Transformers Explained

→ Self - Attention Mechanism

By: Kanav Bansal
(That AI Guy)

By: Karan Bansal
(That AI Guy)

Summary ↴

2014 → Sequence to Sequence Learning with Neural Network
Proposed – Encoder Decoder architecture

2015 → Neural Machine Translation with Joint Learning to align & Translate
Proposed – Bahdanau Attention Mechanism

2017 → Attention is all you need
Proposed – Transformer Architecture with Self-Attention

2018 → Universal Language Model Fine-Tuning for Text Classification
Proposed – Language Modeling as pre-training Task

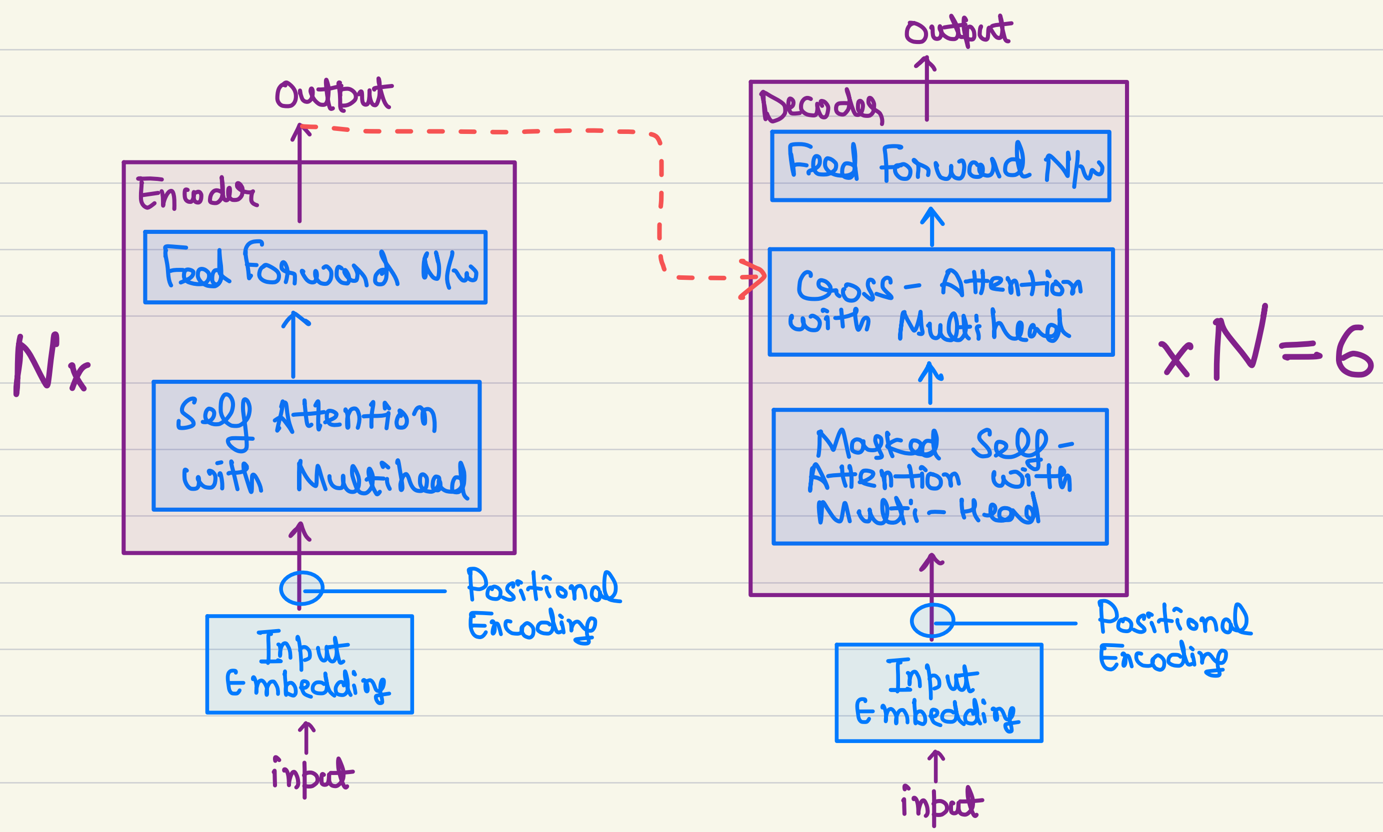
By: Kanav Bansal
(That AI Guy)

Paper: Attention is all you need

2017 → Proposed Transformer Architecture
which used:
→ Encoder Decoder Architecture
→ Self - Attention Mechanism

BUILDING BLOCKS

(Note: This is a simplified view of Transformer)



Encoder is good at understanding text

Decoder is good at generating text

ENCODER BUILDING BLOCK

1. Positional Embedding
   - Adds the notion of seq info

2. Self - Attention with Multihead
   - Attention replaced need of recurrence & makes the process of embedding calculation parallel
   - Attention mechanism will find how each word relates to all other words in a sequence & generates contextual Embeddings

Ques: How to find word similarities?
- Attention will run dot products b/w word vectors & determine the strongest relationships of a word with all other words

Ques: How to speed up the calculations?
- For each attention sublayer, the original transformer model runs not only one but eight attention mechanisms in parallel to speed up the calculations. This process is done using 'multi-head attention'.

3. Fully Connected Positionwise Feed Forward N/w
   - Improves the word association by applying non-linear transformations.

By: Kanav Bansal
(That AI Guy)

TRANSFORMER'S

↓

Attention is all you need
(2017 Paper)

Why CELEB STATUS ?

1. Scalable & Parallel Training
2. Revolutionized NLP with LLMs
3. Unified DL Approaches for text, image, audio & video data
4. Multi-Modal
5. Accelerated GenAI

Encoder	Decoder
Has self-Attention with multi-head & Feed forward ANN	Mask self-Attention & Cross-Attention with Multi-head & Feed forward ANN
↑	↑
Great at understanding Text	Great at Generating Text
Eg: BERT	Eg: GPT-1

By: Karan Bansal
(That AI Guy)

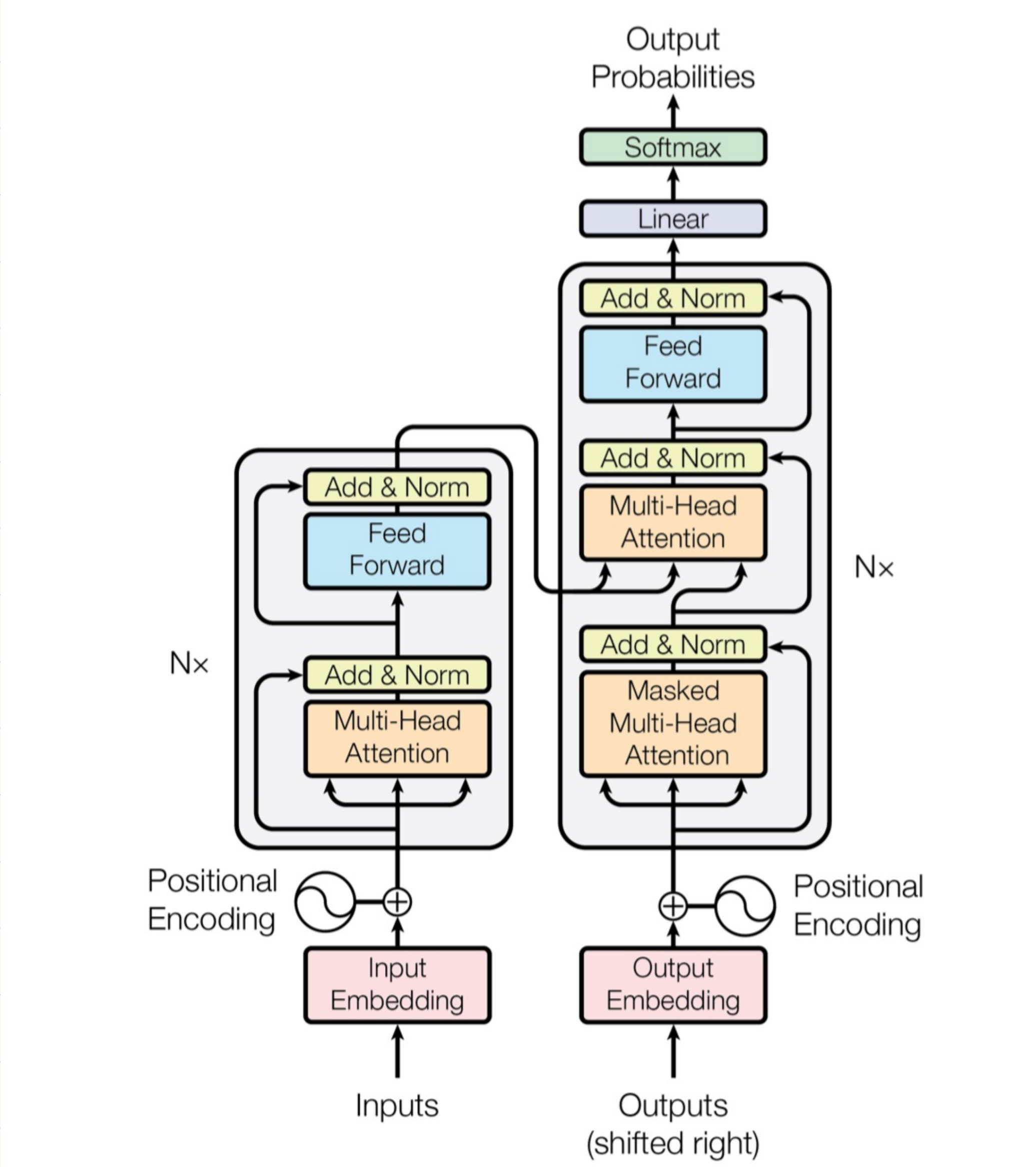


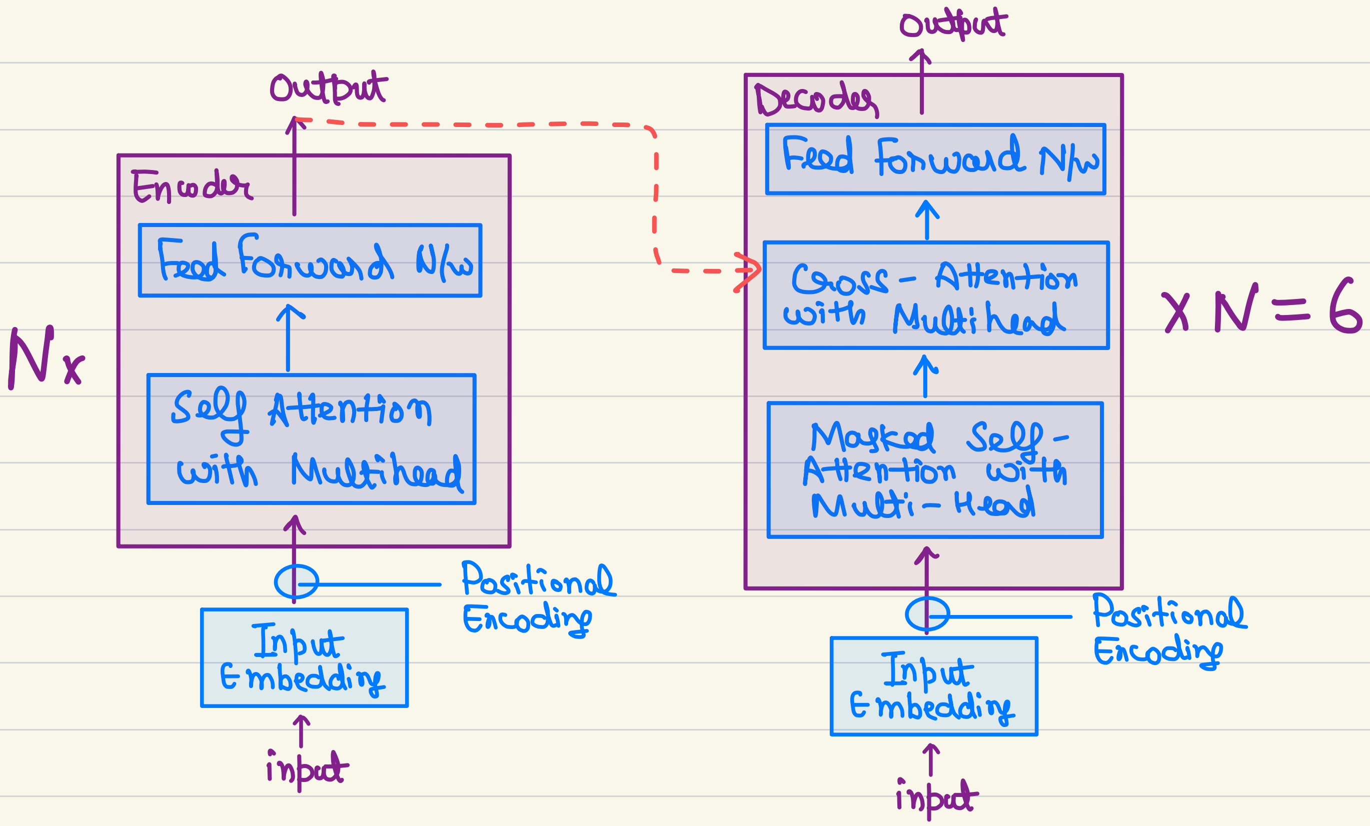
Figure 1: The Transformer - model architecture.

Advantages →
- Parallel & Scalable
- Multi Modal input & output

Disadvantages →
- Huge compute resources
- Huge amount of data
- Overfit
- Energy

BUILDING BLOCKS

By: Kanav Bansal
(That AI Guy)

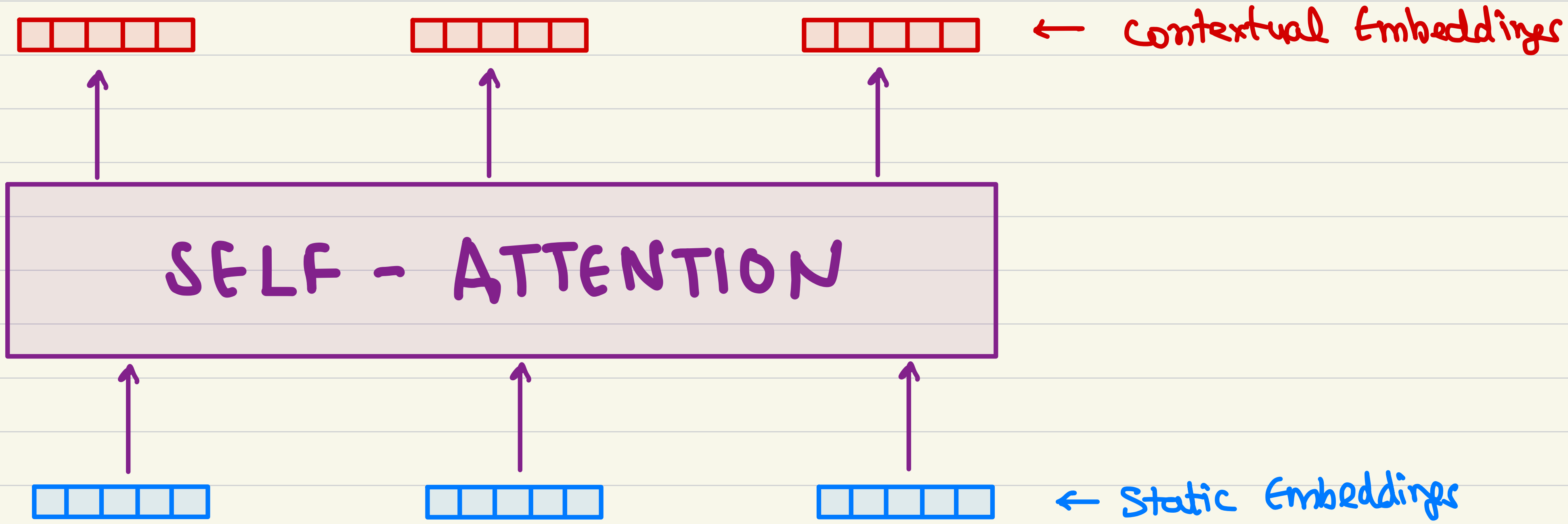


Encoder is good at understanding text

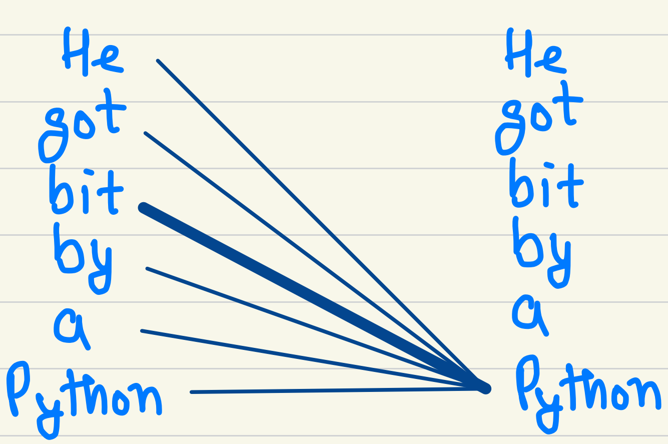
Decoder is good at generating text

By: Karan Bansal
(That AI Guy)

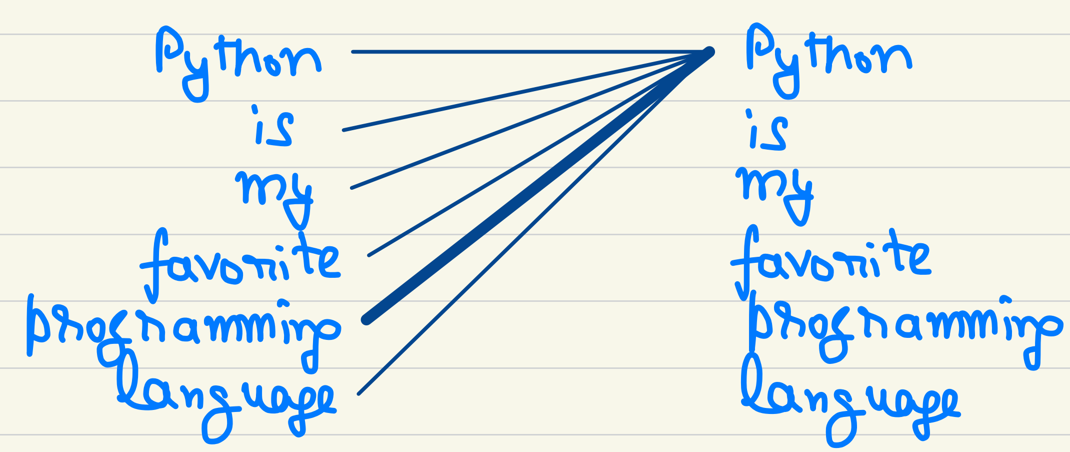
Understanding Self-Attention



Ex: Sent A → He got bit by a Python



Sent B → Python is my favorite programming language



By: Kanav Bansal
(That AI Guy)

Example ↴

Sentence 1 : Python executes quickly

Sentence 2 : Python crawls quickly

$$\text{Python} = 0.5\ \text{Python} + 0.3\ \text{executes} + 0.2\ \text{quickly}$$
$$\text{executes} = 0.2\ \text{Python} + 0.7\ \text{executes} + 0.1\ \text{quickly}$$
$$\text{quickly} = 0.1\ \text{Python} + 0.1\ \text{executes} + 0.8\ \text{quickly}$$

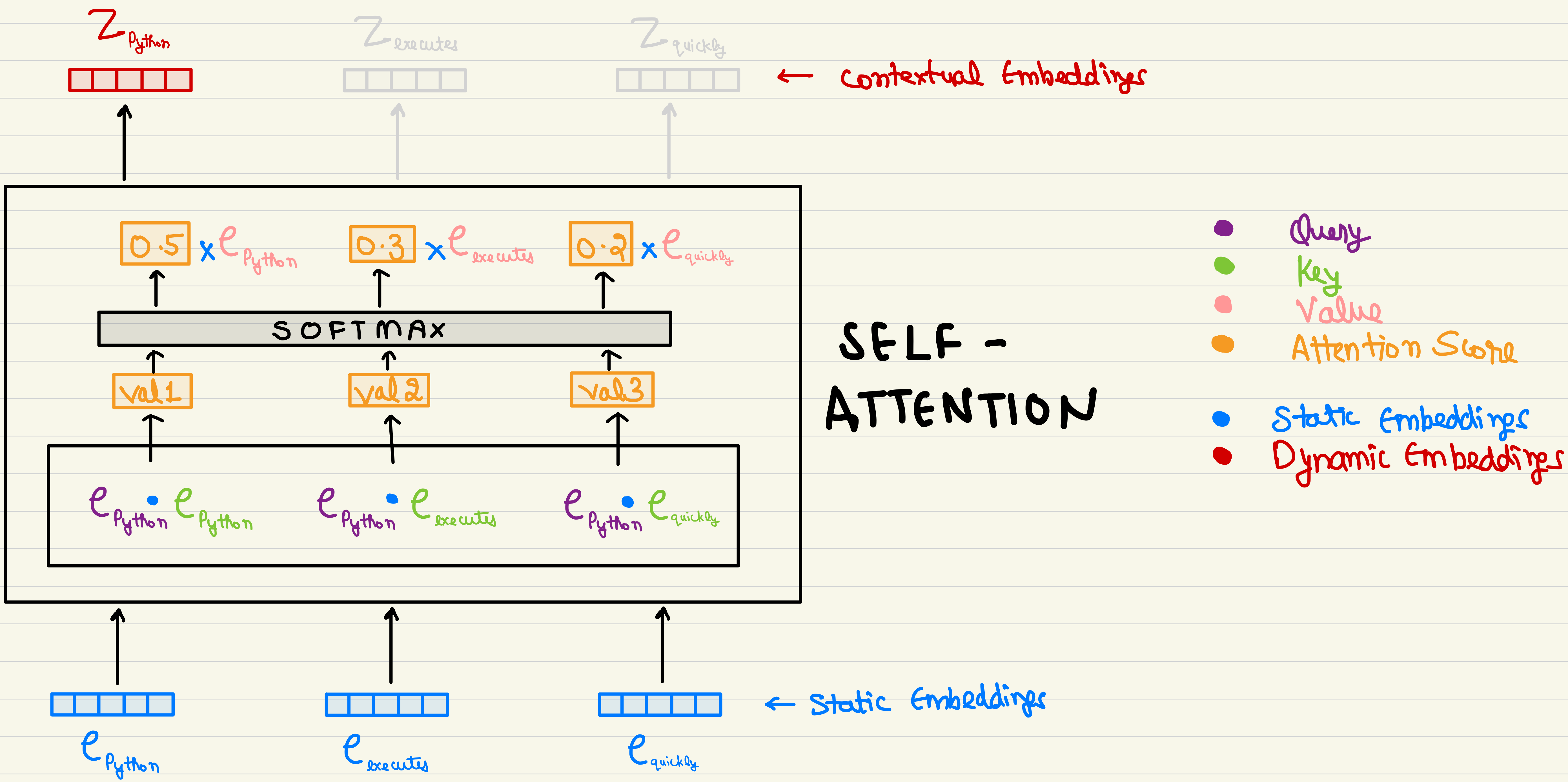
$$\text{Python} = 0.4\ \text{Python} + 0.4\ \text{crawls} + 0.2\ \text{quickly}$$
$$\text{crawls} = 0.3\ \text{Python} + 0.6\ \text{crawls} + 0.1\ \text{quickly}$$
$$\text{quickly} = 0.1\ \text{Python} + 0.1\ \text{crawls} + 0.8\ \text{quickly}$$

$$Z_{Python} = \overset{e_{Python} \cdot e_{Python}}{\overset{\uparrow}{\boxed{0.5}}}\ e_{Python} + \overset{e_{Python} \cdot e_{executes}}{\overset{\uparrow}{\boxed{0.3}}}\ e_{executes} + \overset{e_{Python} \cdot e_{quickly}}{\overset{\uparrow}{\boxed{0.2}}}\ e_{quickly}$$
$$Z_{executes} = 0.2\ e_{Python} + 0.7\ e_{executes} + 0.1\ e_{quickly}$$
$$Z_{quickly} = \underset{e_{quickly} \cdot e_{Python}}{\underset{\downarrow}{\boxed{0.1}}}\ e_{Python} + 0.1\ e_{executes} + 0.8\ e_{quickly}$$

$$Z_{Python} = 0.4\ e_{Python} + \overset{e_{Python} \cdot e_{crawls}}{\overset{\uparrow}{\boxed{0.4}}}\ e_{crawls} + 0.2\ e_{quickly}$$
$$Z_{crawls} = 0.3\ e_{Python} + 0.6\ e_{crawls} + 0.1\ e_{quickly}$$
$$Z_{quickly} = 0.1\ e_{Python} + 0.1\ e_{crawls} + 0.8\ e_{quickly}$$

By: Kanav Bansal
(That AI Guy)

The complete process → (Calculation for the "Python" Query)



By: Karan Bansal
(That AI Guy)

Question: What is Q, K and V?

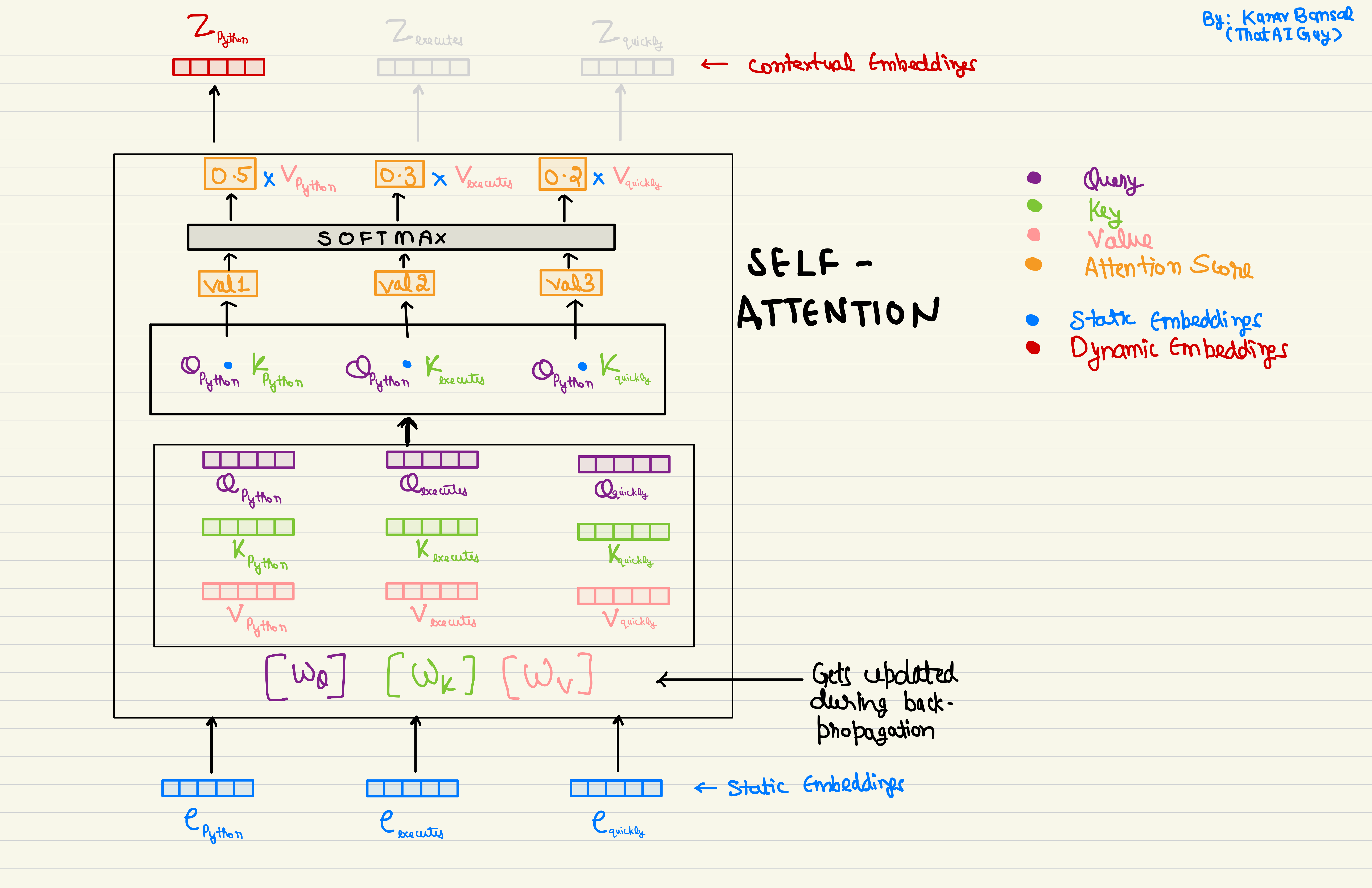
- Query represents what information the model is trying to extract.
- Key represents the content or features that the model can pay attention to.
- Value represents the actual information that is passed along once attention has been calculated.

By computing the dot product between the Query of one token and the Key of another, the model determines the relevance or importance of one token to another. The resulting attention scores are then used to weigh the Values, creating a weighted sum that influences how much attention is paid to different parts of the sequence.

Question: Why Q, K and V?

- If you were to use the token embeddings directly without this transformation into Query, Key, and Value, the model would be limited to just one fixed way of computing relationships between tokens.
- By introducing separate linear transformations for Query, Key, and Value, the model gains flexibility. It allows the model to learn different representations of relationships, enabling more nuanced and complex interactions between tokens. This separation is crucial for capturing different types of relationships, like syntactic dependencies or semantic connections, within the input sequence.

Transformers typically use multi-head attention, where multiple sets of Query, Key, and Value projections are learned independently. This allows the model to focus on different aspects of the input sequence simultaneously, capturing a richer set of relationships.
Without the QKV separation, the model wouldn't be able to perform these parallel attention operations, which are essential for capturing diverse patterns in data.



By: Karan Bansal (That AI Guy)
SELF - ATTENTION
Z Python
Z executes
Z quickly
← contextual Embeddings
0.5 x V Python
0.3 x V executes
0.2 x V quickly
SOFTMAX
val 1
val 2
val 3
Q Python • K Python
Q Python • K executes
Q Python • K quickly
Q Python
Q executes
Q quickly
K Python
K executes
K quickly
V Python
V executes
V quickly
[W Q] [W K] [W V]
Gets updated during back-propagation
e Python
e executes
e quickly
← Static Embeddings
Query
Key
Value
Attention Score
Static Embeddings
Dynamic Embeddings

SELF ATTENTION ↴

By: Karan Bansal
(That AI Guy)

These weights get updated at training time

1. init the 3 weight matrix

$\boxed{W_q}$ $\boxed{W_K}$ $\boxed{W_V}$

512 ↓ Shape = $D \times d$ ↑ 64

2. Calc 1 :

$Q = XW_q$ $K = XW_K$ $V = XW_V$

$[Q]_{T \times d}$ $[K]_{T \times d}$ $[V]_{T \times d}$

Shape $= (T \times D) \times (D \times d)$
$= T \times d$

3. Calc2 :

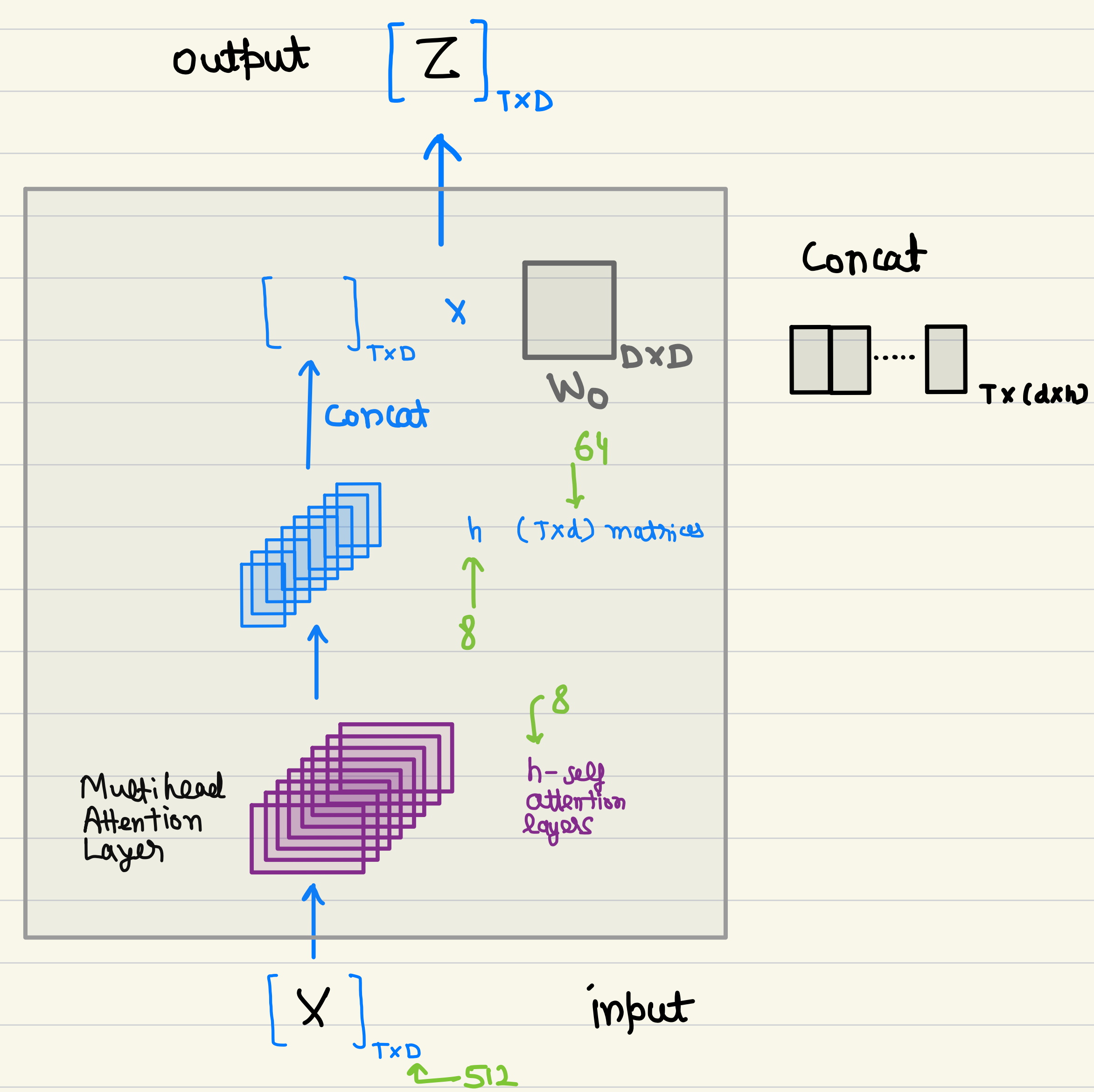
$$\text{Softmax}\left(\frac{Q K^T}{\sqrt{d}}\right) V$$

This returns a matrix of shape T×T.
Intuitively, we can think of this matrix as the importance of each word on another.
AKA Attention Score

Shape $= ((T \times d) \times (d \times T)) \times (T \times d)$
$= (T \times T) \times (T \times d)$
$= (T \times d)$

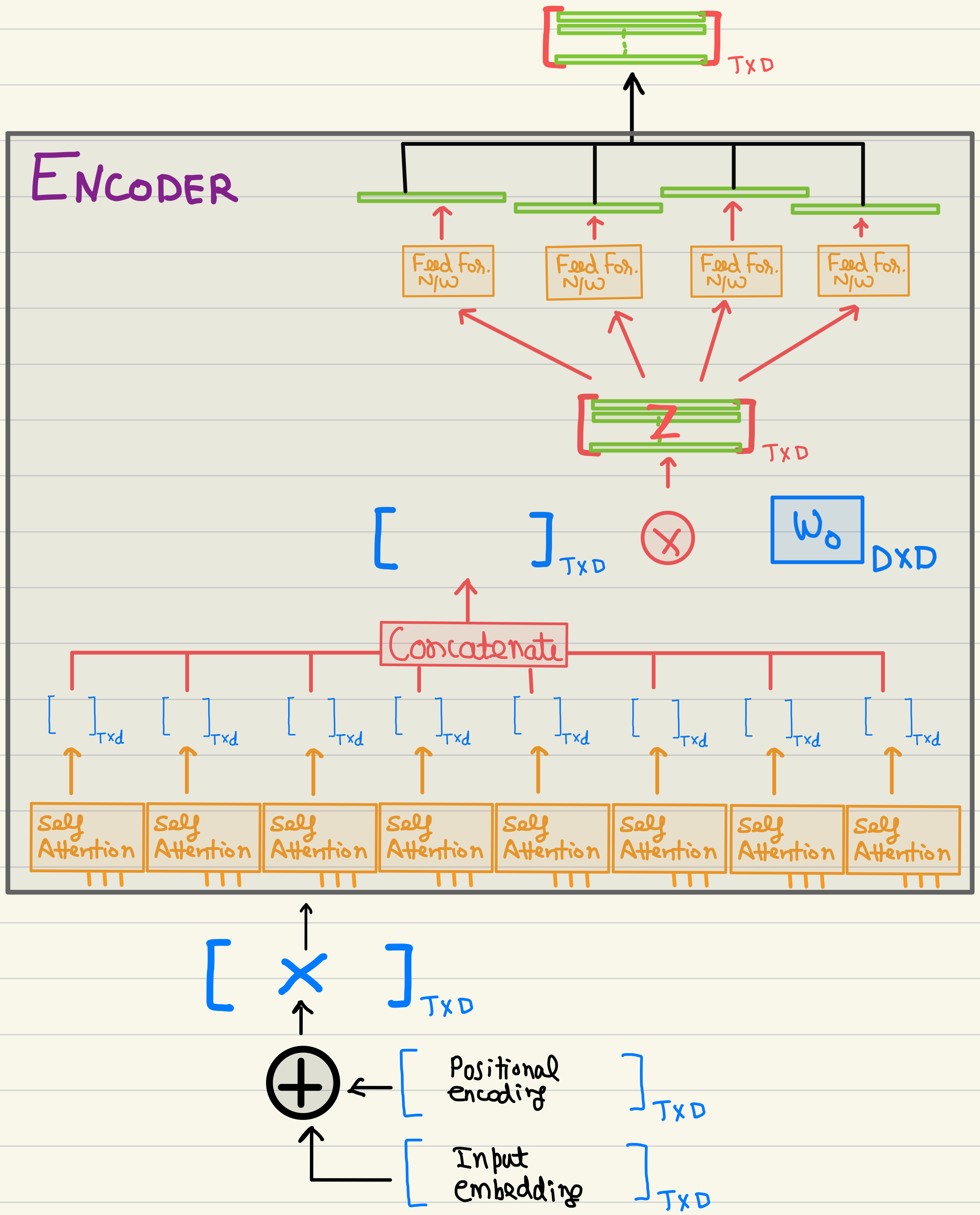
MULTI-HEAD SELF-ATTENTION ↴

As it uses multiple layers of self-attention in parallel



ENCODER

By: Kanav Bansal
(That AI Guy)



$$PE_{(pos,2i)} = sin(pos/10000^{2i/D})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/D})$$