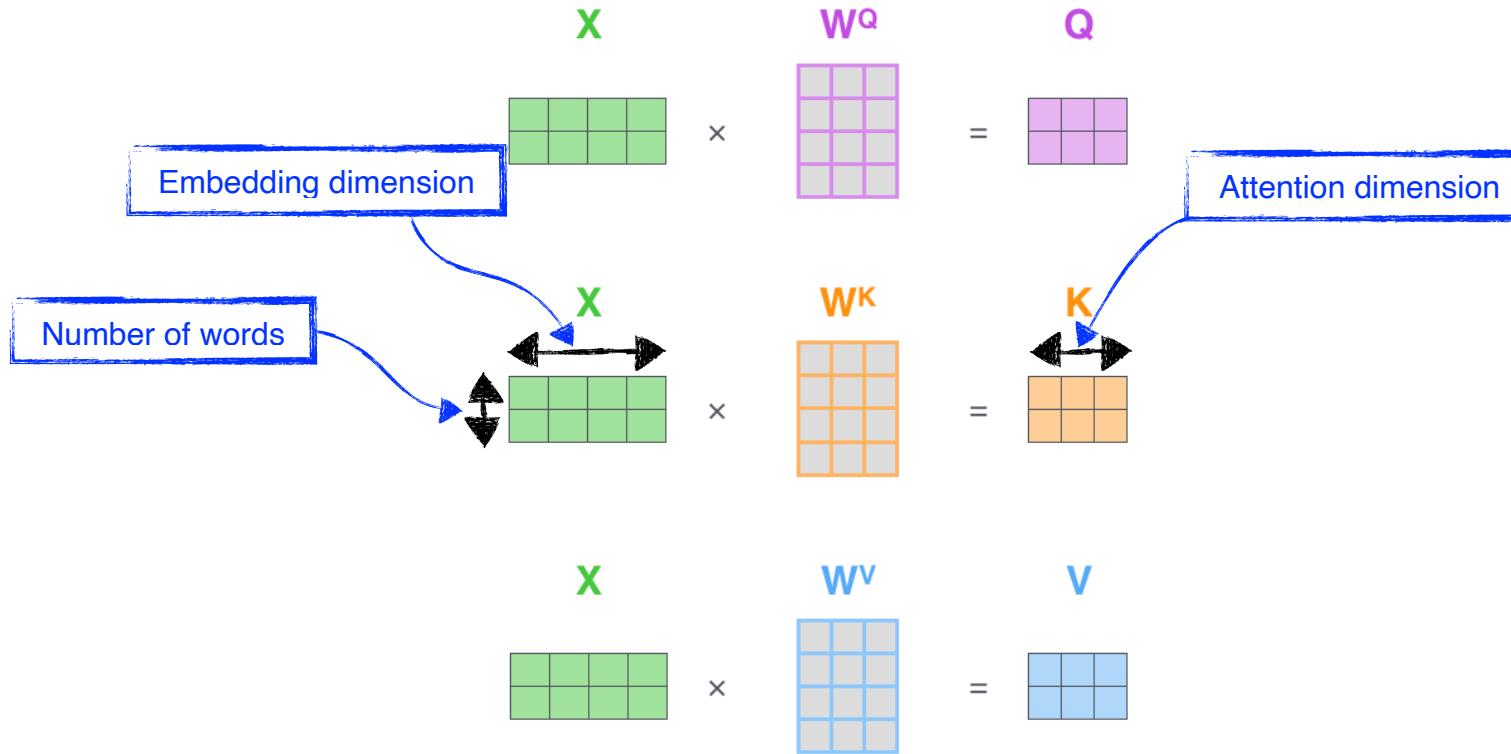
- Then, we compute the **attention weights** by scaling followed by softmax.

# Transformer



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

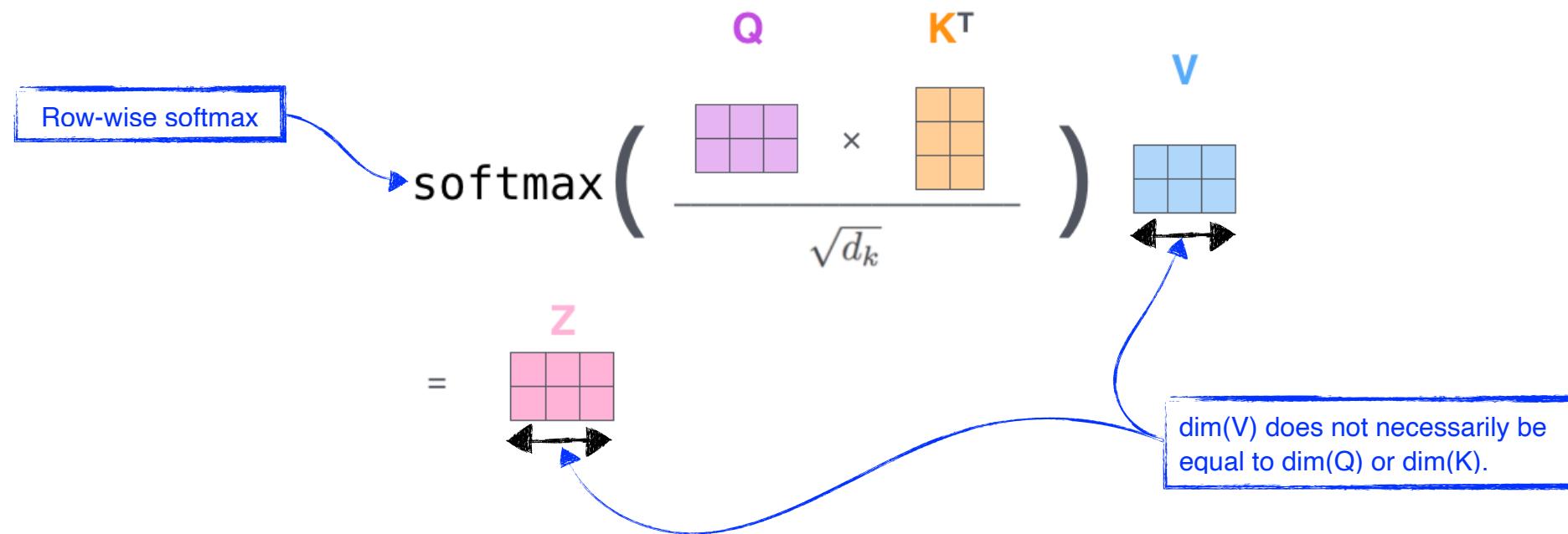
# Transformer



- Calculating Q, K, and V from X in a matrix form.

# Transformer

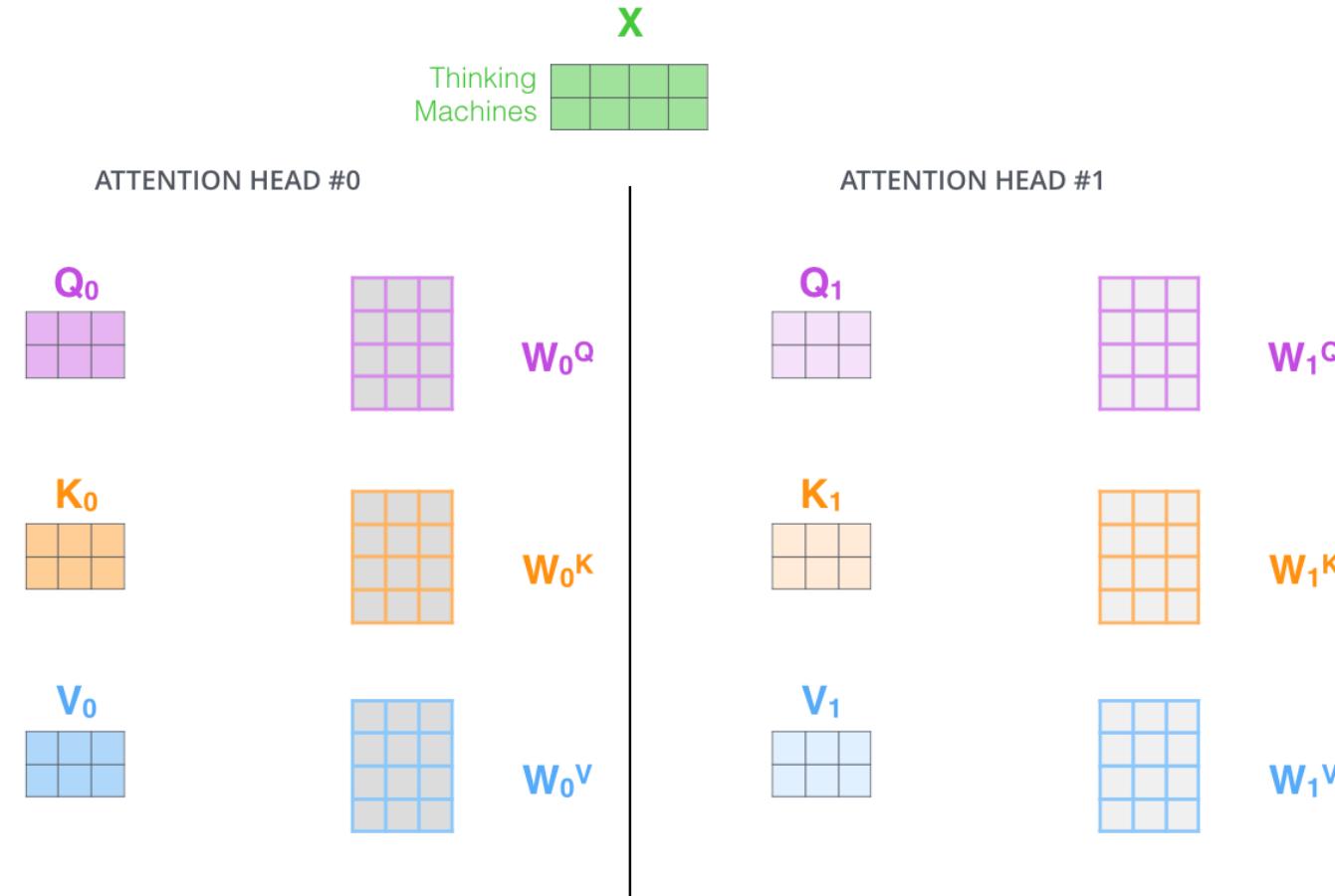
---



- Calculating Q, K, and V from X in a matrix form.

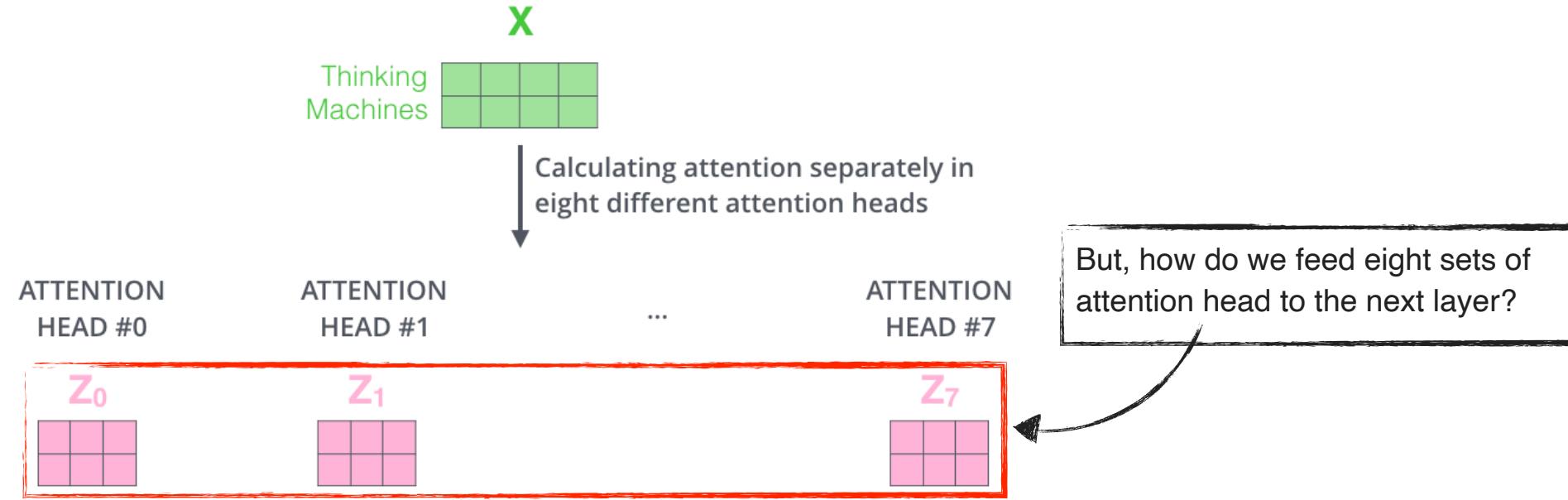
# Transformer

---



- Multi-headed attention (MHA) allows Transformer to focus on different positions.

# Transformer



- If eight heads are used, we end up getting eight different sets of encoded vectors (**attention heads**).

# Transformer

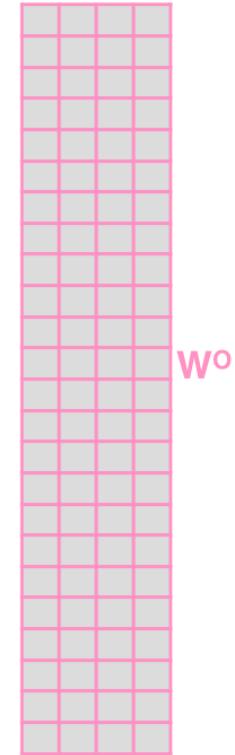
---

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

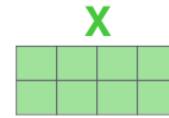
$$= \begin{matrix} z \\ \hline \end{matrix}$$

- We simply pass them through additional (learnable) linear map.

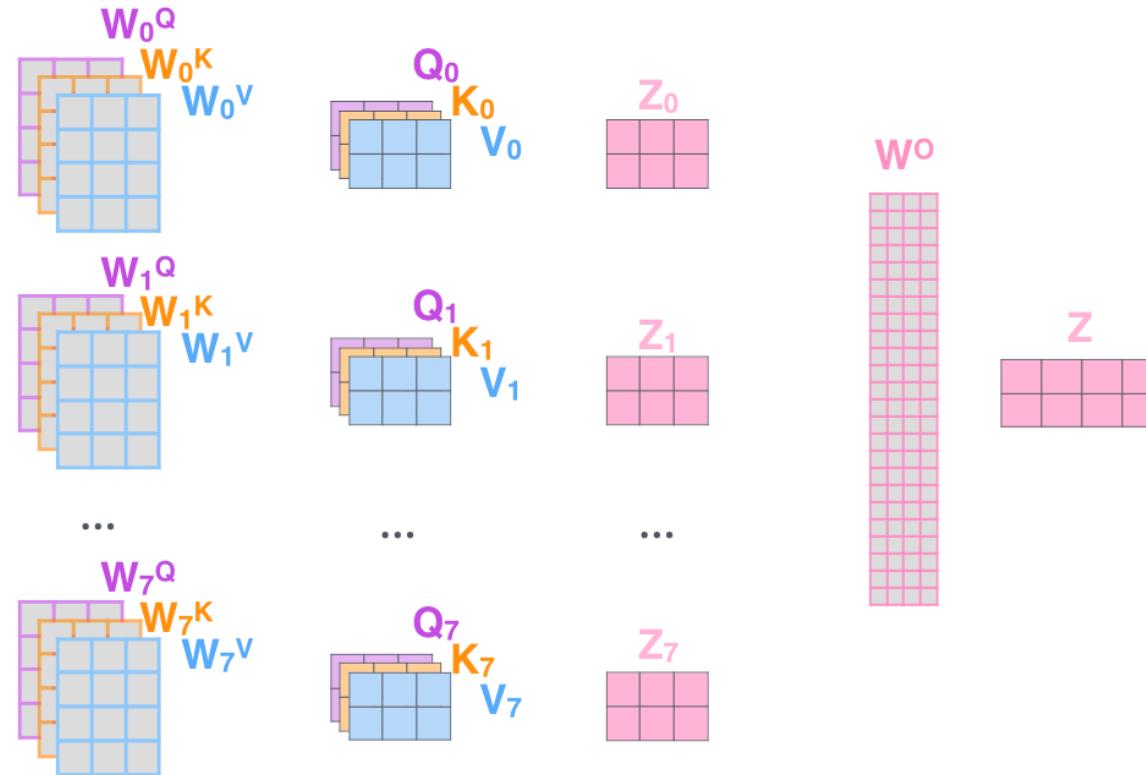
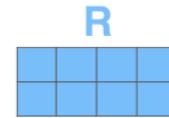
# Transformer

- 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

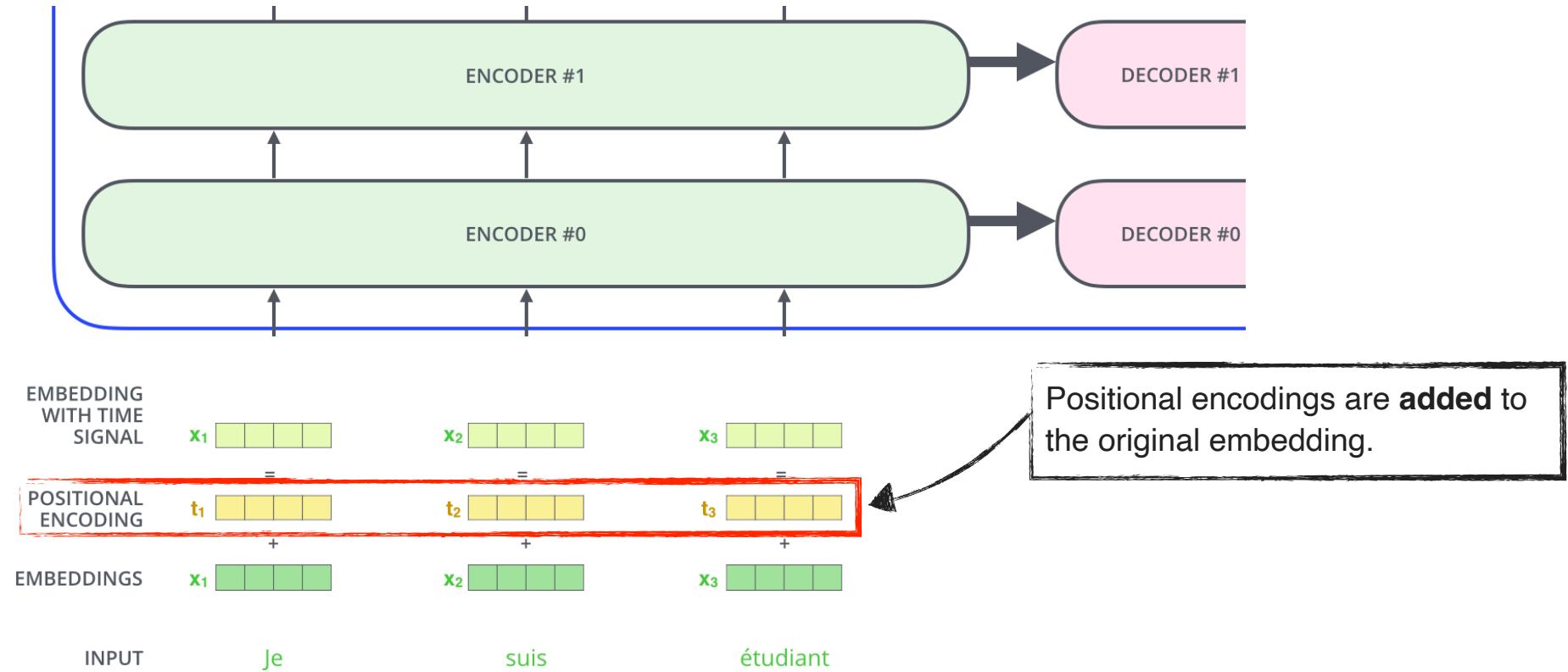
Thinking  
Machines



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



# Transformer



- Why do we need positional encoding?

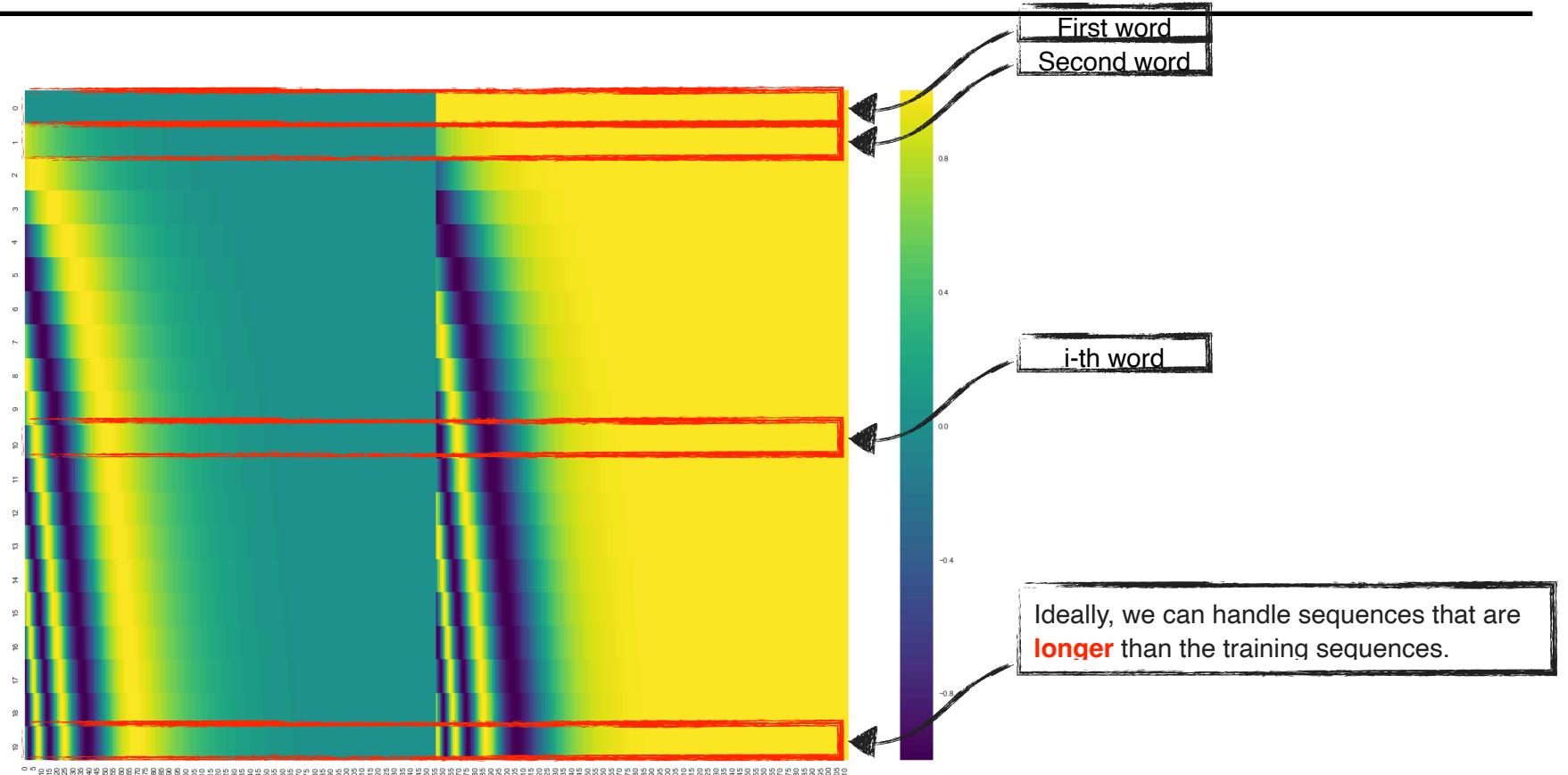
# Transformer

---



- This is the case for 4-dimensional encoding.

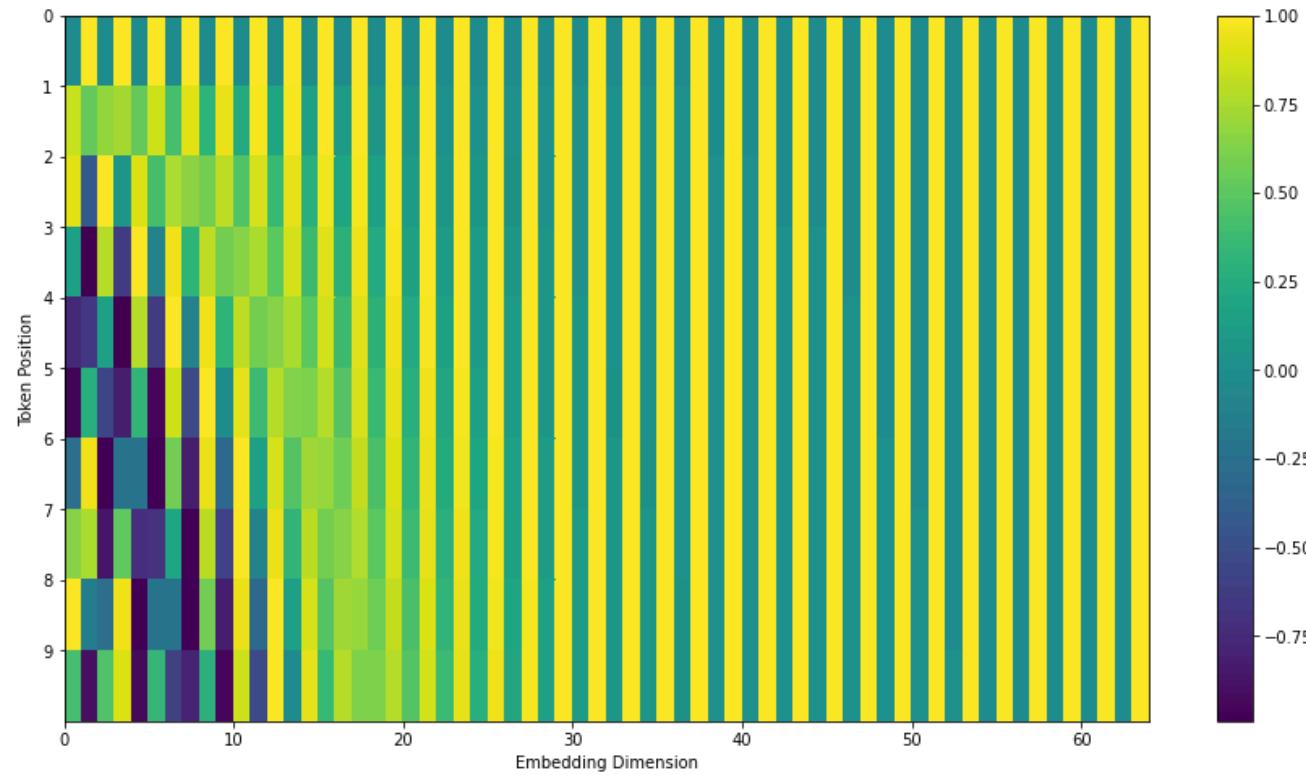
# Transformer



- This is the case for 512-dimensional encoding.

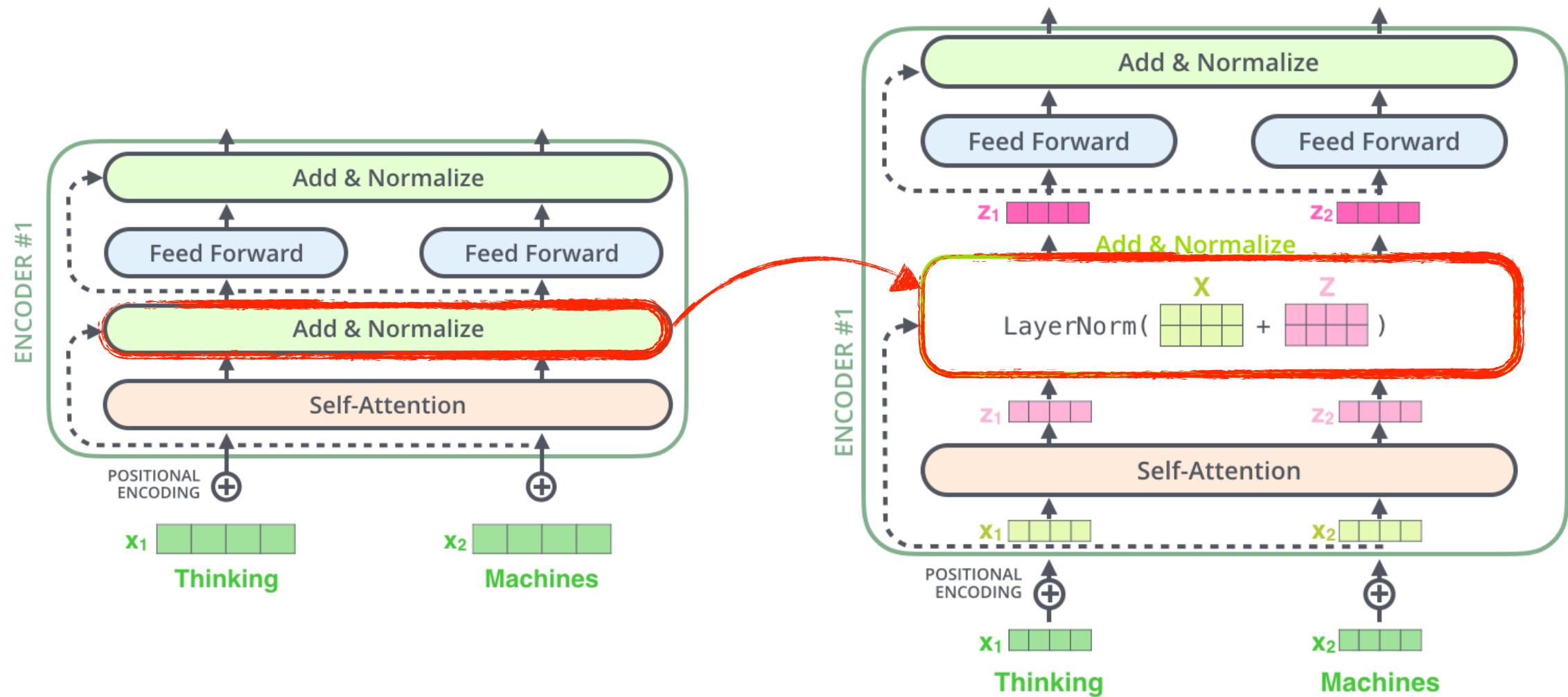
# Transformer

---



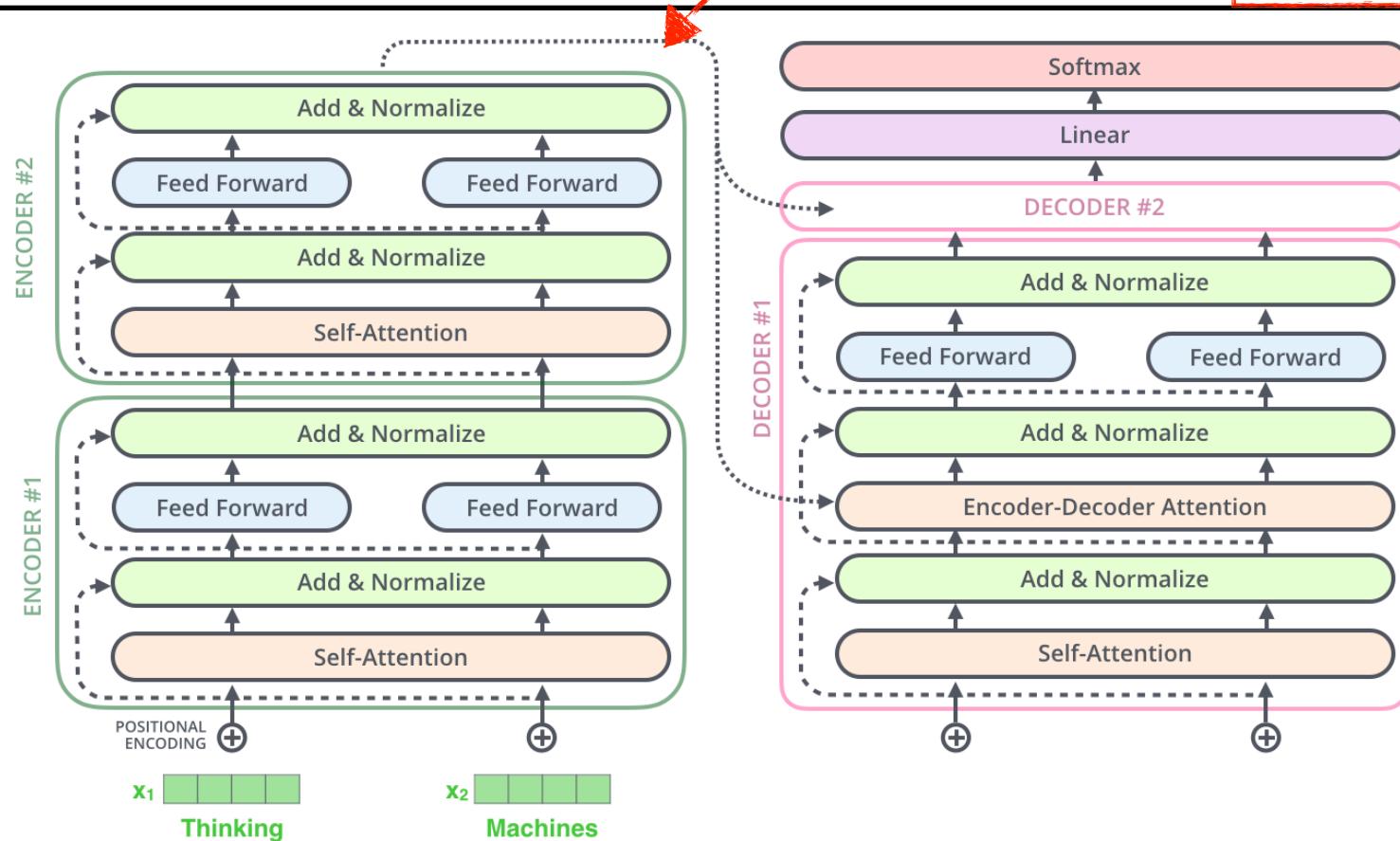
- Recent (July, 2020) update on positional encoding.

# Transformer



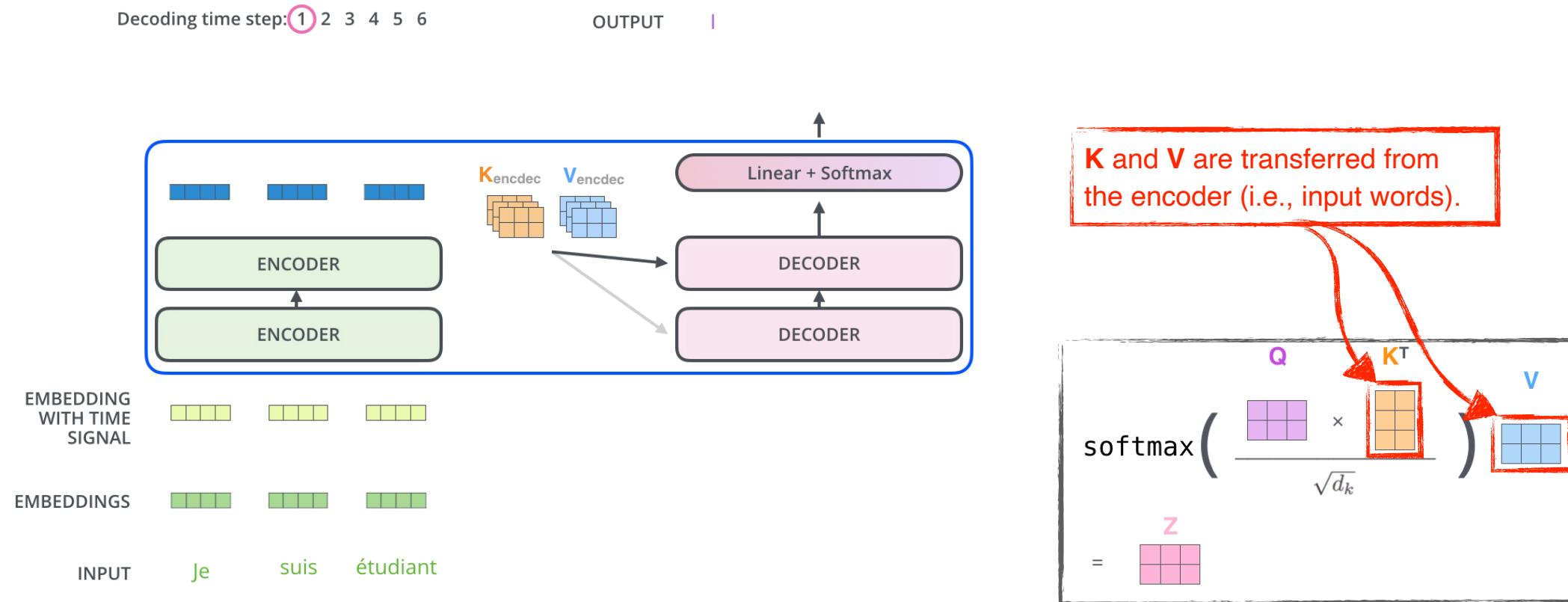
# Transformer

What is the information being sent from encoders to decoders?



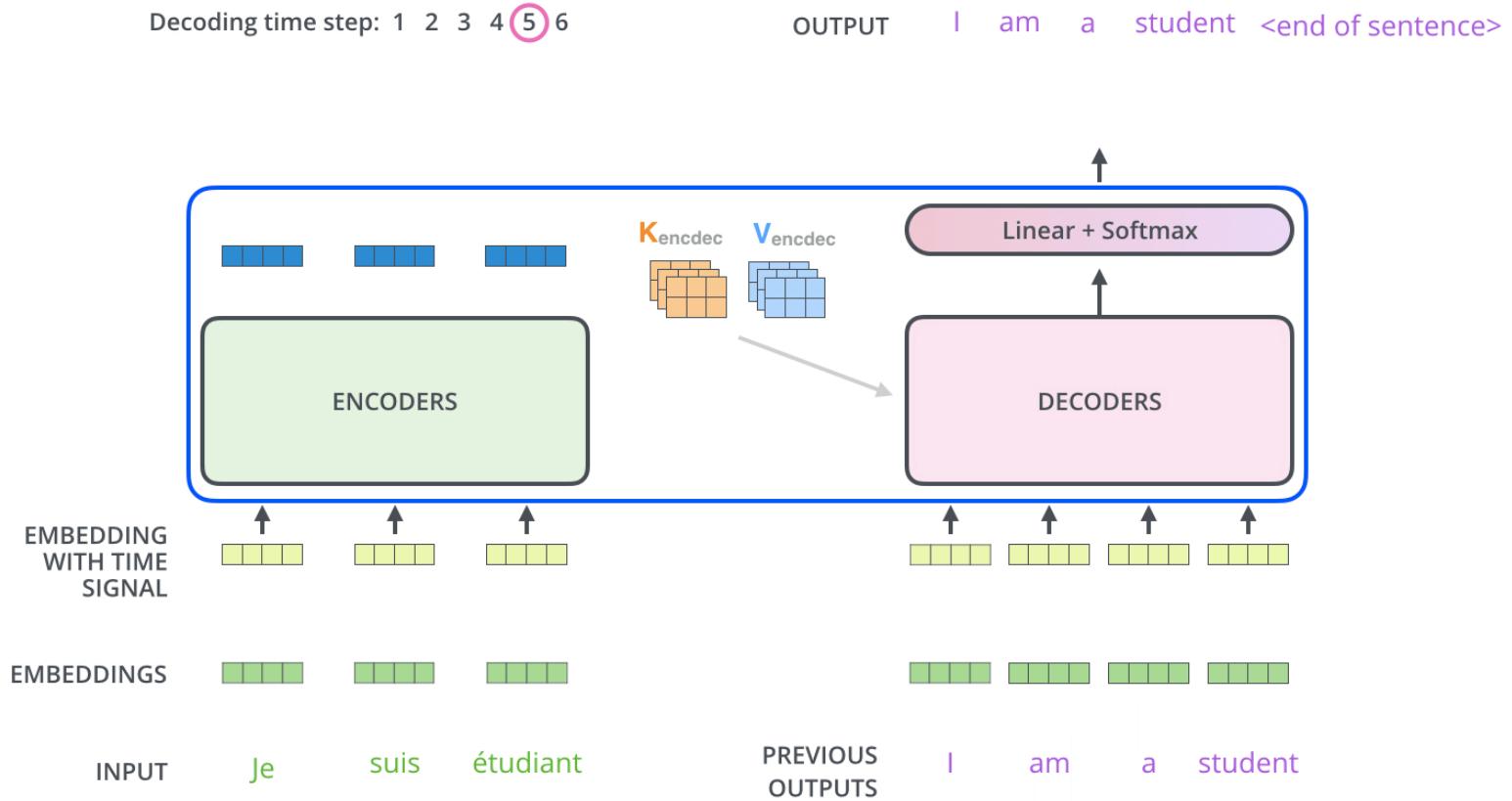
- Now, let's take a look at the **decoder** side.

# Transformer



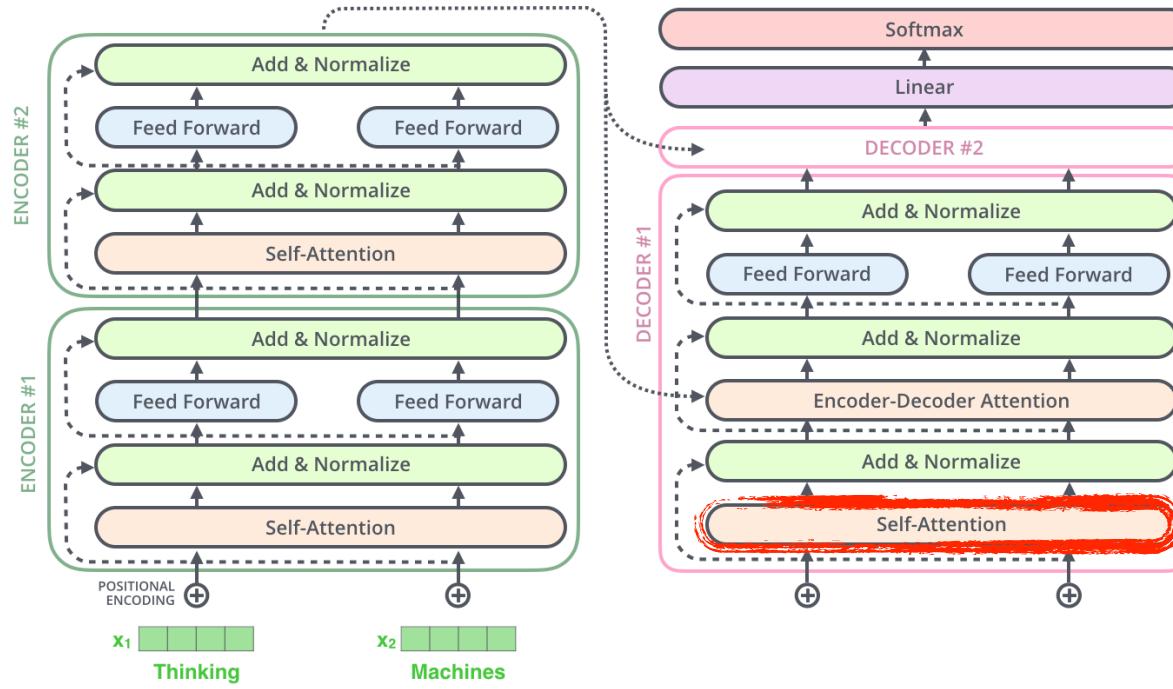
- Transformer transfers **key (K)** and **value (V)** of the topmost encoder to the decoder.

# Transformer



- The output sequence is generated in an autoregressive manner.

# Transformer



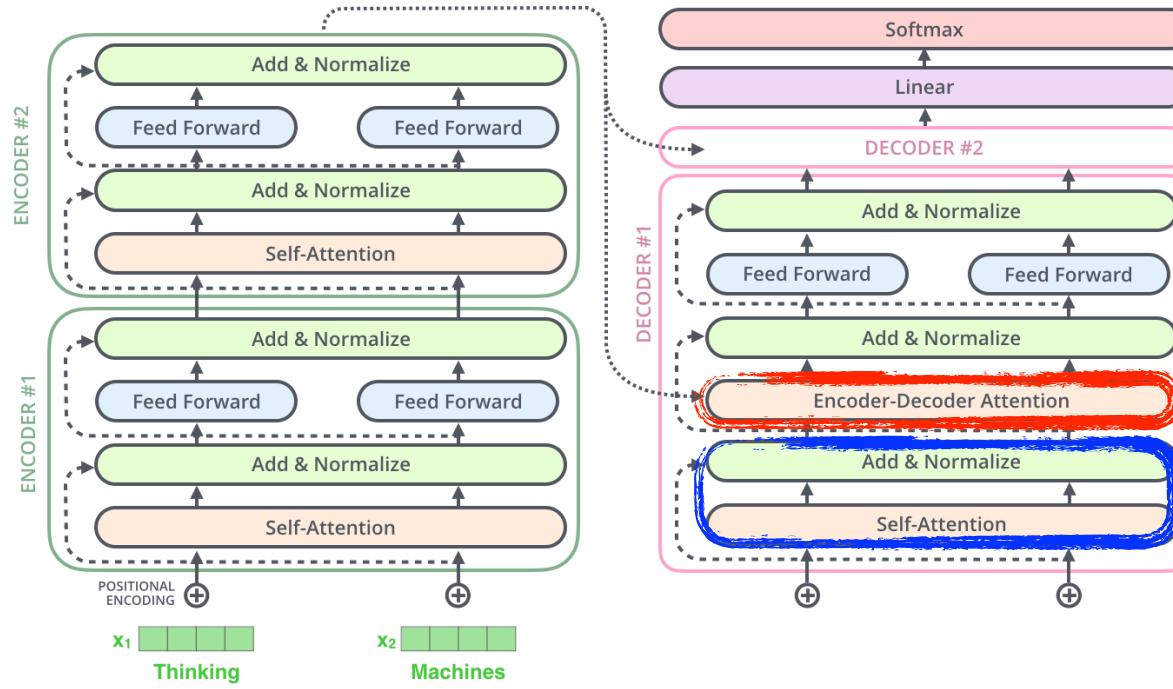
$$\text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) = Z$$

Diagram illustrating the computation of attention weights. The query matrix  $Q$  (purple) and key matrix  $K^T$  (orange) are multiplied, and the result is scaled by  $\sqrt{d_k}$  before being passed through a softmax function to produce the attention distribution  $Z$ .

- In the **decoder**, the **self-attention layer** is only allowed to attend to earlier positions in the output sequence which is done by masking future positions before the softmax step.

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

# Transformer



$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) = \mathbf{Z}$$

A detailed formula for multi-headed attention:

$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) = \mathbf{Z}$$

Where  $\mathbf{Q}$  and  $\mathbf{K}^T$  are matrices representing Queries and Keys respectively, and  $d_k$  is the dimension of the key.

- The “Encoder-Decoder Attention” layer works just like multi-headed self-attention, except it creates its **Queries** matrix from **the layer below it**, and takes the **Keys** and **Values** from the encoder stack.

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

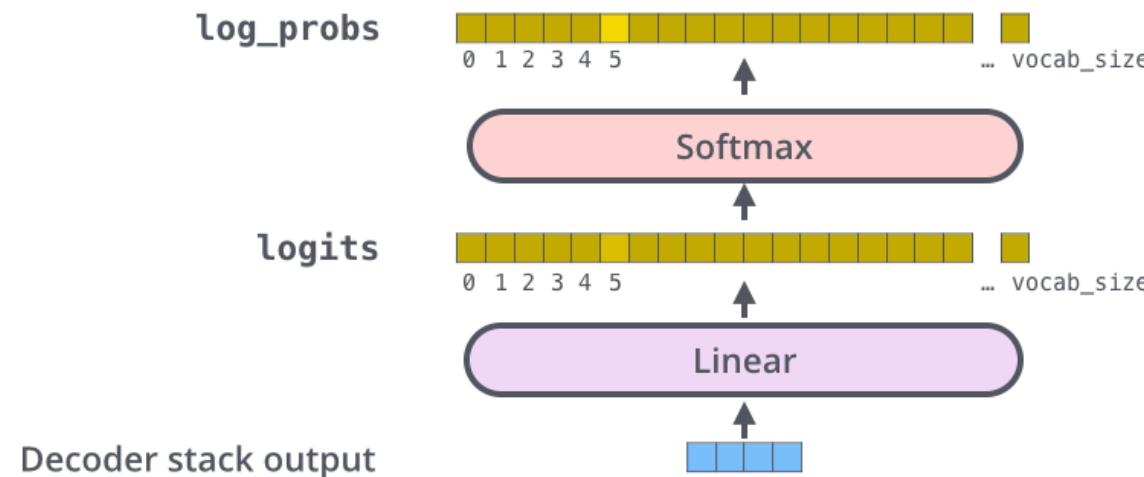
# Transformer

Which word in our vocabulary  
is associated with this index?

am

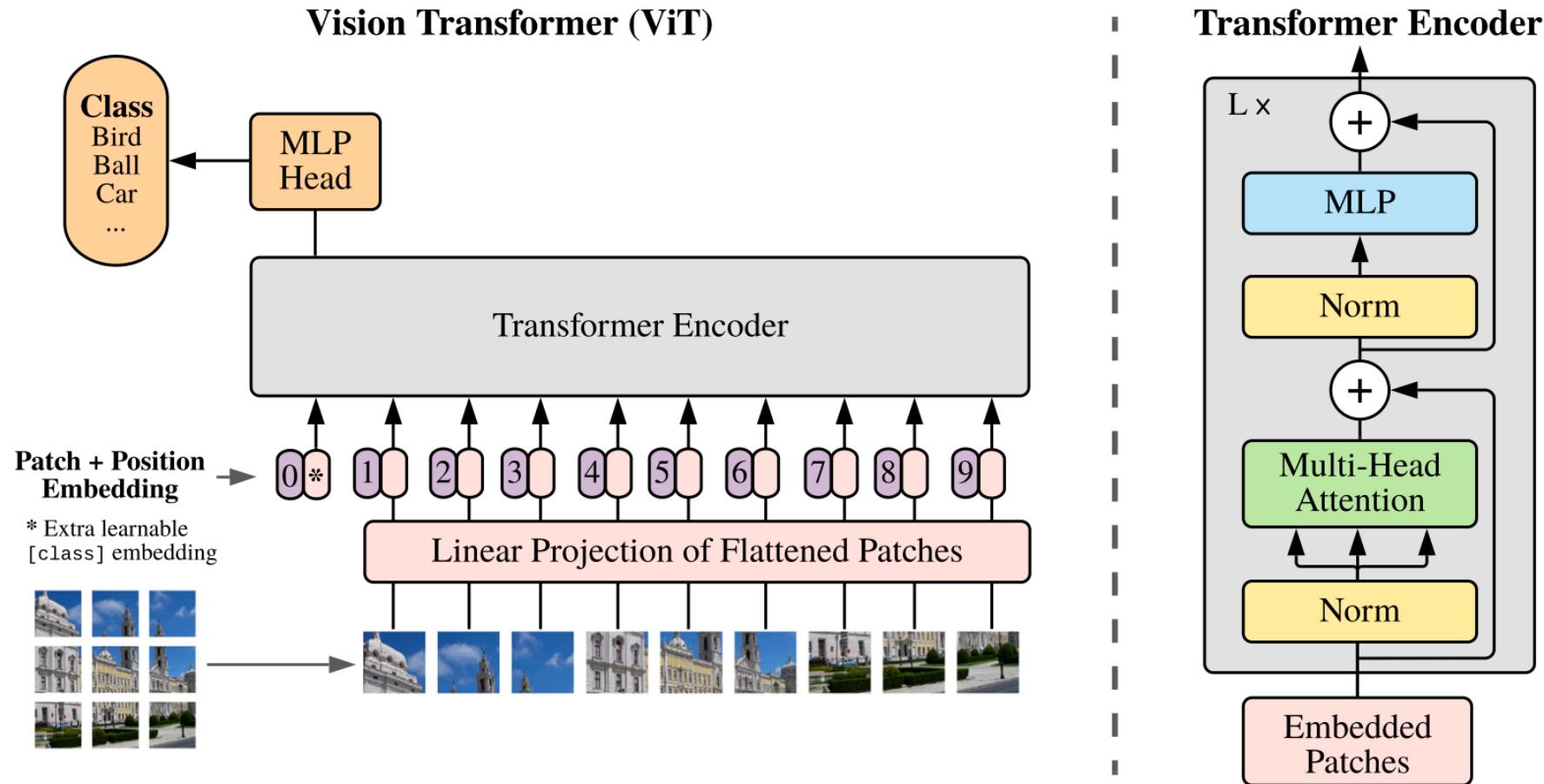
Get the index of the cell  
with the highest value  
(**argmax**)

5



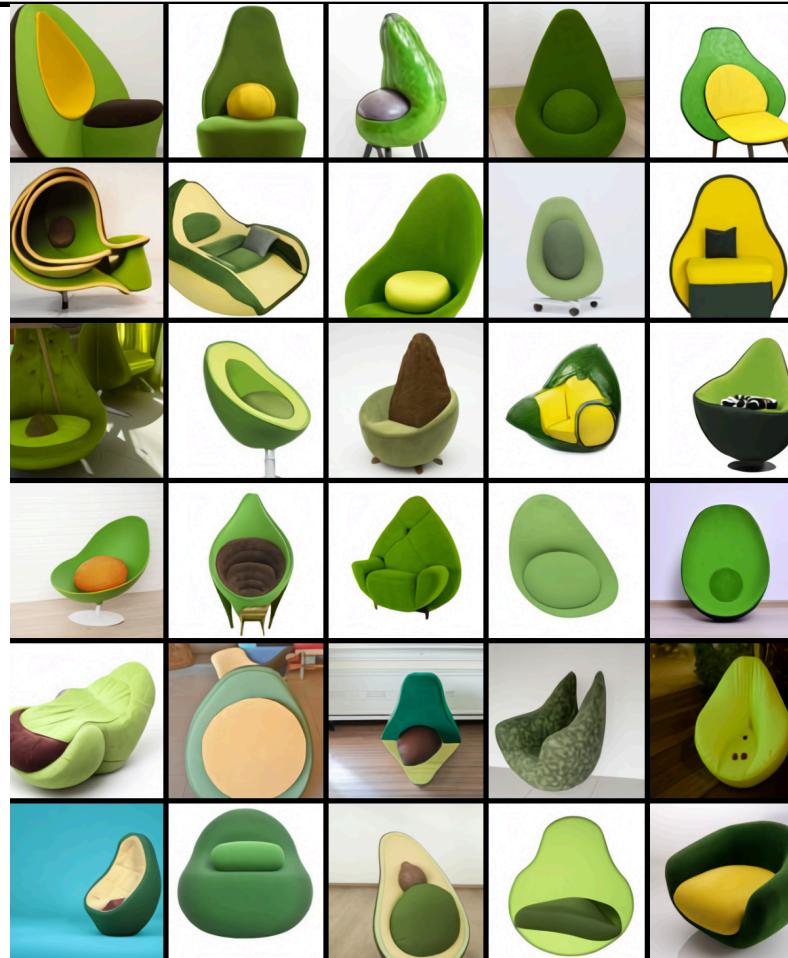
- The final layer converts the stack of decoder outputs to the distribution over words.

# Vision Transformer



# DALL-E

---



An armchair in the shape of an avocado.

<https://openai.com/blog/dall-e/>

# Thank you for listening

---