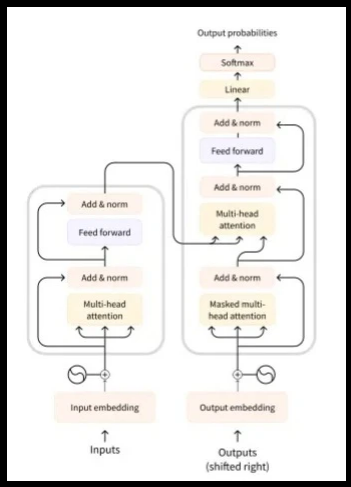
Transformers

- ANN → tubular data
- CNN → Image data
- RNN → Sequential data
- Transformers → Sequence to sequence task (i/p & o/p is sequence)
  - → Machine translation
  - → Q&A system
  - → Text summarization

Introduced in 2017



→ In transformers, we uses Self-attention. that's why it can handle massive amount of data parallely.
→ Hence, transformers are a neural network architecture to handle sequence-to-sequence data.
→ Transformer have encoder-decoder, but instead of LSTM it uses self-attention mechanism.
→ Due to parallel data processing, training becomes faster. so it is usefull for large-datasets.

Impact of transformers

- Revolution in NLP
- Democratizing AI
- Multi-model capability
- Acceleration of Gen AI
- Unification of Deep-learning

Advantages

- → Scalability
- → Transfer learning
- → Multi-model I/O
- → Flexible Architecture
- → Eco-system
- → Integrated AI Techniques
  - → GAN + Transformers = Image generator
  - → Reinforcement learning + Transformers = Game playing agent.
  - → CNN + Transformers = Image generator / Visual (Vision Transformer)

Disadvantages

- High computational infrastructure
- Data
- Energy
- Overfitting
- Bias

Future

- Improvement in efficiency
  - pruning
  - Quantization
  - knowledge distillation
- Multi-modal capability
  - Text
  - Image / Speech
  - Sensory data
  - Biometric feed back
  - Time Series
- Responsible
  - Bias elimination
  - Focusing on ethical concerns.
- Domain Specific
- Multi-lingual (Multi-language Support)
- Introperability
  - Difficult to understand internal process.
  - Black Box

Self-attention Mechanism

What is self attention....?

-> In NLP applications, the most important part is vectorization (words-to-number conversation), like how you convert words into numbers.

like:

OneHot Encoding ⟶ 1. cat mat rat
2. mat mat rat

	cat	mat	rat
cat	1	0	0
mat	0	1	0
rat	0	0	1

Inefficient

Bag Of Words ⟶

	mat	cat	rat
Sentence-1	1	1	1
Sentence-2	2	0	1

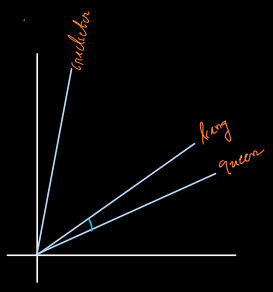
TF-IDF (Term frequency - Inverse Document Frequency)

Word Embeddings → Capability of storing Semantic Meaning.

Process:

1. Get training data

Hypothetical graph of 2D.



2. Pass it to neural network & it will convert it into n-dimen-tional vector space

Ex.

king → [0.81, 0.38, 0.96, 0.67, 0.49]

queen → [0.59, 0.39, 0.91, 0.66, 0.49]

cricketer → [0.1, 0.9, 0.78, 0.60, 0.39]

- Similar words will have very similar vectors.
- Based on data, each number needs some aspects.
- Word-embedding captures average meaning. like "On an average with which meaning, that particular word used in your data."

→ Now suppose we're generating embedding of a word "Apple" of 2D [x, y] where: x: Taste & y: Technology.

Example:

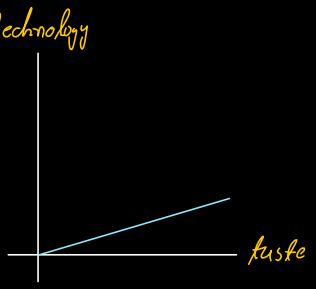
→ An apple a day keeps doctor away → [0.6, 0]

→ Apple is healthy. → [0.7, 0]

→ Apple better than orange → [0.8, 0]

→ Apple makes great phones. → [0.8, 0.2]

- Now suppose, if our dataset is of 10,000 lines. where 9000 rows, Apple is represented as fruit & in 1000 rows, Apple is represented as company.
- Now, obviously overall vector of Apple will looks like this [0.9, 0.3]. because the average meaning of apple word in dataset is more biased towards taste.



- So word-embeddings are generated once & will be used multiple times.
- It means that they are static, which causes issue.

Sample Sentence.

Apple launched a new phone, while i was eating an orange.

- Currently, embedding of apple is [0.9, 0.3] which tells that whenever apple word is used, it always be in the context of a fruit.
- So there should be Context-based Embedding instead of Static Embedding.

- Value should get changed based on context.

like

Apple launched a new phone, while i was eating an orange.

[0.3, 0.4]

To solve this problem, we need to use Self-attention.

It can give you smartly converted contextual embedding from your word embedding.

Embeddings of each words of a sentence. → Self-attention Mechanism → Newly converted smart contextual embeddings of each word in given sentence.

Behind the scene of Self-attention

Sample text :

S1 = Money bank grows , S2 = River bank flows

Now let's distinguish the word "river" based on their context

for S1 : bank = 0.3 money + 0.7 bank + 0.1 grows

for S2 : bank = 0.5 river + 0.4 bank + 0.1 flows

S1

money = 0.7 money + 0.2 bank + 0.1 grows

bank = 0.25 money + 0.7 bank + 0.05 grows

grows = 0.2 money + 0.2 bank + 0.7 grows

S2

river = 0.8 river + 0.15 bank + 0.05 flows

bank = 0.2 river + 0.78 bank + 0.02 flows

flows = 0.4 river + 0.01 bank + 0.59 flows

Mathematical notation

$$e_{money}^{(new)} = 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows}$$

$$e_{bank}^{(new)} = 0.25\, e_{money} + 0.7\, e_{bank} + 0.05\, e_{grows}$$

$$e_{grows}^{(new)} = 0.2\, e_{money} + 0.2\, e_{bank} + 0.7\, e_{grows}$$

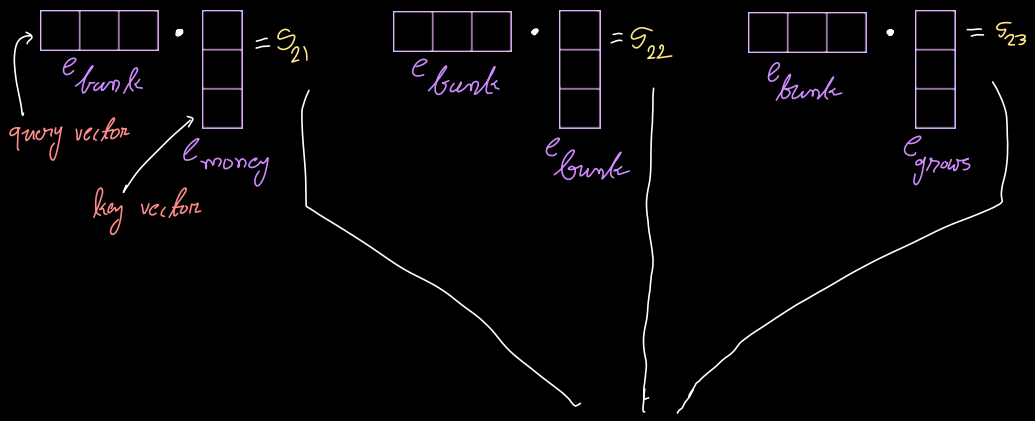
$$e_{river}^{(new)} = 0.8\, e_{river} + 0.15\, e_{bank} + 0.05\, e_{flows}$$

$$e_{bank}^{(new)} = 0.2\, e_{river} + 0.78\, e_{bank} + 0.02\, e_{flows}$$

$$e_{flows}^{(new)} = 0.4\, e_{river} + 0.01\, e_{bank} + 0.59\, e_{flows}$$

More depth

$$e_{bank}^{(new)} = [e_{bank} \cdot e_{money}^{T}]\, e_{money} + [e_{bank} \cdot e_{bank}^{T}]\, e_{bank} + [e_{bank} \cdot e_{grows}^{T}]\, e_{grows}$$



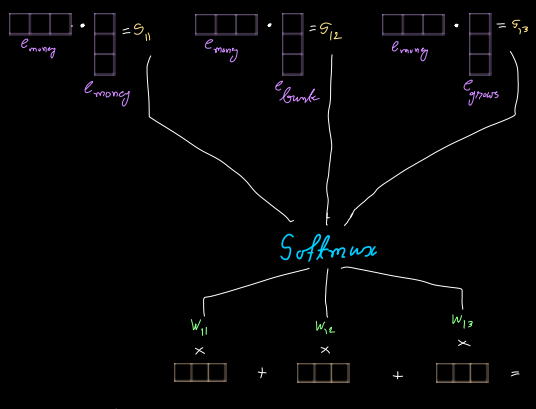
Softmax

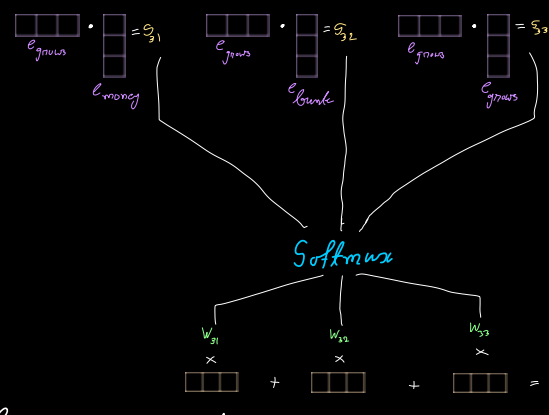
$$W_{21} = \frac{e^{S_{21}}}{e^{S_{21}} + e^{S_{22}} + e^{S_{23}}} \qquad W_{22} = \frac{e^{S_{22}}}{e^{S_{21}} + e^{S_{22}} + e^{S_{23}}} \qquad W_{23} = \frac{e^{S_{23}}}{e^{S_{21}} + e^{S_{22}} + e^{S_{23}}}$$

Value Vector → × + × + × = $e_{bank}^{(new)}$

contextual word embedding

→ With this same process we will calculate Y_{money} and Y_{grows}.

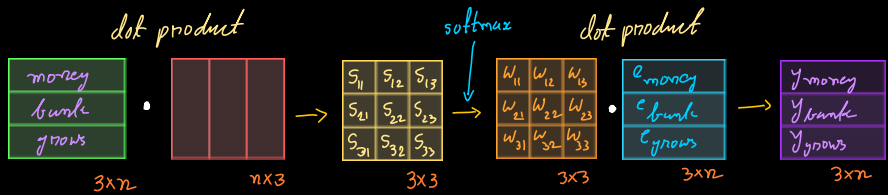




→ Now, for sentence-2 Money bank grows we have unique contextual embeddings of each words

Consider....!

- Parallel calculation process
- Due to parallel computation, we're loosing sequence information.
- No learnable parameters.

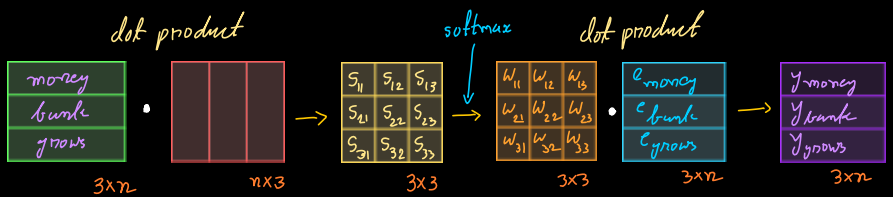


- Newly generated embeddings are general contextual embeddings, not task specific contextual embedding which is a big problem.

→ So, it is not sufficient to generate only contextual embeddings, we also need task specific contextual embeddings.
(embedding of a word based on it's surrounded words).

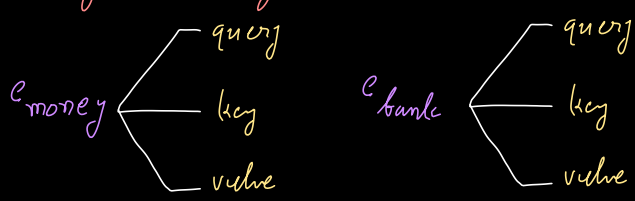
Ex.
Sentence
Piece of cake
meaning-1: केक का टुकड़ा
meaning-2: बहुत आसानी वाला

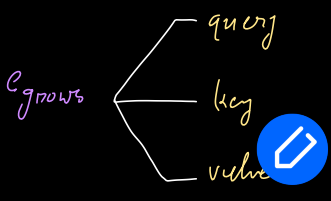
→ In short we need to add trainable parameters in calculation process.



→ Our current process, there are 2 possible locations where we can set-up trainable parameters which is dot-product operation.

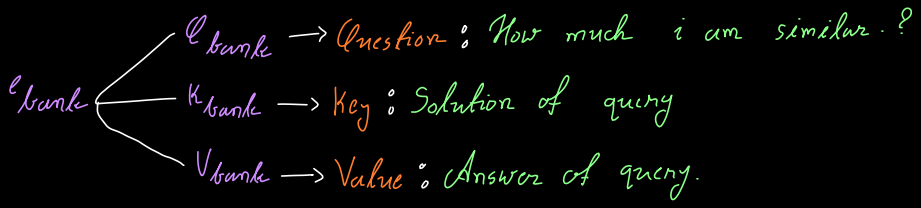
→ Now, here e_{money} or e_{bank} or e_{grows} they're individually performs their different role. as query vector, key vector & value vector





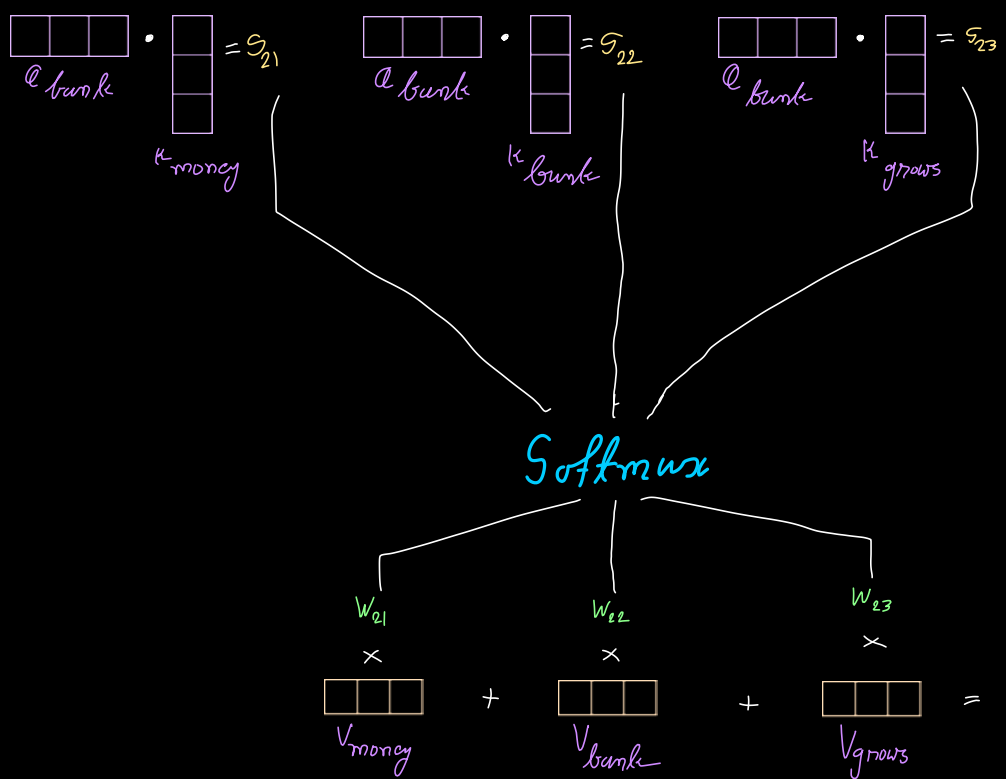
→ Different role as query, key & value vector of a particular word (e_{money} | e_{bank} | e_{grows}) is same here which is not recommended. Because those query, key & value vector must be different for their respective task.

→ So we will create different vectors based on their task such as...



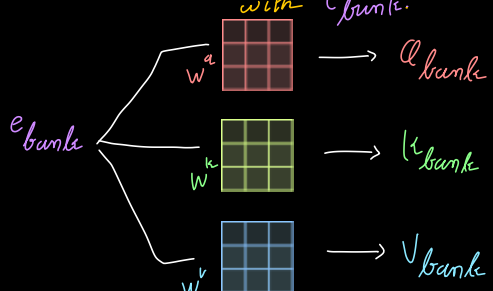
These 3 Query, Key & Value Vectors will be made by data.

→ After as we got these data, now we will send Query Vectors of particular word instead of its embedding.



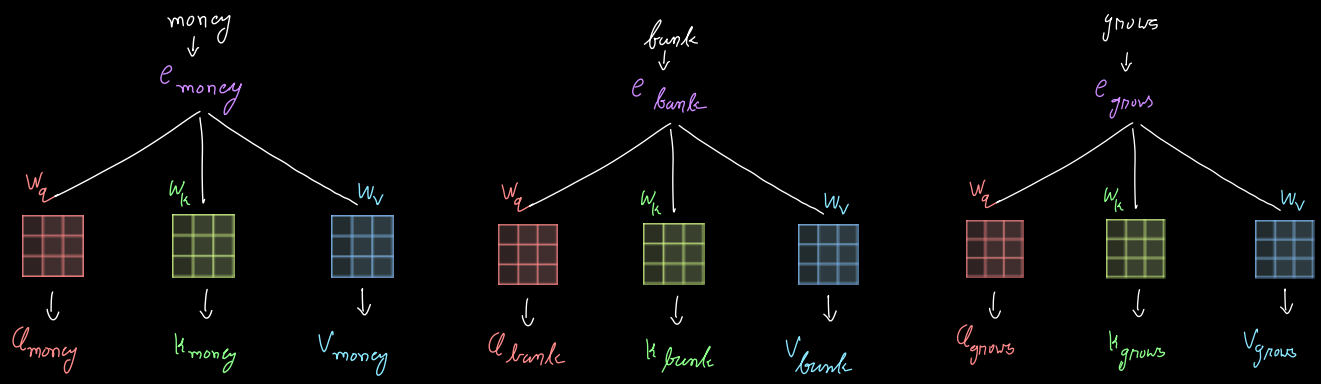
Q. How to generate these new vectors from embeddings...?

matrix multiplication of different matrix with e_{bank}.

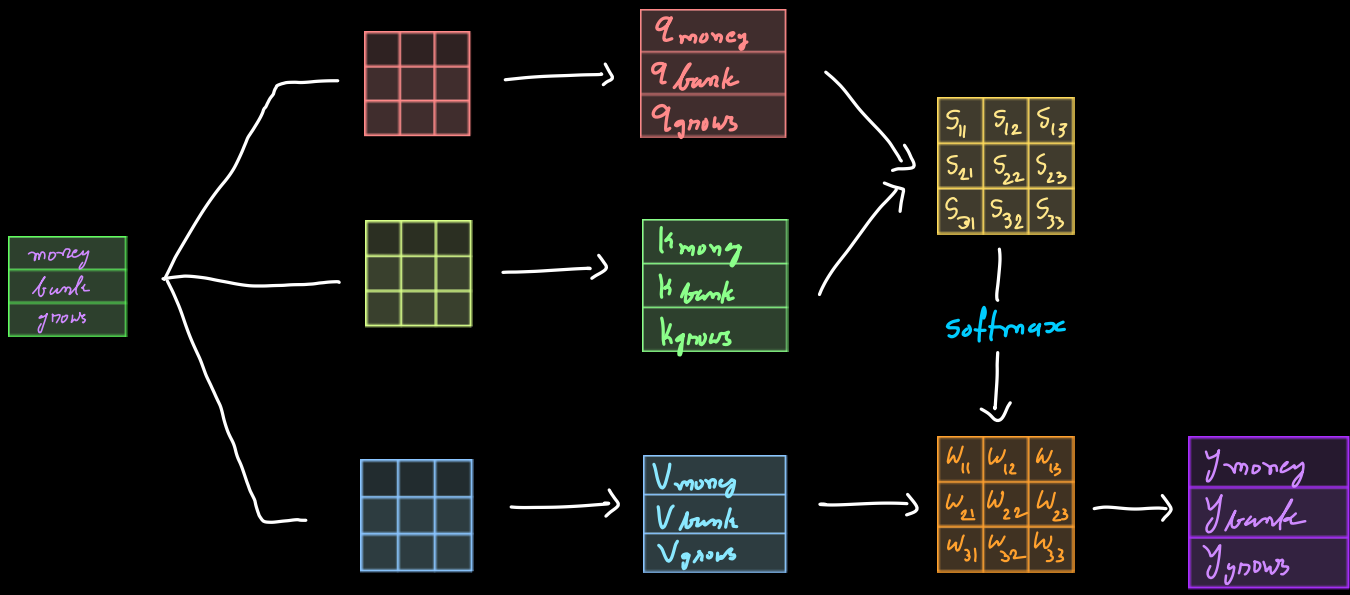


Now the value of all of these individual matrix will be generated by given data during training.

- This process will be repeated for each word in a sentence.



→ Here all W_q, W_k and W_v will be same for all words in given sentence.



This is self-attention mechanism.

In short.....

$$Attention(q, k, v) = (qk^T)v$$

Formula, mentioned in document....

$$Attention(q, k, v) = \left(\frac{qk^T}{\sqrt{d_k}}\right)v$$

→ This formula is called as **Scaled Dot Product**.

→ Here d_k = dimensionality of k (key) vector.

Q. Why scalling...?

A. Due to the nature of dot product.

Nature of dot-product : If we have low dimentional vector space than the dot-product of those vectors will give low variance.

- If we have high dimentional vector space than the dot-product of those vectors will give high variance
- high variance = big problem because softmax gives high probability to higher value & very small probability to small value. & that will create a gap between those numbers & due to this problem backpropagation will focus on larger numbers & small numbers gets ignore & gradients of those parameter faces vanishing gradient problem & that will not let those parameter update. so we need to reduce variance

→ We can divide each value of matrix with a constant one & that will cause low variance. (scalling factor)

number of dimentions & variance has linear relationship

in ideal senario, variance must remain constant even dimention increases.

A mathematical rule.

If you have a random variable X with a variance of Var(X), & you create a new variable Y by scalling X with a constant c, so that Y = cX, the variance of Y (Var(Y)) is related to variance of X by the square of scalling factor c. Mathematically this relationship is expressed as.

$$Var(Y) = c^2 Var(X)$$

X → Variance = Var(X)

Y → created by scalling X with constant c.

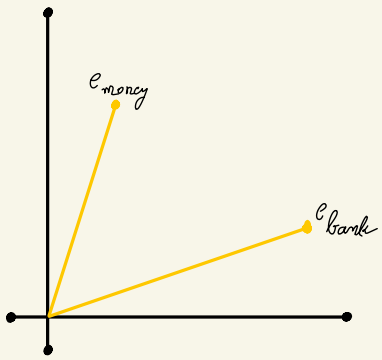
→ Hence, variance of Y = $c^2 Var(X)$

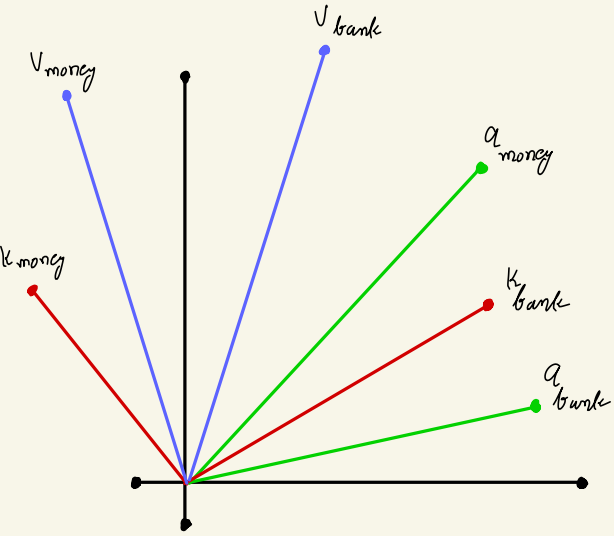
→ Now we need to divide it by such constant number which will always give Var(X) in o/p regardless of number of dimentions. hence we will divide our vector by $\frac{1}{\sqrt{d_k}}$.

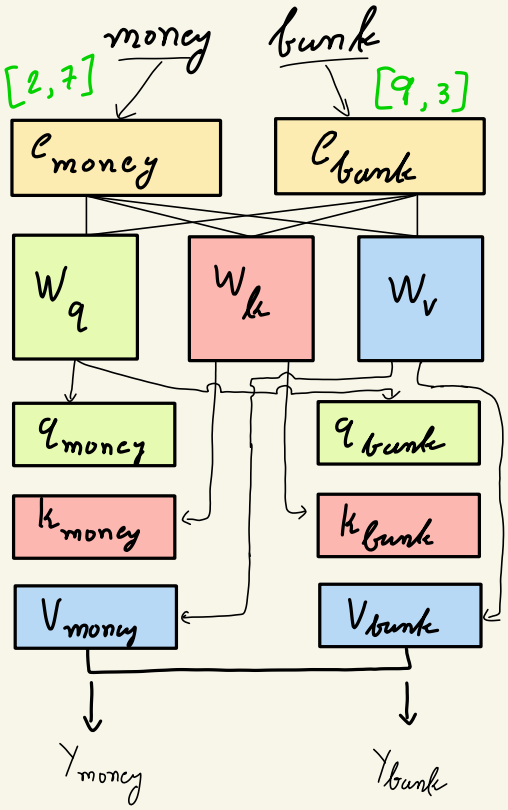
Geometric Intuition

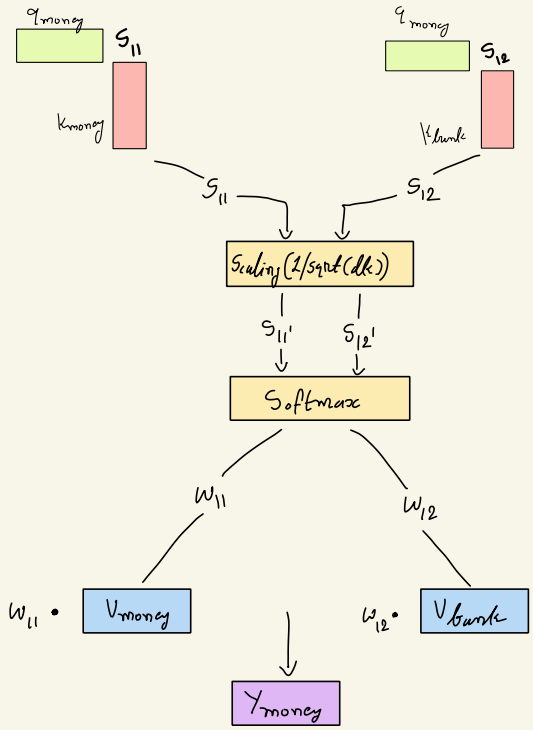
$$W_q = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad W_k = \begin{bmatrix} 3 & 4 \\ 5 & 1 \end{bmatrix} \quad W_v = \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix}$$

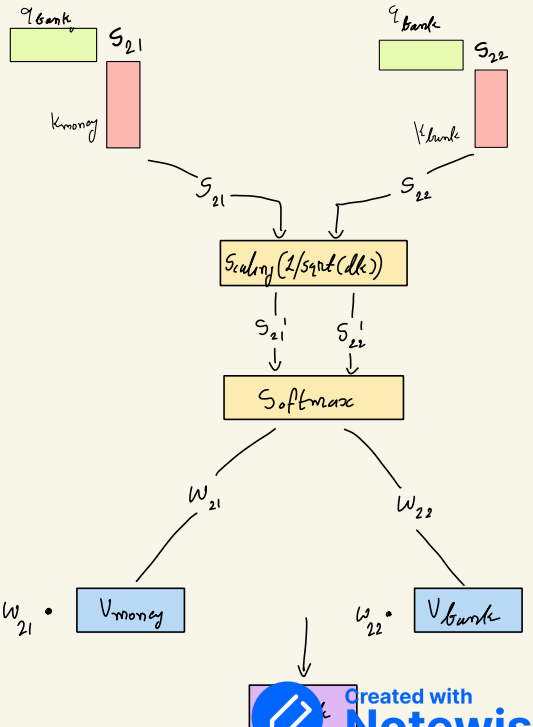
All these values are hypothetical.

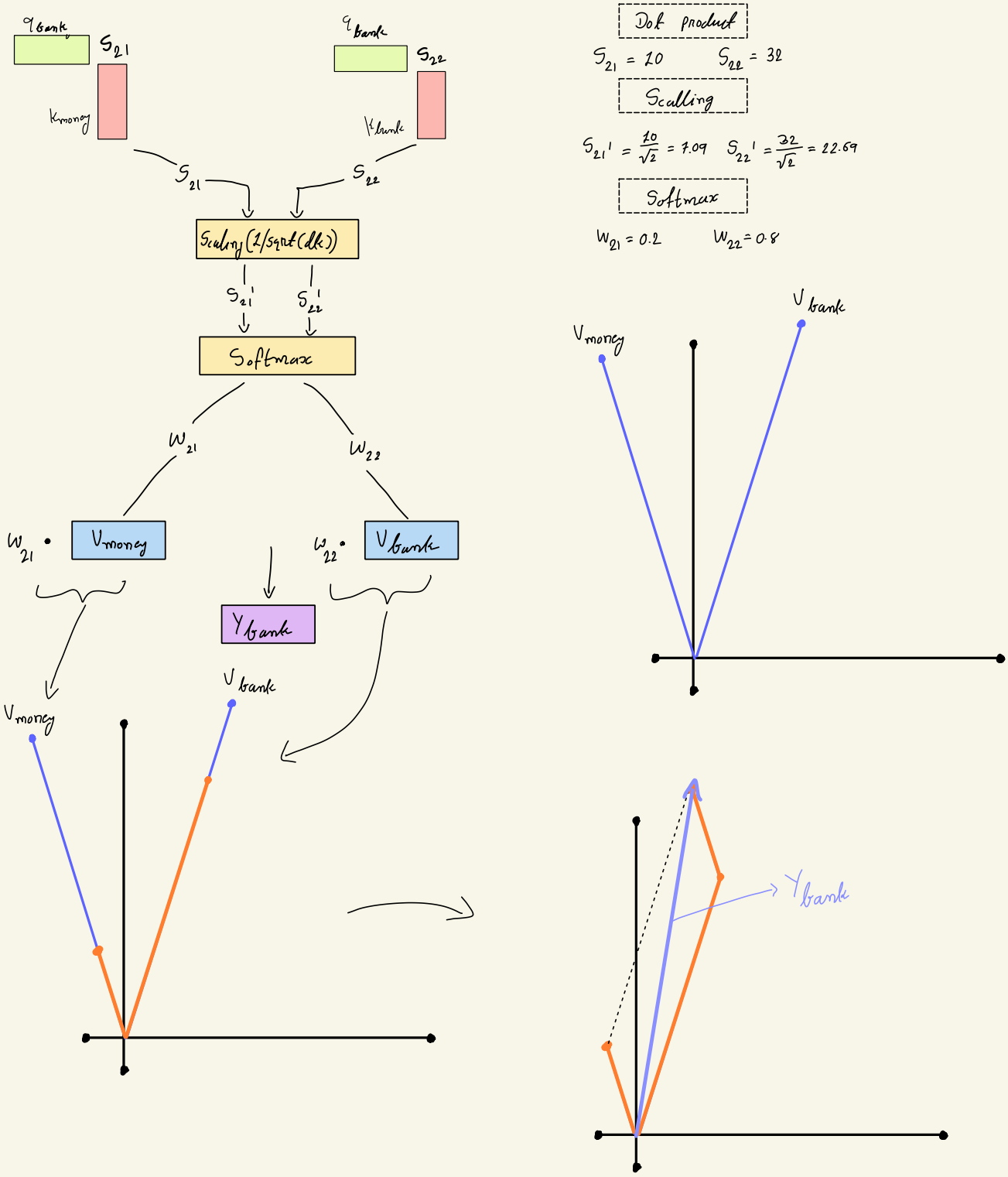






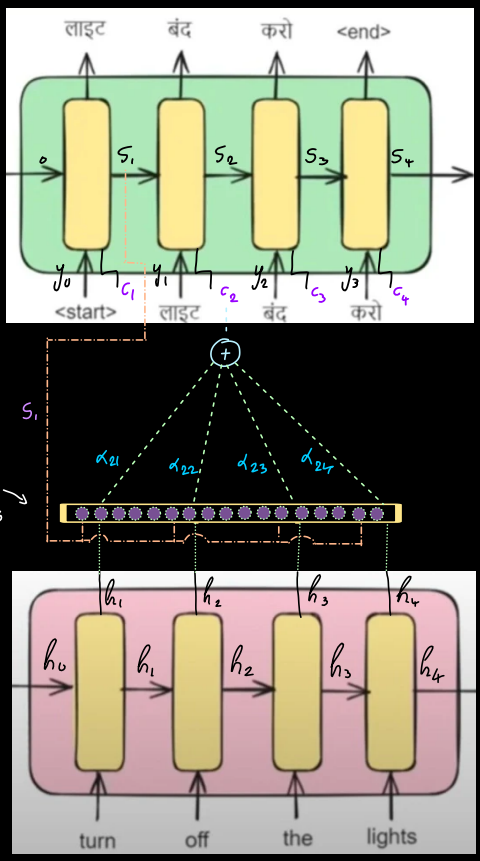




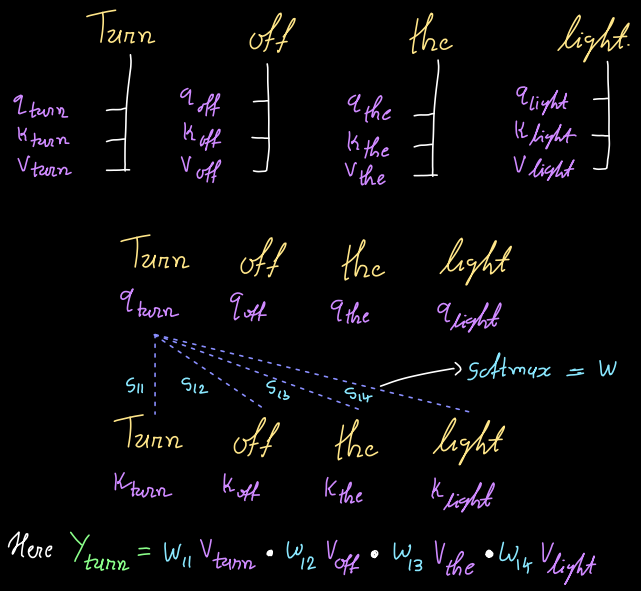


And that is the brilliance of self-attention.

Q. How is self-attention similar to Bahdanau & Luong Attention..?



Self-attention



Here $Y_{turn} = W_{11} V_{turn} \cdot W_{12} V_{off} \cdot W_{13} V_{the} \cdot W_{14} V_{light}$

Since we're calculating attention between same sequence, hence it is called as "Self-attention".

Disadvantage

→ Self-attention can only figure-out single perspective from given sentence.

Example

- The man saw the astronomer with a telescope.
  - ↳ Meaning-1 : A man saw an astronomer on his telescope.
  - ↳ Meaning-2 : A man saw an astronomer working with his telescope.

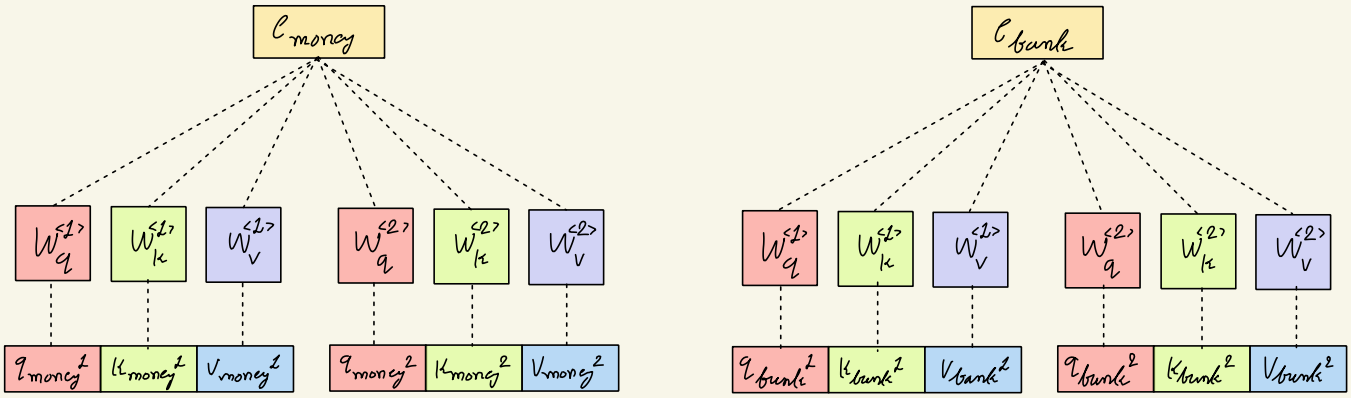
→ We want such mechanism which can figurecut maximum possible perspectives from given sentences / paragraph.

→ Hence, there is a new solution called as **Multi-head attention.**

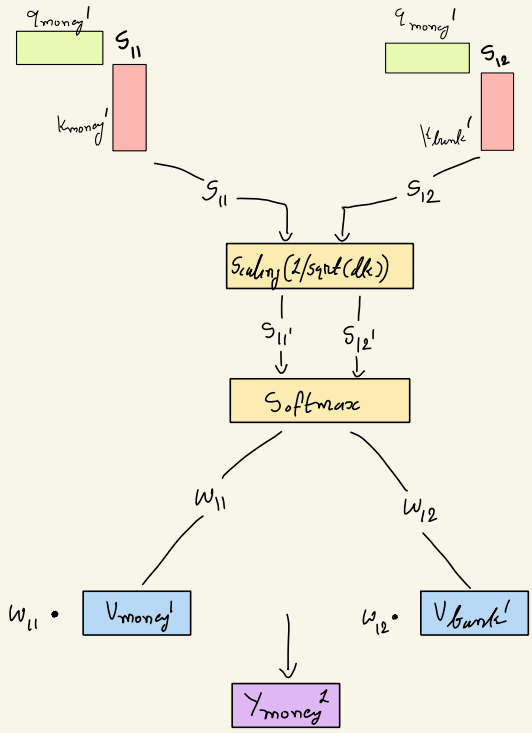
→ **Multi-head attention** gives simple solution use **multiple self-attention architecture.** it is called as **multi-head attention.**

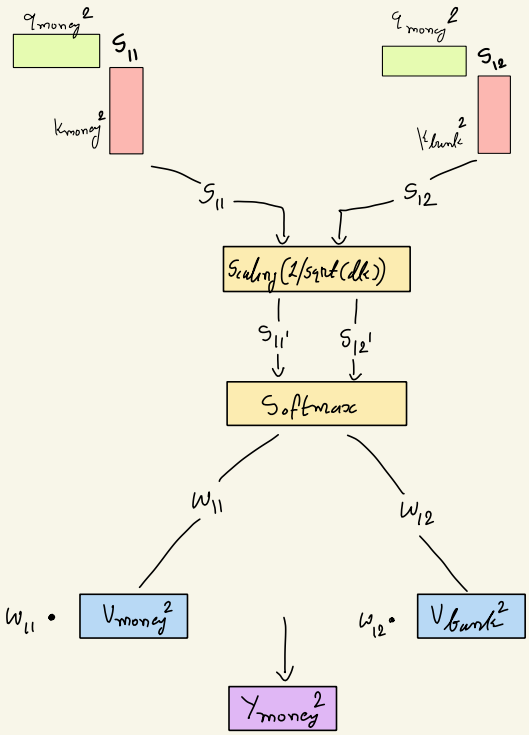
→ for multi-head architecture, we will expand the layer of weights, for each self-attention architecture.

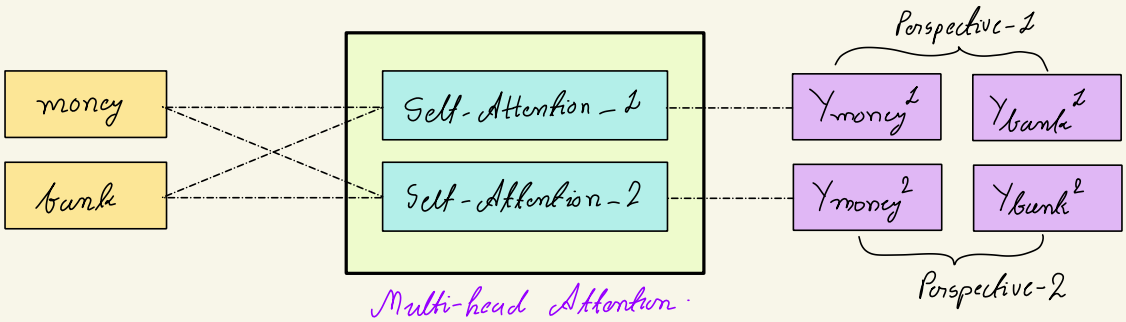
Dual Self-attention



- Now next process will execute twice, due to 2 self-attention architecture.







Positional Encoding

→ Self-attention mechanism was able to handle all words of sentence synonkronously
→ But this feature caused a disadvantage that self-attention mechanism can't capture the order of words in given i/p sentence.
→ hence, for self-attention mechanism
- Parth killed lion
- Lion killed parth

and are same. self-attention doesn't know about how words are exist in i/p sentence.

- hence positional encoding encodes the position of each word in given sentence.
- The simpal way is to append index number at the end of the context vector.
- This solution is unbounded (No upper limit) because, if the index-number of a perticular word is huge than it will cause problems like exploding gradient prob.
- We can normalize it by dividing each word index with total-number-of-words.
- But this problem will confuse neural network because due to this approach, there will be different normalized-index of same position of different sentences.
- position number shouldn't change to ensure consistent behaviour across training samples.
- Also we can't catch relative positioning in given sentence

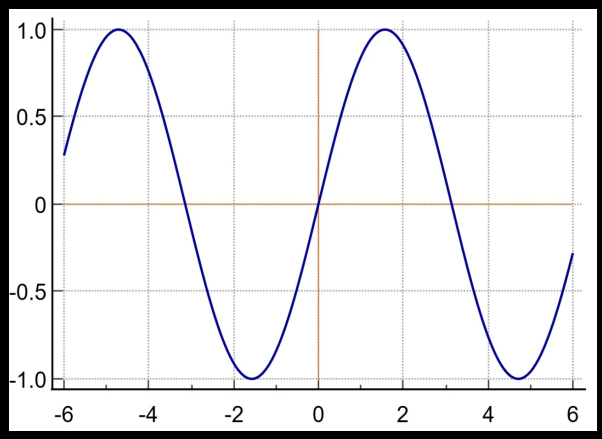
Parth killed a Lion
1 2 3 4

here we're assigning unique positions to each words it is called as absolute positioning. hence model can't be able to calculate the distance between the and Parth. but we need **Relative Positioning.**

- So, instead of discreate function, it would be better if there is a periodic function
- This periodic function gives bounded values. also it should continous & periodic.
- turn that we will use **sine** function.

$$Y = \sin(\text{Position})$$

$\sin(1) = 0.84$
$\sin(2) = 0.87$
$\sin(3) = 0.29$

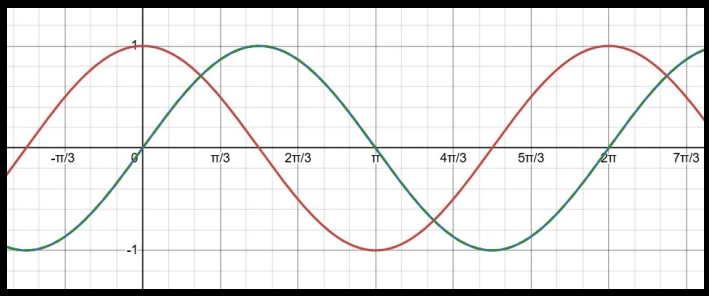


→ But still this method will cause the problem because the positional values are repeating.

→ One solution is to use one more trignometric function with sin.

$Y = \sin(\text{position})$ and $Y = \cos(\text{position})$

→ So we will represent each positional_value as a vector instead of single value.



Ex sentence

Parth killed lion

1 2 3

$Y_1 = \sin(1) = 0.84$

$Y_2 = \cos(1) = 0.50$

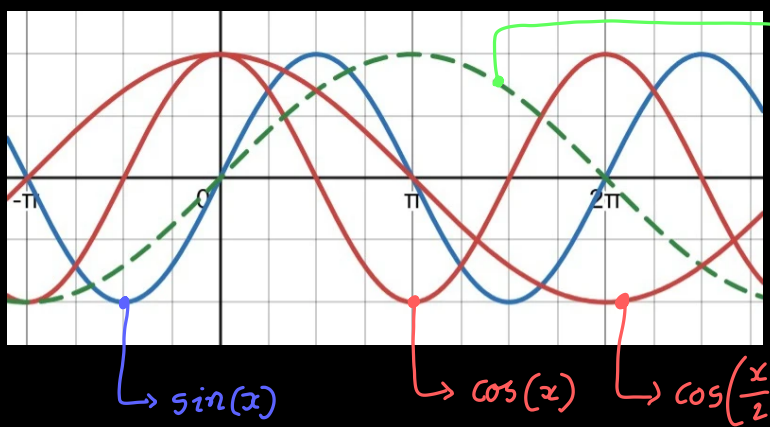
Now the embedding of a word Parth at index position 1 will looks like this.

Parth = | 0.11 | 0.23 | 0.7 | 0.99 | 0.23 | 0.84 | 0.50 |

→ Same as we will calculate position of all words & append it at the end of their embeddings.

→ But still it is possible that sin and cos function can also give same value for a particular word.

→ For that we can also add more curves (functions).



→ $\sin\left(\frac{x}{2}\right)$

Now for each position we will have 4 values.

$Y_1 = \sin(x)$ $Y_2 = \cos(x)$

$Y_3 = \sin\left(\frac{x}{2}\right)$ $Y_4 = \cos\left(\frac{x}{2}\right)$

Now embedding will looks like this.

| 0.11 | 0.23 | 0.7 | 0.99 | 0.23 | 0.84 | 0.50 | 0.94 | 0.73 |

→ In case of maching we can also add one more layer like $Y = \sin(x/3)$ and $Y = \cos(x/3)$ and continues.....

Points must be remember

- Each positional encoding will be a vector.
- The dimentions of positional encoded vector will be same as dimention of embedding vector

| 0.26 | 0.75 | 0.36 | 0.8 | 0.01 | 0.21 | 0.33 | 0.48 |

↳ But according to original research paper "Attention is all you need" we will perform elementwise addition between positional vector & encodding vector.

| 0.26 | 0.75 | 0.36 | 0.84 | + | 0.01 | 0.21 | 0.33 | 0.48 | = | 0.27 | 0.36 | 0.69 | 0.56 |

↳ Now this combined vector will be sent to self-attention mechanism.

→ Previous vector concatination approch is clearly increasing number of dimenshions. This will leads to slower training time.

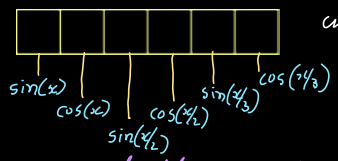
→ As we have discussed previously that....

" The dimentions of positional encoded vector will be same as dimention of embedding vector ".

Suppose embedding vector is a 6D vector. that means positional encoder will also be a 6D vector. and Let's see how we will fill 6D positional vector.

Embedding vector

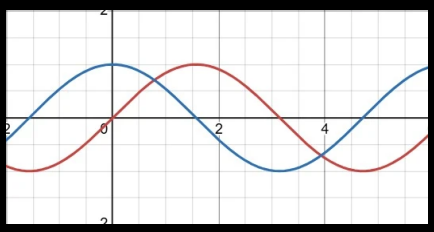
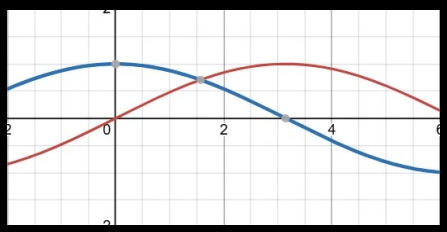
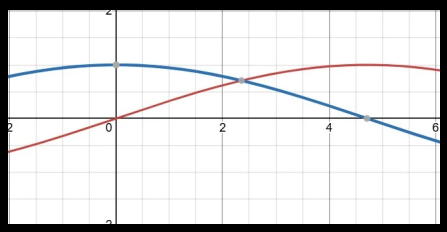
0.6	0.8	0.11	0.03	0.6	0.97



we will work in pairs with sine & cosine curve. for 1st pair $Y_1 = \sin(x)$ & $Y_2 = \cos(x)$.
for 2nd pair $Y_1 = \sin(x/2)$ & $Y_2 = \cos(x/2)$
for 3rd pair $Y_1 = \sin(x/3)$ & $Y_2 = \cos(x/3)$

- In orizinal research paper embedding vector is of 512D, hence positional encoding will also be 512D.
- As we increases pairs, the frequency of neat sin, cosine curve frequency gets reduce.

$Y = \sin(x)$ $Y = \cos(x)$ $Y = \sin(x/2)$ $Y = \cos(x/2)$ $Y = \sin(x/3)$ $Y = \cos(x/3)$

Q. How frequency will be decided...?

- At which rate, we will reduce frequency.
- Till now we have used natural numbers to reduce frequency.

Based on "Attention is all you need"

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \qquad PE_{(pos,2i+2)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

pos → position
d_{model} → dimentionality of embeddings

Let's see an example of a word "River bank".
1 2

Suppose we're working with 6D embedding vector & positional encoder.

hence, frequency(i) = $6/2 = 3$ 6: Total dimention 2: Pair

"River"

for $i = 0$

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

$PE(0,1) = \cos\left(0 / 10000^{0}\right) = 1$

for $i = 1$

$PE(0,2) = \sin\left(0 / 10000^{2/3}\right) = 0$

$PE(0,3) = \cos\left(0 / 10000^{2/3}\right) = 1$

for $i = 2$

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

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

"Bank"

for $i = 0$

$PE(1,0) = \sin\left(1 / 10000^{0}\right) = 0.84$

$PE(1,1) = \cos\left(1 / 10000^{0}\right) = 0.54$

for $i = 1$

$PE(1,2) = \sin\left(1 / 10000^{2/3}\right) = 0.04$

$PE(1,3) = \cos\left(1 / 10000^{2/3}\right) = 0.99$

for $i = 2$

$PE(1,4) = \sin\left(0 / 10000^{2/3}\right) = 0.00$

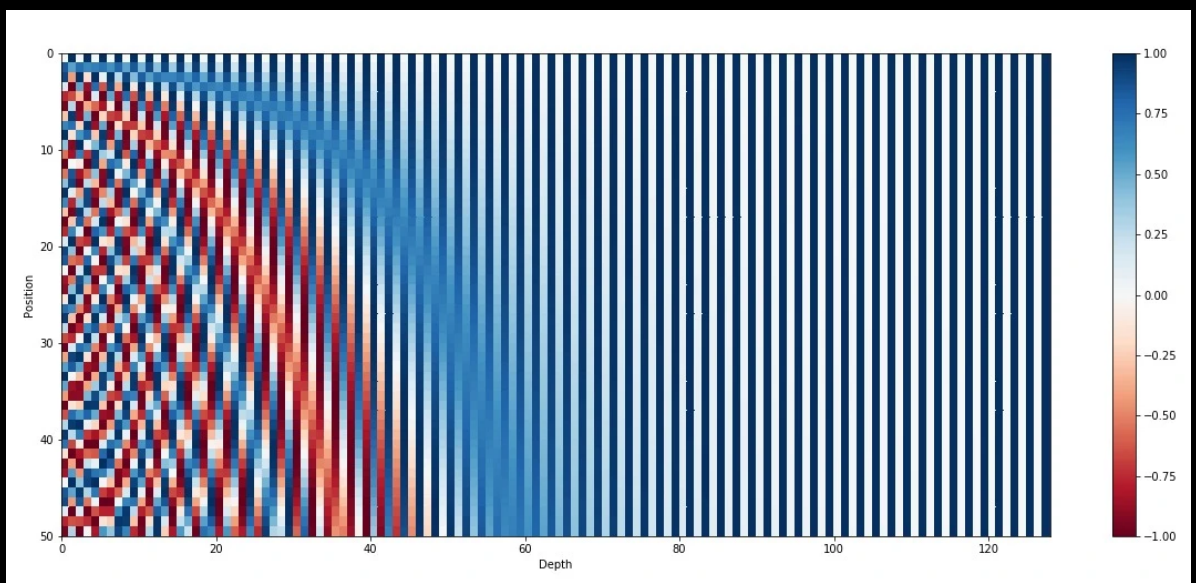
$PE(1,5) = \cos\left(0 / 10000^{2/3}\right) = 0.99$

0	1	0	1	0	1

"River"

0.84	0.54	0.04	0.99	0.00	0.99

"Bank"



Based on above graph

- Total words in sentence : 50
- Embedding dimention of each word : 128 ($d_{model} = 128$)
- Total positional encodings : 50
- Dimention of each positional encoding : 128

- We can see that majority changes are happening on lower dimentions (Depth 0 - 75)

•> for higher dimentions (Depth > 79) there is no change because we're reducing frequency as number of dimentions increases.

Q. How sine, cosine can capture relative positioning ?

•> Due to this sine, cosine set-up, our positional encodding vector got an interesting property.

•> Let's imagine each 50 words as vectors (V_i). Now if we apply linear transformation on V_{20} then we will reach at V_{20}. same transformation matrix if we apply on V_{20} then we will reach at V_{30}.

•> This particular linear transformation matrix has capability of covering delta of 10 on distance. & this property is available for each delta.

Example

M_1 = Matrix ⟶ This matrix M_1 has capability of covering delta/distance of 5.

$V_5 \rightarrow M_1 \rightarrow V_{10}$ This feature helps to understand relative positioning.

$V_{12} \rightarrow M_1 \rightarrow V_{17}$

$V_{21} \rightarrow M_1 \rightarrow V_{26}$

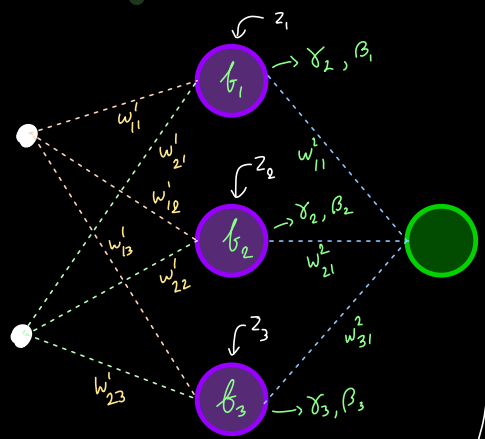
Layer Normalization

Q. what is normalization

•> In deep learning, normalization refers to the process of transforming data or model o/p to have specific properties, typically $\mu = 0$, $\sigma = 1$.

•> Layer Normalization normalizes activations (Hidden layer).

Quick recap of batch normalization



batch_size = 5

f_1	f_2	z_1	z_2	z_3
2	3	7	5	4
1	1	2	3	4
5	4	1	2	3
6	1	7	5	6
7	1	3	3	4
μ_1	μ_2	μ_3		
σ_1	σ_2	σ_3		

$z_1 = 2 \cdot w'_{11} + 3 \cdot w'_{21} + b_1 = 7$

$z_2 = 2 \cdot w'_{12} + 3 \cdot w'_{22} + b_2 = 5$

$z_3 = 2 \cdot w'_{13} + 3 \cdot w'_{23} + b_3 = 4$

$\frac{7 - \mu_1}{\sigma_1} = 0.36 \cdot \gamma_1 + \beta_1 = 0.36$ → activation ($\sigma(0.36)$)

$\frac{2 - \mu_1}{\sigma_1} = 0.71 \cdot \gamma_1 + \beta_1 = 0.71$

$\frac{5 - \mu_2}{\sigma_2} = -0.21 \cdot \gamma_2 + \beta_2 = -0.21$

$\frac{4 - \mu_3}{\sigma_3} = 0.11 \cdot \gamma_3 + \beta_3 = 0.11$

• Here γ_i and β_i are trainable parameters whose initial values will be 1 and 0 respectively.

- We can't use batch normalization on self-attention mechanism
- We can also send multipul sentences in self-attention mechanism in the form of batches.

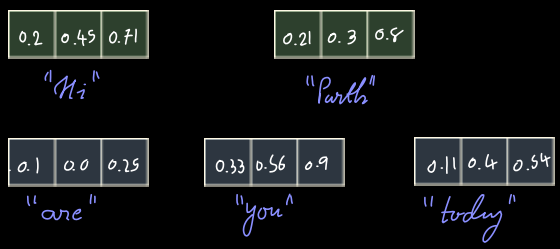
Example

Review	Sentiment
Hi parth	1
How are you today	0
I am good	0
You ?	1

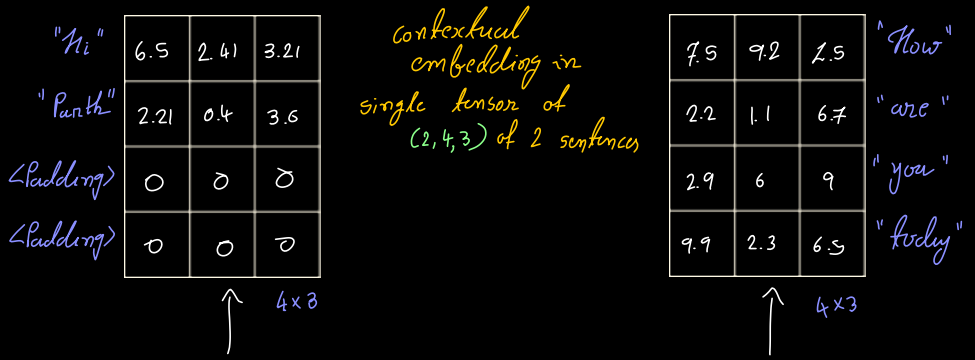
Embedding_dim = 3
batch_size = 2 (At a time, 2 sentences will be passed in self-attention mechanism)

Step-1

- Embedding vector of first 2 sentences



- We need to apply padding on first sentence, beause we can clearly see that both sentences have different number of words.



→ Now we need to normalize o/p matrix. for that we will stack-up both o/p matrix & perform normalization as shown demo previously. each column will have it's own γ and β.

	d_1	d_2	d_3
"Hi"	6.5	2.41	3.21
"Parth"	2.21	0.4	3.6
<Padding>	0	0	0
<Padding>	0	0	0
"How"	7.5	9.2	1.5
"are"	2.2	1.1	6.7
"you"	29	6	9
"today"	9.9	2.3	6.5
	μ_1	μ_2	μ_3
	σ_1	σ_2	σ_3
	γ_1	γ_2	γ_3
	β_1	β_2	β_3

- Here is a big problem arrises, just imagine we have batch_size = 32 (32 sentences at a time) & the longest sentence is of 200 words. and remained sentences have average 32 words. so we need to add padding (will create sparse matrix) and padding will bring majority 0s. Now this unnecessary 0s (Not helpfull in calculations just used for maintaining stable i/p.) won't let us get the actual μ and σ of data because they just hold the space.

Hence, we can't use normalization in sequential data. due to 0s in padding.

So the solution is layer Normalization.....

→ in layer Normalization we will apply normalization across features (Horizontal direction)

Example

f_1	f_2	z_1	z_2	z_3
2	3	7	5	4
1	1	2	3	4
5	4	1	2	3
6	1	7	5	6
7	1	3	3	4

only of z_i

μ_1, σ_1 $\left(\frac{7-\mu_1}{\sigma_1}\right)\gamma_1+\beta_1$ $\left(\frac{2-\mu_2}{\sigma_2}\right)\gamma_2+\beta_2$

μ_2, σ_2 $\left(\frac{5-\mu_1}{\sigma_1}\right)\gamma_2+\beta_2$ $\left(\frac{3-\mu_2}{\sigma_2}\right)\gamma_2+\beta_2$

μ_3, σ_3 $\left(\frac{4-\mu_1}{\sigma_1}\right)\gamma_3+\beta_3$ $\left(\frac{4-\mu_2}{\sigma_2}\right)\gamma_3+\beta_3$

μ_4, σ_4

μ_5, σ_5

and so on.....

Now let's see with embedding example

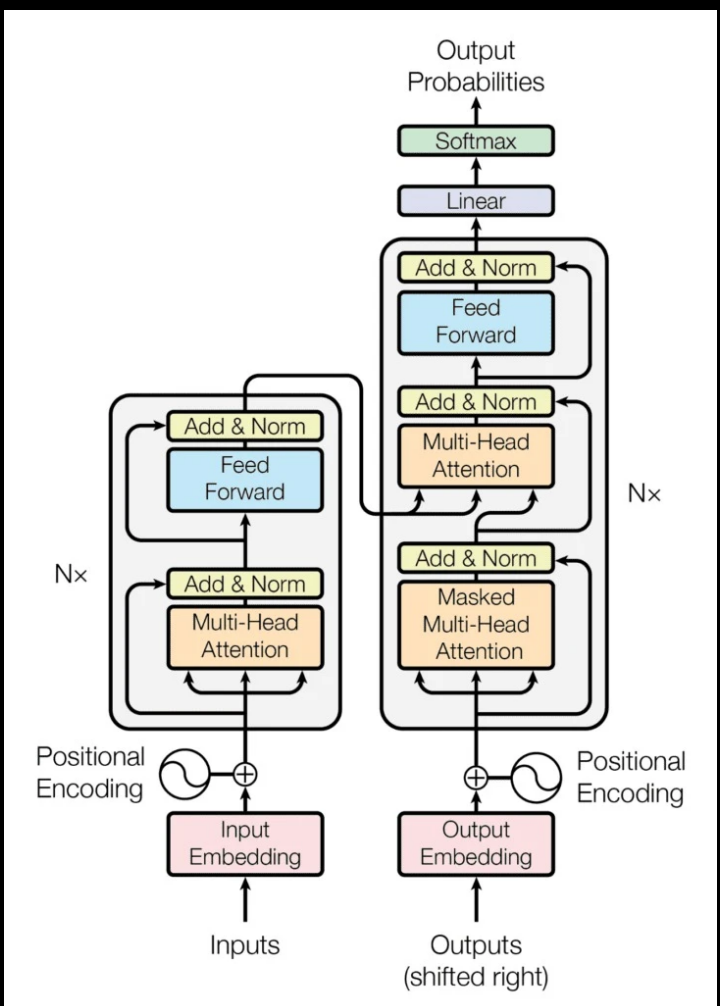
	d_1	d_2	d_3	
"Hi"	6.5	2.41	3.21	μ_1, σ_1
"Panth"	2.21	0.4	3.6	μ_2, σ_2
<Padding>	0	0	0	μ_3, σ_3
<Padding>	0	0	0	μ_4, σ_4
"How"	7.5	9.2	2.5	μ_5, σ_5
"are"	2.2	1.1	6.7	μ_6, σ_6
"you"	2.9	6	9	μ_7, σ_7
"today"	9.9	2.3	6.5	μ_8, σ_8
	γ_1 β_1	γ_2 β_2	γ_3 β_3	

$$\left(\frac{6.5 - \sigma_1}{\mu_1}\right)\gamma_1 + \beta_1 \qquad \left(\frac{7.5 - \mu_5}{\sigma_5}\right)\gamma_1 + \beta_1$$

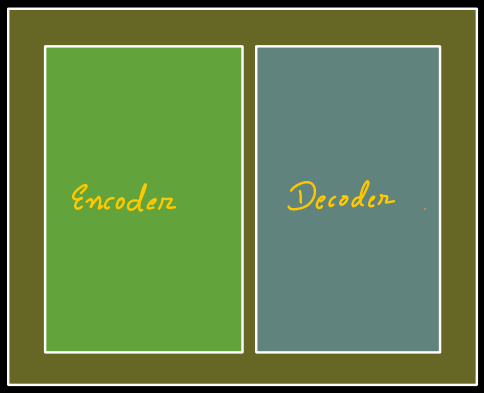
$$\left(\frac{0.4 - \mu_2}{\sigma_2}\right)\gamma_2 + \beta_2 \qquad \left(\frac{9 - \mu_7}{\sigma_7}\right)\gamma_3 + \beta_3$$

- In this process padding valued rows won't make any impact and that is very good benefit.

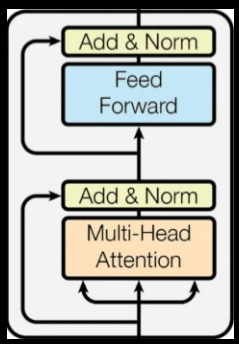
Transformer Architecture

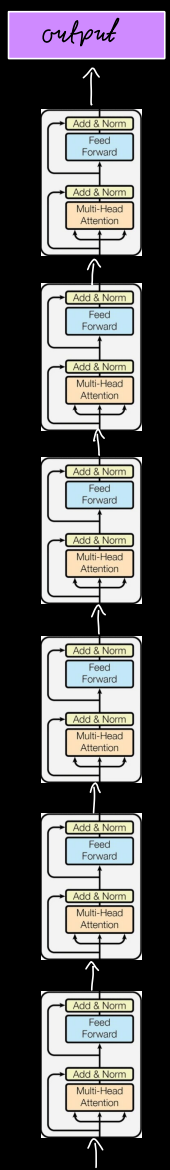


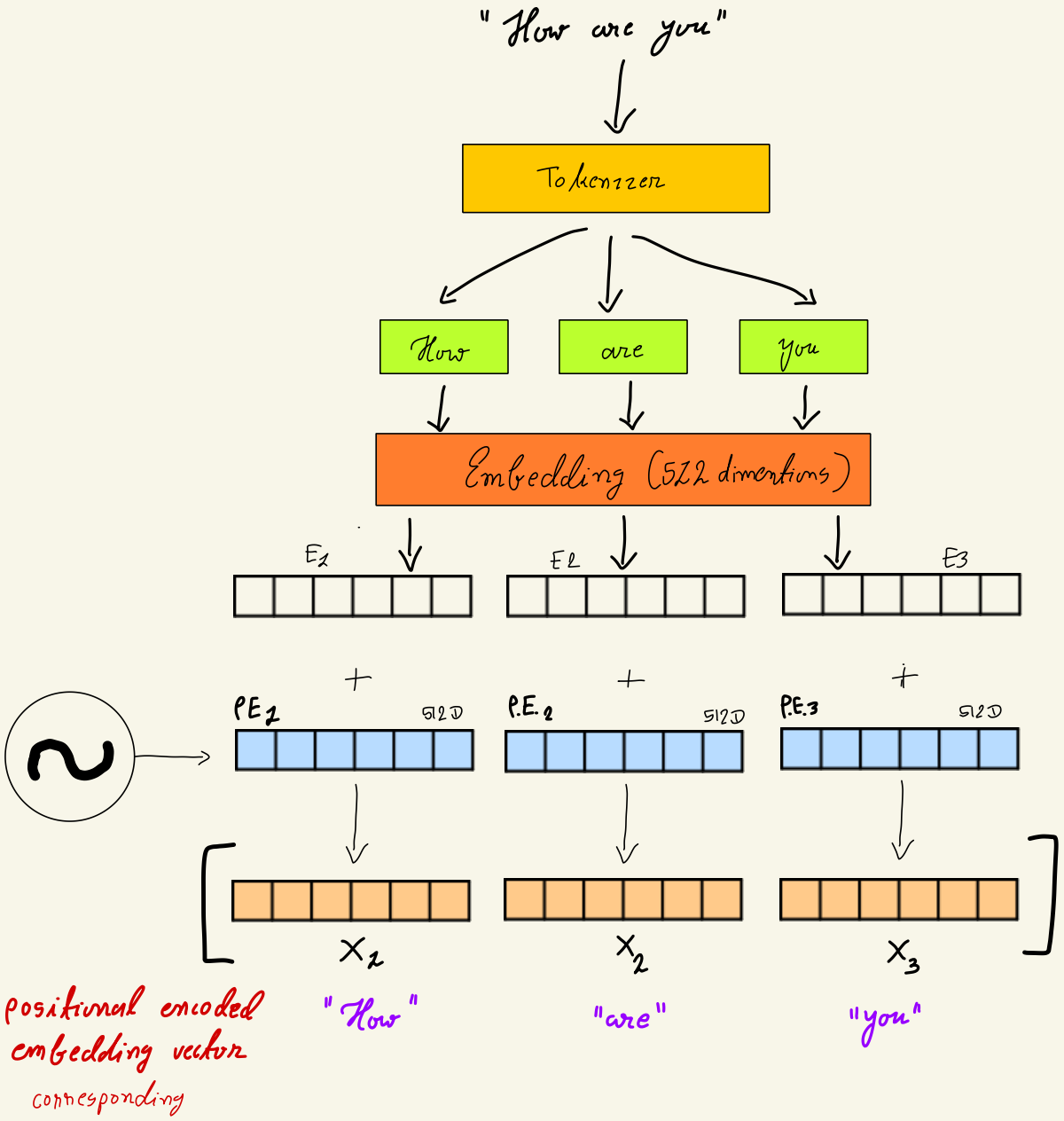
Simplified representation



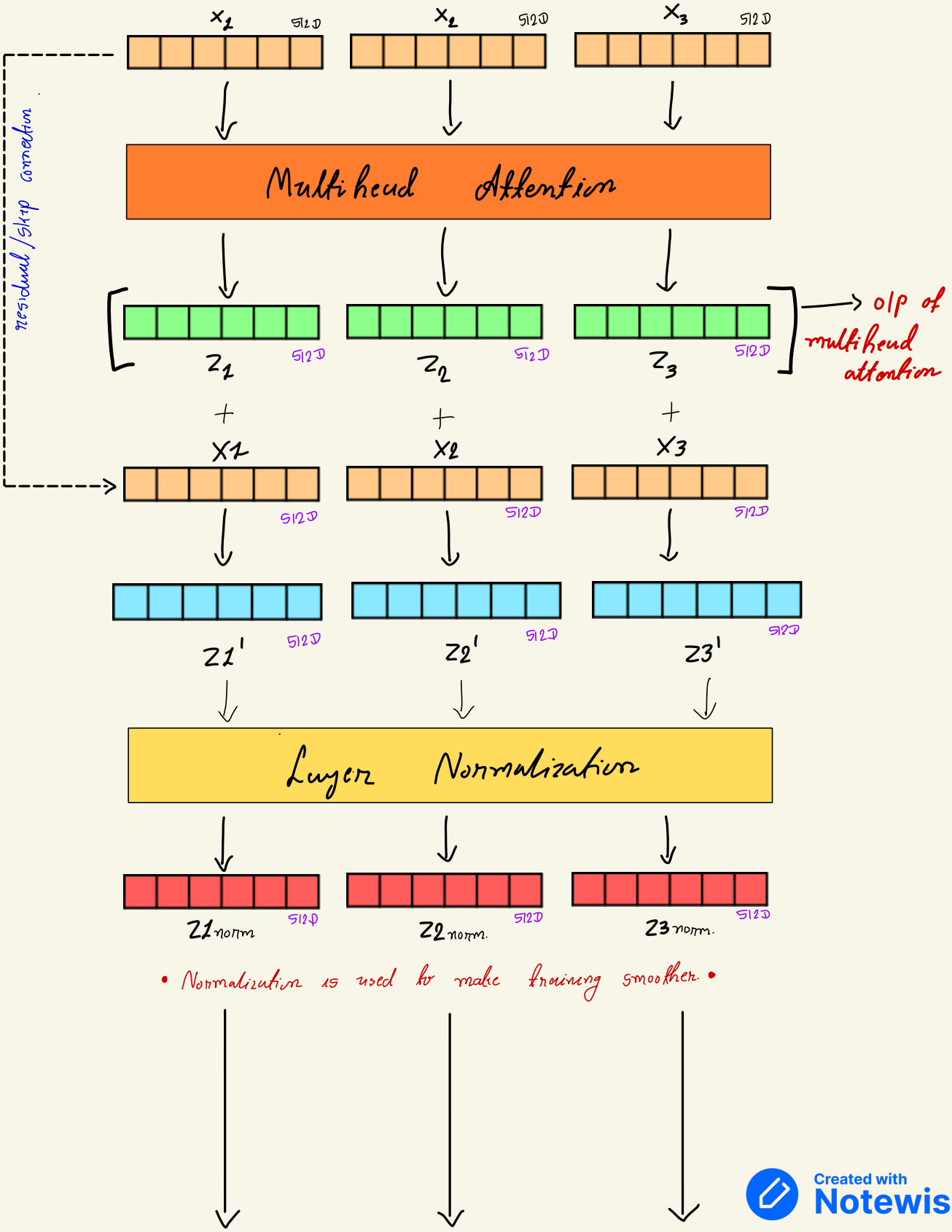
each block are identical

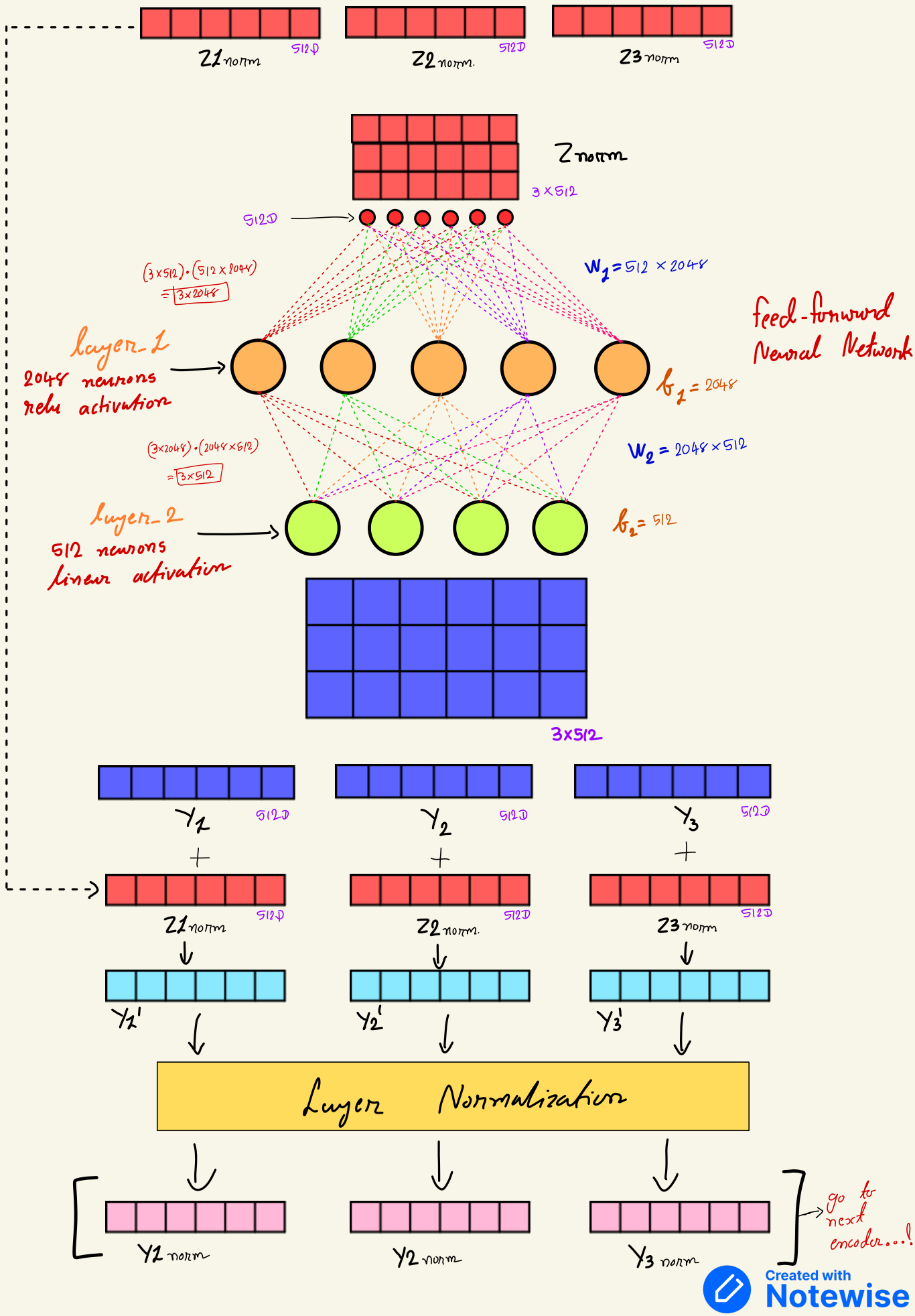






This is what i/p block does before passing data to encoder architecture for further process.




Z1 norm
512D
Z2 norm.
512D
Z3 norm
512D
Z norm
3 x 512
512D
(3x512) . (512 x 2048)
= 3x2048
W1 = 512 x 2048
feed-forward
Neural Network
layer_1
2048 neurons
relu activation
b1 = 2048
(3x2048) . (2048x512)
= 3x512
W2 = 2048 x 512
layer_2
512 neurons
linear activation
b2 = 512
3x512
Y1
512D
Y2
512D
Y3
512D
Z1 norm
512D
Z2 norm.
512D
Z3 norm
512D
Y1'
Y2'
Y3'
Layer Normalization
Y1 norm
Y2 norm
Y3 norm
go to
next
encoder...!

→ Each encoder block has separate trainable parameters, even though the architecture is same.

Q1. Why use residual connections.?

- → Stable training.
- → Forwards original features as it is.

Q2. Why use a FFNN ?

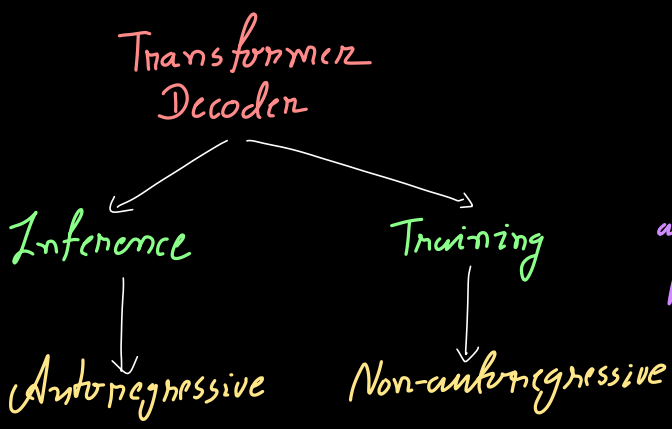
- → To capture non-linearity / non-linear relationship.

Q3. Why use 6 encoder blocks ?

- → To capture the context of complex language.

Autoregressive Model

The transformer decoder is autoregressive at inference (prediction) time and non-autoregressive at training time.



In the context of DL, autoregressive models are class of models that generates data points in a sequence by conditioning each new point on the previously generated points.

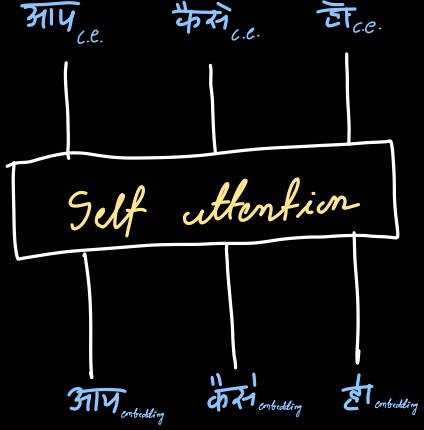
Ex. Stock price prediction.

In short current newly generated datapoint is depended on previous datapoints.

Remember LSTM architecture ....!

Transformer's decoders are autoregressive at inferential time not at training time is due to Masked Self-attention.

Sentence Number	English Sentence	Hindi Sentence
1	How are you.?	आप कैसे हैं
2	Congratulations	बधाई हो
3	Thank you	धन्यवाद



$\text{आप}_{c.e.} = 0.8\,\text{आप}_{embedding} + 0.1\,\text{कैसे}_{embedding} + 0.1\,\text{हो}_{embedding}$

Hypothetical numbers

$\text{कैसे}_{c.e.} = 0.15\,\text{आप}_{embedding} + 0.75\,\text{कैसे}_{embedding} + 0.2\,\text{हो}_{embedding}$

$\text{हो}_{c.e.} = 0.2\,\text{आप}_{embedding} + 0.2\,\text{कैसे}_{embedding} + 0.7\,\text{हो}_{embedding}$

Big Problem: We're using future embedding (which is not even exist yet) to calculate current o/p token. This isn't possible at inference time.

Hence, we can't solve $\text{आप}_{c.e.} = 0.8\,\text{आप}_{embedding} + 0.1\,\text{कैसे}_{embedding} + 0.1\,\text{हो}_{embedding}$ this equation, because at current stage we're calculating $\text{आप}_{c.e.}$ and we've not even idea about next words, then we can't calculate even this formula.

This is data leakage problem in parallel approach.

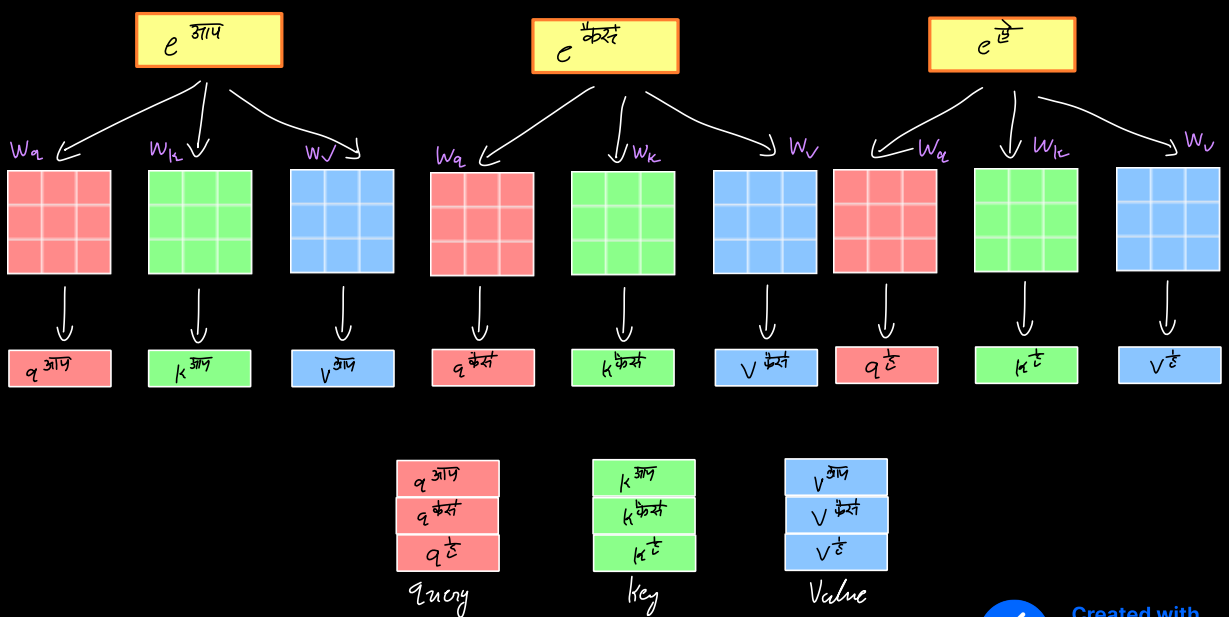
Autoregressive approach

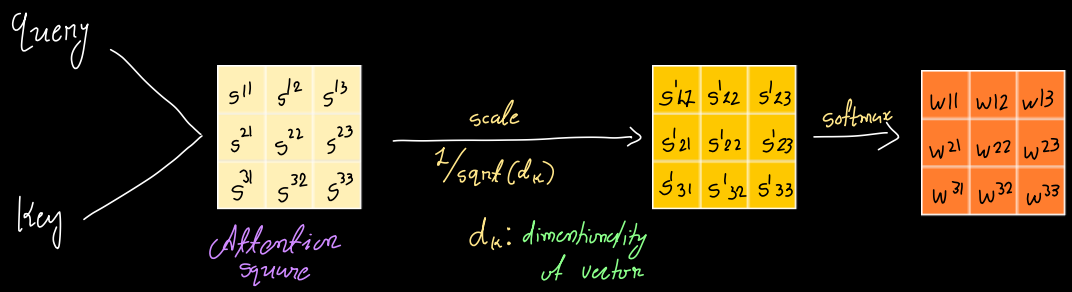
- No data leakage
- Slow training

Non-autoregressive approach

- Fast training
- Data leakage (Your current token can see your future token which is unfair & it is not possible at prediction time)

finding the answer



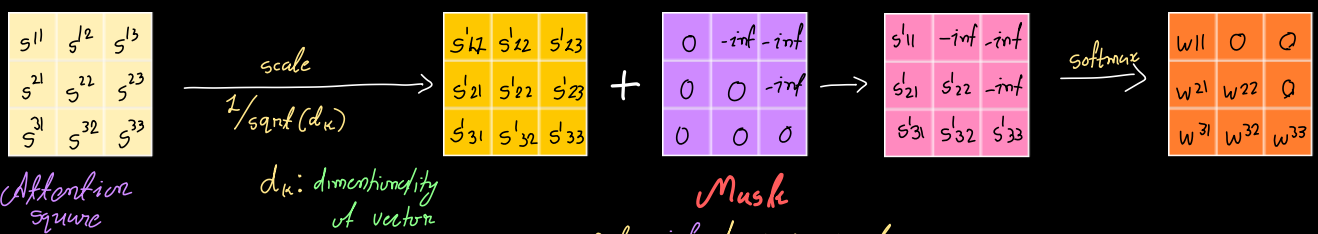


$$\text{आप}_{ce} = w_{11} \cdot v^{\text{आप}} + w_{12} \cdot v^{\text{कैसे}} + w_{13} \cdot v^{\text{हो}}$$

$$\text{कैसे}_{ce} = w_{21} \cdot v^{\text{आप}} + w_{22} \cdot v^{\text{कैसे}} + w_{23} \cdot v^{\text{हो}}$$

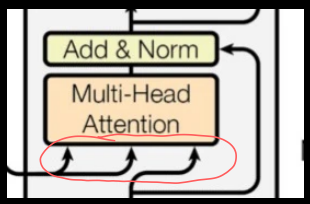
$$\text{हो}_{ce} = w_{31} \cdot v^{\text{आप}} + w_{32} \cdot v^{\text{कैसे}} + w_{33} \cdot v^{\text{हो}}$$

- Here, while calculating आप_{ce} we don't need to include $w_{12} \cdot v^{\text{कैसे}}$ and $w_{13} \cdot v^{\text{हो}}$ because they weren't in the picture yet so we will make w_{12} & $w_{13} = 0$.
- Similarly while calculating कैसे_{ce} we don't need $w_{23} \cdot v^{\text{हो}}$ because it isn't picture yet. so w_{23} will be assigned 0.
- Here is the masking concept comes in picture to make $w_{12} = w_{13} = w_{23} = 0$.
- for that we will add one more step in above matrix operations.



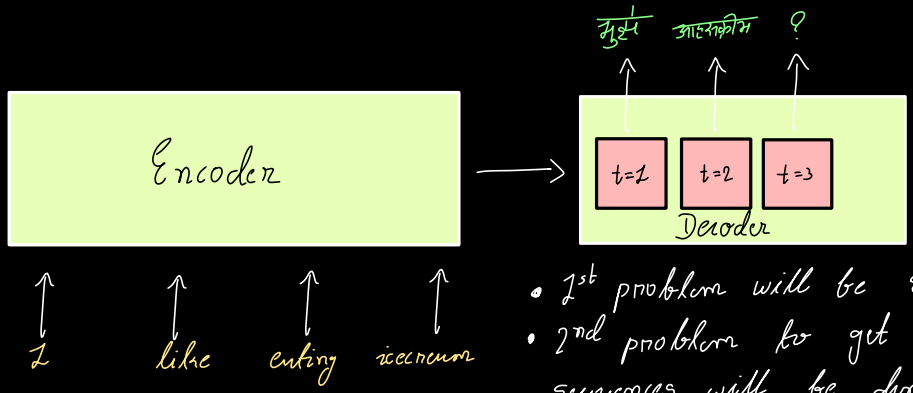
That's how we achieved non-autoregressive modeling in training phase.

Cross Attention / Encoder-decoder Attention



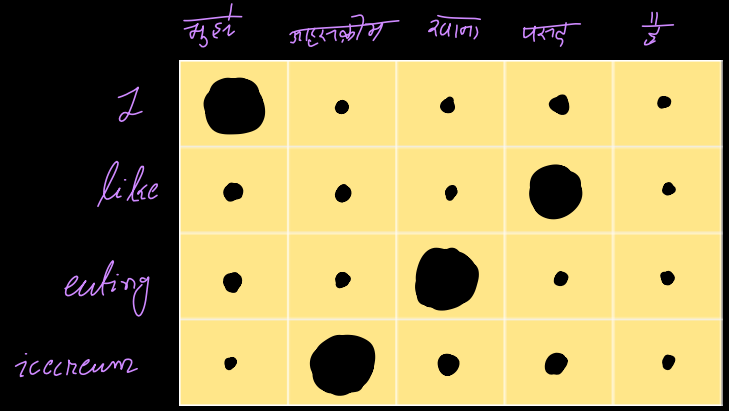
- A mechanism used in transformer architecture, particularly in tasks involving seq.-to-seq. data like translation or summarization. It allows a model to focus on different part of an i/p seq. when generating an o/p seq.

Example of Machine translation



at $t=3$ upcoming prediction depends on 2 things.
1. Generated content till $t=3$.
2. Incoming information from encoder.

- 1st problem will be solved by self-attention.
- 2nd problem to get similarity between two sequences will be done by cross attention.



Same sequence → similarity → self-attention

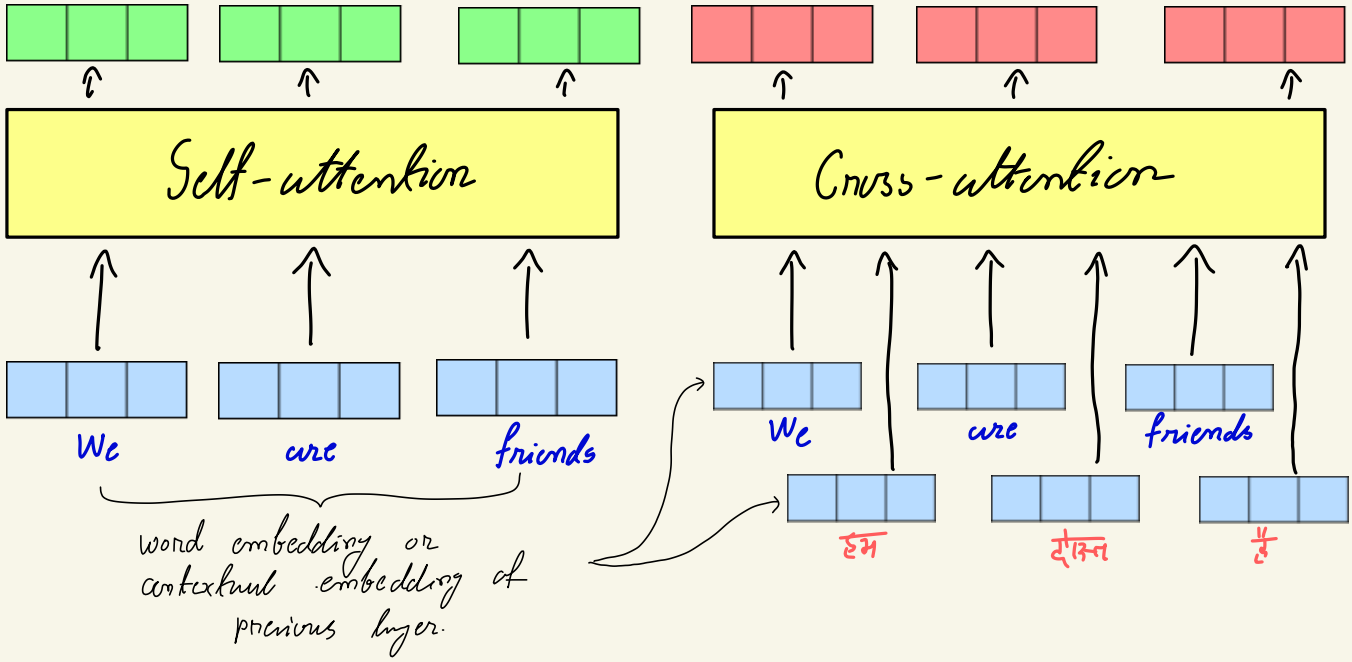
Different sequences → similarity → Cross-attention
(calculating similarity between different sequences)

Cross-attention is conceptually similar to self-attention

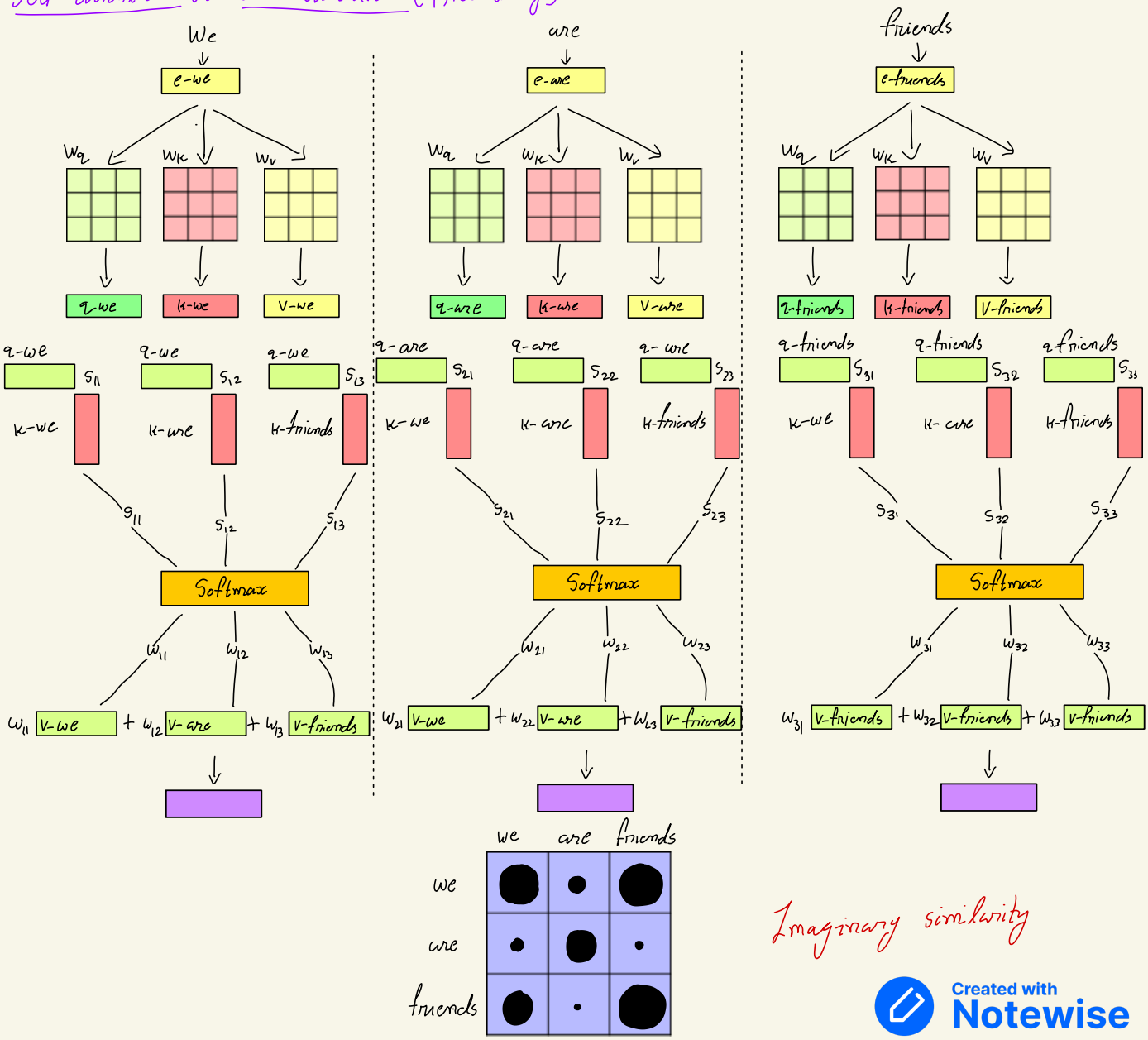
Self attention VS Cross-attention

1. The i/p.
2. The processing.
3. The o/p.

Self attention VS Cross-attention (input)



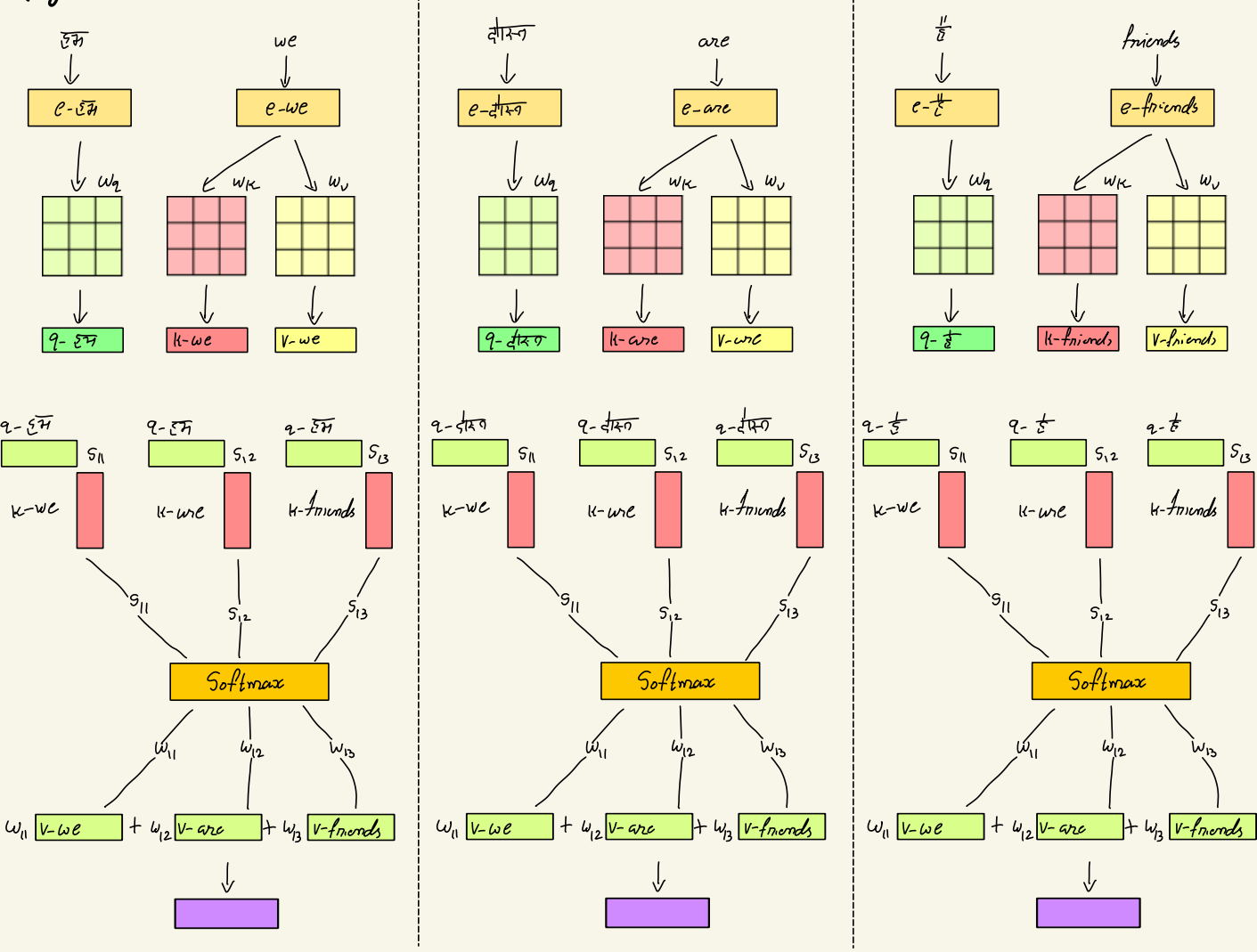
Self attention VS Cross-attention (Processing)

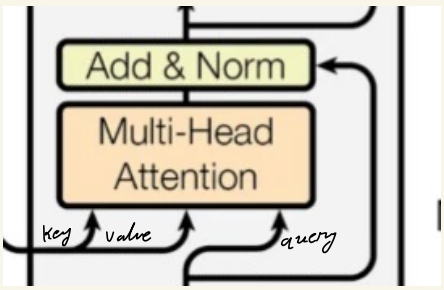


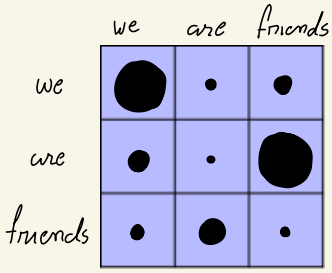
query → O/p vector

key / value → I/p vector

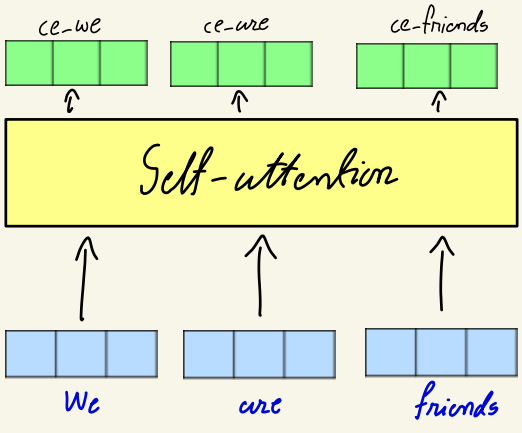
Cross-attention

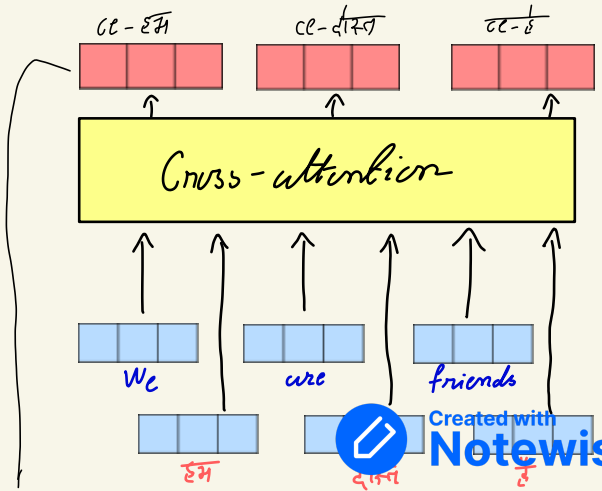






Self attention VS Cross-attention (Output)





ce_we = 0.8 * e_we + 0.1 * e_wre + 0.2 * e_friends

ce_wre = 0.15 * e_we + 0.75 * e_wre + 0.1 * e_friends

ce_friends = 0.2 * e_we + 0.1 * e_wre + 0.7 * e_friends

Number of o/p will be same as number of incoming o/p label

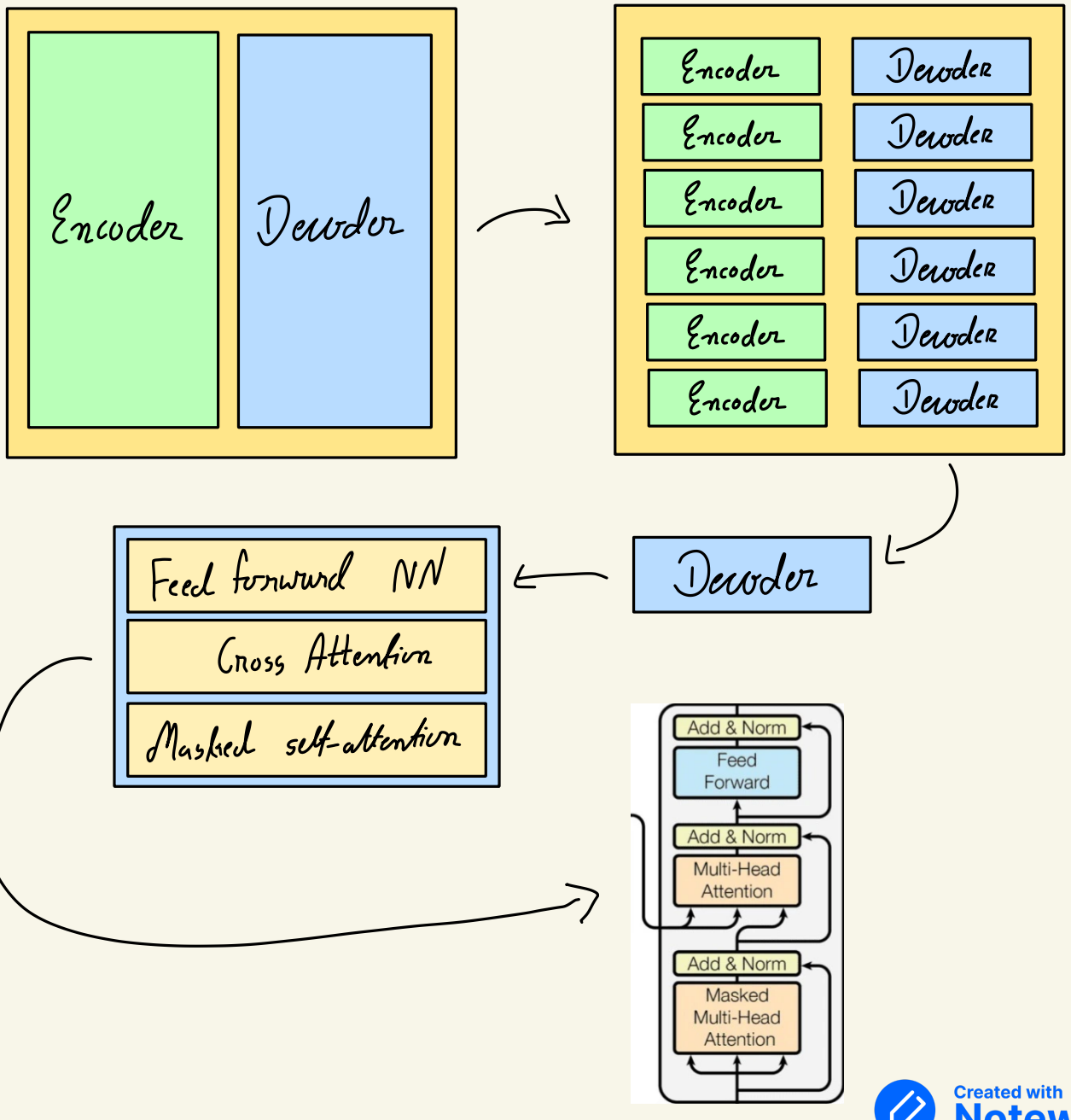
ce_मैं = 0.5 * e_we + 0.3 * e_wre + 0.2 * e_friends

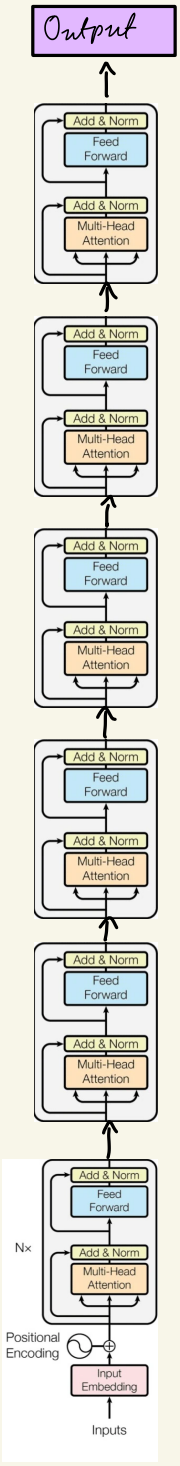
ce_दोस्त = 0.2 * e_we + 0.2 * e_wre + 0.6 * e_friends

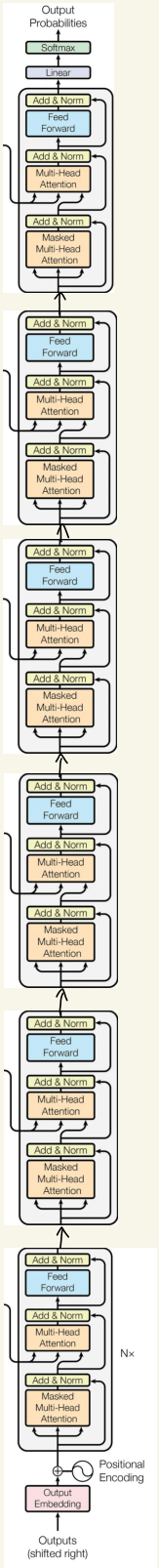
ce_है = 0.3 * e_we + 0.4 * e_wre + 0.3 * e_friends

Use cases

- Image - to - text
- Text - to - image
- Text - to - speech





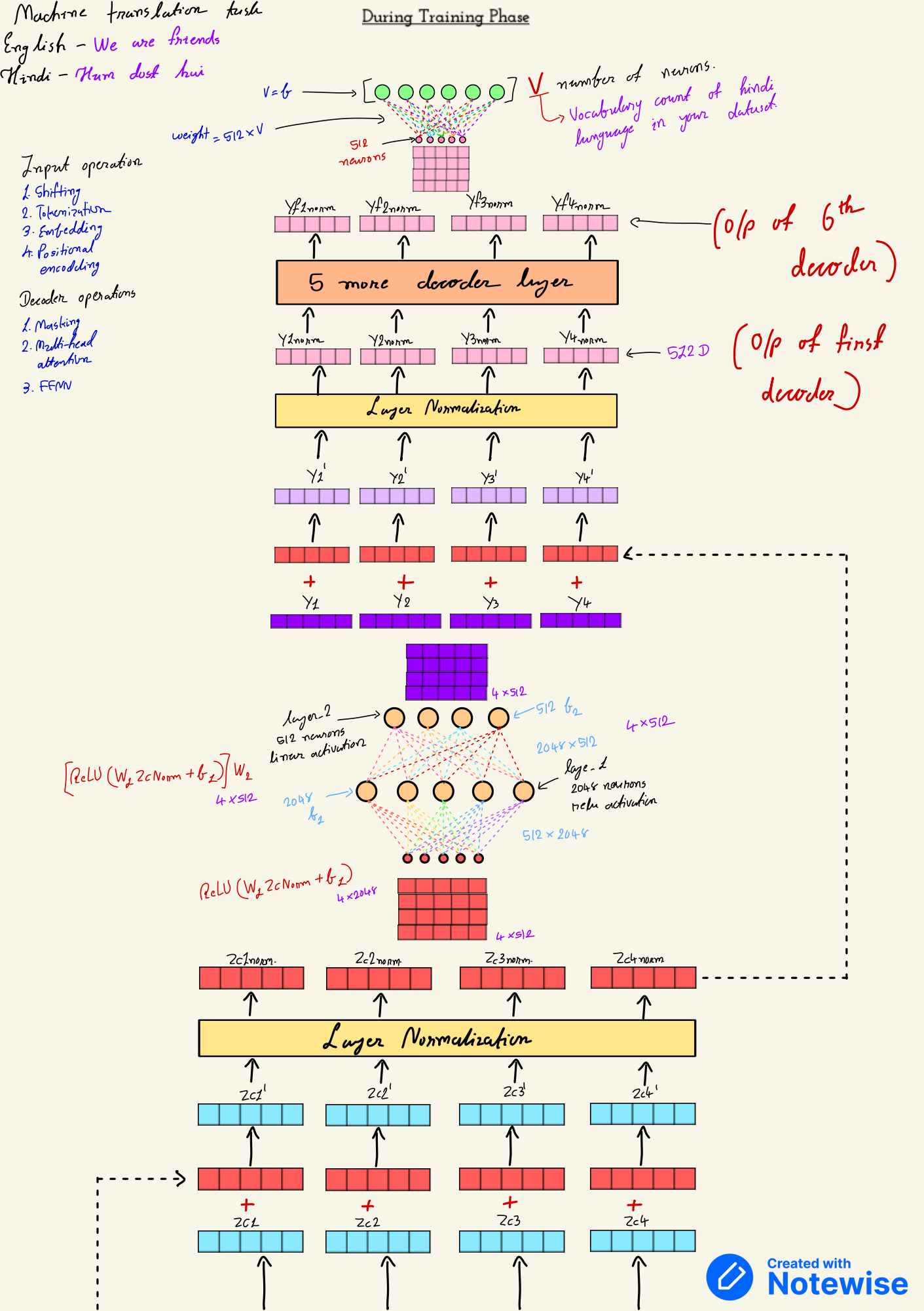


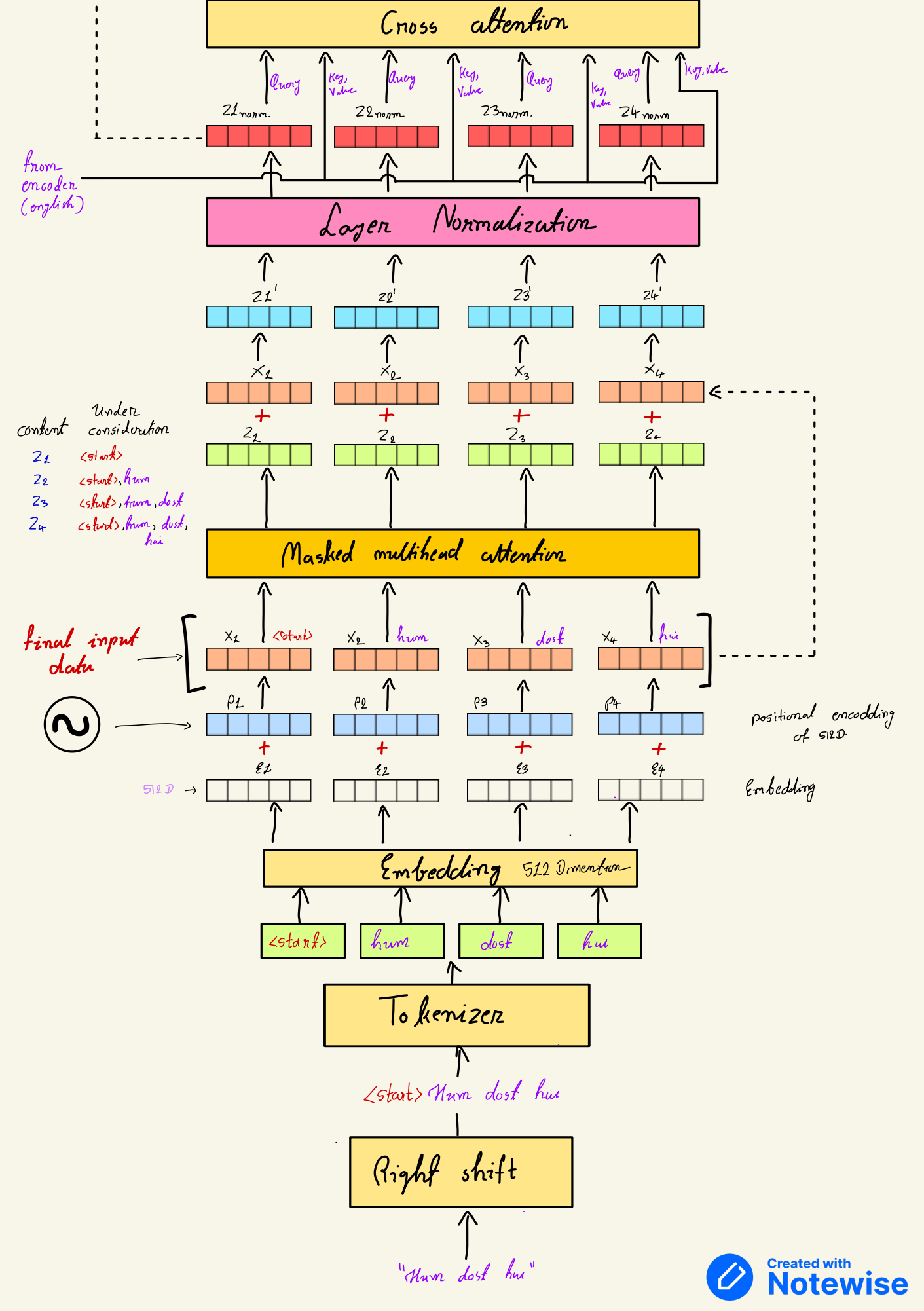
Machine translation task

English - We are friends

Hindi - Hum dost hai

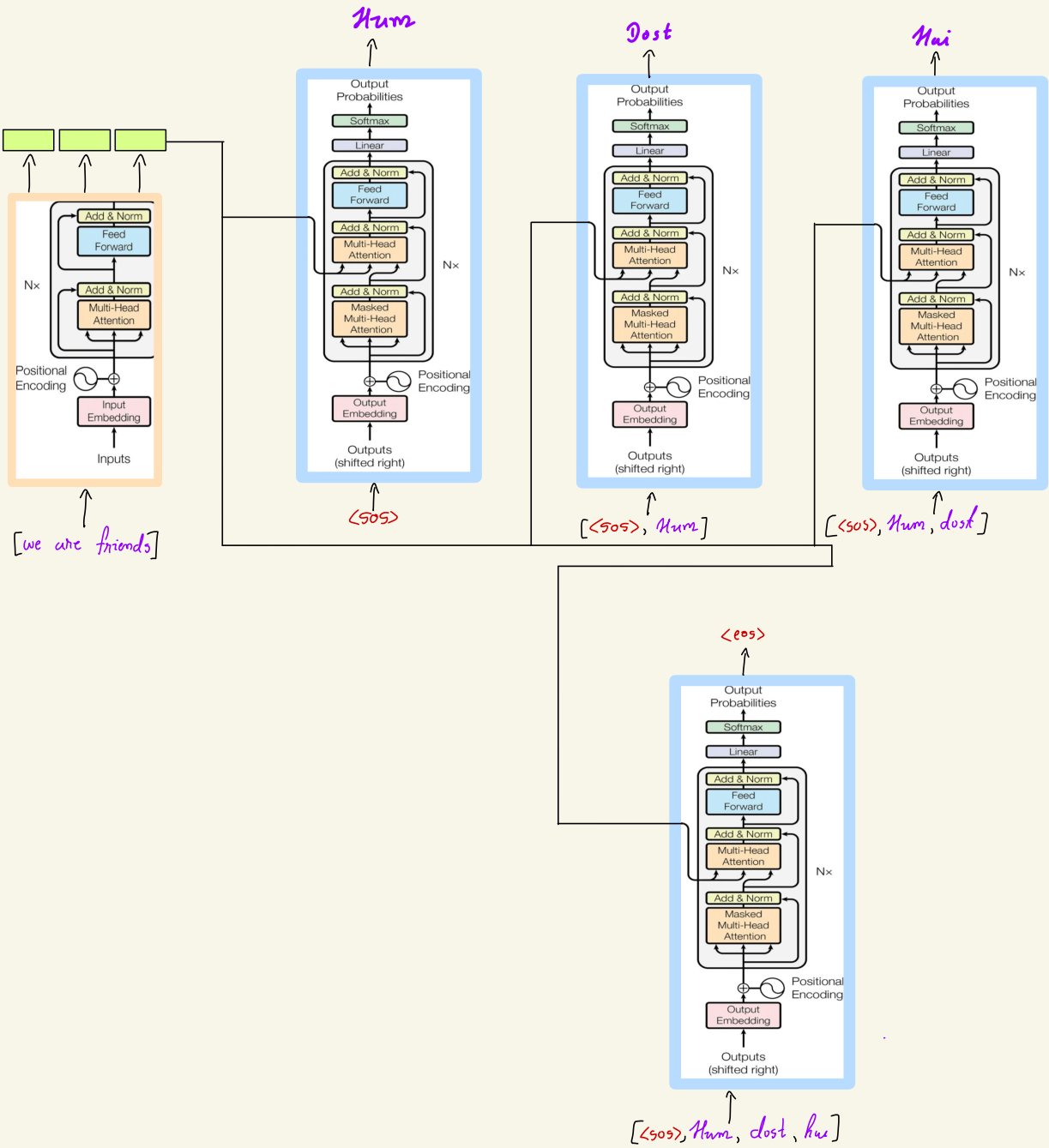
During Training Phase



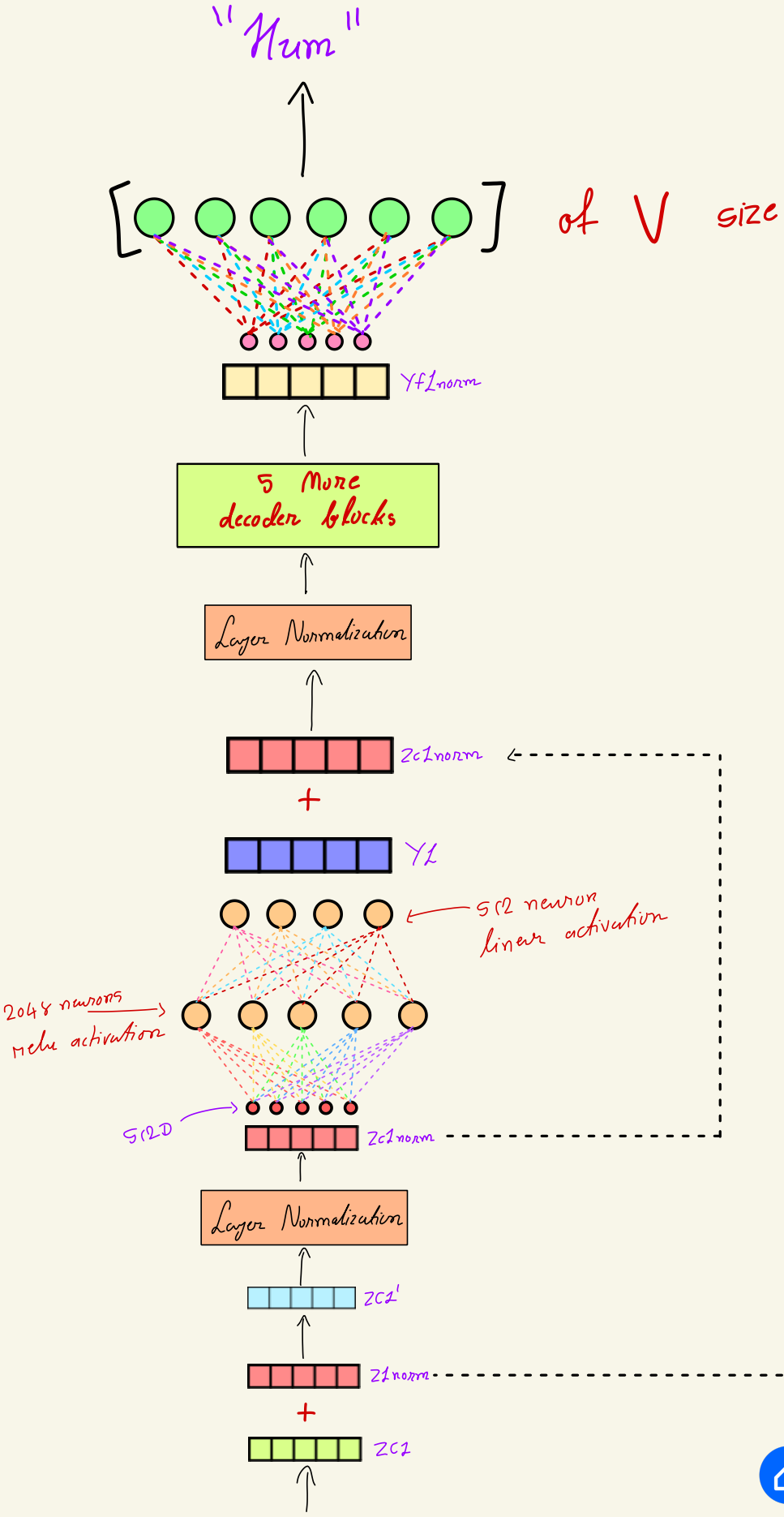


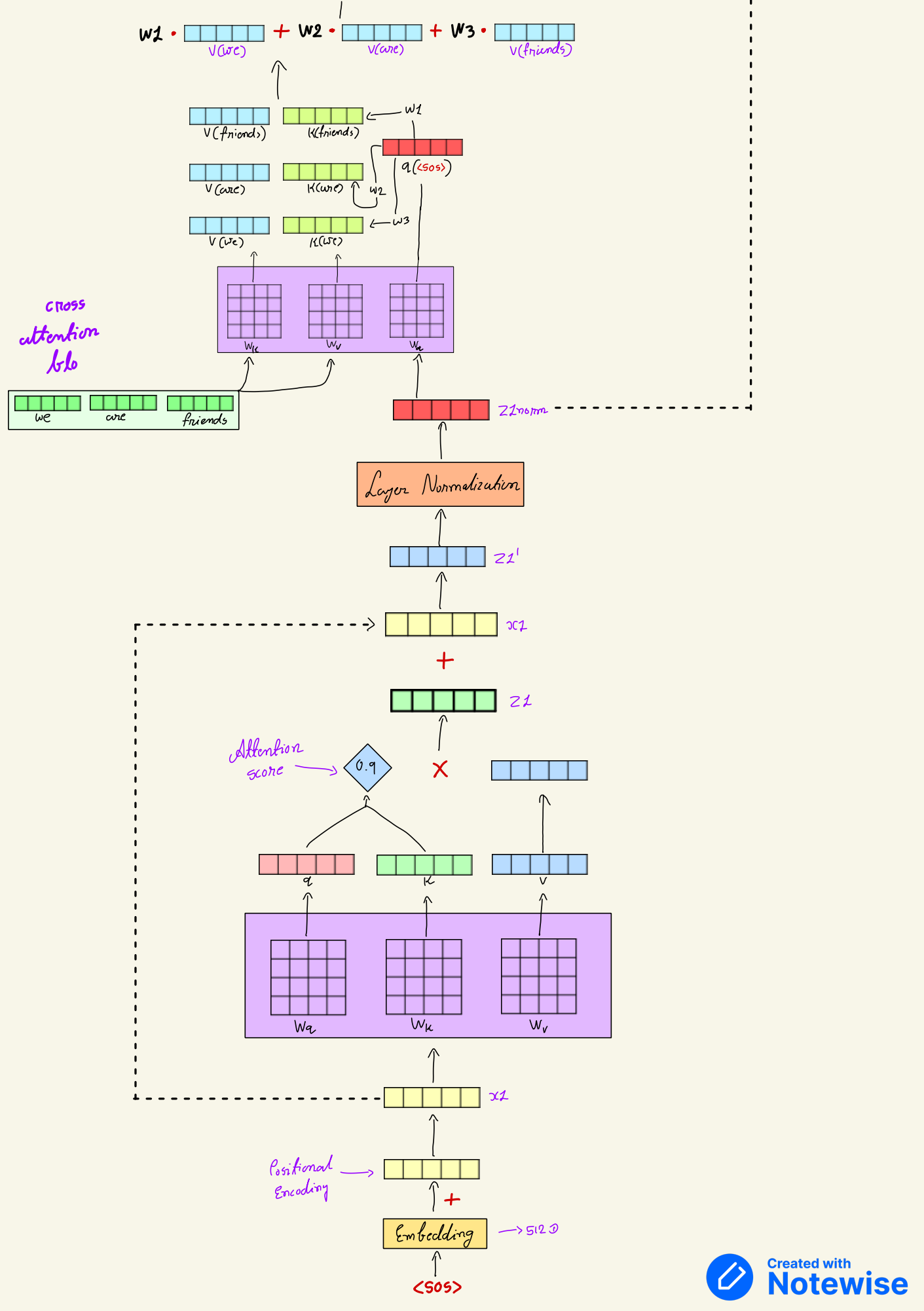
Inferential (Prediction) Phase

- During training phase, decoder works as non-autoregressor & autoregressor during inferential phase.
- While encoder stays non-autoregressor in both phase.
- Firstly, we will send <SOS> (Start of Sentence) flag to start decoding process.



Phase-1





Phase-2