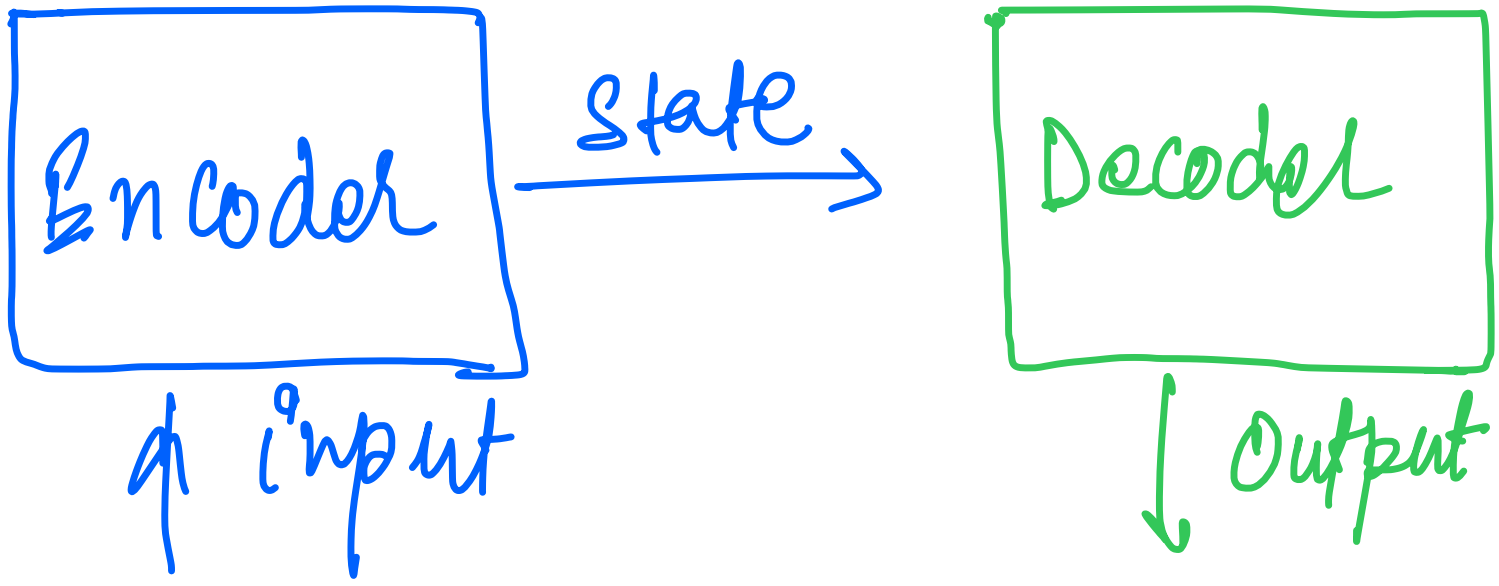
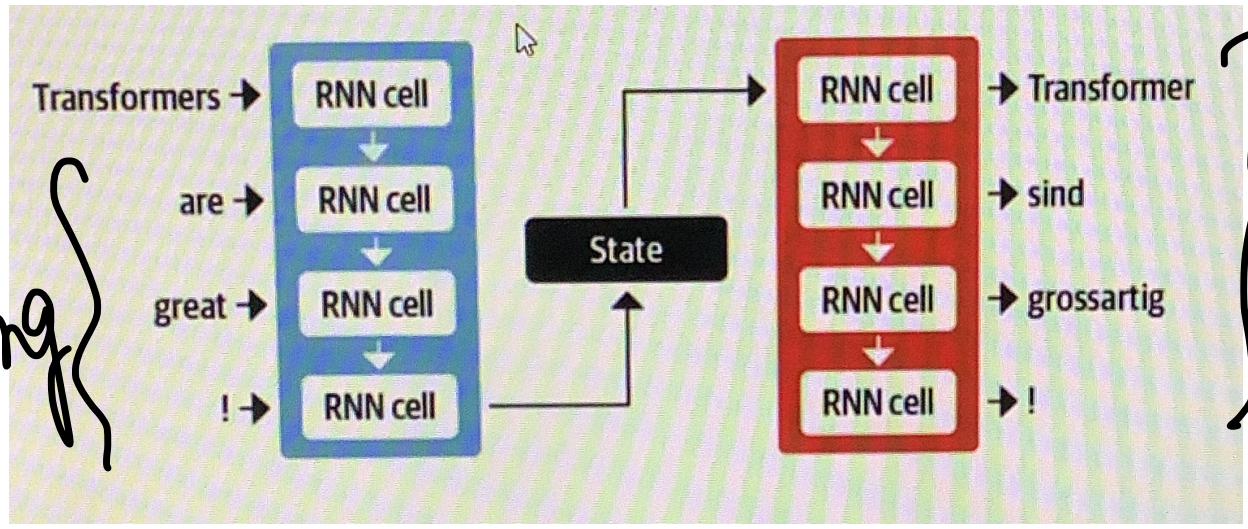
Transformers

Attention Mechanism

Encoder-Decoder-Architecture





Eng

German

RNN Cell is LSTM in

↑ Encoder

↓ Decoder

paper: Seq to Seq learning with Neural Networks.

Problem with this architecture is it kind of start missing initial words. for long sentences. (30 words)
which leads to "Attention Mechanism".

SELF Attention

NLP → WORDS → Numbers (Vectorization)

Methods : One Hot Encoding (OHE):

Mat | Cat | Rat

	mat	cat	rat	
mat →	1	0	0	<Vector><mat>
cat →	0	1	0	<Vector cat>
rat →	0	0	1	<Vector Rat>

TF-IDF

Word2Vec

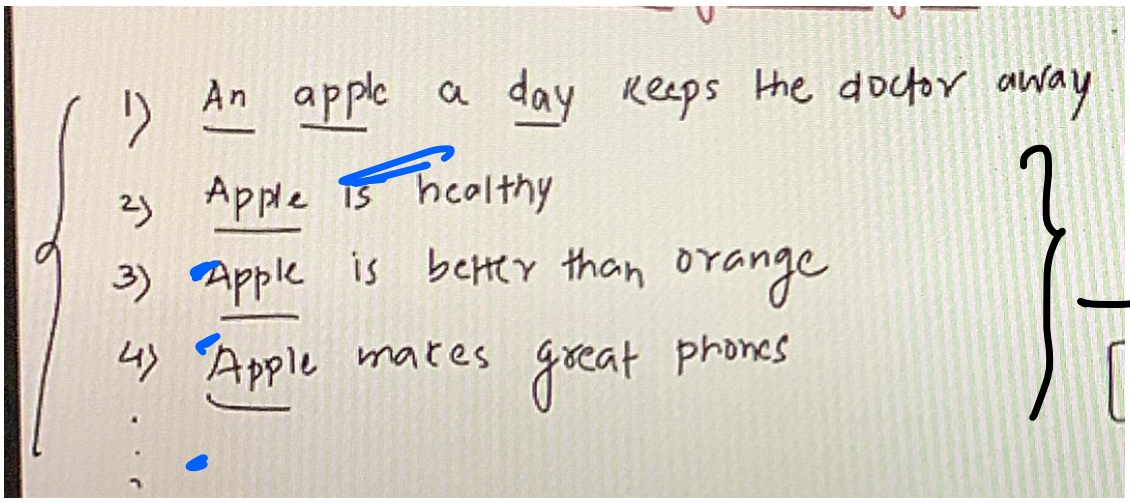
Word Embeddings

King → [0.6, 0.1, 1.0, 0.9]

Queen → [0.3, 0.2, 0.9, 1, 0]

Problem with Word Embedding

Word Embedding captures "Average Meaning" of a word. Depending on the Data set. Ex:



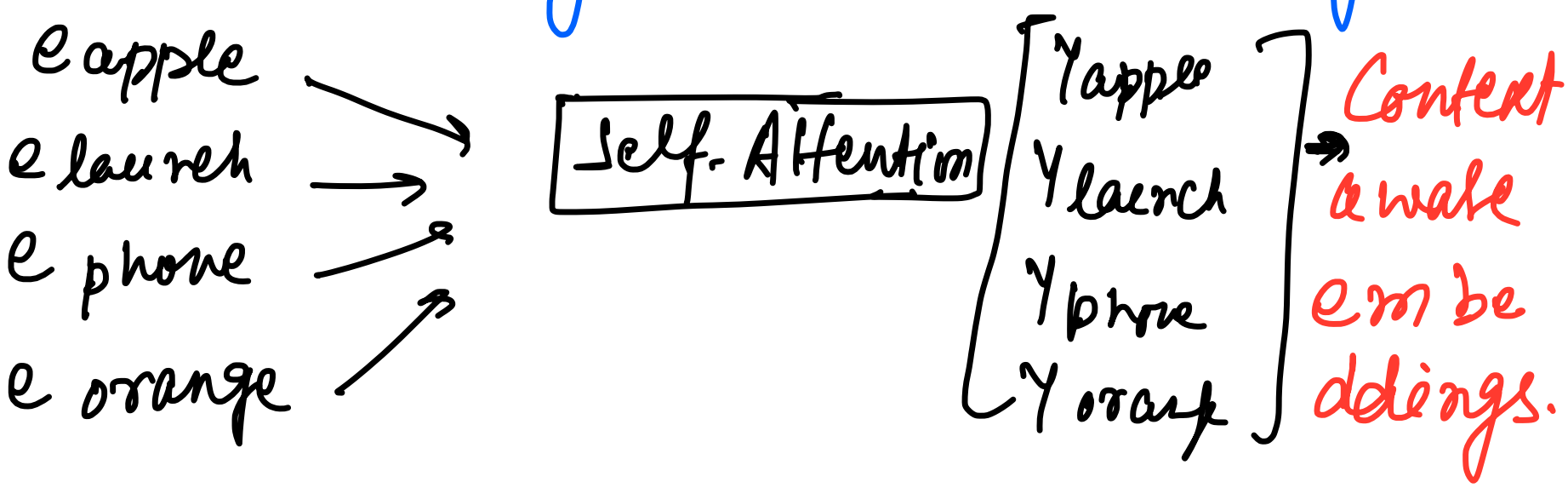
[taste, technology]

→ [0.6, 0.1]

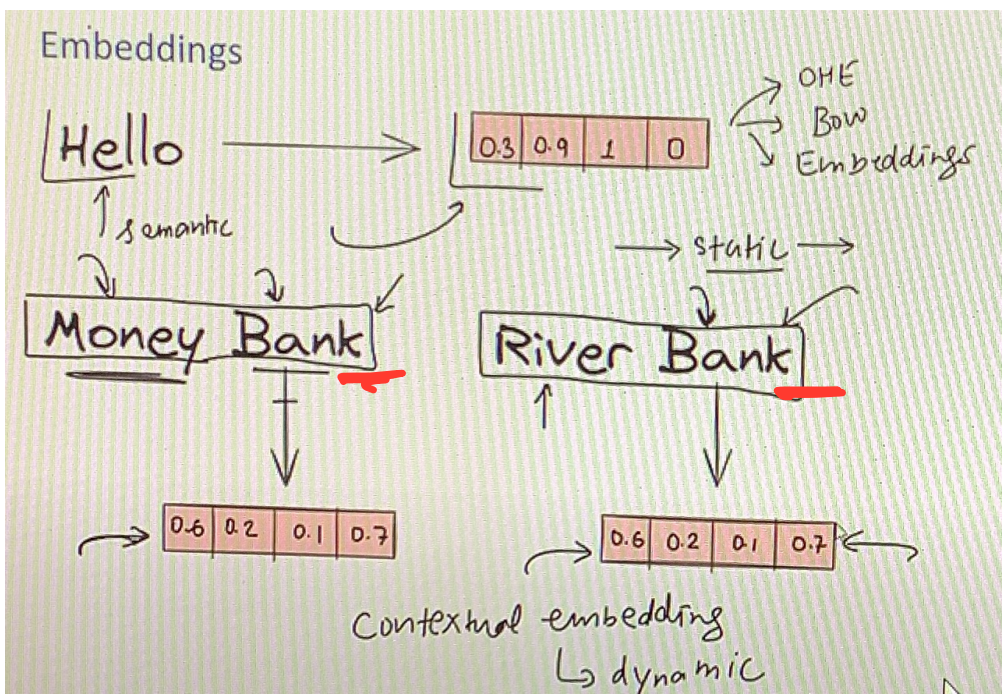
Let say we 10K sentences & 9K as fruit & 1K as Technology. Hence Vector will be something like [0.9, 0.1] tilted towards food (Average weighted), Hence word embeddings let depend upon Data set. & embeddings are STATIC.

But we need contextual embeddings which can generate embedding for the context. Ex: Apple as Tech Apple as fruit both should have different embeddings.

* Self-Attention helps us to achieve Dynamic Embeddings.

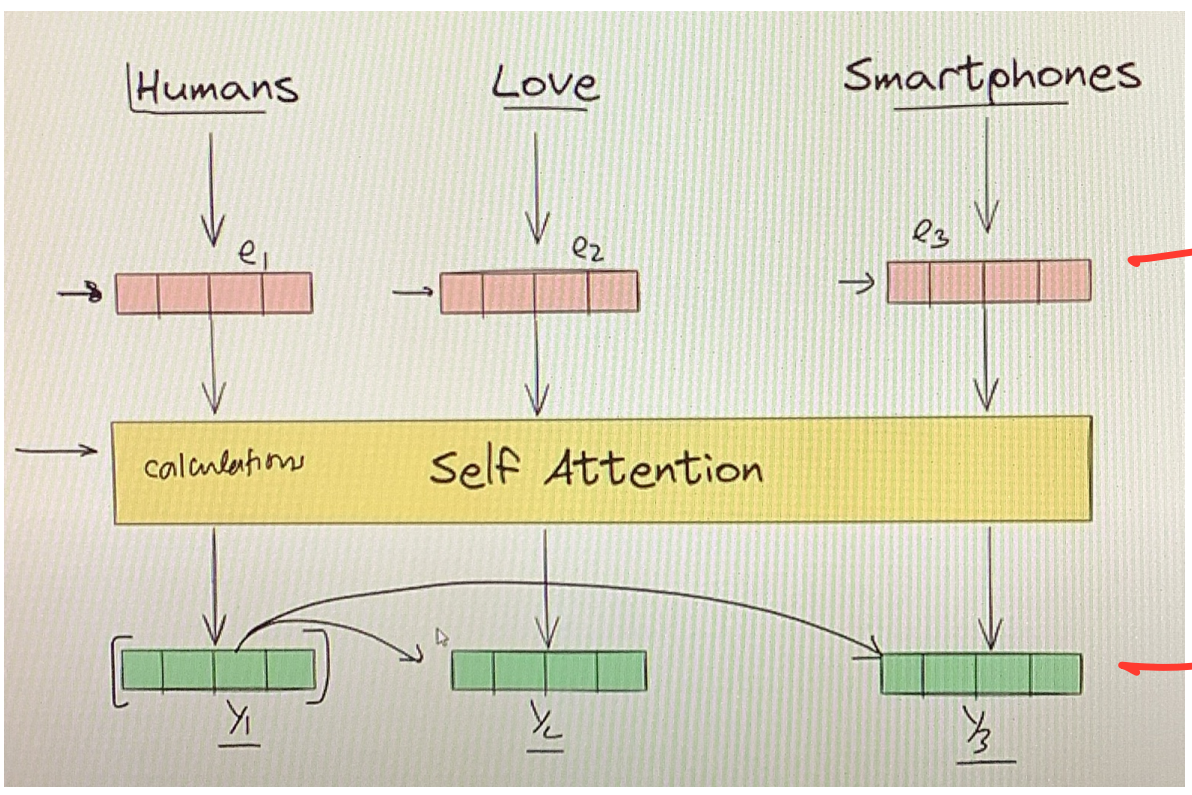


Static Embeddings :



Word "Bank"

has very different meaning in both context but embe - ddings is same.



→ Static Embeddings (Word2Vec ---)

→ Context Aware Embe - dding.

Context Embedding understand the context of word (ex: Apple as a fruit or apple as Tech). Now we will understand "Self-Attention"

money bank grow → (SBS, HDFC etc..)

river bank flows → (River Bank)

Will think "bank" is a combination (Money, Bank, Grow)

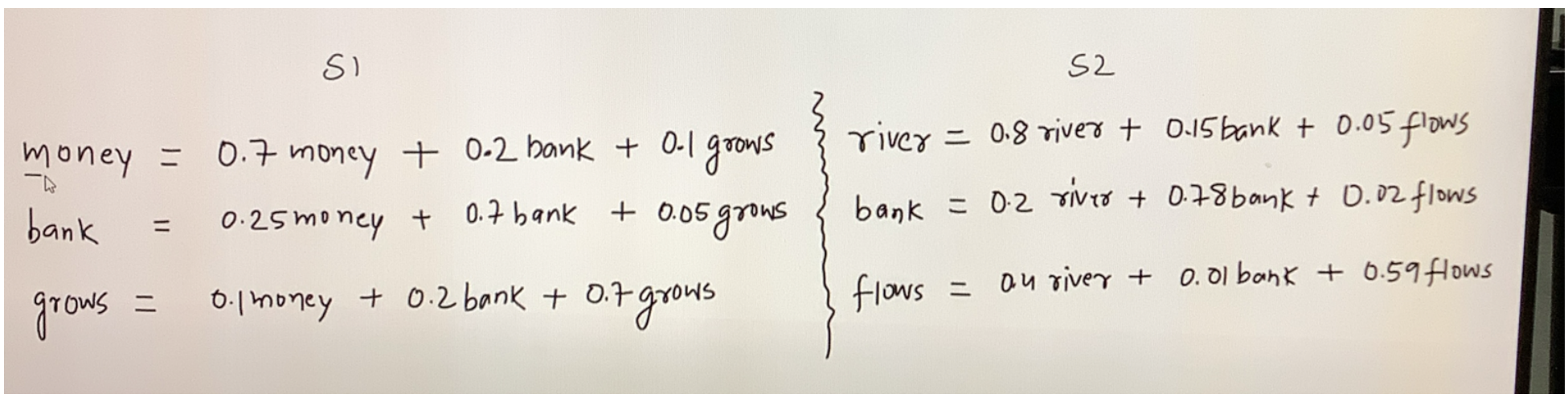
money bank grows

bank = 0.2 * Money + 0.7 Bank + 0.1 grows

river bank flows (river, bank, flows)

bank = 0.5 river + 0.4 bank + 0.1 flows

Now bank has become context dependent. Below is the detail for both sentences. Each sentence is



presented as combination of other words in that sentences.

Now will replace words with their embeddings. EX:-

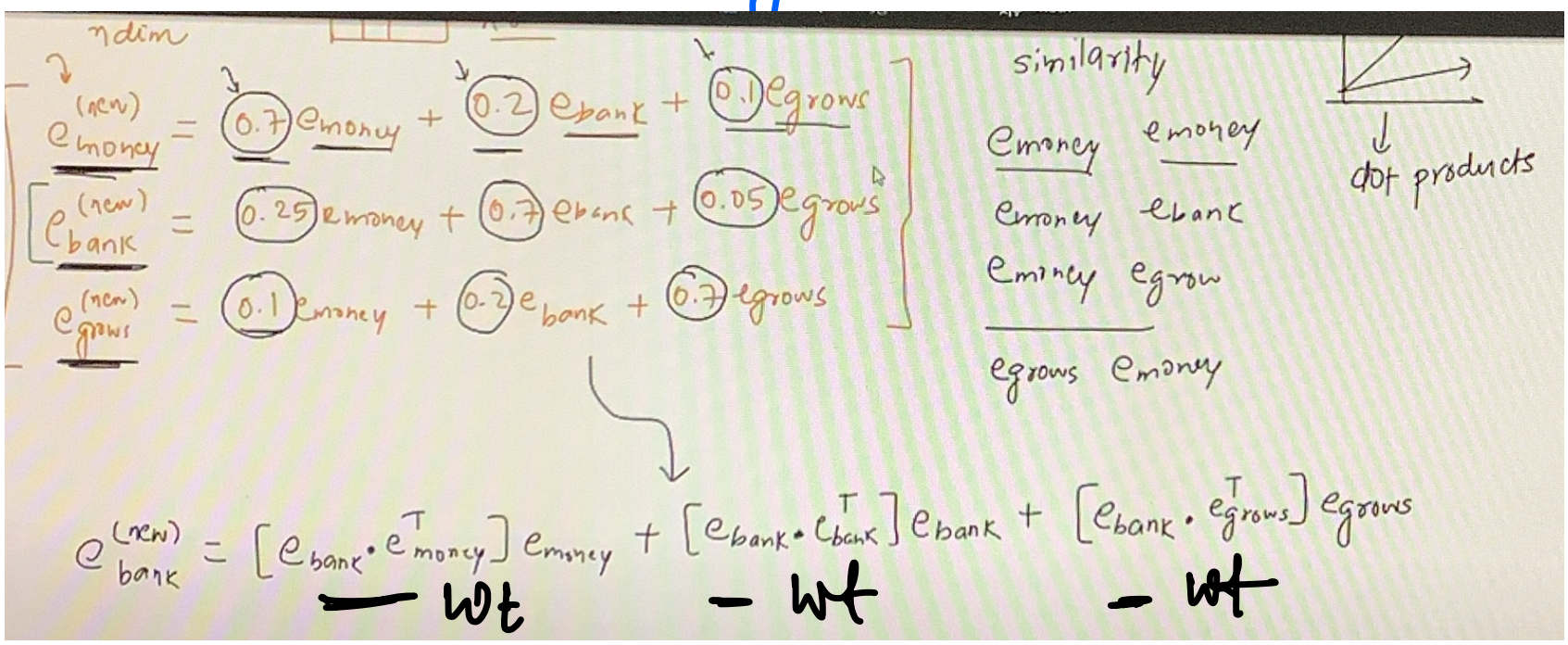
$$
\begin{aligned}
e_{money}^{(new)} &= 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows} \\
e_{bank}^{(new)} &= 0.25\, e_{money} + 0.7\, e_{bank} + 0.05\, e_{grows} \\
e_{grows}^{(new)} &= 0.1\, e_{money} + 0.2\, e_{bank} + 0.7\, e_{grows}
\end{aligned}
$$

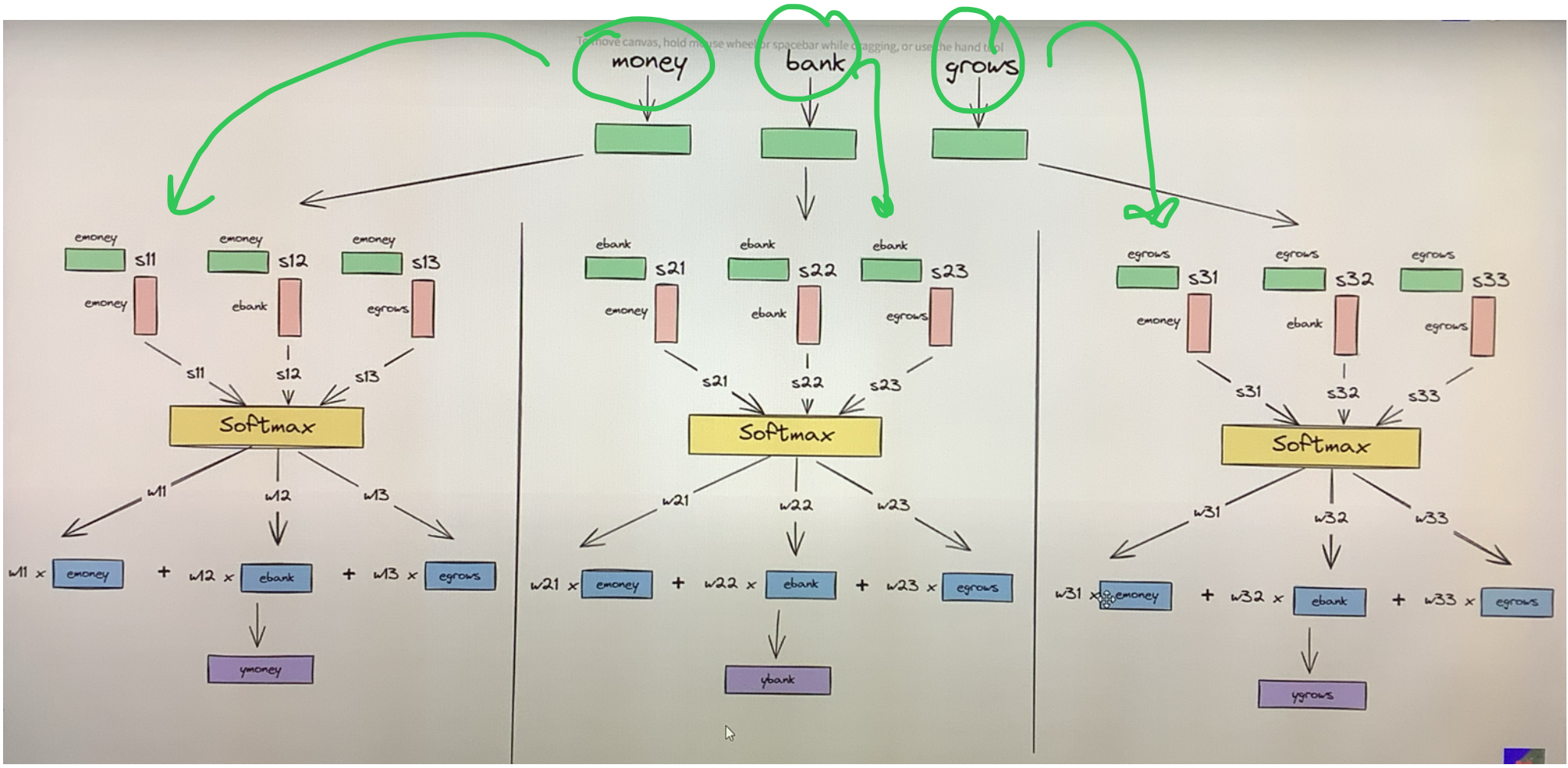
e_{money}^{new} is weighted average of (e_{money}, e_{bank}, e_{grows})

these Weights (0.7, 0.2, 0.1) represents similarity e_{money}^{new} & (e_{money}, e_{bank}, e_{grows}).

Similarity between embedding (=Vector) can be obtained using

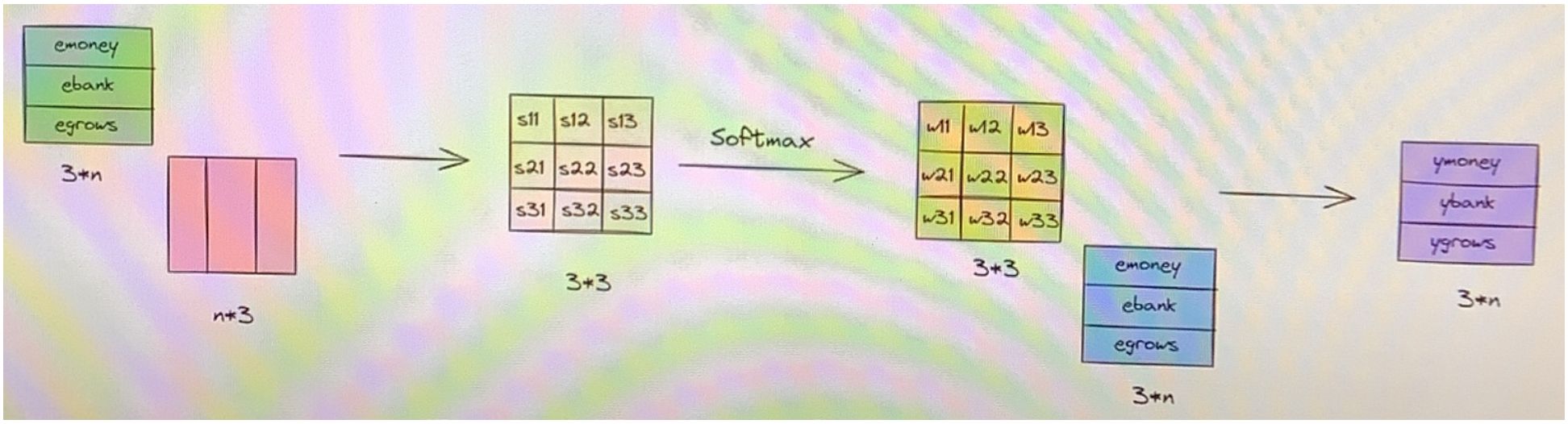
Dot product). Hence weights (0.7, 0.2, 0.1) can be considered as Dot product between those embeddings.





@ The above operation is parallel embeddings for each word can be calculated parallely they don't dependency of previous step/word.

Below is the Parallel Approach.



⊗ There is no Learning Parameter.

what it means our embeddings is not dependent on the data. Ex:-

eng → hindi

< ENG >	< HINDI >
How are You	तुम कैसे हो

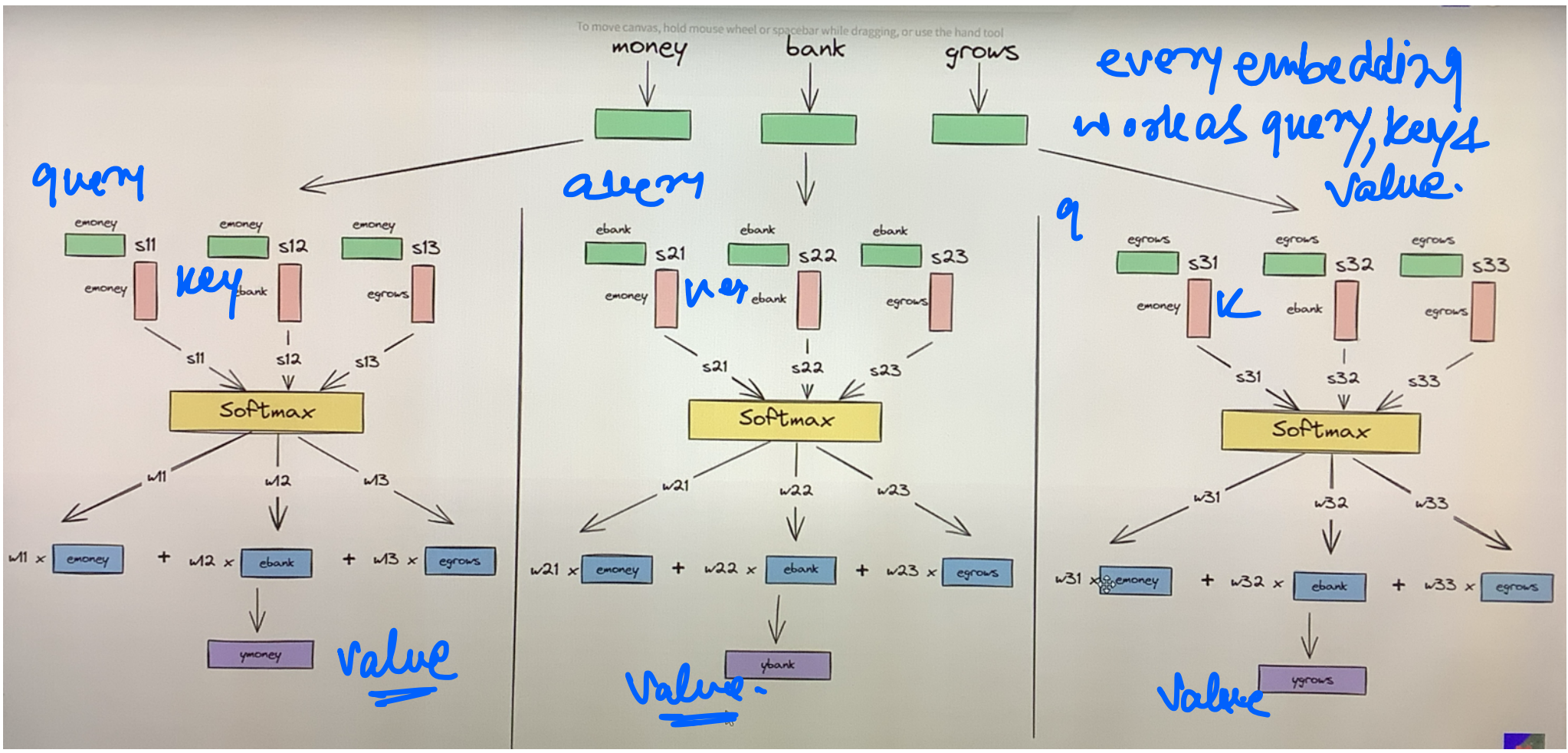
Translation is not TASK specific. problem

Ex: piece of cake → आसान काम] Dataset

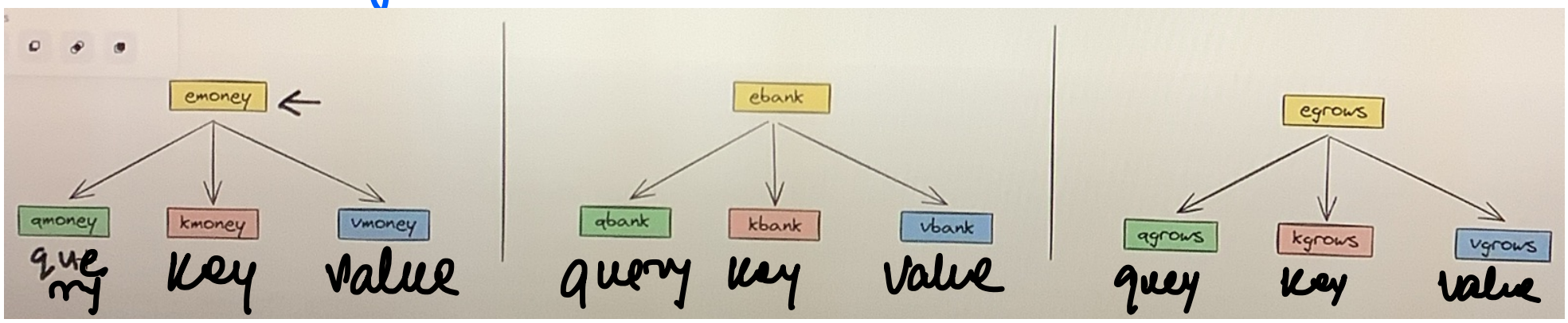
but it will generate केक का टुकड़ा

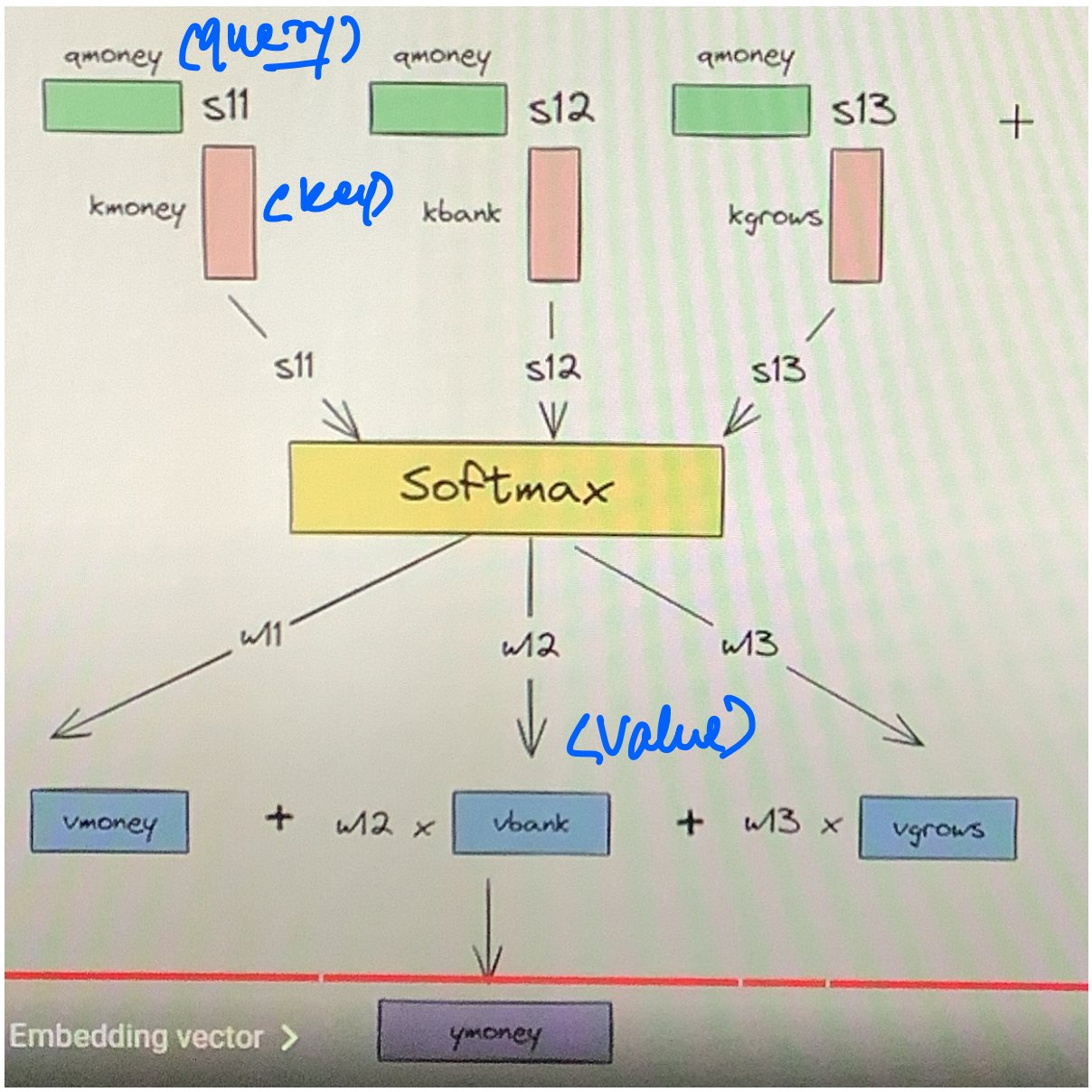
because their is no learning for DATASET. Hence we need

TASK specific embeddings. Hence we need Weights & Biases into process.



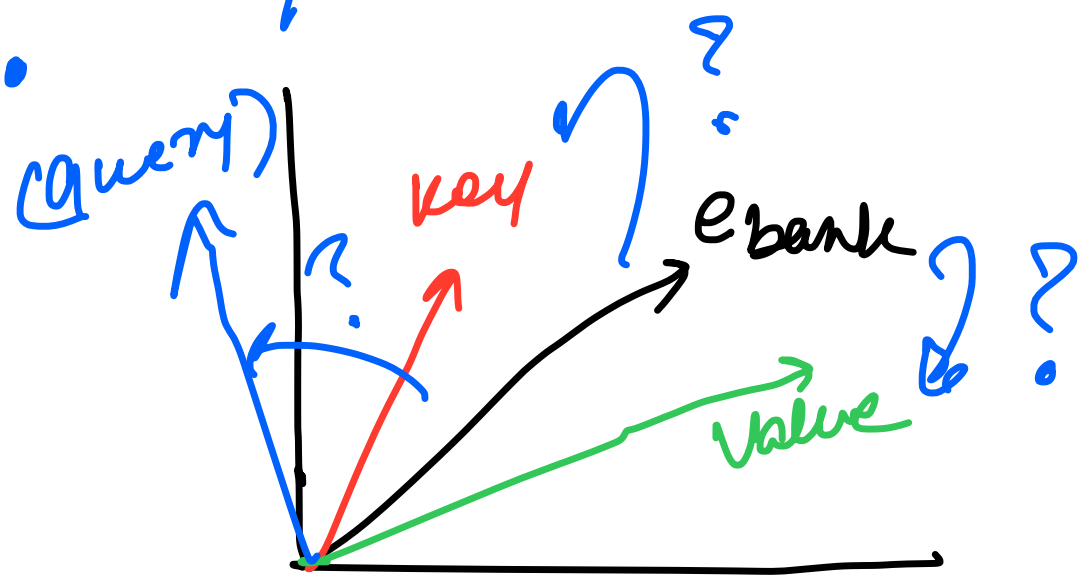
Now will take embedding create 3 embeddings out of that.



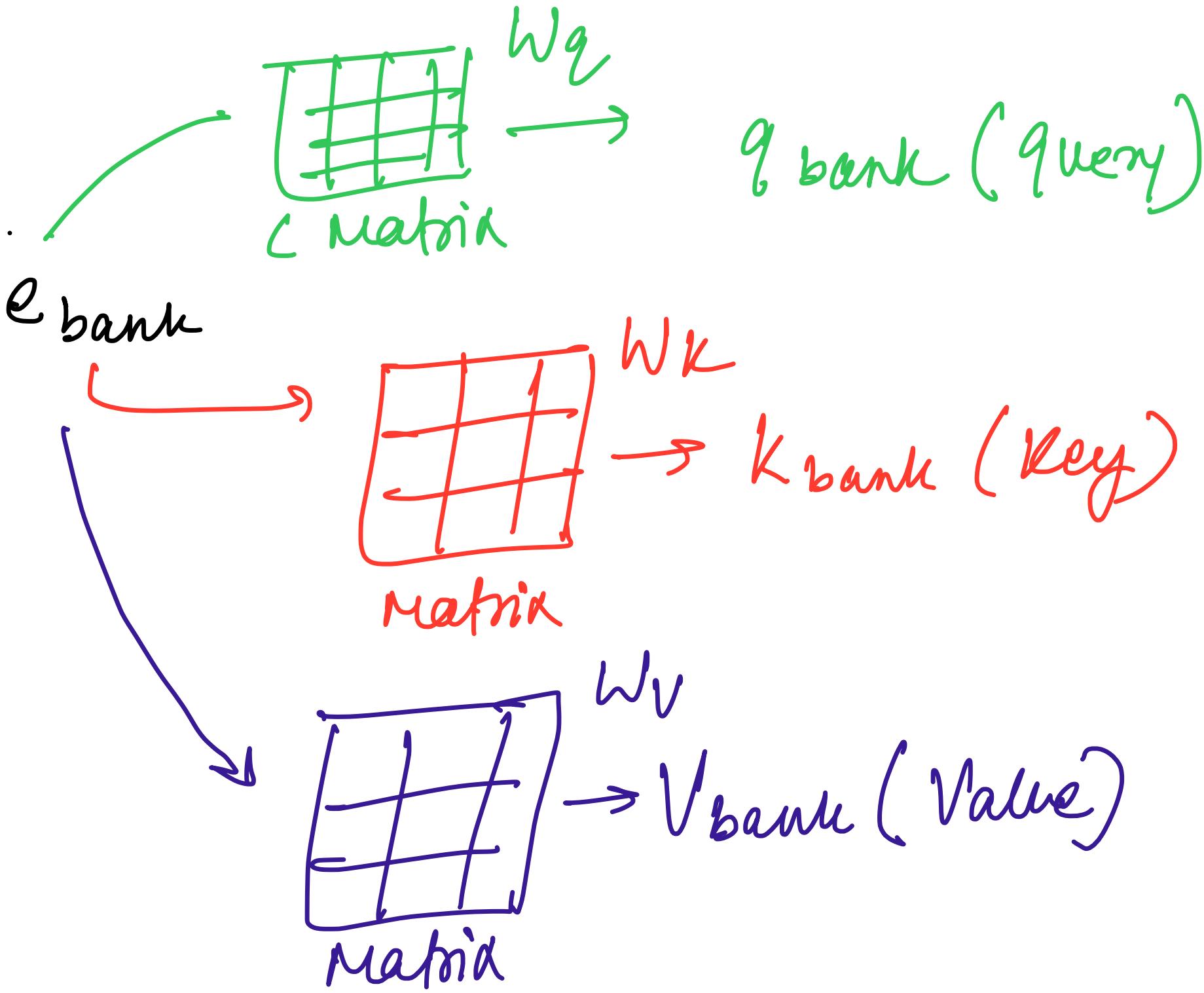


Will replace embeddings generated in this Architecture

Now we have to figure out how to make 3 embeddings (3 Vectors) out of ONE.

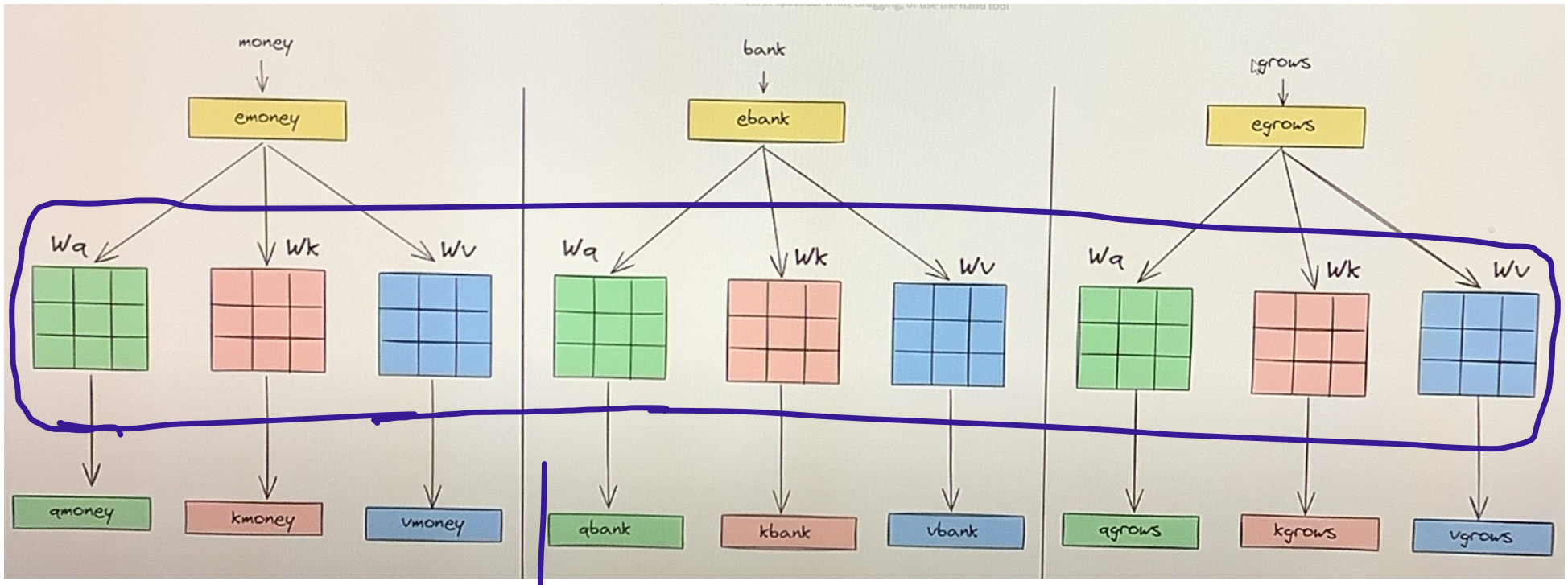


In Algebra we have limited way to do this in Linear Algebra Magnitude Transform or Linear Transform.

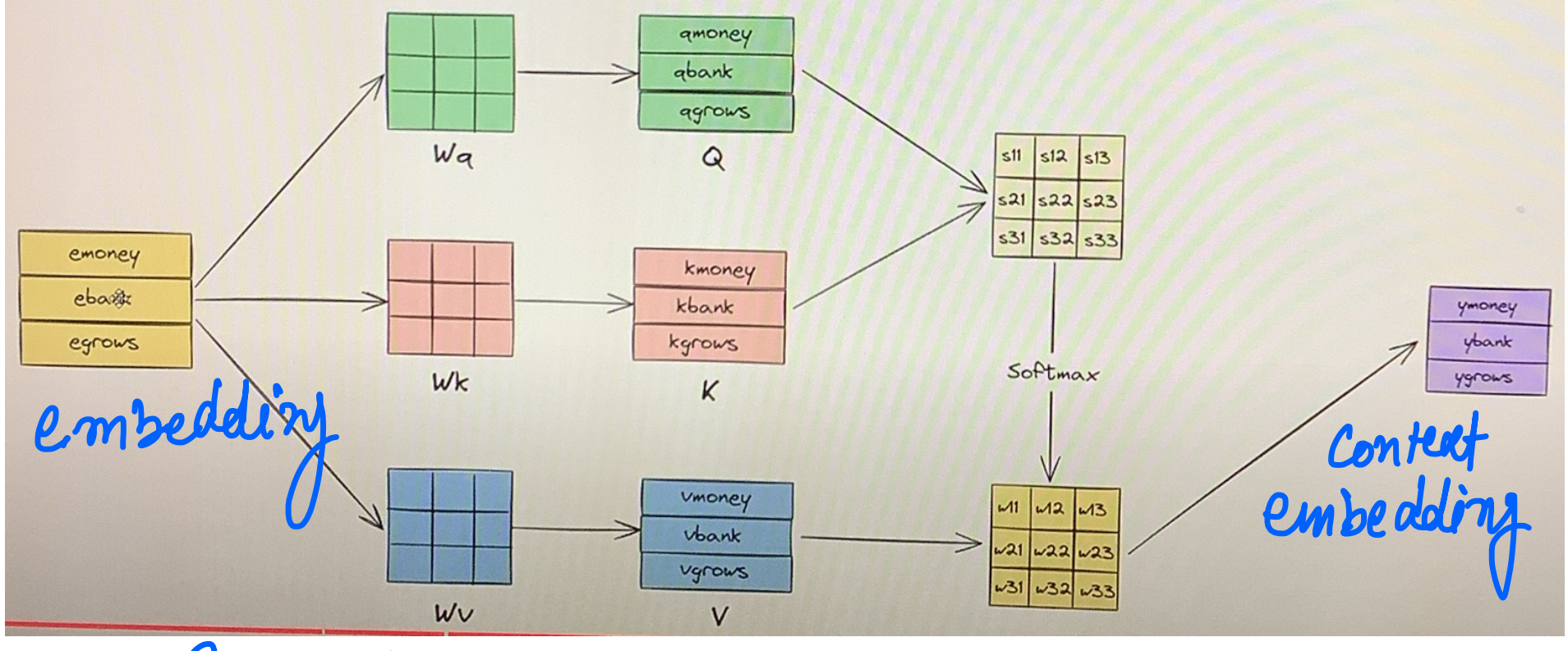


Now how to decide W^q, W^K, W^V.

Here comes to learning from Data.
will start with "RANDOM Weights".



Creating (Query, Key, Value) from embeddings. ↳ (Same matrix for all words)



(Parallelization)

Now Attention can be put mathmetically like below:-

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(QK^T\right)V$$

Q: query, K: key, V: value.

Same formula from paper

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

we are scaling with $\sqrt{d_k}$. Hence in paper we called it Scaled Dot product.

Now Why Scaling? Why $\sqrt{dk}$

dk = dimension of context embeddings.

Why Scaling: Nature of Dot Product

Low Dim → Dot Product → Low Variance

High Dim → Dot Product → High Variance

1 dim → Var(X)

2 dim → 2*Var(X)

3 dim → 3 * Var(X)

- - - - - - -

d dim → d * Var(X)

$$\left[\begin{array}{l} \text{if } X \rightarrow \mathrm{Var}(X) \text{ if } Y = cX \\ Y \rightarrow \mathrm{Var}(Y) = c^2 \mathrm{Var}(X) \end{array}\right] \otimes$$

Now we want to retain Var(X) independent of Vector dimension. So if we do $\frac{Y}{\sqrt{2}} \rightarrow \frac{1}{2}\mathrm{Var}(Y) \rightarrow \frac{1}{2}\times \mathrm{Var}(X)$

$\approx$ it will retain $\approx$ Var(X).

hence divide by $\sqrt{d_k}$

$$\mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \underline{\text{Attention.}}$$

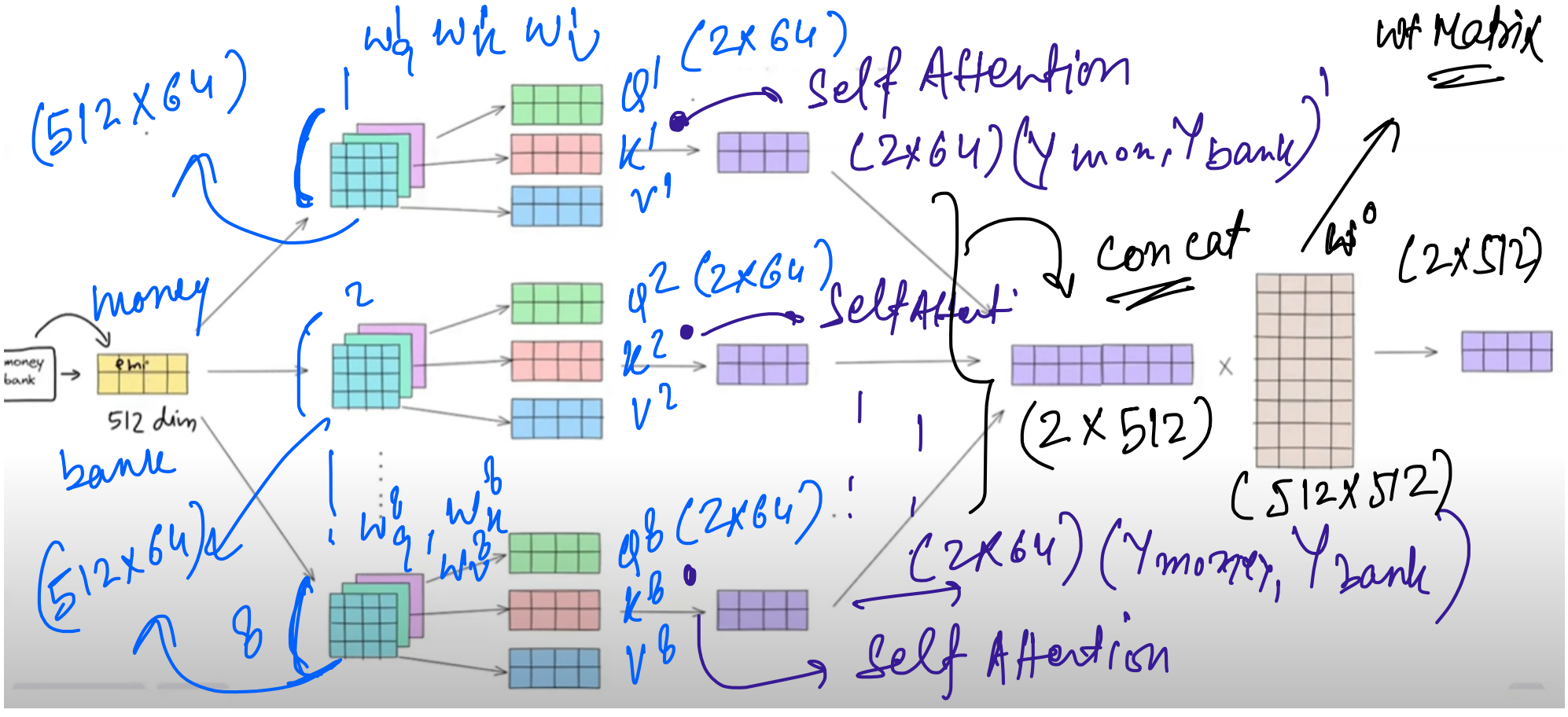
Multi Head Attention := Single Head Attention capture only ONE perspective off Text. Ex: Summarize document can be done in multiple ways but Self-Attention capture only one perspective.

[The man saw the astronomer with a telescope.]

Sentence have Two meanings.

So we use more than one Self-Attention

to capture different meanings. Ex
& self-attention. 512 dim vector



We reduce 512 → 64 to handle

computation.

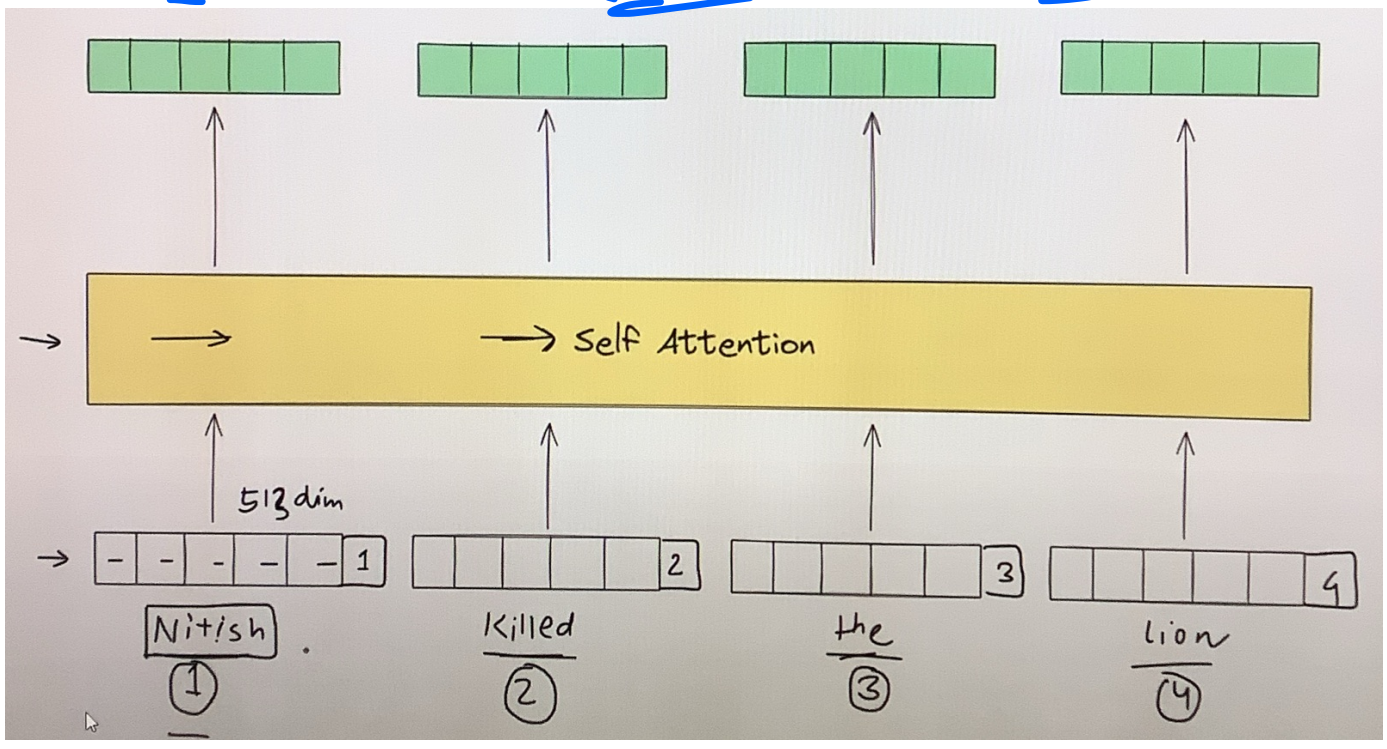
Positional Encoding :- Self attention has no mechanism to understand the order of words.

- Nitish killed lion } → S1
- Lion killed Nitish } → S2

Hard to differentiate S1 & S2.

- Because RNN is One Word at a time so this problem is not faced.

Simple Solution Word Count !=



Problems: 1. Unbounded

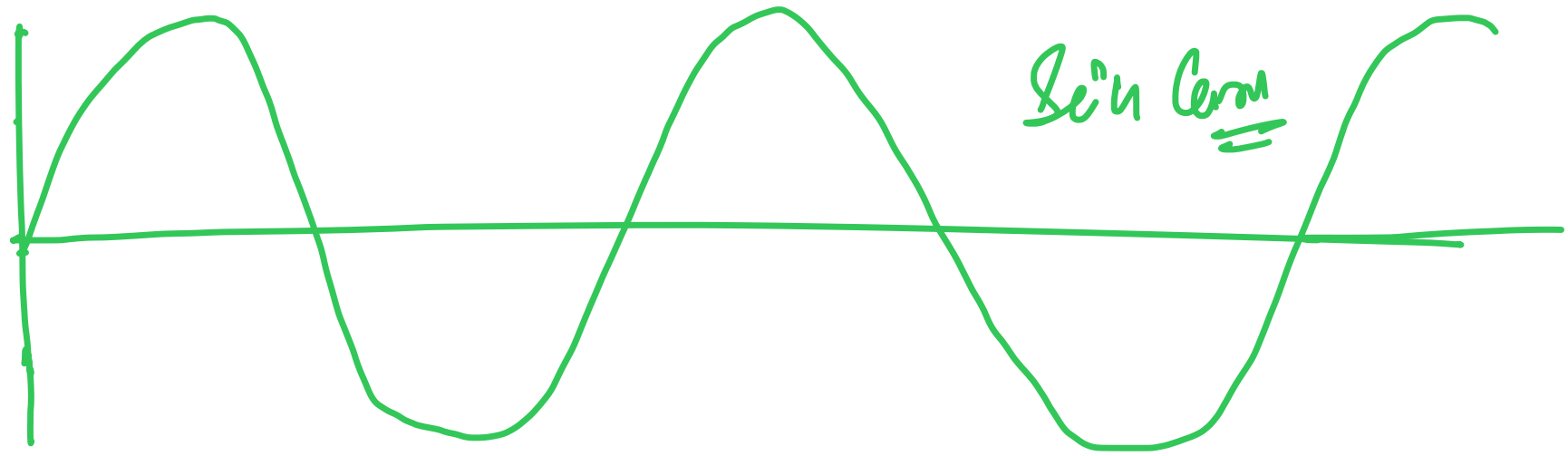
2) Discrete positions not smooth transition

3) No relative positioning.

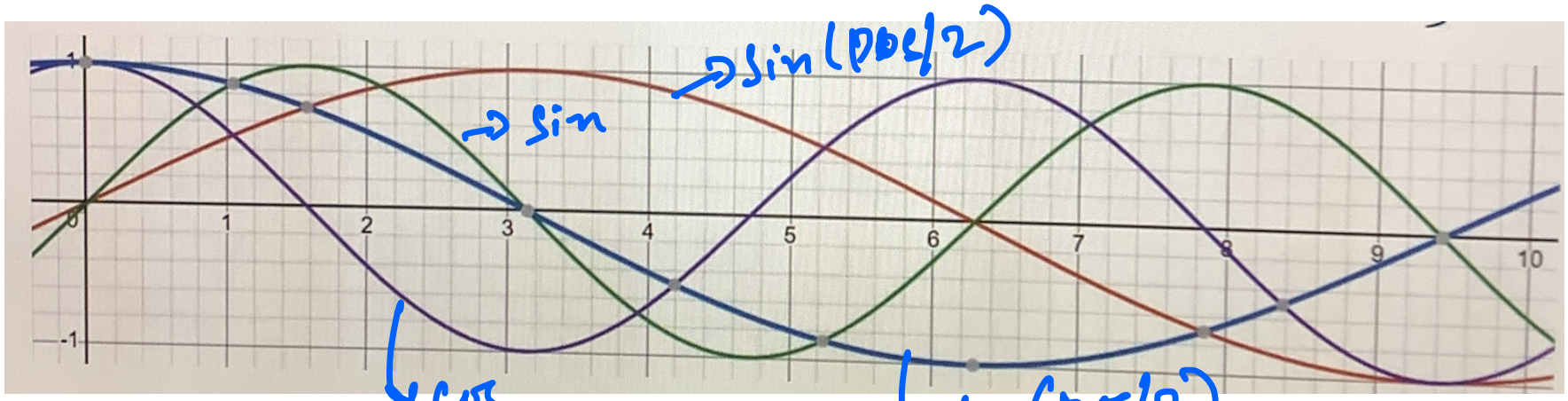
What we want a function to be

1. bounded 2. continous 3. periodic

Options are Trignometric functions



$Y = \sin(\text{position})$ as Sin is periodic we can have same Y for different position. Will use multiple functions.



So the positional embedding will be a vector. Chances of having same embedding is less. We can keep on increasing numbes of function.

② Size of Word embedding & positimal encoding vector will be same size.

embedding = (WordEmbedding + PosEmbedding)

$$PE(pos, 2i) = \sin\left(pos / 10000^{2i/d_{model}}\right)$$

$$PE(pos, 2i+1) = \cos\left(pos / 10000^{2i/d_{model}}\right)$$

pos = Position of the word

d_{model} = Dim of embedding

$i = (0 \rightarrow d_{model}/2 - 1)$ $(0 \rightarrow 2)$

0 1 $(0, 1, 2)$

river bank

$\langle Pos=0, i=0 \rangle$

$P(0,0) = \sin\left(0 / 10000^{0/d_m}\right) = 0$

$P(0,1) = \cos(0 / 10000^{0}) = 1$

$\langle Pos=0, i=1 \rangle$

$P(0,2) = \sin(0/10000^{1/3}) = 0$

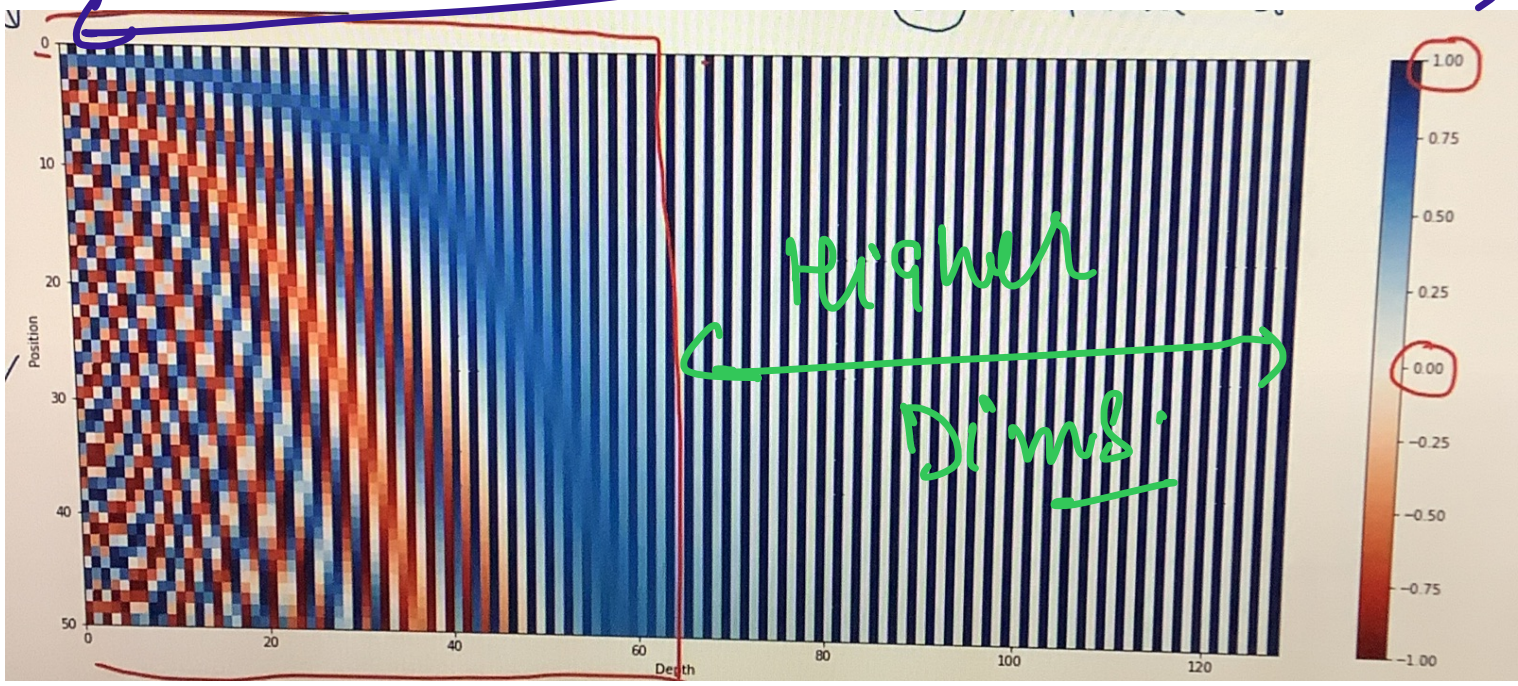
$P(0,3) = \cos(0/10000^{1/3}) = 1$

$\langle Pos=0, i=2 \rangle$

$P(0,4) = \sin(0/10000^{2/3}) = 0$

$P(0,5) = \cos(0/10000^{2/3}) = 1$

River $\Rightarrow$ 0, 1, 0, 1, 0, 1



50 words sentence.

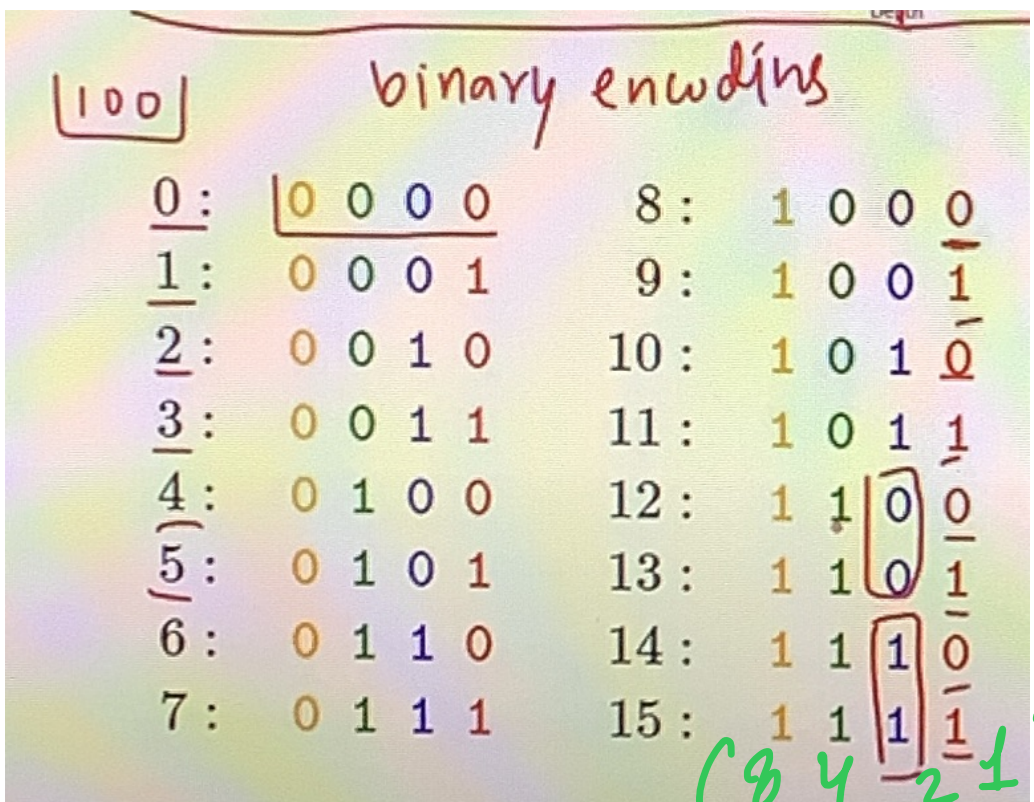
As we can see pos=0, have 0, 1, 0, 1 ...

as we have found for word river.

So as we can see Higher Dimension

is almost same.

Analogy to binary encodings.

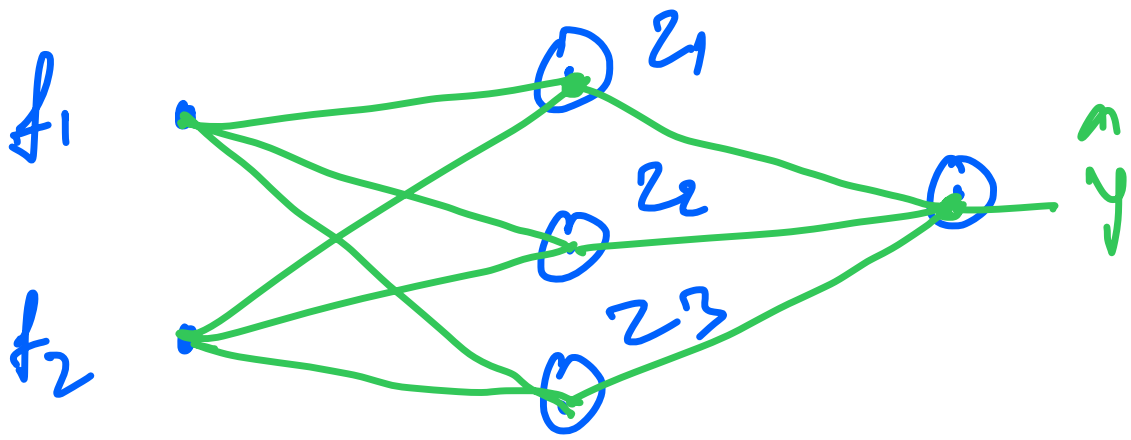


(8 4 2 1) frequency.

As we can see higher bit level less frequently it changes

Layer Normalization :=

Revise Batch Normalization :=



BATCH NORM

S	f_1	f_2	z_1	z_2	z_3
S_1	2	3	7	5	4
S_2	1	1	2	3	4
S_3	5	4	1	2	3
S_4	6	1	7	5	6
S_5	7	1	3	3	4

Across Batch

is Batch Norm

Batch Direction

Column Wise

Across features → (Layer Norm)

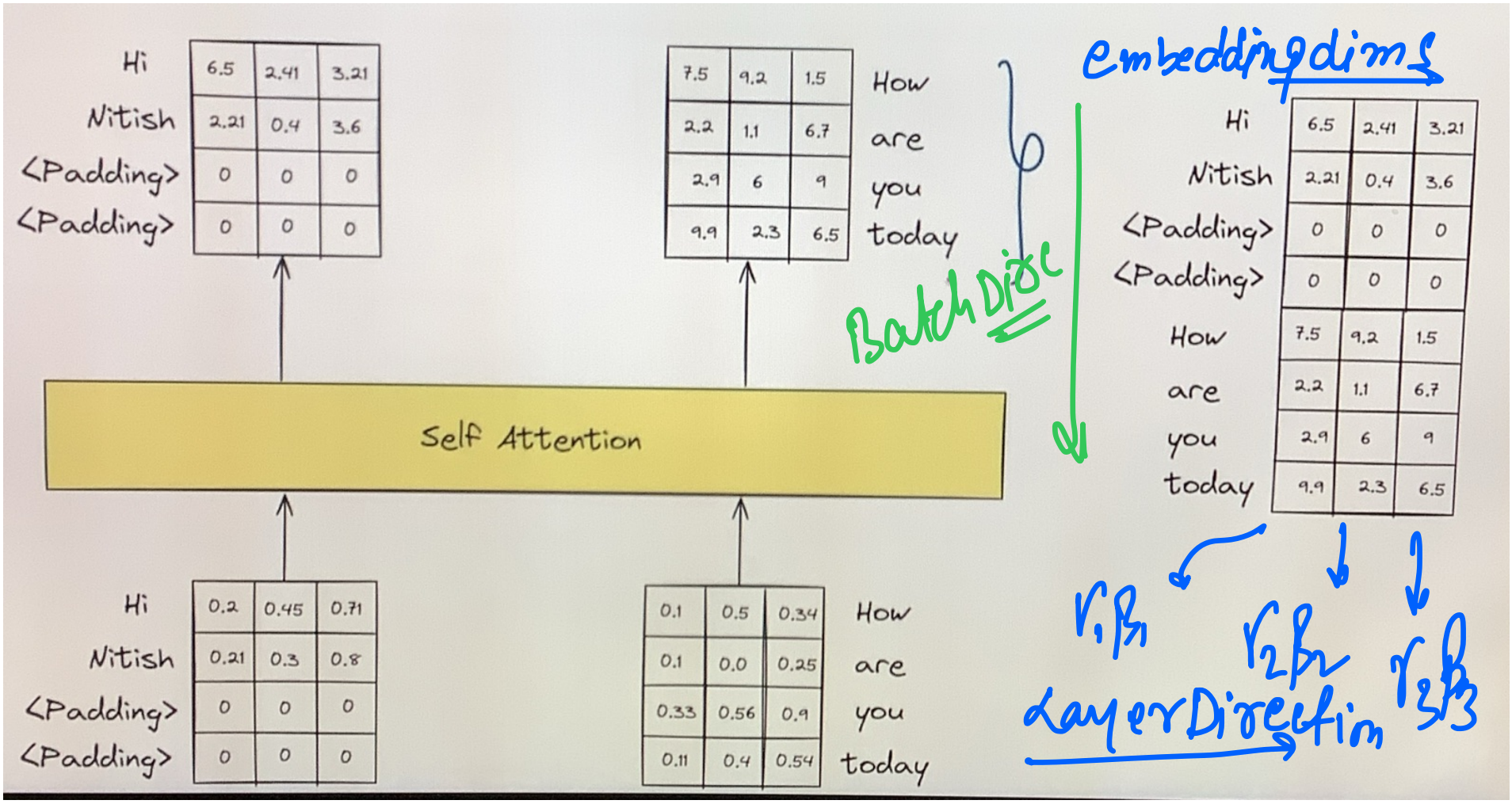
S	f_1	f_2	z_1	z_2	z_3
S_1	2	3	7	5	4
S_2	1	1	2	3	4
S_3	5	4	1	2	3
S_4	6	1	7	5	6
S_5	7	1	3	3	4

Across features

Layer Norm

→ Layer Norm Direction

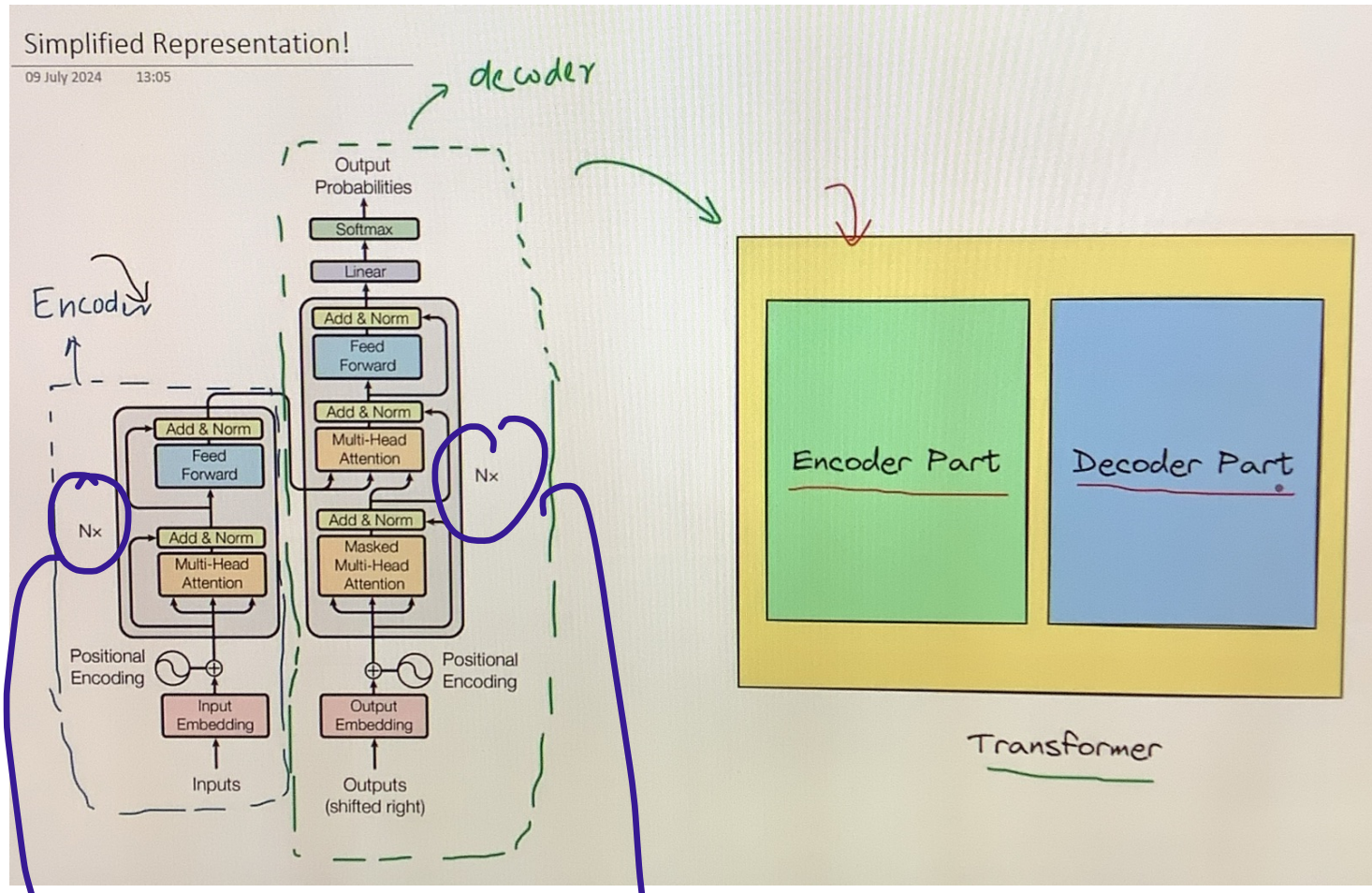
Why Layer Norm In Transformers?

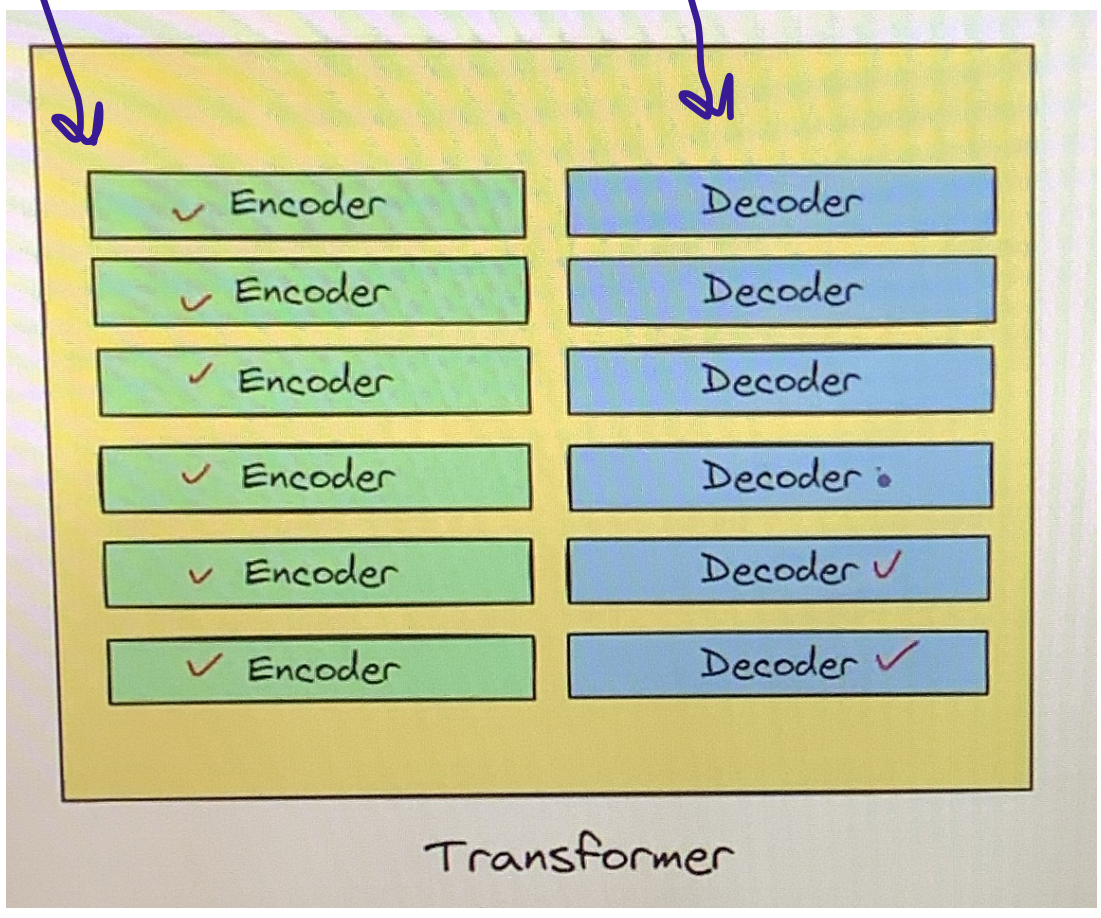


As we can see in Batch Norm (Padding) zeros will effect over all calculation.

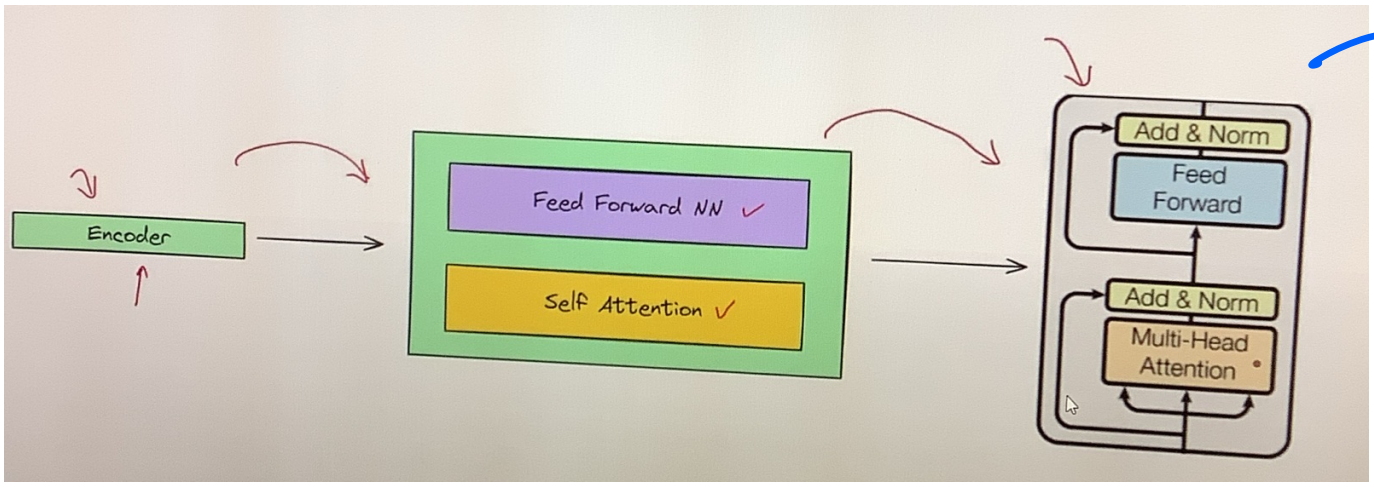
Assume a sentence with 3 words & >100 trailing zeros (Padding)

Transformer Architecture :-





Nx means there are multiple Encoder & Decoder block.



ONE encoder block.

OUTPUT

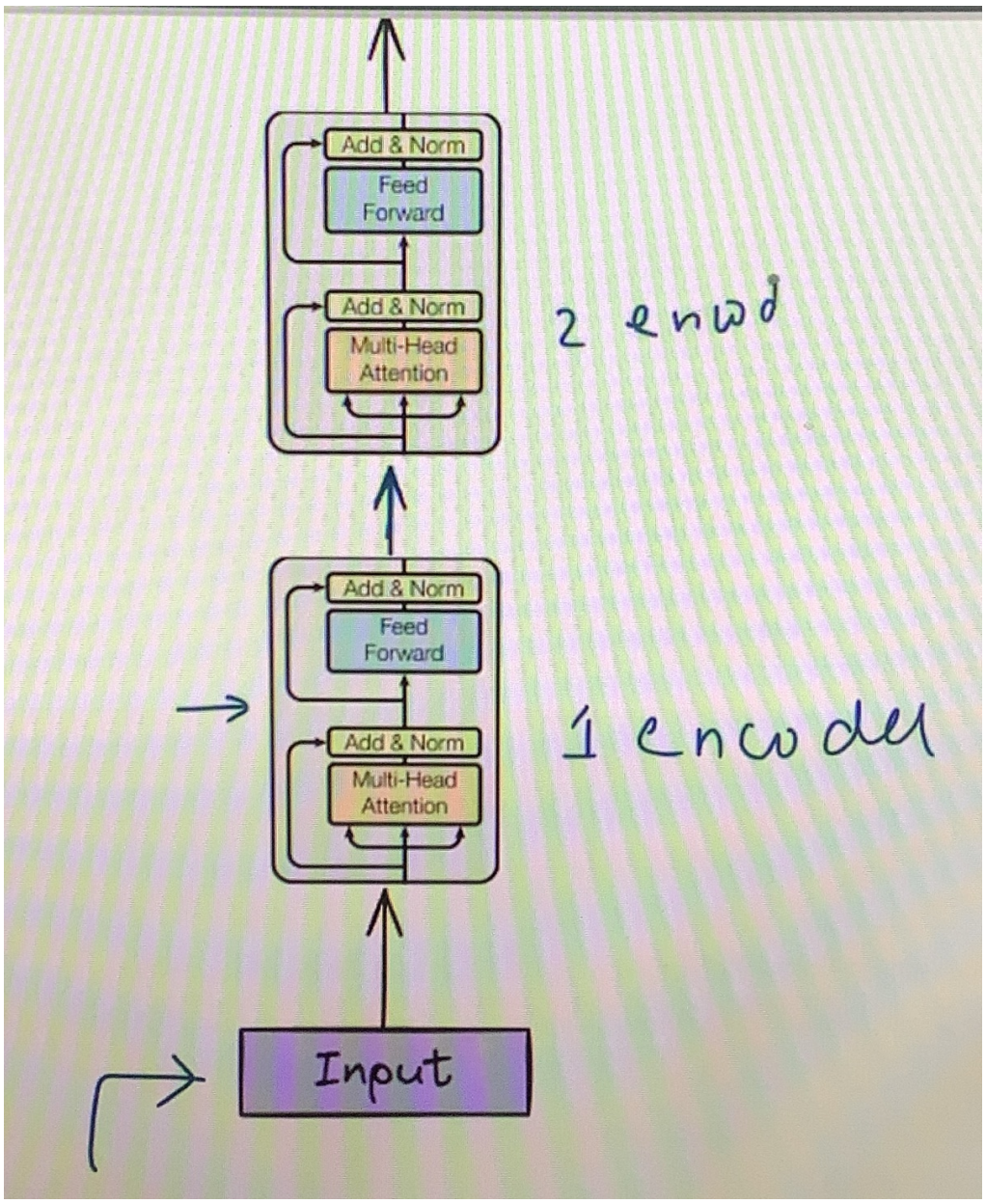
6th ENCODER BLOCK

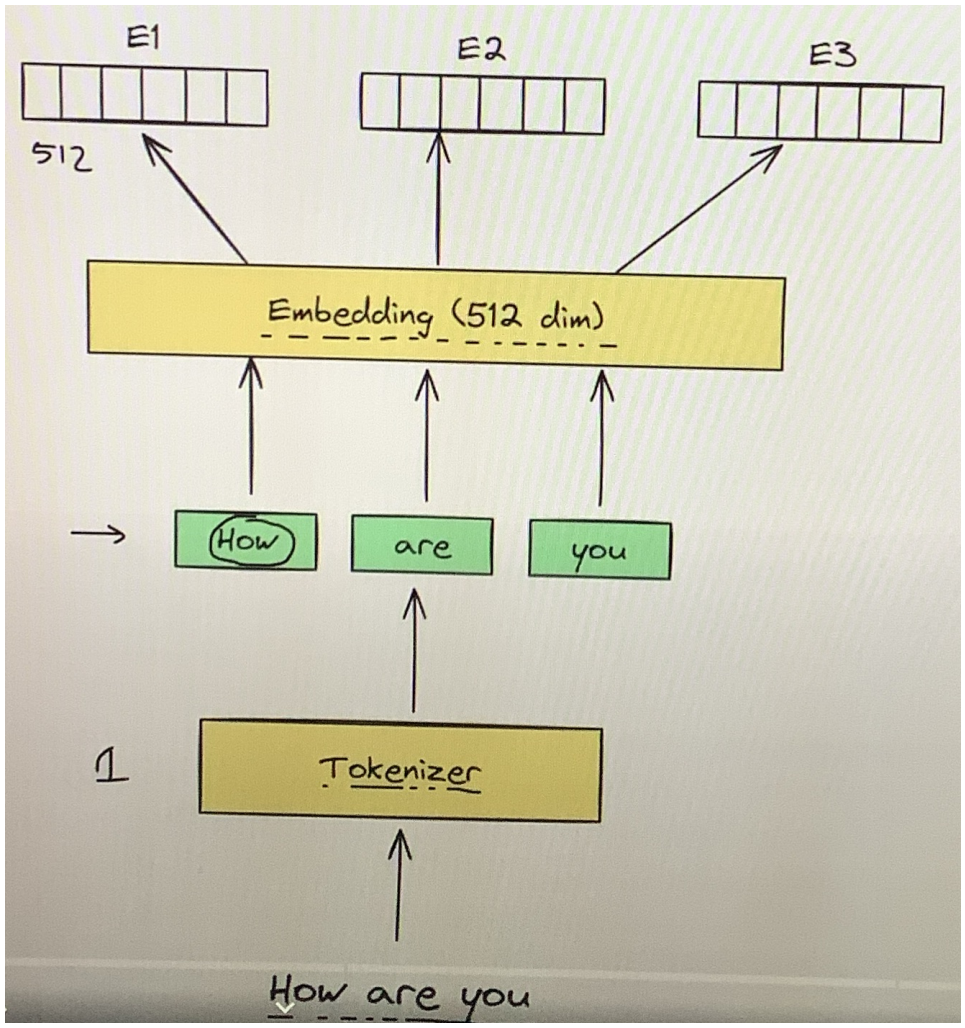
5th

4th

3rd encoder

Decoder Block



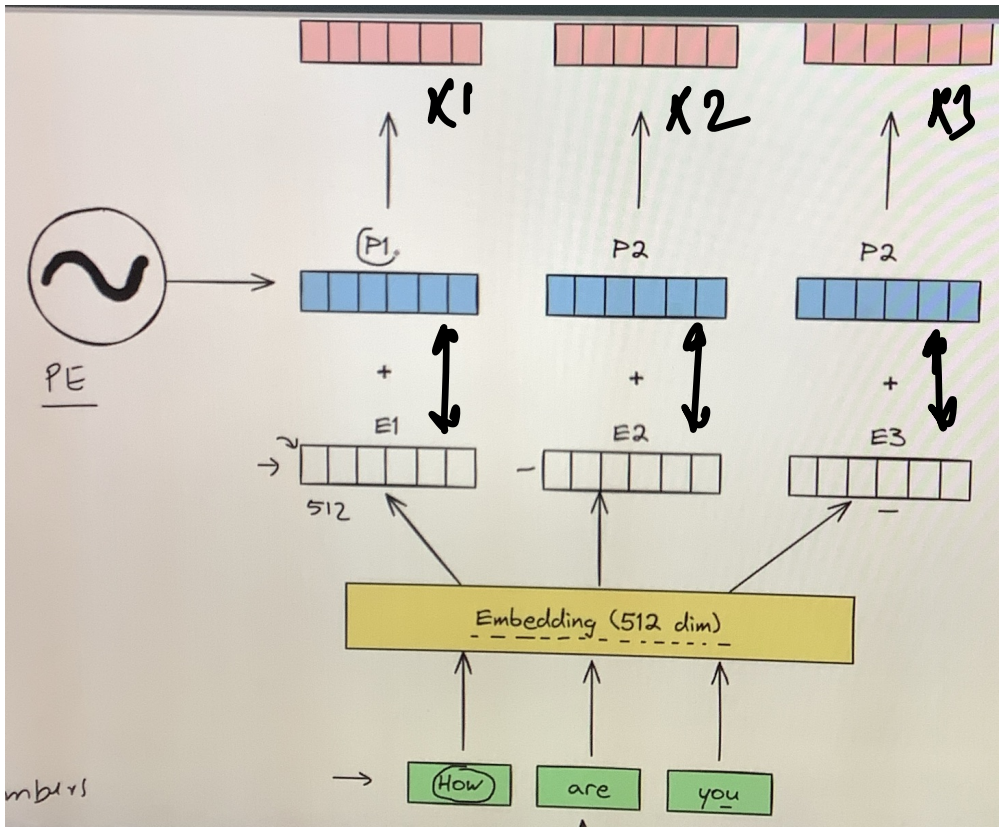


Step1: Text = "How are you"

get Tokenize. That Tokens

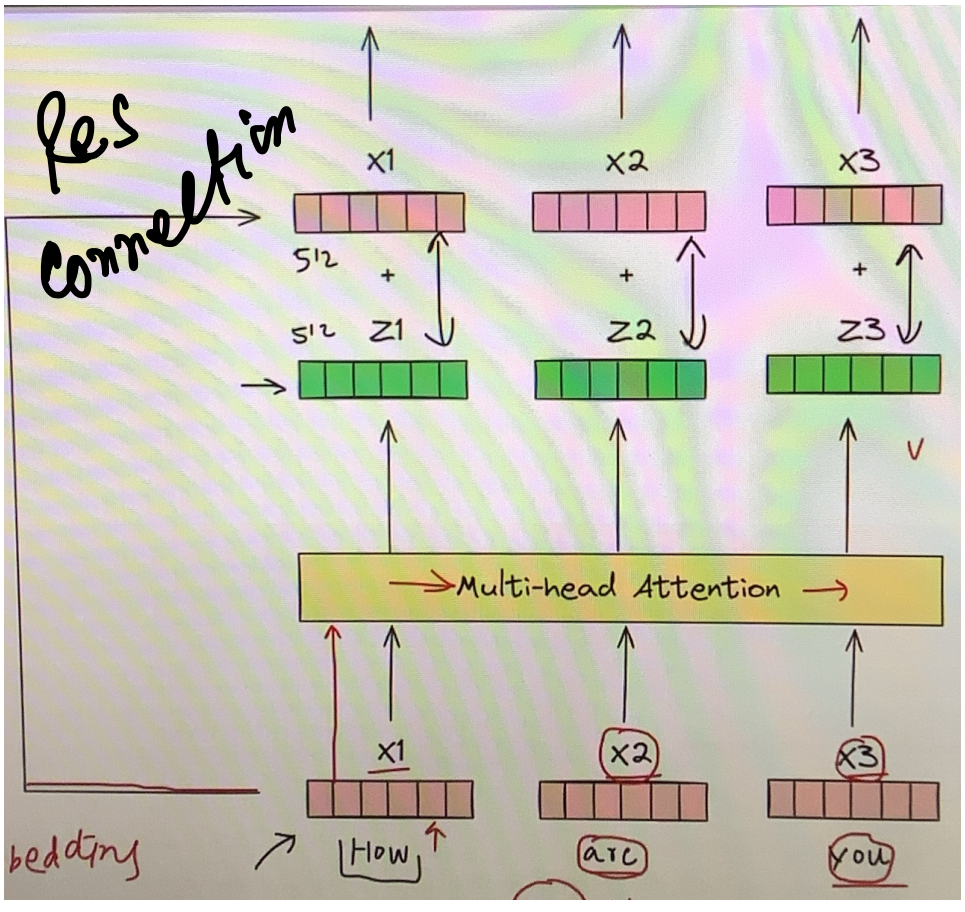
goes to embedding (word2Vec) etc

& create E1, E2 & E3.



Step 2: Incorporate positional encoding (P1, P2 & P3) + (E1, E2 & E3) generate (X1, X2 & X3)

(8) Now Input is ready :=

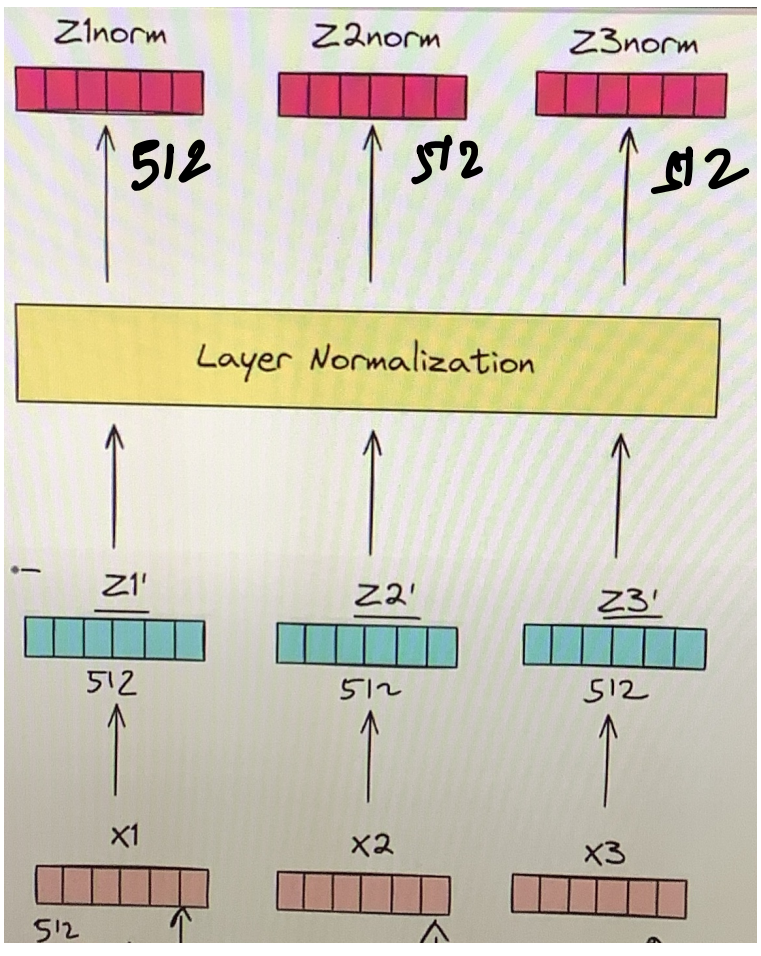


step 3: With

Self-Attention or Multi Head Attention

We create "CONTEXT"

aware embedding (z_1, z_2, z_3)



Step 4: $z_1' = z_1 + x_1$

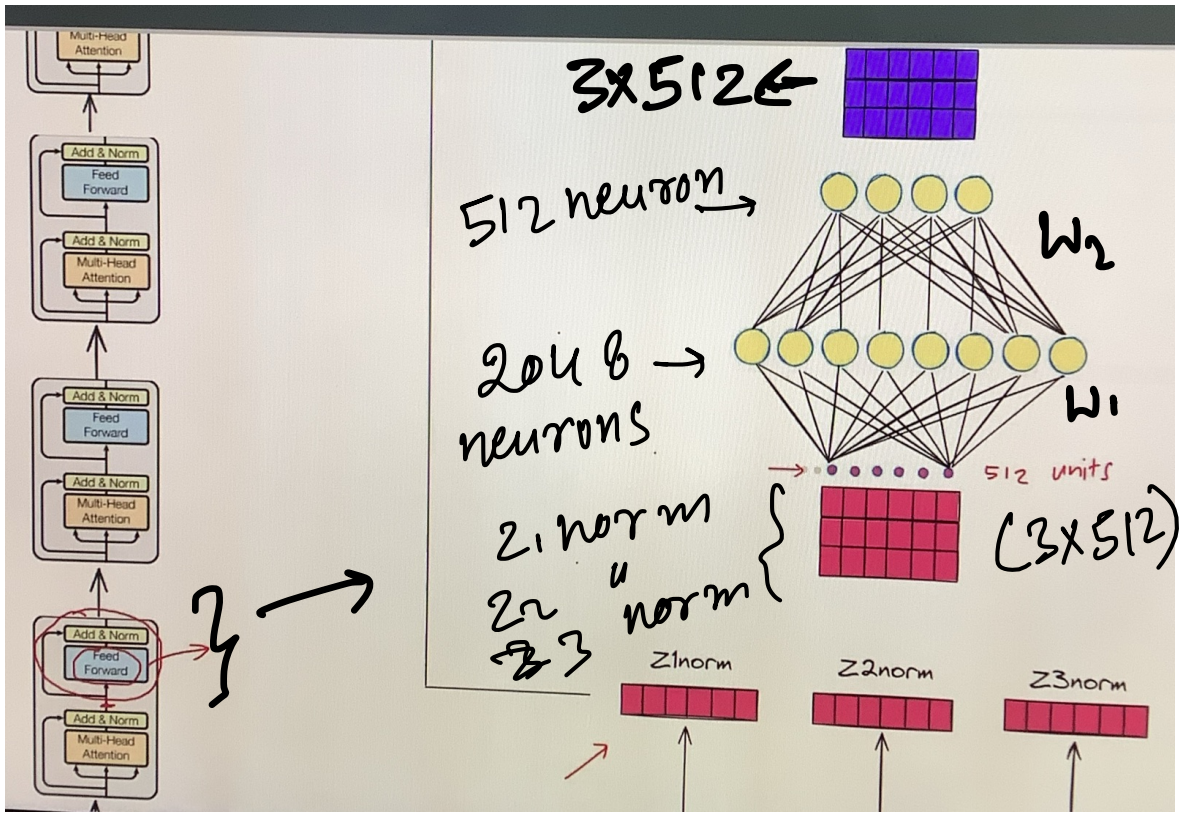
$z_2' = z_2 + x_2$

$z_3' = z_3 + x_3$

↳ Residual Connection

Then perform Layer Norm.

$z_i \rightarrow z_i$ norm.



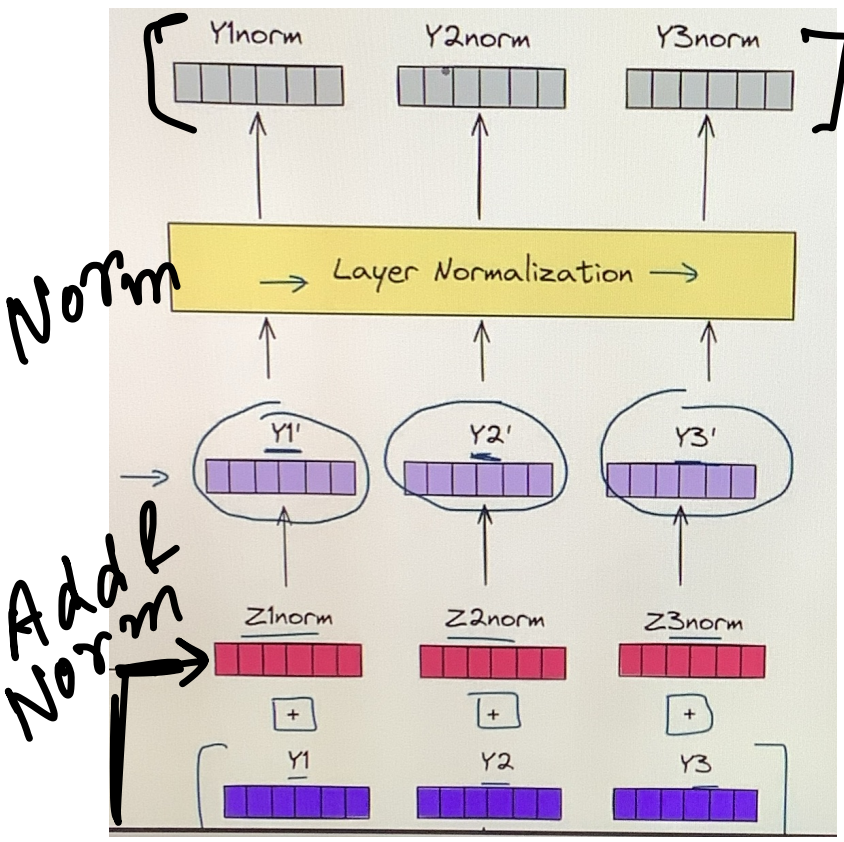
Step 3: W1 = 512 X 2048.

W2 : 2048 X 512

$$[(3\times 512)\times(512\times 2048)]+b_1=(3\times 2048)$$

$$(3\times 2048)\times(2048\times 512)+b_2=(3\times 512)$$

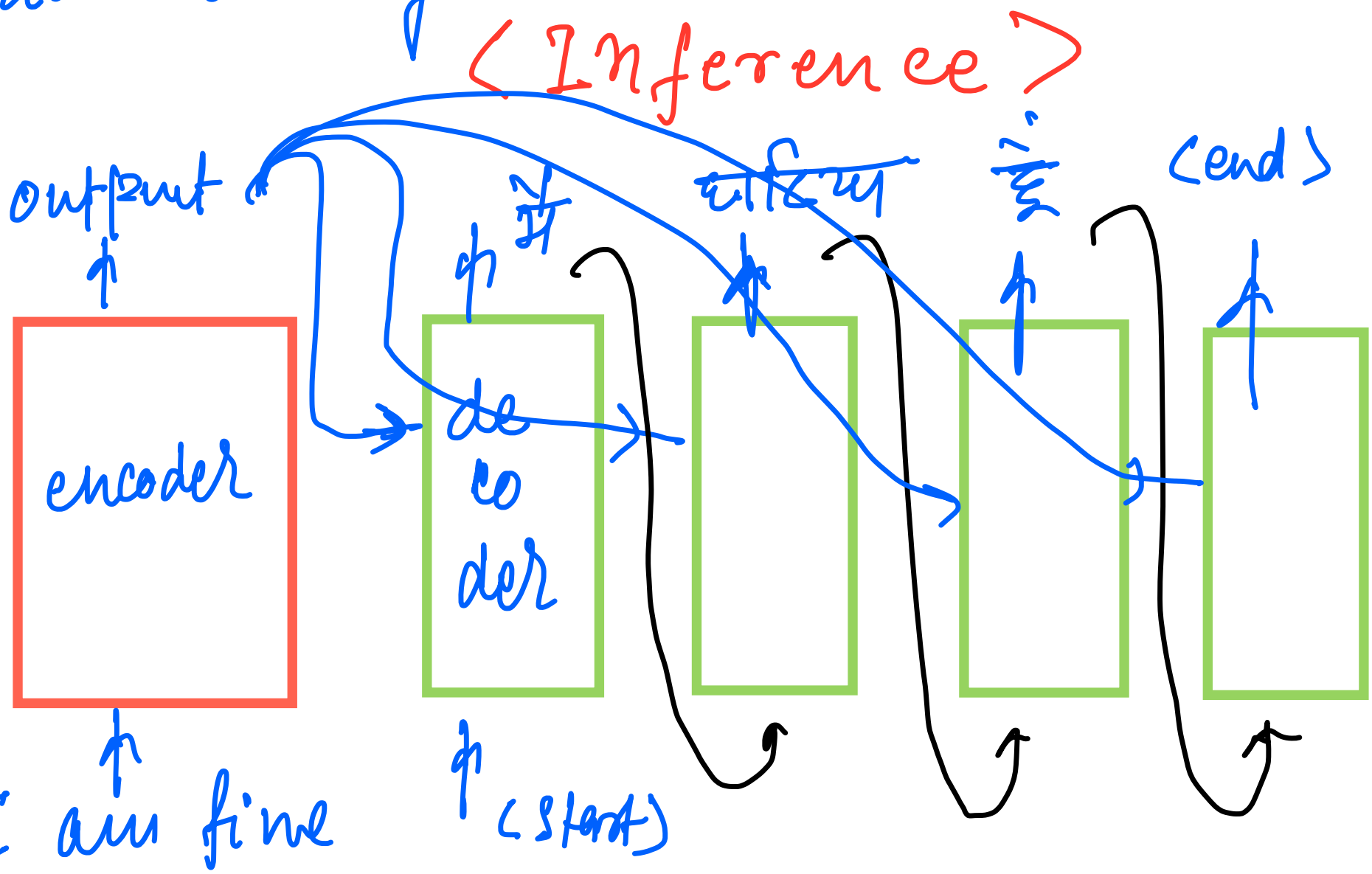
⊛ Added some Non-Linearity.



Output will be Input for next Encoder block. Each Encoder block have their own Wts & Biases.

Masked Self Attention :

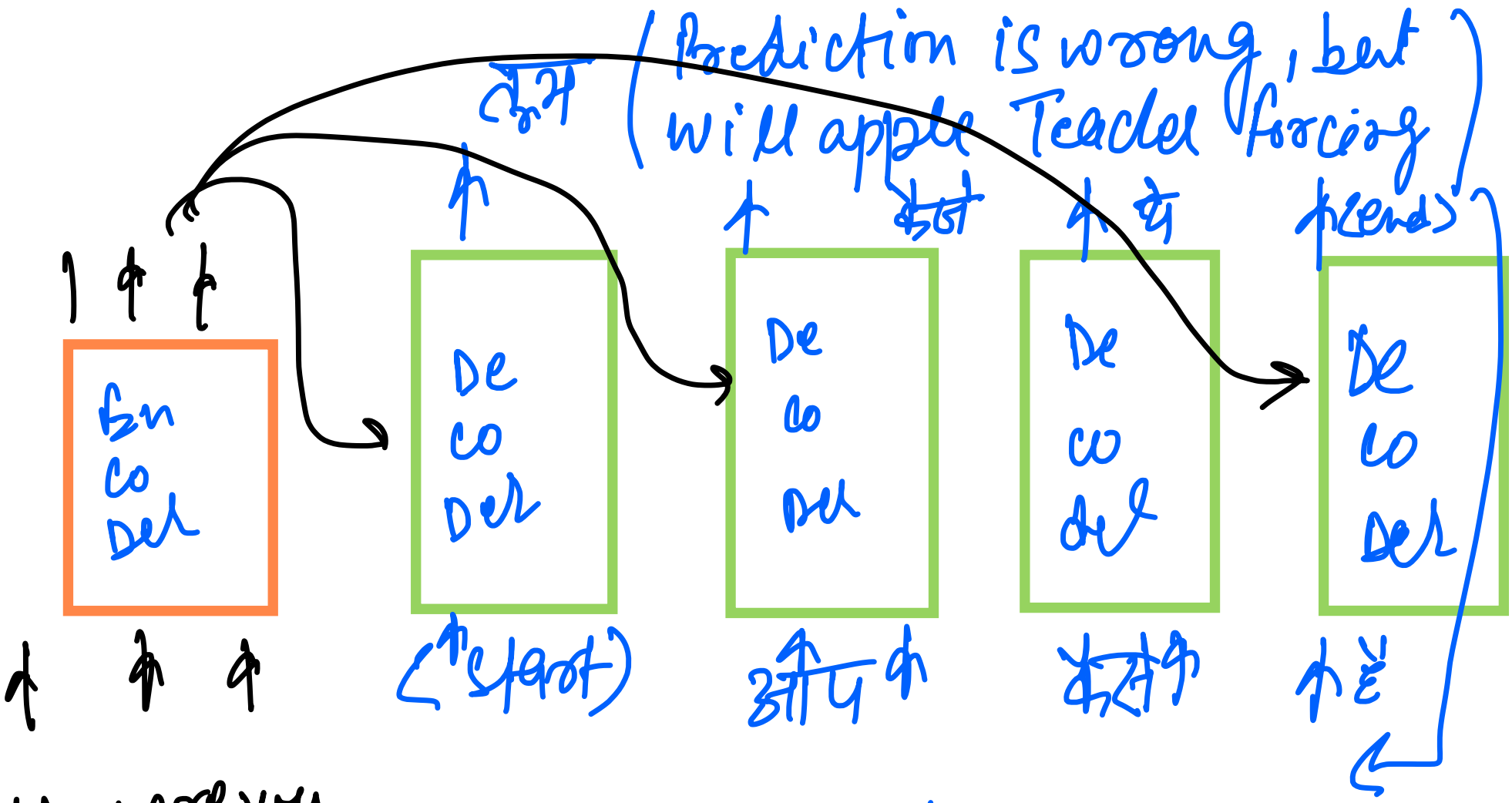
Transformer Decoder is auto regressive at inference time & non auto regressive at training time.



Training

S.No	English Sentence	Hindi Sentence
1	How are you?	आप कैसे हैं
2	Congratulations	बधाई हो
3	Thank you	धन्यवाद

(Data Set)



at training time will send the correct next prediction from our data set.

True label | Prediction

} Will calculate the loss & Backprop wt update.

The above process in auto-regressive steps are involved. Which makes the training process very slow.

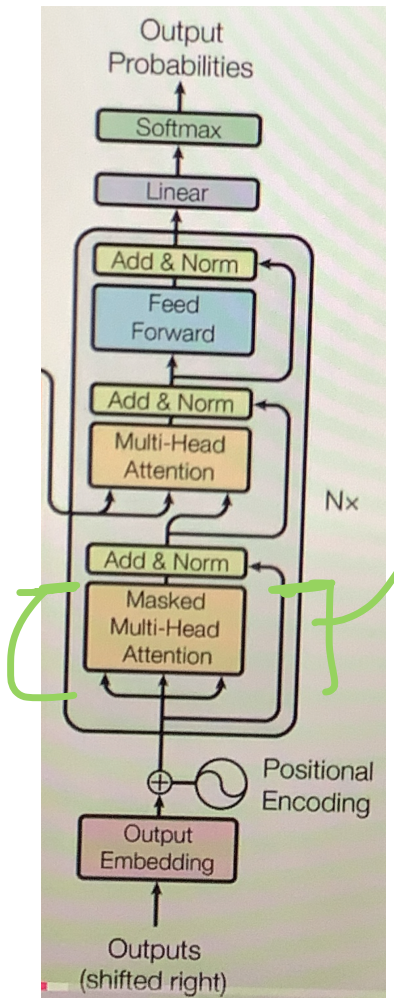
Making Parallel Training: There is no reason we should wait for previous step as the next Token coming from "DATASET". Hence "आप कैसे हैं", can be passed parallely.

Masked Multi-Head Attention :-

Just for the time being

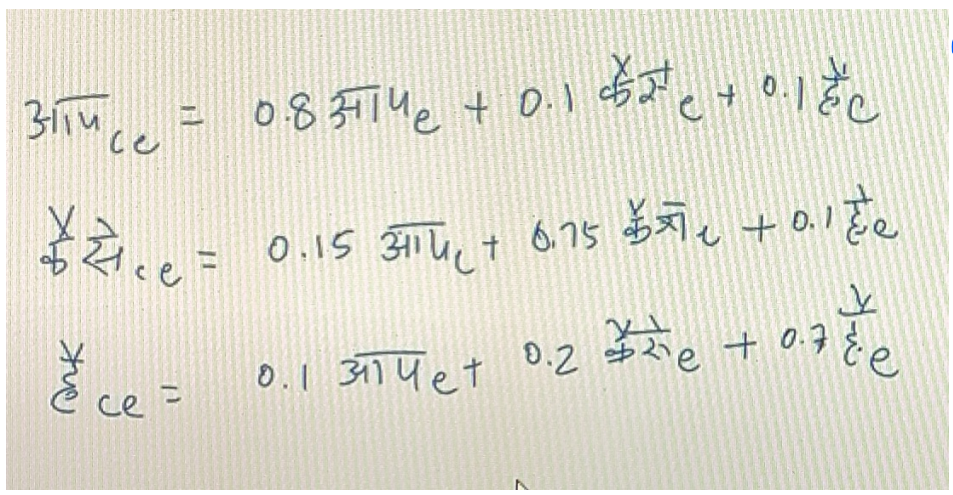
let's assume

Masked Multi Head Attention → Multi head Attention



Self Attention.

Word embedding → Self Attention → context embedding



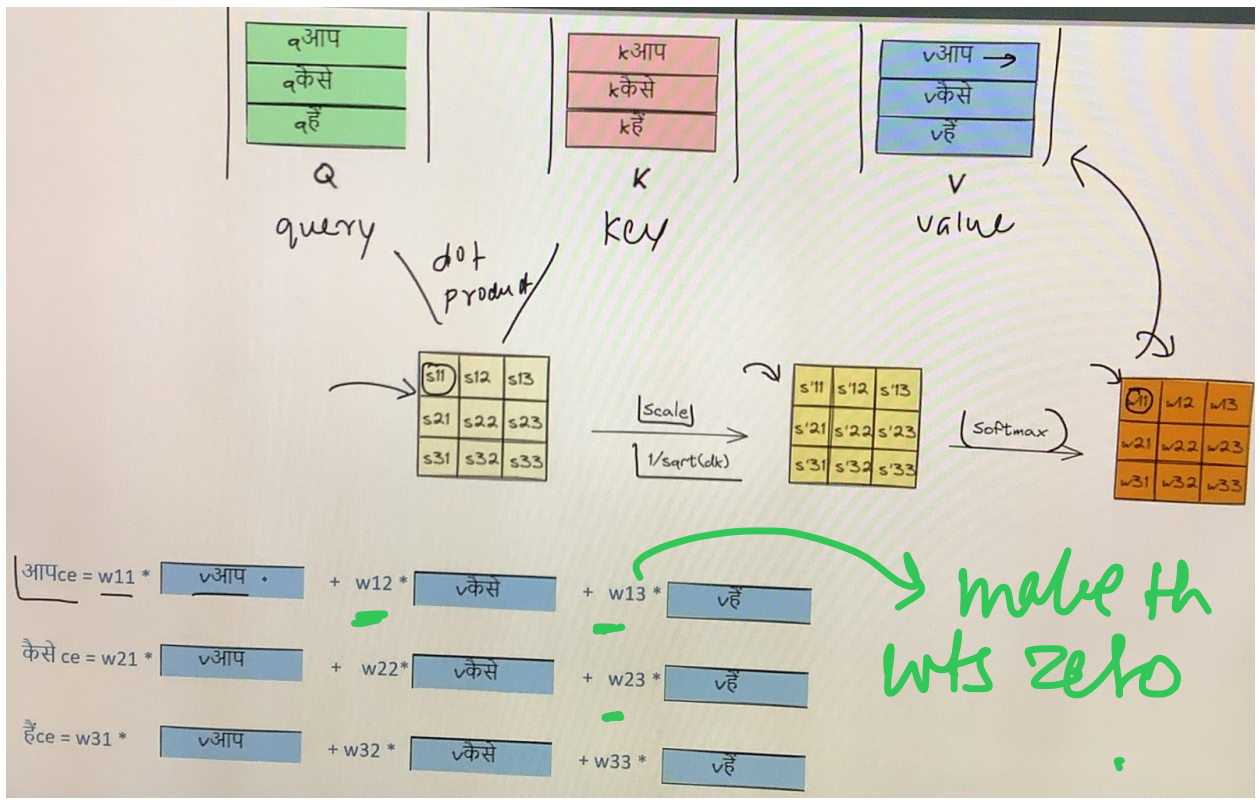
① $\overline{आप}_{ce} = 0.8\,\overline{आप}_{e} + 0.1\,\overline{कैसे}_{e} + 0.1\,\overline{है}_{e}$

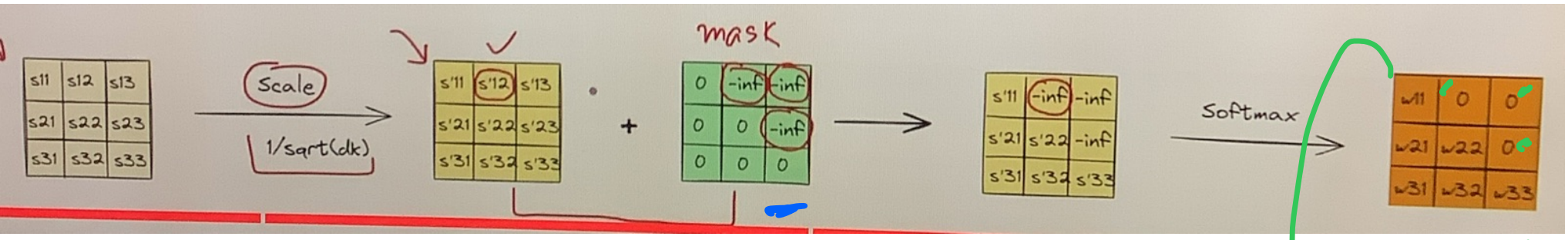
② $\overline{कैसे}_{ce} = 0.15\,\overline{आप}_{e} + 0.75\,\overline{कैसे}_{e} + 0.1\,\overline{है}_{e}$

③ $\overline{है}_{ce} = 0.1\,\overline{आप}_{e} + 0.2\,\overline{कैसे}_{e} + 0.7\,\overline{है}_{e}$

→ It's possible we can get some-thing like

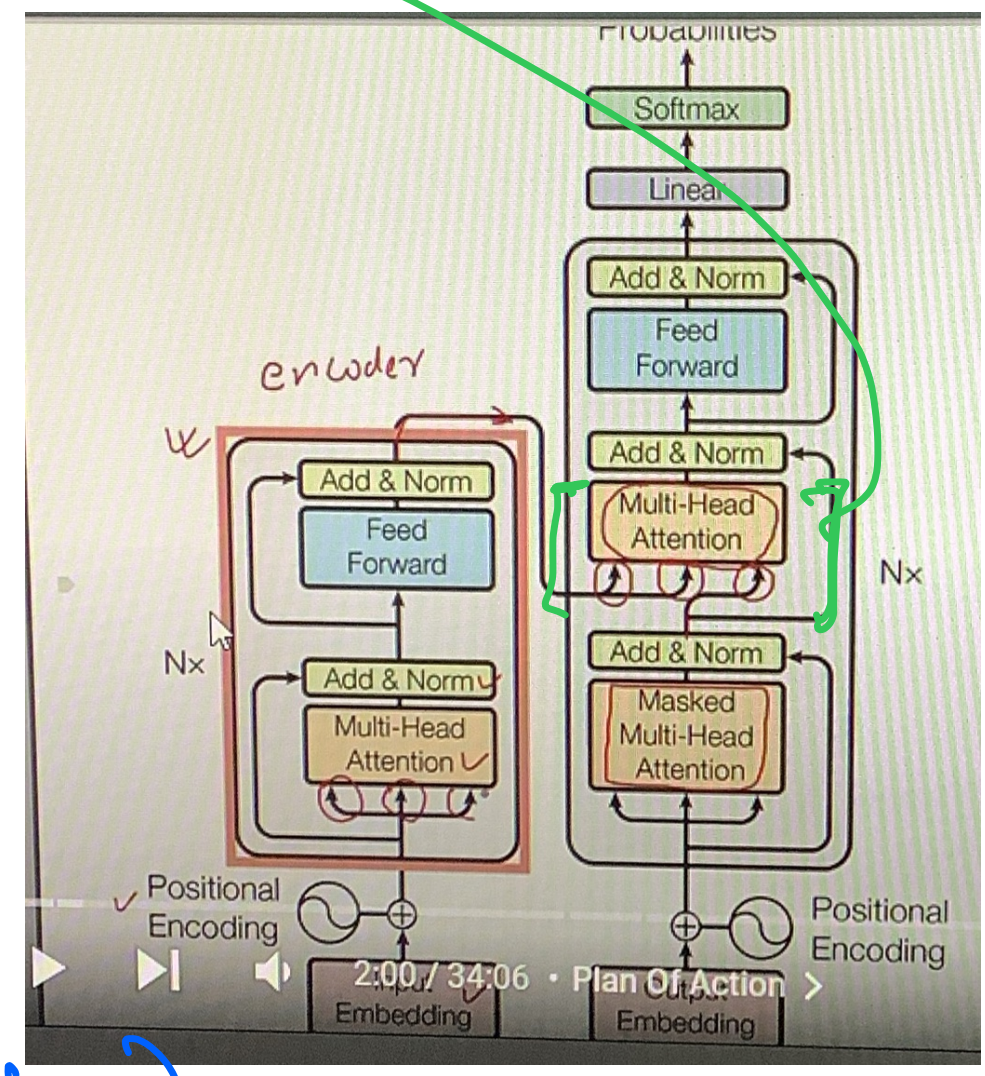
from eq ①, ② & ③ to represent a word we need all the words in sentence which is available at "Train Time" but not on "Inference time. Hence become case of "Data Leak" a





Cross Attention

This first instance where Encoder & Decoder input meets. (Cross Attention).

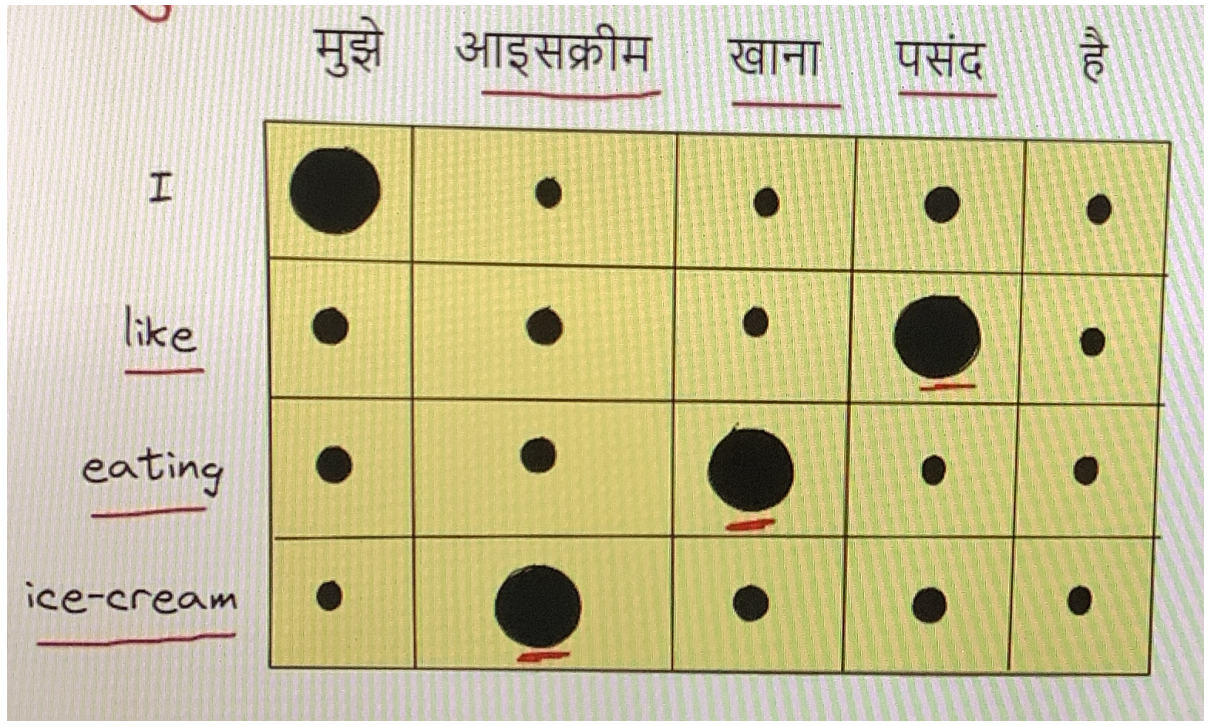


Let's take an example translation :-

I like eating ice-cream

↳ मुझे आईसक्रीम खाना पसन्द है।

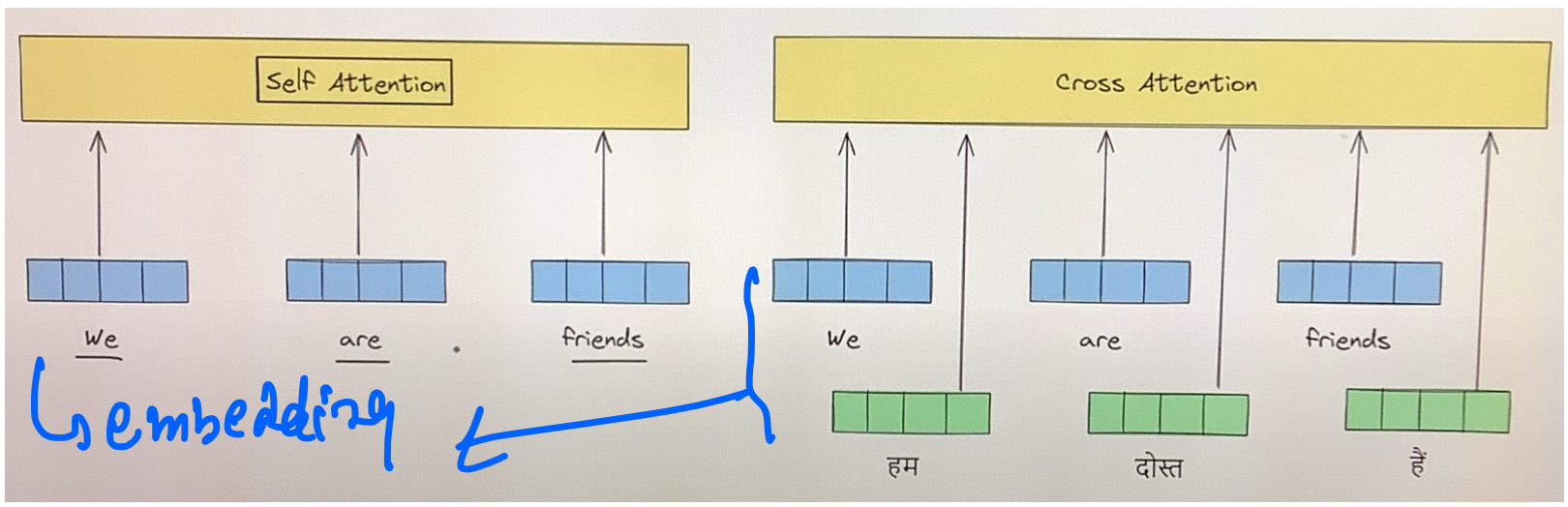
When we are looking for co-relation between "Two Different" sequences for:- →



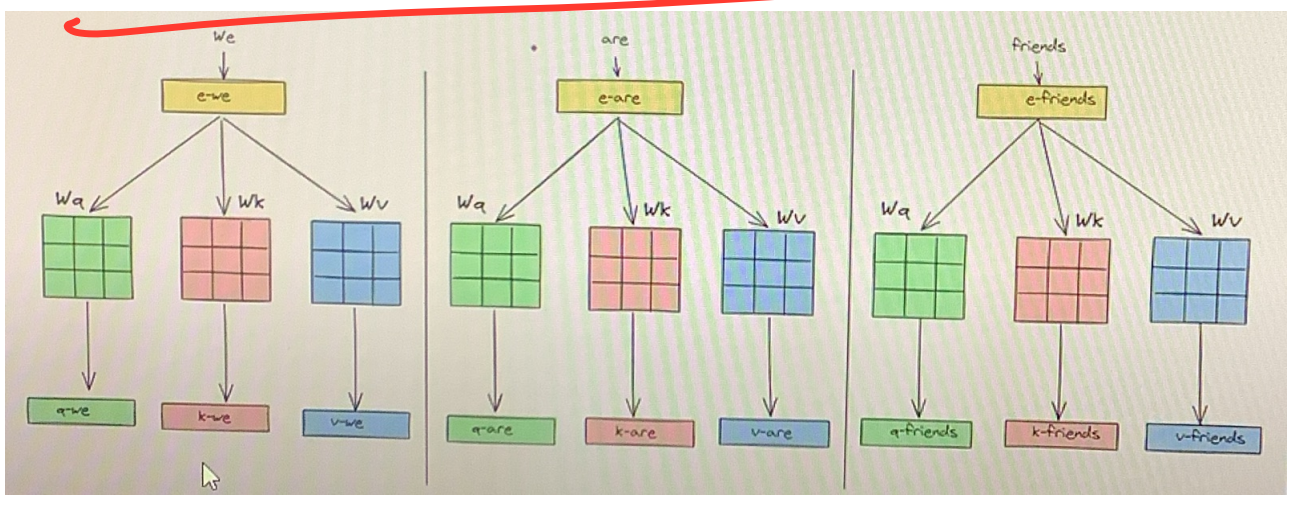
This mechani-sm known as "Cross Attention".

Cross attention is a mechanism used in transformer architectures, particularly in tasks involving sequence-to-sequence data like translation or summarization. It allows a model to focus on different part of an input sequence when generating output seq.

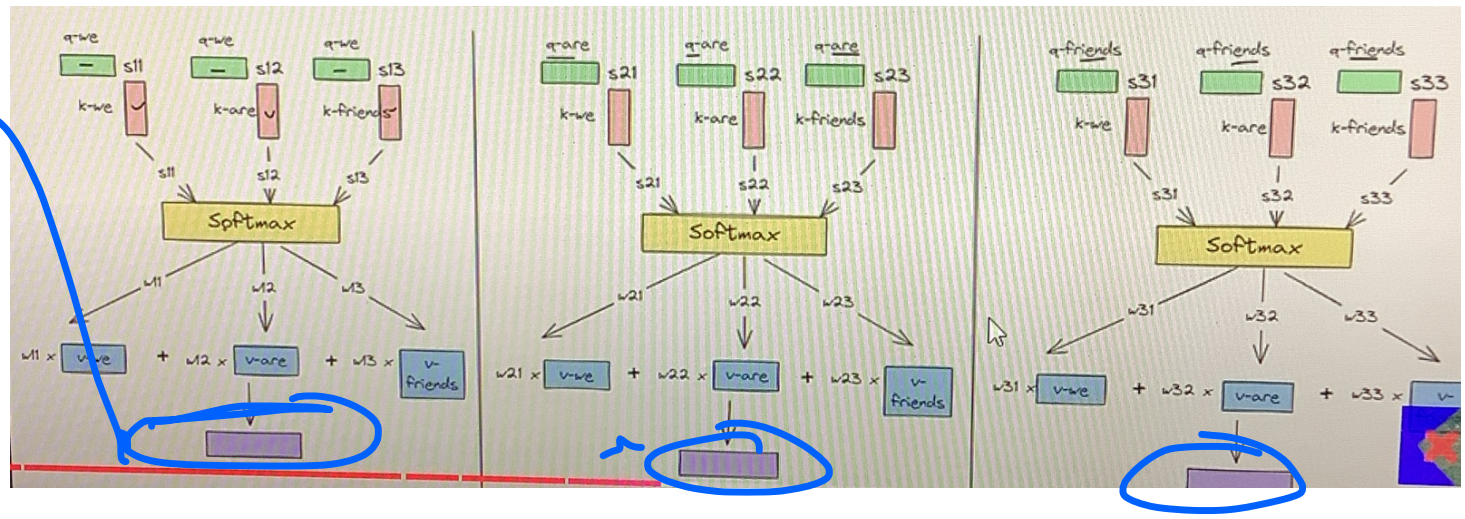
Self VS Cross Attention :- A high level view.



Self Attention Quick View :-



Context embedding

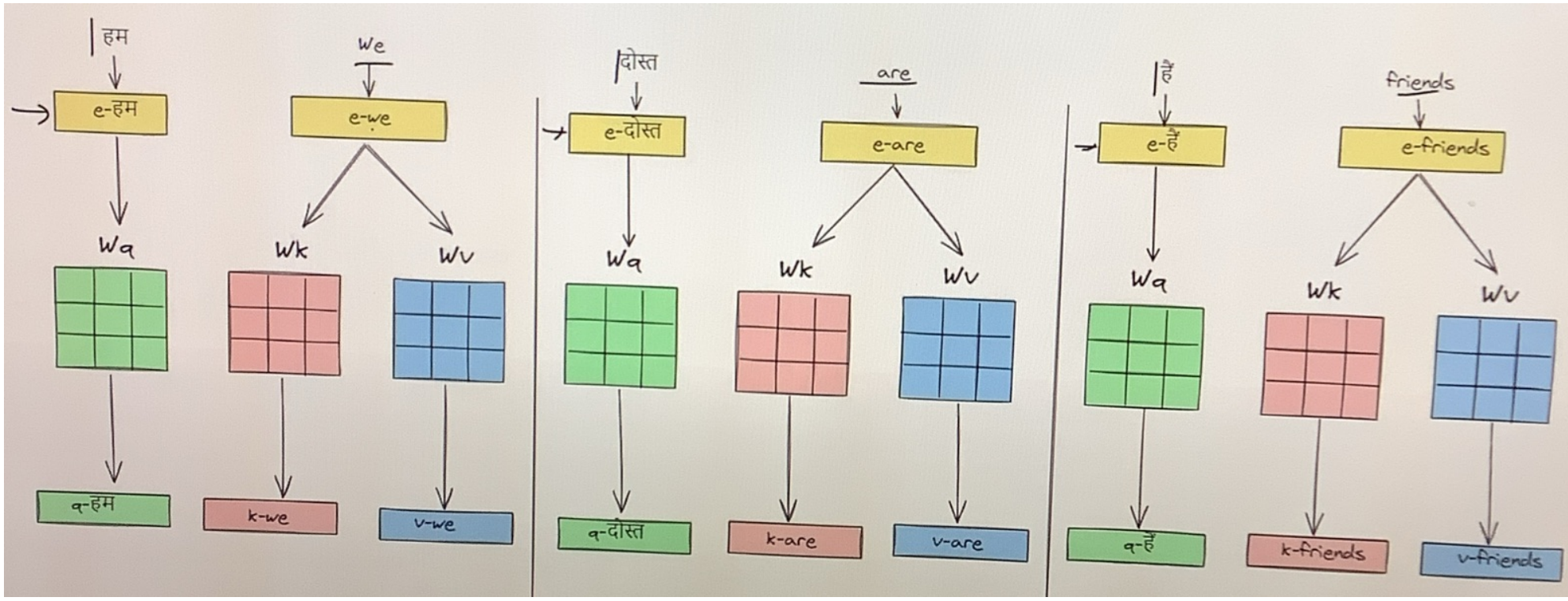


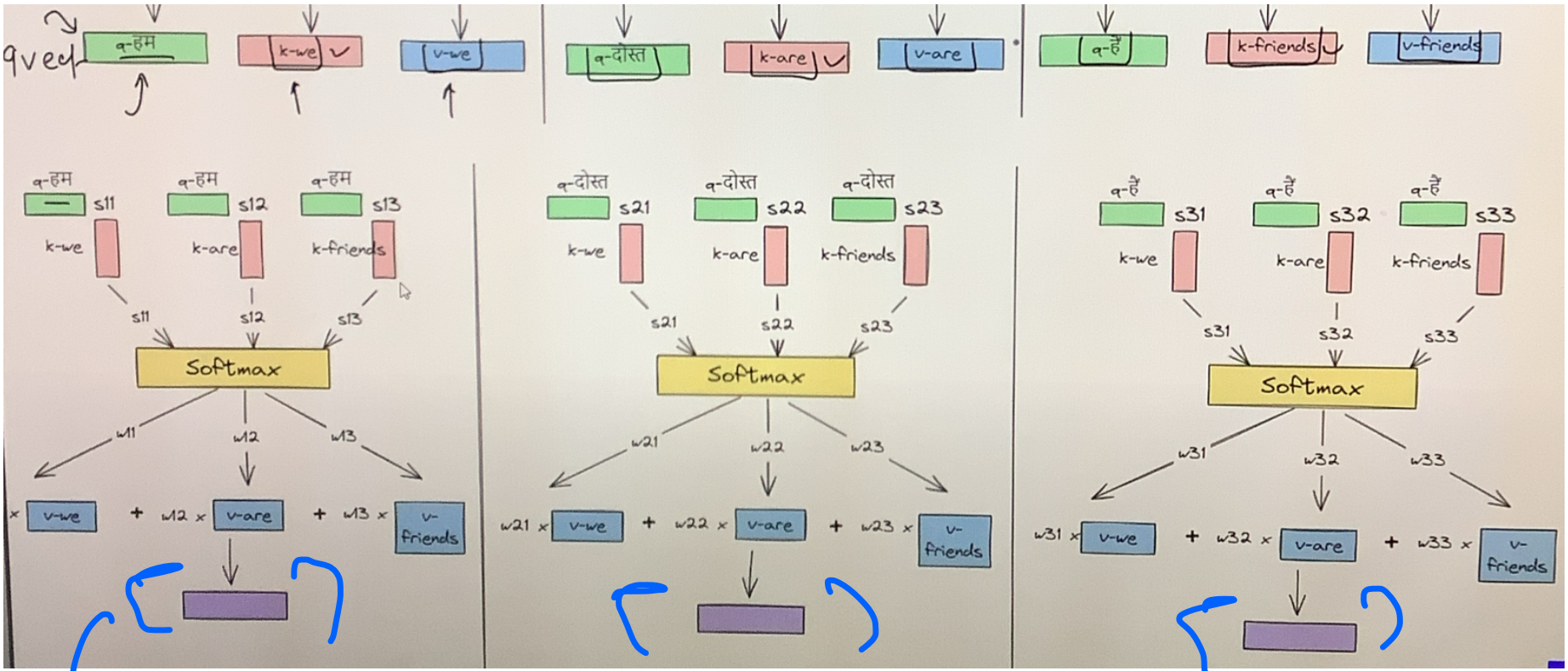
We are doing translation

English (Input) → Hindi (output)

⊗ query (come from output sequence)

Key & Value (Input sequence).





final context embedding

* Number of output from "Cross Attention"

is == No. of output sequence.

* Translation / * Multi-model

(Image captioning, text to speech)

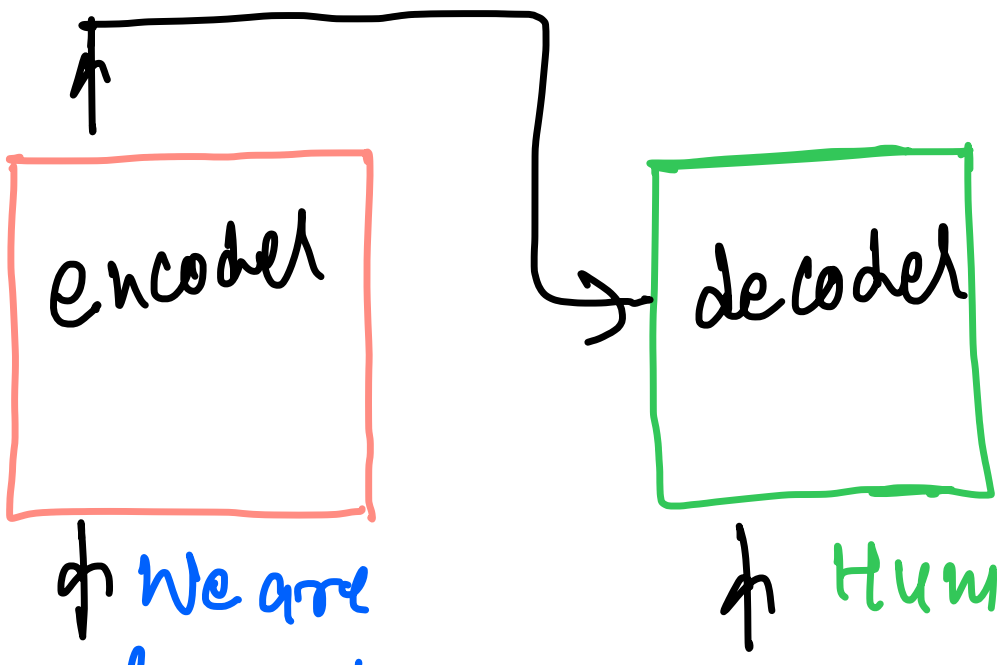
(text to image).

Decoder Architecture :-

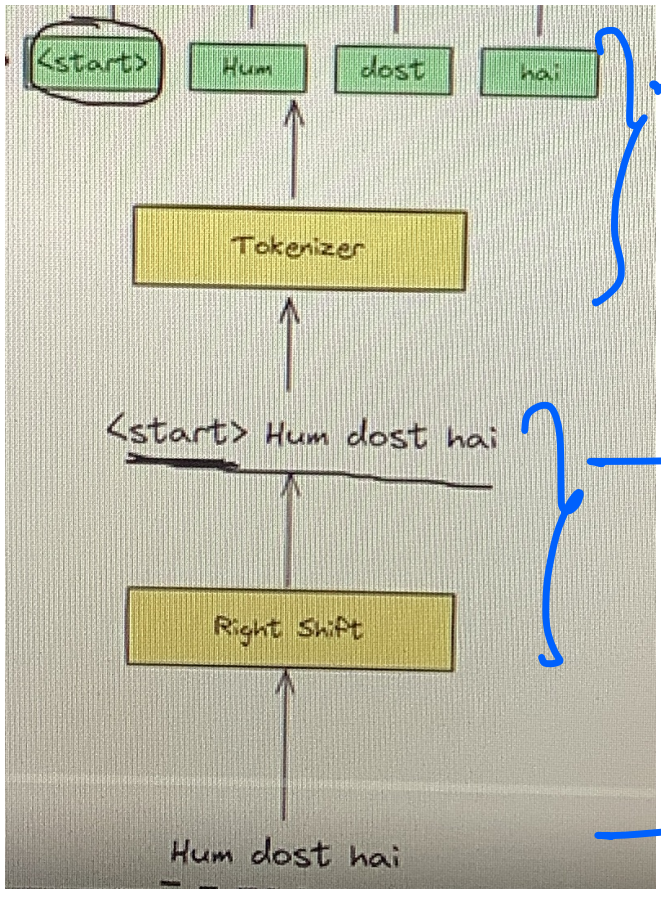
We are freinds

↓

Hum Dost Hai



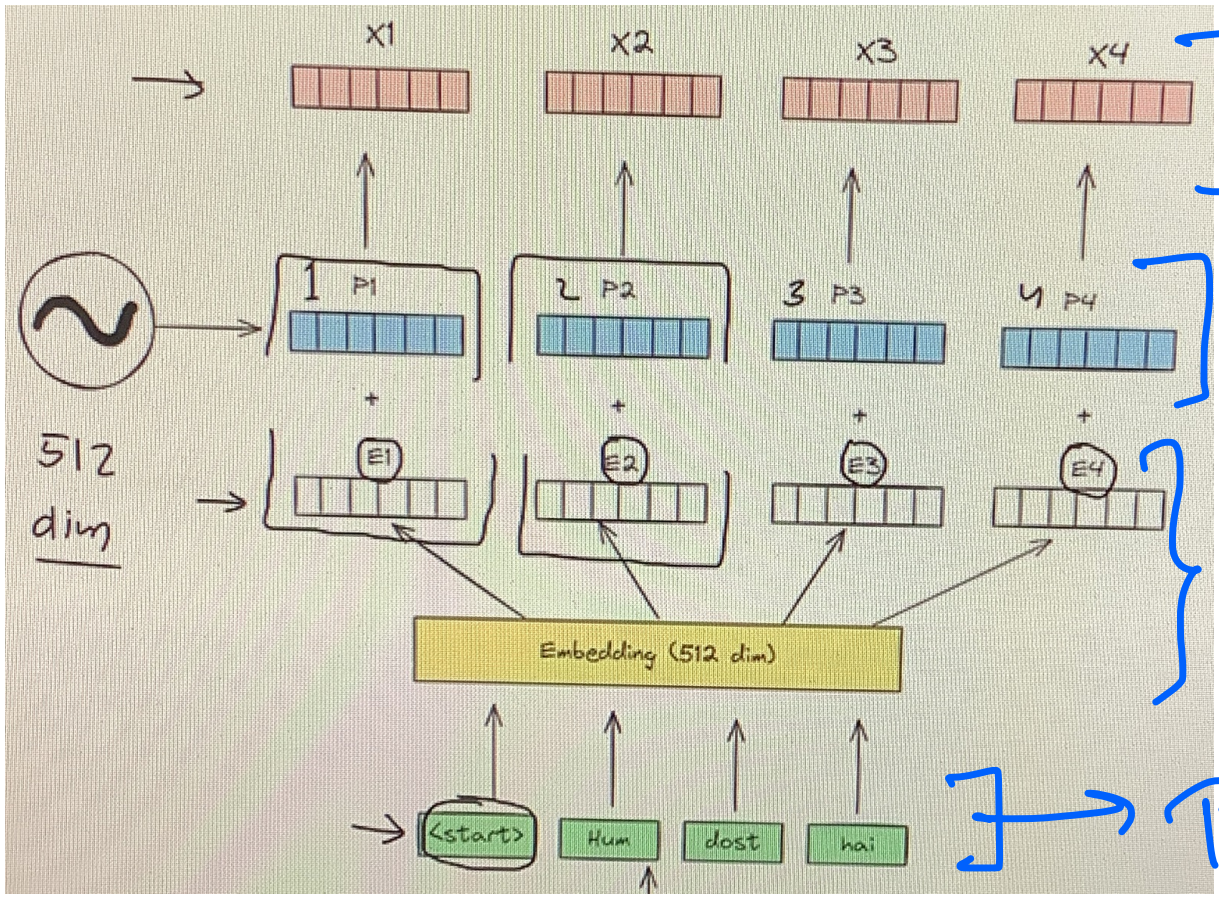
(Step 1)



→ Tokenization.

→ Right shift by append <start> token.

→ Output sequence



< Step 2 >

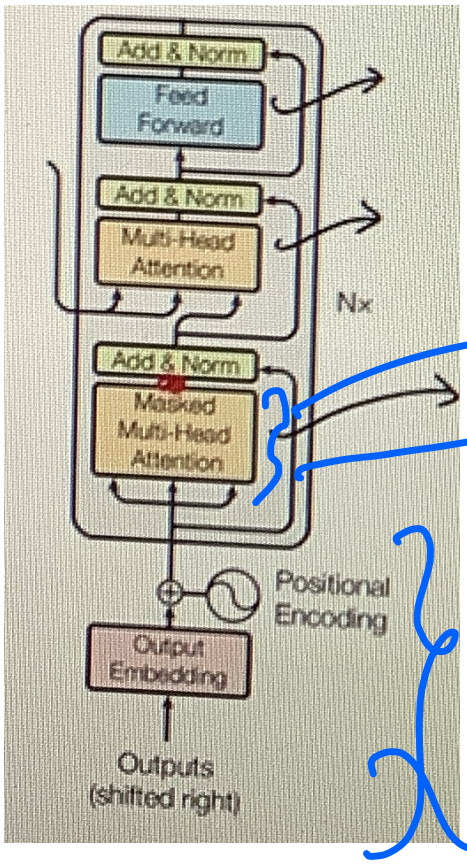
Input Vector for decoder.

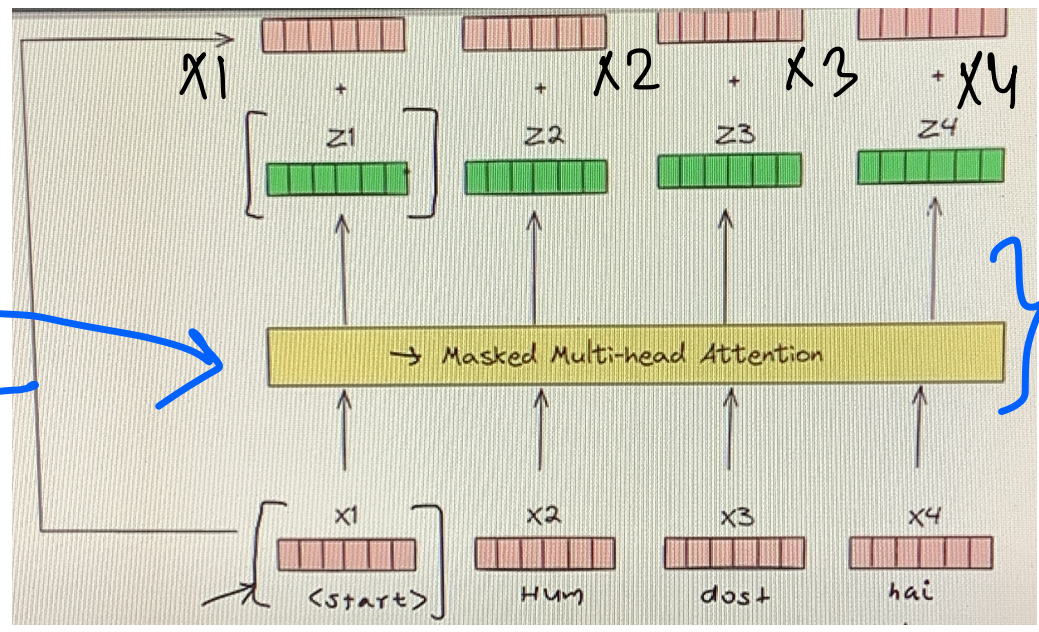
Positional Encoding

embedding

Tokens

< Step 3 >

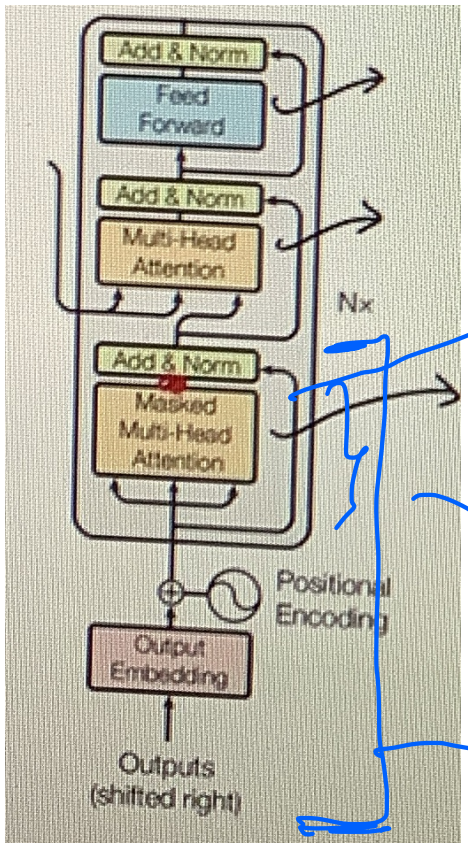




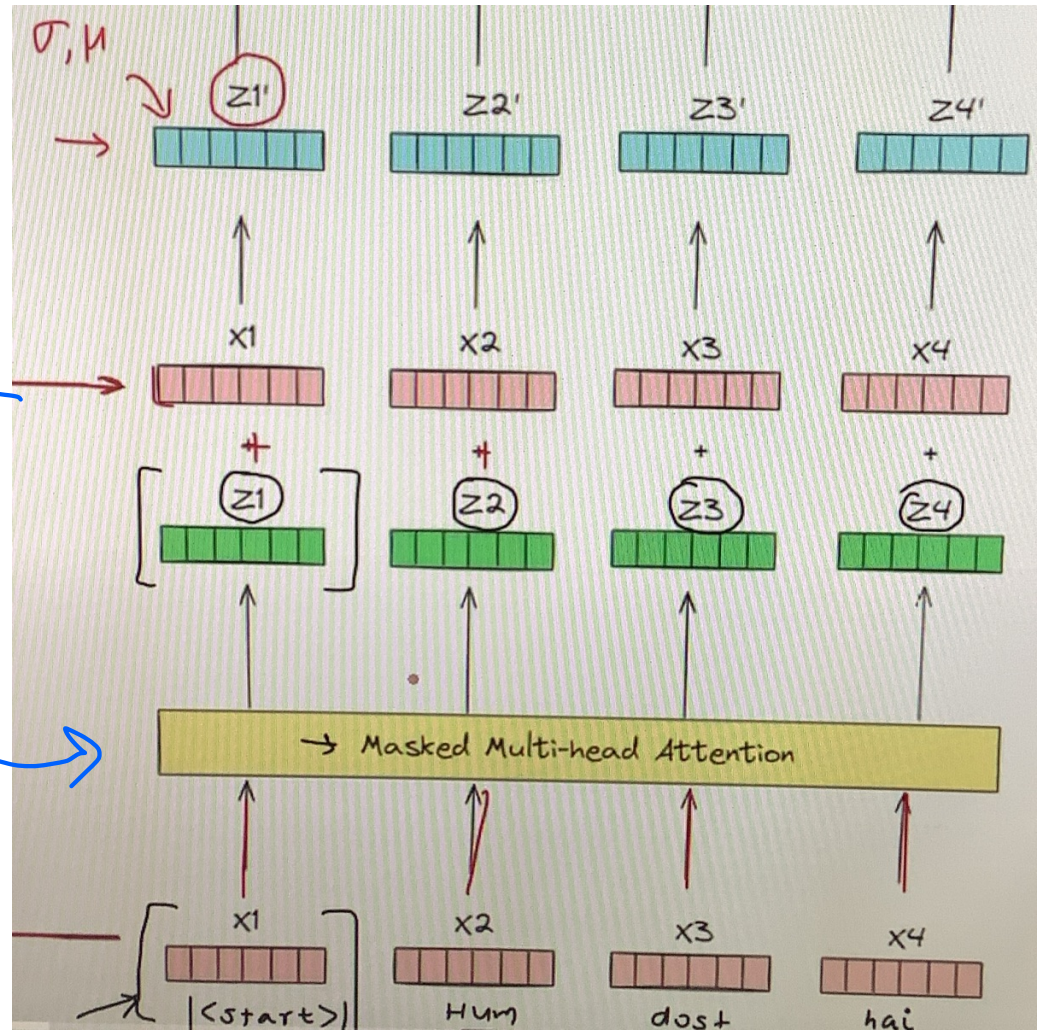
Add

Mask Multi Head.

Done in Step1 & Step2

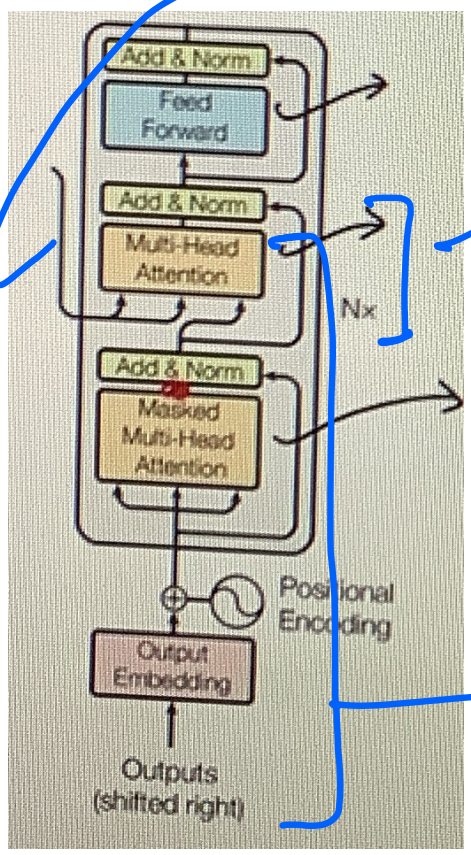


Add + Norm

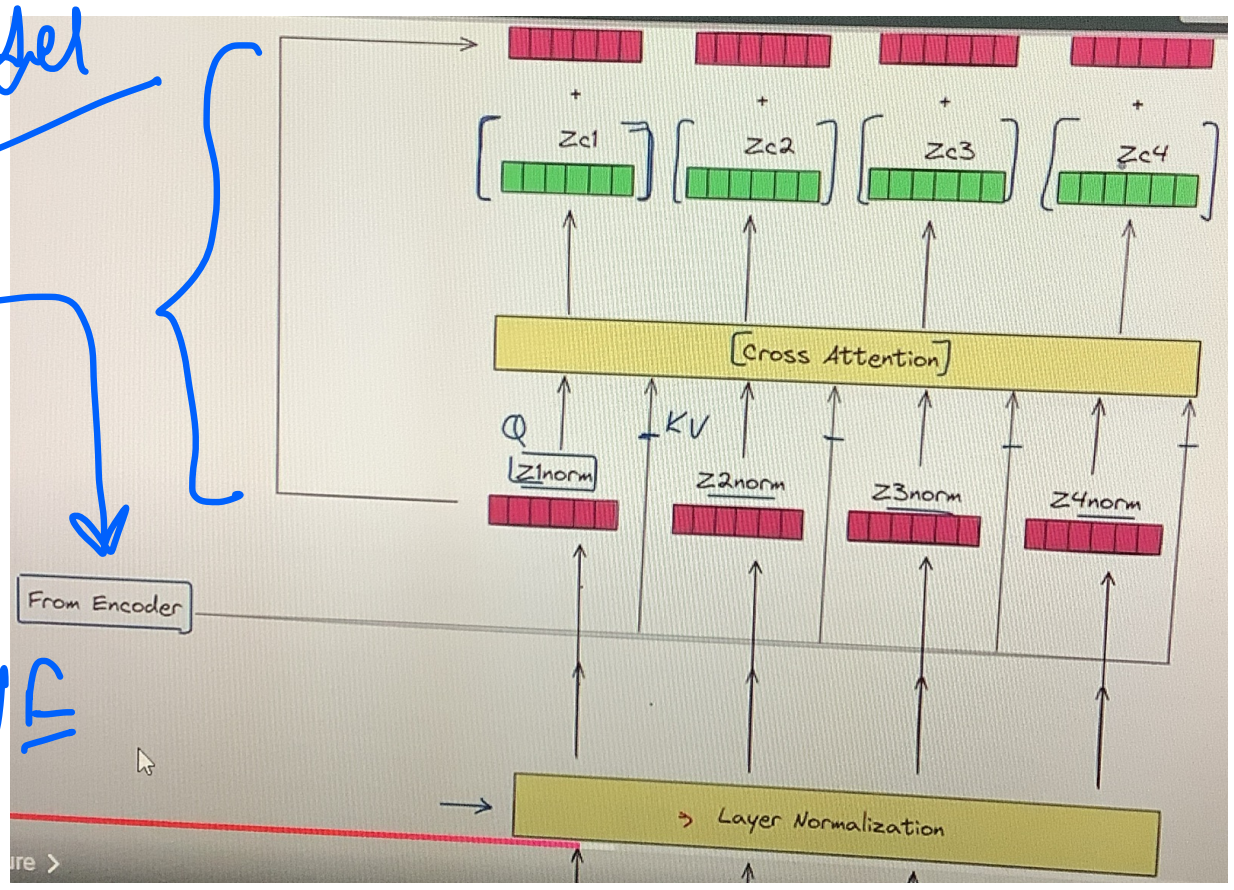


DONE

encoded input

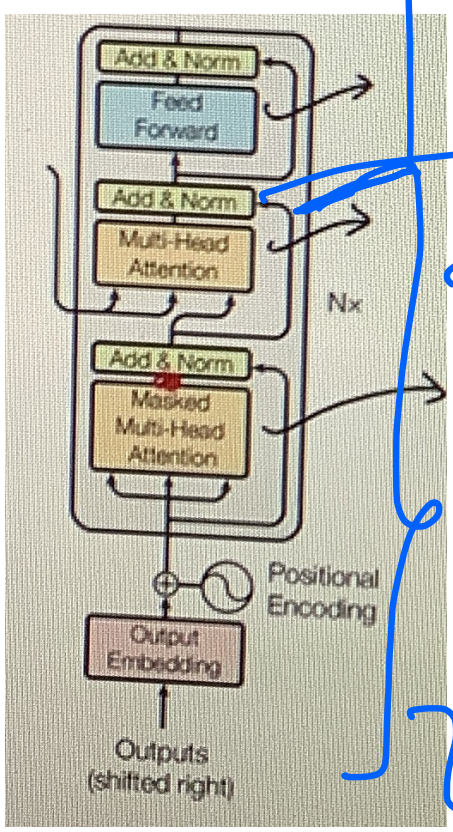


DONE

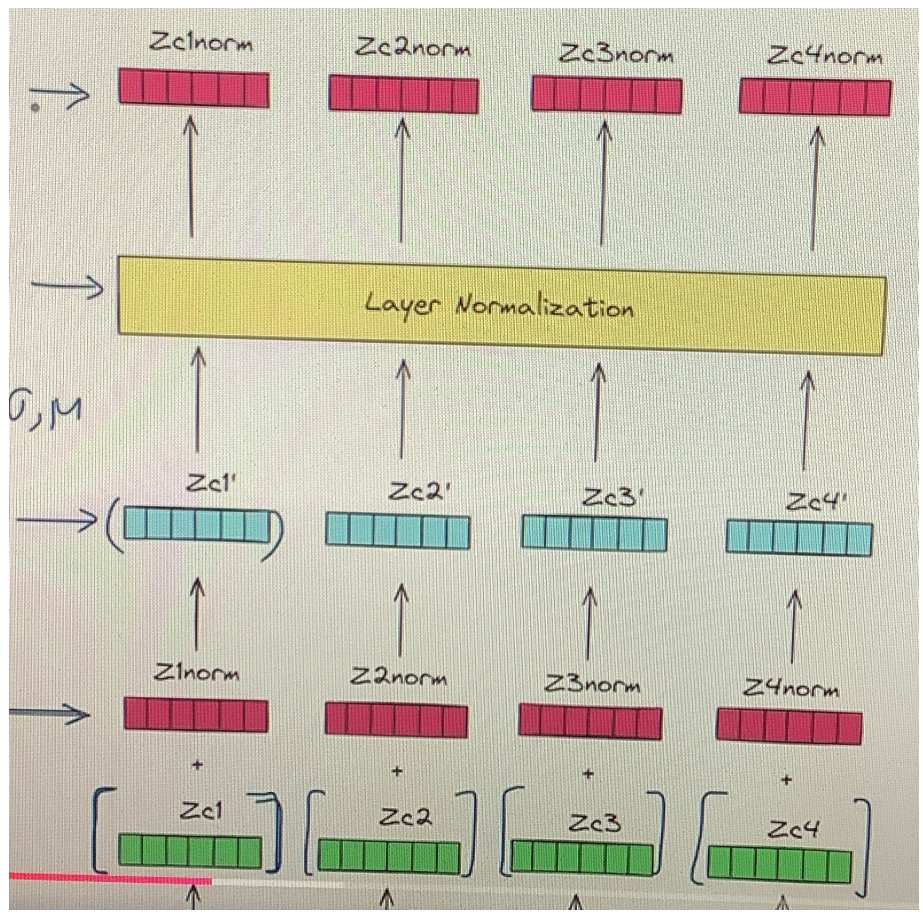


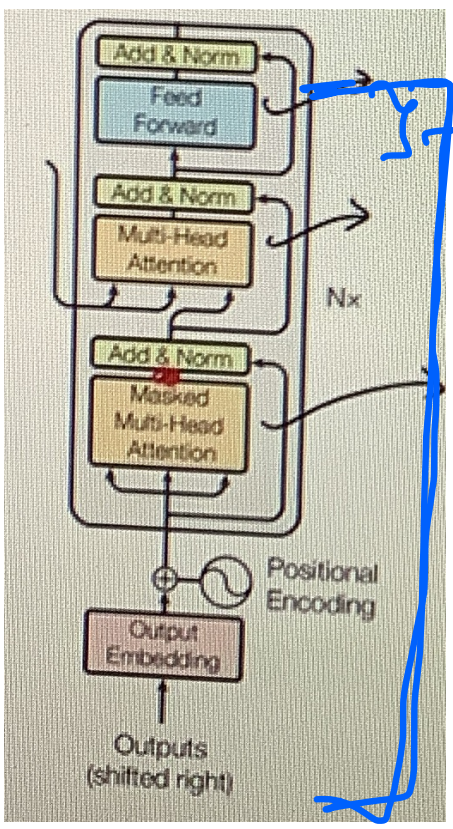
Output sequence will make query
K & V will be from Encoder.

Now Add & Norm :=

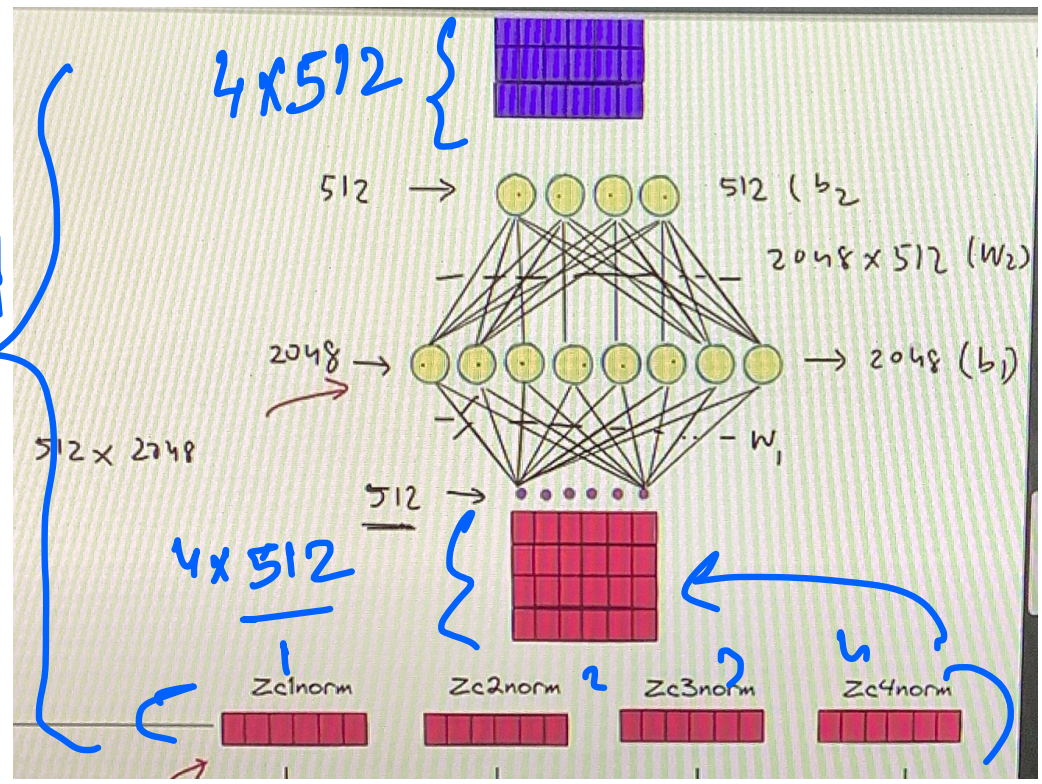


Add & Norm

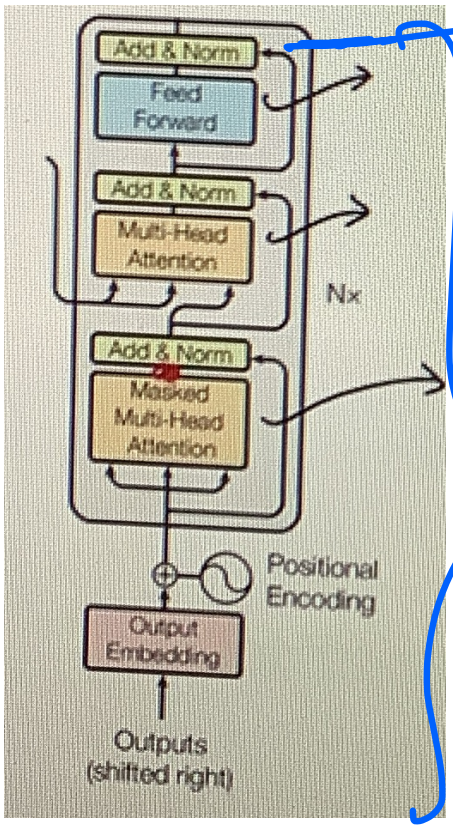




Feed Forward

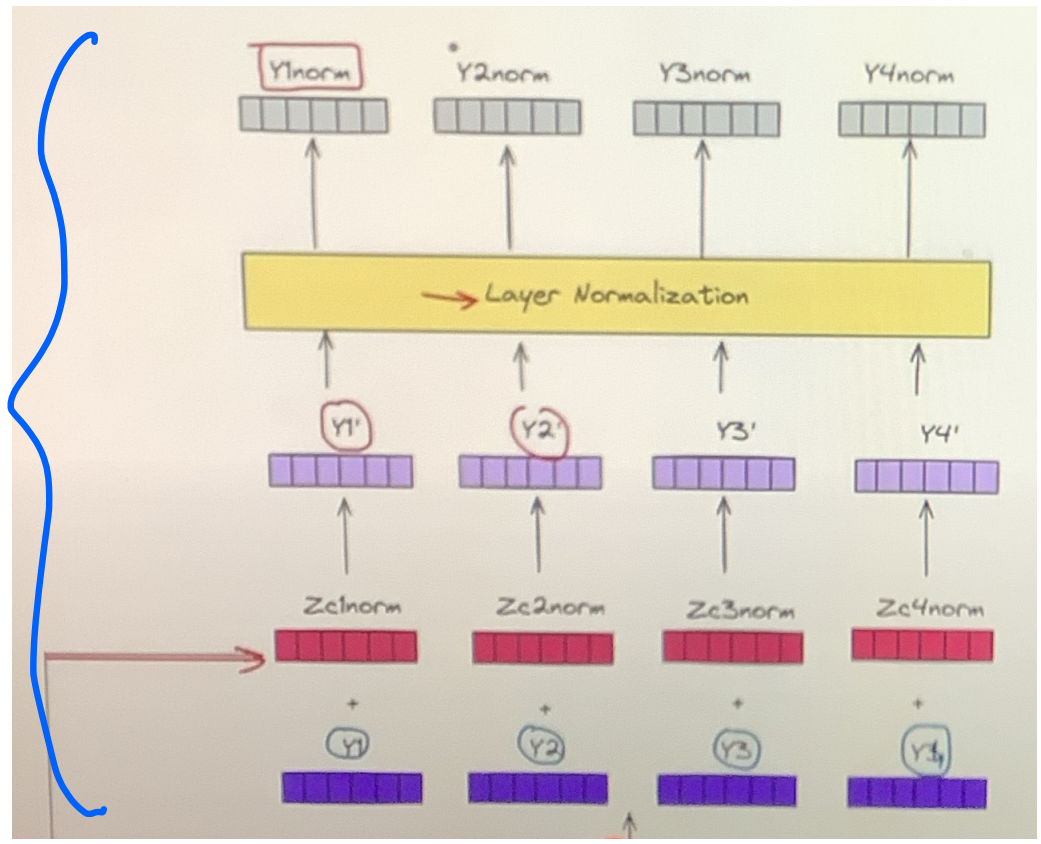


DONE



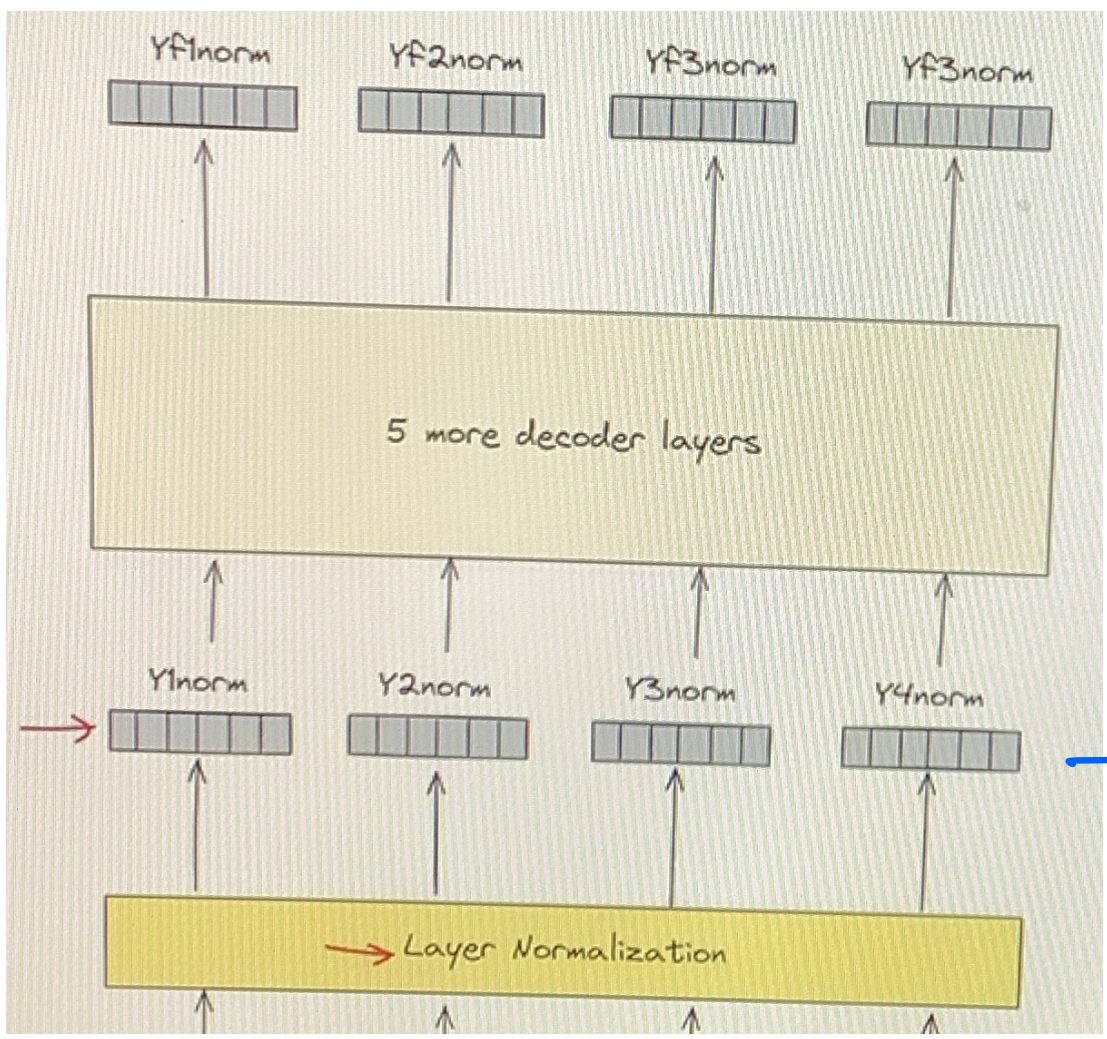
ADD & NORM

DONE



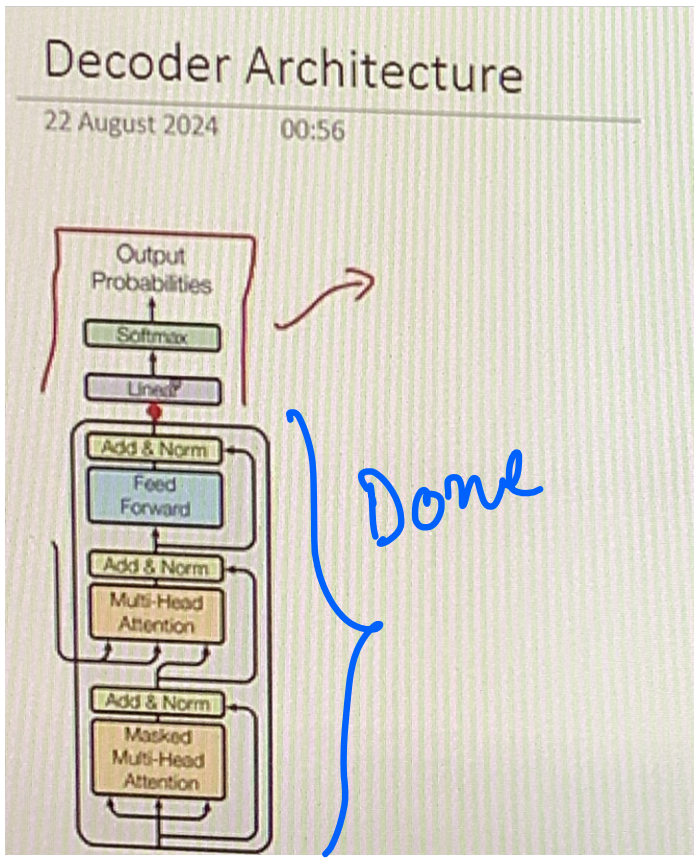
(Y1 norm, Y2 norm, Y3 norm & Y4 norm)

is Output from FIRST decoder.



Same operation will repeat.

→ output from first decoder



Now last Linear & Softmax.

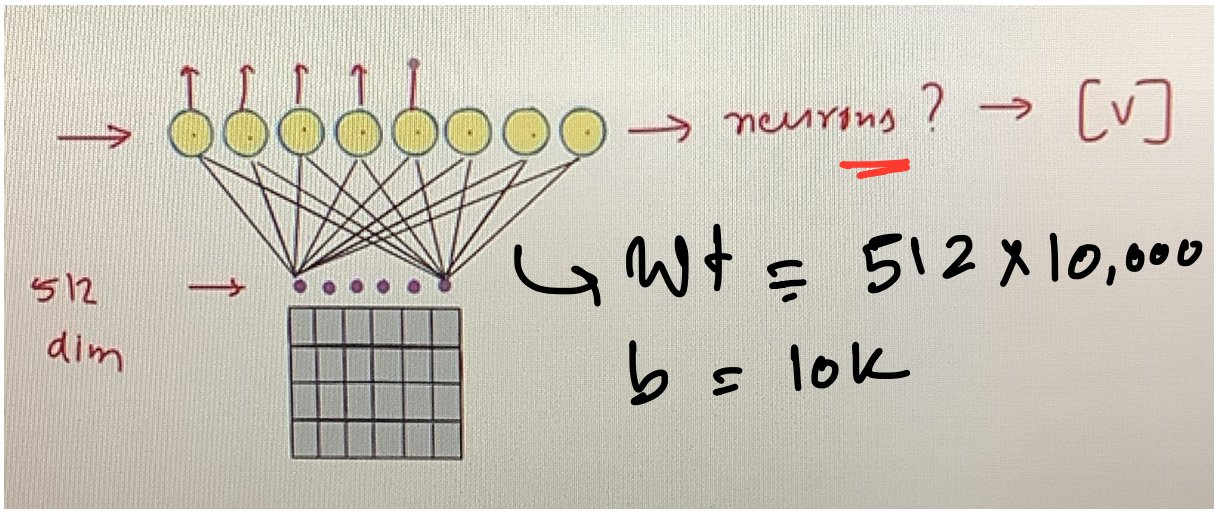
Dataset :-

I am good	Mai badhiya hoon
:	:

5000 samples.

We count unique words in Output seq.

say in 5K hindi samples we have 10K hindi words. So the Linear layer will be having 10K neurons.



V = No. of neurons

≒ 10K.

each index of neurons represent a word.

Transformer During Inference :-