→ Natural Language Processing

going towards → transformers (different type of neural network architecture)

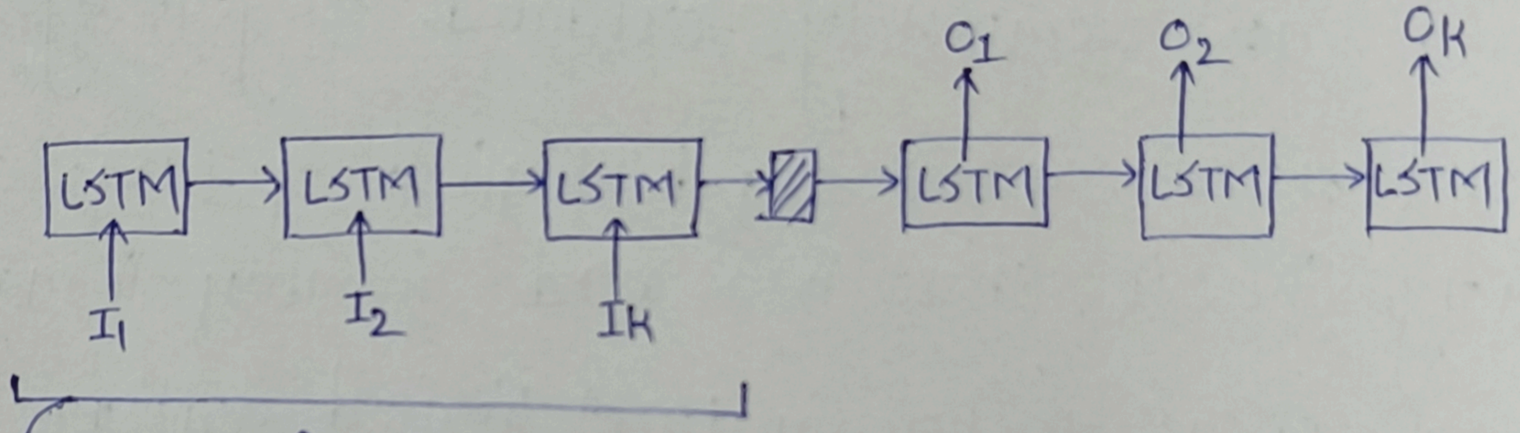
OR transformer architecture (majorly used for NLP & time series type of data)

contains → encoder & decoder unit

* → what is latent representation? (see Page-16)
* → what is <sos>? (see Page-15)

→ Recurrent Neural Network and long short Term Network

working model of LSTM



unrolling its own architecture with time and i/ps 'K'.

Here inputs are many and outputs are also so many so these type of networks are also known as "many to many" networks.

→ This specific problem / process known as "sequence to sequence" modelling. It is basically used for language translation. e.g. Hindi to English, German to Gujarati, and etc.

→ Issues OR problems with LSTM and RNN:

RNN has this issue → Long term dependencies are difficult to manage.

(1) long sentences (not able to give much more efficiency on it)

(2) large amount of memory requirement

(3) only one representation (Representation Bottleneck)

(4) attention or other model utilization

(5) recurrence (due to this it is difficult to solve sentences)

(6) regularization is difficult. (Prone to overFitting)

(7) exploding & vanishing problem, (8) sequentials i/p & o/p

LSTM & GRU are used to overcome this issue.

* Transformer can be subject.

→ Issues (1) and (7) are related to each other.

* → sir has said; we can solve exploding by using clipping? But how? what is it? (see Page-22)

→ LSTM can extract long term dependencies better than RNN. But at certain extent LSTM is also not working. To overcome this issue and want to extract more long term dependencies then more memory is required.

→ LSTM are now completely outdated. (1 year ago)

→ In LSTM stored context can be limited in terms of amount of memory. like if there is a word "it" then it may represent cat, dog, and microphone. so here in LSTM limited context can be stored. That is why there is an issue "only one representation" OR "representation bottleneck". ↖ solution of this issue

→ We also need to apply attention to LSTM due to its limited representation. Using LSTM along with attention is gives good representation in terms of context.

→ Major issue with the LSTM and RNN is recurrence / they are sequential in nature.

→ In the LSTM For regularization, Batch Normalization can not be used instead of it Layer Normalization is used. This is also issue with the transformers.

→ Batch Normalization vs. Layer Normalization

→ covariates in data
OR in batches / covariate shift

→ Difference b/w data of various batches

Batch Normalization is not applied ~~on~~ over the weights of the layer of neural network. For the regularization of weights of the layer(s) of neural network "L_1 & L_2 regularization" can be applied.

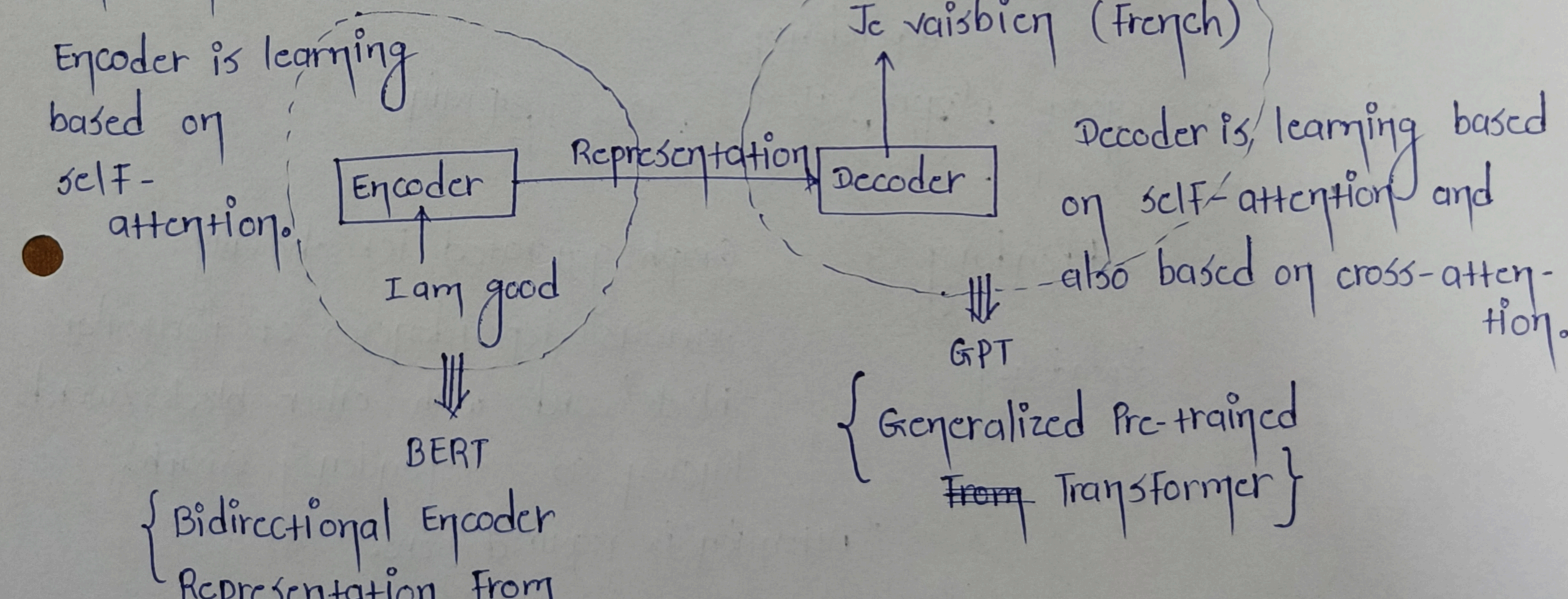
Handling the covariate shift is all about the batch normalization.

* → what is the truncated backpropogation in time? (See Page-20)

→ Natural Language Processing ≅ Transformer (Nowadays Reality)

→ Landmark paper: "Attention is all you need?"

Transformer Abstract View:



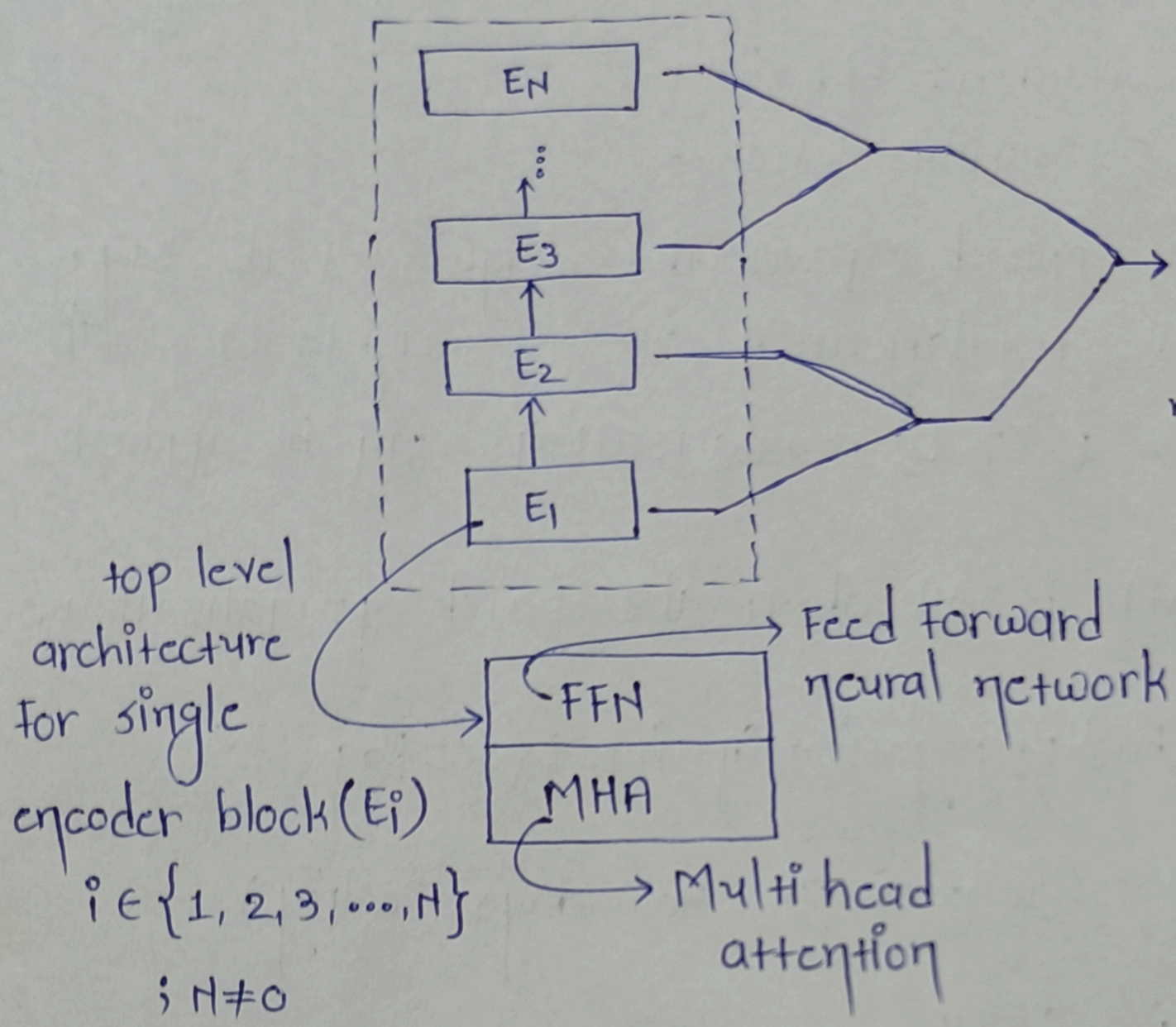
{ Bidirectional Encoder Representation From Transformer }

{ Generalized Pre-trained ~~From~~ Transformer }

* → what is difference b/w Encoder architecture in transformer and auto encoder architecture?

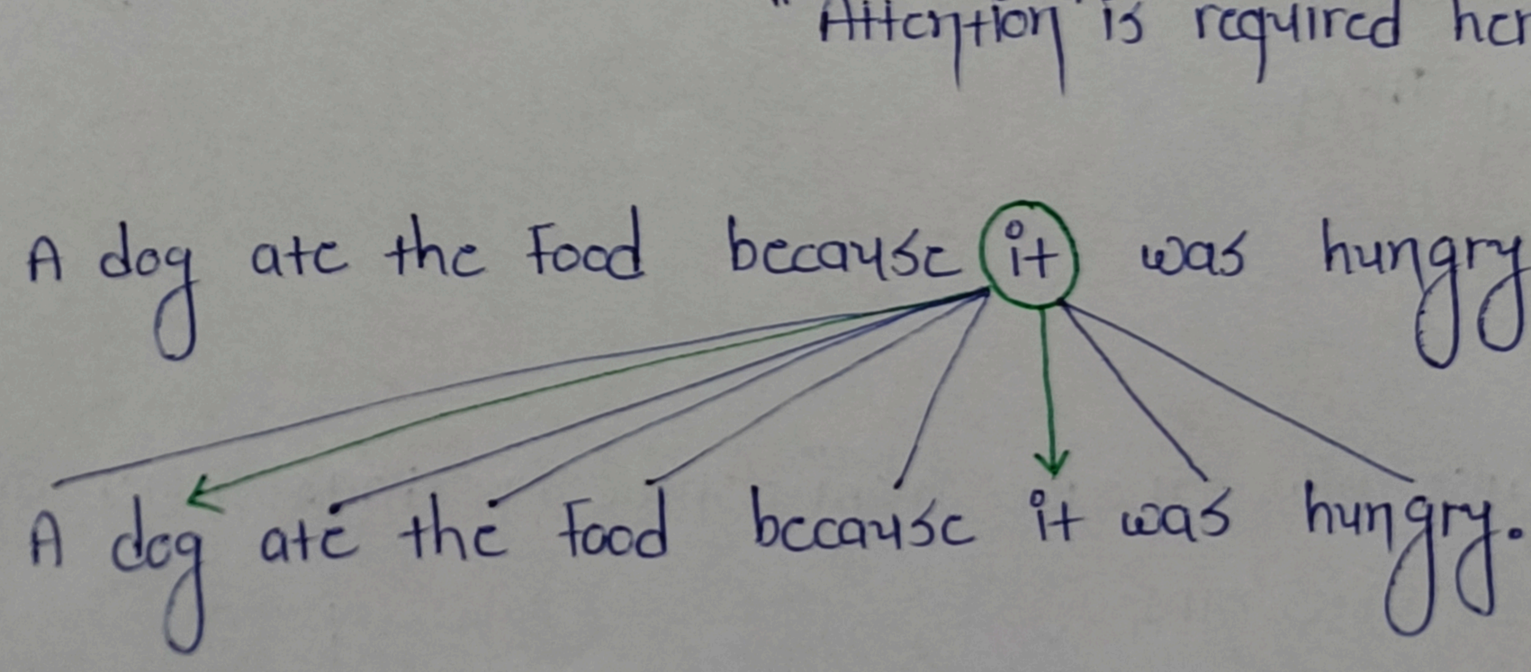
→ Encoder explanation:

Use "top level architecture" instead of "abstract view" word.



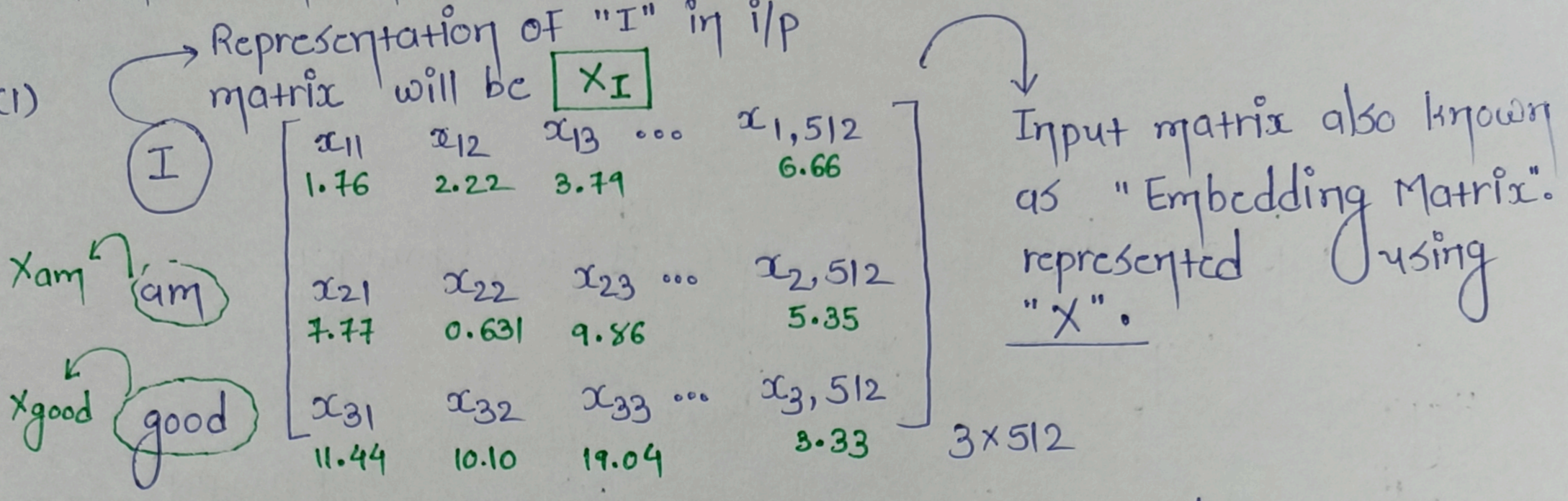
consider a scenario,

i.e. (2)



→ An attention given to each & every word of OTHER sentences known as "cross-attention."

e.g.(1) → Representation of "I" in i/p matrix will be X_I



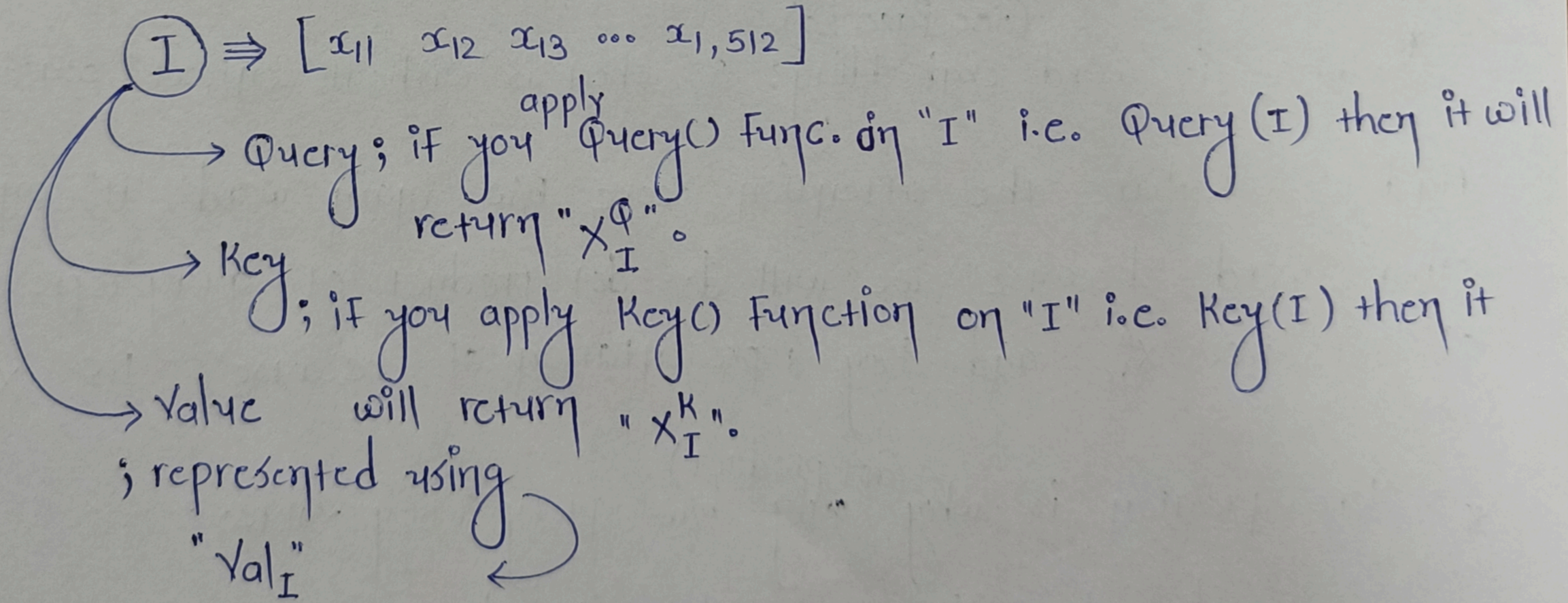
* → How to generate vector for given specific word?

and the dimension of that vector is (1,512)?

* → ~~is~~ are Word2Vec & glove is embedding methods? (YES) (see paper "Attention is all you need")

Each and every is being seen using 3 rules:

consider single word vector of I,



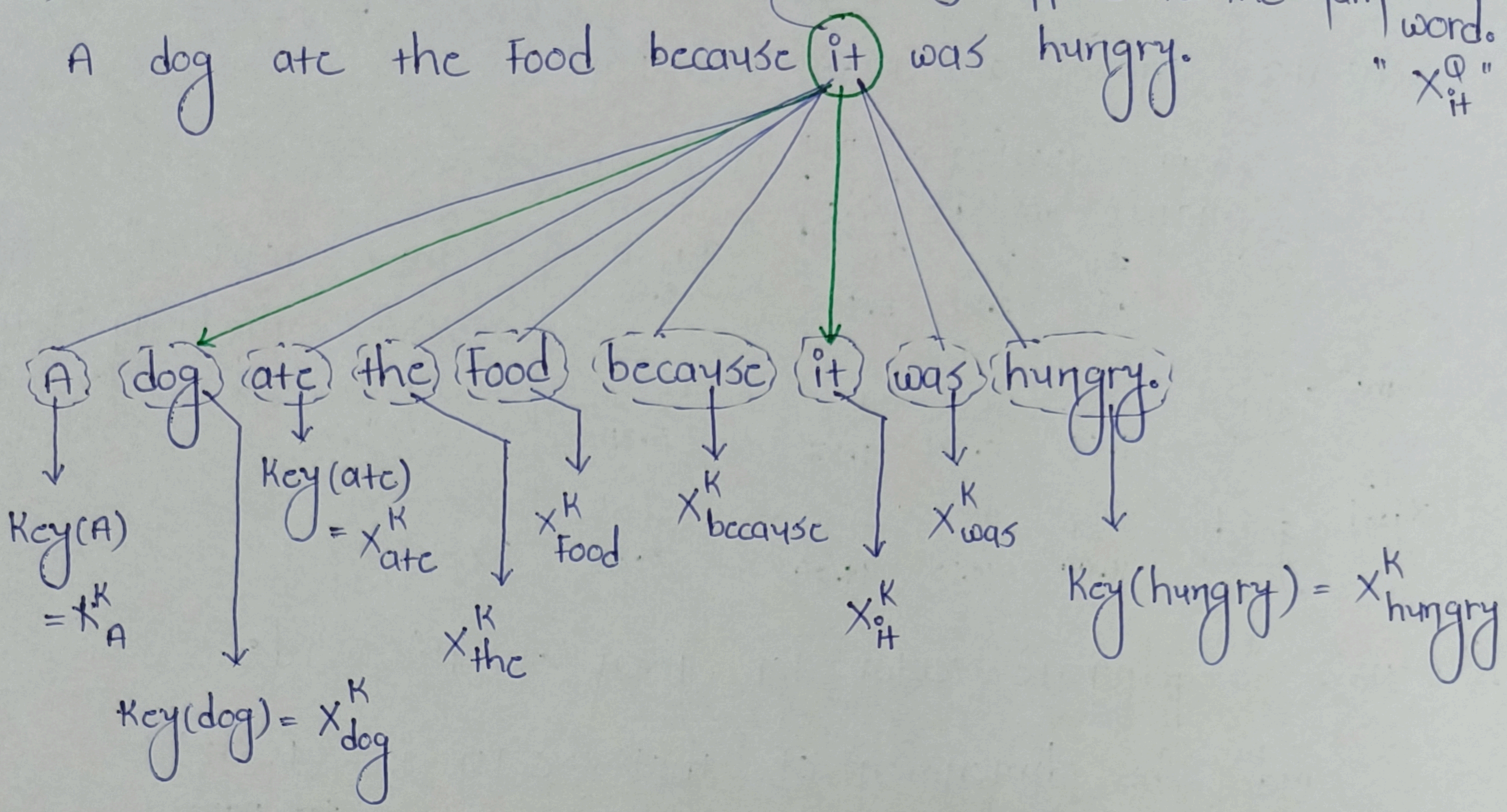
→ Additional:

An embedding is a vector representation of word or phrase. It is a way of representing text data in a way that is understandable by machines. Typically a fixed-length vector, such as 300 or 1024 dimensions.

Eg. (2)

A dog ate the food because it was hungry.

Query(it) will be applied for the main word. " X^{Q}_{it} "



Here in given example, it is trying to find attention value to each & every word within same sentence with respect to that main word, here it is **"it"**. For the main Query value will be found and Key value will be found for opposite compared words.

(see Page 20)

* → what is word embedding? & what is "without context" concept?

And to calculate the attention score b/w main word "it" and other word "because" we will do dot product of query representation of "it" i.e. X^{Q}_{it} and Key representation of "because" i.e. $X^{K}_{because}$.

(dot product)

* → Here question is why only b/w query & Key values?

and why only dot product based attention score?

(see paper "Attention is all you need")

→ e.g. (3)

consider a sentence "I am good"

representation will be,

$$x_1 \Leftarrow x_I = [x_{11} \; x_{12} \; x_{13} \cdots x_{1,512}]_{1\times512}$$

$$x_2 \Leftarrow x_{am} = [x_{21} \; x_{22} \; x_{23} \cdots x_{2,512}]_{1\times512}$$

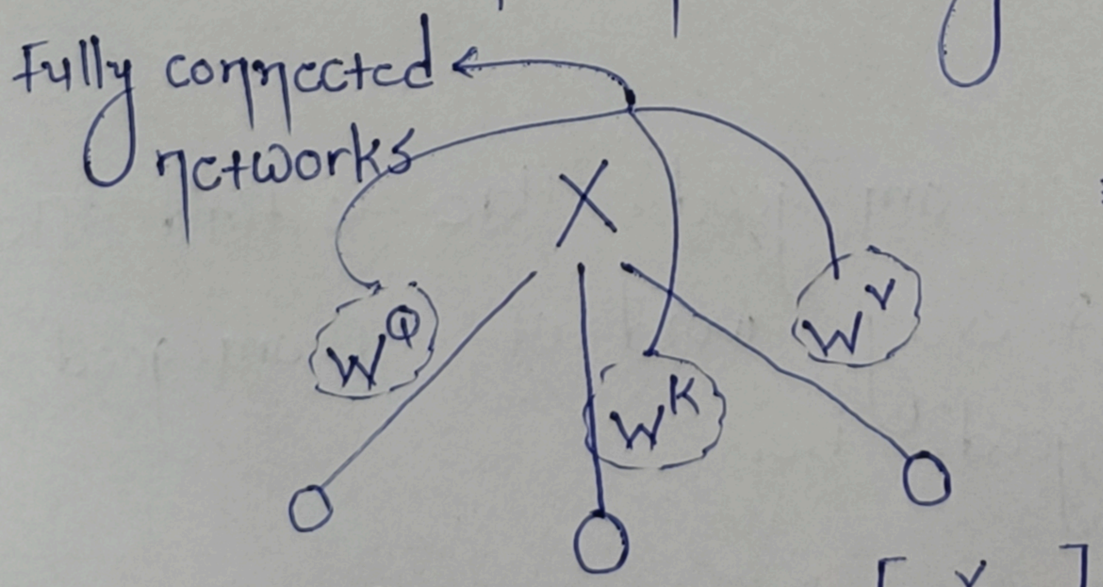
$$x_3 \Leftarrow x_{good} = [x_{31} \; x_{32} \; x_{33} \cdots x_{3,512}]_{1\times512}$$

→ $[x_1]_{512\times1}$, $[x_2]_{512\times1}$, and $[x_3]_{512\times1}$

column matrix of all will be input matrix,

- input matrix/vector $(X) = \begin{bmatrix} x_I \\ x_{am} \\ x_{good} \end{bmatrix} \sim \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

↓ For this input matrix three different neural networks (Fully connected networks) will be trained. One for query, one for key, and one for value. Those neural networks are represented using W^Q, W^K, and W^V accordingly.



This is similar for remaining i/p vectors (x_2 and x_3) and W^K and W^V, & their outputs (K_i & V_i); $i \in \{1,2,3\}$

$\Rightarrow [x_1]_{512\times1}$ from X (input matrix) is given to the W^Q, W^K, and W^V to get q_1, K_1, and V_1 accordingly. So if

dimension of x_1 is (512×1) and
" " q_1 is (64×1) then
" " W^Q will be (64×512)

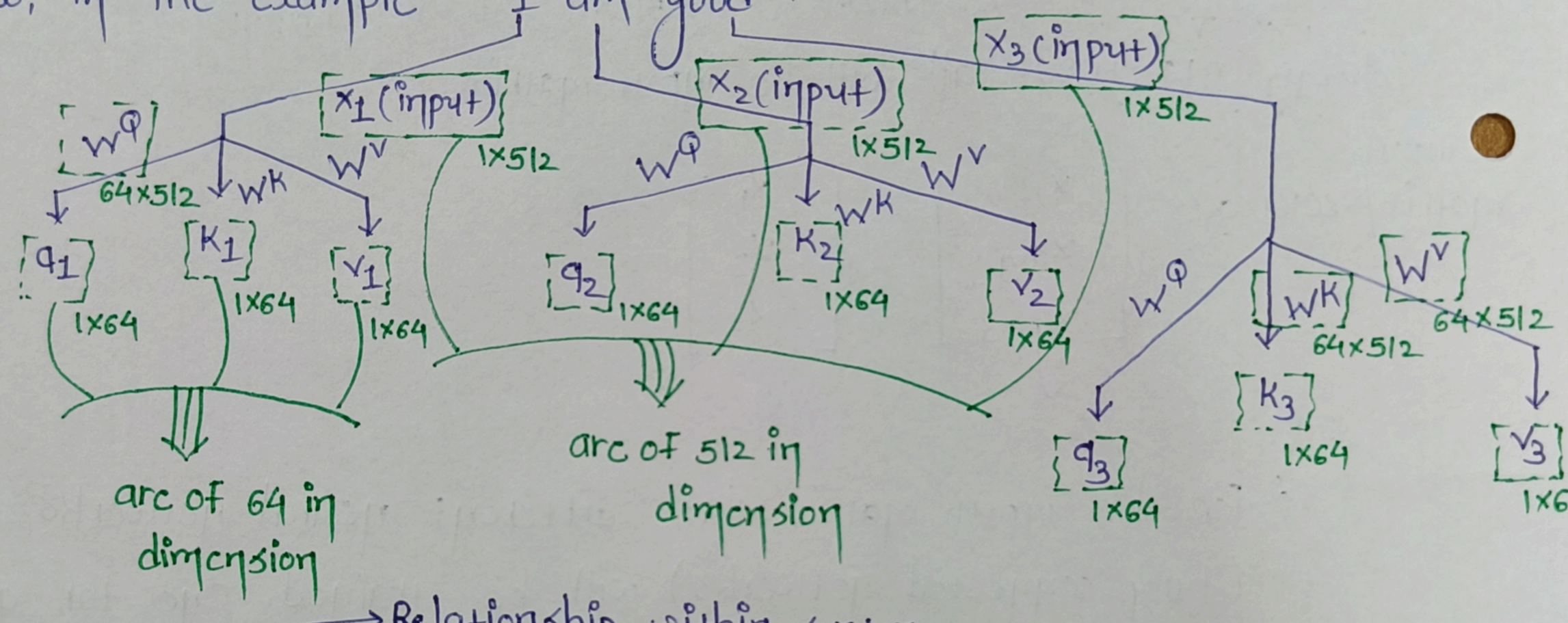
Because there is multiplication b/w W^Q and input vector x_1.

$$\therefore W^Q * x_1 = q_1$$

→ $\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$, $\begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix}$, and $\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix}$ are represented by capital letters Q, K, and V.

pronounciation of all is Q-matrix, K-matrix, and V-matrix respectively.

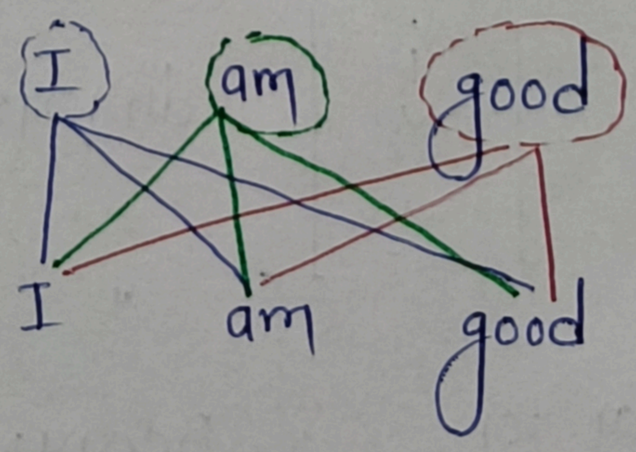
So, in the example "I am good".



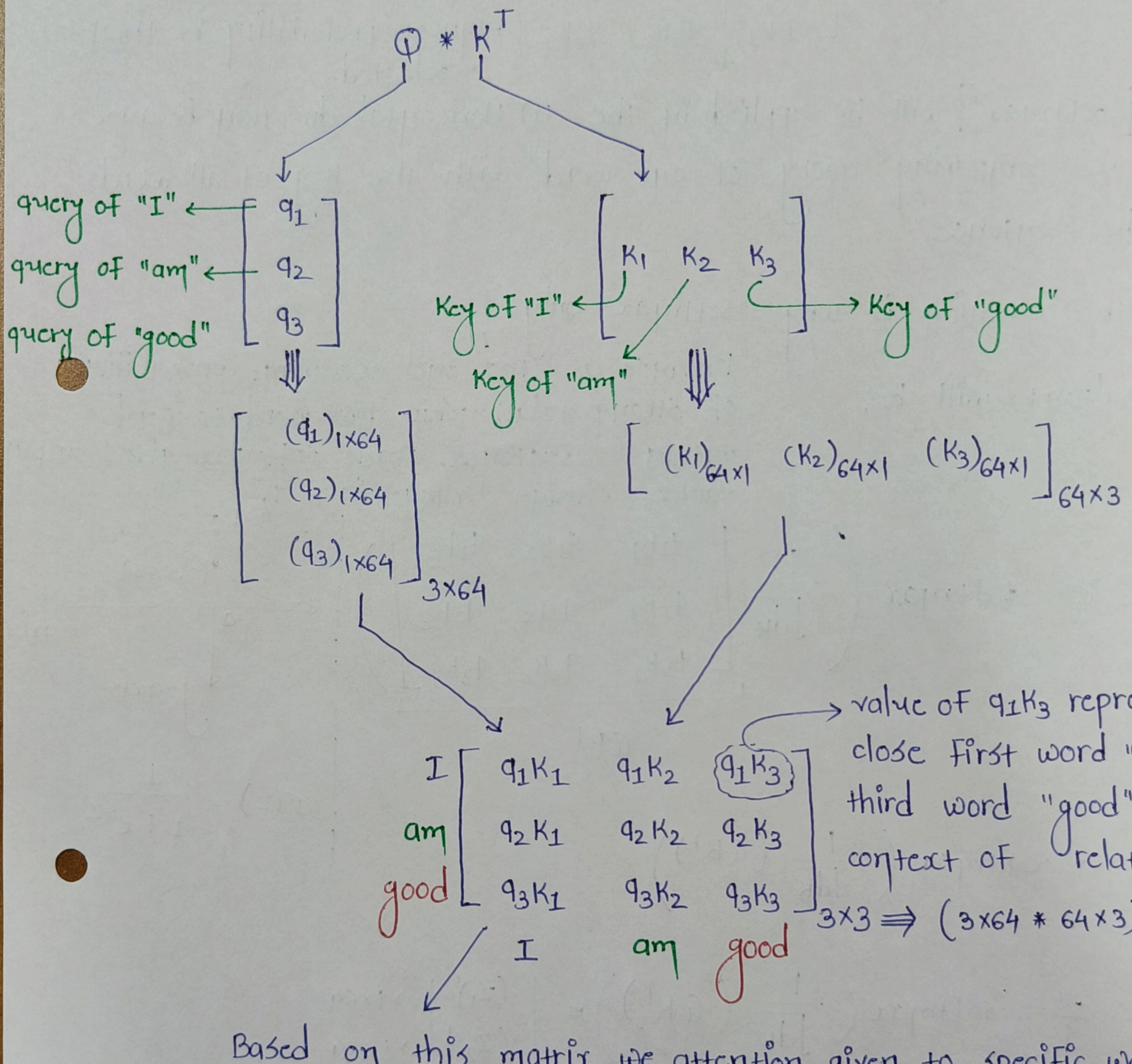
→ Relationship within sentence

→ Now, [self-attention] scores are calculated based on Q-matrix and K-matrix. That is why here Q-matrix and K-matrix are calculated.

now consider same example, "I am good". Here to find attention score / relation value of each & every word of "I am good" with the same sentence "I am good", i.e.



so, those attention scores will be found based on matrix generated by matrix multiplication of Q-matrix and transpose of K-matrix, i.e.



Based on this matrix, we attention given to specific word by specific word can be known. size of the matrix (resultant of QK^T) is based on no. of words particular sentence contains.

→ Now, it is important to normalize the matrix values. so for that resultant matrix of QK^T is multiplied by $\frac{1}{\sqrt{RMD}}$; RMD = Resultant Matrix Dimension, of particular query or key vector. i.e. 64

so matrix after normalization will be,

$$\frac{1}{\sqrt{RMD}}\begin{bmatrix} q_1k_1 & q_1k_2 & q_1k_3 \\ q_2k_1 & q_2k_2 & q_2k_3 \\ q_3k_1 & q_3k_2 & q_3k_3 \end{bmatrix}$$

(H) → These scores are normalized to produce a probability distribution over input tokens. Input token with highest probability is then selected.

(V) ↓

then "softmax" will be applied in the (H) Horizontal direction because we are comparing query of one word with the key of all words in the sentence.

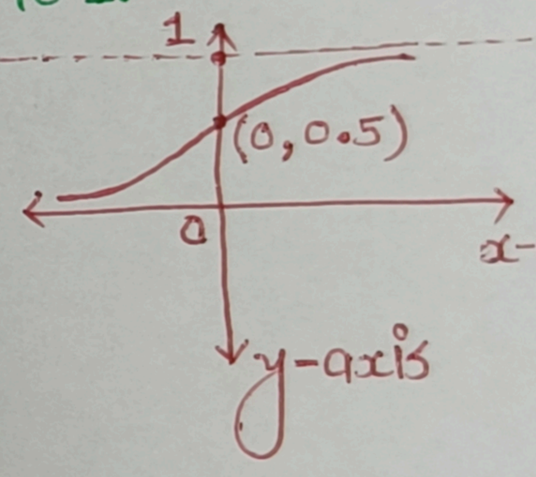
But the question is why "softmax" only?

Because at the end we want probability of strong relationship b/w word-to-word within sentence. And softmax gives output values within range "0 TO 1."

so, resultant will be,

$$\text{softmax}\left(\frac{1}{\sqrt{d_K}}\begin{bmatrix} q_1k_1 & q_1k_2 & q_1k_3 \\ q_2k_1 & q_2k_2 & q_2k_3 \\ q_3k_1 & q_3k_2 & q_3k_3 \end{bmatrix}\right)$$

QK^T



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sim \text{softmax}\left(\frac{1}{\sqrt{d_K}}(QK^T)\right)$$

$$z' = \text{softmax}\left(\frac{1}{\sqrt{d_K}}(QK^T)\right) * \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \begin{matrix} \to I \\ \to am \\ \to good \end{matrix}$$

$$z = z' * V$$

$$= \begin{bmatrix} (z_1)' \\ (z_2)' \\ (z_3)' \end{bmatrix} \begin{matrix} \to 1\times 64 \\ \to 1\times 64 \\ \to 1\times 64 \end{matrix}$$

3×64

output matrix is of "$n \times 64$" in dimension, here n is total no. of words in sentence. Here V represents value-matrix, typically size of value-matrix ~~is~~ equals to the key-matrix.

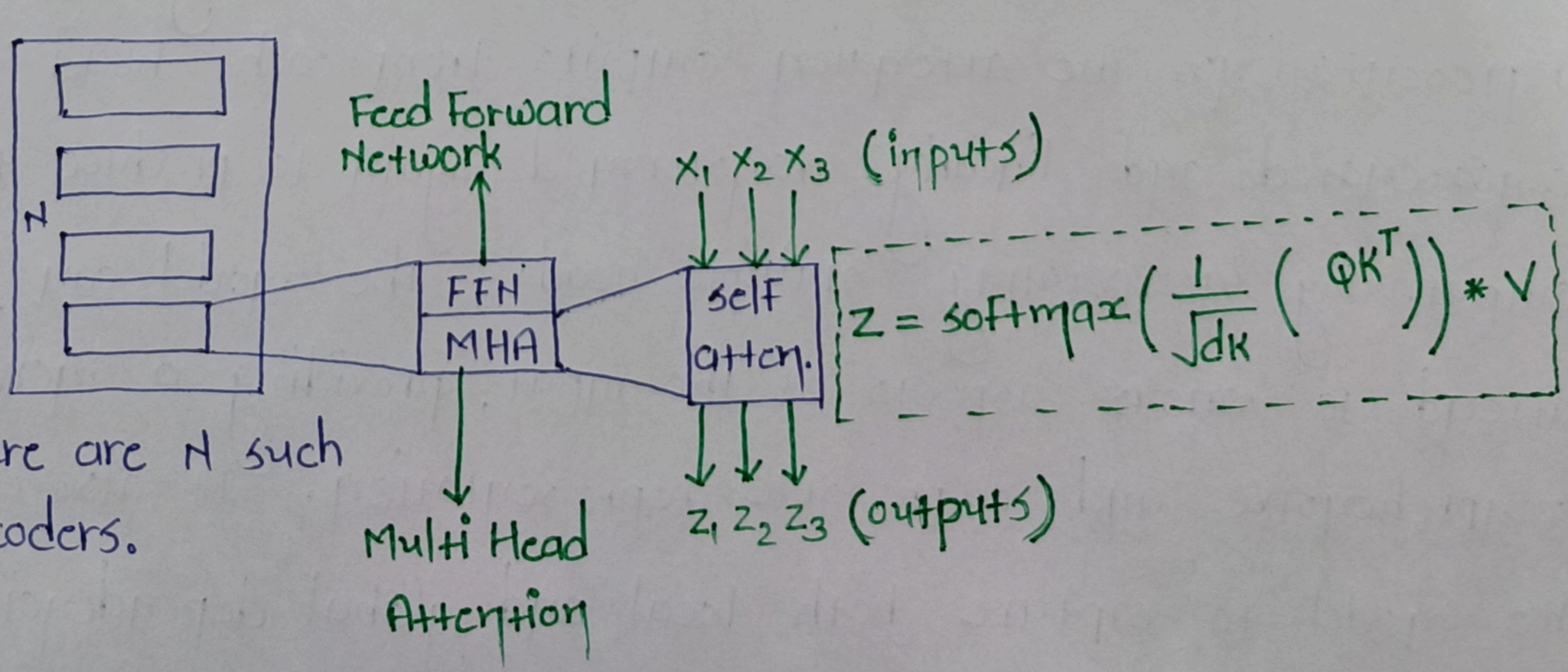
→ Additional:

The input and output of both functions (sigmoid & softmax) are slightly different since sigmoid receives just one input and only outputs a single number that represents the probability of belonging to class 1 (remember that we only have 2 classes so the probability of belonging to class 2 = 1 - P(class 1)). while on the other hand softmax is vectorized, meaning that takes a vector with the same number of entries as classes we have and outputs another vector where each component represents the probability of belonging to that class.

sigmoid	softmax
$out = P(Y = class\ 1 \mid x)$	$out = \begin{bmatrix} P(Y = class\ 1 \mid x) \\ P(Y = class\ 2 \mid x) \\ \vdots \\ P(Y = class\ K \mid x) \end{bmatrix}$

; x = input $\in (-\infty, +\infty)$

→ Now, summarizing encoder part with an example "I am good".



→ Analogy:

these tokens' query are/is given to you.

Query: How closely somebody related to me?

other token — specific token — you need to use your query value

Key: How closely, I am related to someone?

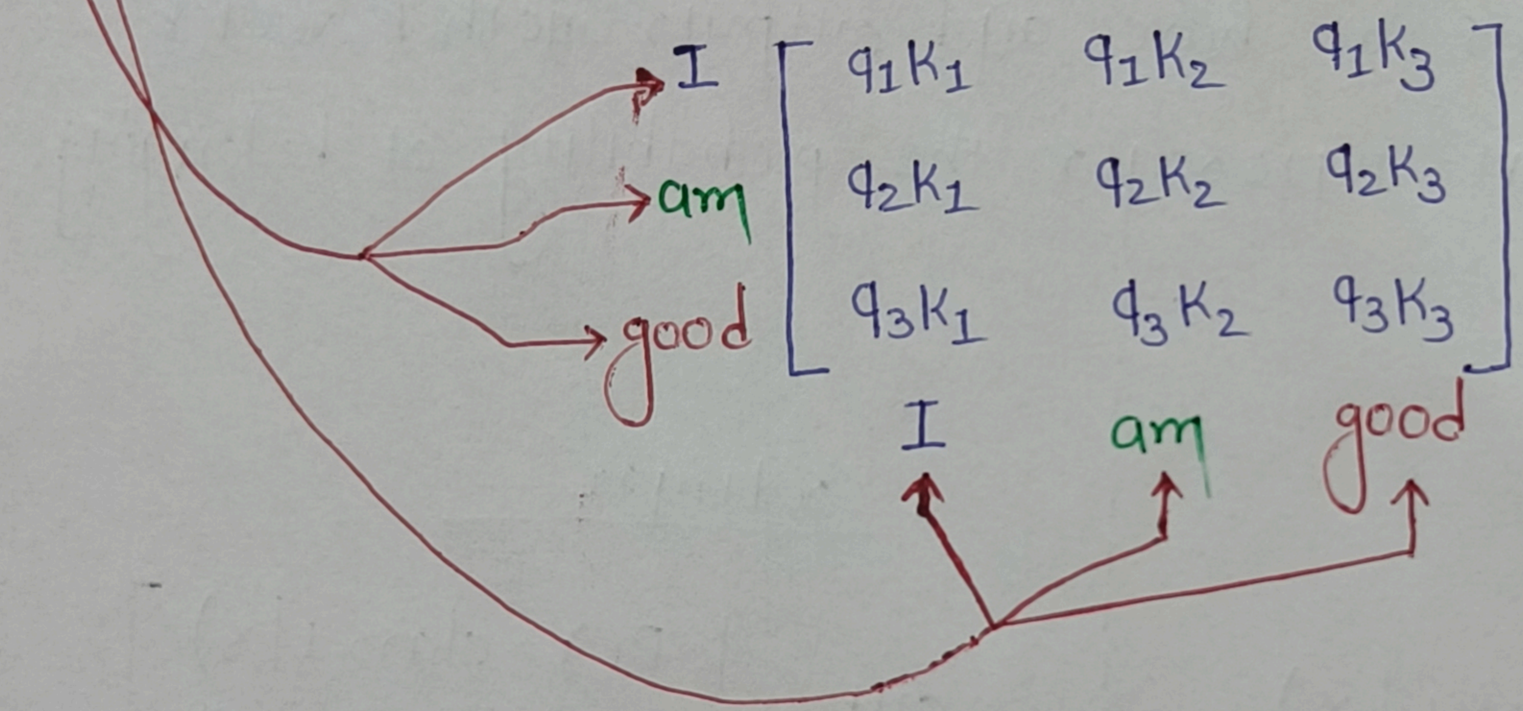
you need to use your key value — specific token — other token — these token's query is given to you.

and the at the end your value will be weighted by

V -matrix

self-attention score.

$$\text{softmax}\left(\frac{1}{\sqrt{d_K}}(QK^T)\right)$$

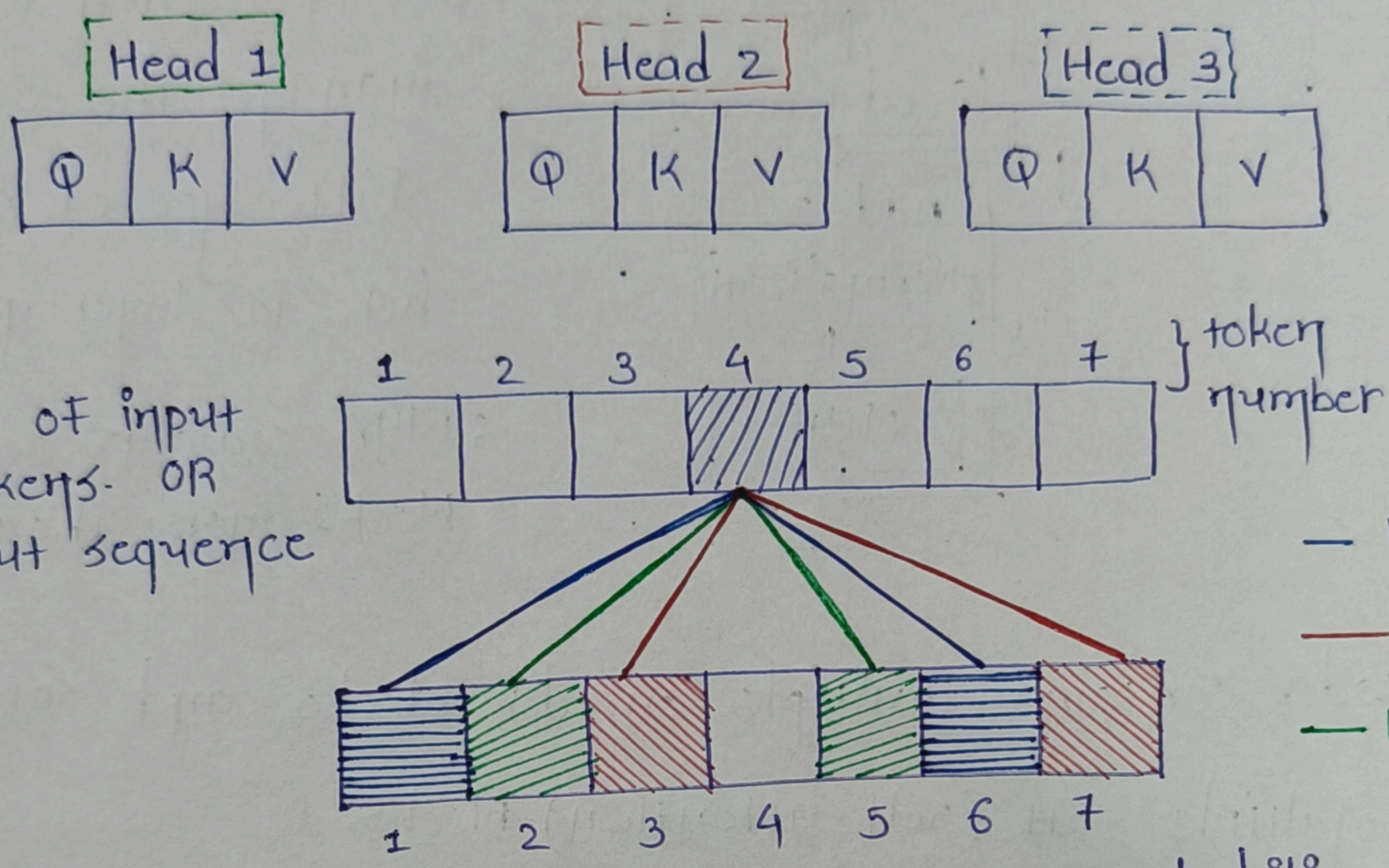


→ Multi-head mechanism:

Typically, multi-head is used to include various aspects or contexts of data in the training From the input sequences. In the multi-head mechanism, each head learns its own set of query, key, and value representations. The attention outputs From each head are then concatenated and linearly transformed again to produce the Final output. By incorporating multiple heads, the model can jointly attend to various aspects of the input, providing a more comprehensive and expressive representation. It also enables the model to capture both local and global dependencies, as different heads can attend to different ranges of the input sequence.

All the attention heads in the multi-head mechanism of transformer are working simultaneously to capture multiple perspectives and contexts from the input sequence.

graphical representation:



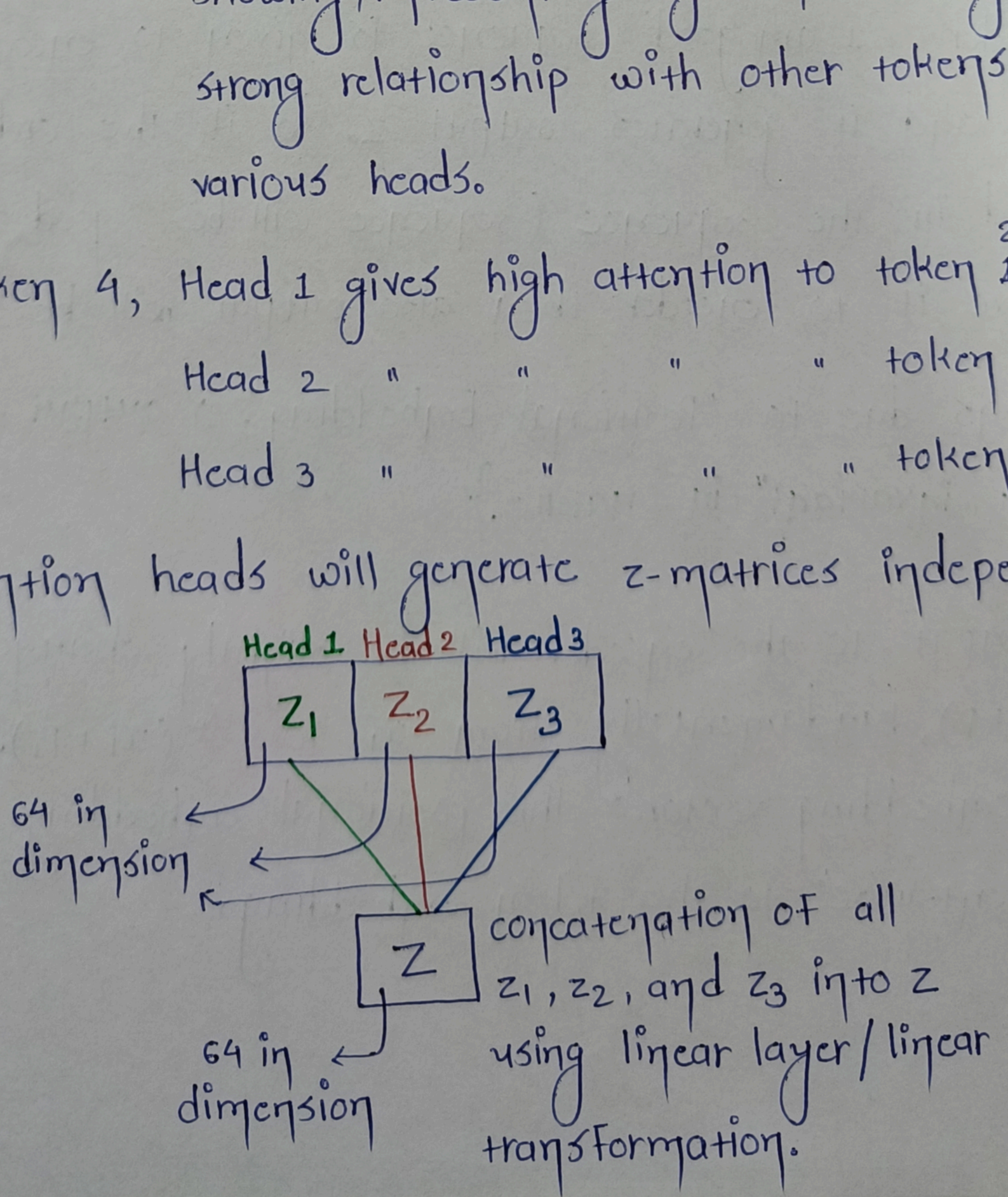
showing/representing highest probability of strong relationship with other tokens by various heads.

For token 4, Head 1 gives high attention to token 2 & 5.

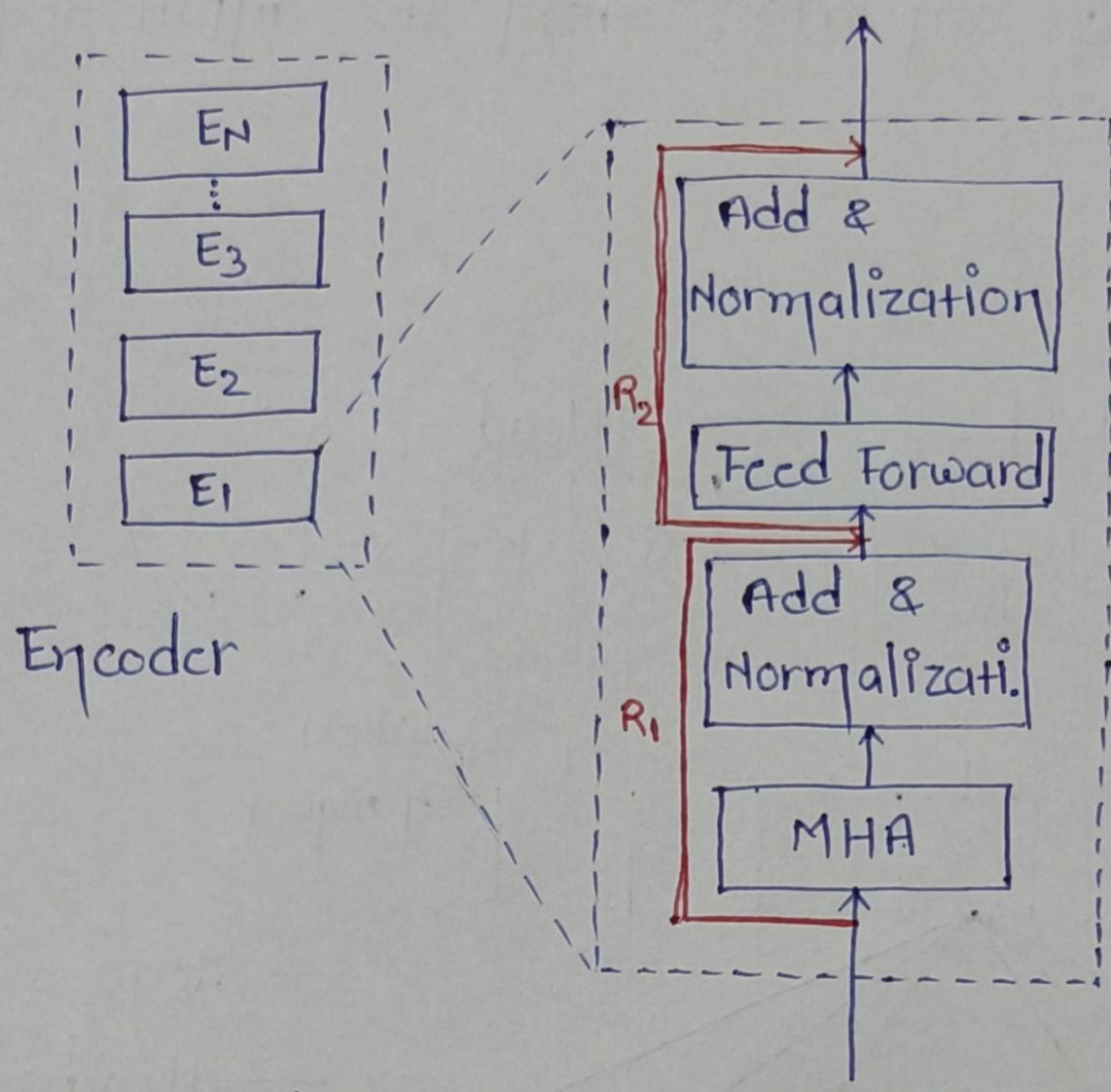
Head 2 " " " " token 3 & 7.

Head 3 " " " " token 1 & 6.

All attention heads will generate z-matrices independently.



→ At last, drawing top-level diagram of single encoder block.



; R_1 and R_2 are residual connections.

; Normalization is used to manage the covariate shift. Type of normalization here is layer normalization

; MHA contains self-attention blocks (more than one.).

single encoder contains multiple encoder blocks and single MHA contains multiple SA (self-attention) blocks.

→ Now, consider "I am good" sequence is given to encoder block as an input and it generates output z. so if the order of tokens is changed in the sequence (sequence will be "good am I") and again give it to encoder then also output will be same i.e. z

This happens due to "Positional Embedding". {output of encoder is "position invariant" in nature.}

Now, the input to the encoder will be slightly changed from X TO $(X+P)$, fundamentally From x_i TO $(x_i + P_i)$. Here P_i is positional embedding vector. This positional embedding vector is added element-wise to the token embeddings.

Recurrent Neural Network have an inbuilt mechanism that deals with the order of sequences. The transformer model, does not use recurrence or convolution and treats each data point as independent of the other. Hence, positional information is added to the model explicitly to retain the information regarding the order of words in a sen-tence. "Positional Encoding" is the scheme through which the knowledge of the order of objects in a sequence is maintained.

* → It can be learnable/trainable parameter. But this is not a case for all time.
  → is token embedding also learnable parameter?

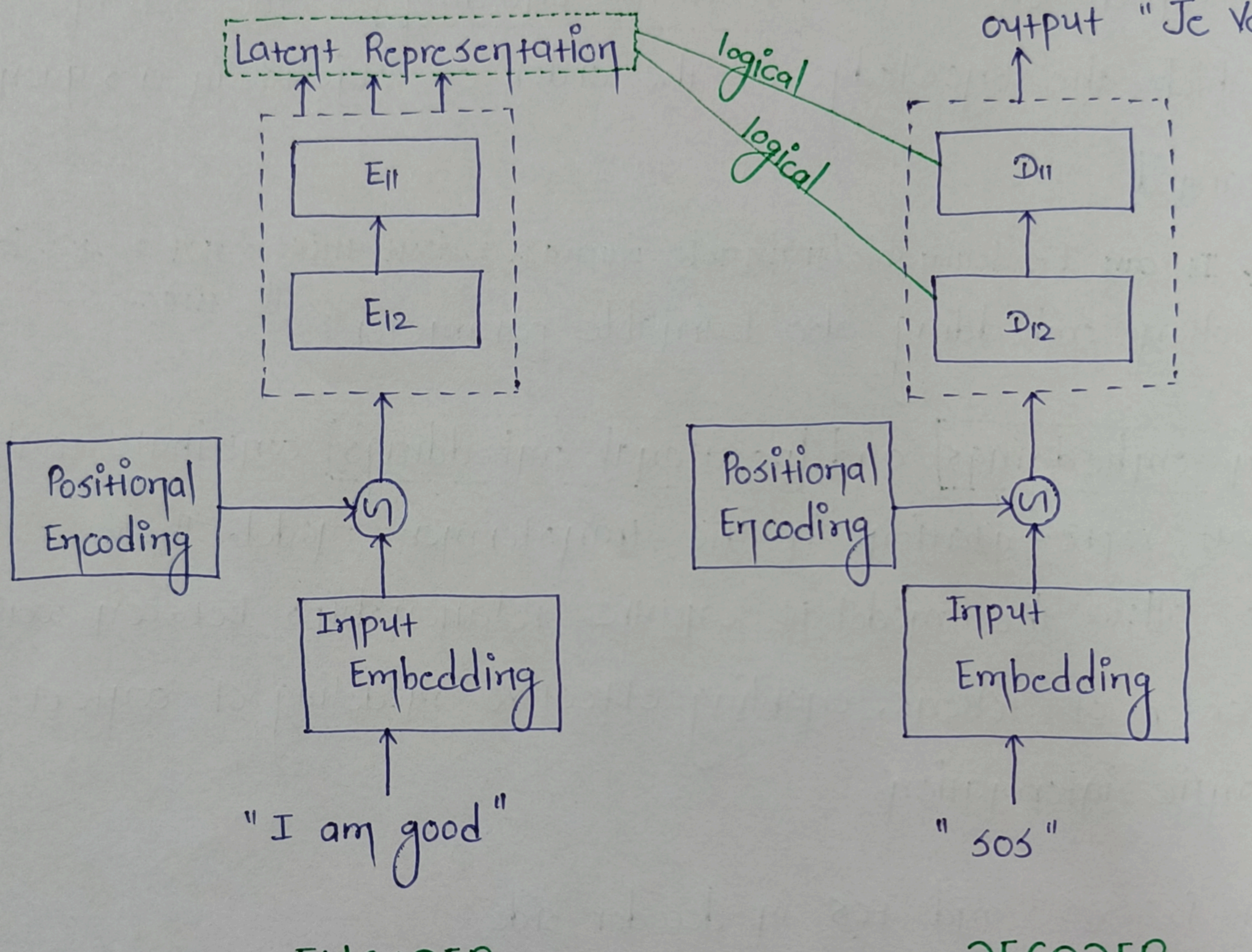
Both token embeddings and positional embeddings contribute to the overall input representation in the transformer model. These embeddings allow the model to capture relationships between words and the order of tokens, enabling effective modeling of context and semantic information.

* → what is SOS and EOS in decoder side?
  - SOS → Start of Sequence
    - it is a special token or symbol used in sequence-to-sequence models (RNNs or transformer) to indicate the beginning of sequence. The "SOS" token is prepended to the input sequence when generating the output, serving as an indicator for the decoder to start generating the target sequence.
  - EOS → End of Sequence
    - When decoder encounters the "SOS" token, it initializes its internal states and begins generating the output tokens one by one until it reaches on EOS token or a predefined maximum length.

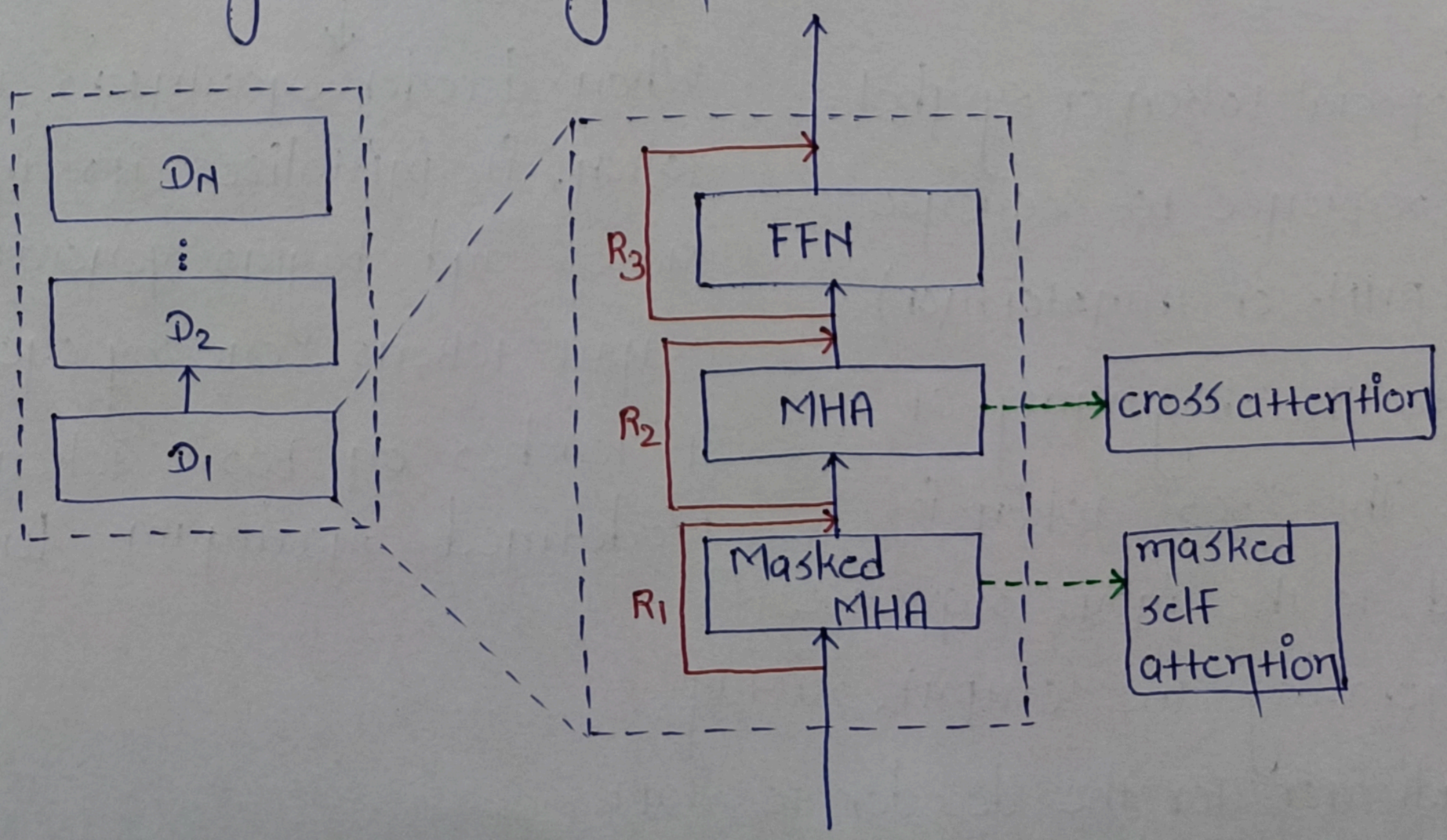
→ Latent representation:

↳ it refers to the intermediate representation of input data at a particular layer within the model.

→ Top-level diagram of transformer:

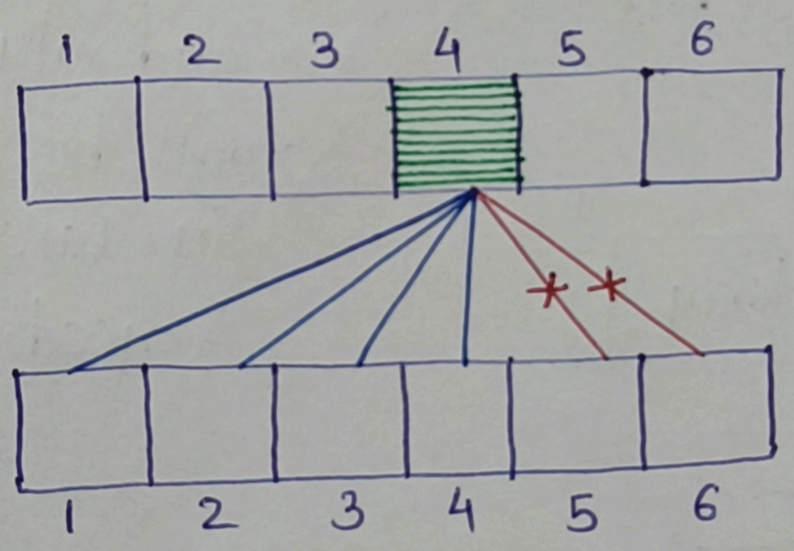


→ Decoder single block diagram:



→ Here in the decoder side output tokens of sequence will not be generated parallel. But encoder side "I am good" sequence given as an input and its all token processed parallel. French translation of "I am good" is "Je vais bien", so decoder side First "Je" token will be generated, then it will feed as an input to decoder again, and decoder will predict next word i.e. "vais". Again "vais" is will feed as an input to decoder and it will predict next word i.e. "bien". Here based on next word prediction capability of decoder, its performance will measure. To achieve this type of mechanism "MASKED SELF ATTENTION" is used.

consider input sequence with 6 tokens



; — attention score can be calculated

; — attention score can not be calculated

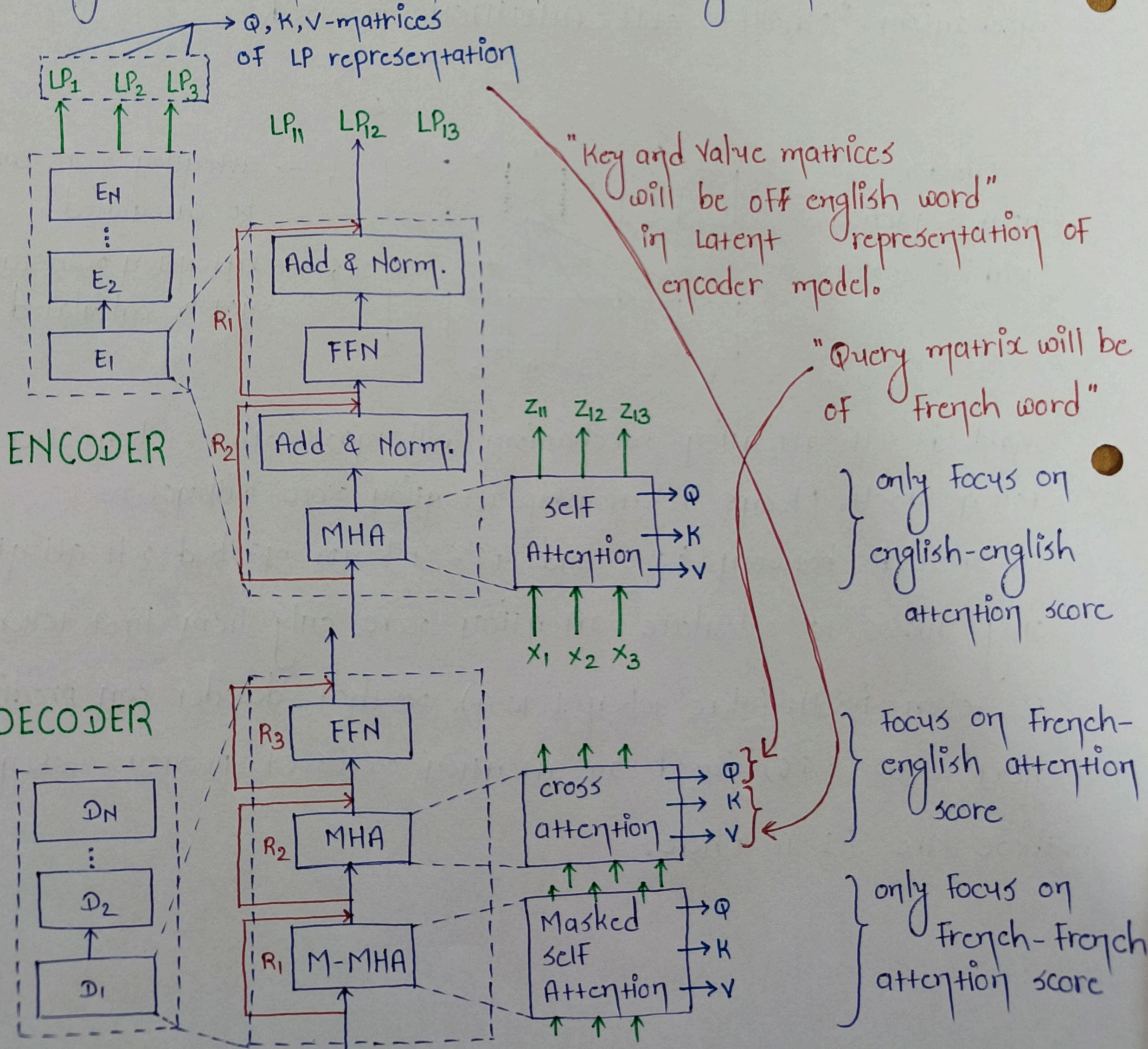
masked self attention mechanism will mask other tokens after token itself. Means relationship/attention score will not be calculated between 4 & 5 and 4 & 6 tokens, masked self attention only allows to calculate attention score only from First token to token itself. (Here token 1 TO 4). so that decoder can predict next words (5 & 6) based an attention scores of previous tokens those are {1 TO 4 here.

→ cross attention:

Here there is an example ["I am good".] so whenever translation will occur from english to French i.e. ["Je Vais Bien"] then in the cross attention there is a relationship between French word query and english word key at decoder side.

↳ and value also

cross-attention score is calculated in same way as self-attention score is calculated. But the difference is only with the Q-matrix and V-matrix. Let's consider an example of english to French translation. (specifically consider at decoder side.)



in the cross-attention, it tries to learn relationship between French query and english key and value matrices.

In the cross-attention French TO english mapping is there bat decoder side because cross-attention tries to learn that howmuch I am closed to given english representation i.e. latent representation of encoder.

→ cross-attention also known as encoder-decoder attention.

→ Final output of decoder of input "I am good" to encoder will be "Je Vais Bien".

- input "I am good" is ~~als~~ represented by X. so, as seen earlier X will be,

$$X = \begin{bmatrix} X_I \\ X_{am} \\ X_{good} \end{bmatrix}$$; $X_i \in \{I, am, good\}$ is word embedding of dimensions 512.

Let's consider output "Je Vais Bien" represented by Y. so Y will be,

$$Y = \begin{bmatrix} Y_{Je} \\ Y_{vais} \\ Y_{bien} \end{bmatrix}$$; $Y_j \in \{Je, vais, bien\}$ is also word embedding of dimensions 512.

Question & Answer

(1) what is word embedding?

word embedding refers to a technique used to represent words in a continuous vector space. Main idea behind word embedding is to capture the semantic and syntactic relationships b/w words based on their context in a given corpus. Word2Vec, GloVe, and FastText all are algorithms for word embedding.

(2) what is token embedding?

token embedding is a broader term that encompasses the representation of any individual token in a sequence of text. A token can be word, subword, character or any other unit that the model operates on. Token embedding captures the meaning and context of a token within a specific sequence.

Additional:
Word embedding is a specific type of token embedding that focuses on representing individual words.

(3) what is truncated backpropogation through time (TBPTT)?

It is a technique used in recurrent neural networks (RNNs) to address the vanishing or exploding gradient problem during training. In standard backpropogation, gradients are computed and propogated through the entire sequence of inputs and hidden states, which can be computationally expensive and can lead to the vanishing or exploding gradient problem. To mitigate this issues, truncated

backpropogation through time breaks the sequence into smaller subsequences or chunks. Instead of propogating gradients through the entire sequence, TBPTT limits the no. of time steps over which gradients are computed. The sequence is divided into chunks of a fixed length, and gradients are computed and updated within each chunk. The hidden state at the end of each chunk is then used as the initial state for the next chunk. By, truncating the backpropogation through time, TBPTT reduces the computational burden and addresses the vanishing or exploding gradient problem. However, it also introduces a limitation in capturing long-term dependencies that span across multiple chunks. The length of the chunks is typically chosen based on a trade-off between computational efficiency and model's ability to capture long-term dependencies.

(4) what is vanishing gradient and exploding gradient problem?

The vanishing gradient problem occurs when the gradients shrink exponentially over time, making it difficult for the model to learn long-term dependencies. The exploding gradient problem occurs when the gradients grow exponentially, leading to instability during training.

(5) what is gradient clipping?

It is a commonly used technique to overcome exploding gradient problem in neural networks. Gradient clipping involves capping the magnitude of gradients during training to prevent them from becoming too large. Two commonly used approaches are;

(a) L2-Norm clipping:

```
if ||grad|| > threshold:
    grad = (threshold / ||grad||) * grad
```

; where ||grad|| represents the L2-Norm of gradient vector.

→ consider a vector, $x = [x_1, x_2, \ldots, x_n]$

Also known as Euclidean Norm → L2-Norm is calculated as:

$$\|x\|_2 = \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}$$

AND

L1-Norm is calculated as:

$$\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$$

(b) value clipping:

Individual elements of the gradient vector are checked against a predefined threshold. if any element exceeds the threshold, it is set to the threshold value. This method is less commonly used compared to L2-Norm clipping.

grad = $[x_1, x_2, x_3, x_4, x_5]$, threshold = y

$x_1 > y$ $x_2 > y$ $x_3 > y$ $x_4 > y$ $x_5 > y$