AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

श्रद्धावान् लभते ज्ञानम्

Introduction to Transformers

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

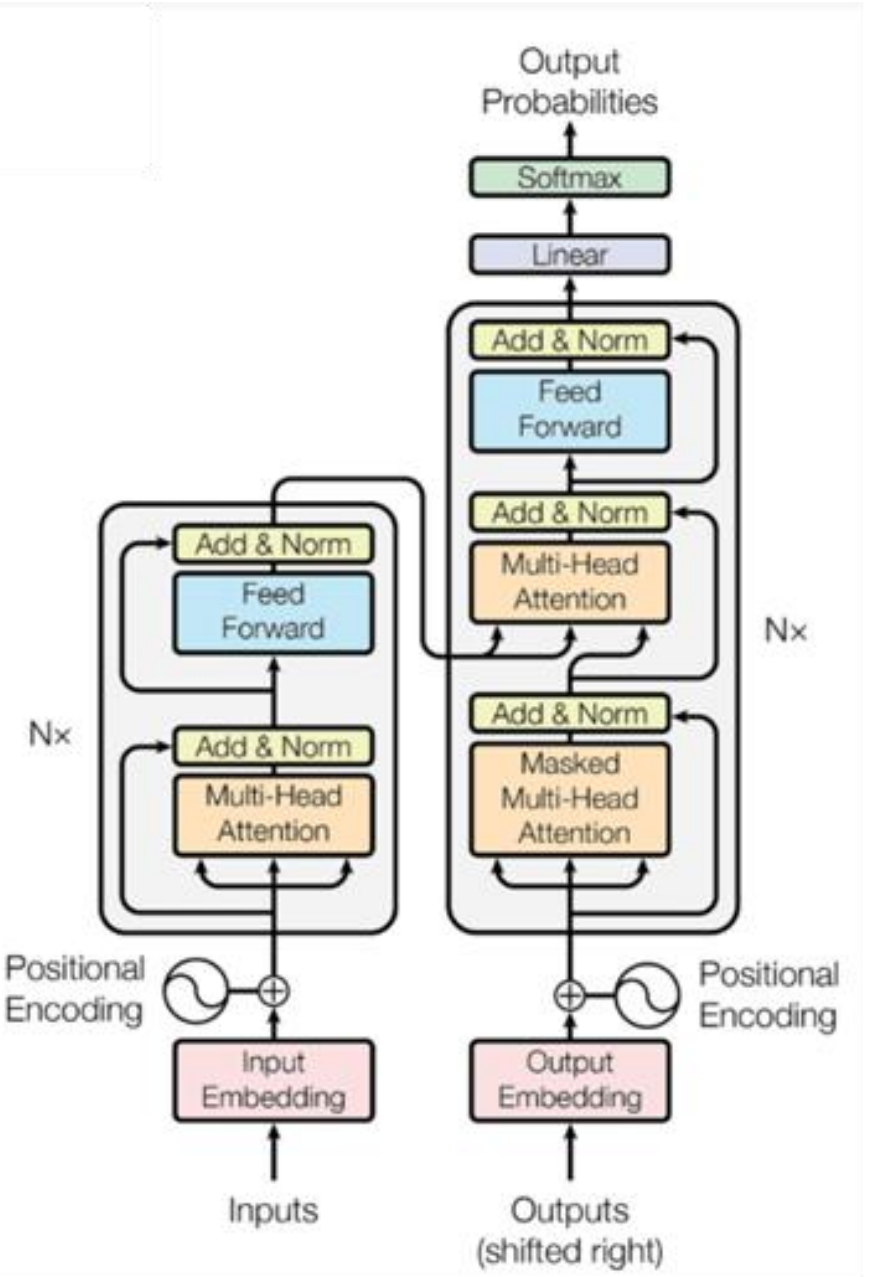
Challenges of RNN/LSTM

- **Sequential processing** Bottle Neck- training slow for large datasets or long sequence.
- **Issue with Long Range Dependencies-** Understanding limited to immediate context
- **Gradient** vanishing and Exploding
- Large # of **training steps**
- **Resource inefficiency-** Sequential nature, limits scalability and training on massive datasets
- Recurrence **prevents parallel** computation

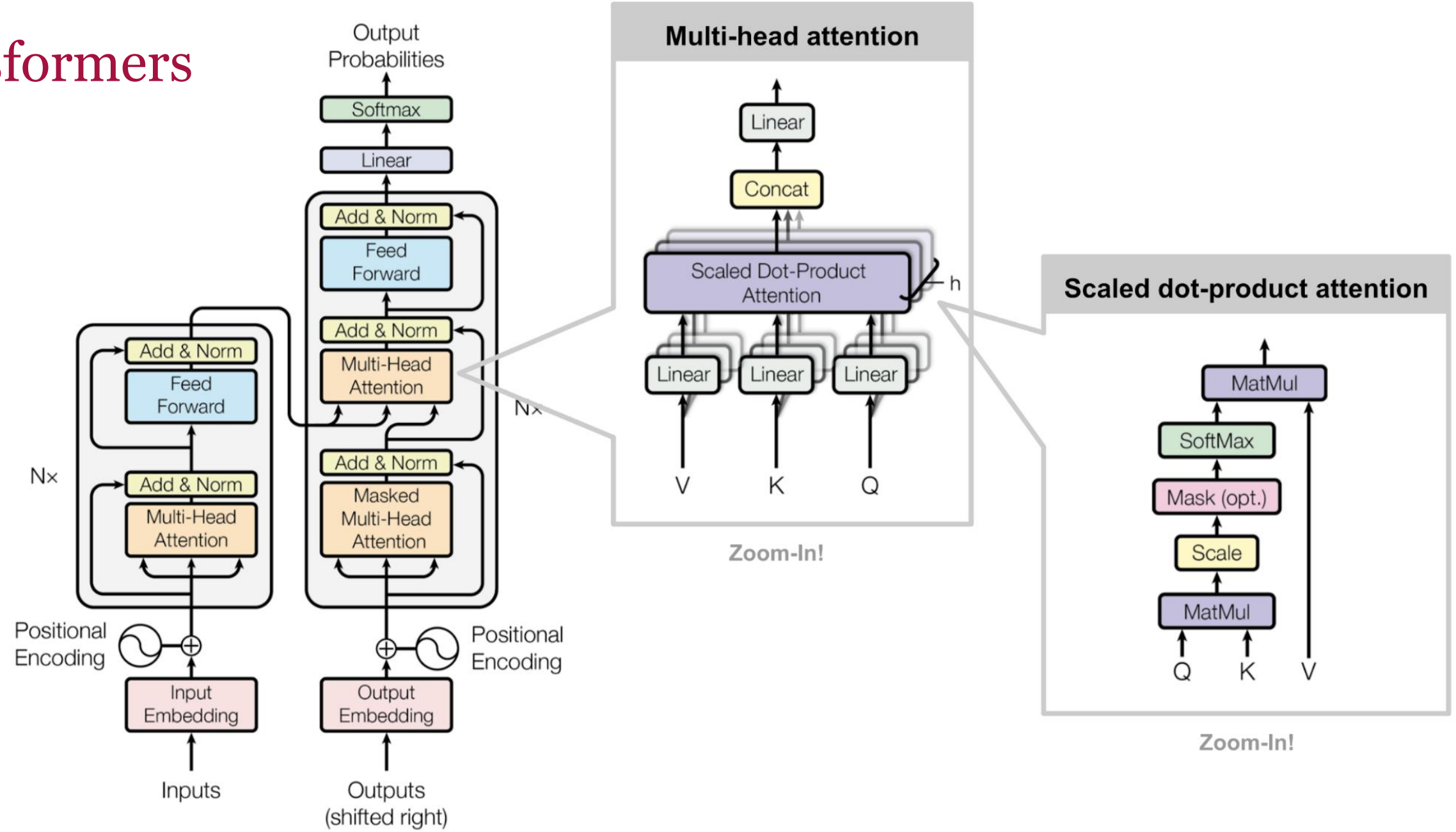
Transformer Networks

- **Parallelisation** through self attention
- **Faster training**
- **Facilitate long range dependencies-** models dependencies between any 2 tokens regardless of position- retain context across long docs
- No gradient vanishing and explosion
- Scalability and Efficiency
- Fewer training steps
- Flexible Context modeling
- No recurrence and that facilitate parallel computation

Transformers



Transformers

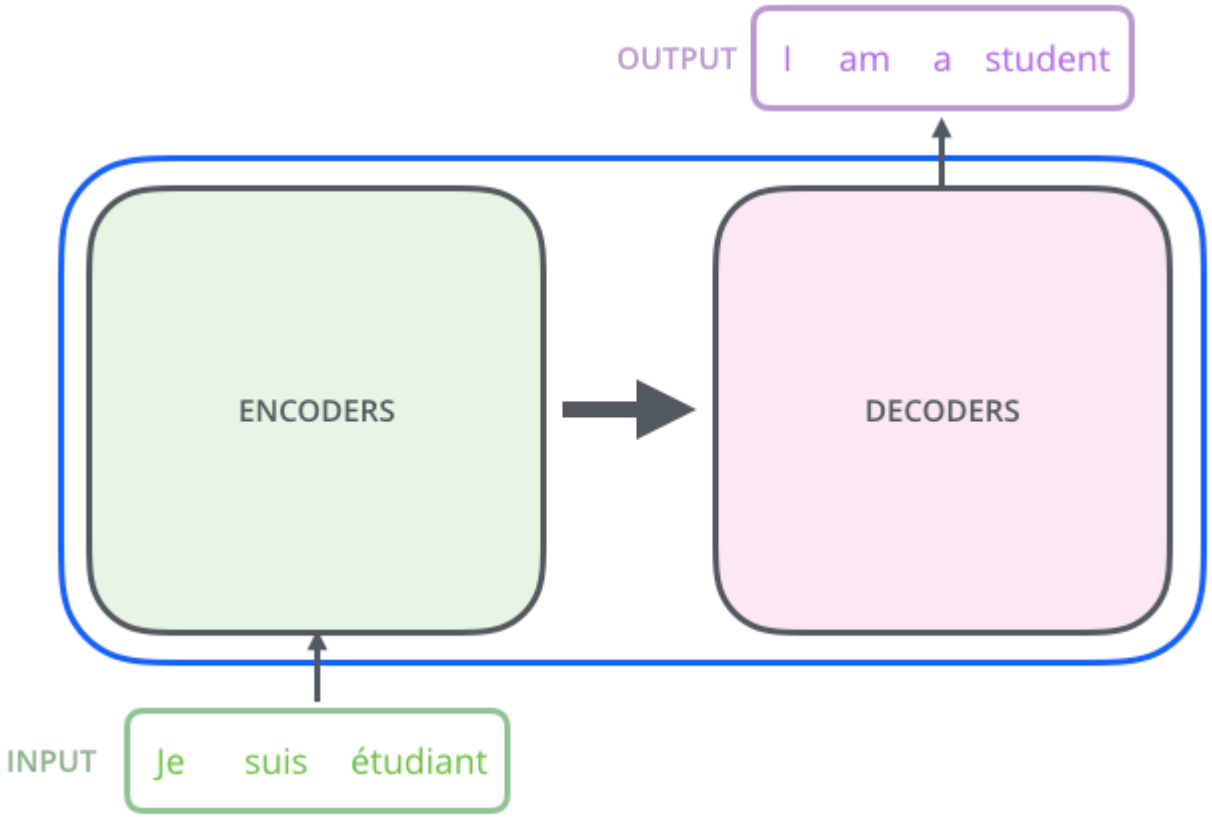


Transformers – Machine Translation example

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

Courtesy : jalammar.github.io

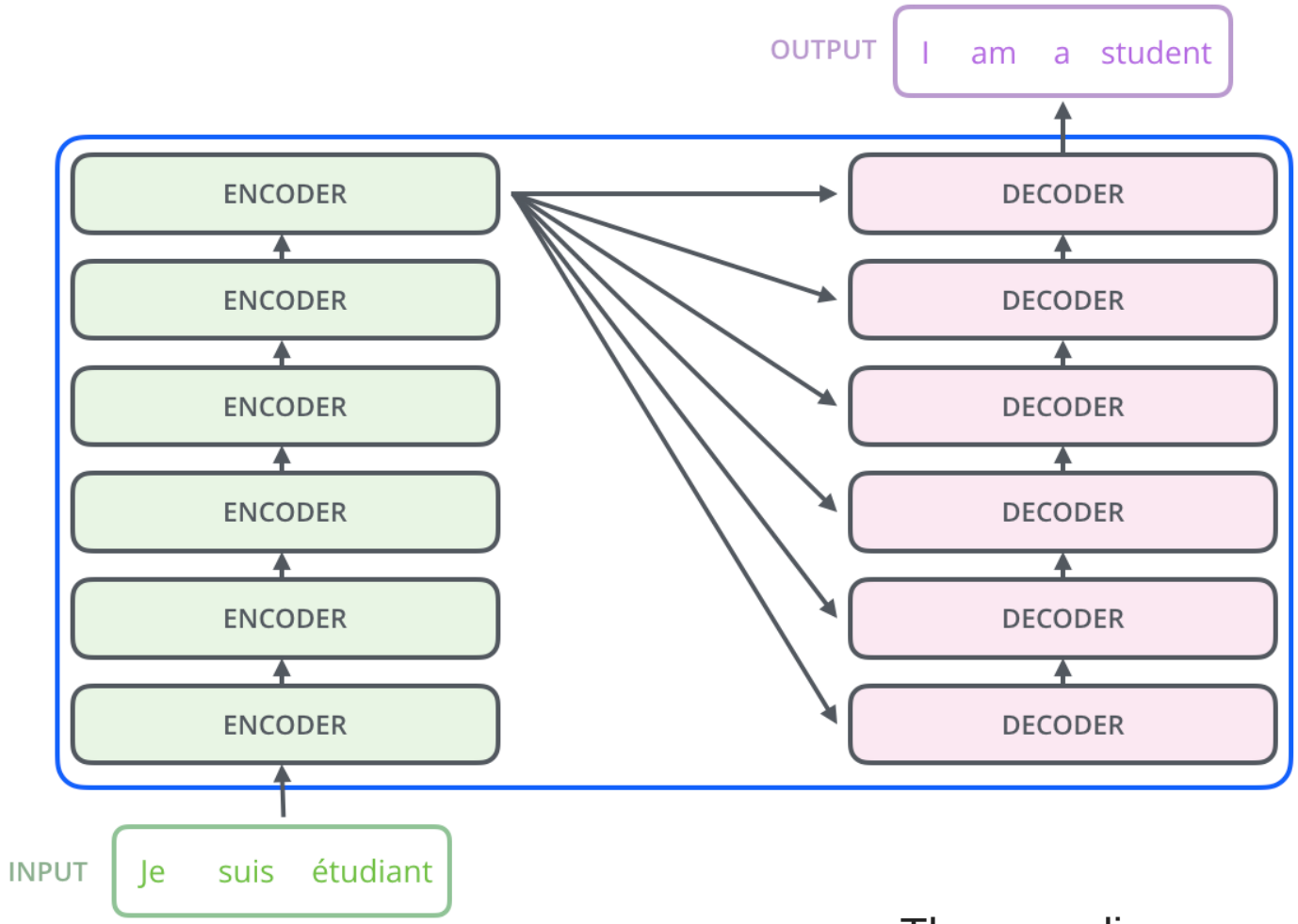
Transformers – Machine Translation example



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

http://jalammar.github.io

Transformers – Machine Translation example



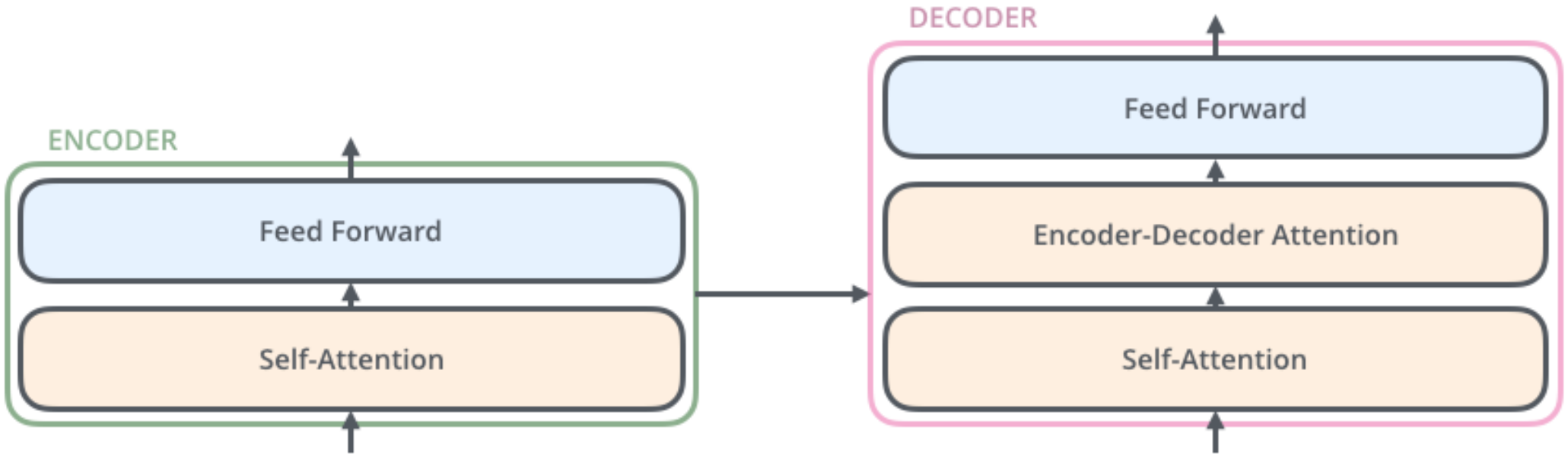
The encoders are all identical in structure (yet they do not share weights).

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

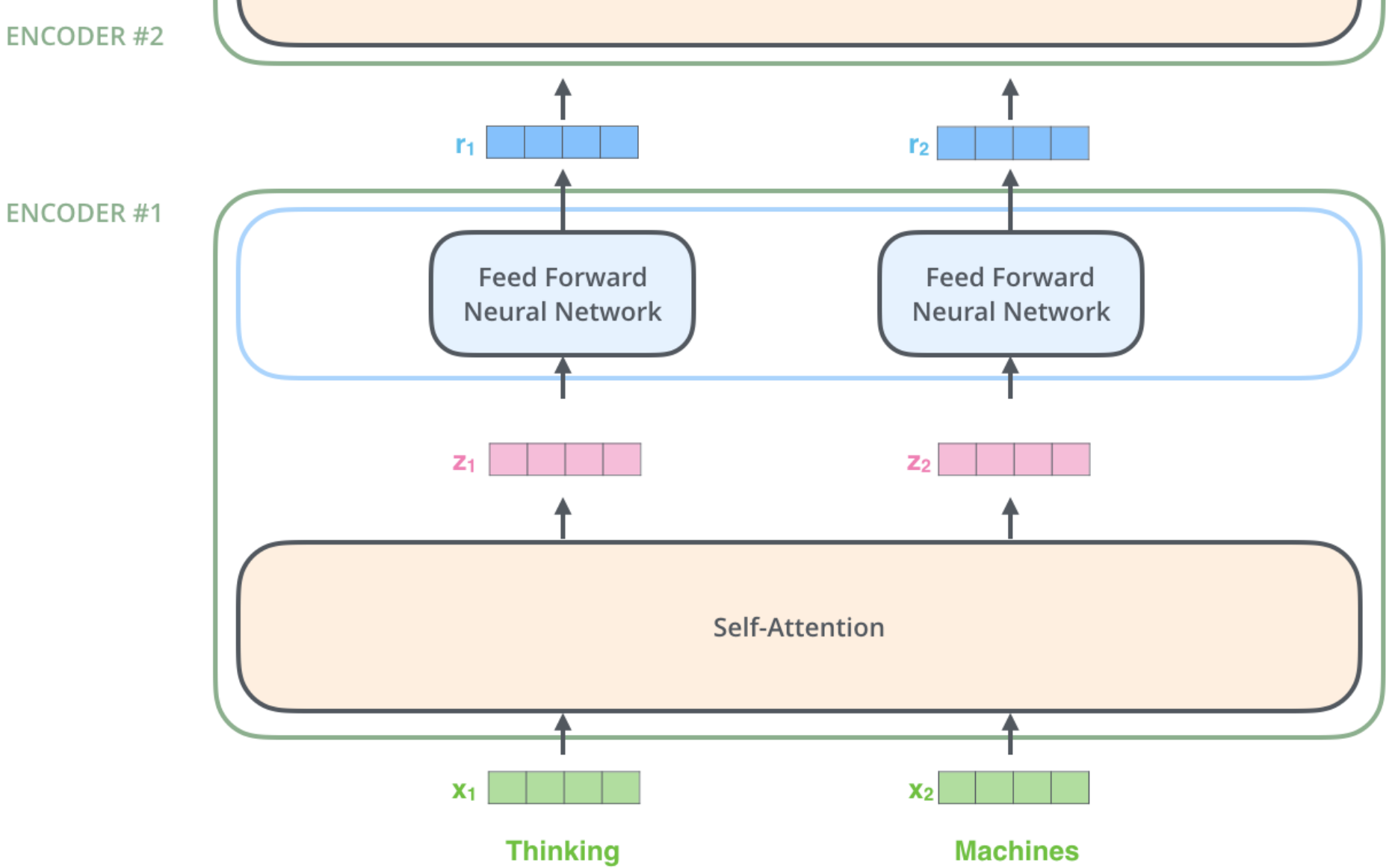
Transformers – Machine Translation example



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.

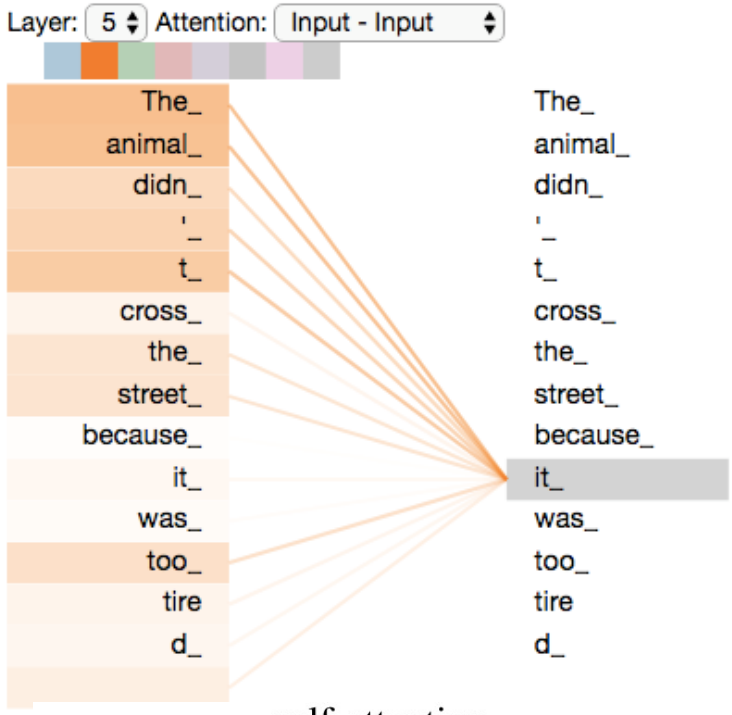
http://jalammar.github.io


ENCODER #2
ENCODER #1
r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Self-Attention

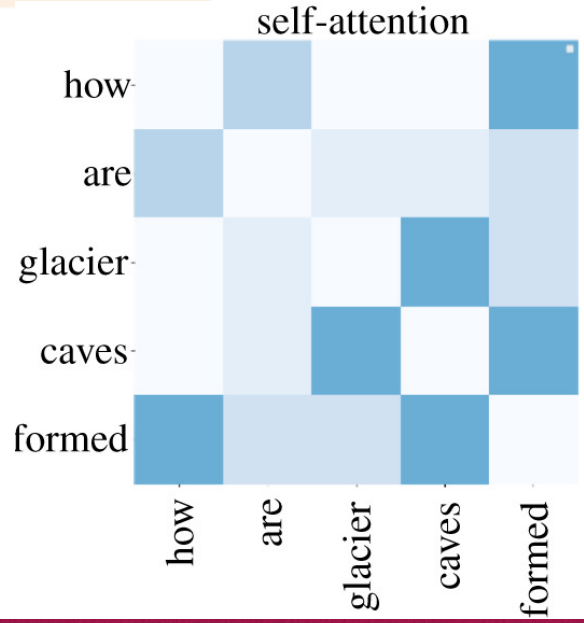
The animal didn't cross the street because it was too tired"

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".

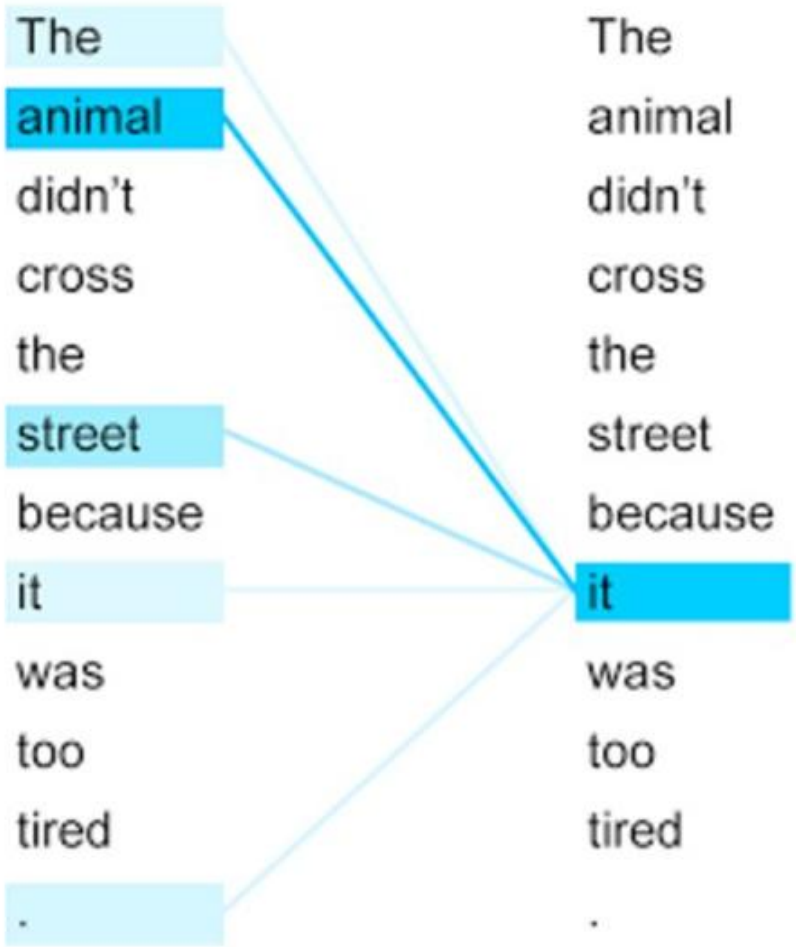
Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing.

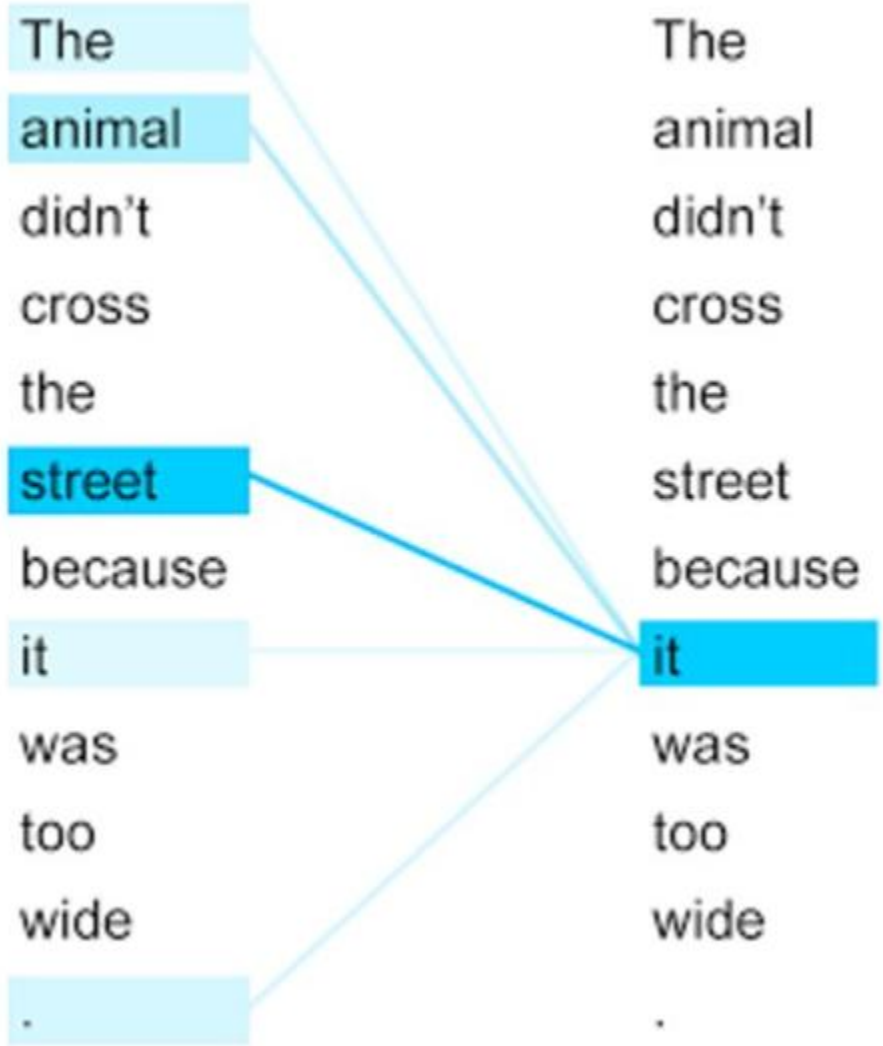


Self-attention is a mechanism in transformers that allows each token in a sequence to weigh and attend to all other tokens, capturing contextual relationships dynamically.

Self attention

Self-attention is a mechanism in transformers where each token in a sequence computes weighted dependencies with all other tokens using **query, key, and value projections**, enabling dynamic context-aware representations with parallelized computation.





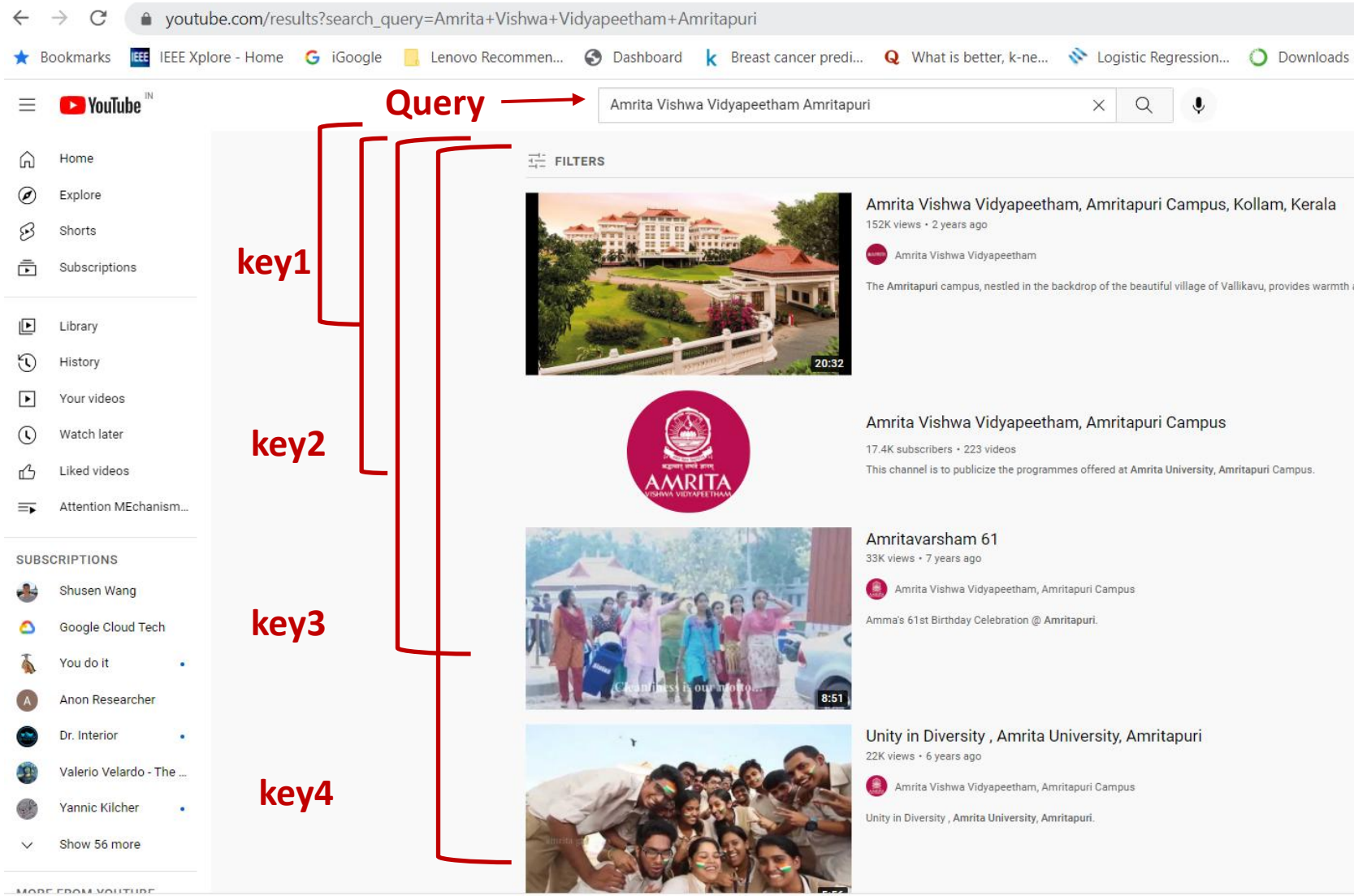
Self Attention

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

  what the element is “offering”,
- **Value vector (vi)**

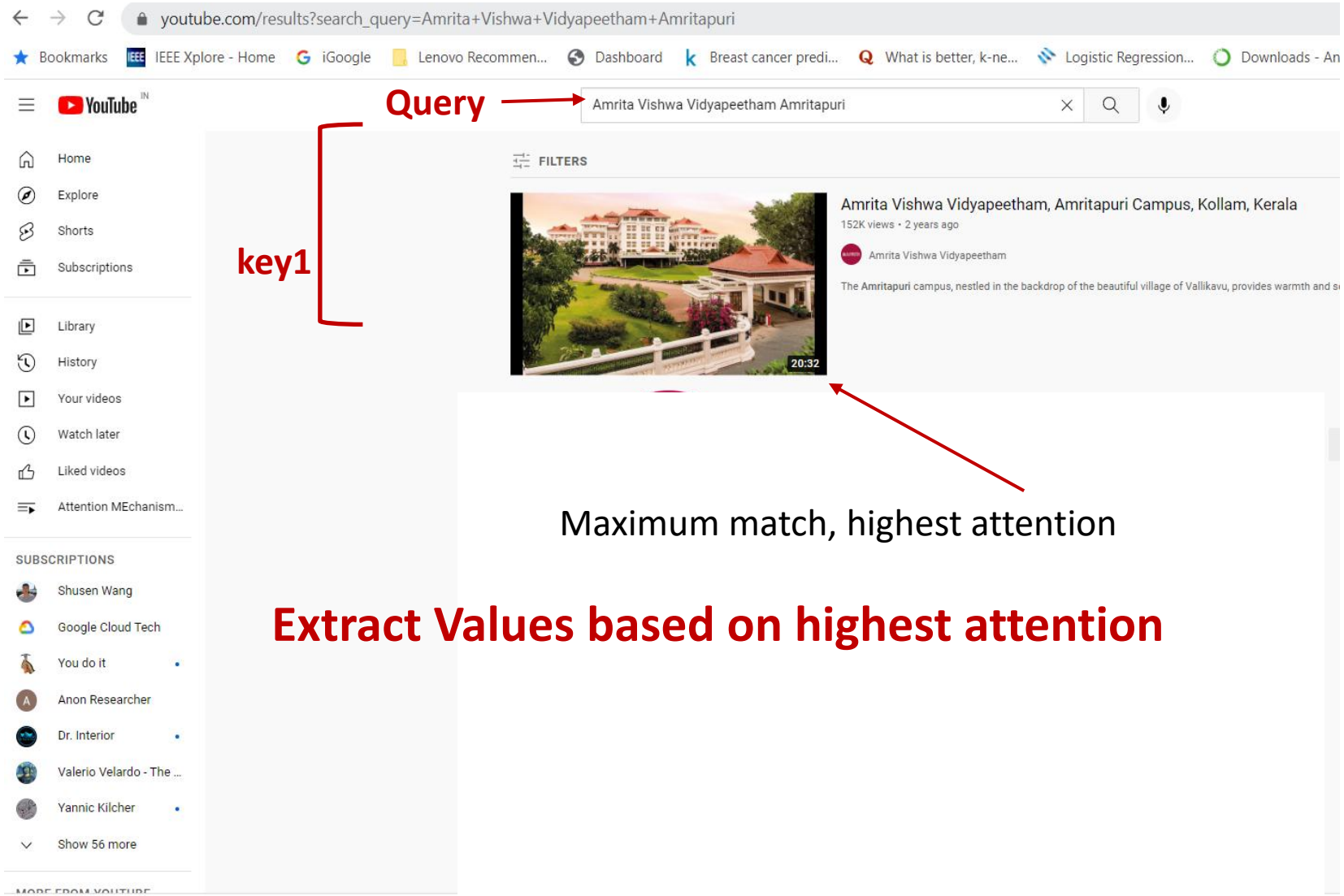
  This feature vector is the one we want to average over and finally get extracted
- **Score**

  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 –

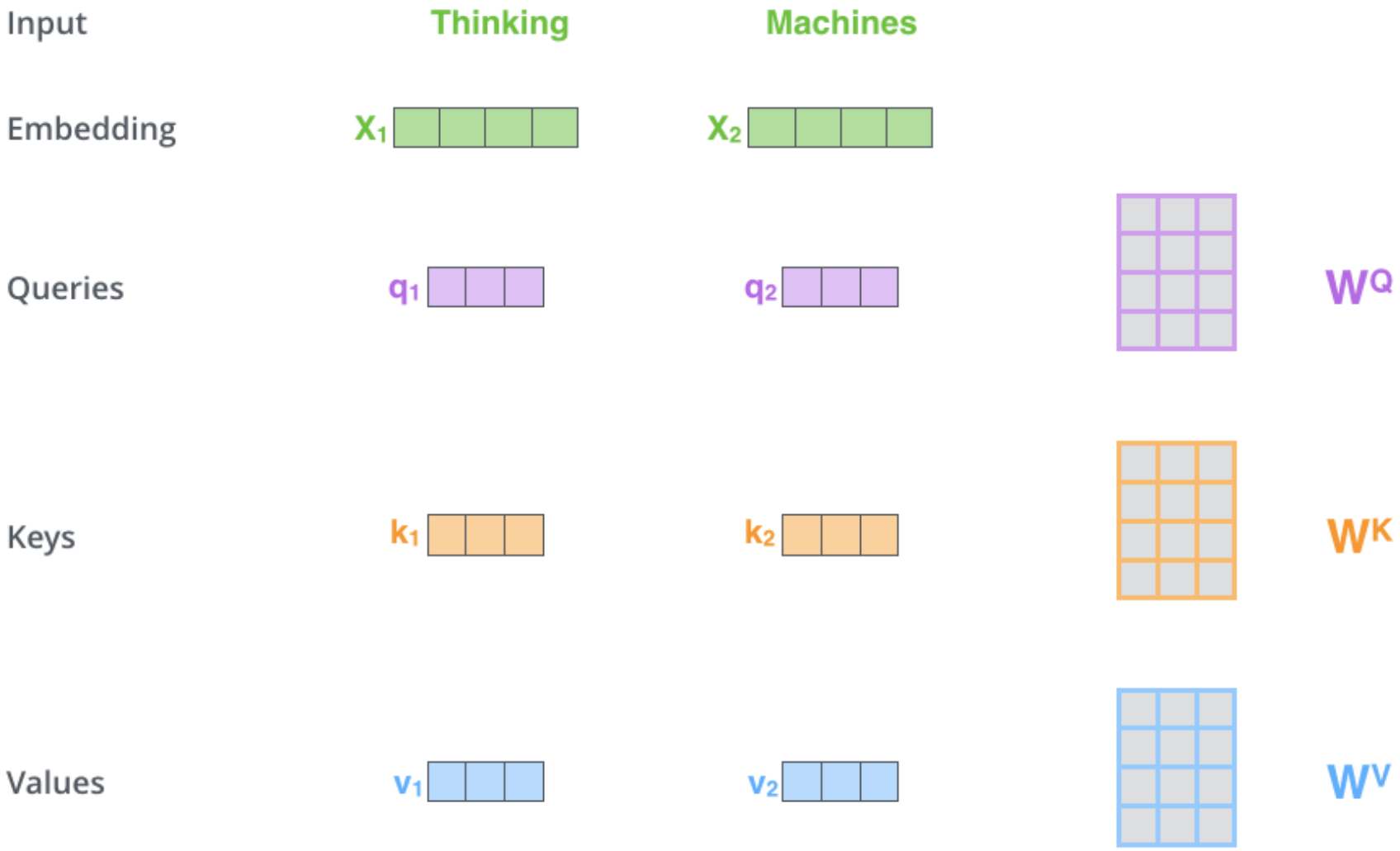
First step

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

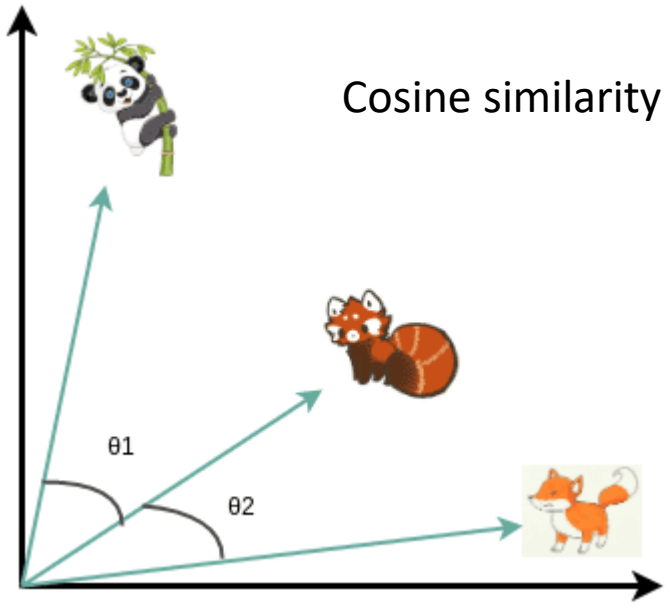


Steps in Self Attention

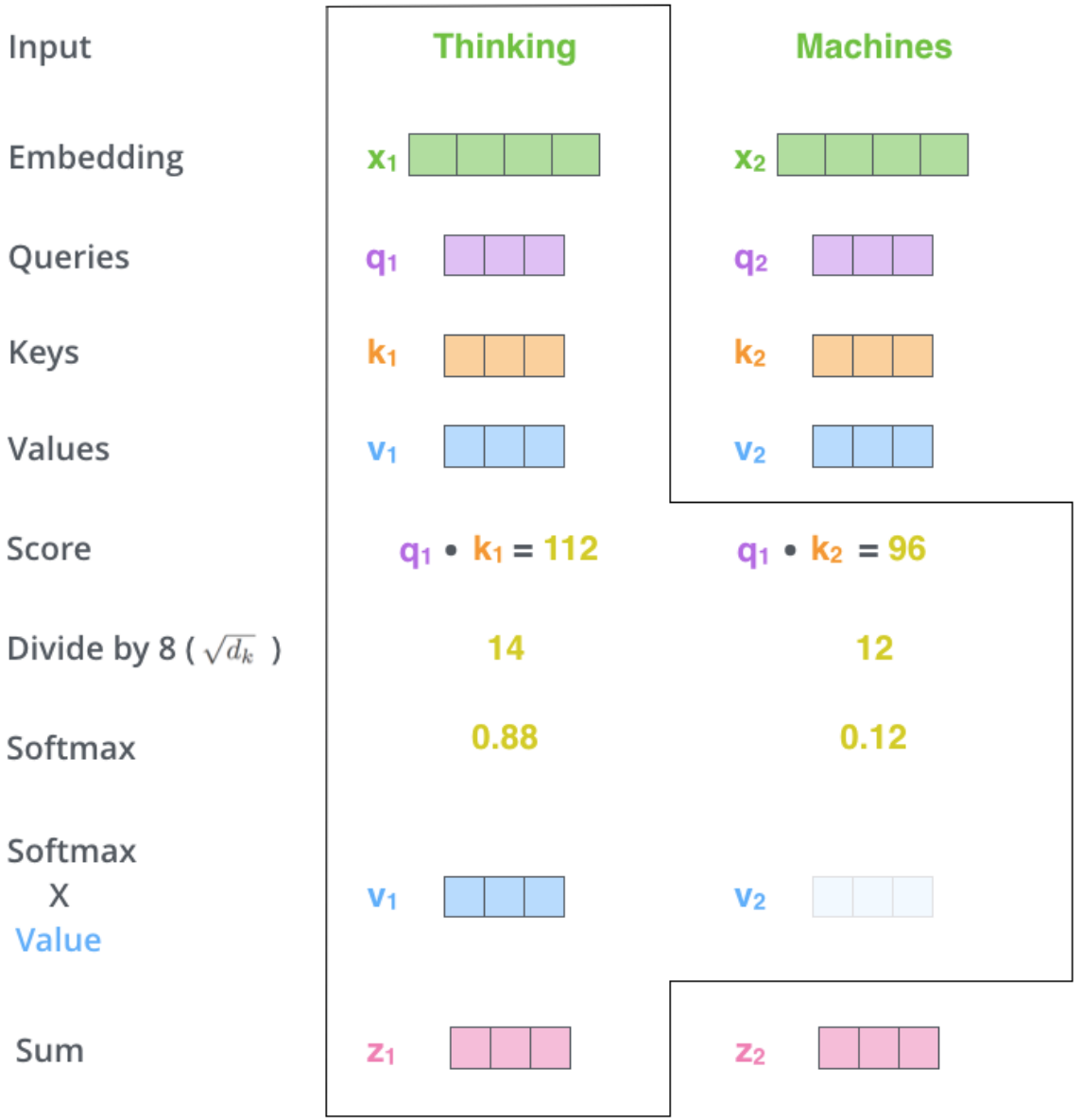
The **second step**

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

$q_1 \cdot k_1 = 112$ $q_1 \cdot k_2 = 96$



Cosine similarity



Steps in Self Attention

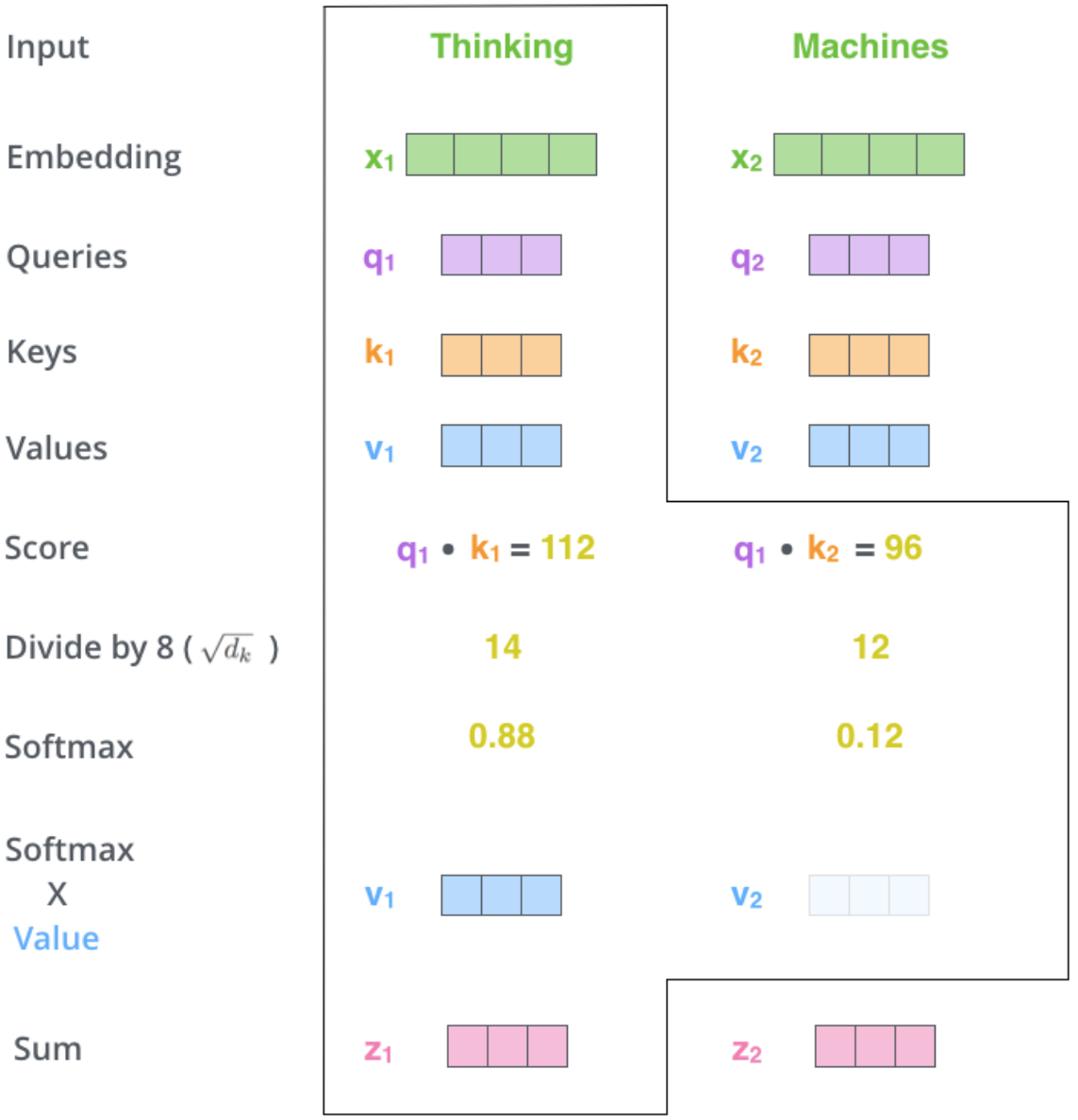
Third step

Word	q vector	k vector	v vector	score	score / 8
thinking	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$
Machines		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$

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.



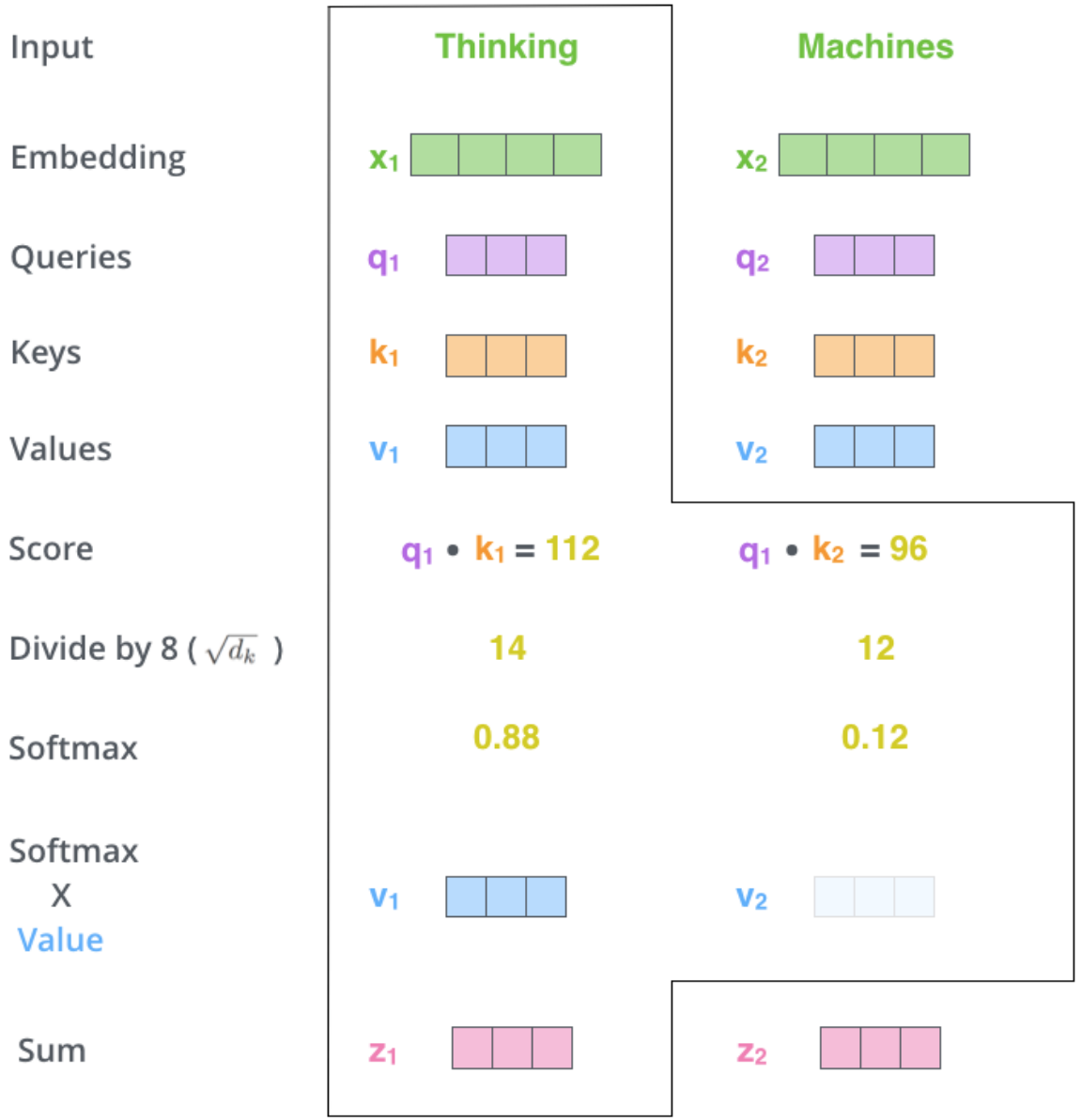
Steps in Self Attention –

Fourth step

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

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



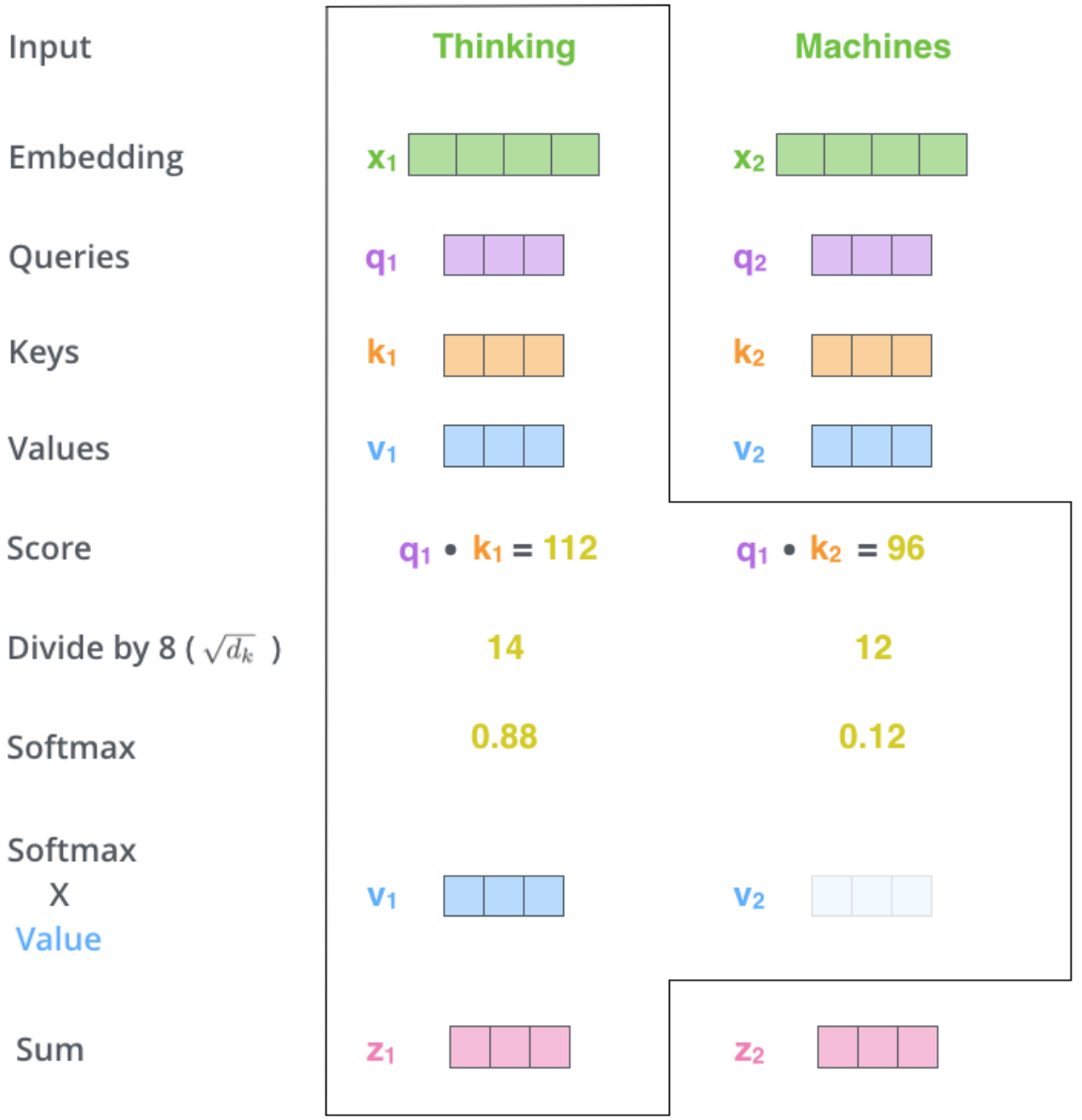
Steps in Self Attention –

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).

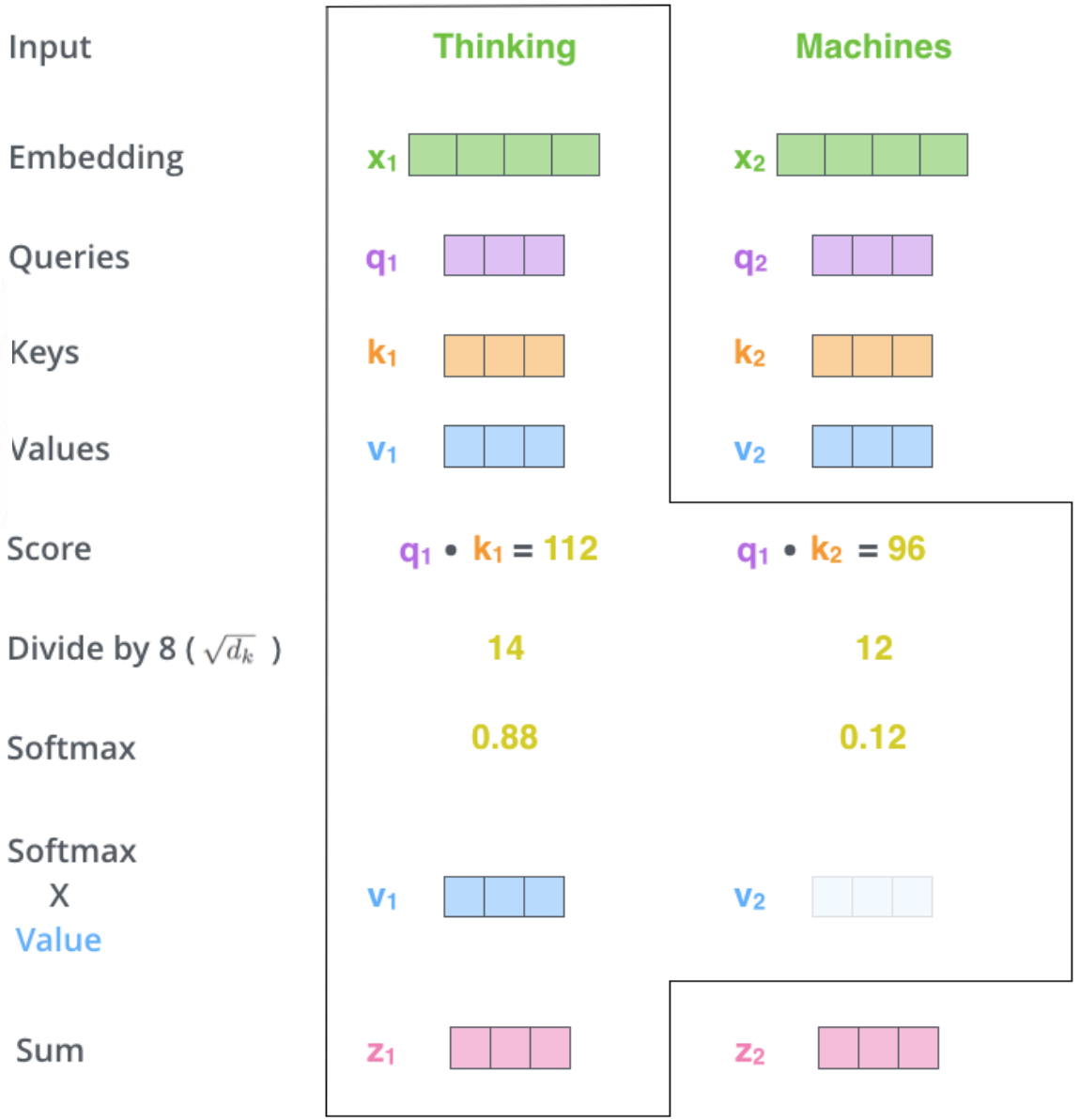


Steps in Self Attention –

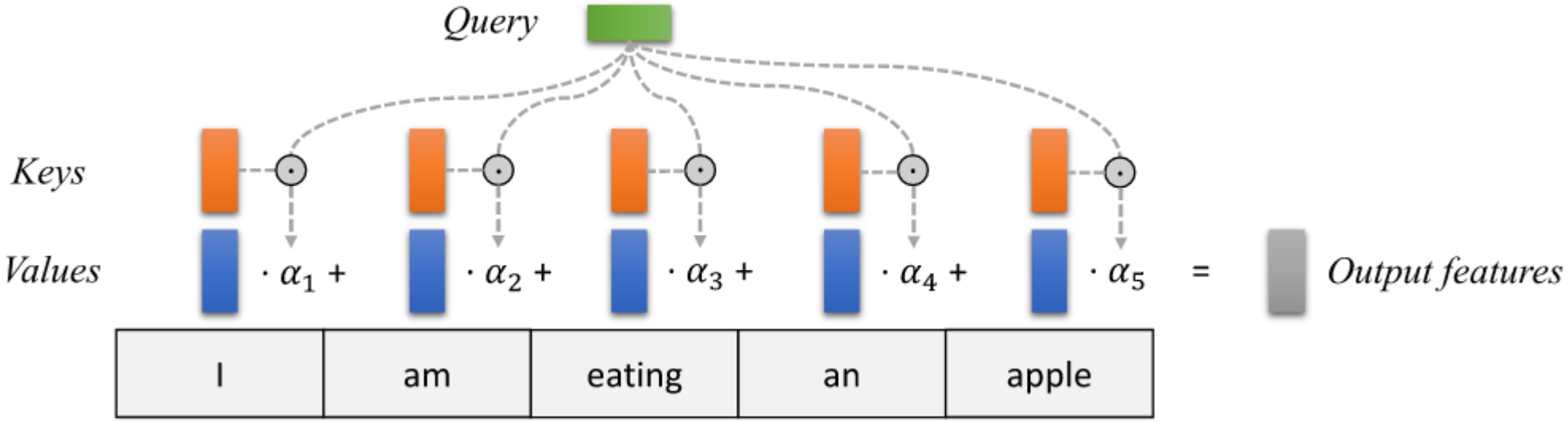
Sixth Step

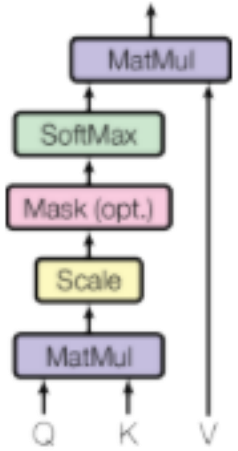
Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
thinking		k_1	v_1	$q_2 . k_1$	$q_2 . k_1 / 8$	x_{21}	$x_{21} * v_1$	
Machines	q_2	k_2	v_2	$q_2 . k_2$	$q_2 . k_2 / 8$	x_{22}	$x_{22} * v_2$	z_2

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 –

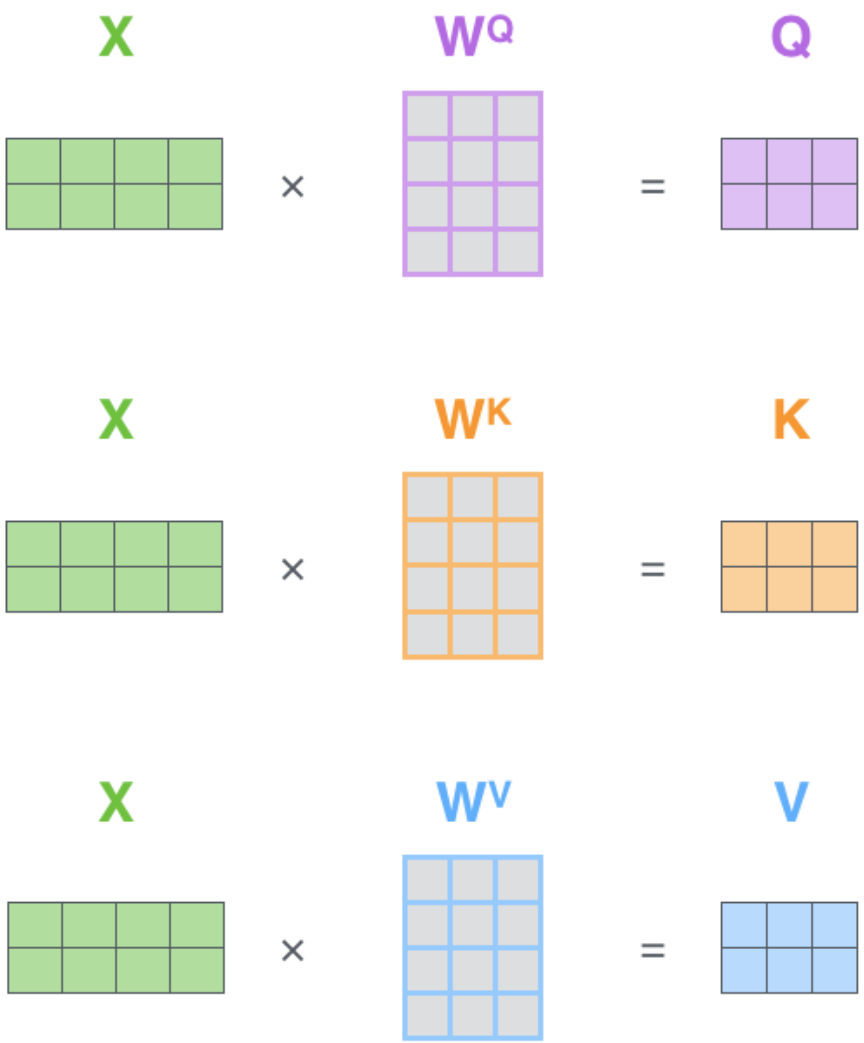




$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

Matrix Calculation of Self-Attention



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

“Multi-headed” attention

1) This is our input sentence*

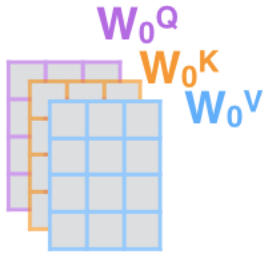
2) We embed each word*

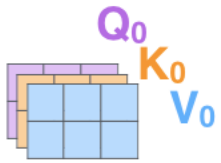
3) Split into 8 heads. We multiply X or R with weight matrices

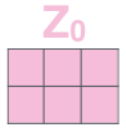
4) Calculate attention using the resulting Q/K/V matrices

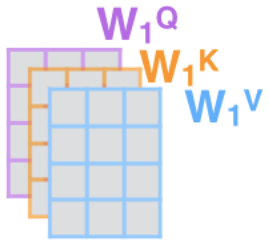
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

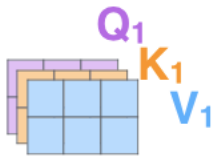
Thinking Machines

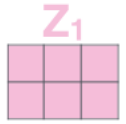




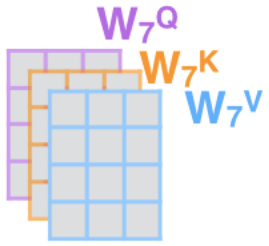


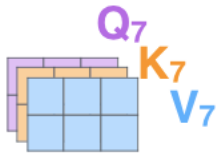


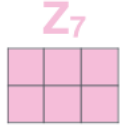


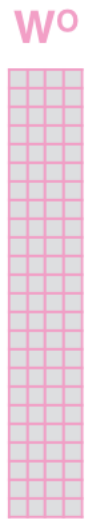


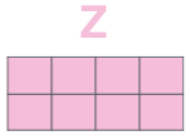
* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one

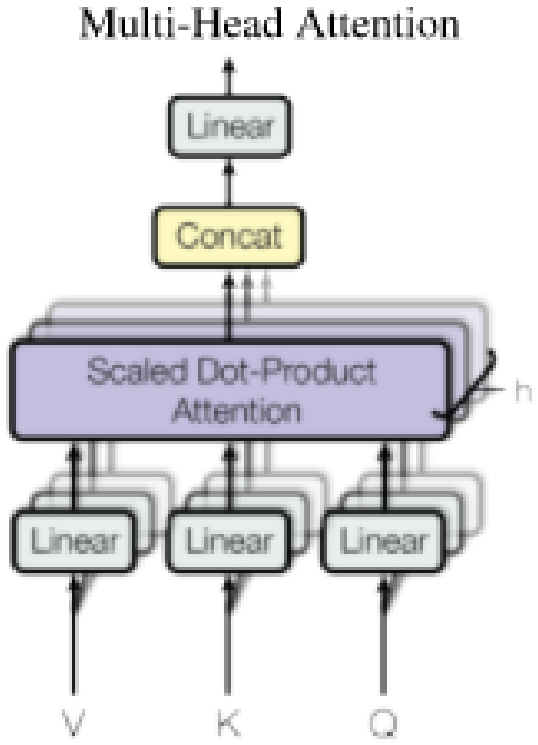












multiple “representation subspaces”

multiple sets of Query/Key/Value weight matrices

Self Attention- Numerical Example

- The embedding of the word *I* is $x_1 = [1.76, 2.22, \ldots, 6.66]$.
- The embedding of the word *am* is $x_2 = [7.77, 0.631, \ldots, 5.35]$.
- The embedding of the word *good* is $x_3 = [11.44, 10.10, \ldots, 3.33]$.

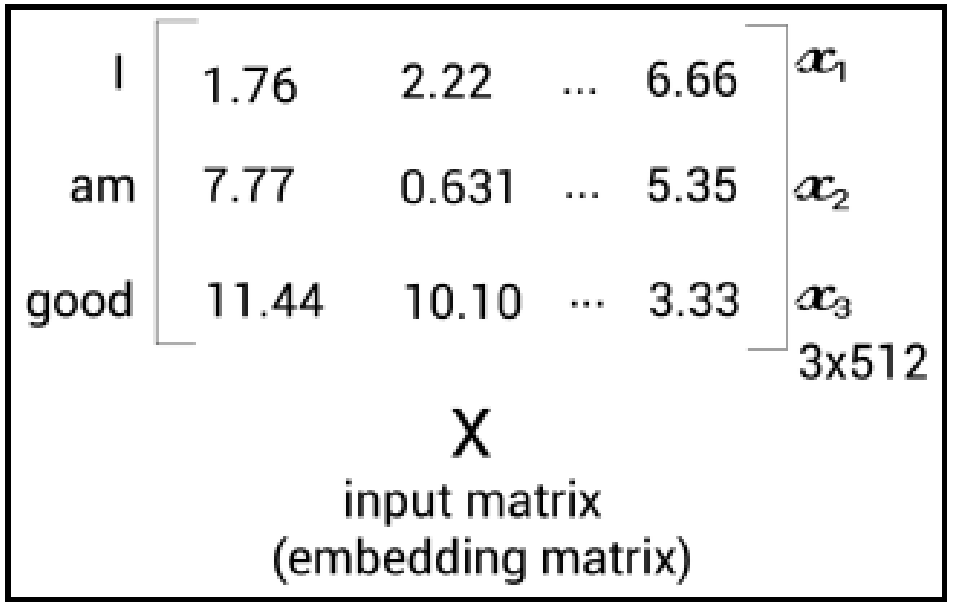


Figure 1.6 – Input matrix

Get Q K V

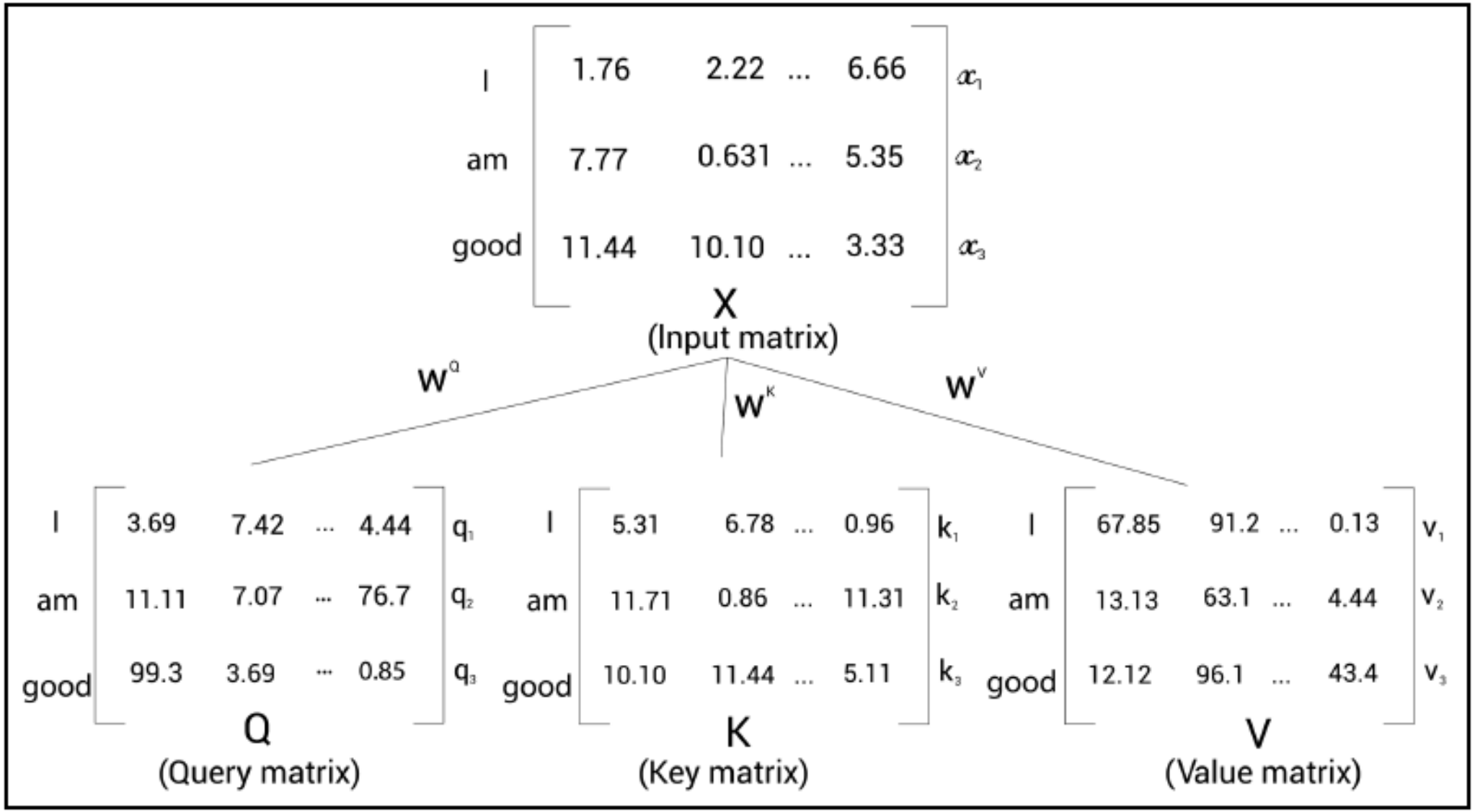


Figure 1.7 – Creating query, key, and value matrices

Step 1

The first step in the self-attention mechanism is to compute the dot product between the query matrix, Q, and the key matrix, K^T:

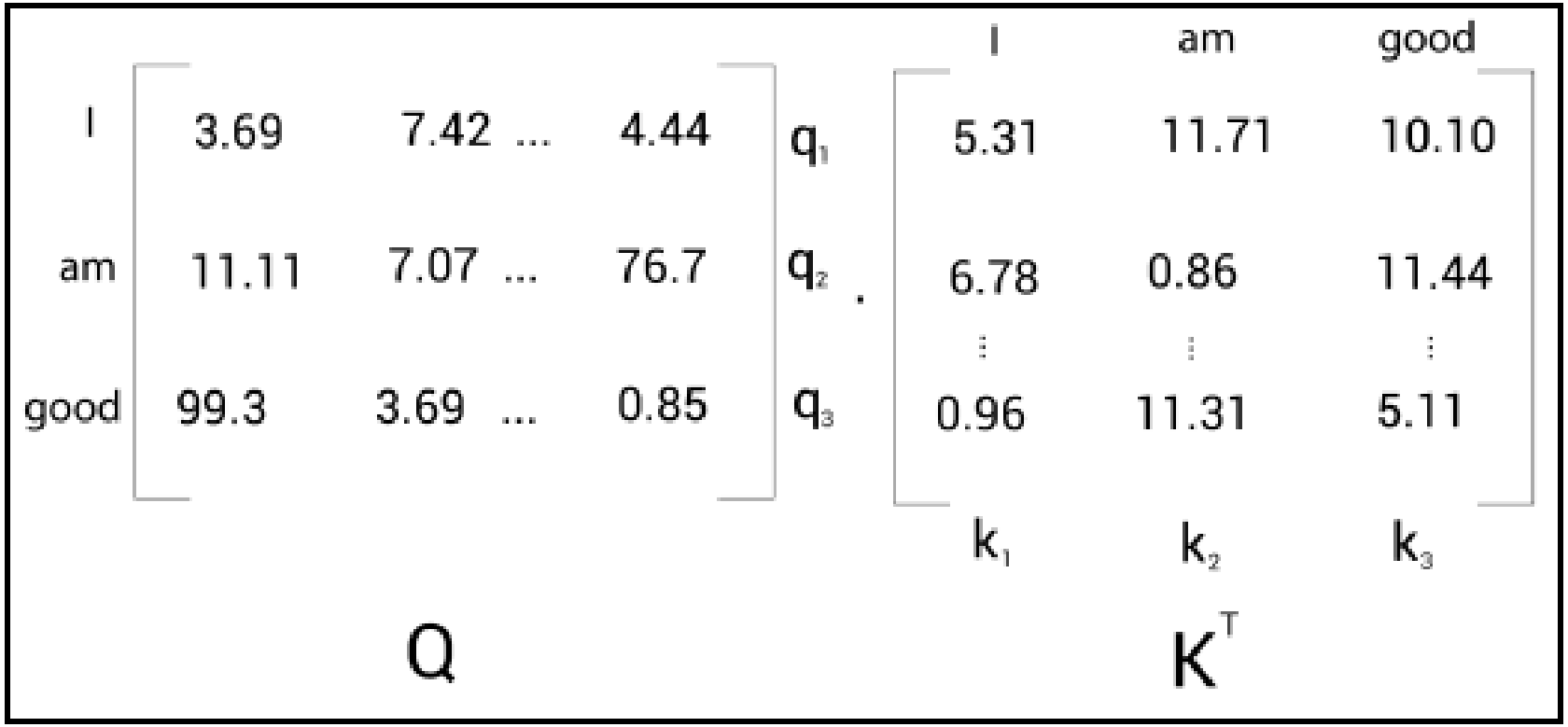


Figure 1.9 – Query and key matrices

Step 2

The next step in the self-attention mechanism is to divide the $Q.K^T$ matrix by the square root of the dimension of the key vector. But why do we have to do that? This is useful in obtaining stable gradients.

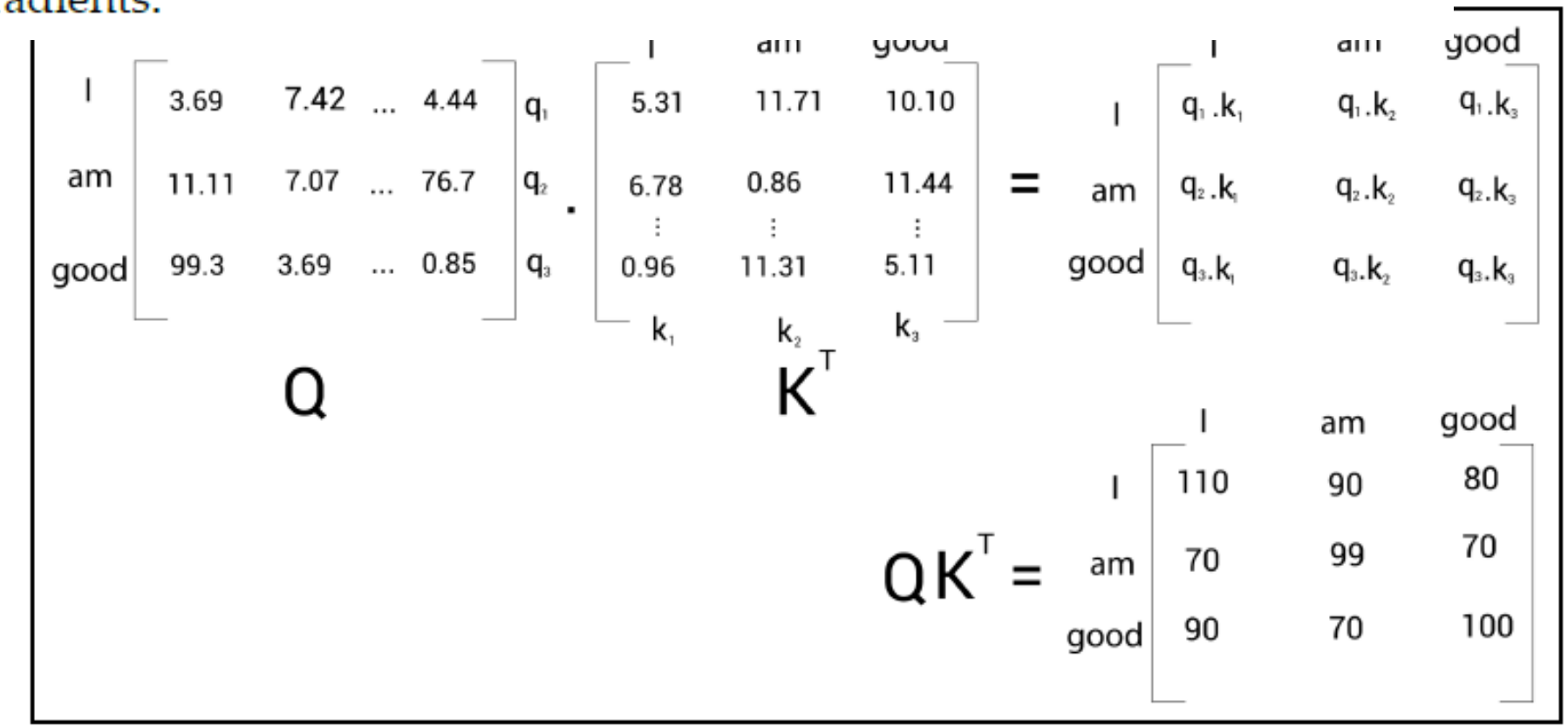


Figure 1.10 – Computing the dot product between the query and key matrices

Normalising the scores

$$\frac{QK^T}{\sqrt{d_K}} = \frac{QK^T}{8} =$$

	I	am	good
I	13.75	11.25	10
am	8.75	12.375	8.75
good	11.25	8.75	12.5

Figure 1.14 – Dividing $Q.K^T$ by the square root of d_k

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) =$$

	I	am	good
I	0.90	0.07	0.03
am	0.025	0.95	0.025
good	0.21	0.03	0.76

Figure 1.15 – Applying the softmax function

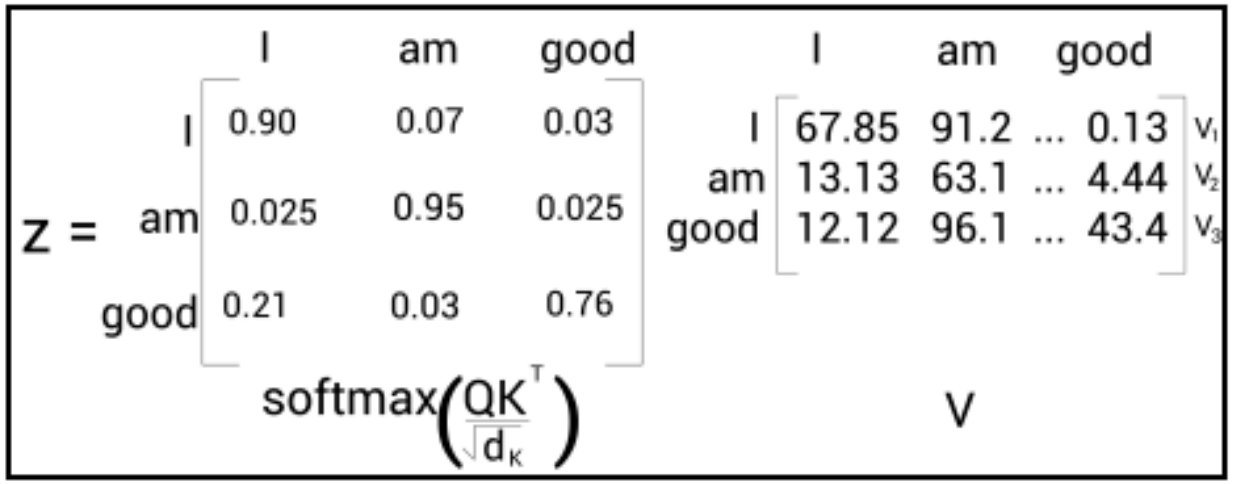


Figure 1.16 – Computing the attention matrix

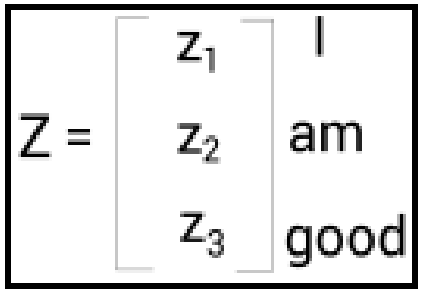


Figure 1.17 – Result of the attention matrix

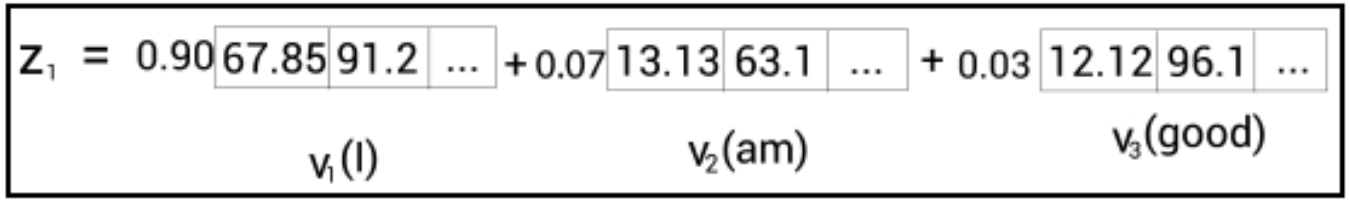


Figure 1.18 – Self-attention of the word I

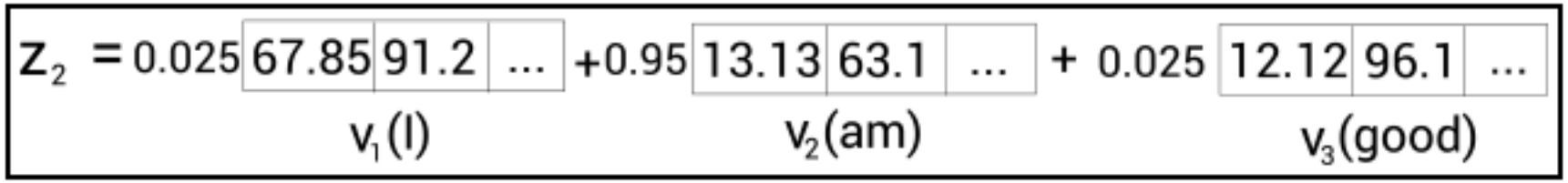


Figure 1.20 – Self-attention of the word am

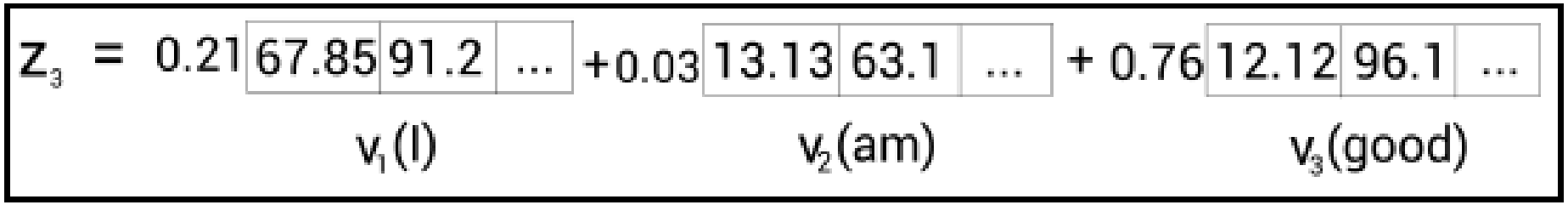


Figure 1.21 – Self-attention of the word good

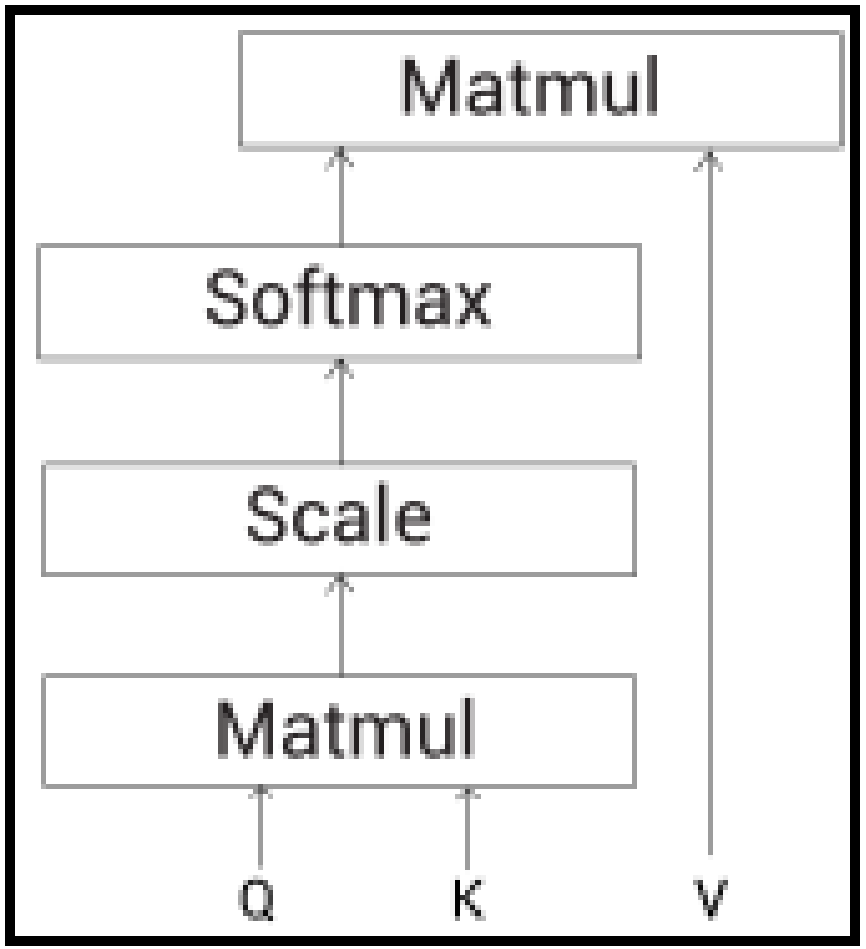
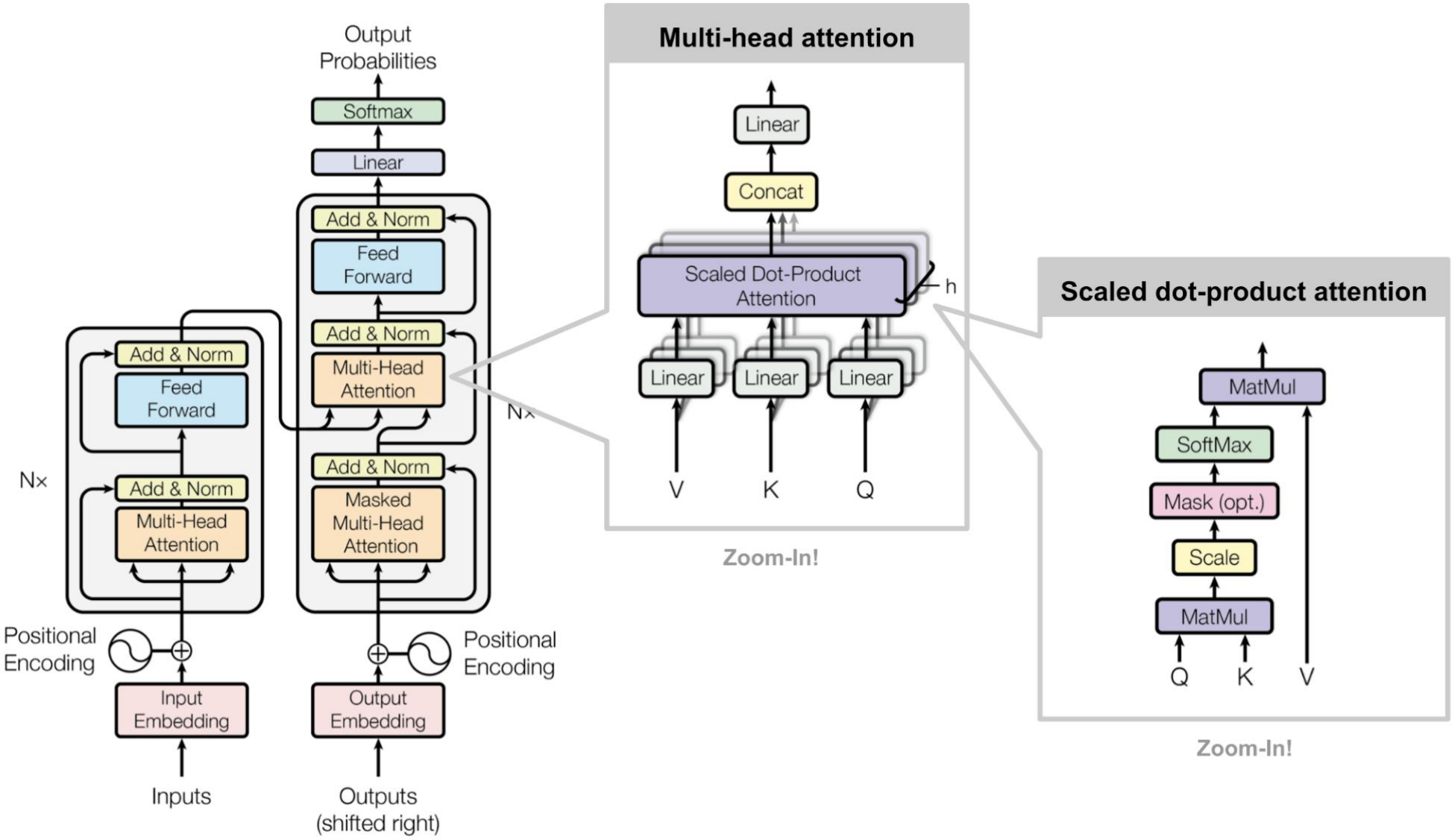
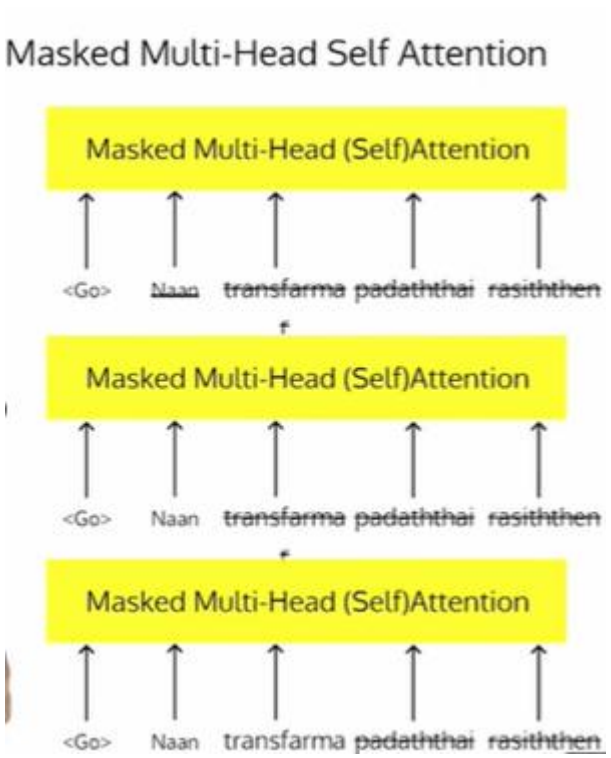


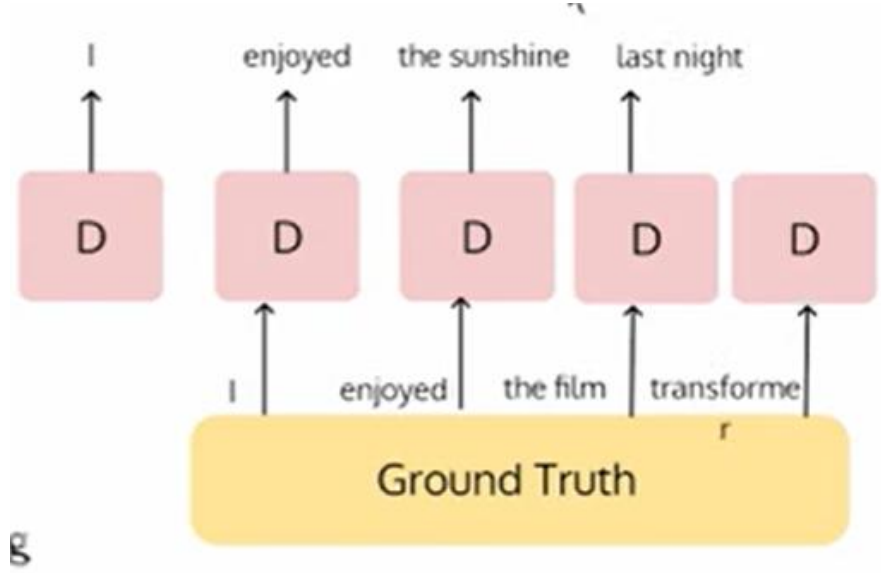
Figure 1.22 – Self-attention mechanism

Transformers



Masked Multi Head Attention





Teacher Forcing

$$e^0 = 1$$

$$e^{-\infty} = 0$$

Masking is done by inserting negative infinite at the respective positions.

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

T (left) T (below)

This actually forms an triangular matrix with one half all zeros and the other half all negative infinity

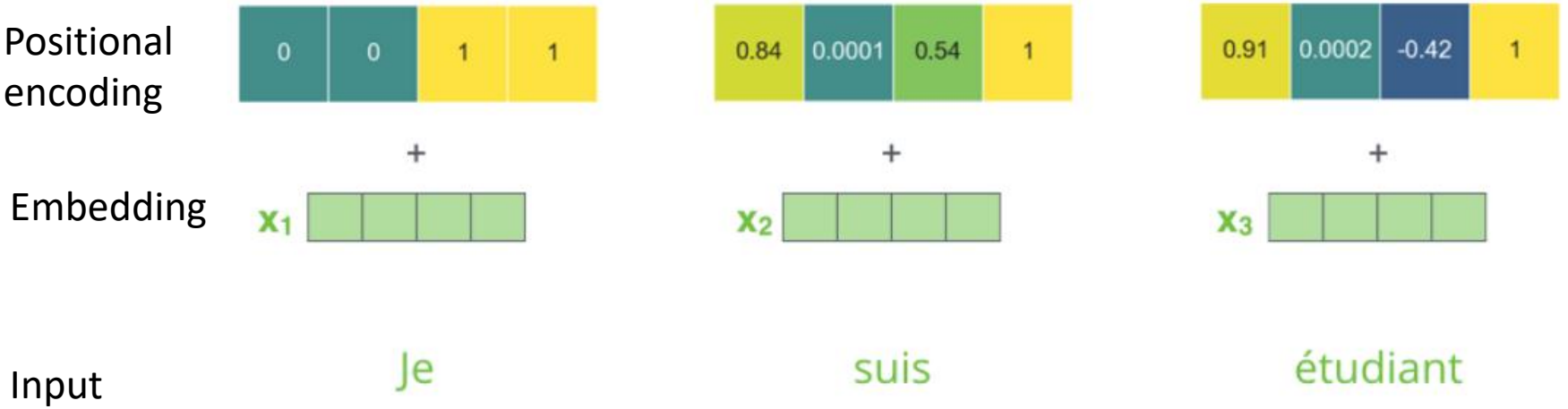
$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Masking in Multi-Head Attention: Why Is It Needed?

✓ **Prevents Attending to Unimportant Tokens** (e.g., padding).
✓ **Ensures Autoregressive Decoding** (prevents future peeking).
✓ **Maintains Training Consistency** (aligns with left-to-right generation).
✓ **Optimizes Computation** (avoids unnecessary attention to empty inputs).

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



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

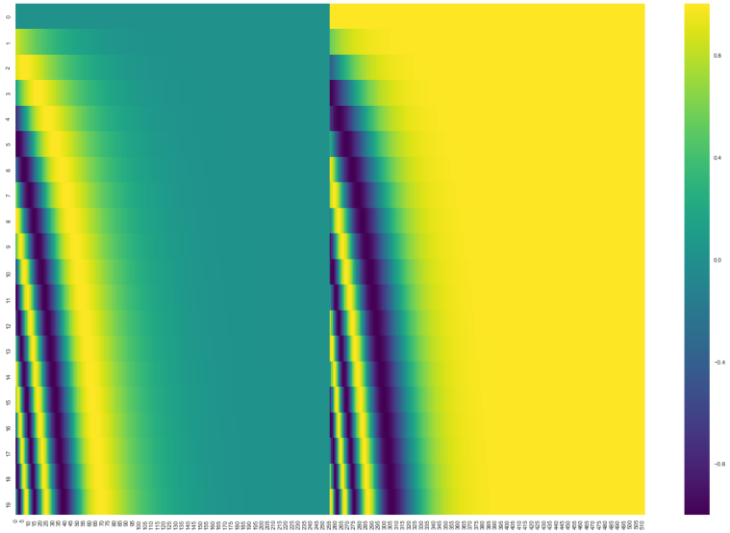
$PE_{(pos,i)}$

Vaswani et al 2017

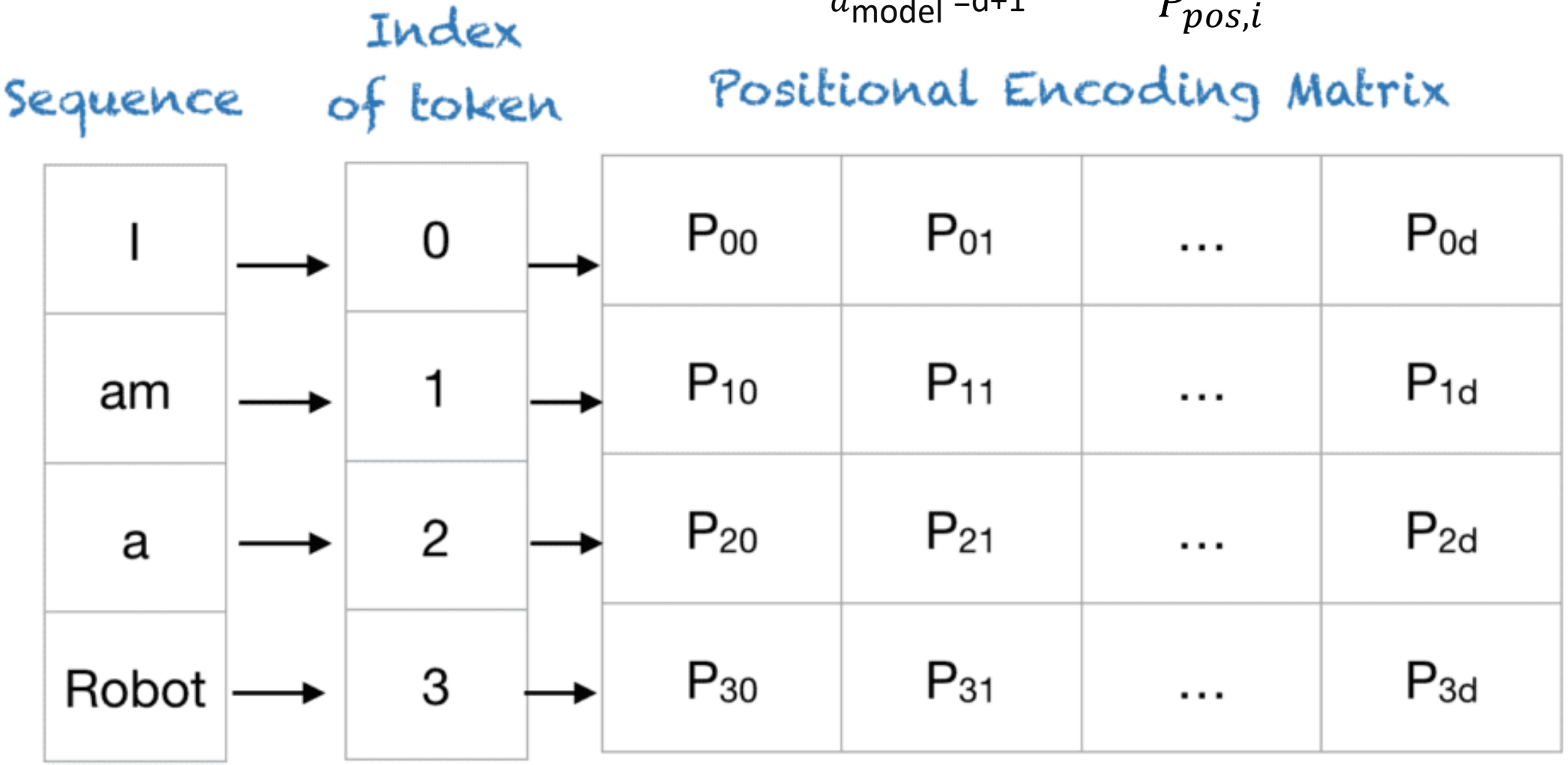
- *pos* is the position of the token in the sequence.
- *i* is the dimension index within the positional encoding.
- *d*model is the dimensionality of the word embedding .

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

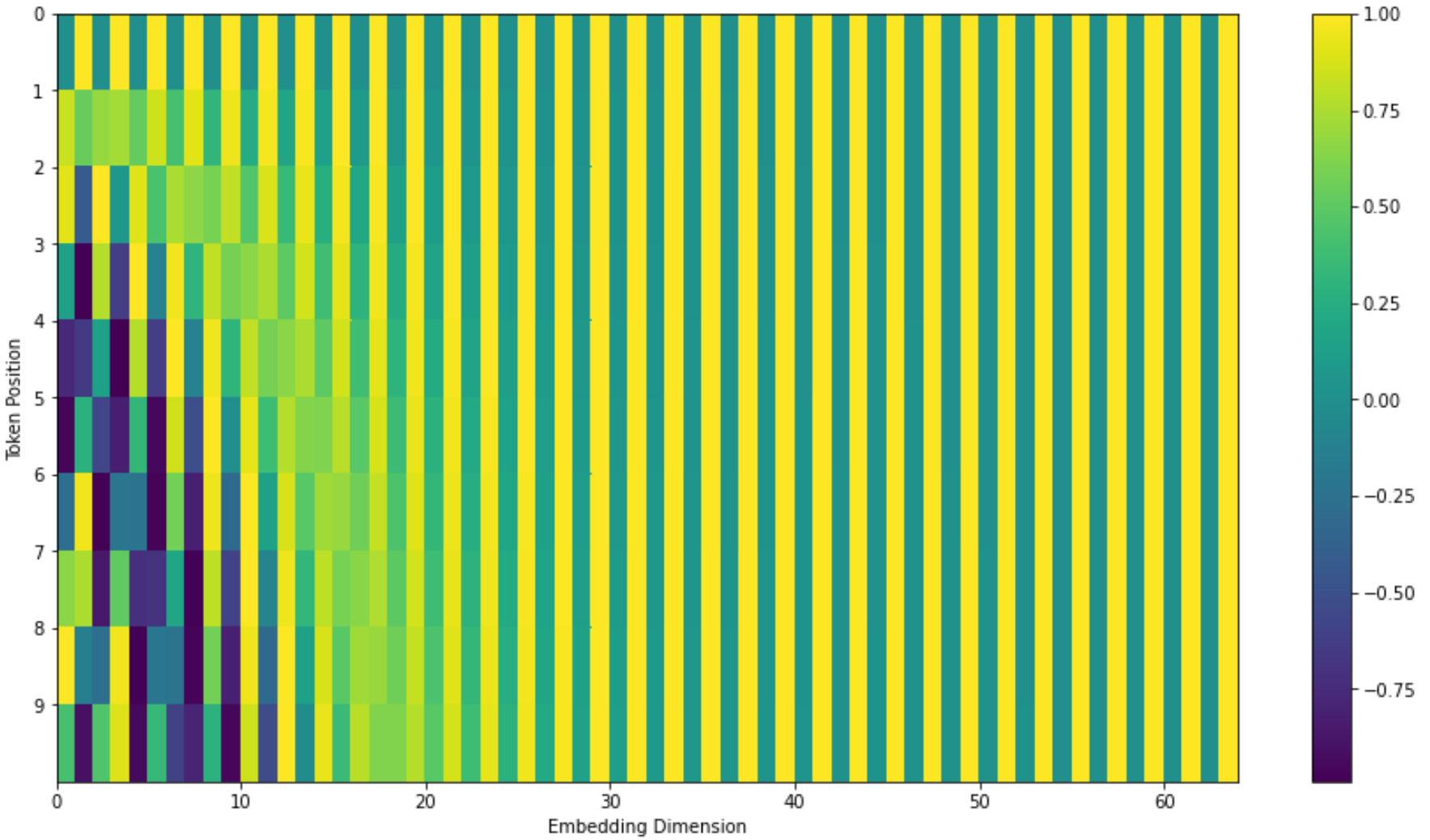
A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns)



d_{model} =d+1 $P_{pos,i}$



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


Token Position
Embedding Dimension
0
1
2
3
4
5
6
7
8
9
0
10
20
30
40
50
60
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

d_{model}=4

X =	I	1.769	2.22	3.4	5.8	x_1
	am	7.3	9.9	8.5	7.1	x_2
	good	9.1	7.1	0.85	10.1	x_3

Input matrix
(embedding matrix)

$$P(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{1000^{2i/d_{\text{model}}}}\right)$$

$$P(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{1000^{2i/d_{\text{model}}}}\right)$$

d_{model} is embedding dimension
pos is the position of the word in the sentence
i is the position of the embedding

X =	1.769	2.22	3.4	5.8	+	0	1	0	1
	7.3	9.9	8.5	7.1		0.841	0.54	0.01	0.99
	9.1	7.1	0.85	10.1		0.909	-0.416	0.02	0.99
	X					P			
=	1.769	3.22	3.4	6.8					
	8.14	10.44	8.51	8.09					
	10.0	6.68	0.87	11.09					

Add input and positional encoding matrix

$$P = \begin{matrix} \text{I} \\ \text{am} \\ \text{good} \end{matrix} \begin{bmatrix} \sin\left(\frac{pos}{10000^{0}}\right) & \cos\left(\frac{pos}{10000^{0}}\right) & \sin\left(\frac{pos}{10000^{2/4}}\right) & \cos\left(\frac{pos}{10000^{2/4}}\right) \\ \sin\left(\frac{pos}{10000^{0}}\right) & \cos\left(\frac{pos}{10000^{0}}\right) & \sin\left(\frac{pos}{10000^{2/4}}\right) & \cos\left(\frac{pos}{10000^{2/4}}\right) \\ \sin\left(\frac{pos}{10000^{0}}\right) & \cos\left(\frac{pos}{10000^{0}}\right) & \sin\left(\frac{pos}{10000^{2/4}}\right) & \cos\left(\frac{pos}{10000^{2/4}}\right) \end{bmatrix}$$

Figure 1.27 – Computing the positional encoding matrix

$$P = \begin{matrix} \text{I} \\ \text{am} \\ \text{good} \end{matrix} \begin{bmatrix} \sin(pos) & \cos(pos) & \sin\left(\frac{pos}{100}\right) & \cos\left(\frac{pos}{100}\right) \\ \sin(pos) & \cos(pos) & \sin\left(\frac{pos}{100}\right) & \cos\left(\frac{pos}{100}\right) \\ \sin(pos) & \cos(pos) & \sin\left(\frac{pos}{100}\right) & \cos\left(\frac{pos}{100}\right) \end{bmatrix}$$

Figure 1.28 – Computing the positional encoding matrix

$$P = \begin{matrix} \text{I} \\ \text{am} \\ \text{good} \end{matrix} \begin{bmatrix} \sin(0) & \cos(0) & \sin(0/100) & \cos(0/100) \\ \sin(1) & \cos(1) & \sin(1/100) & \cos(1/100) \\ \sin(2) & \cos(2) & \sin(2/100) & \cos(2/100) \end{bmatrix}$$

Figure 1.29 – Computing the positional encoding matrix

$$P = \begin{matrix} \text{I} \\ \text{am} \\ \text{good} \end{matrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0.841 & 0.540 & 0.009 & 0.999 \\ 0.909 & -0.416 & 0.019 & 0.999 \end{bmatrix}$$

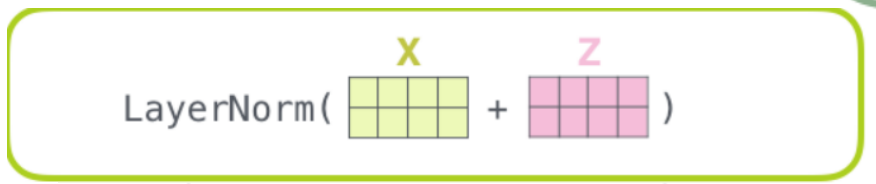
Figure 1.30 – Positional encoding matrix

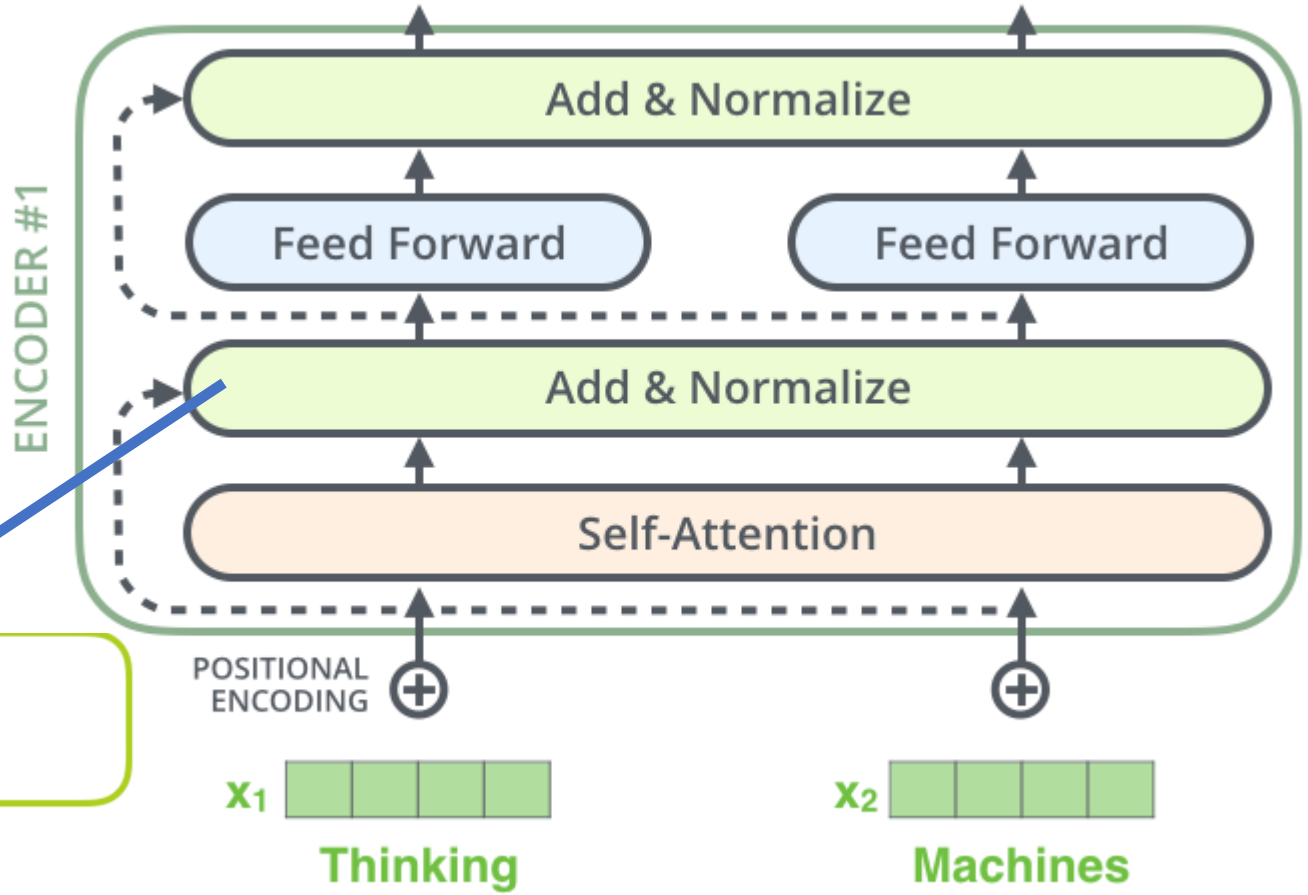
Residual connection, Layer Norm, FFN

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.

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

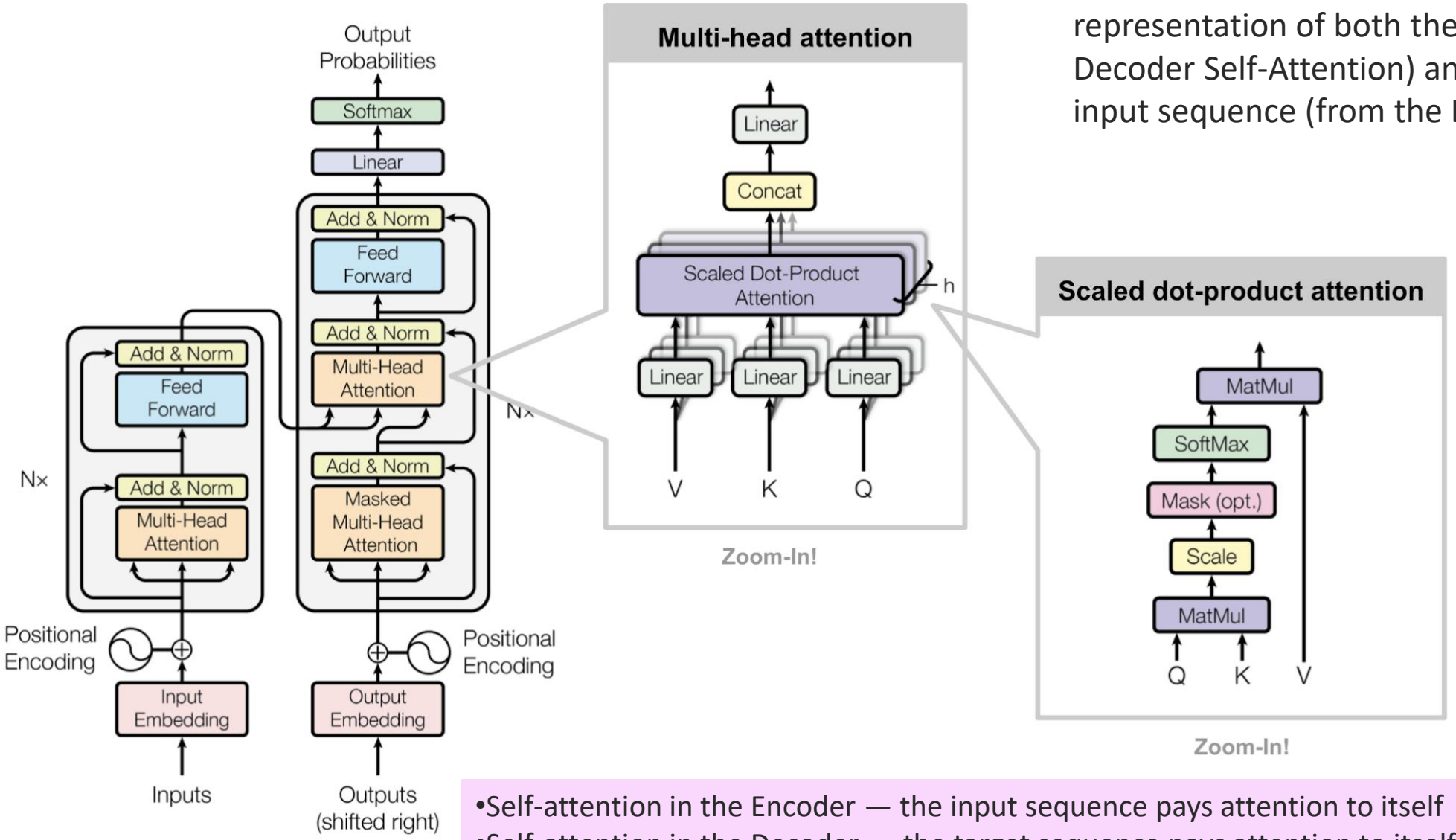




$$h_{ij}^{norm} = \frac{h_{ij} - \mu_j}{\sigma_j}$$

$$h_{ij}^{final} = \gamma_j \cdot h_{ij}^{norm} + \beta_j$$

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).

- 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

Encoder –Decoder Cross Attention

- The decoder generates tokens **one by one** and, at each step, it needs to determine **which encoder outputs to attend to**.
- To do this, the decoder uses **cross-attention**, which works as follows:

  Query (Q) → Comes from the decoder (its current hidden state).
  Key (K) **& Value** (V) → Come from the **encoder outputs** H_{enc}.

$$K = H_{\text{enc}} W_K$$

$$V = H_{\text{enc}} W_V$$

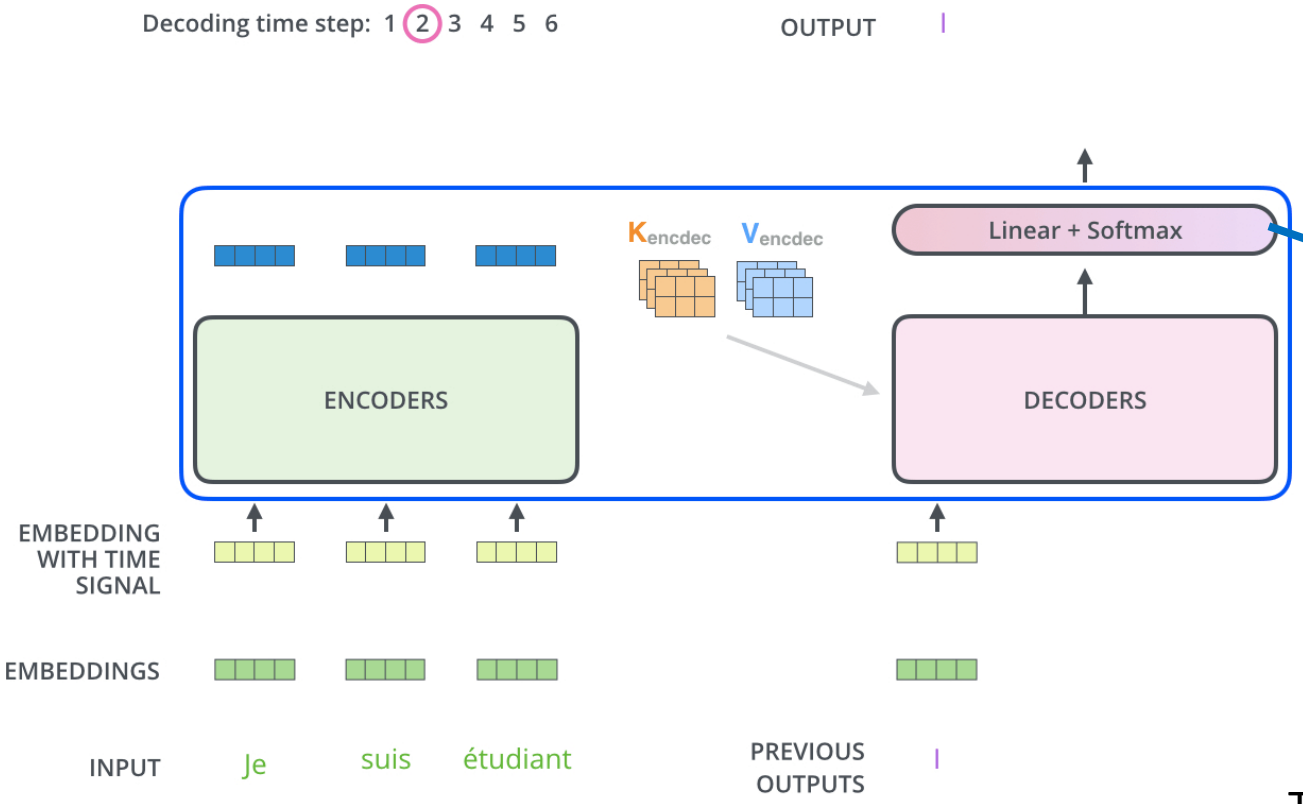
- The cross-attention mechanism computes **attention scores** using the **scaled dot-product attention**:

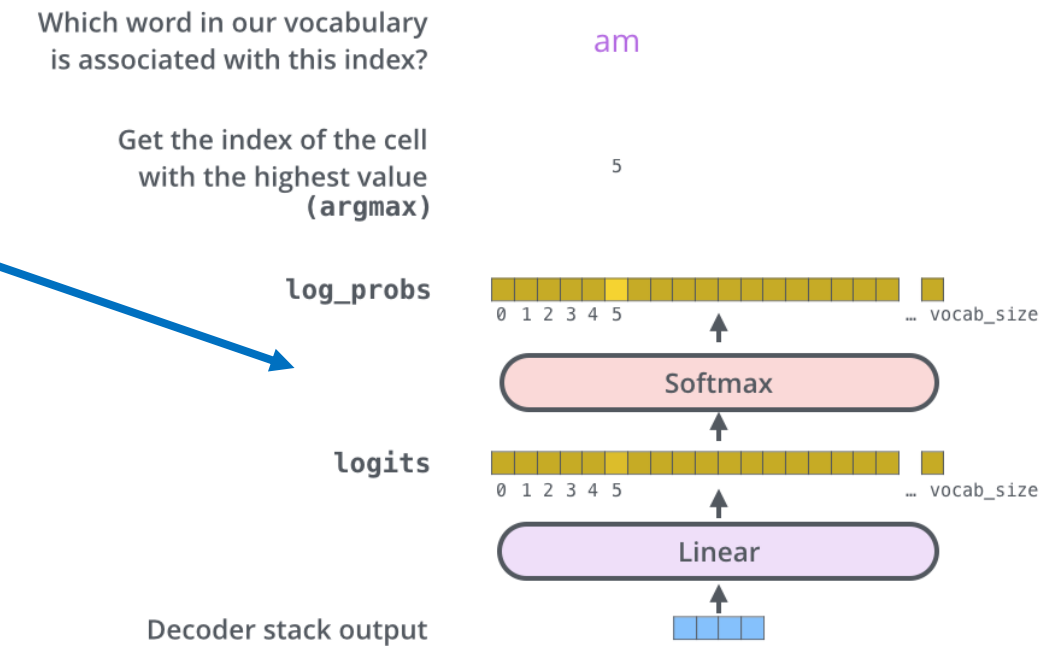
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

  - The **dot product** QK^T measures how relevant each encoder output h_i is to the current decoder query.
  - The **softmax** function normalizes these scores into attention weights.
  - The final output is a **weighted sum** of the encoder representations.

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.

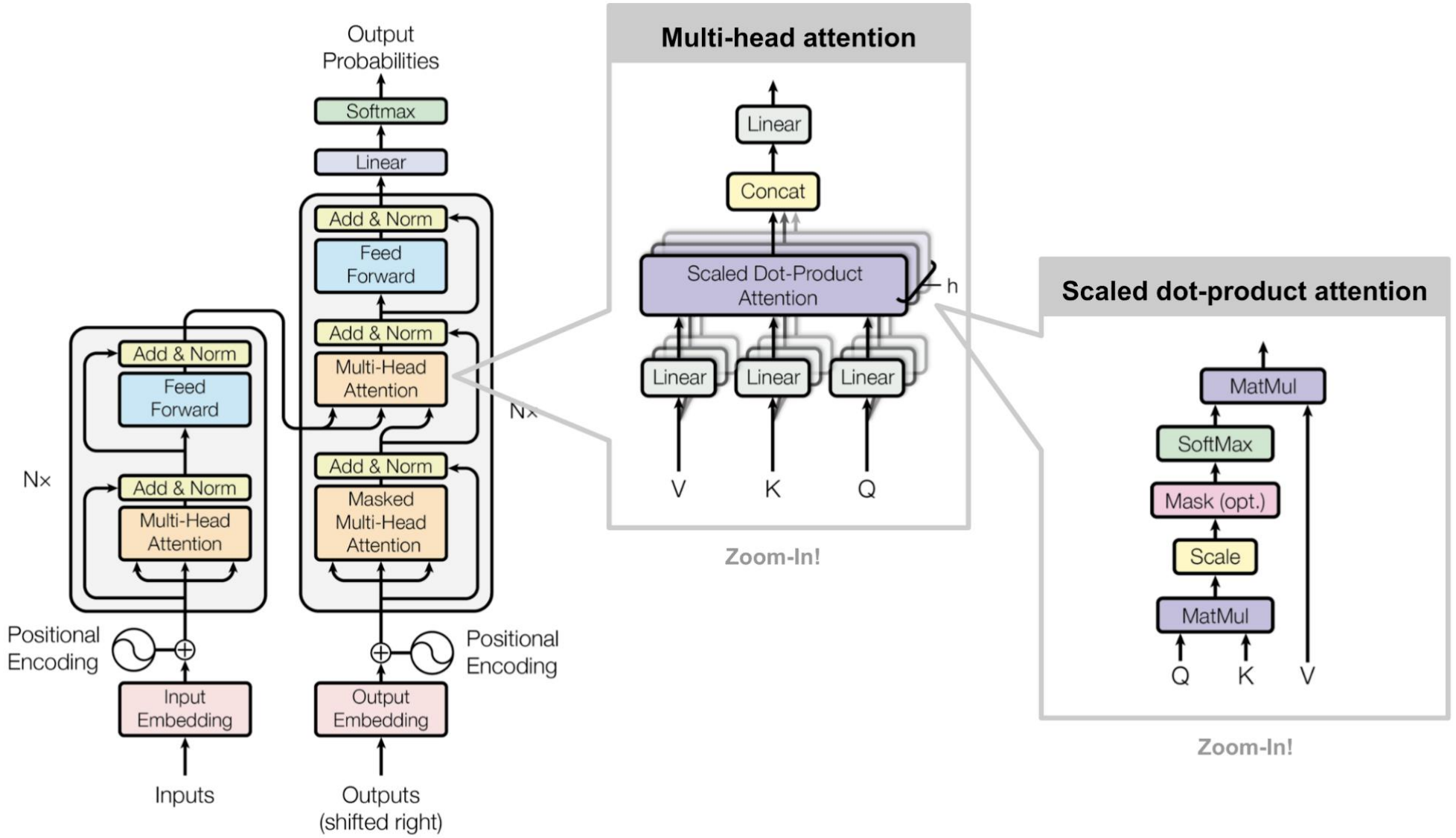




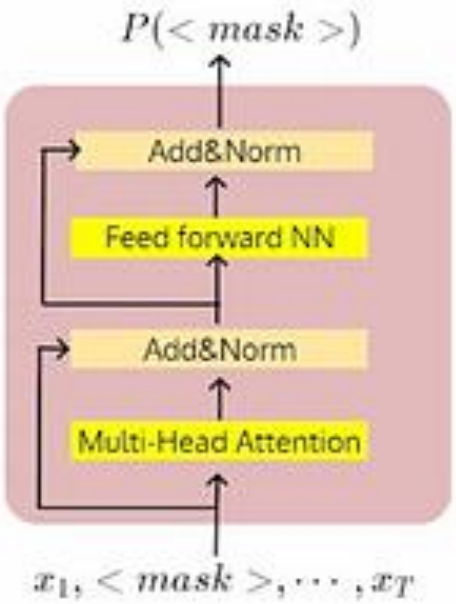
The self attention layers in the decoder operate in a slightly different way than the one in the encoder:

The decoder's output is passed through a **linear layer** to map it from **dmodel** dimensions to **vocabulary size**.

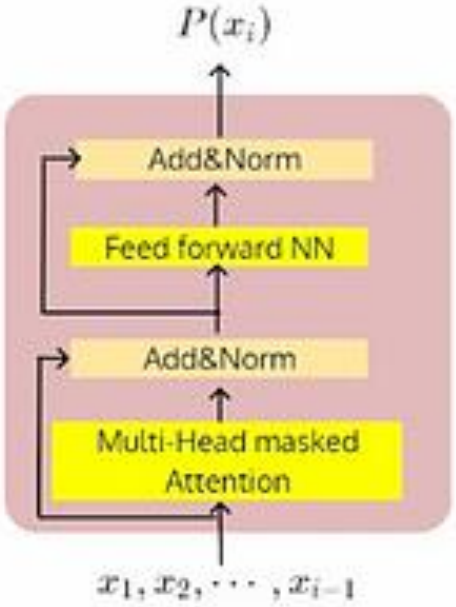
Are you clear with all the blocks?



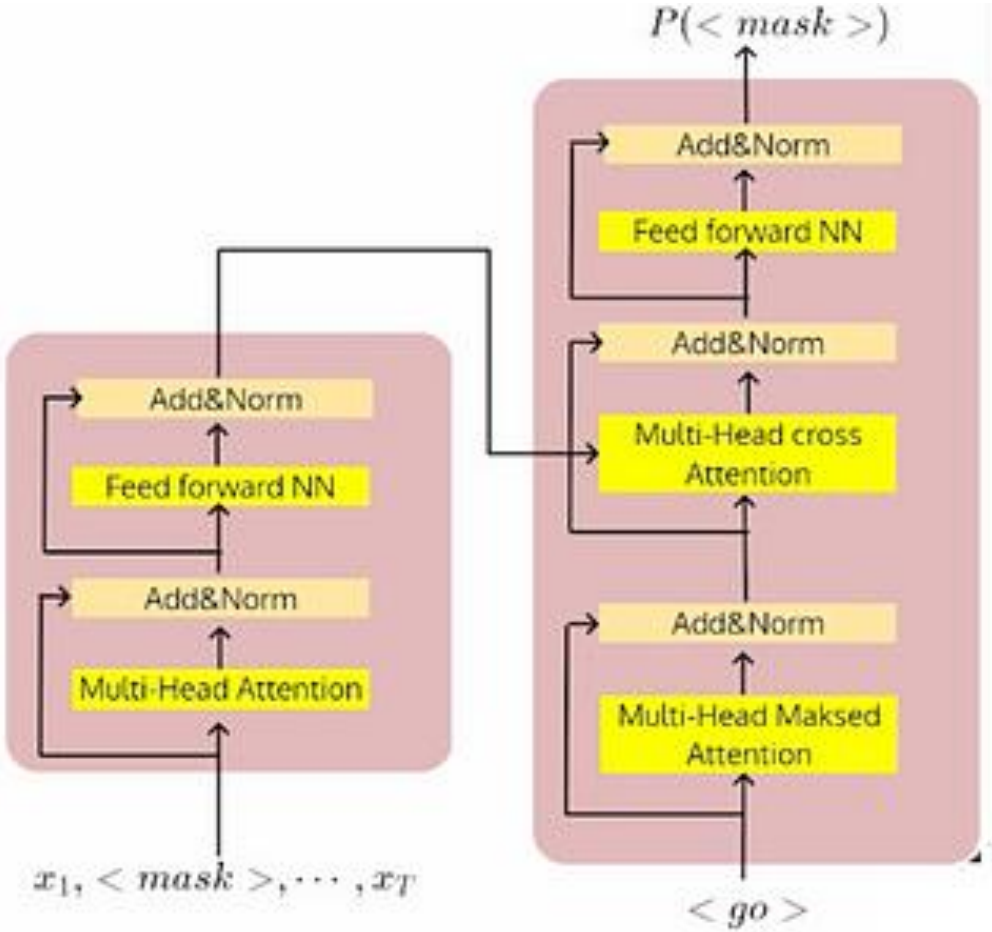
Some Possibilities



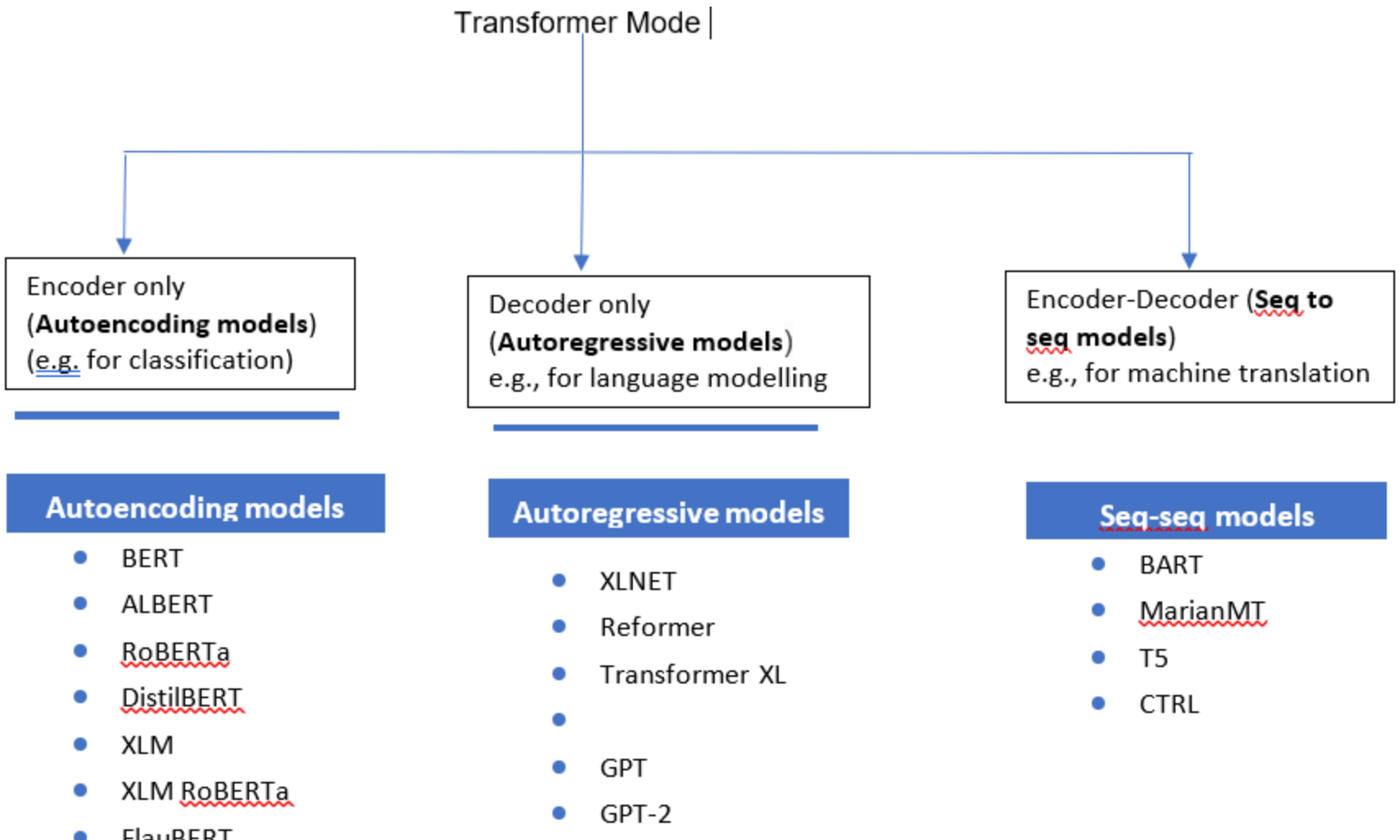
Using only the encoder of the transformer (encoder only models)



Using only the decoder of the transformer (decoder only models)



Using both the encoder and decoder of the transformer (encoder decoder models)



Longformer, Linformer, BigBird, Reformer, Long Range Arena, Poolingformer, conformer,GPT 3.5, GPT 4,4.5

Outline

KeAi
CHINESE ROOTS
GLOBAL IMPACT

AI Open

Volume 3, 2022, Pages 111-132

A survey of transformers

Tianyang Lin, Yuxin Wang, Xiangyang Liu, Xipeng Qiu

Show more

https://doi.org/10.1016/j.aiopen.2022.10.001

Get rights and content

Under a Creative Commons license

open access

Abstract

Transformers have achieved great success in many artificial intelligence fields, such as natural language processing, computer vision, and audio processing. Therefore, it is natural to attract lots of interest from academic and industry researchers. Up to the

Recommended art

Causes of transforme
diagnostic methods
Renewable and Sustainabl
Christina AJ, ..., William V

Purchase PDF

Self-directed machir
AI Open, Volume 3, 2022,
Wenwu Zhu, ..., Pengtao

View PDF

Hierarchical label wi
attributed network s
AI Open, Volume 3, 2022,
Shu Zhao, ..., Jie Tang

View PDF

Show 3 more articles

Computer Science > Computation and Language

[Submitted on 12 Feb 2023 (v1), last revised 16 Feb 2023 (this version, v2)]

Transformer models: an introduction and catalog

Xavier Amatriain

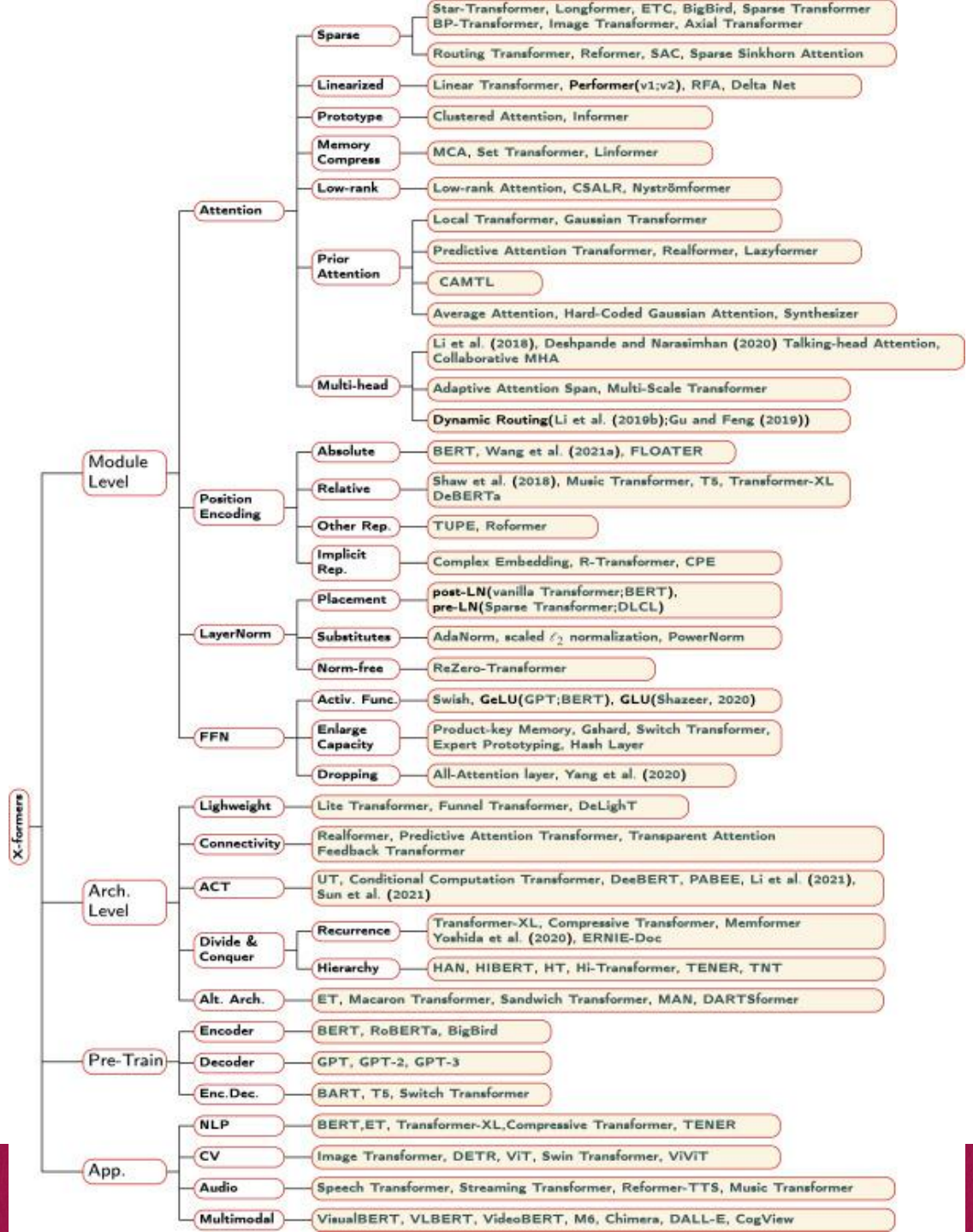
In the past few years we have seen the meteoric appearance of dozens of models of the Transformer family, all of which have funny, but not self-explanatory, names. The goal of this paper is to offe somewhat comprehensive but simple catalog and classification of the most popular Transformer models. The paper also includes an introduction to the most important aspects and innovation in Transformer models.

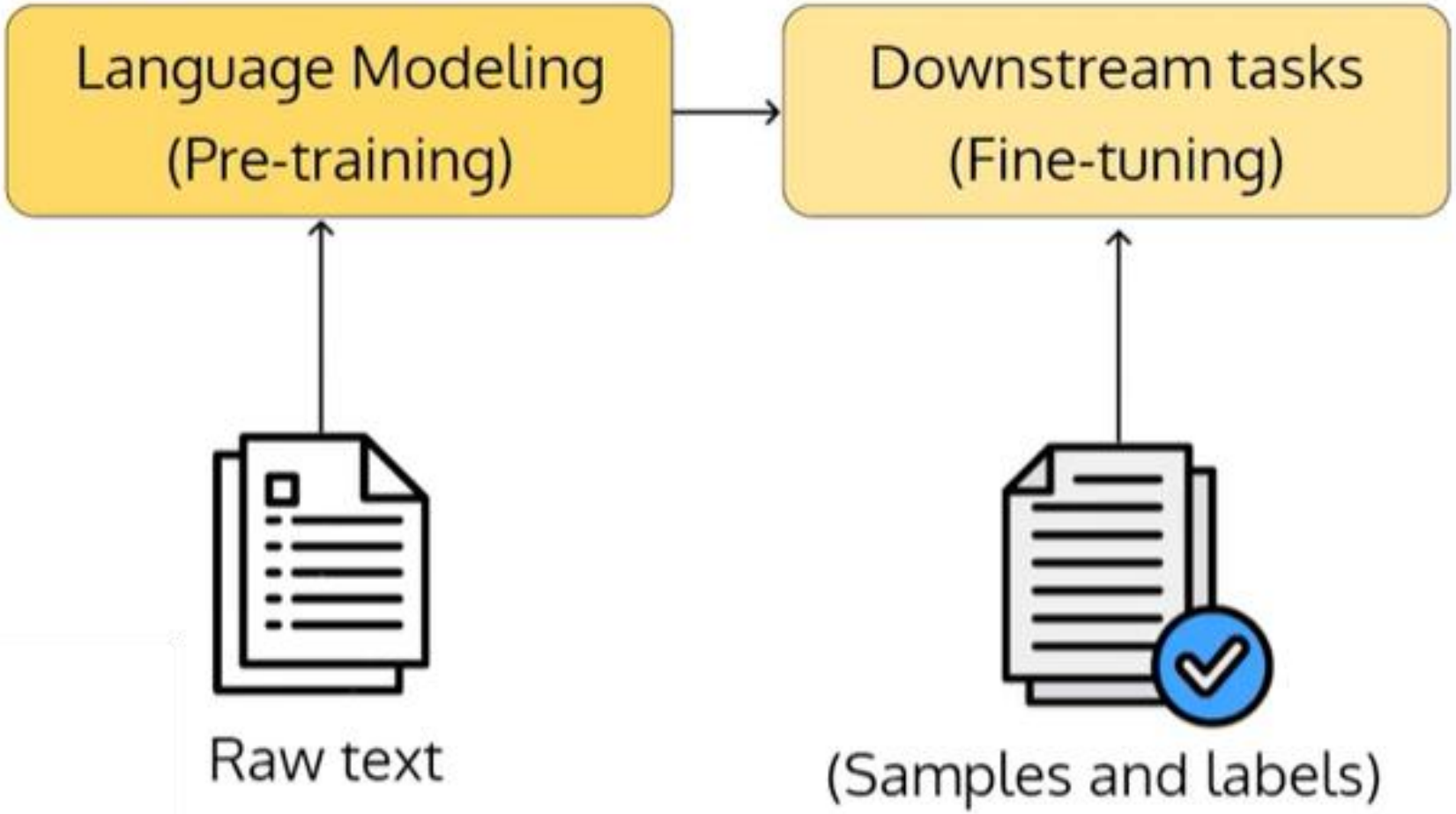
Subjects: **Computation and Language (cs.CL)**
Cite as: arXiv:2302.07730 **[cs.CL]**
(or arXiv:2302.07730v2 **[cs.CL]** for this version)
https://doi.org/10.48550/arXiv.2302.07730

Submission history

From: Xavier Amatriain [view email]
[v1] Sun, 12 Feb 2023 01:26:49 UTC (3,568 KB)
[v2] Thu, 16 Feb 2023 05:31:15 UTC (3,568 KB)

https://arxiv.org/pdf/2302.07730v2.pdf


X-formers
Module Level
Attention
Sparse
Star-Transformer, Longformer, ETC, BigBird, Sparse Transformer
BP-Transformer, Image Transformer, Axial Transformer
Routing Transformer, Reformer, SAC, Sparse Sinkhorn Attention
Linearized
Linear Transformer, Performer(v1;v2), RFA, Delta Net
Prototype
Clustered Attention, Informer
Memory Compress
MCA, Set Transformer, Linformer
Low-rank
Low-rank Attention, CSALR, Nyströmformer
Prior Attention
Local Transformer, Gaussian Transformer
Predictive Attention Transformer, Realformer, Lazyformer
CAMTL
Average Attention, Hard-Coded Gaussian Attention, Synthesizer
Multi-head
Li et al. (2018), Deshpande and Narasimhan (2020) Talking-head Attention, Collaborative MHA
Adaptive Attention Span, Multi-Scale Transformer
Dynamic Routing(Li et al. (2019b);Gu and Feng (2019))
Position Encoding
Absolute
BERT, Wang et al. (2021a), FLOATER
Relative
Shaw et al. (2018), Music Transformer, T5, Transformer-XL DeBERTa
Other Rep.
TUPE, Roformer
Implicit Rep.
Complex Embedding, R-Transformer, CPE
LayerNorm
Placement
post-LN(vanilla Transformer;BERT), pre-LN(Sparse Transformer;DLCL)
Substitutes
AdaNorm, scaled ℓ_2 normalization, PowerNorm
Norm-free
ReZero-Transformer
FFN
Activ. Func.
Swish, GeLU(GPT;BERT), GLU(Shazeer, 2020)
Enlarge Capacity
Product-key Memory, Gshard, Switch Transformer, Expert Prototyping, Hash Layer
Dropping
All-Attention layer, Yang et al. (2020)
Arch. Level
Lighweight
Lite Transformer, Funnel Transformer, DeLighT
Connectivity
Realformer, Predictive Attention Transformer, Transparent Attention Feedback Transformer
ACT
UT, Conditional Computation Transformer, DeeBERT, PABEE, Li et al. (2021), Sun et al. (2021)
Divide & Conquer
Recurrence
Transformer-XL, Compressive Transformer, Memformer Yoshida et al. (2020), ERNIE-Doc
Hierarchy
HAN, HIBERT, HT, Hi-Transformer, TENER, TNT
Alt. Arch.
ET, Macaron Transformer, Sandwich Transformer, MAN, DARTSformer
Pre-Train
Encoder
BERT, RoBERTa, BigBird
Decoder
GPT, GPT-2, GPT-3
Enc.Dec.
BART, T5, Switch Transformer
App.
NLP
BERT,ET, Transformer-XL,Compressive Transformer, TENER
CV
Image Transformer, DETR, ViT, Swin Transformer, ViViT
Audio
Speech Transformer, Streaming Transformer, Reformer-TTS, Music Transformer
Multimodal
VisualBERT, VLBERT, VideoBERT, M6, Chimera, DALL-E, CogView


Language Modeling
(Pre-training)
Downstream tasks
(Fine-tuning)
Raw text
(Samples and labels)

Down-stream task

What if we want to use the transformer architecture for other NLP tasks?

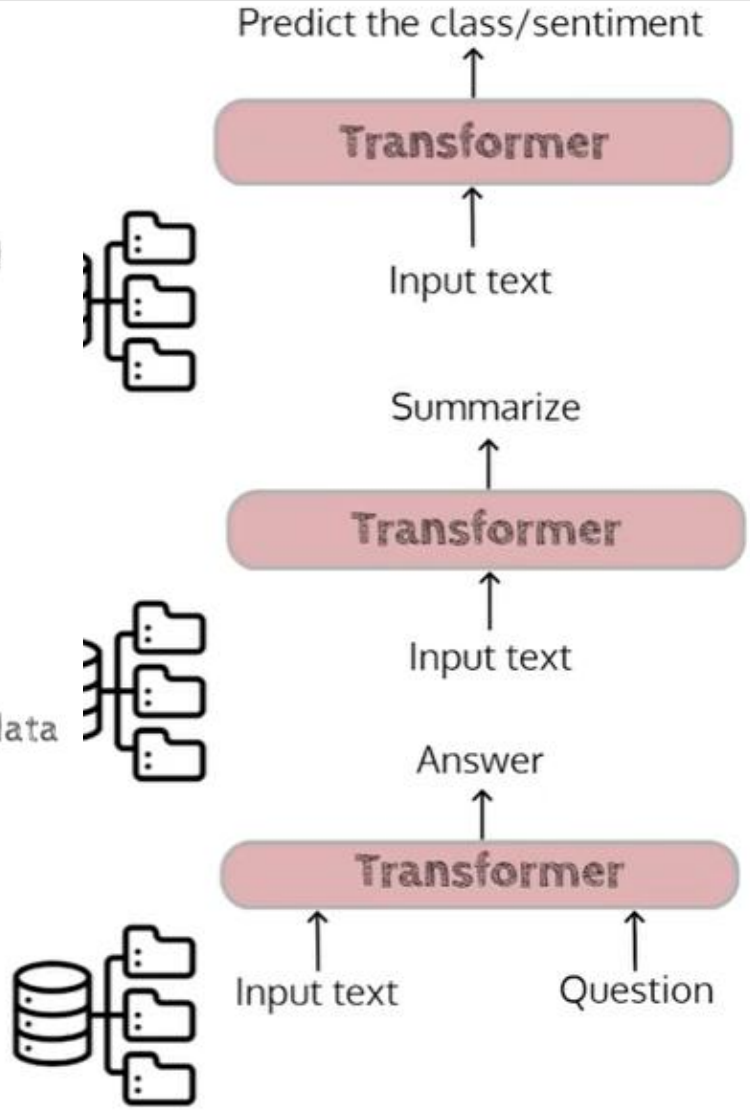
We need to train a separate model for each task using dataset specific to the task

If we train the architecture from scratch (that is, by randomly initializing the parameters) for each task, it takes a long time for convergence

Often, we may not have enough labelled samples for many NLP tasks

We have a large amount of unlabelled text easily available on the internet

Can we make use of such unlabelled data to train a model?



Assume that we ask questions to a lay person based on a statement or some excerpt

" Wow, India has now reached the moon"

An excerpt from business today "What sets this mission apart is the pivotal role of artificial intelligence (AI) in guiding the spacecraft during its critical descent to the moon's surface."

He likes to stay
He likes to stray
He likes to sway

Is this sentence expressing a positive or a negative sentiment?

Did the lander use AI for soft landing on the moon?

Are these meaningful sentences?

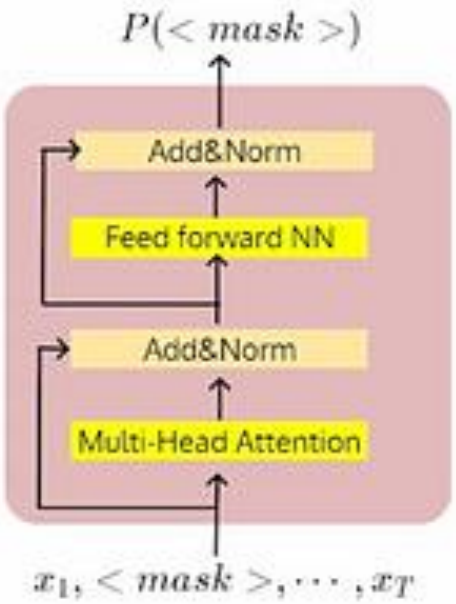
The person will most likely answer all the questions, even though he/she may not be explicitly trained on any of these tasks. How?

We develop a strong understanding of language through various language based interactions(listening/reading) over our life time without any explicit supervision

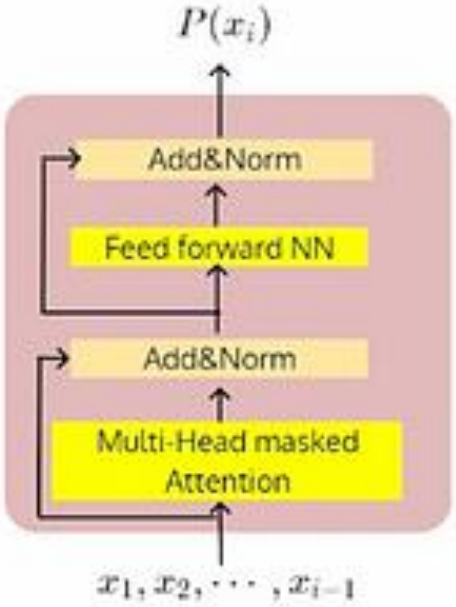
Can a model develop basic understanding of language by getting exposure to a large amount of raw text? [pre-training]

More importantly, after getting exposed to such raw data can it learn to perform well on downstream tasks with minimum supervision? Supervised fine tuning

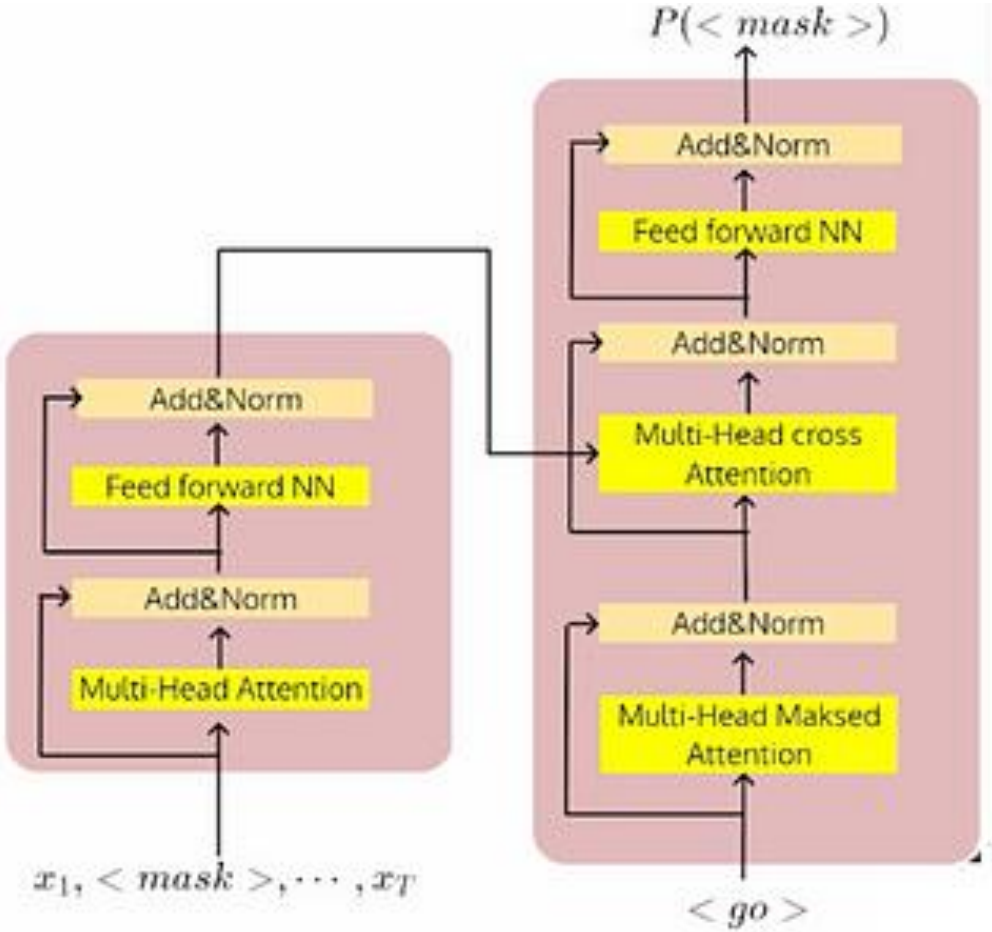
Some Possibilities



Using only the encoder of the transformer (encoder only models)



Using only the decoder of the transformer (decoder only models)



Using both the encoder and decoder of the transformer (encoder decoder models)

Namah Shivaya