AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

# Introduction to Deep Learning

# Transformers : Self Attention

- Google Brain , University of Toronto 2017 [Vaswani et al. 2017].

# Attention Is All You Need

**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

## 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.

# Why Transformers?

- Recurrent architectures rely on sequential processing of input at the encoding step that results in computational inefficiency, as the processing cannot be parallelized
- Transformer architecture completely eliminates sequential processing and recurrent connections It relies only on **self attention m**echanism to capture global dependencies between input and output
- Significant **parallel processing**, **shorter training time** and **higher accuracy** for Machine Translation without any recurrent component

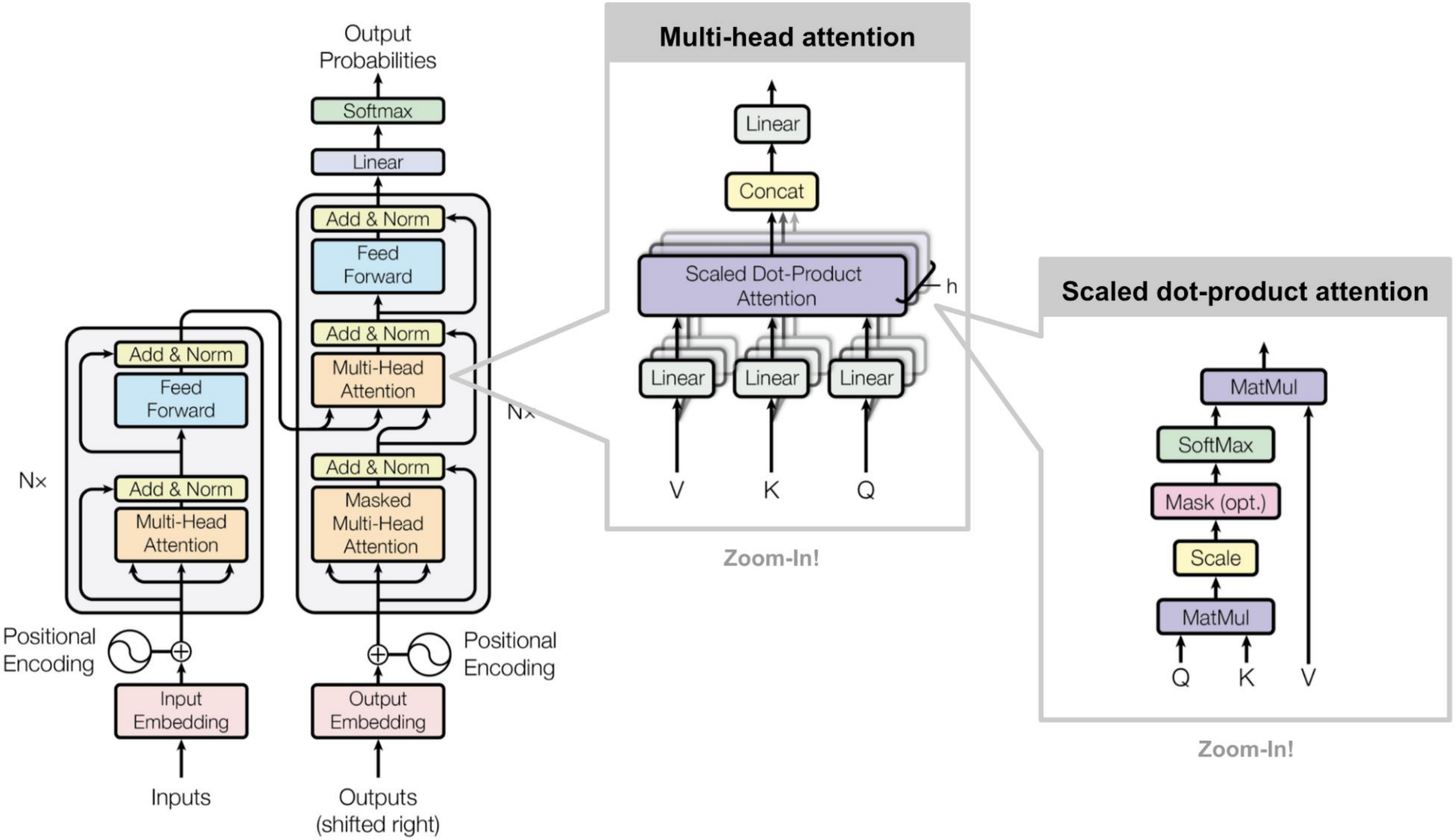
Challenges of RNN

- Long Range Dependencies
- Gradient vanishing and Exploding
- Large # of training steps
- Recurrence prevents parallel computation

Transformer Networks

- Facilitate long range dependencies
- No gradient vanishing and explosion
- Fewer training steps
- No recurrence and that facilitate parallel computation

# Transformers

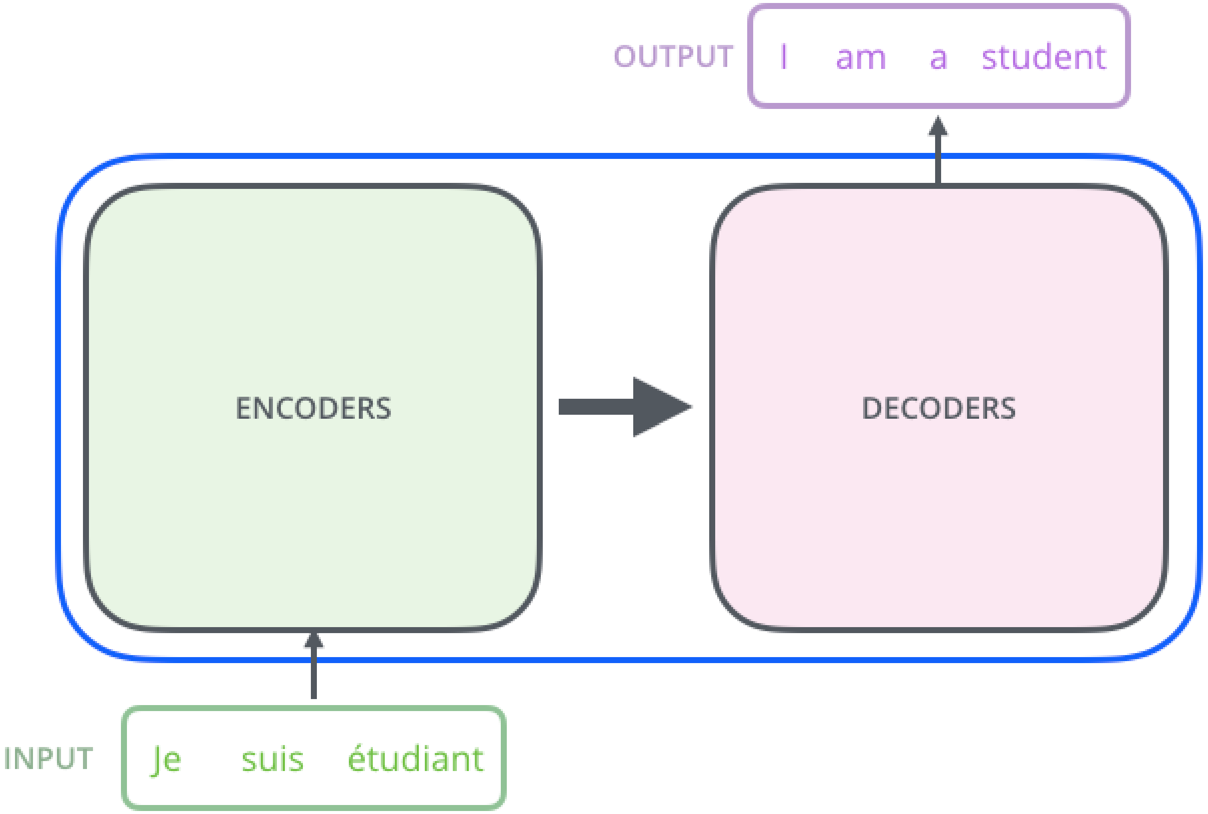


# Transformers – Machine Translation example


In a machine translation application, it would take a sentence in one language, and output its translation in another.

Courtesy : jalammar.github.io

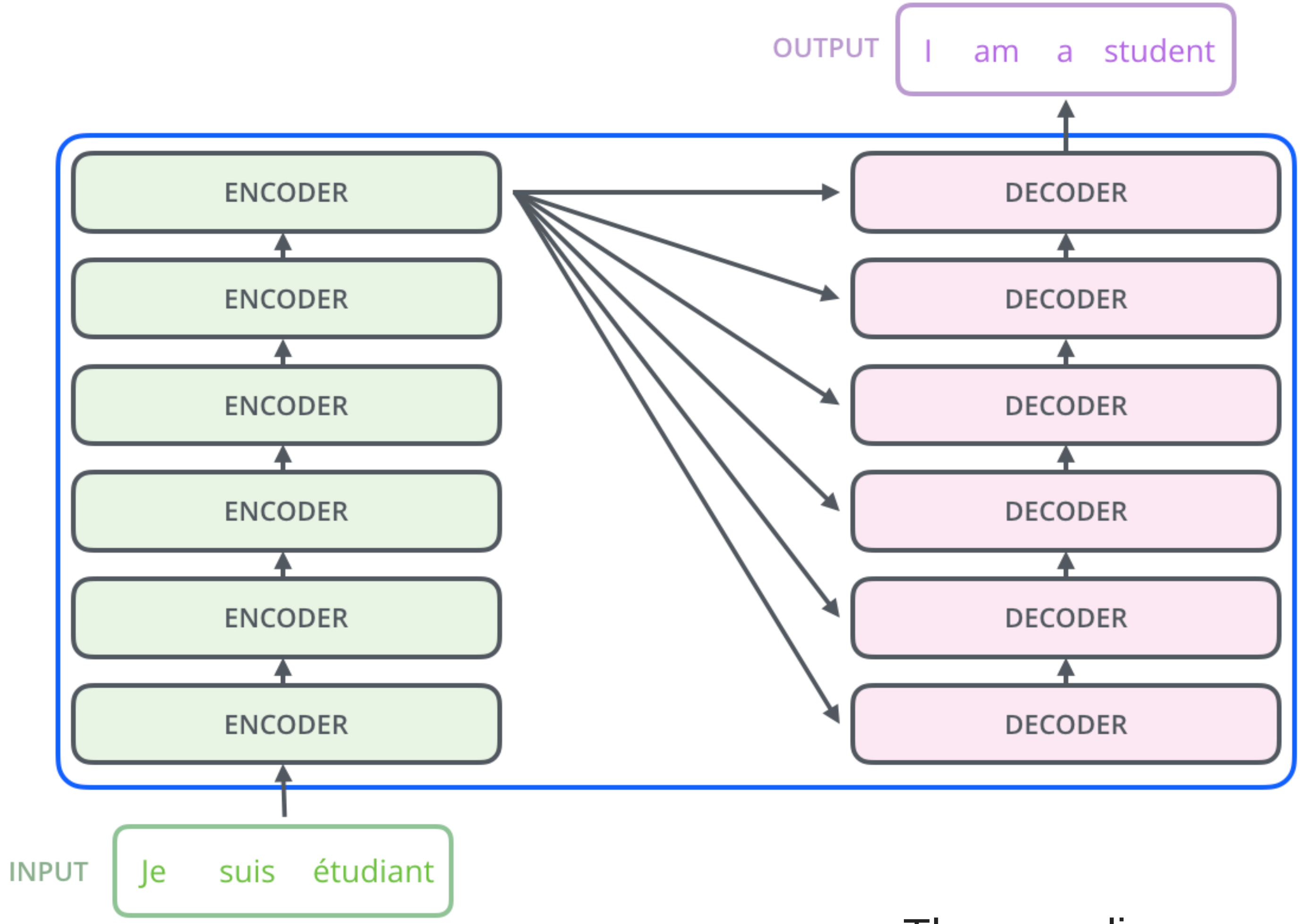
# Transformers – Machine Translation example




An encoding component, a decoding component, and connections between them.

http://jalammar.github.io

# Transformers – Machine Translation example



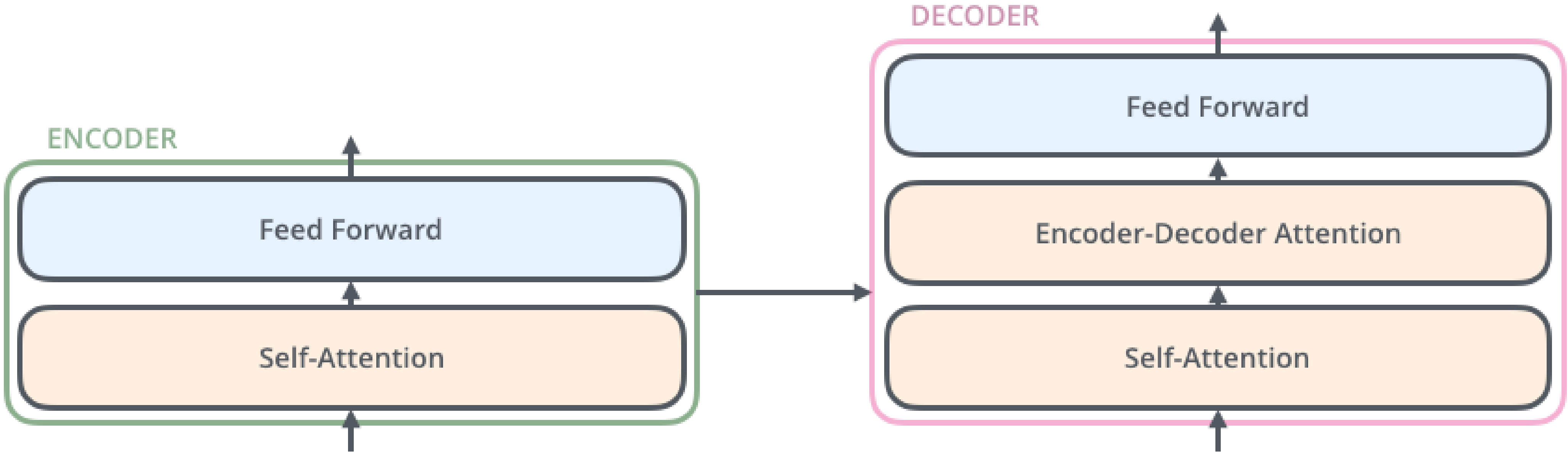

The encoders are all identical in structure (yet they do not share weights).

Each one is broken down into two sub-layers:

The encoding component is a stack of encoders .The decoding component is a stack of decoders of the same number.

http://jalammar.github.io

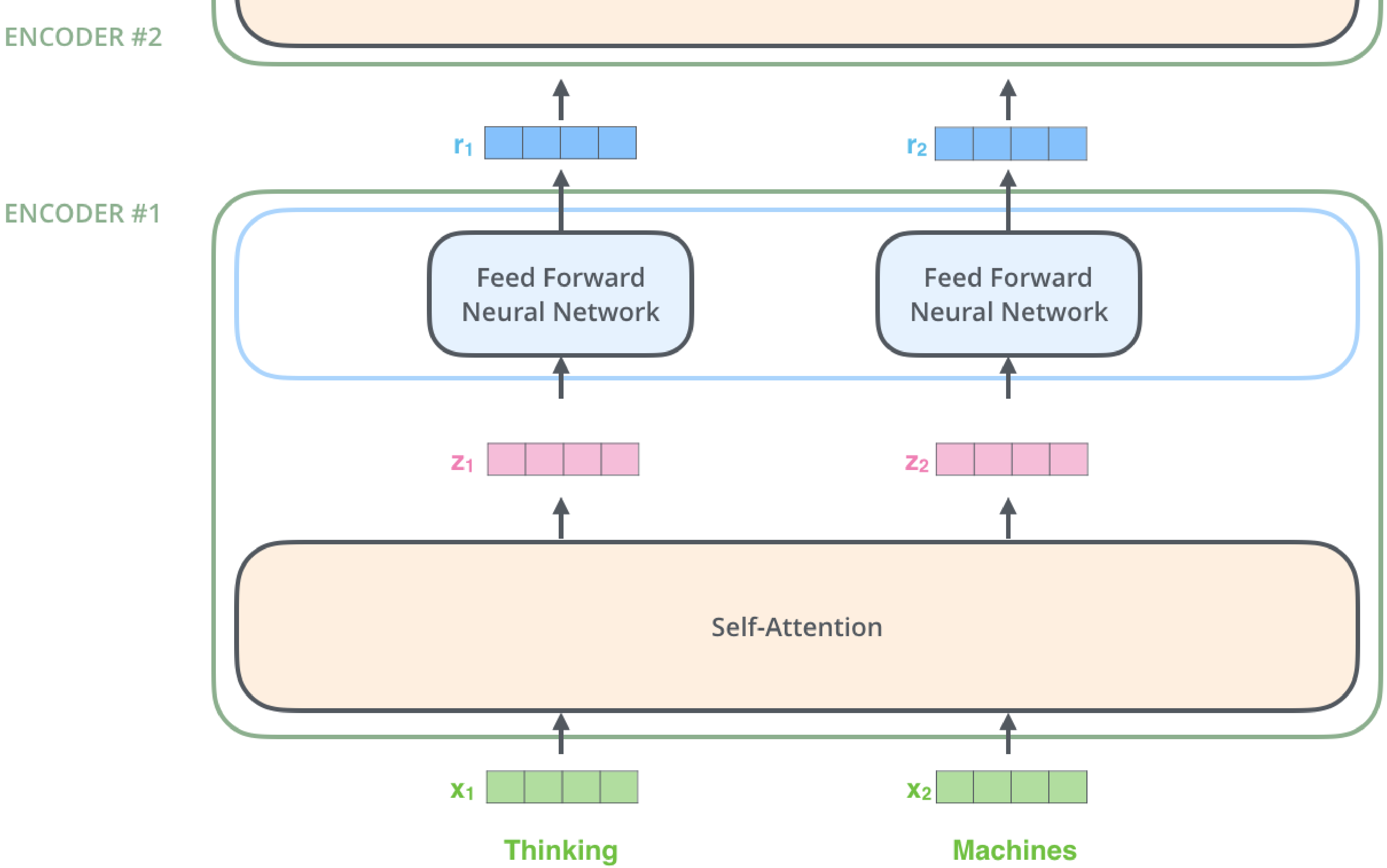
# Transformers – Machine Translation example




The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.

The outputs of the self-attention layer are fed to a feed-forward neural network.

http://jalammar.github.io


ENCODER #2
$r_1$
$r_2$
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
$z_1$
$z_2$
Self-Attention
$x_1$
$x_2$
Thinking
Machines

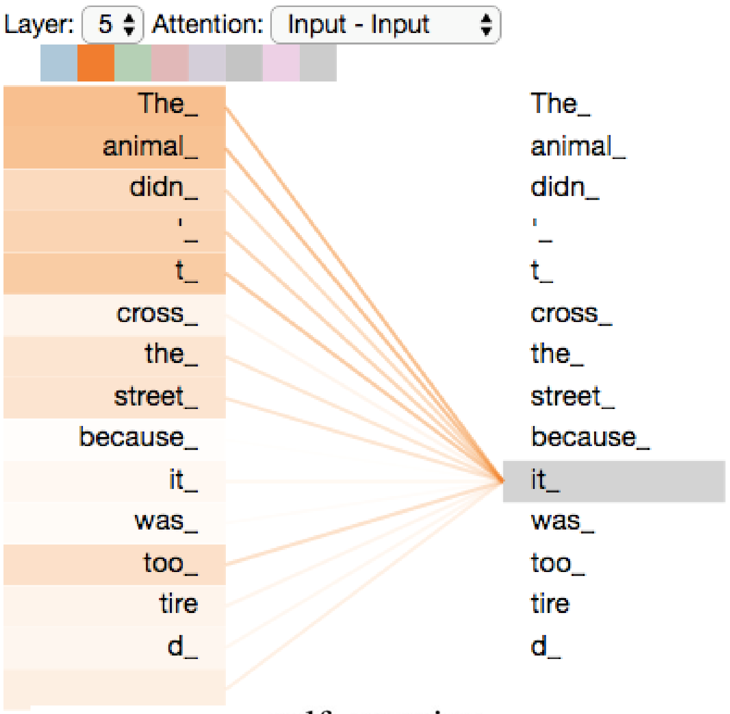
# Self-Attention

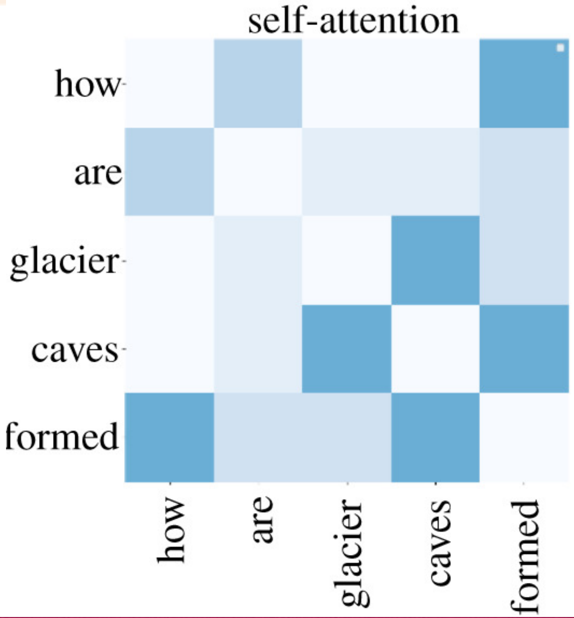
**The animal didn't cross the street because it was too tired"**

What does **"it"** in this sentence refer to? Is it referring to the ***street*** or to the ***animal***? It's a simple question to a human, but not as simple to an algorithm.

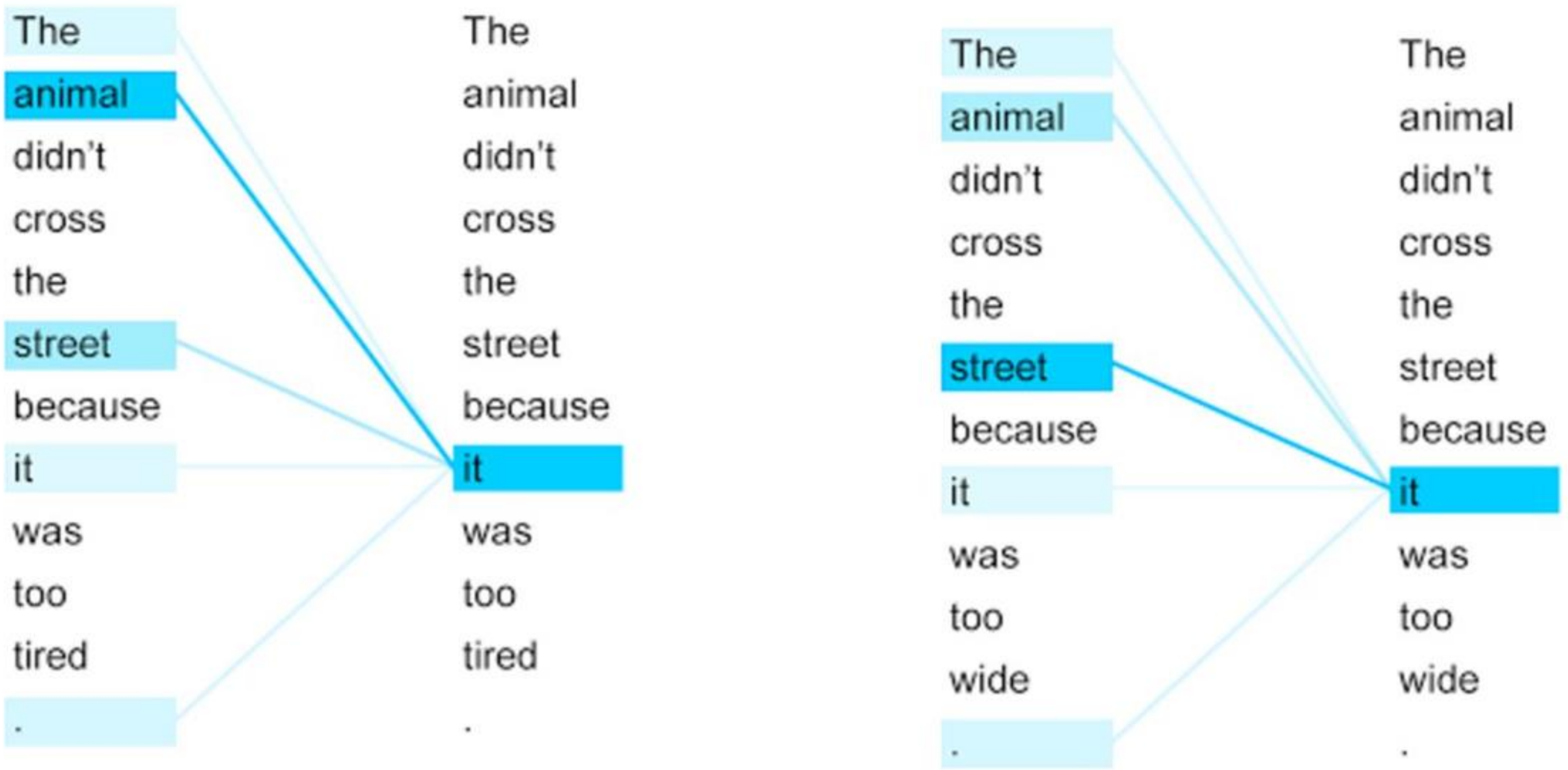
When the model is processing the word "it", self-attention allows it to associate "it" with "animal".

Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing.



# Self attention



# Self Attention

- **Query vector ( qi)**
  - what we are looking for in the sequence
- **Key vector (ki)**

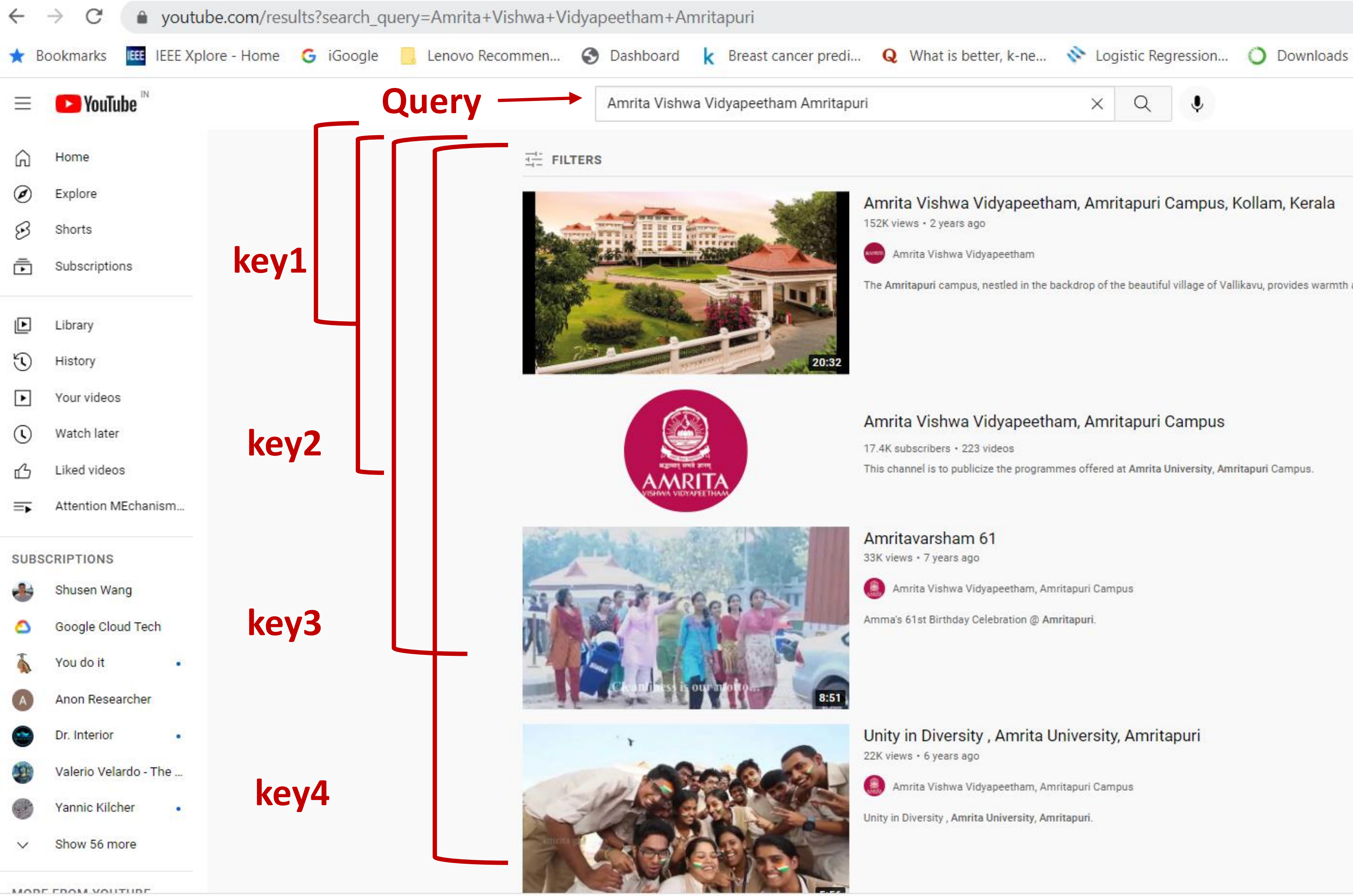
  what the element is “offering”,

- **Value vector (vi)**

  This feature vector is the one
  we want to average over

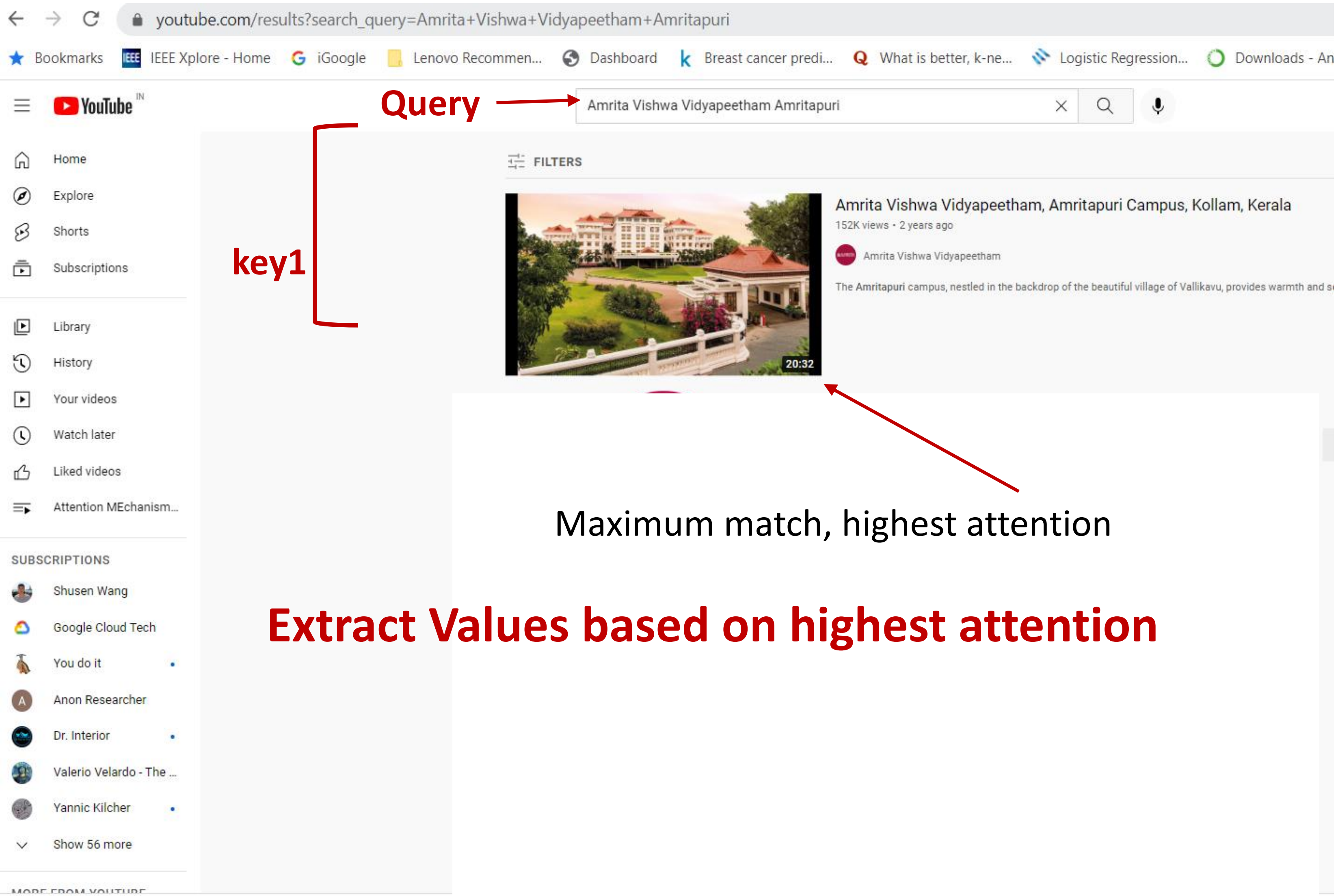
- **Score**

  To rate which elements we want to
  pay attention to



# Steps in Self Attention

- **Query vector ( qi)**
- **Key vector (ki)**
- **Value vector (vi)**



# Steps in Self Attention –

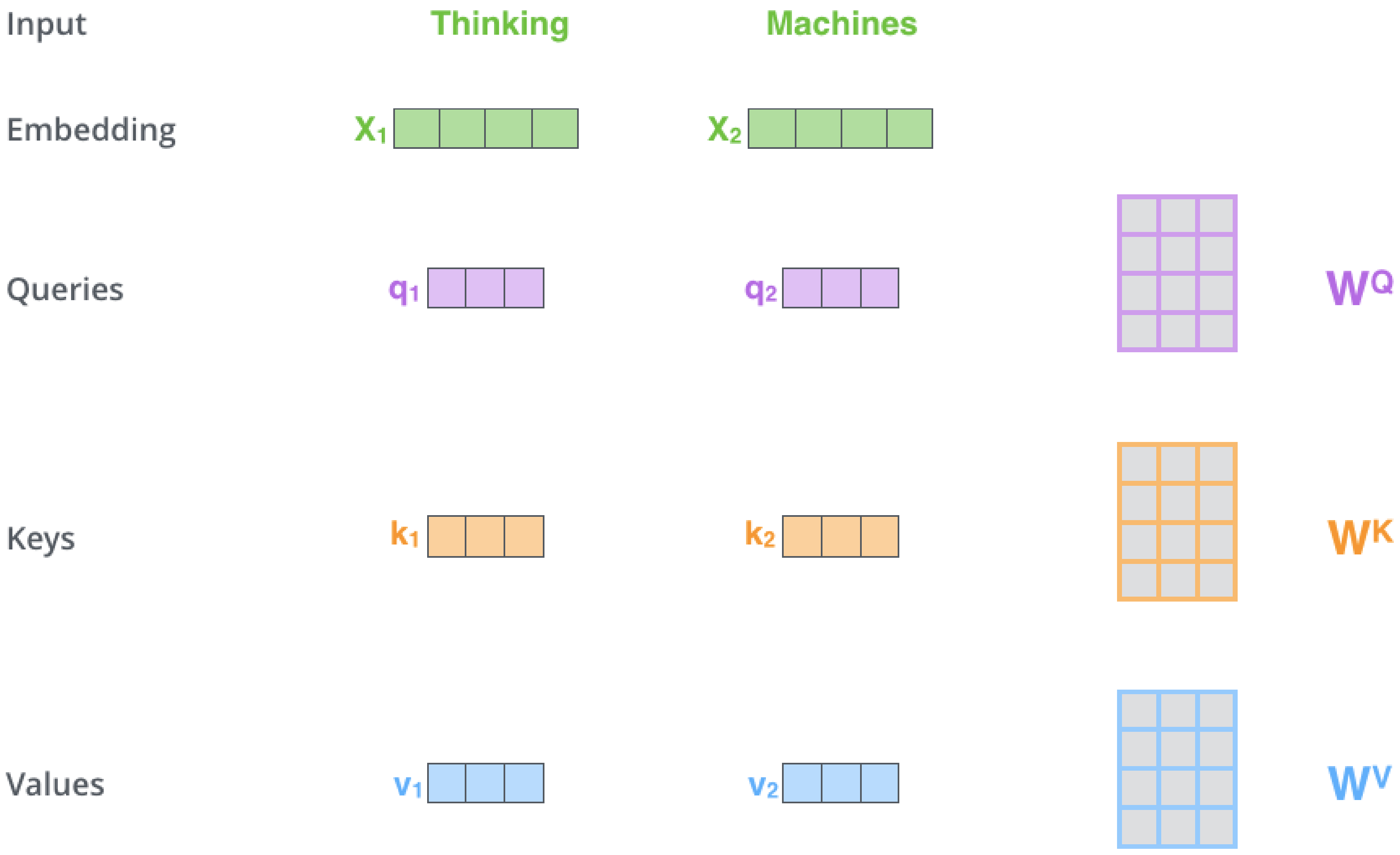
## First step

Create 3 vectors

- **Query vector ( qi)**
- **Key vector (ki)**
- **Value vector (vi)**

From each of the encoder's input vectors –ie the embedding of each word).

These 3 vectors are created by multiplying the embedding by three matrices that we trained during the training process

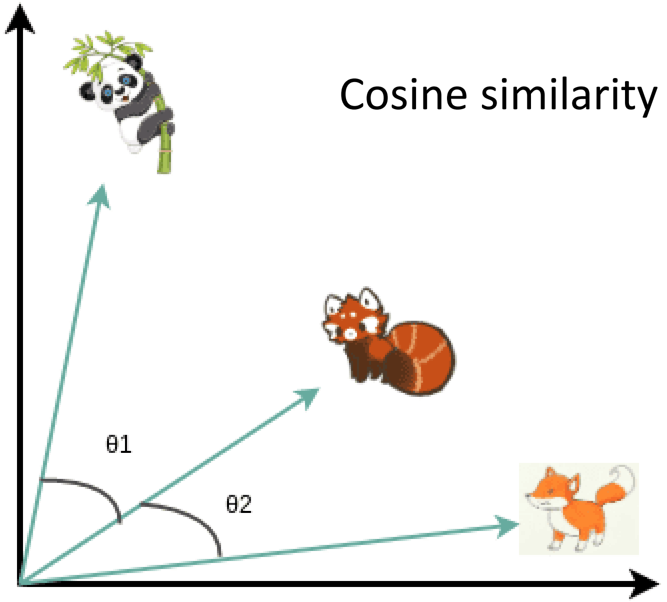


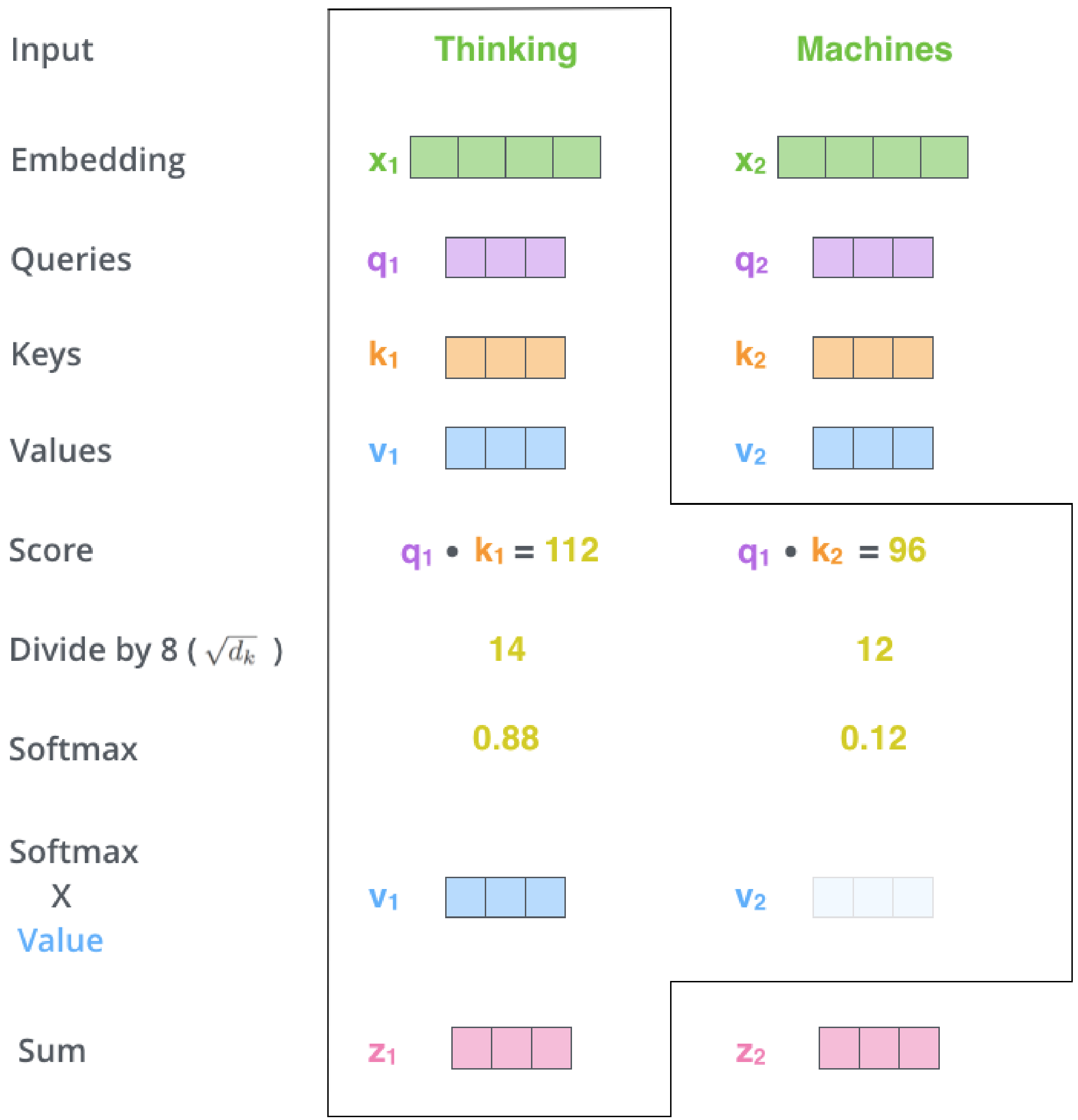
# Steps in Self Attention

The **second step**

- Calculate a score : dot product of the query vector with the key vector of the respective word

$q_1 \bullet k_1 = 112$ $\qquad$ $q_1 \bullet k_2 = 96$



# Steps in Self Attention

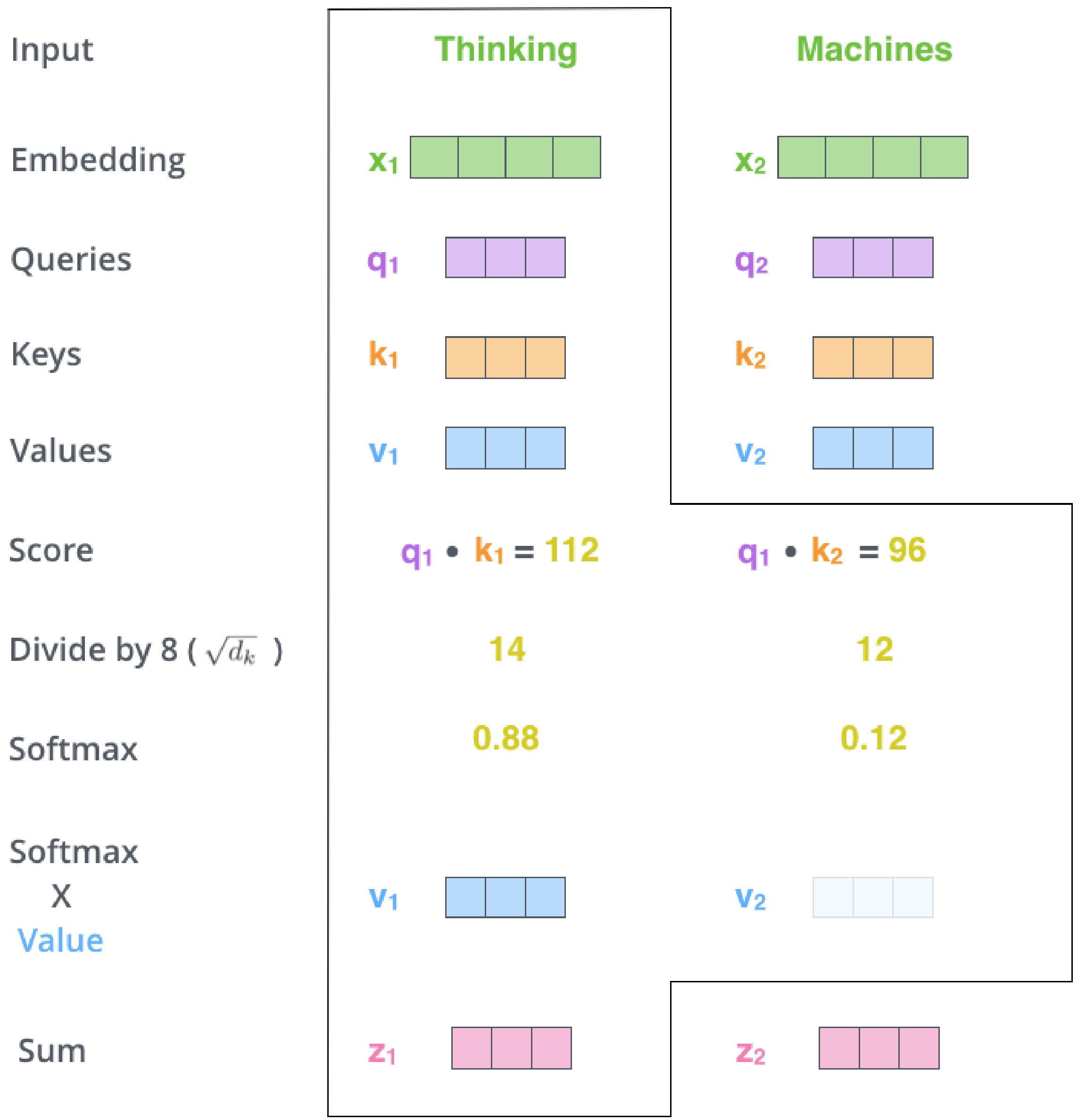
## Third step

| Word | q vector | k vector | v vector | score | score / 8 |
|---|---|---|---|---|---|
| thinking | $q_1$ | $k_1$ | $v_1$ | $q_1 \cdot k_1$ | $q_1 \cdot k_1 / 8$ |
| Machines | | $k_2$ | $v_2$ | $q_1 \cdot k_2$ | $q_1 \cdot k_2 / 8$ |

### Divide by 8 ( $\sqrt{d_k}$ )

Normalizing : This leads to having more stable gradients

The square root of the dimension of the key vectors used in the paper – 64.



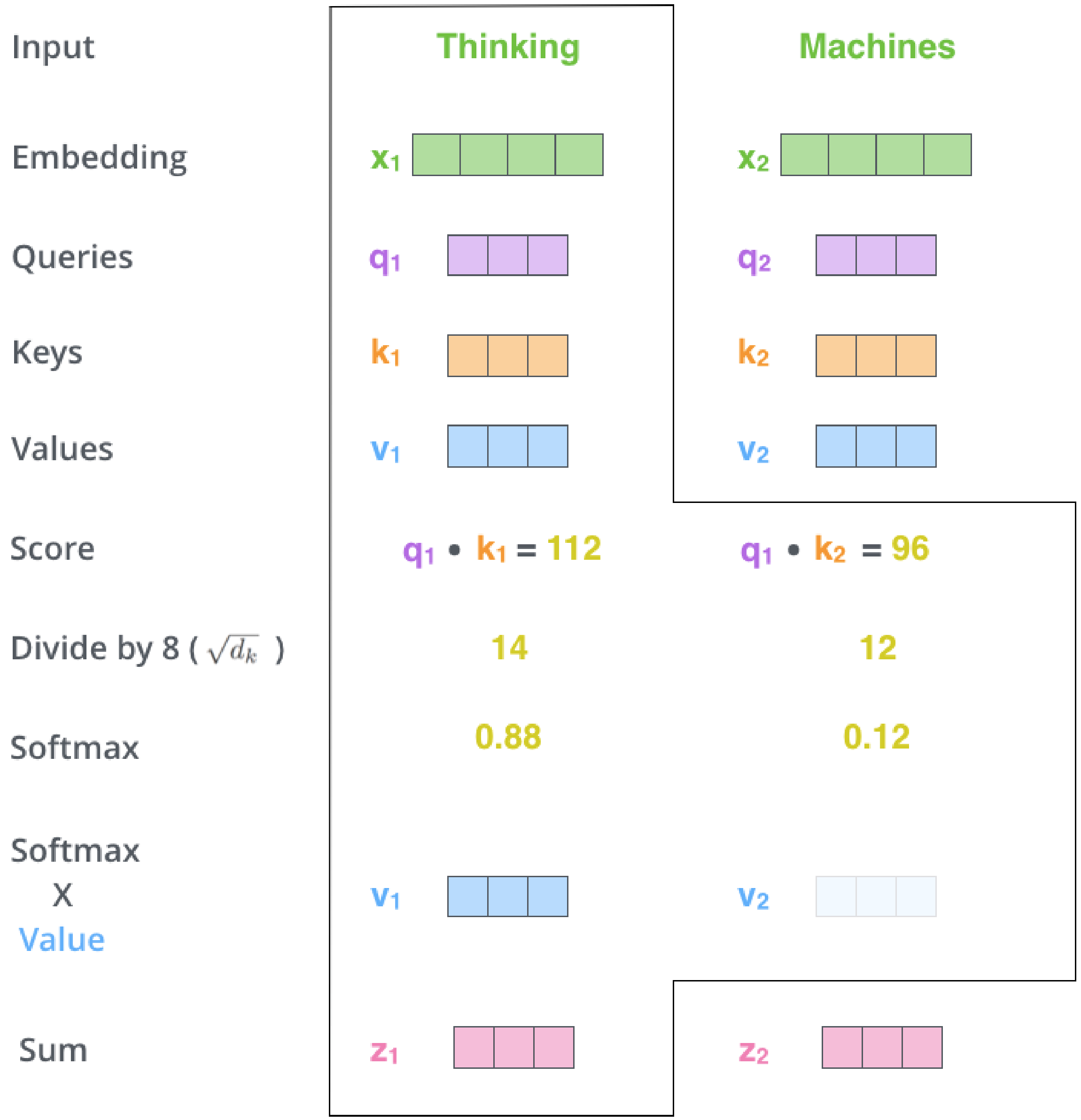
# Steps in Self Attention –

**Fourth step**

| Word | q vector | k vector | v vector | score | score / 8 | Softmax | Softmax * v | Sum |
|---|---|---|---|---|---|---|---|---|
| thinking | $q_1$ | $k_1$ | $v_1$ | $q_1 \cdot k_1$ | $q_1 \cdot k_1 / 8$ | $x_{11}$ | $x_{11} * v_1$ | $z_1$ |
| Machines | | $k_2$ | $v_2$ | $q_1 \cdot k_2$ | $q_1 \cdot k_2 / 8$ | $x_{12}$ | $x_{12} * v_2$ | |

Multiply each value vector by the softmax score

The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words



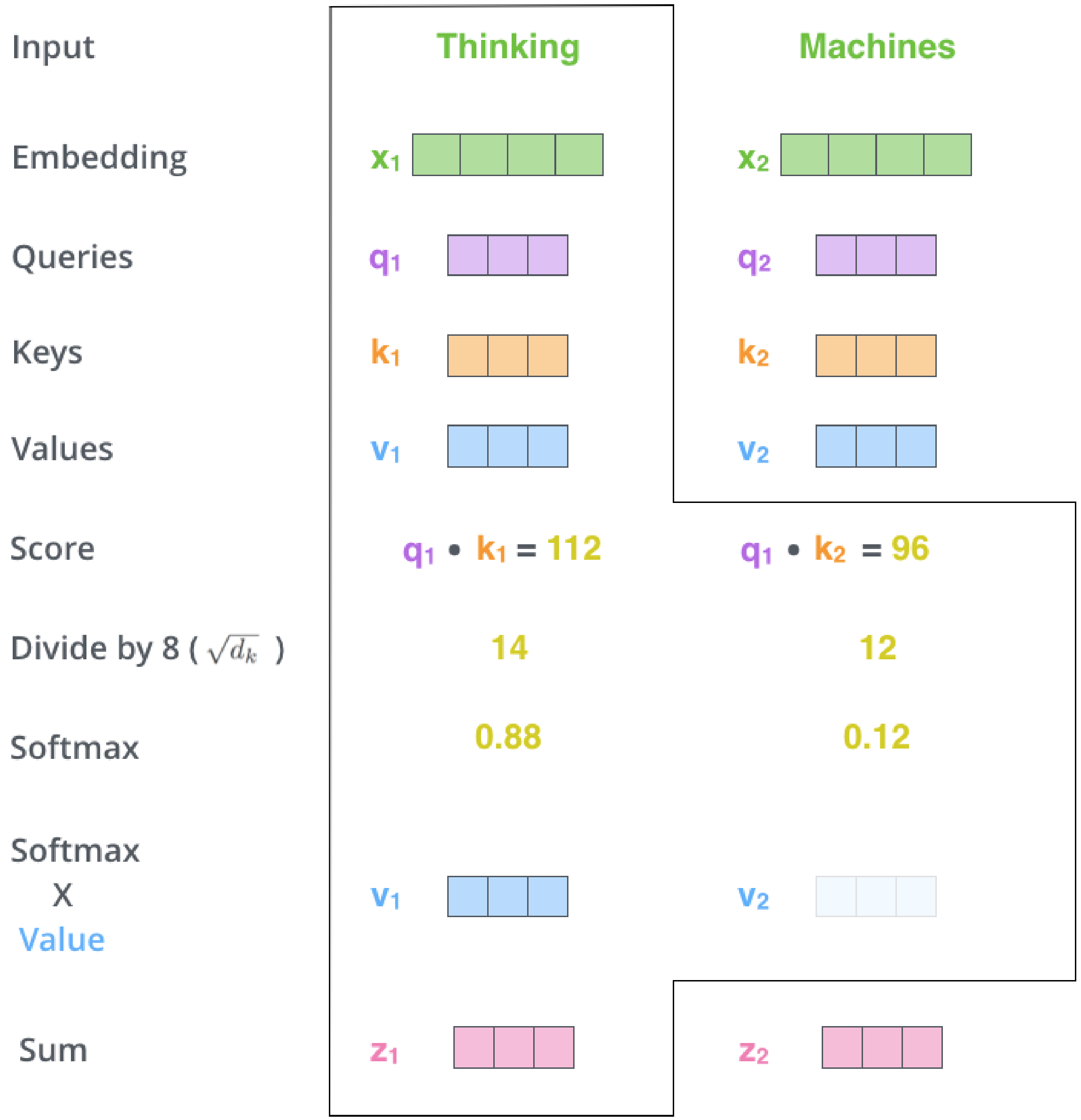
# Steps in Self Attention –

**Fifth step**

| Word | q vector | k vector | v vector | score | score / 8 | Softmax | Softmax * v | Sum |
|---|---|---|---|---|---|---|---|---|
| thinking | $q_1$ | $k_1$ | $v_1$ | $q_1 . k_1$ | $q_1 . k_1 / 8$ | $x_{11}$ | $x_{11} * v_1$ | $z_1$ |
| Machines | | $k_2$ | $v_2$ | $q_1 . k_2$ | $q_1 . k_2 / 8$ | $x_{12}$ | $x_{12} * v_2$ | |

Pass the result through a softmax operation.
Softmax normalizes the scores so they're all positive and add up to 1.

Sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

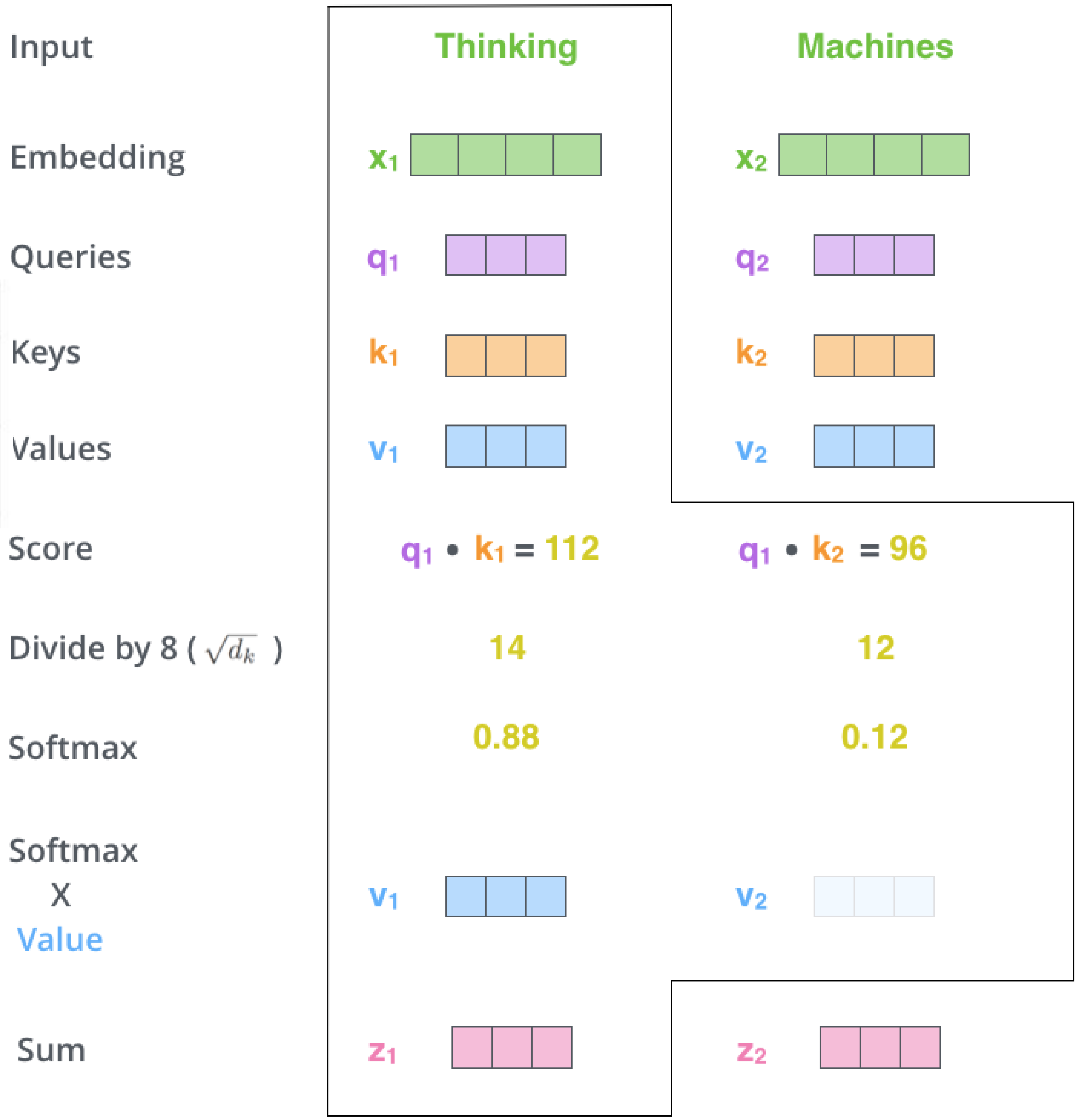


# Steps in Self Attention –

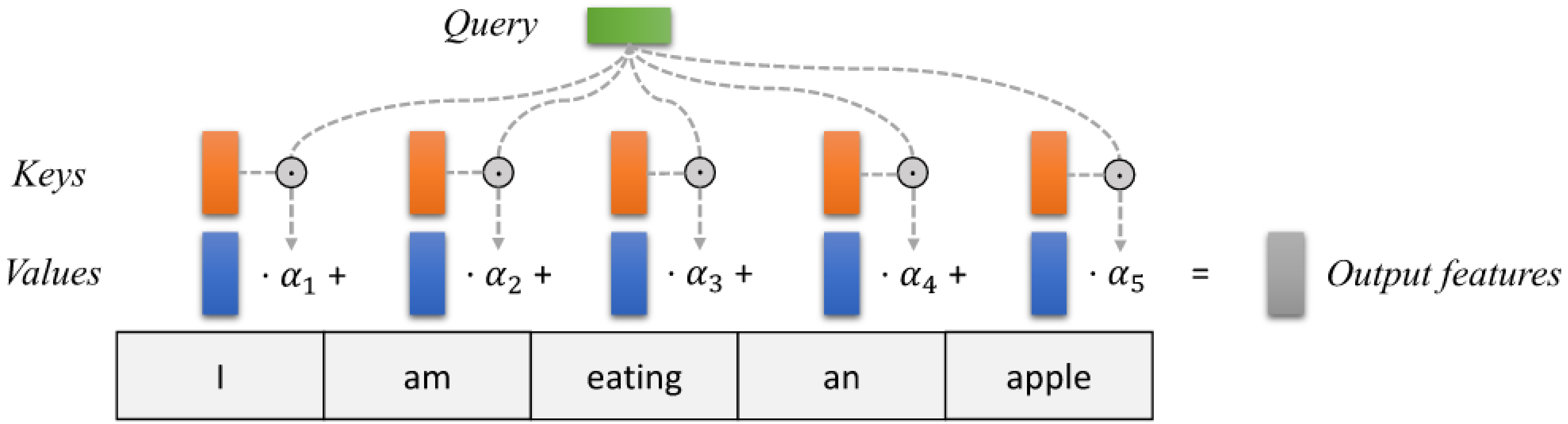
**Sixth Step**

| Word | q vector | k vector | v vector | score | score / 8 | Softmax | Softmax * v | Sum |
|---|---|---|---|---|---|---|---|---|
| thinking | | $k_1$ | $v_1$ | $q_2 . k_1$ | $q_2 . k_1 / 8$ | $x_{21}$ | $x_{21} * v_1$ | |
| Machines | $q_2$ | $k_2$ | $v_2$ | $q_2 . k_2$ | $q_2 . k_2 / 8$ | $x_{22}$ | $x_{22} * v_2$ | $z_2$ |

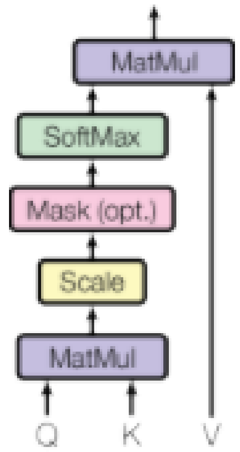
We need to score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.



# Self Attention –



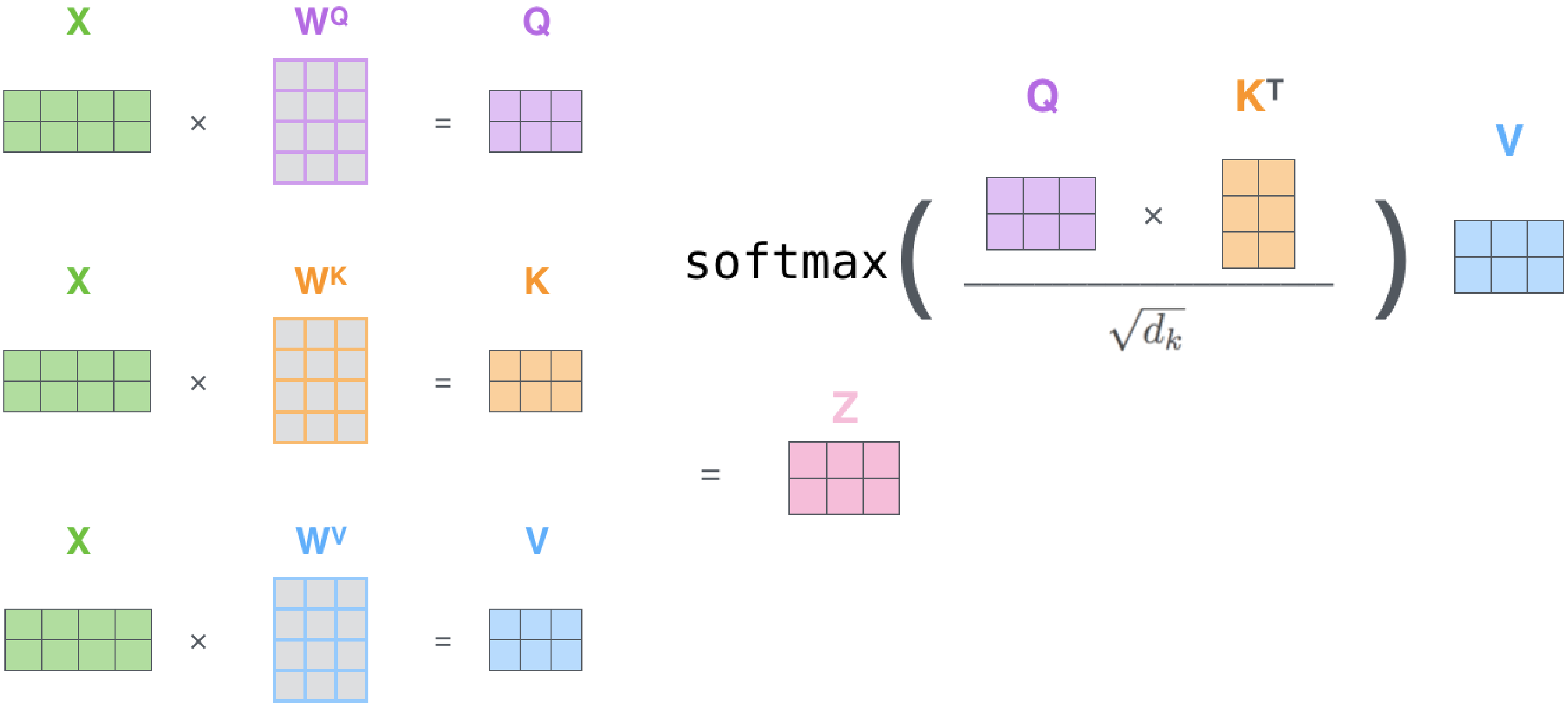

Scaled Dot-Product Attention



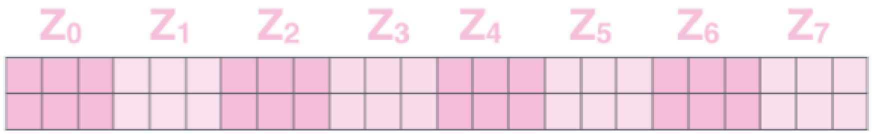

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\alpha_i = \frac{\exp\left(f_{attn}\left(\mathrm{key}_i, \mathrm{query}\right)\right)}{\sum_j \exp\left(f_{attn}\left(\mathrm{key}_j, \mathrm{query}\right)\right)}, \quad \mathrm{out} = \sum_i \alpha_i \cdot \mathrm{value}_i$$

# Matrix Calculation of Self-Attention



# “Multi-headed” attention



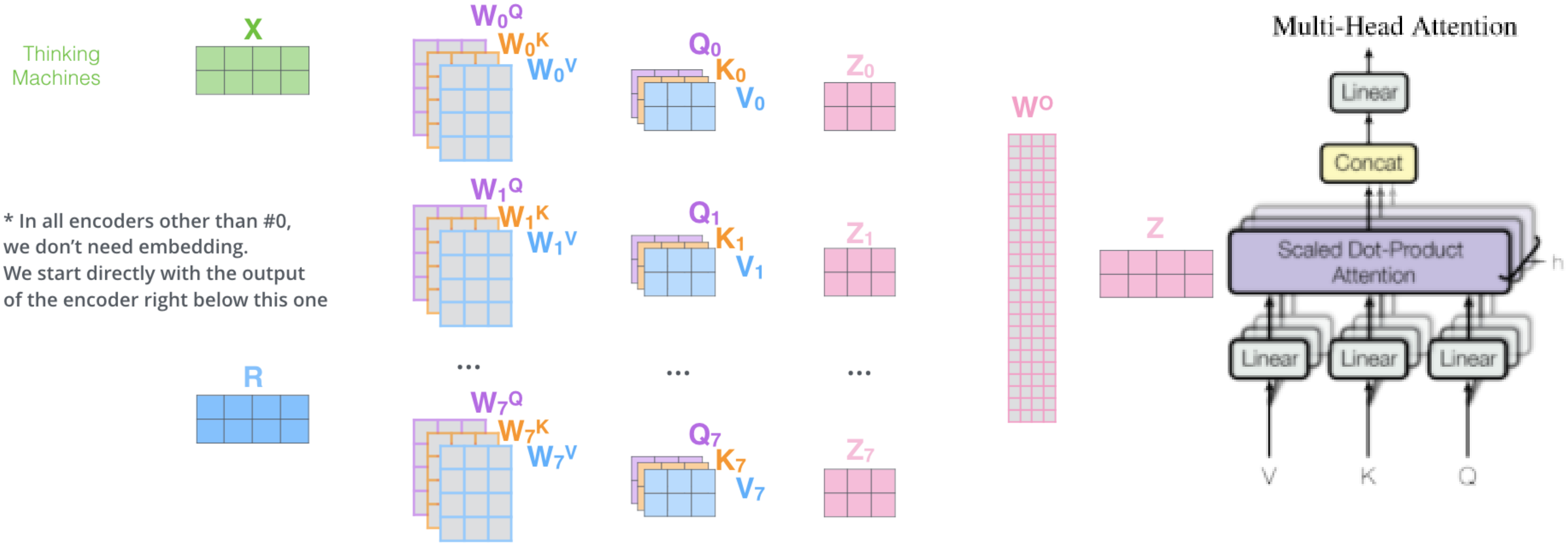

1) This is our input sentence*

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

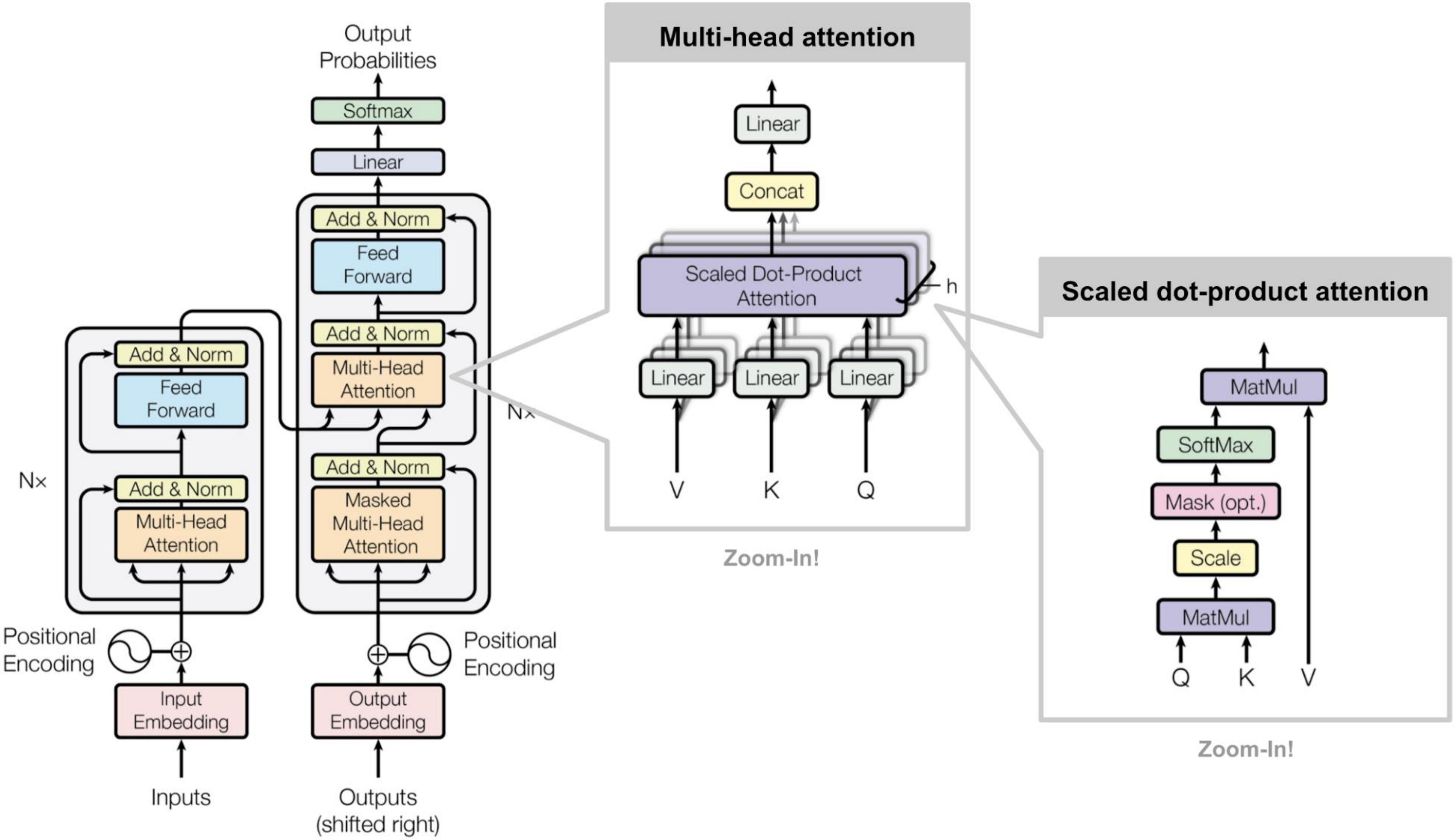



* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one

multiple “representation subspaces”

multiple sets of Query/Key/Value weight matrices

# Transformers

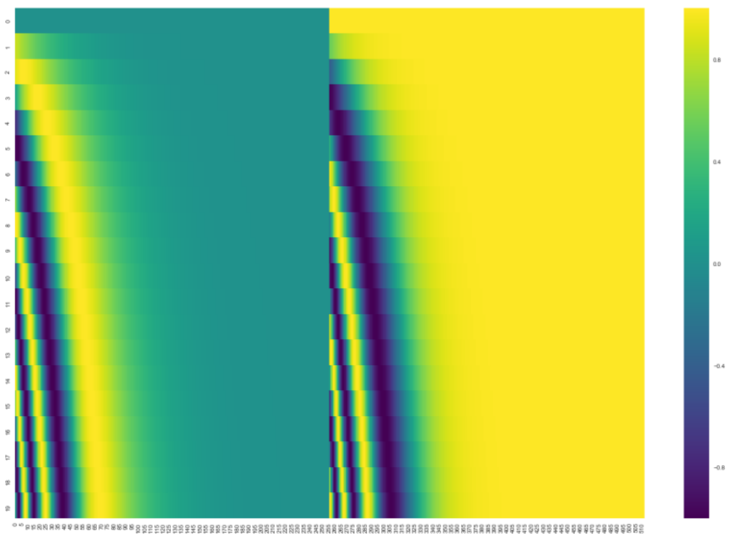


# Representing The Order of The Sequence Using Positional Encoding

Multi-Head Attention block is permutation-equivariant, w.r.t its inputs and cannot distinguish whether an input comes before another one in the sequence or not.- hence positional encoding

Permutation-equivariant, :means that if we switch two input elements in the sequence, e.g. X 1 ↔ X 2 the output is exactly the same besides the elements 1 and 2 switched


A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns)



$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{model}}}}\right) & \text{if } i \bmod 2 = 0 \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{model}}}}\right) & \text{otherwise} \end{cases}$$

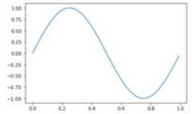
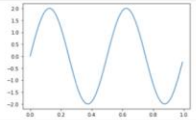
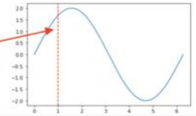
$PE_{(pos,i)}$ represents the position encoding at position pos in the sequence, and hidden dimensionality i.

Vaswani et al 2017

# Positional Encoding Layer in Transformers

| Sequence | Index of token | Positional Encoding Matrix | | | |
|---|---|---|---|---|---|
| I | 0 | $P_{00}$ | $P_{01}$ | … | $P_{0d}$ |
| am | 1 | $P_{10}$ | $P_{11}$ | … | $P_{1d}$ |
| a | 2 | $P_{20}$ | $P_{21}$ | … | $P_{2d}$ |
| Robot | 3 | $P_{30}$ | $P_{31}$ | … | $P_{3d}$ |

Positional Encoding Matrix for the sequence 'I am a robot'

| Equation | Graph | Frequency | Wavelength |
|---|---|---|---|
| $\sin(2\pi t)$ |  | 1 | 1 |
| $\sin(2 * 2\pi t)$ |  | 2 | 1/2 |
| $\sin(t)$ |  | $1/2\pi$ | $2\pi$ |
| $\sin(ct)$ | Depends on c | $c/2\pi$ | $2\pi/c$ |

Positional Encoding Matrix with d=4, n=100

| Sequence | Index of token, k | i=0 | i=0 | i=1 | i=1 |
|---|---|---|---|---|---|
| I | 0 | $P_{00}=\sin(0) = 0$ | $P_{01}=\cos(0) = 1$ | $P_{02}=\sin(0) = 0$ | $P_{03}=\cos(0) = 1$ |
| am | 1 | $P_{10}=\sin(1/1) = 0.84$ | $P_{11}=\cos(1/1) = 0.54$ | $P_{12}=\sin(1/10) = 0.10$ | $P_{13}=\cos(1/10) = 1.0$ |
| a | 2 | $P_{20}=\sin(2/1) = 0.91$ | $P_{21}=\cos(2/1) = -0.42$ | $P_{22}=\sin(2/10) = 0.20$ | $P_{23}=\cos(2/10) = 0.98$ |
| Robot | 3 | $P_{30}=\sin(3/1) = 0.14$ | $P_{31}=\cos(3/1) = -0.99$ | $P_{32}=\sin(3/10) = 0.30$ | $P_{33}=\cos(3/10) = 0.96$ |

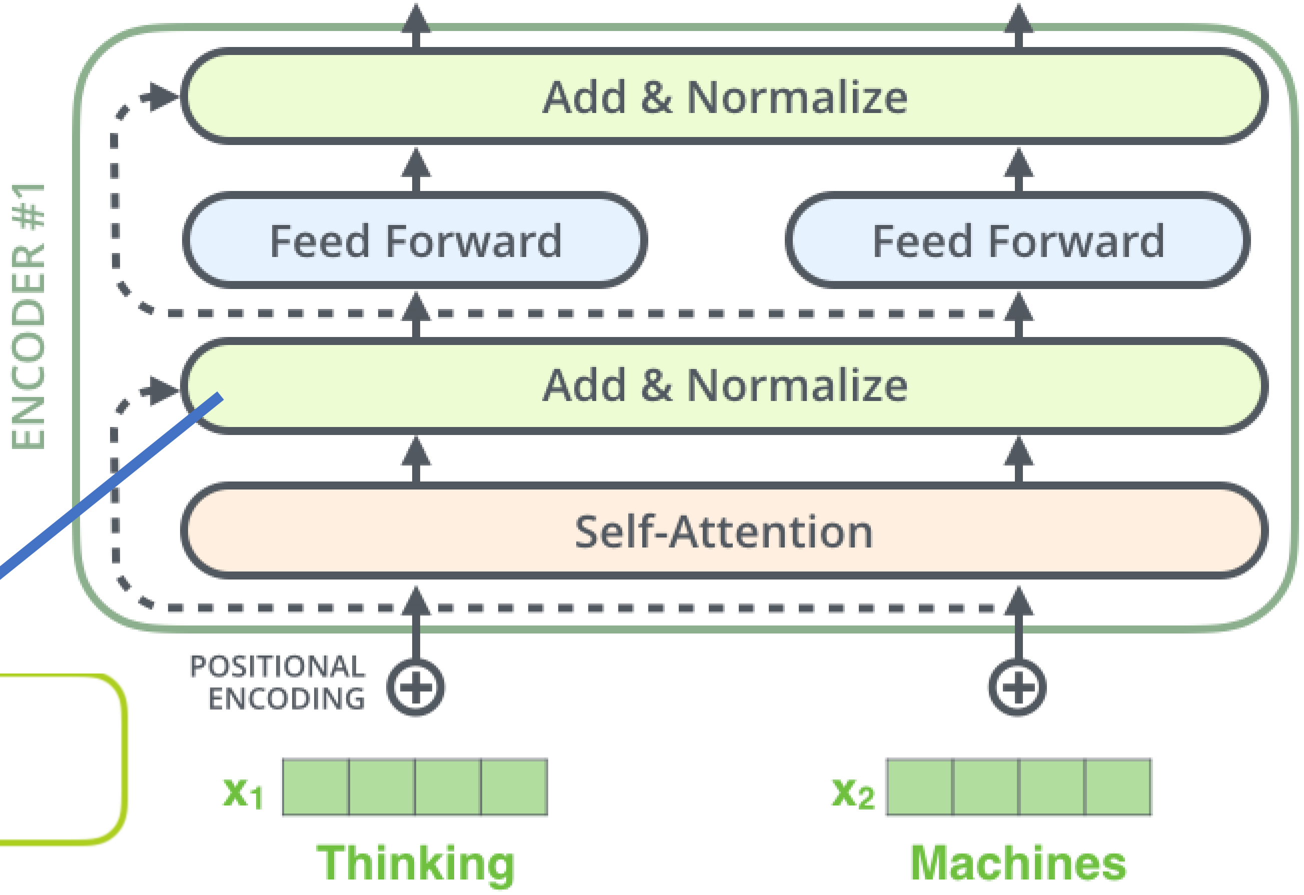
Positional Encoding Matrix for the sequence 'I am a robot'

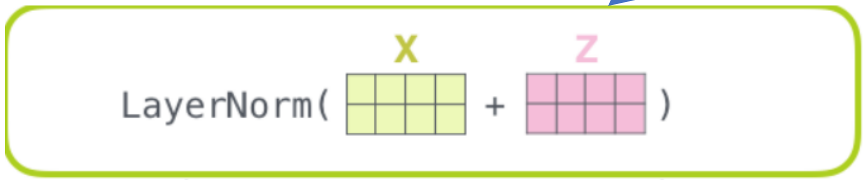
# Residual connection, Layer Norm, FFN

Each encoder has a

- Residual connection around it,
- Followed by a layer normalization step
- Feed forward : -deepens our network, employing linear layers to analyse patterns in the attention layers output.

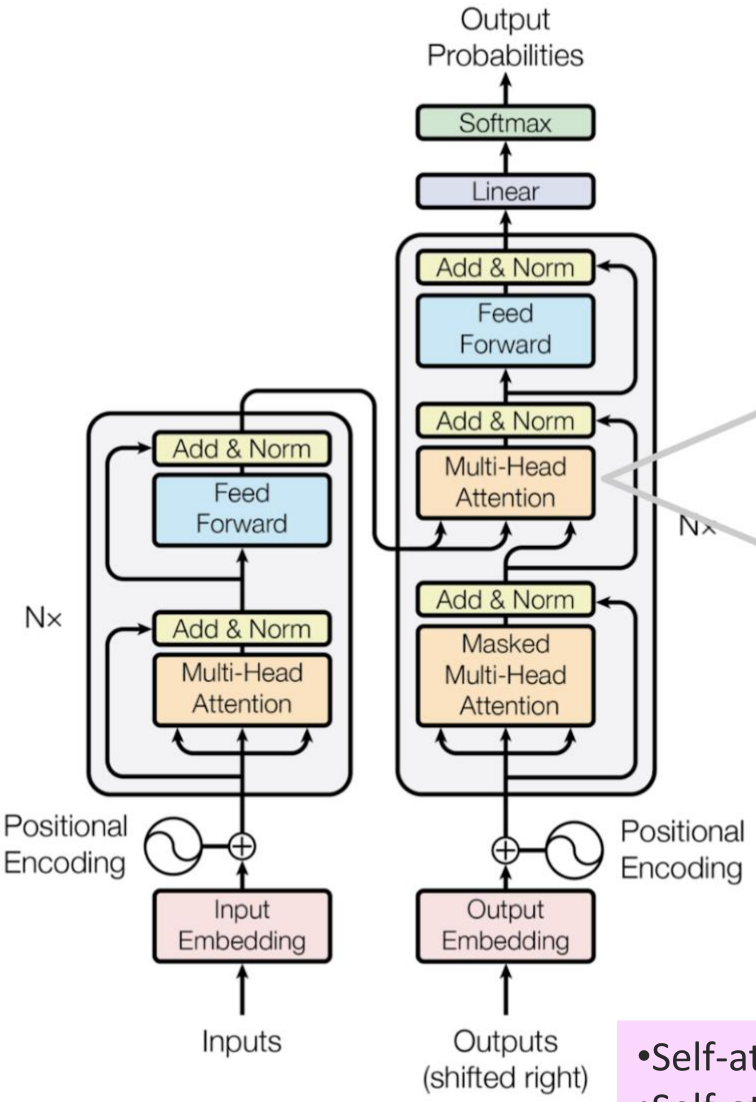
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$
$$x = \mathrm{LayerNorm}(x + \mathrm{FFN}(x))$$

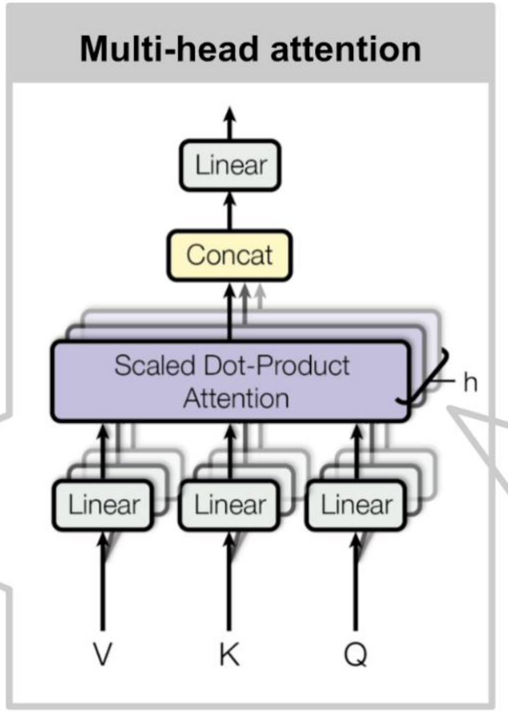


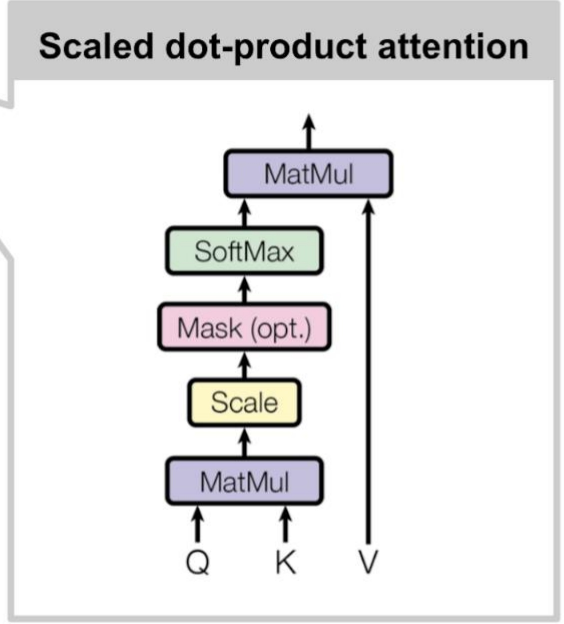
# Transformers

The Encoder-Decoder Attention is getting a representation of both the target sequence (from the Decoder Self-Attention) and a representation of the input sequence (from the Encoder stack).




Zoom-In!

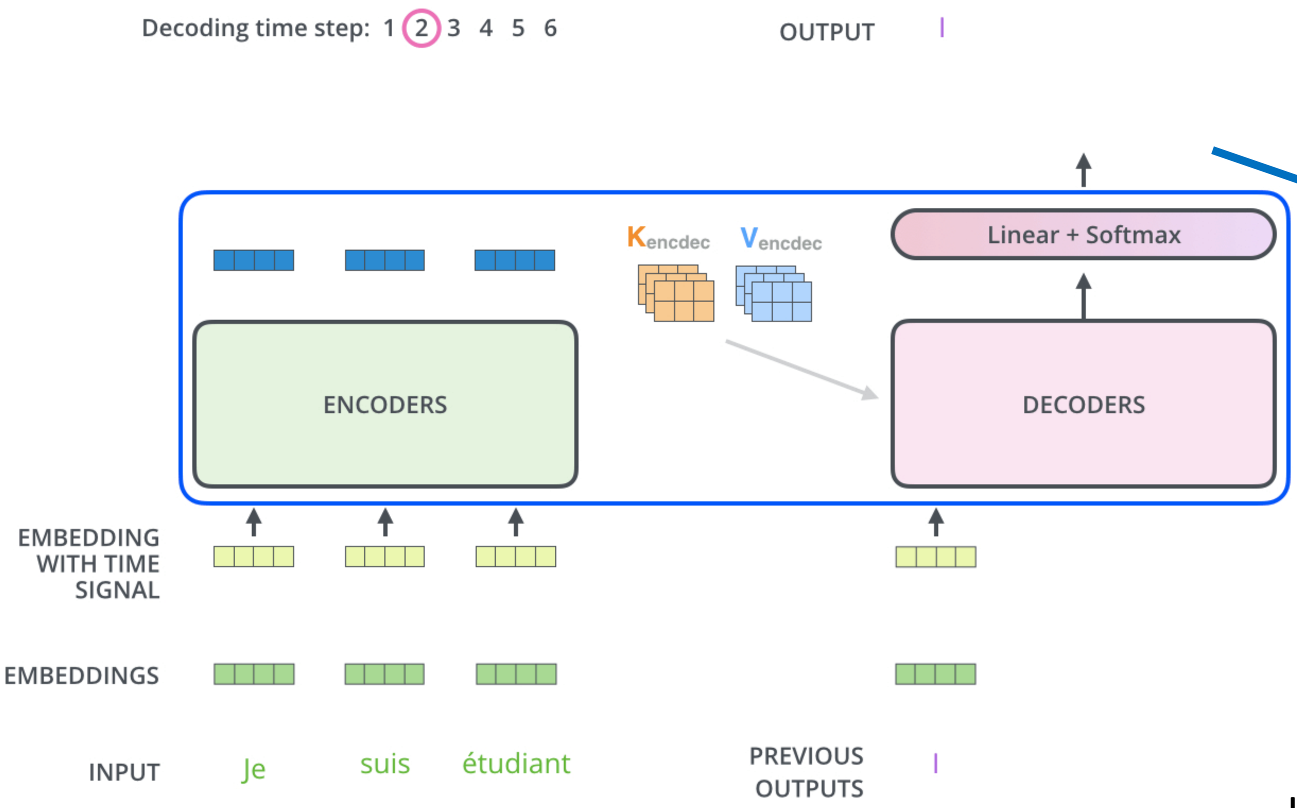


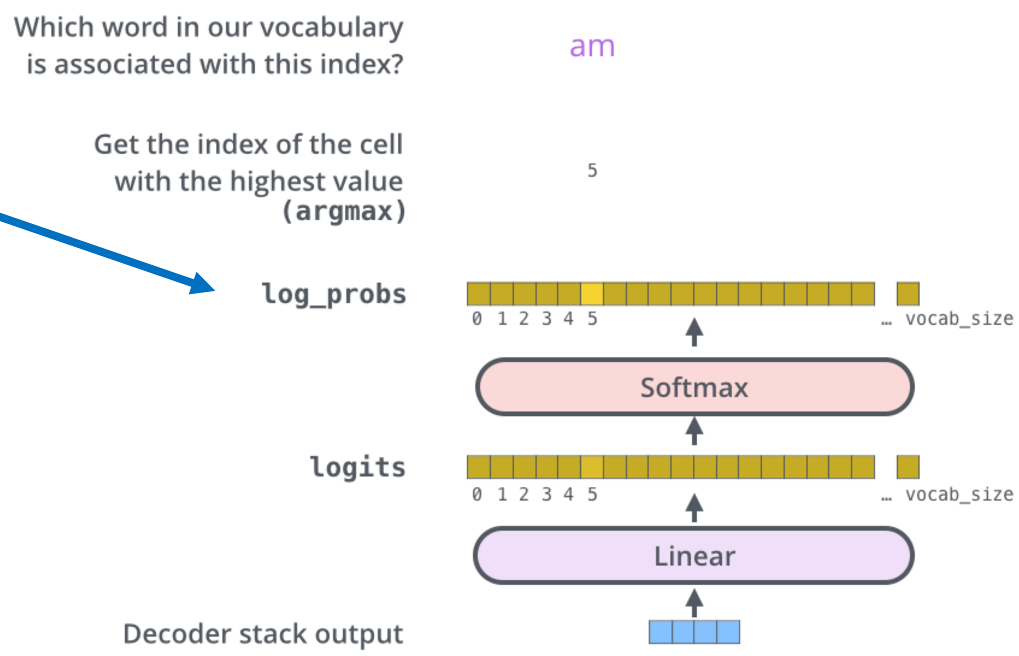

Zoom-In!

- Self-attention in the Encoder — the input sequence pays attention to itself
- Self-attention in the Decoder — the target sequence pays attention to itself
- Encoder-Decoder-attention in the Decoder — the target sequence pays attention to the input sequence

# Transformer

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




The self attention layers in the decoder operate in a slightly different way than the one in the encoder:

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

Transformer Mode

Encoder only
(**Autoencoding models**)
(e.g. for classification)

Decoder only
(**Autoregressive models**)
e.g., for language modelling

Encoder-Decoder (**Seq to seq models**)
e.g., for machine translation

**Autoencoding models**

- BERT
- ALBERT
- RoBERTa
- DistilBERT
- XLM
- XLM RoBERTa
- FlauBERT
- ELECTRA
- Longformer

**Autoregressive models**

- XLNET
- Reformer
- Transformer XL
- CTRL
- GPT
- GPT-2

**Seq-seq models**

- BART
- MarianMT
- T5
- CTRL
- GPT
- GPT-2

# Deep Learning Models with attention

**Natural Language Processing**
BART
(BERT)
RoBERTa
DistilBERT
Generative Pre-trained Transformer (GPT)
GPT-2 GPT-3
Transformer-XL
XLNet combines BERT and Transformer-XL

**Object Detection**

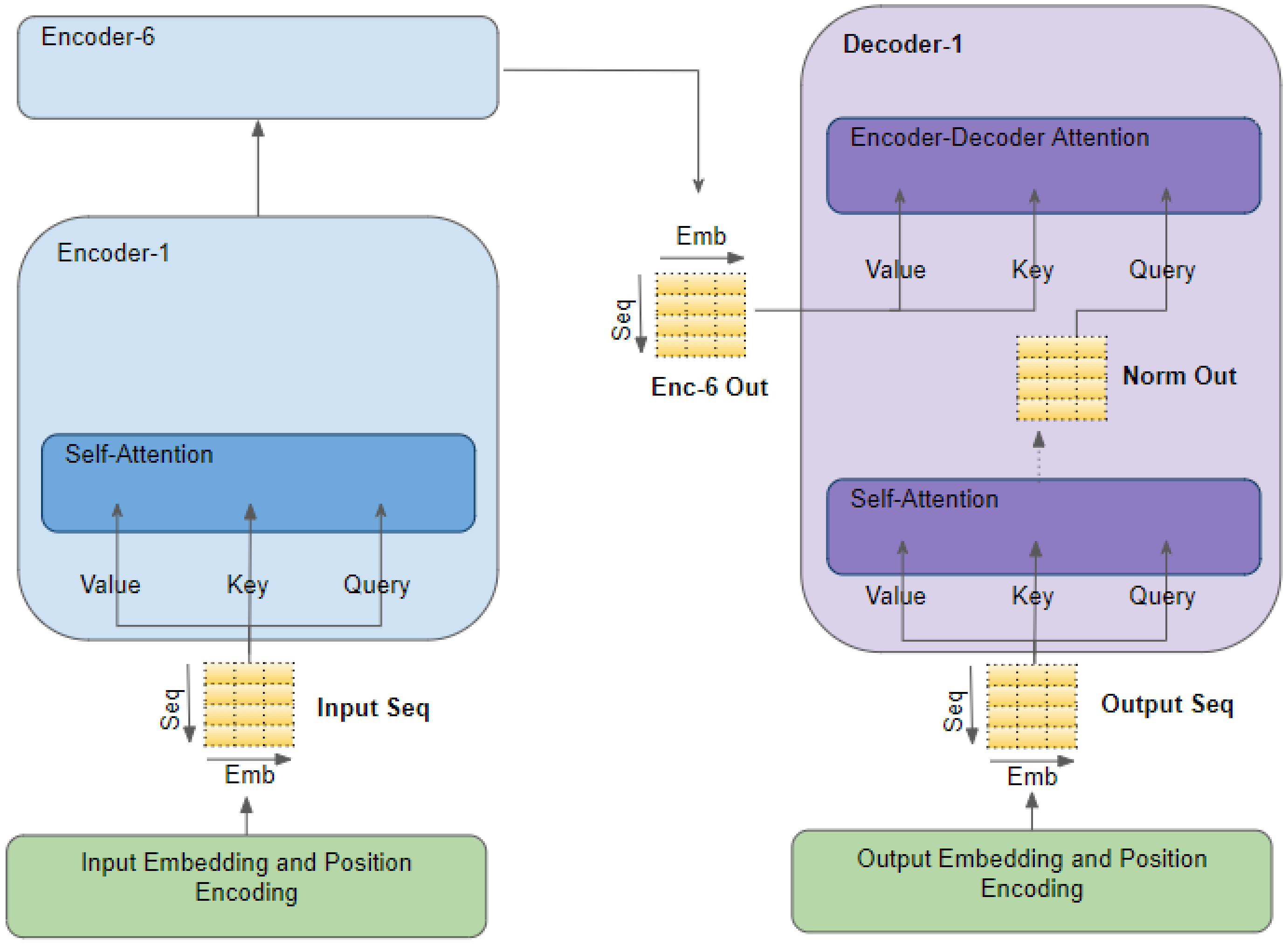
Deep Recurrent Attentive Writer (DRAW)

**Multimodel**
- videoBERT
- visualBERT

**Transformers for vision tasks**
Vision Transformer (ViT)
Detection Transformer (DETR)

**Image Generation**

Generative Adversarial Networks (SAGANs)



Encoder-6
Encoder-1
Self-Attention
Value
Key
Query
Seq
Emb
Input Seq
Input Embedding and Position Encoding
Emb
Seq
Enc-6 Out
Decoder-1
Encoder-Decoder Attention
Value
Key
Query
Norm Out
Self-Attention
Value
Key
Query
Seq
Emb
Output Seq
Output Embedding and Position Encoding

# Thank You

# Namah Shivaya