Neural Attention & Transformers

Evangelos Kalogerakis

Attention vs Convolution

Convolution has been extremely popular in 2D/3D vision:

- ability to exploit local dependencies in the input data
- highly parallelizable / efficient to compute on GPUs

Attention vs Convolution

Convolution has been extremely popular in 2D/3D vision:

- ability to exploit local dependencies in the input data
- highly parallelizable / efficient to compute on GPUs

Capturing long-range interactions between pixels, points, voxels... is not trivial with convolution

Attention vs Convolution

Convolution has been extremely popular in 2D/3D vision:

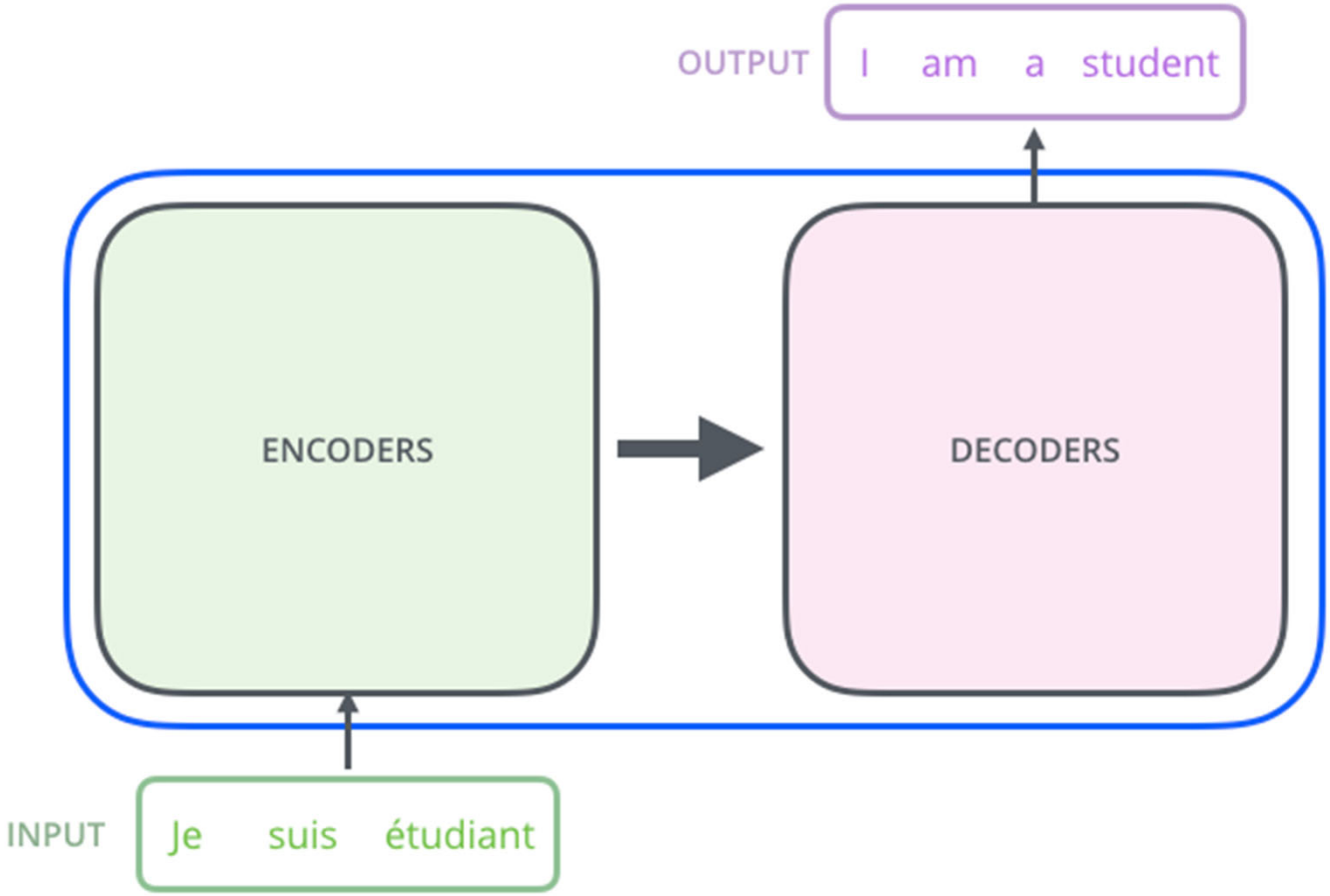
- ability to exploit local dependencies in the input data
- highly parallelizable / efficient to compute on GPUs

Capturing long-range interactions between pixels, points, voxels... is not trivial with convolution

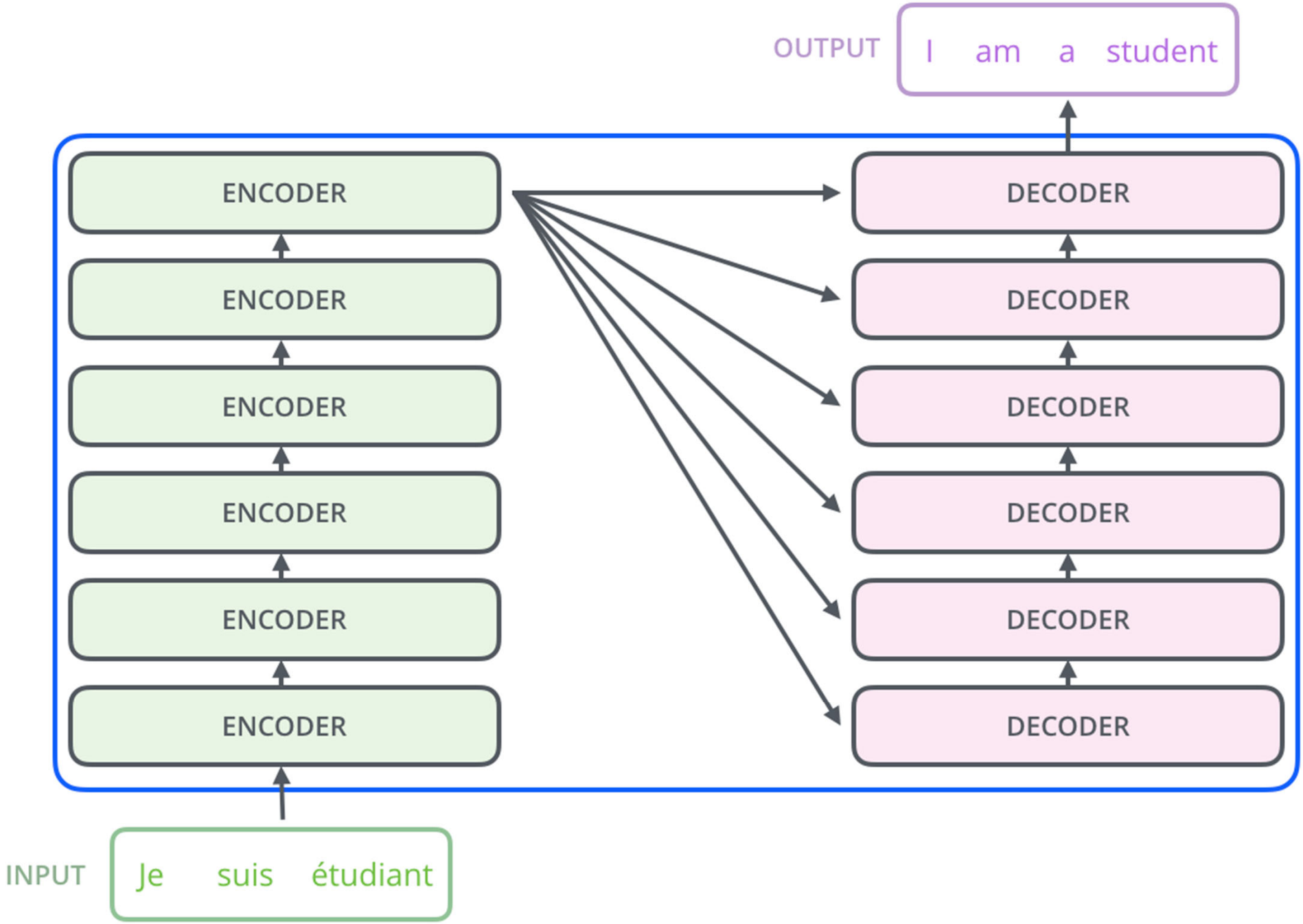
Attention was first introduced in NLP (e.g., machine translation) to capture such interactions!

Vaswani, Attention Is All You Need, 2017

Machine Translation

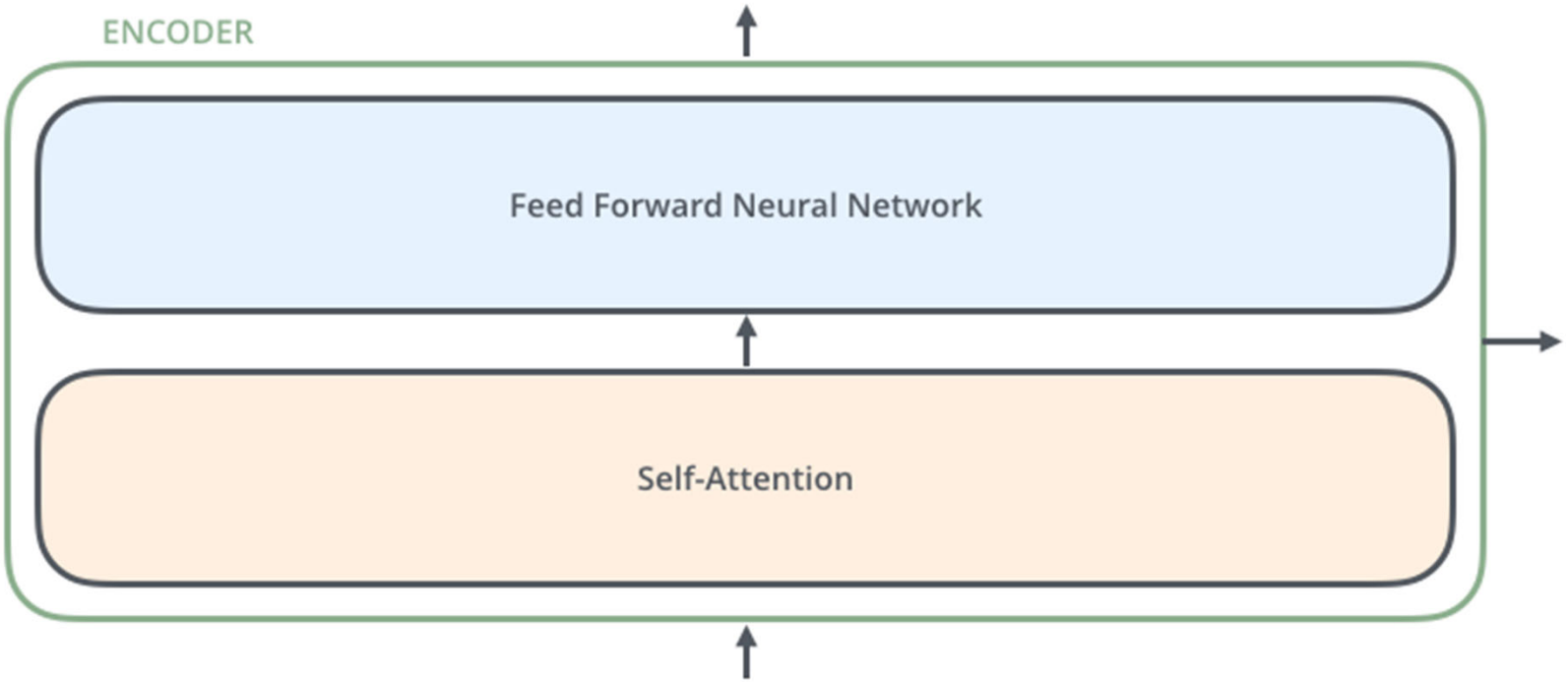


https://jalammar.github.io/illustrated-transformer/

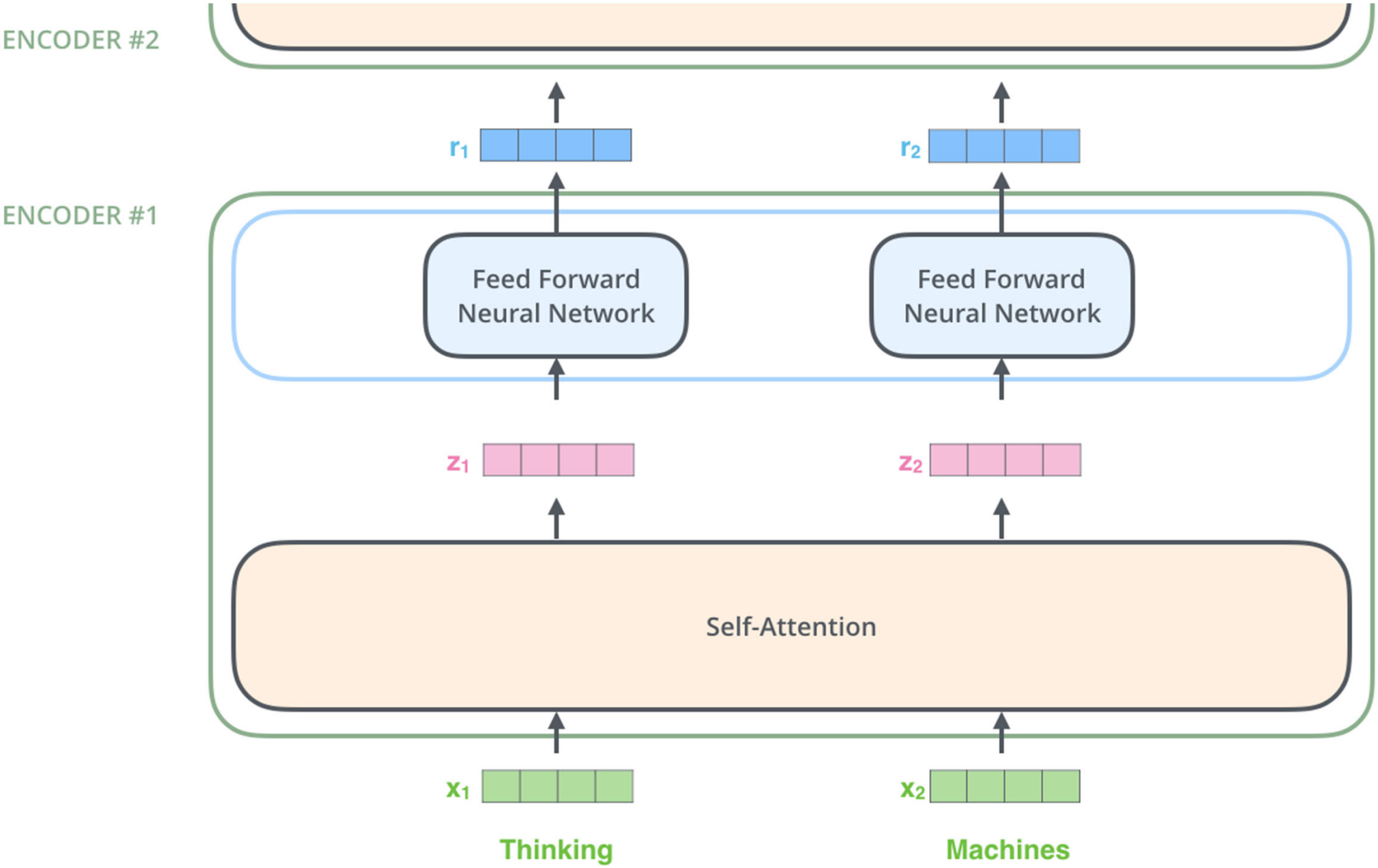


https://jalammar.github.io/illustrated-transformer/

The Encoder



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



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

What's the deal with attention?

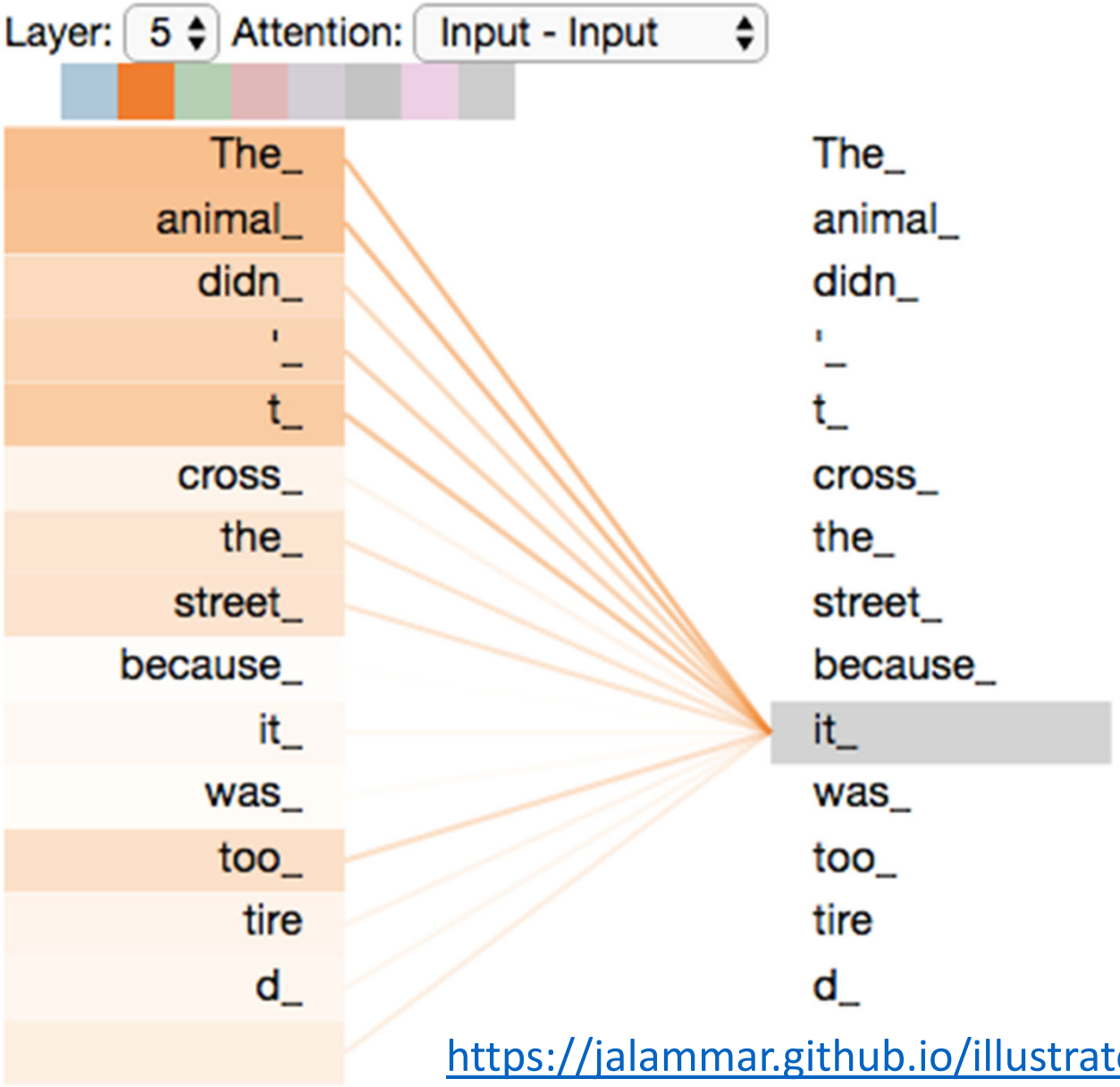
Say we want to translate the following sentence:

"The animal didn't cross the street because it was too tired"

What does "***it***" in this sentence refer to? Is it referring to the street or to the animal?

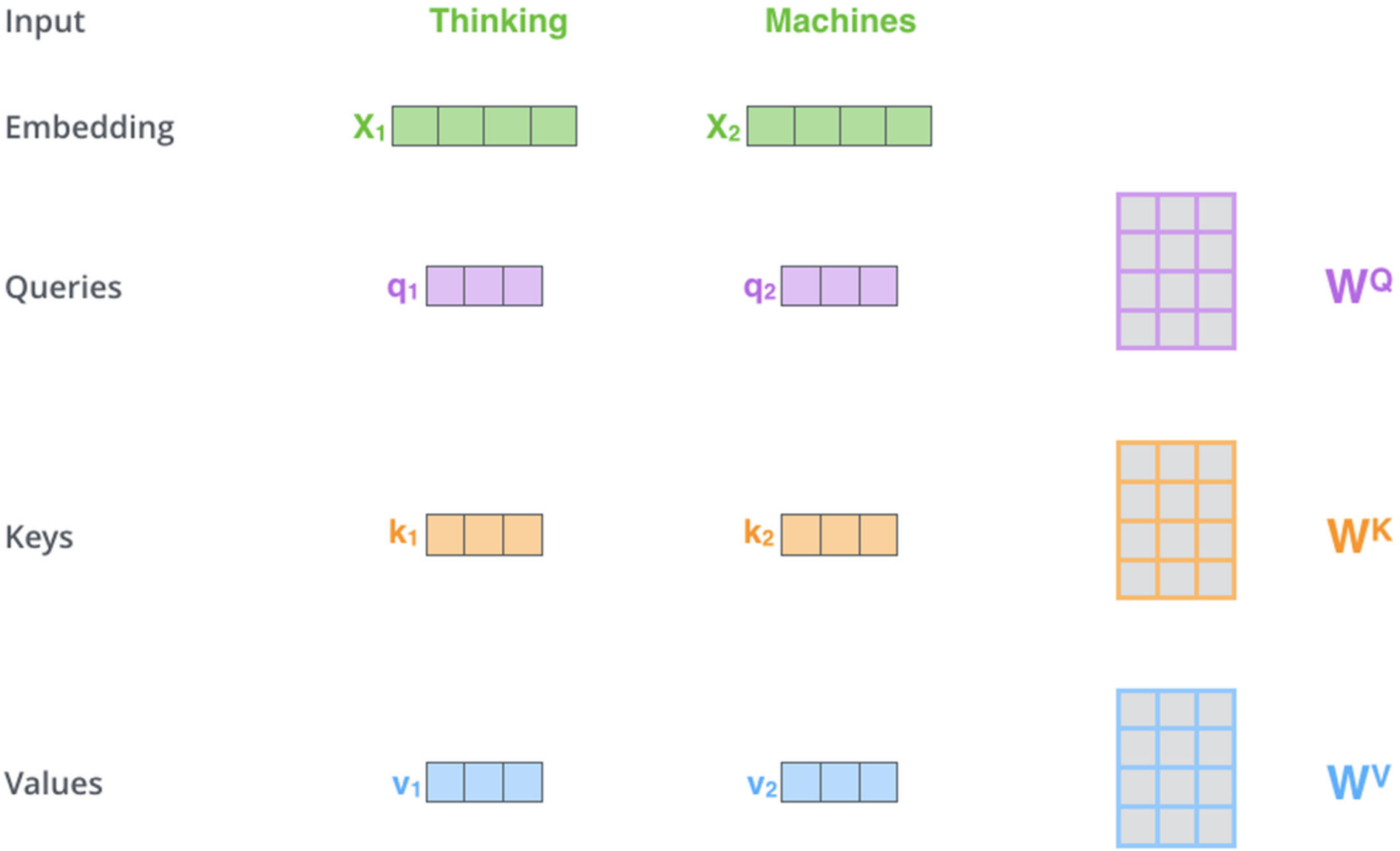
https://jalammar.github.io/illustrated-transformer/

Attention: allows it to look at other tokens in the input sequence that can help lead to a better encoding for this token!



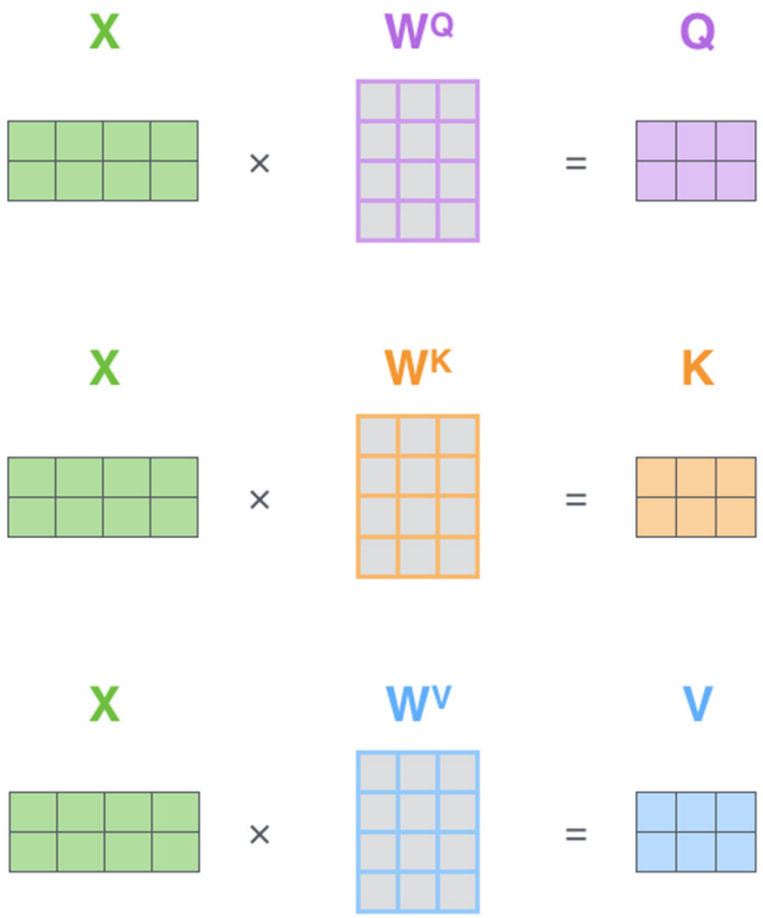
https://jalammar.github.io/illustrated-transformer/

Step A: create three vectors from each of the encoder's input token embeddings:
a Query vector, a Key vector, and a Value vector.



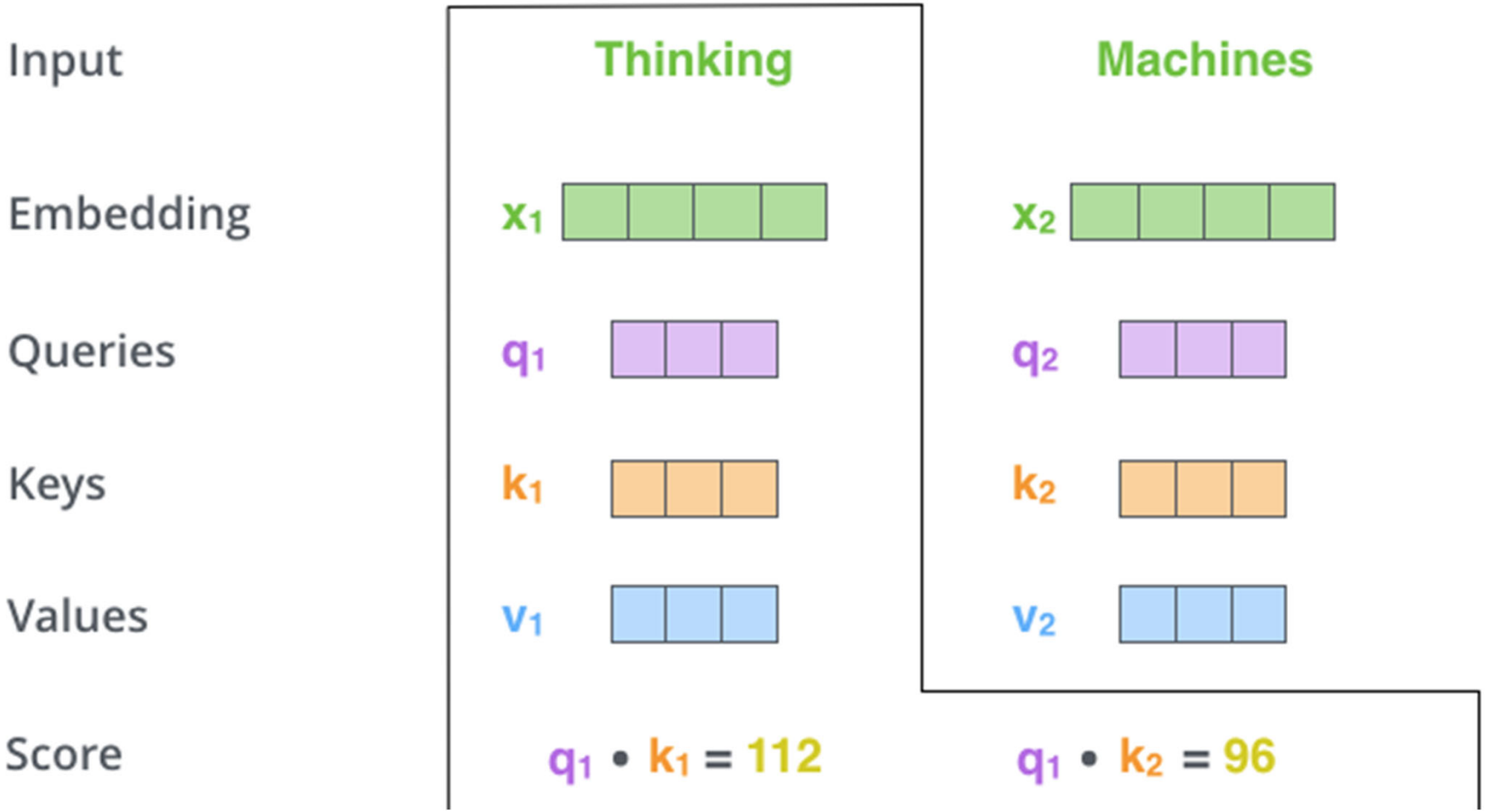
https://jalammar.github.io/illustrated-transformer/

How to obtain these **query**, **key**, **value** vectors?
Matrix multiplication with (learned) matrices W^q, W^k, W^v!



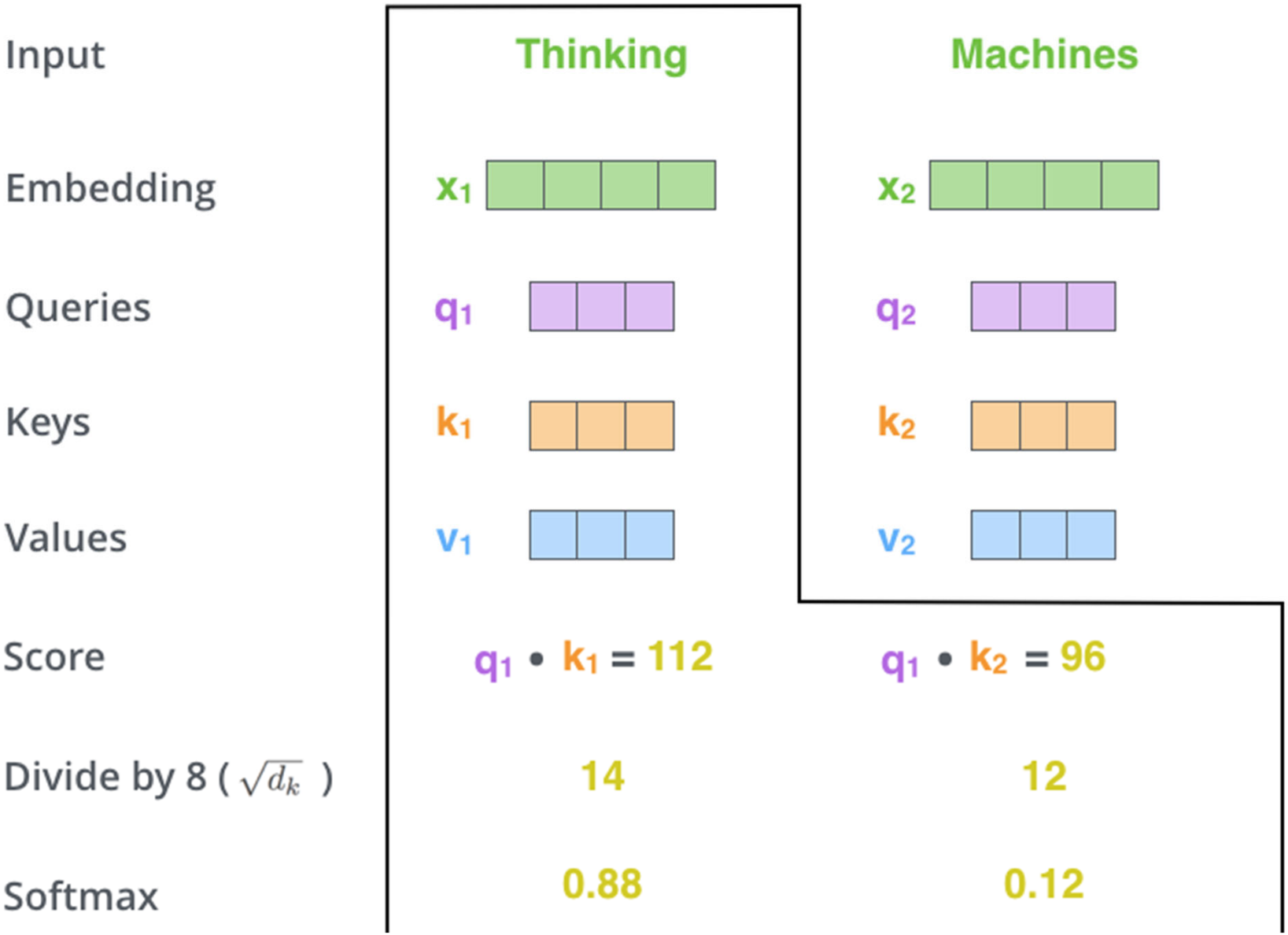
https://jalammar.github.io/illustrated-transformer/

Step B: calculate a **score** determining how much focus to place on other parts of the input sentence as we encode a token at a certain position.

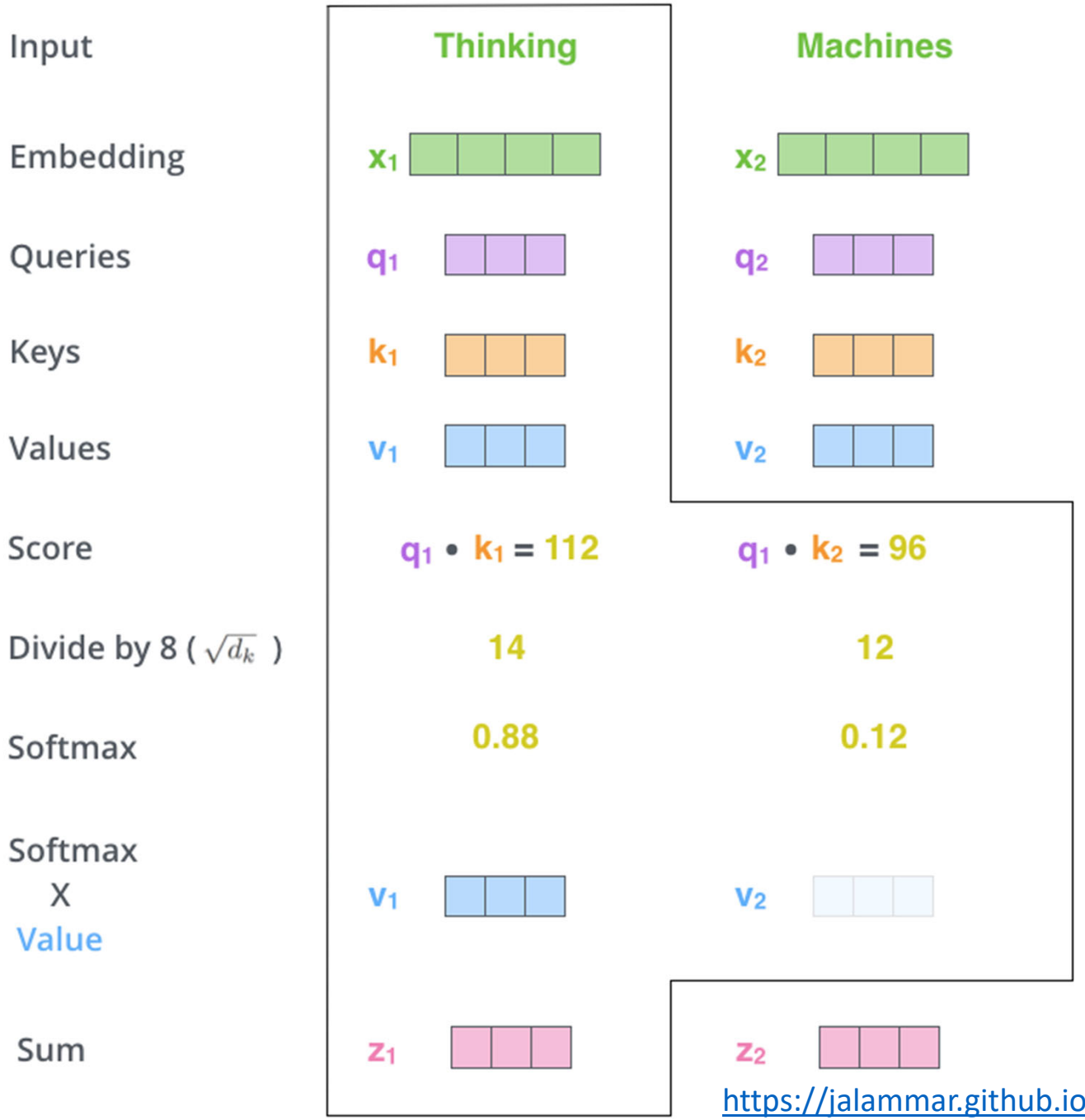


https://jalammar.github.io/illustrated-transformer/

Step C: divide the **scores** with the sqrt of the dimension of the vectors (64) -- helps with stable gradients & pass them through **softmax** so they're positive & sum up to 1.

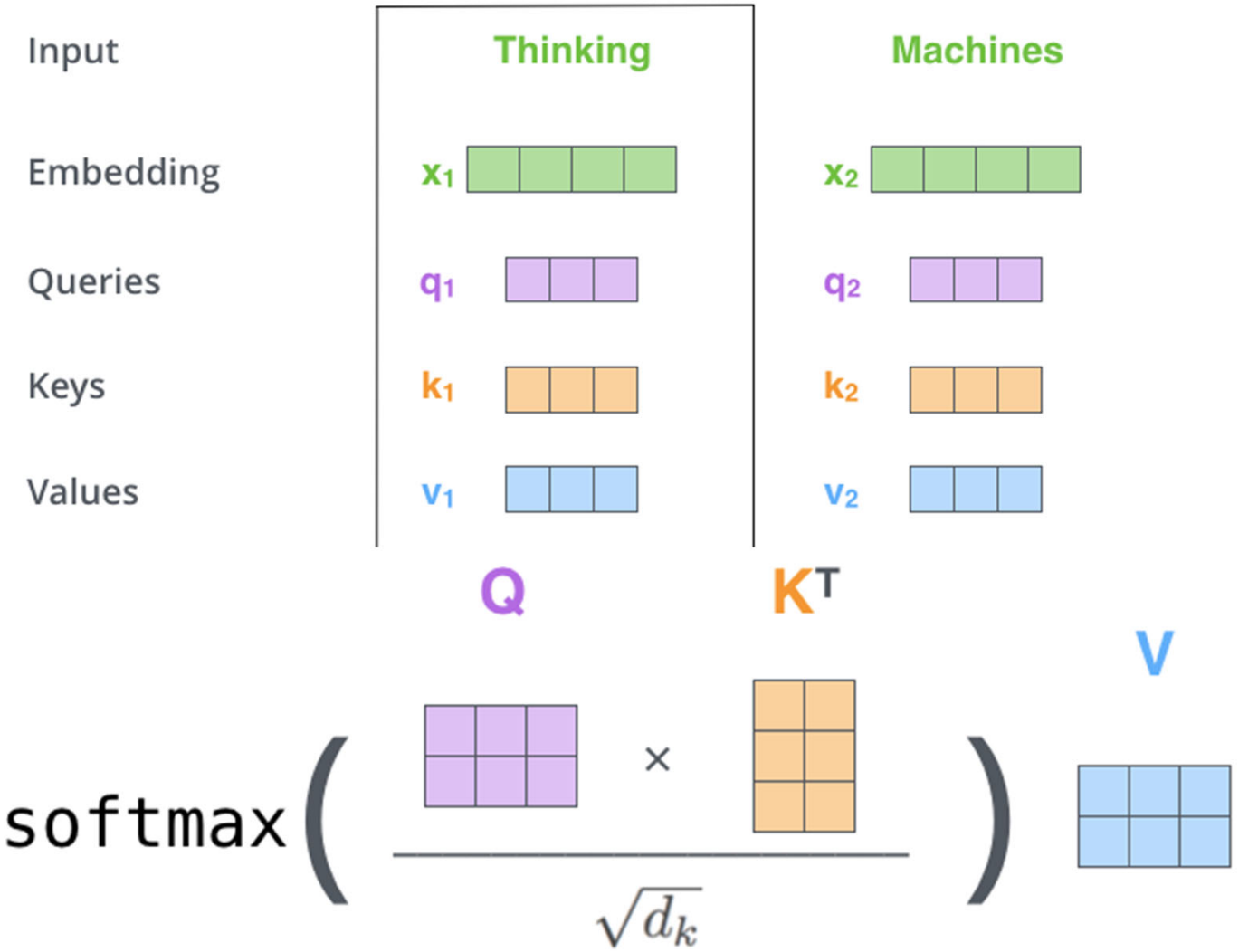


Step D: Multiply each **value** vector by the **softmax** score (keep intact the values of the tokens we want to focus on & drown-out irrelevant ones). Sum up the weighted value vectors.



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

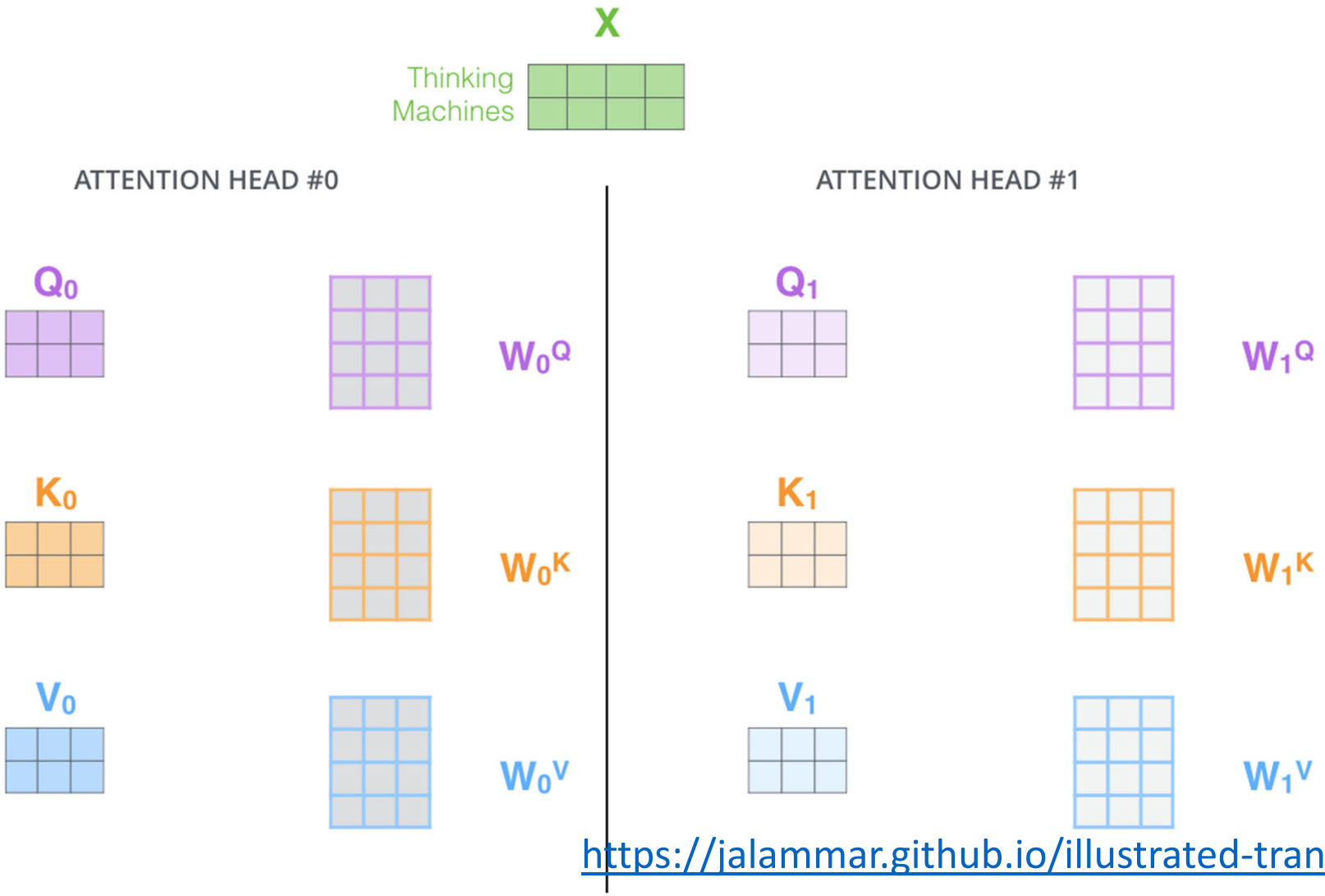
Step D: Multiply each **value** vector by the **softmax** score (keep intact the values of the tokens we want to focus on & drown-out irrelevant ones). Sum up the weighted value vectors.



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

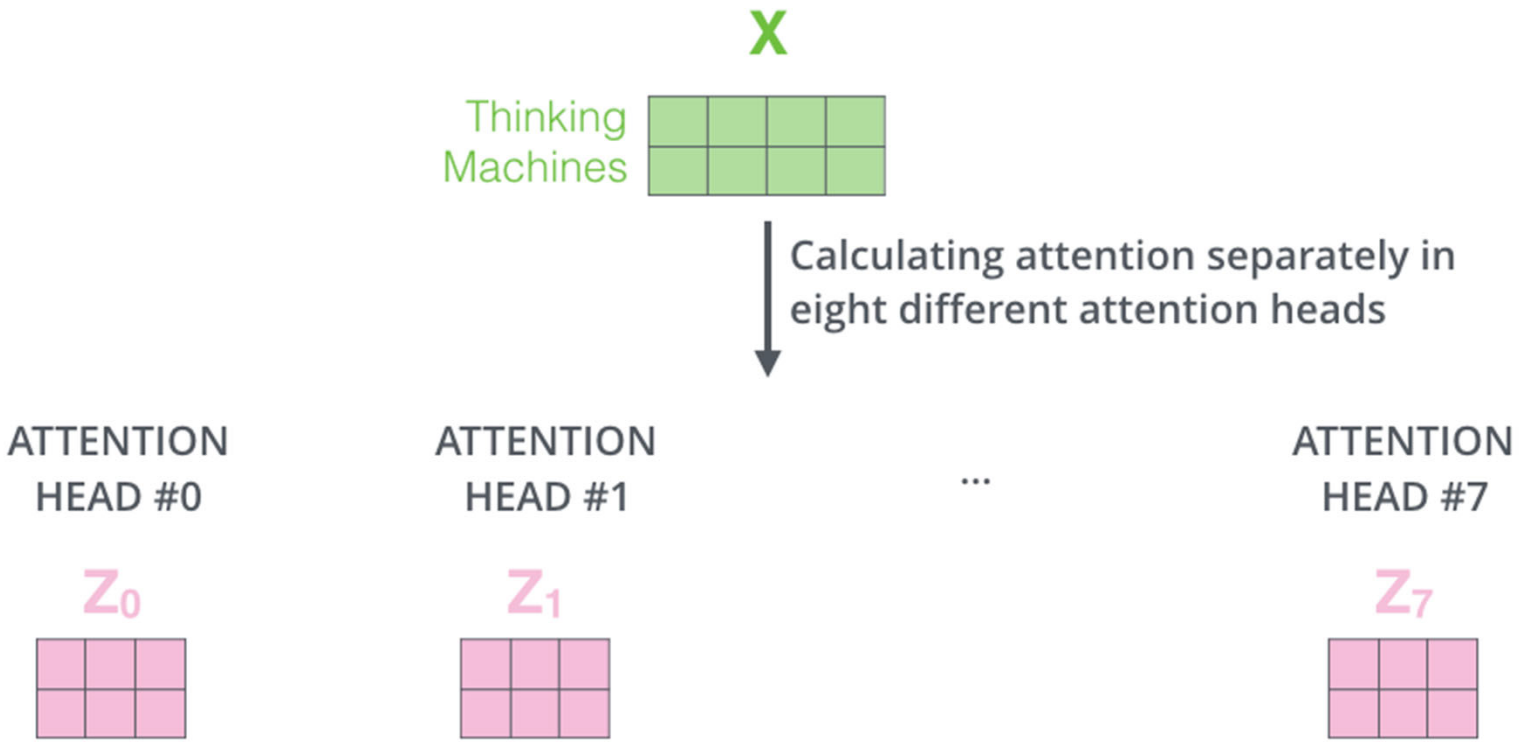
The beast with many “heads”!

Use multiple attention transformations to encode a token in multiple “representation” contexts e.g., in the sentence: ***“The animal didn't cross the street because it was too tired”***
encode the context of “it” wrt “animal” & also encode its context wrt “**was too tired**”



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

Processing with multiple attention heads…



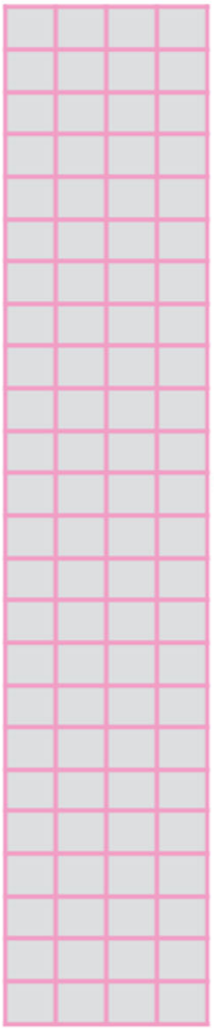
https://jalammar.github.io/illustrated-transformer/

Processing with multiple attention heads…

1) Concatenate all the attention heads

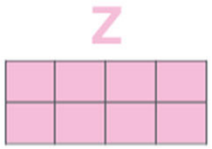
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



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

Summary…

1) This is our input sentence*

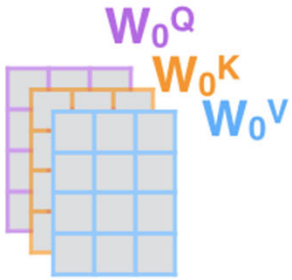
2) We embed each word*

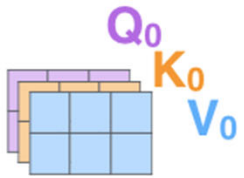
3) Split into 8 heads. We multiply X or R with weight matrices

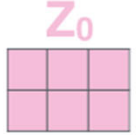
4) Calculate attention using the resulting Q/K/V matrices

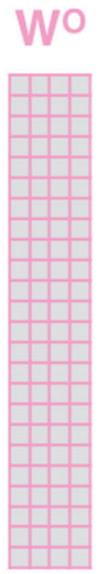
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

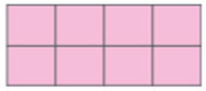
Thinking Machines



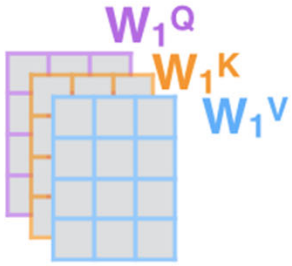


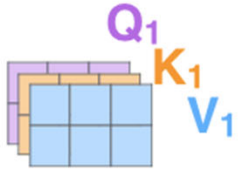


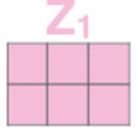




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



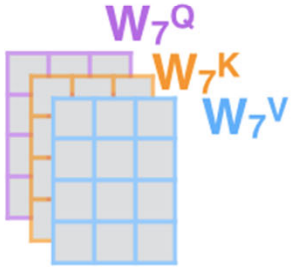


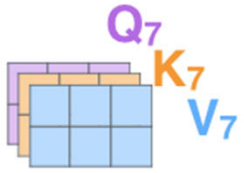


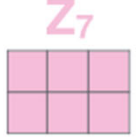
…

…

…

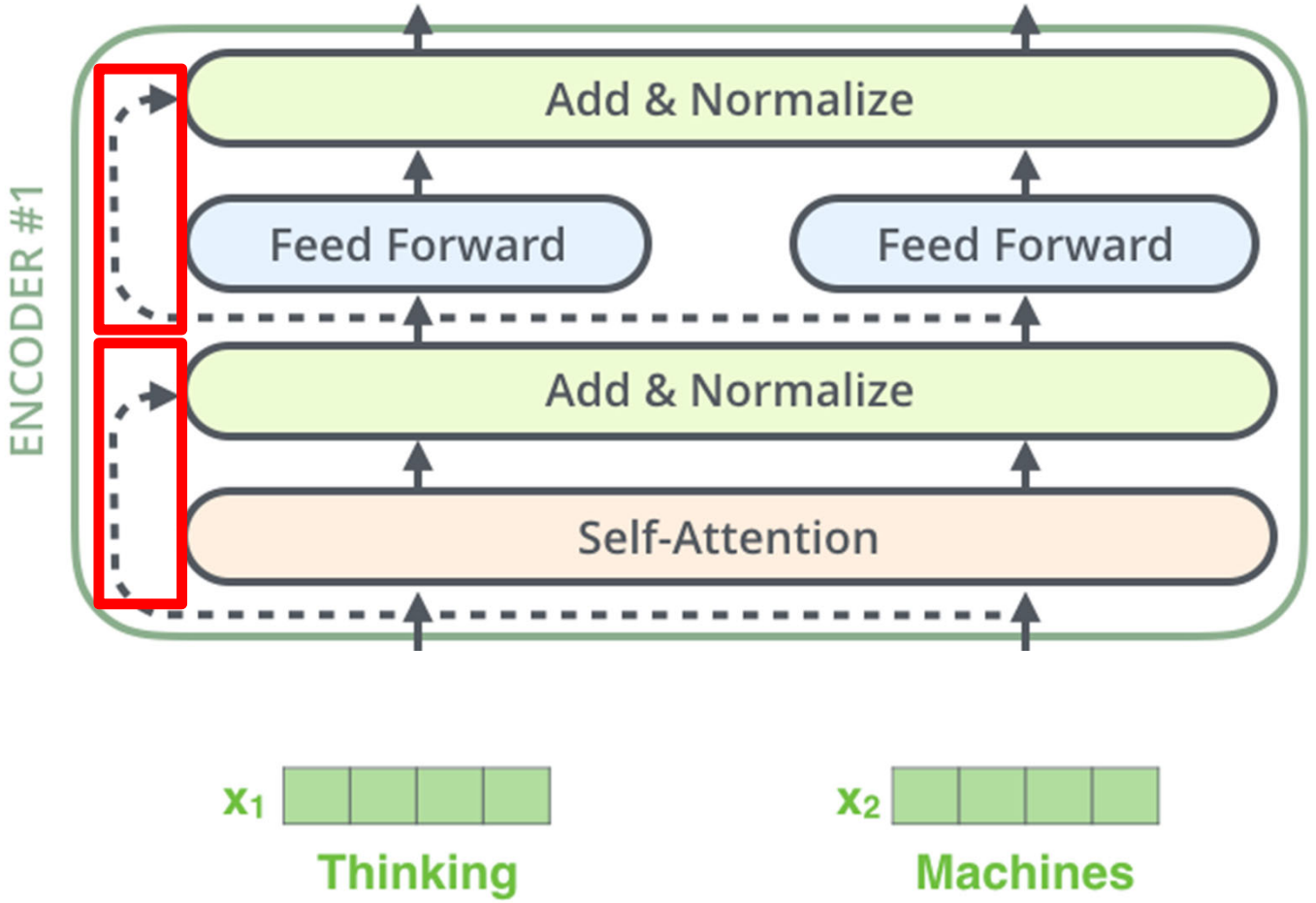






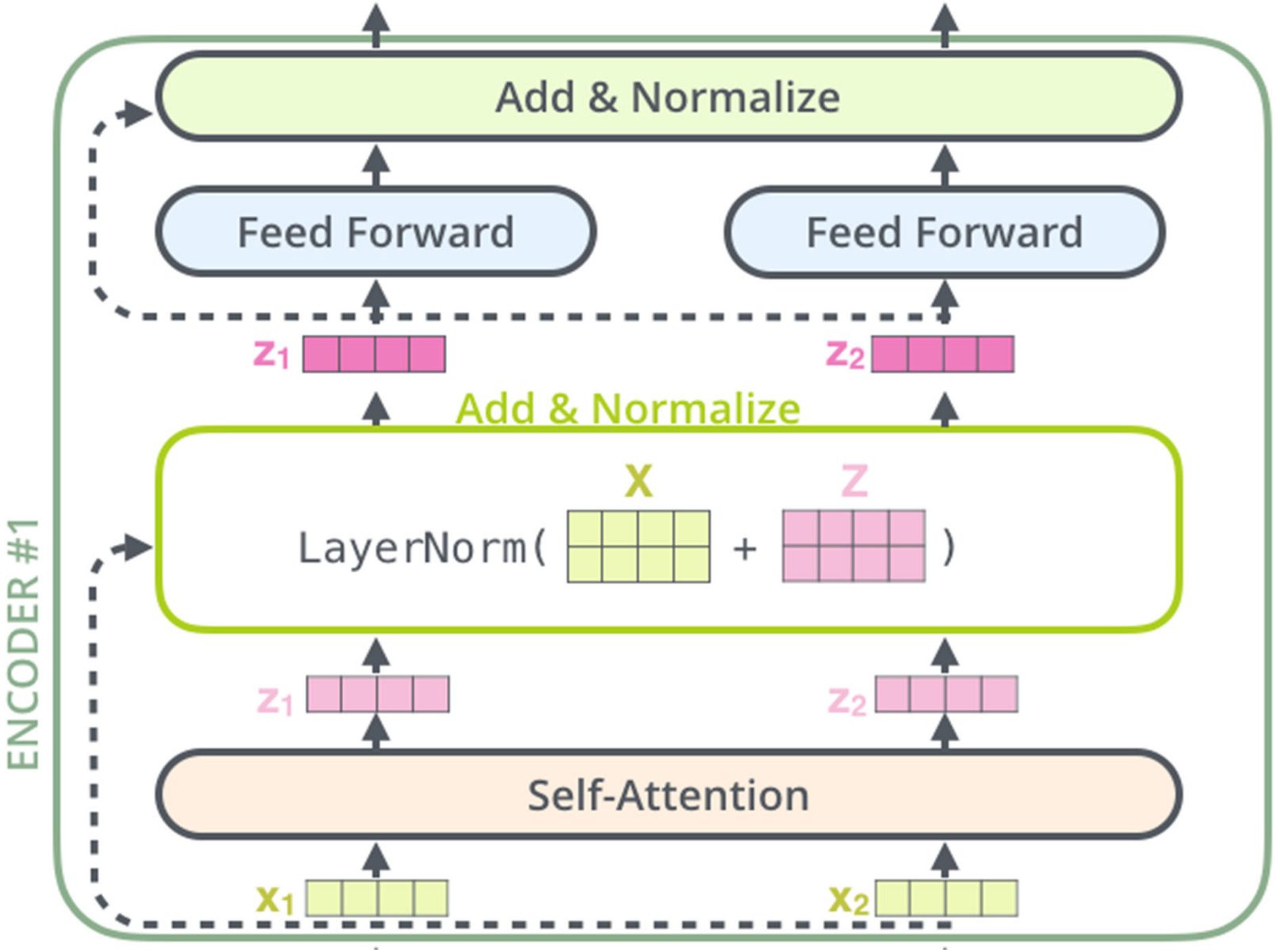
https://jalammar.github.io/illustrated-transformer/

More details: residual connections



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

More details: layer normalizattion



Thinking Machines

Another detail: so far the model has no idea of what the position of the token is in a sentence!

"The animal didn't cross the street because it was too tired"

0 1 2 3 4 5 6 7 8 9 10 ?

"Thinking Machines"

0 1 ?

Another detail: so far the model has no idea of what the position of the token is in a sentence!

"The animal didn't cross the street because it was too tired"

0 1 2 3 4 5 6 7 8 9 10 ?

"Thinking Machines"

0 1 ?

Problem: if the largest sentence is 50 tokens long during training, how can the model generalization to sentences longer than 50 during testing?

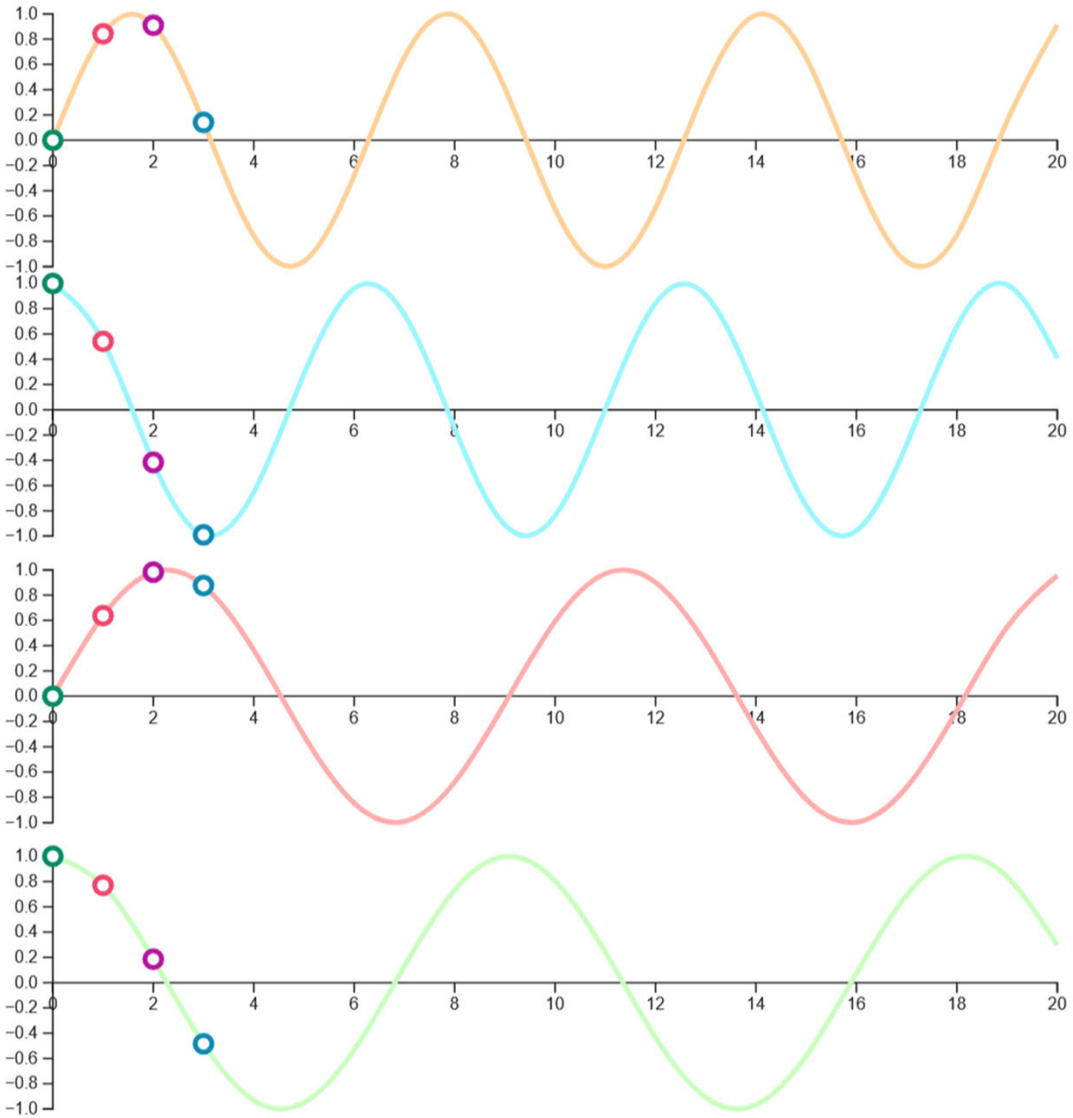
Another detail: so far the model has no idea of what the position of the token is in a sentence!

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

Positional encoding visualization



p0	p1	p2	p3	
0.000	0.841	0.909	0.141	**i=0**
1.000	0.540	-0.416	-0.990	**i=1**
0.000	0.638	0.983	0.875	**i=2**
1.000	0.770	0.186	-0.484	**i=3**

Positional Encoding

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

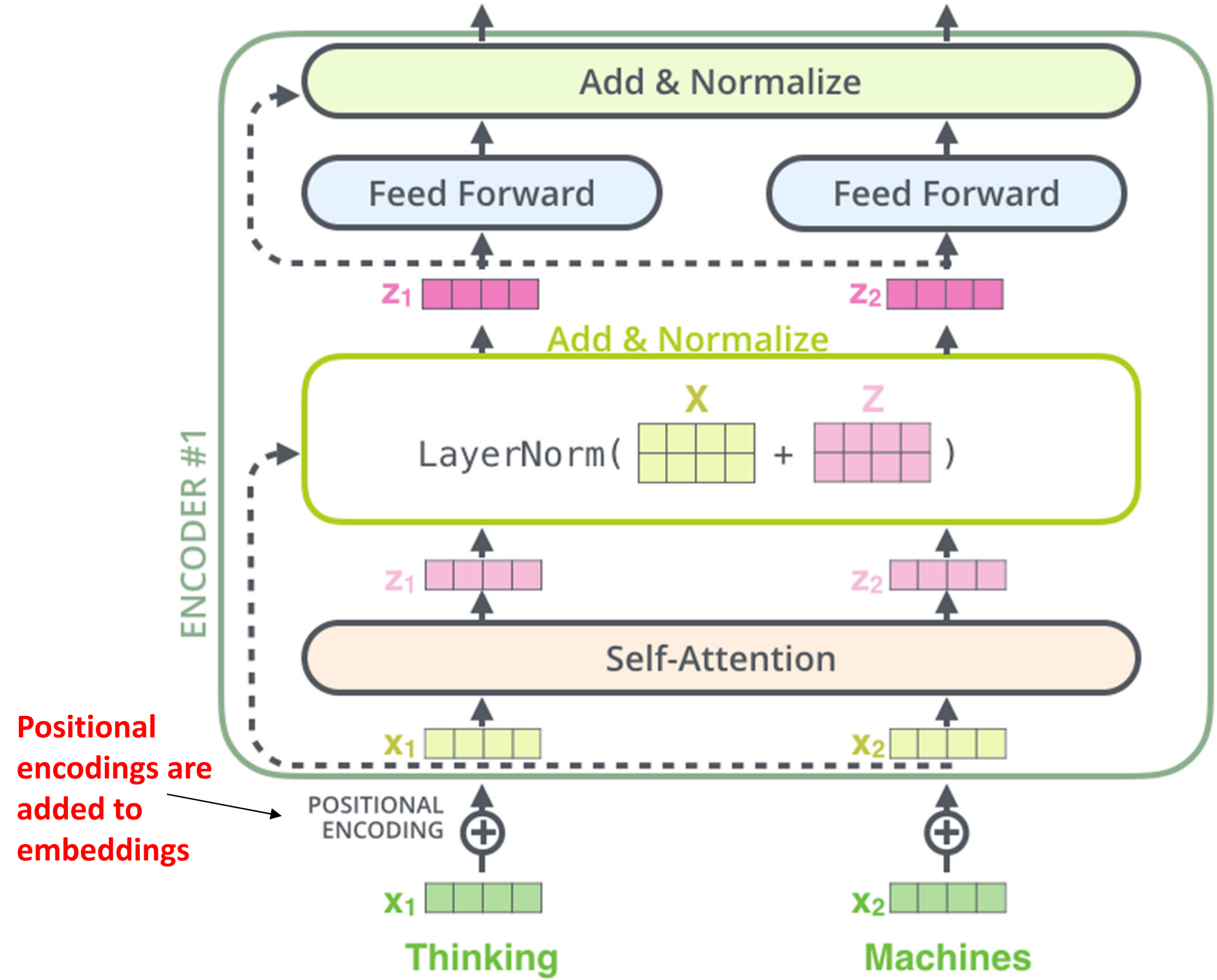
$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

Settings: d = 50

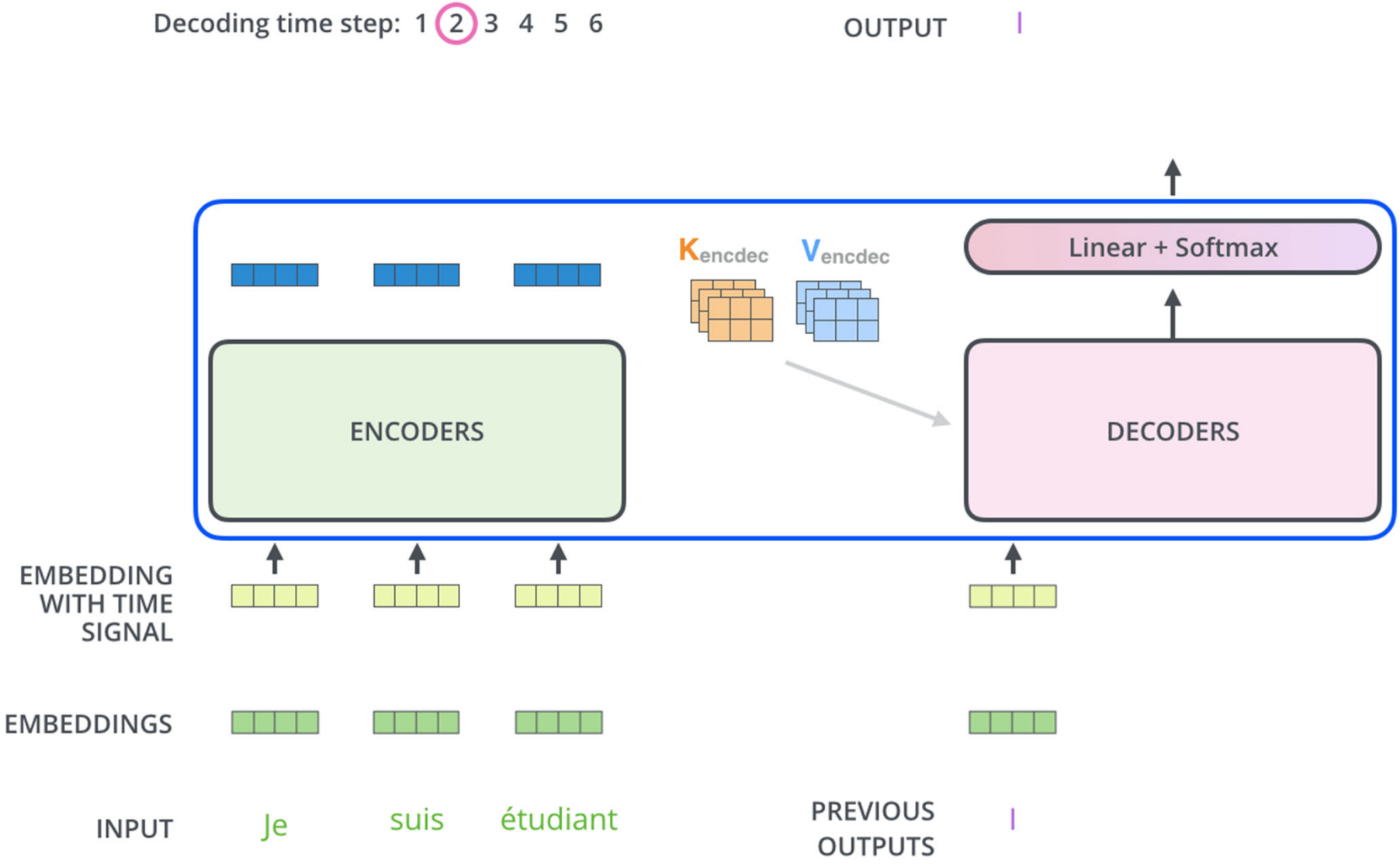
The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

pos0 pos1 pos2 pos3

https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

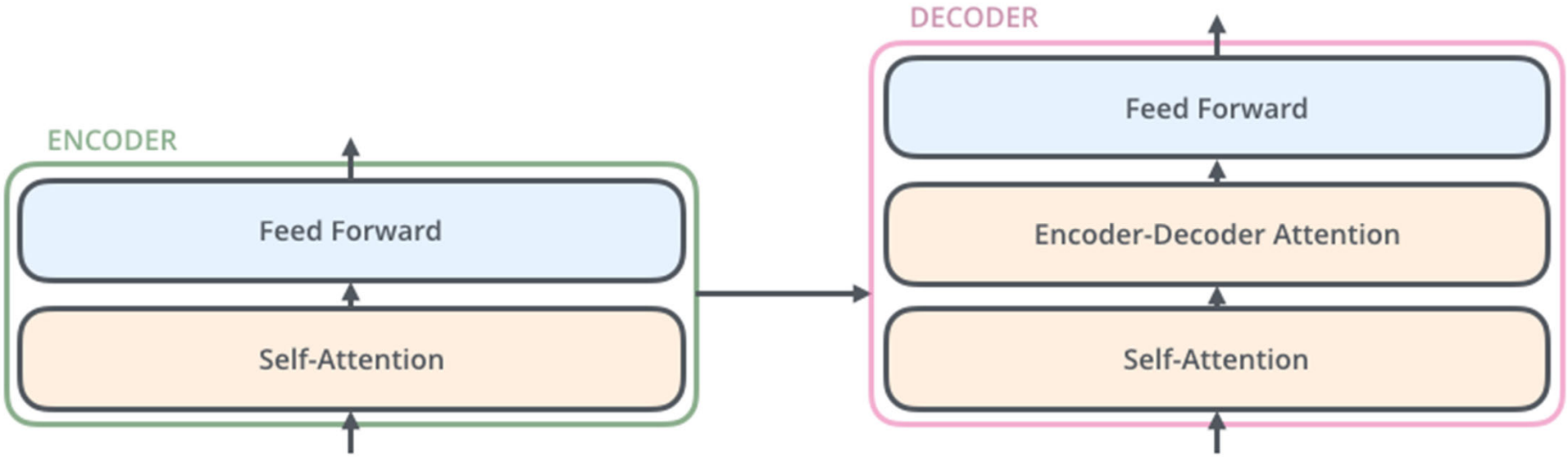

Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
ENCODER #1
z1
z2
Self-Attention
x1
x2
Positional encodings are added to embeddings
POSITIONAL ENCODING
x1
x2
Thinking
Machines

The decoder



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

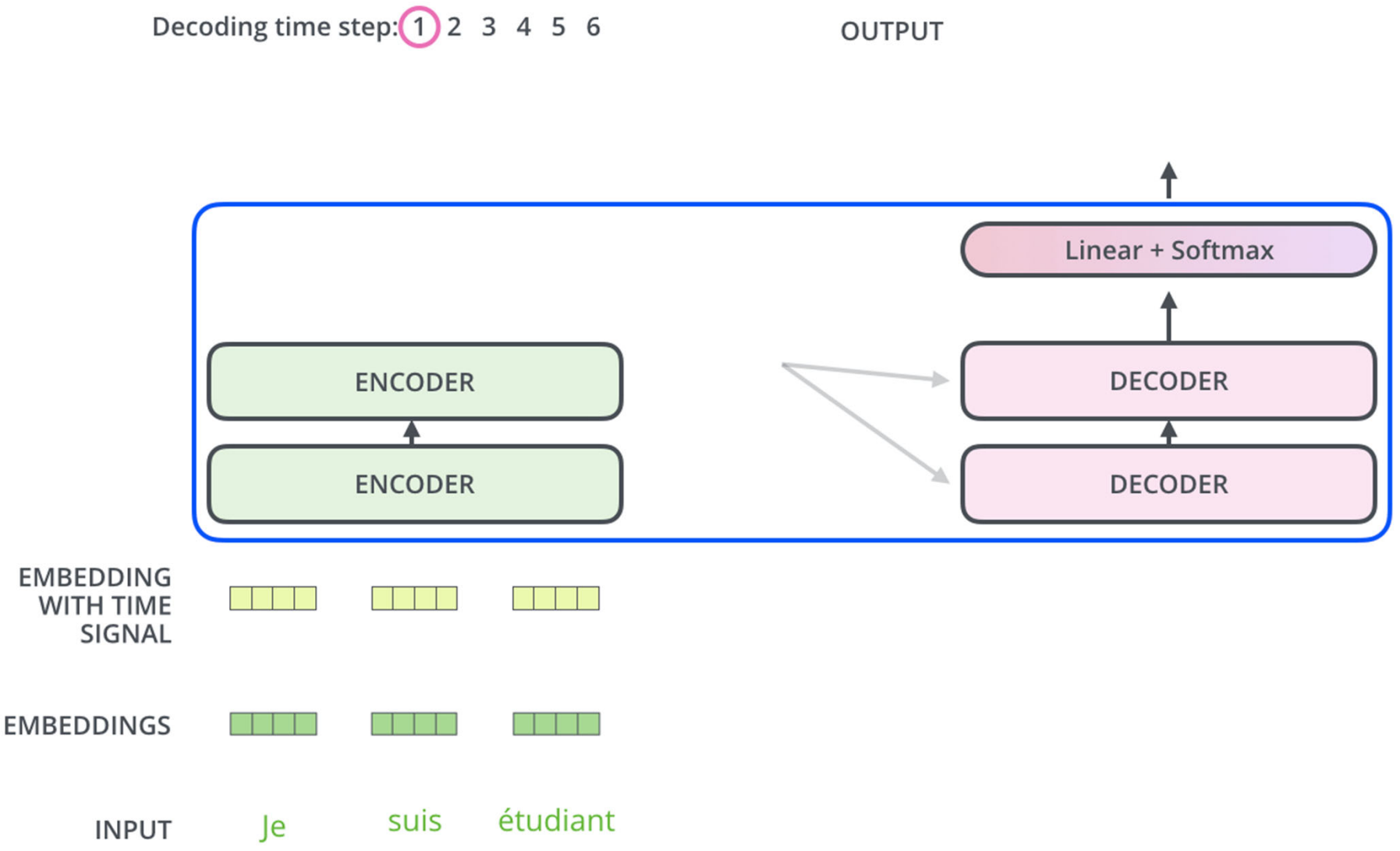
The decoder



the self-attention layer is similar to the encoder's self attention, yet attends only to earlier positions in the output sequence

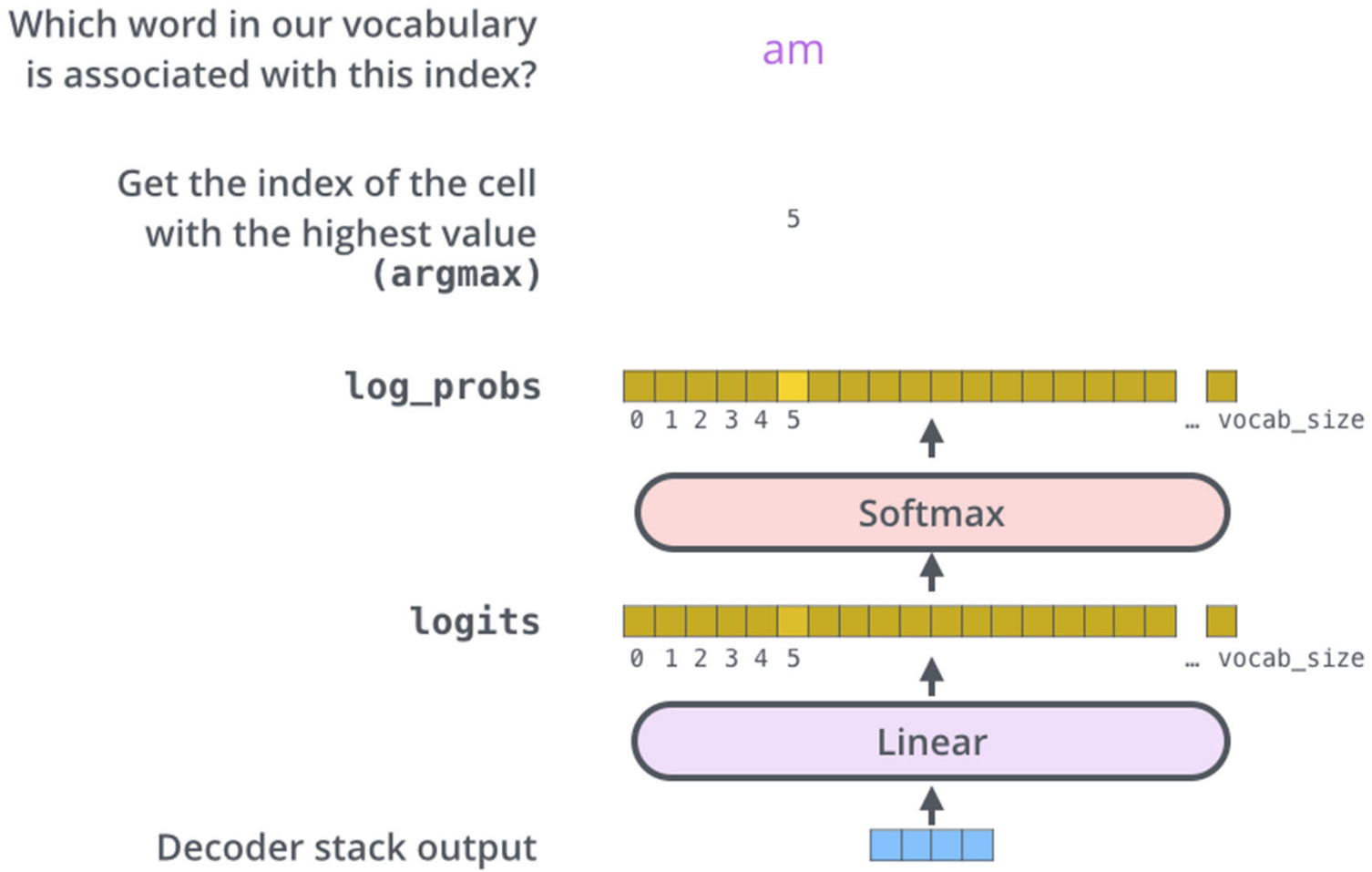
https://jalammar.github.io/illustrated-transformer/

The output of the last encoder block provides an attention vector **K** and **V** for each **input token**. These are used by the decoder blocks in the "**encoder-decoder attention**" layers helping the decoder focus on appropriate places in the input sentence!



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

The final layer of the decoder

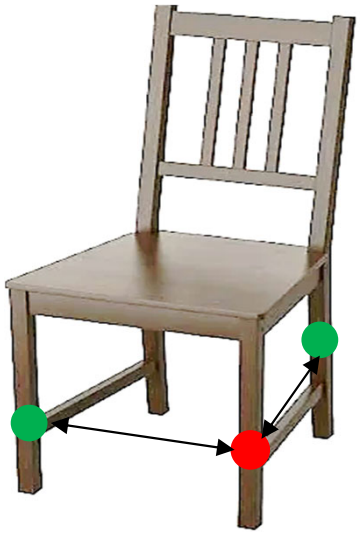


Attention for visual data (images/shapes)

Help encoder look at other pixels/patches in image while encoding a pixel/patch due to various relations that may exist in objects or scenes in general

Capture visual context

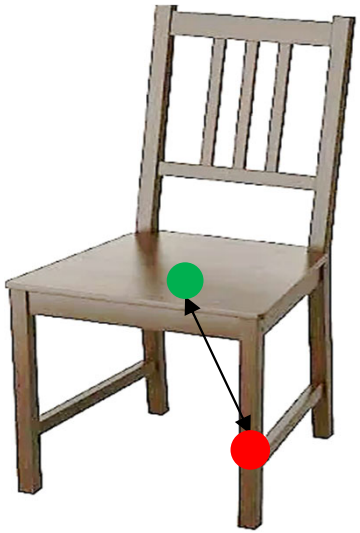
Help encoder look at other pixels/patches in image while encoding a pixel/patch due to various relations that may exist in objects or scenes in general



e.g., symmetries

Capture visual context

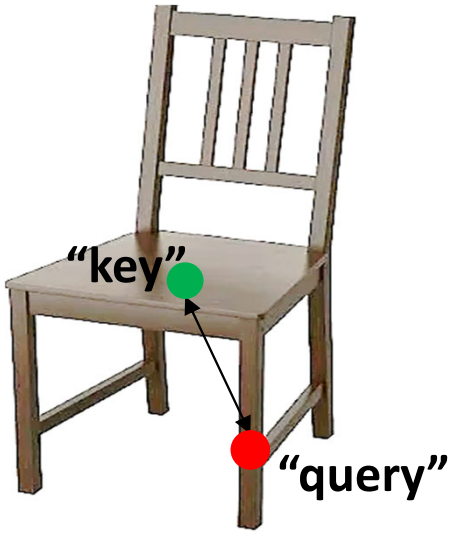
Help encoder look at other pixels/patches in image while encoding a pixel/patch due to various relations that may exist in objects or scenes in general



... or any pixel/patch relations may help recognize objects

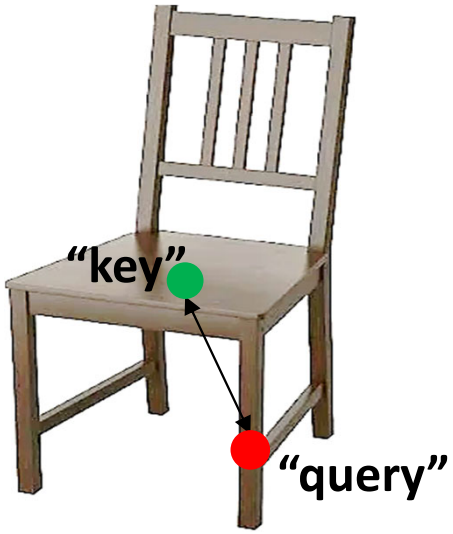
Attention transformations

How much a pixel/patch (query) is related to others (keys)?



Attention transformations

How much a pixel/patch (query) is related to others (keys)?



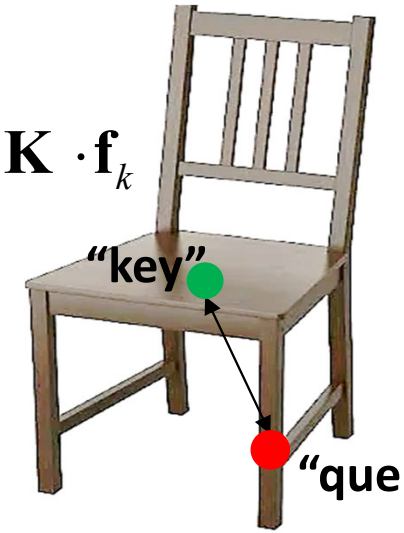
$$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$$

A linear transformation on the input feature vector of the query pixel/patch (e.g., MLP on input point/patch raw features)

Attention transformations

How much a pixel/patch (query) is related to others (keys)?

$$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$$



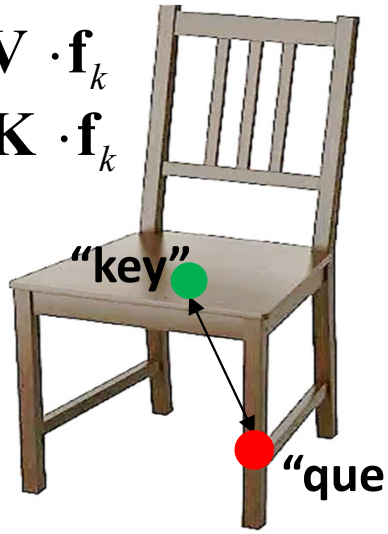
Another transformation on the input feature vector of the key point

$$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$$

Attention transformations

How much a pixel/patch (query) is related to others (keys)?

$$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$$
$$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$$



One more transformation on feature vector of the key point i.e., we have a "key-value" pair

$$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$$

Attention transformations

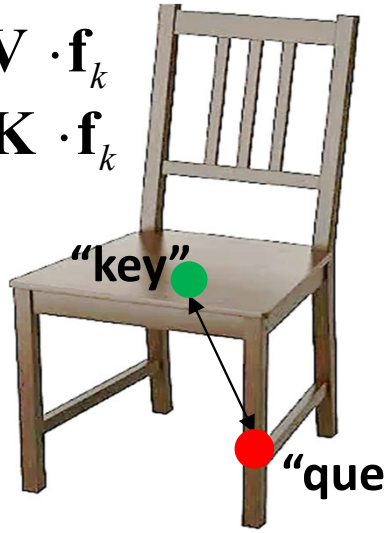
How much a pixel/patch (query) is related to others (keys)?

Attention "score": $a(\mathbf{f}_q, \mathbf{f}_k) = k(\mathbf{f}_k) \cdot q(\mathbf{f}_q)$

(dot product between key-query vectors)

$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$

$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$



$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$

Attention score

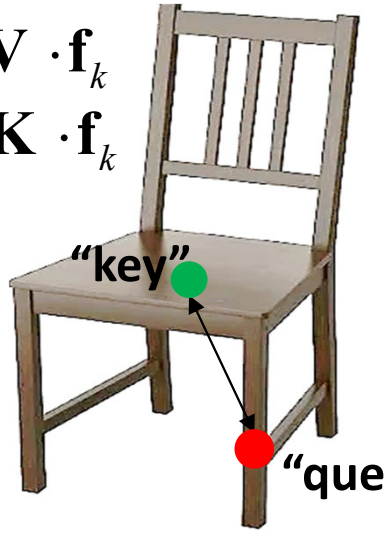
How much a pixel/patch (query) is related to others (keys)?

Attention "score": $\hat{a}(\mathbf{f}_q, \mathbf{f}_k) = \frac{\exp\{ a(\mathbf{f}_q, \mathbf{f}_k) \}}{\sum_{k'} \exp\{ a(\mathbf{f}_q, \mathbf{f}_{k'}) \}}$

(i.e., use softmax)

$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$

$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$



$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$

Attention score

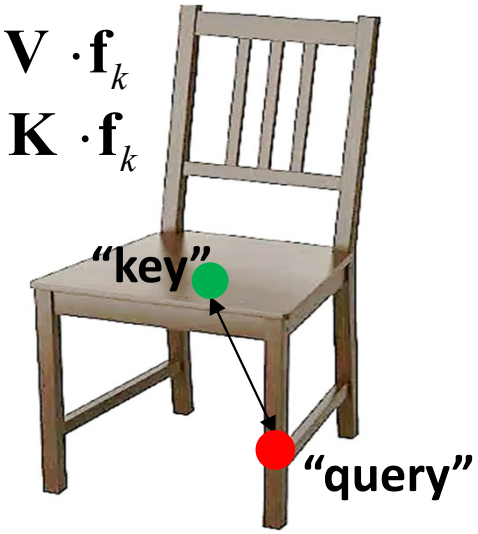
How much a pixel/patch (query) is related to others (keys)?

Attention "score": $\hat{a}(\mathbf{f}_q, \mathbf{f}_k) = \frac{\exp\{ a(\mathbf{f}_q, \mathbf{f}_k) \}}{\sum_{k'} \exp\{ a(\mathbf{f}_q, \mathbf{f}_{k'}) \}}$

(i.e., use softmax)

$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$

$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$



$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$

Problem: As the number of dimensions D of the input feature vector gets large, the variance of the dot product increased ...the input to softmax gets high values... the softmax is too peaked ...
=> tiny gradients

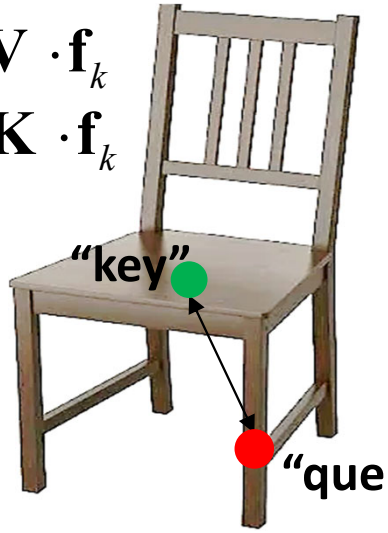
Scaled dot-product attention

How much a pixel/patch (query) is related to others (keys)?

Attention "score": (i.e., use softmax)

$$\hat{a}(\mathbf{f}_q, \mathbf{f}_k) = \frac{\exp\{ a(\mathbf{f}_q, \mathbf{f}_k) / \sqrt{D} \}}{\sum_{k'} \exp\{ a(\mathbf{f}_q, \mathbf{f}_{k'}) / \sqrt{D} \}}$$

$$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$$
$$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$$



$$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$$

https://ai.stackexchange.com/questions/41861/why-use-a-square-root-in-the-scaled-dot-product

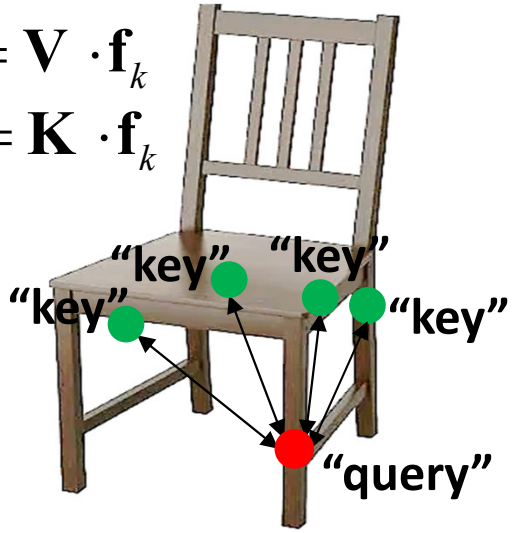
Vaswani, Attention Is All You Need, 2017

New feature representations

Compute a new encoding for query pixel/patch as a **weighted sum of values of key points:**

$$New\ features: \mathbf{f}_q' = \sum_k \hat{a}(\mathbf{f}_q, \mathbf{f}_k)\, v(\mathbf{f}_k)$$

$$v(\mathbf{f}_k) = \mathbf{V} \cdot \mathbf{f}_k$$
$$k(\mathbf{f}_k) = \mathbf{K} \cdot \mathbf{f}_k$$



$$q(\mathbf{f}_q) = \mathbf{Q} \cdot \mathbf{f}_q$$

... these can be added back to the input feature vector as a residual and further processed by an MLP (fully connected layer)

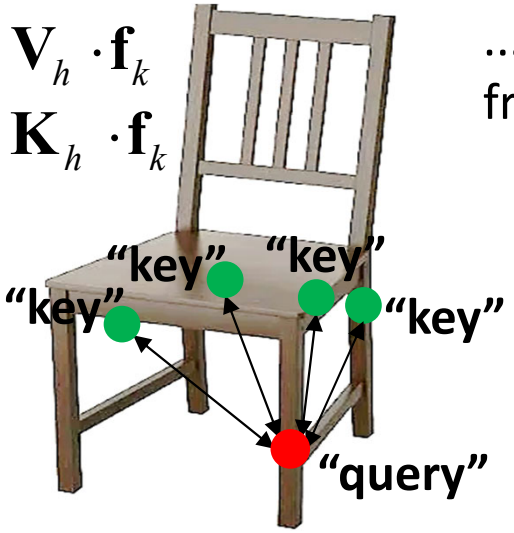
Vaswani, Attention Is All You Need, 2017

Multi-head Attention

Learn different query, key, value transformations for each attention "head".

$$Multi-head\ \ attention: \mathbf{f}_{q,h}' = \sum_k \hat{a}_h(\mathbf{f}_q, \mathbf{f}_k)\, v_h(\mathbf{f}_k)$$

$$v(\mathbf{f}_k) = \mathbf{V}_h \cdot \mathbf{f}_k$$
$$k(\mathbf{f}_k) = \mathbf{K}_h \cdot \mathbf{f}_k$$



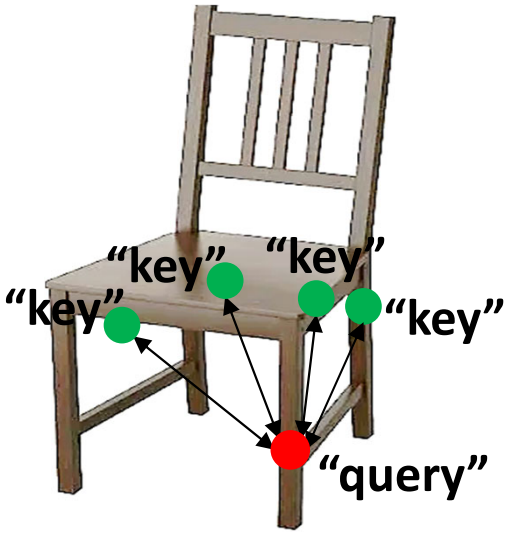
... concatenate the resulting features from all heads, and process them with the MLP

$$q(\mathbf{f}_q) = \mathbf{Q}_h \cdot \mathbf{f}_q$$

Vaswani, Attention Is All You Need, 2017

Quadratic complexity!

Comparing each query (N pixels) with every key (N pixels) yields **quadratic complexity**!



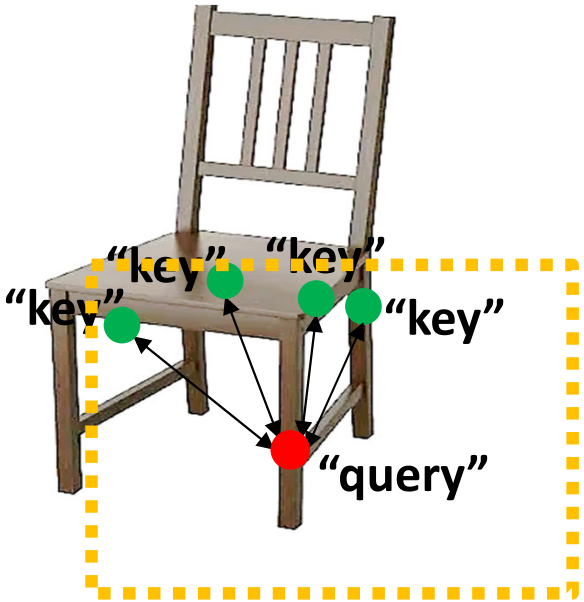
Vaswani, Attention Is All You Need, 2017

Quadratic complexity!

Solutions:

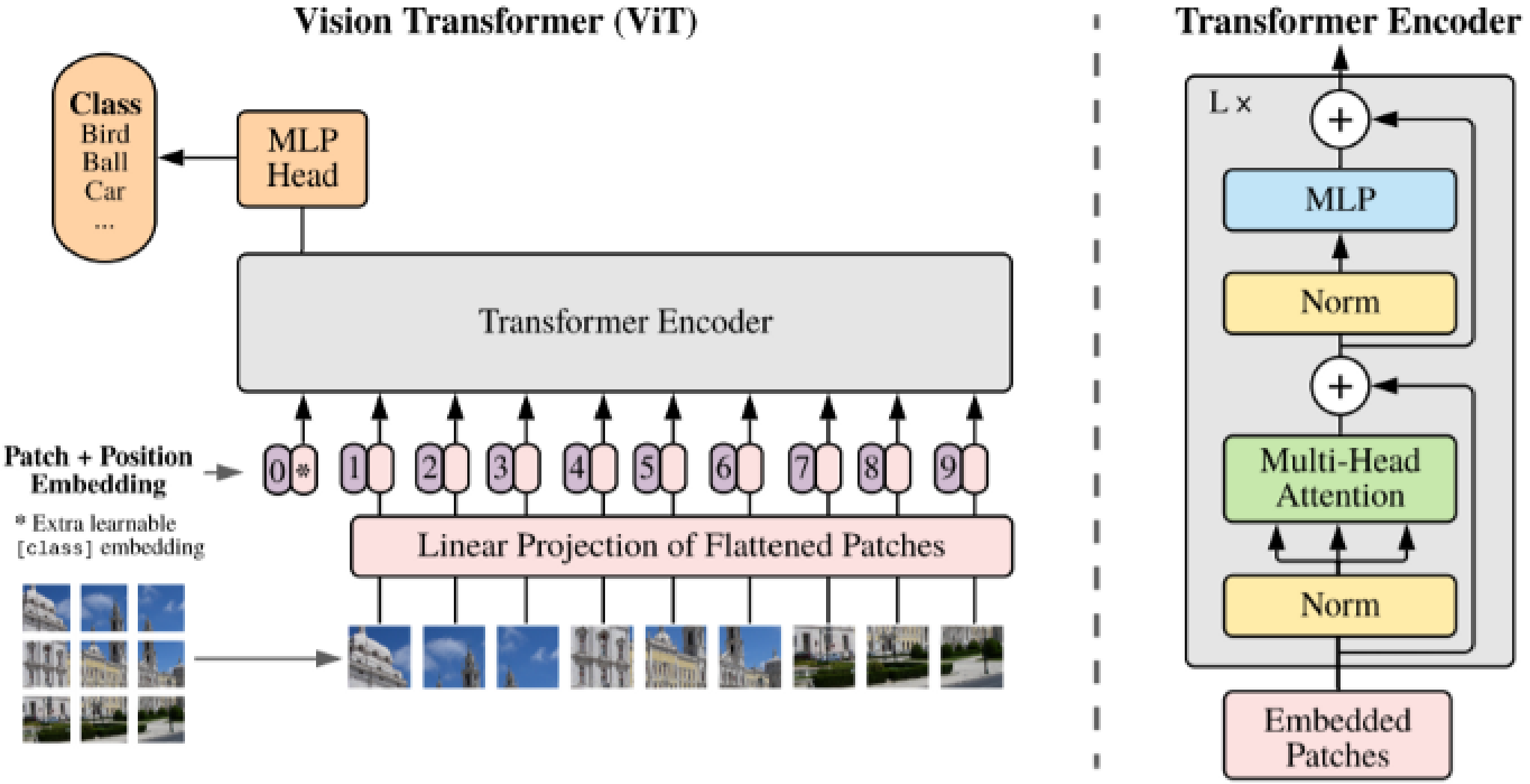
(a) Compute attention between patches instead of pixels (common in 2D vision transformers)

(b) Limit attention within regions e.g., 3D Euclidean balls (common in point transformers)



Vaswani, Attention Is All You Need, 2017

Vision Transformers



"An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", 2021

Transformers as Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \ldots\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles, or a sentence is made out of tokens ...

Transformers as Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \ldots\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles, or a sentence is made out of tokens ...

Data distribution is modeled as:

$$P(\mathbf{X}) = P(\mathbf{x}_1) \cdot P(\mathbf{x}_2 \mid \mathbf{x}_1) \cdot P(\mathbf{x}_3 \mid \mathbf{x}_1, \mathbf{x}_2) \cdot P(\mathbf{x}_4 \mid \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \ldots$$

Transformers as Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

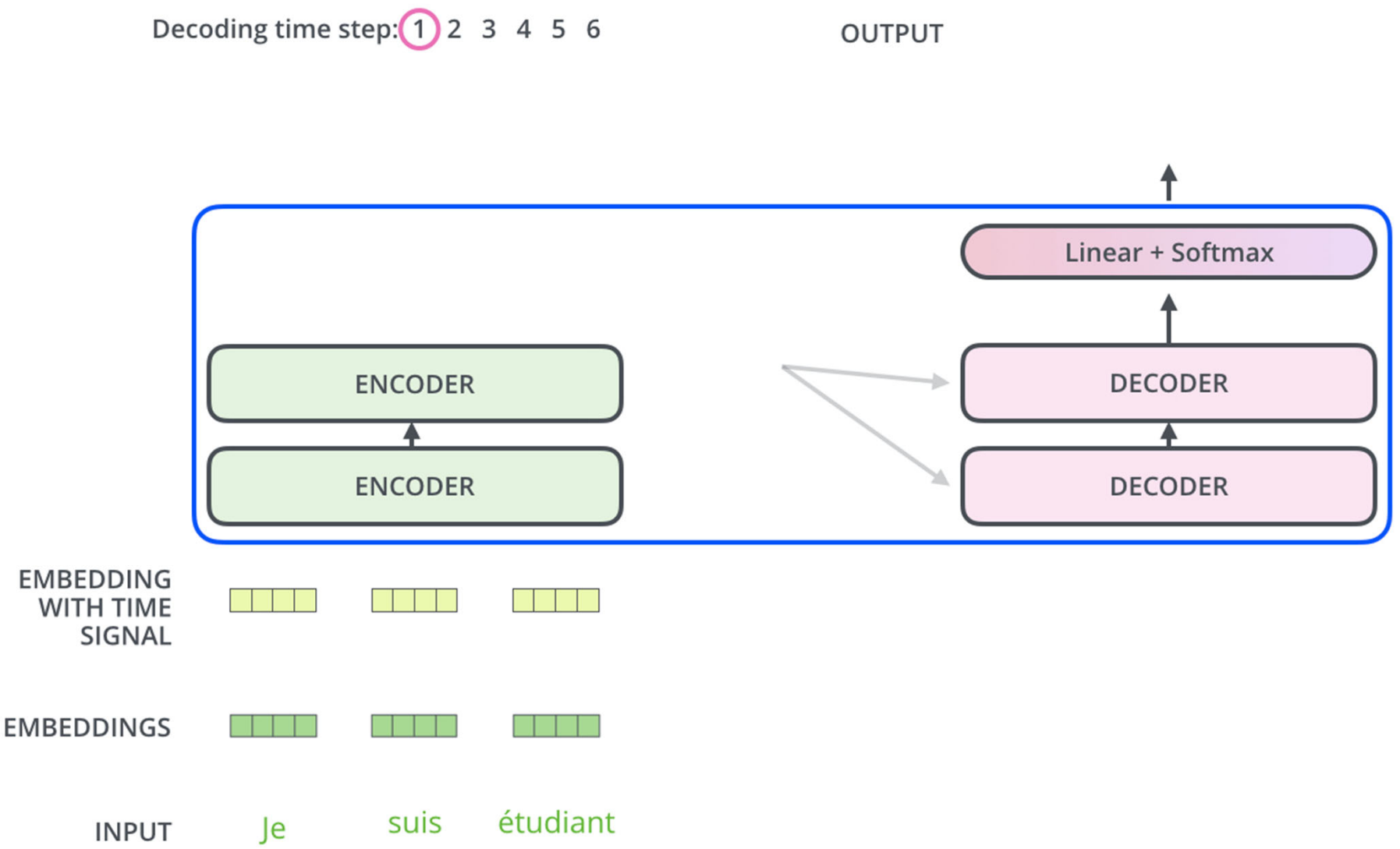
$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \ldots\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles, or a sentence is made out of tokens ...

Data distribution is modeled as:

$$P(\mathbf{X}) = \prod_{t=0}^{T} P(\mathbf{x}_{t+1} \mid \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_t)$$

Transformers as Autoregressive Models



Generative models (to be discussed)

- **Autoregressive models [done]**
- Variational Autoencoders [TODO this/next week]
- Diffusion Models [TODO this/next week]
- Generative Adversarial Networks [lower priority, after Easter]

Announcements

- **Lecture to replenish lost hours**

  Fri Apr 11, 3-5pm (will also discuss midterm as well)
  Room: [145Π58]

- Assignment 1 to be released next week

- List of papers to choose for presentation