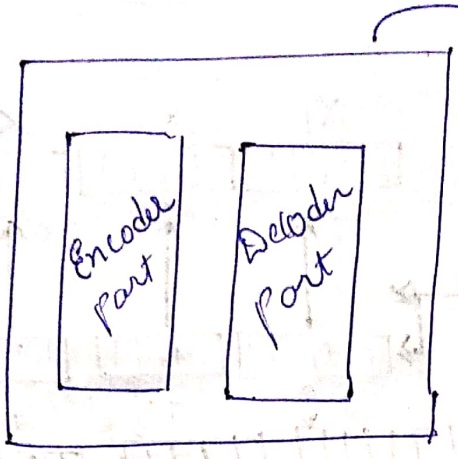
Transformer Architecture

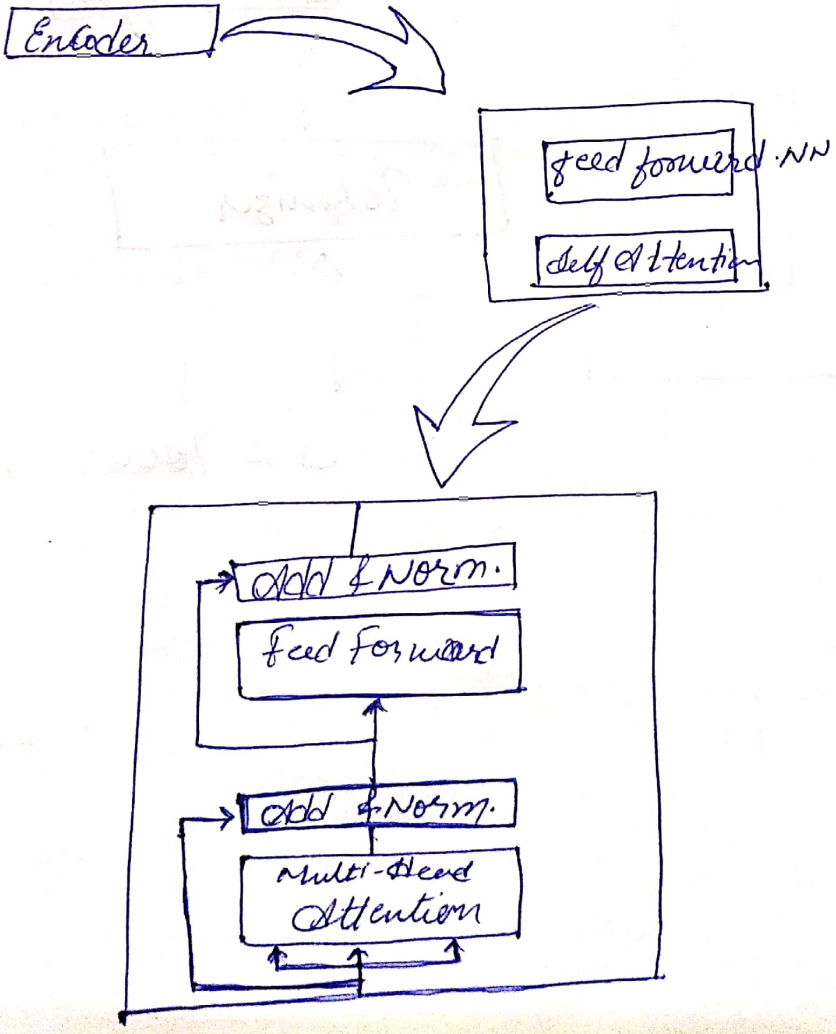
#



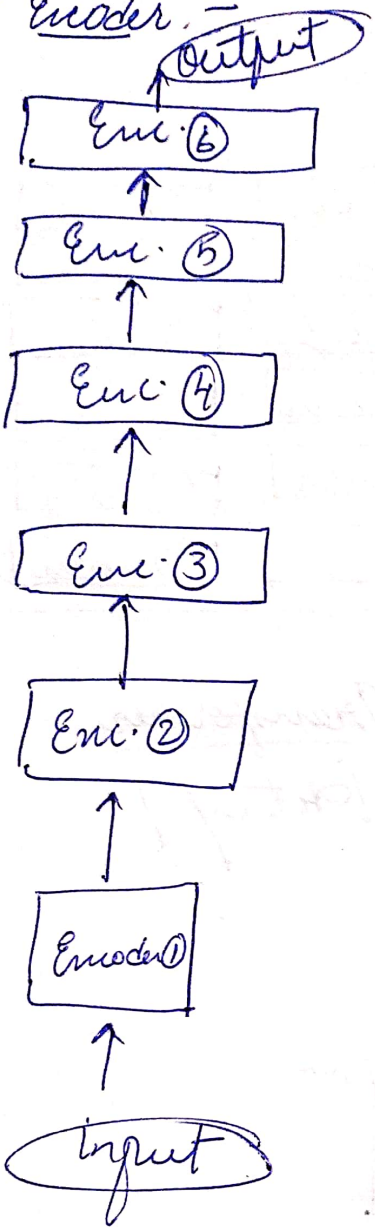
Transformer

Transformer [Actual]

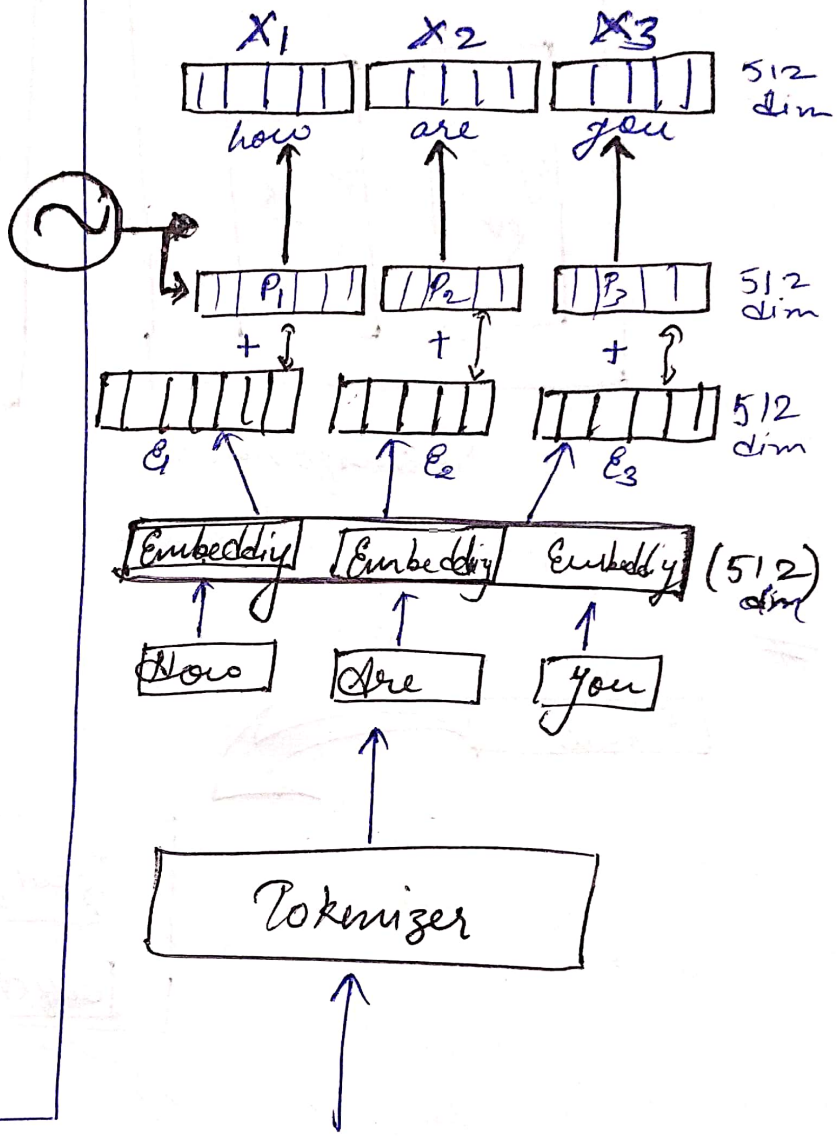
#



Encoder :-

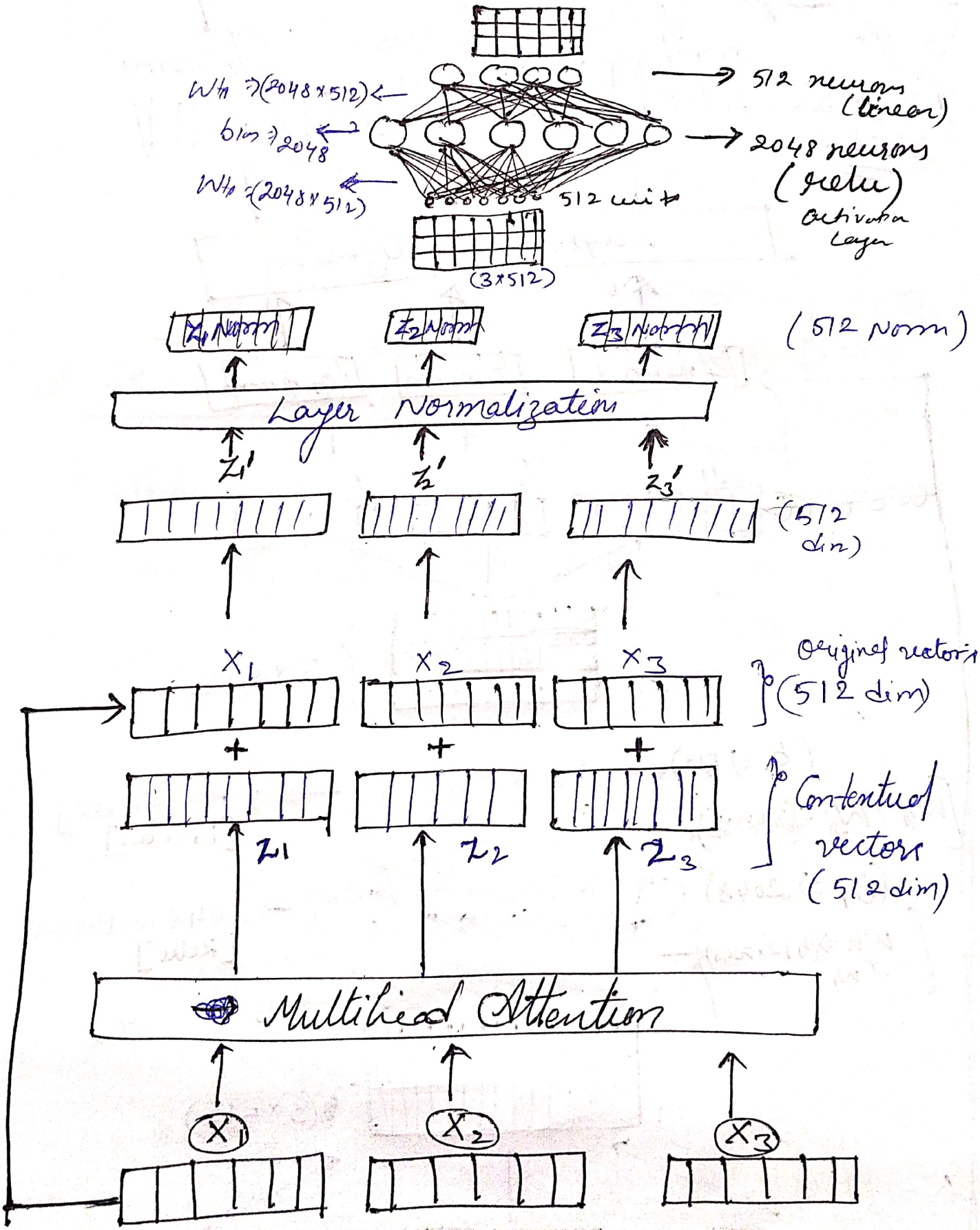


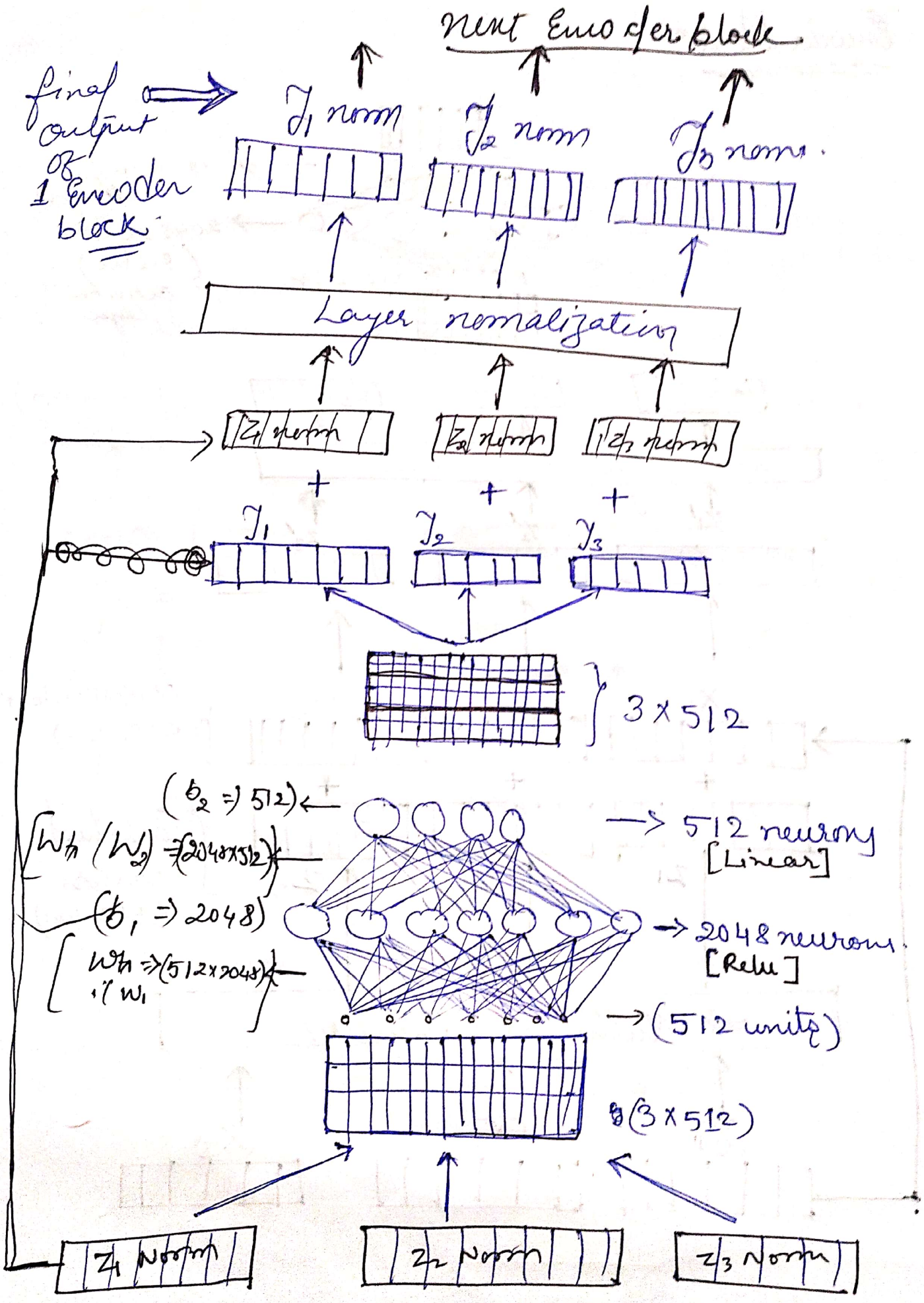
Input block :-



How Are You.

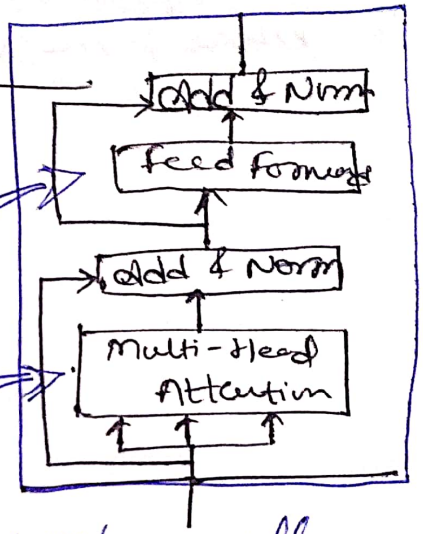
Encoder block:-





Equations :- Neural Network :-

$$[\text{relu}(Z_{norm} \cdot W_1 + b_1)] \cdot W_2 + b_2$$



Why use residual Connections?

Reasons to use :-

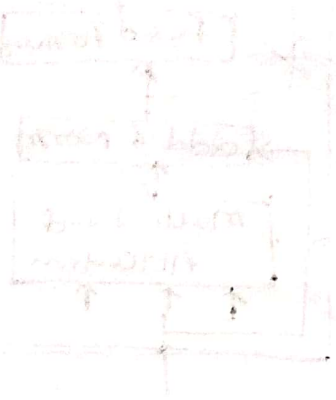
1) Stable training.
2) Original features are always preserved, if multihead attention doesn't work well.

Why to use FFNN??

*) Self-Attention has all the linear operations in it, that is why it doesn't capture non-linear relations that well, so, To capture the non-linear relations well a FFNN is used.

Why use more than 1 encoder block
Why 6 encoder blocks?

↳ ∵ language is a very complex to understand that's why more Encoder blocks help in understanding more & more deeper patterns of languages.



Masked Self Attention.

* **Autoregressive Models :-**

* The Transformer decoder is autoregressive at inference time & non-autoregressive at training time.

What is Autoregression?

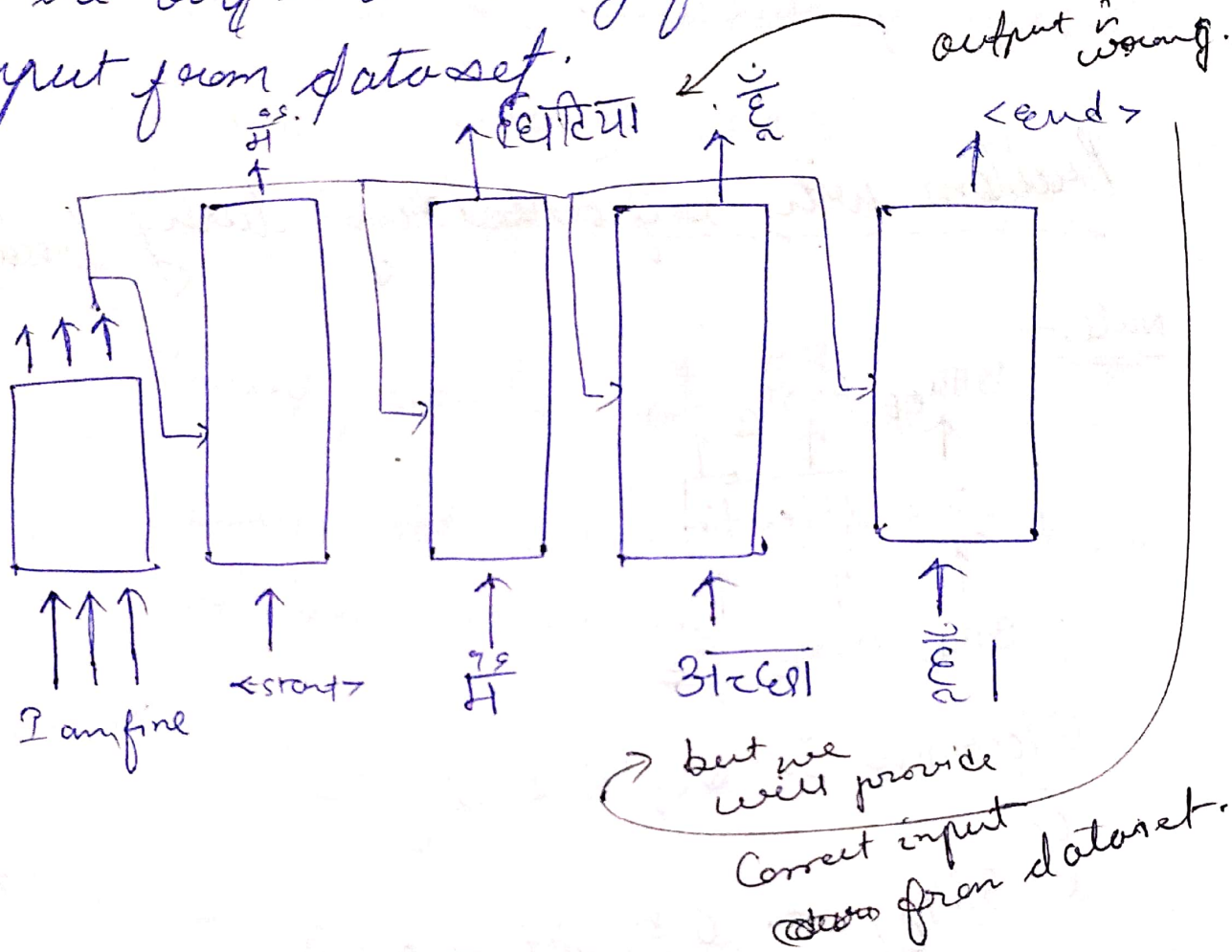
Autoregressive models are the class of models that generate data points in a sequence by conditioning each new point on a previously generated points.

* Inference = prediction.

* **Training :-**

°° During training we have the data-set with us, we need not to send the output of every step to next & as it may cause false training, because, if output become wrong during a step & we provides the wrong output next step, it will be trained on wrong outputs.

that is why during training whatever be the output we always provide correct input from dataset.



* Due to above reasons during training we need to be non-autoregressive.

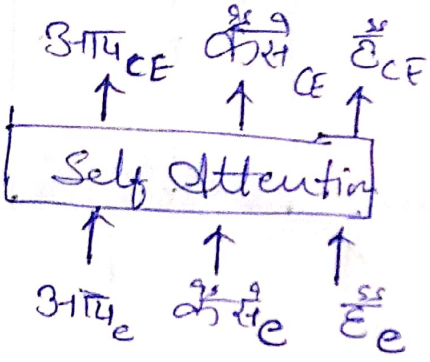
Prediction :-

∵ During prediction, we don't have dataset with us, so that is why we need to be depended upon previous values to predict next word.

* that is we need to be autoregressive during prediction.

Problem with self embedding during prediction

Note!—



According to self Embedding!—

Eg.!—

$$\text{आज}_{CE} = 0.8\ \text{आज}_{e} + 0.1\ \text{कौन से}_{e} + 0.1\ \text{है}_{e}$$

$$\text{कौन से}_{CE} = 0.25\ \text{आज} + 0.9\ \text{कौन से}_{e} + 0.75\ \text{है}_{e}$$

$$\text{है}_{CE} = 0.5\ \text{आज}_{e} + 0.5\ \text{कौन से}_{e} + 0.9\ \text{है}_{e}$$

Now During Prediction, at first time stamp we will only have आज_{CE} (first word) generated, but self attention to make

Contextual Embedding of the word, will demand future words Embedding as well, which during prediction we don't even know.

i.e.

During prediction also,

Current token value is dependent on future token values, which is not possible during prediction,

During training it is possible as we have the dataset available with us.

"That is why simple self-Attention Arch. will not work here"

This issue is called "Data leakage" - as we had future data during training but we don't have it during prediction.

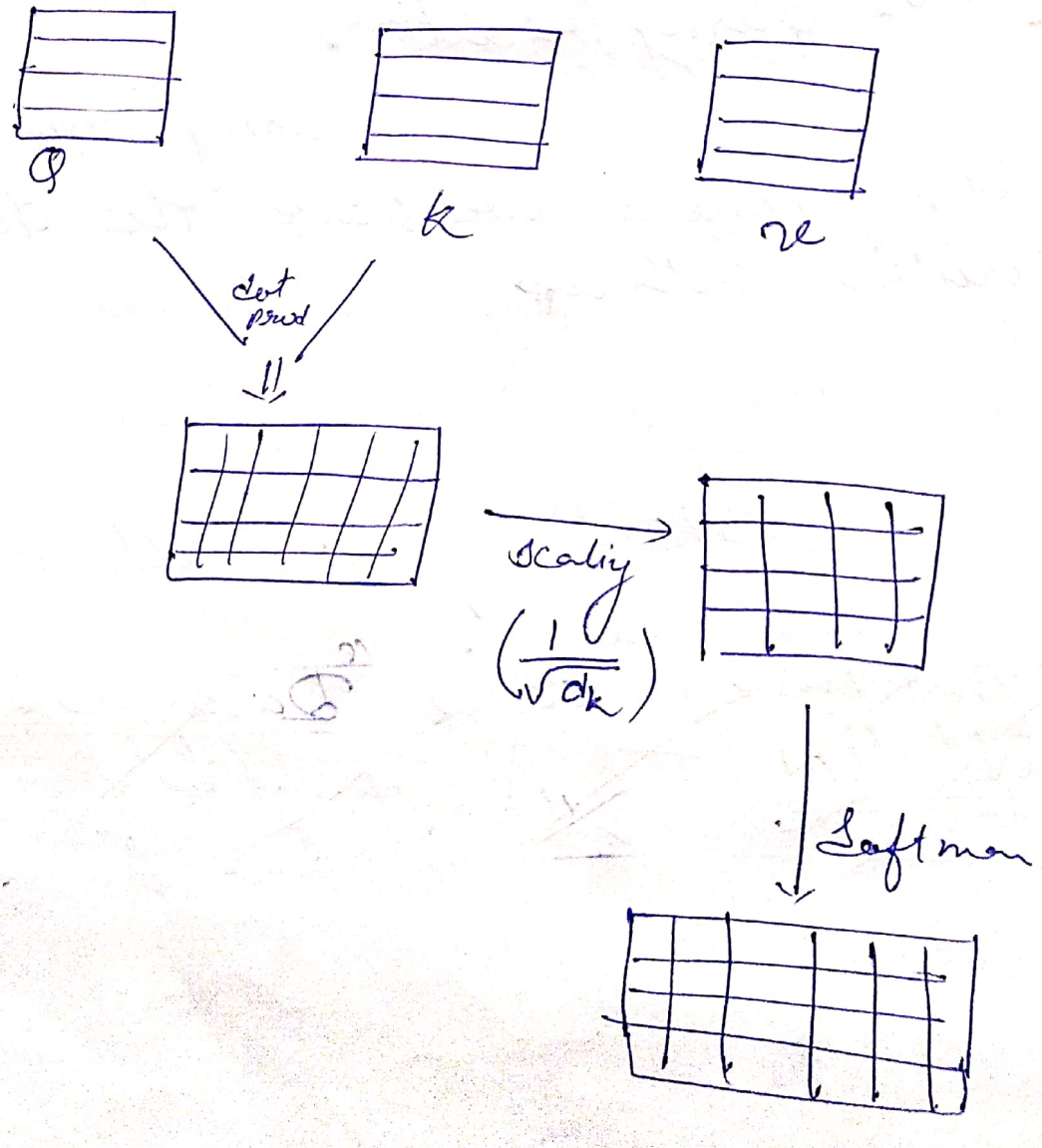
Solution :- Masked Self Attention

*) Data leakage:-

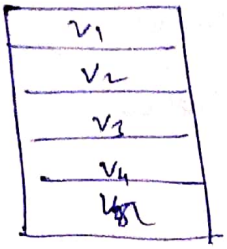
∵ During training, the Decoder model has future tokens available & that is why it was able to perform training but during prediction it is not available, this issue is called data leakage.

Solution => "Masked Self Attention"

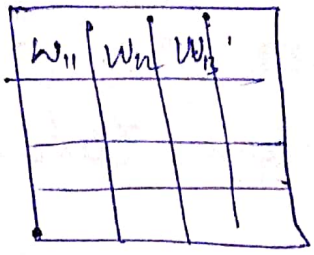
Let:-



Now,



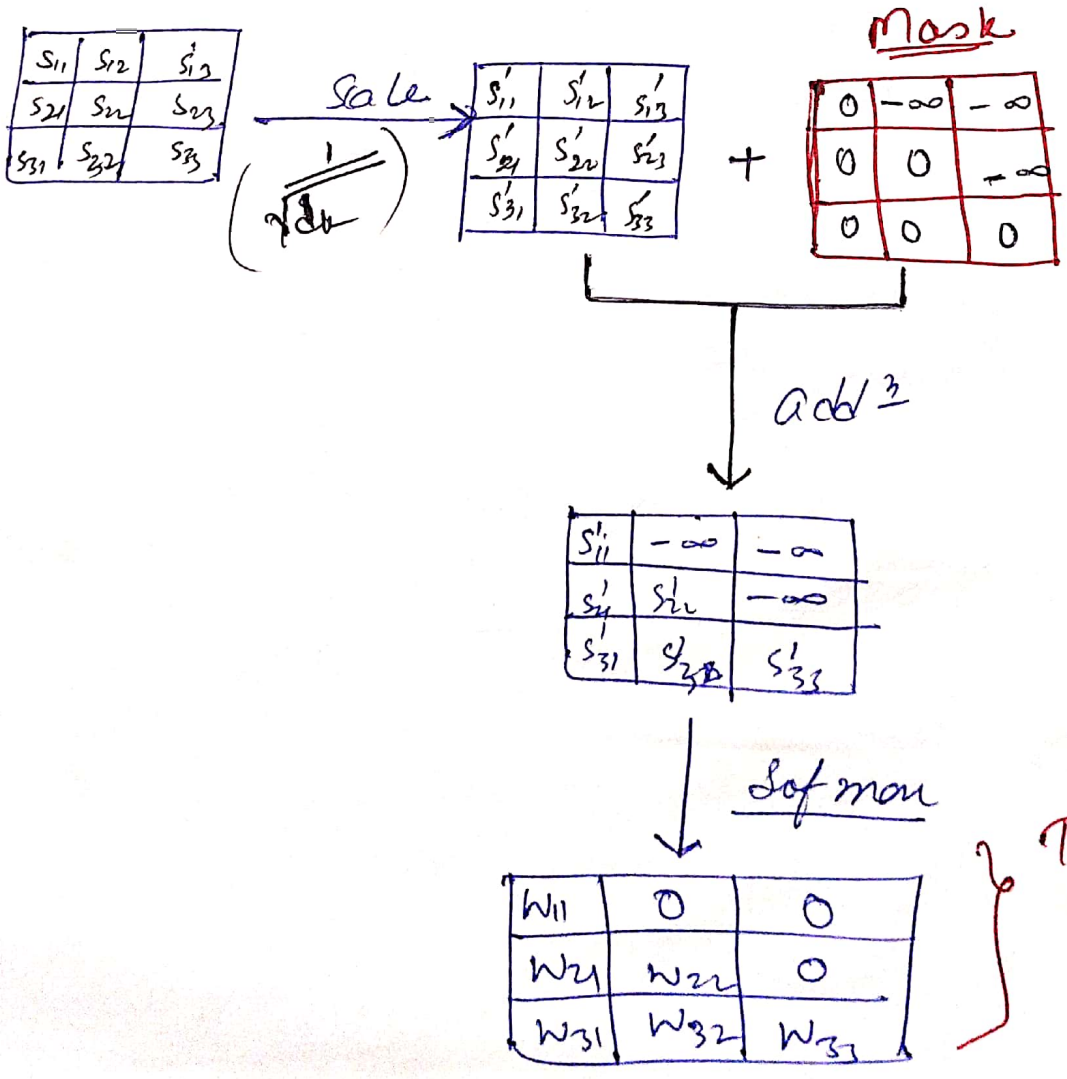
"v"



$$\overline{3114}_{CE} = W_{11} * (v_{3114}) + W_{12} * (v_{\text{previous}}) + W_{13} * (v_{\text{next}})$$

∵ during prediction these two will not be available.

Masked self Attention to avoid future tokens.



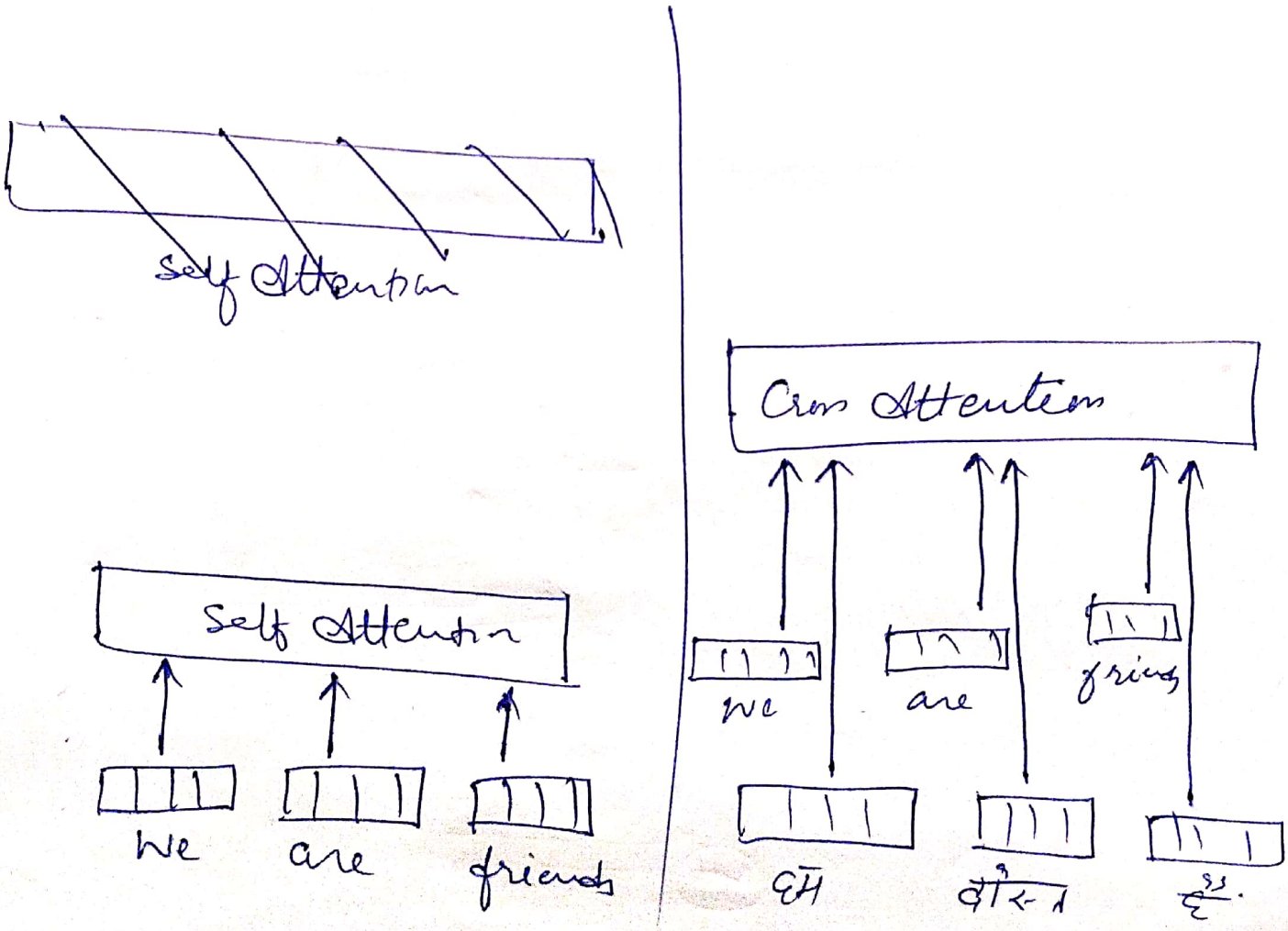
Now, in this matrix we can see the future weights are "0", this will avoid future tokens.

Cross Attention.

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

Self Attention vs Cross Attention

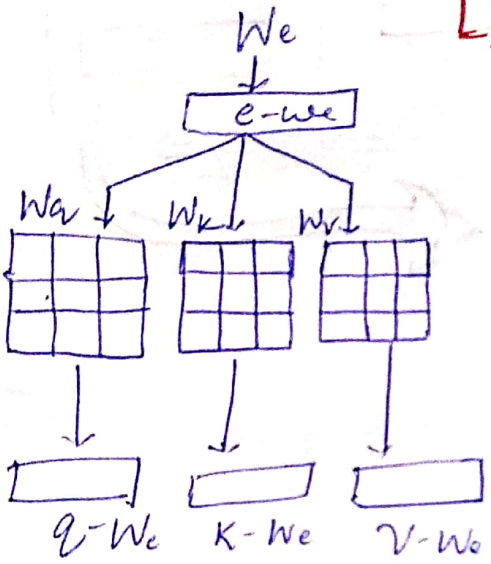
[Input Difference]



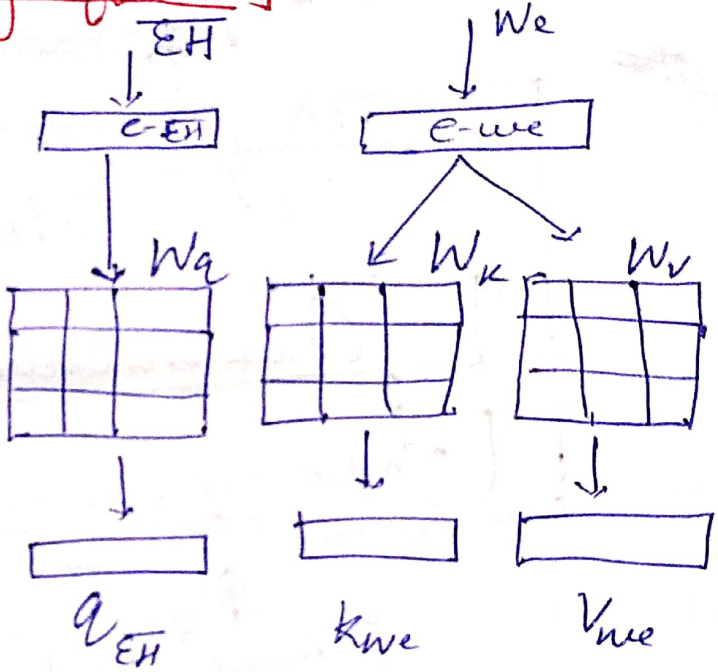
So,

Cross attention need both input & output sequences embedding while self attention need only input sequence.

[Processing Difference]



↓↓
further processing are same.



↓↓
further processing are same.

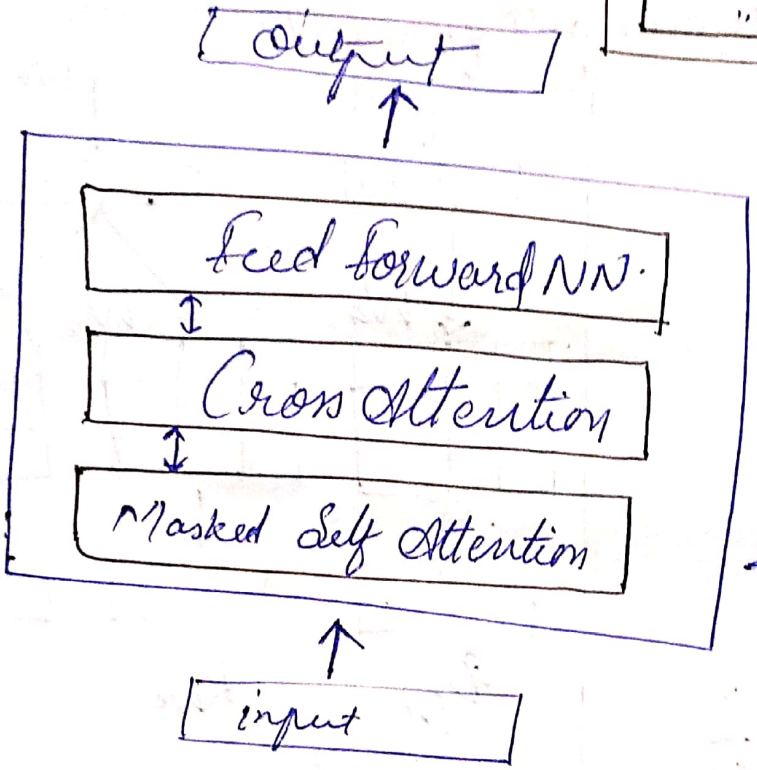
* In Cross Attention, &

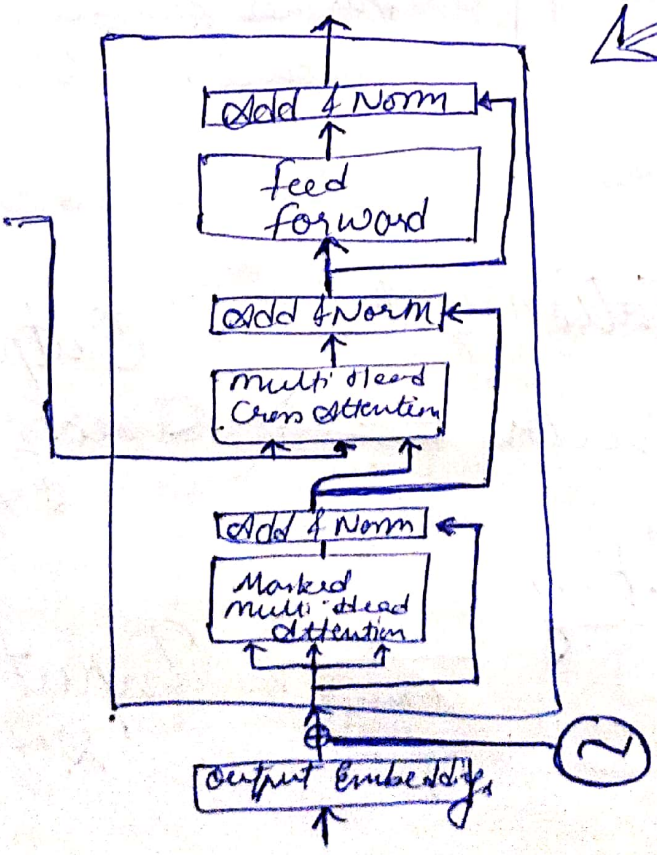
↳* the query vector ⟹ (comes from) Output & Sequence

↳* the value & key vector ⟹ (comes from) Input Sequence

Decoder :-


Encoder
Decoder
Transformer.


Output
Feed forward NN.
Cross Attention
Masked Self Attention
input


Add & Norm
Feed forword
Add & Norm
Multi Head Cross Attention
Add & Norm
Masked multi Head Attention
Output Embeddings
Positional Embeddings.

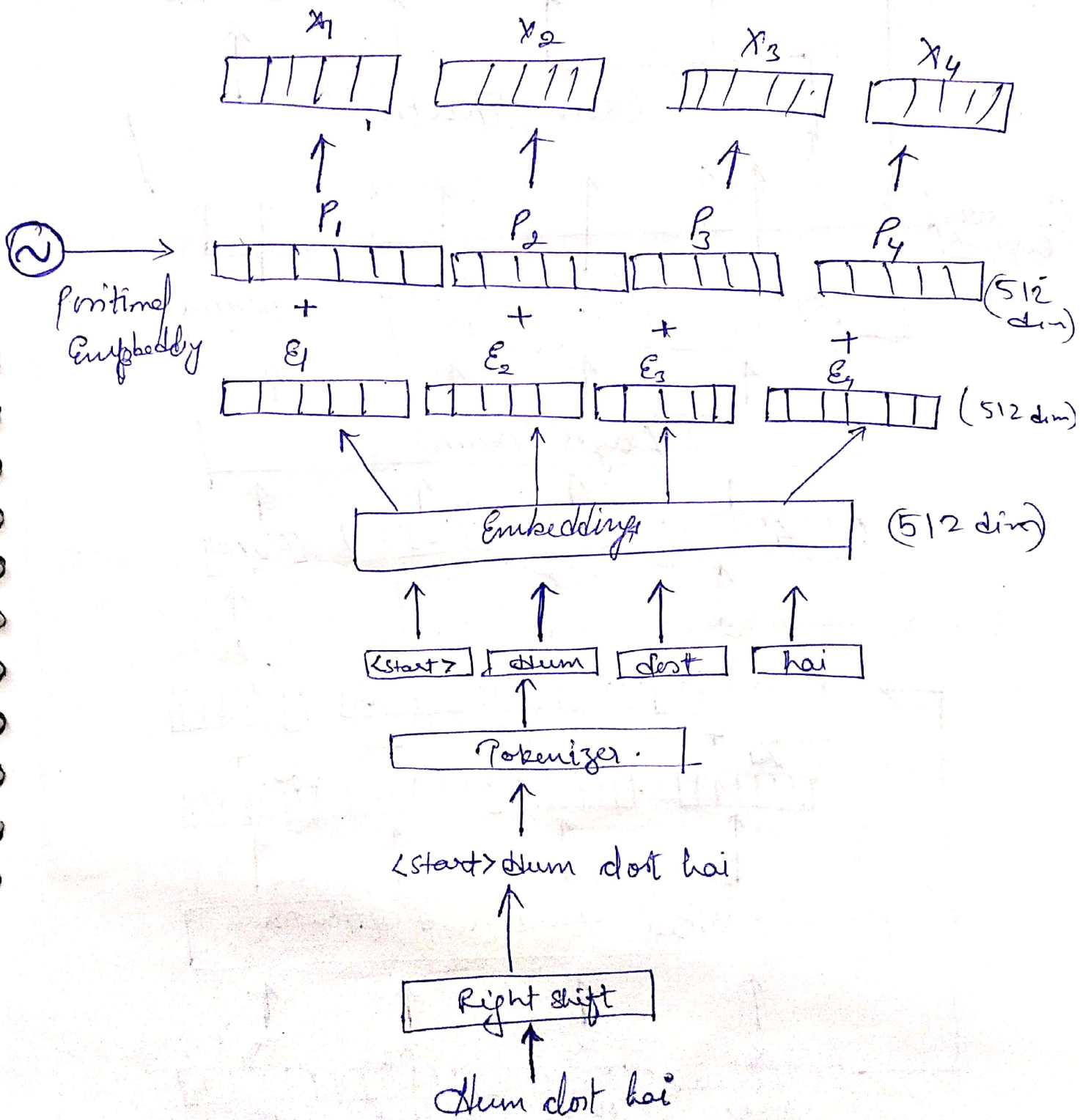
Input Block :-

1. Shifting.
2. Tokenization.
3. Embedding.
4. Position of Encoding.

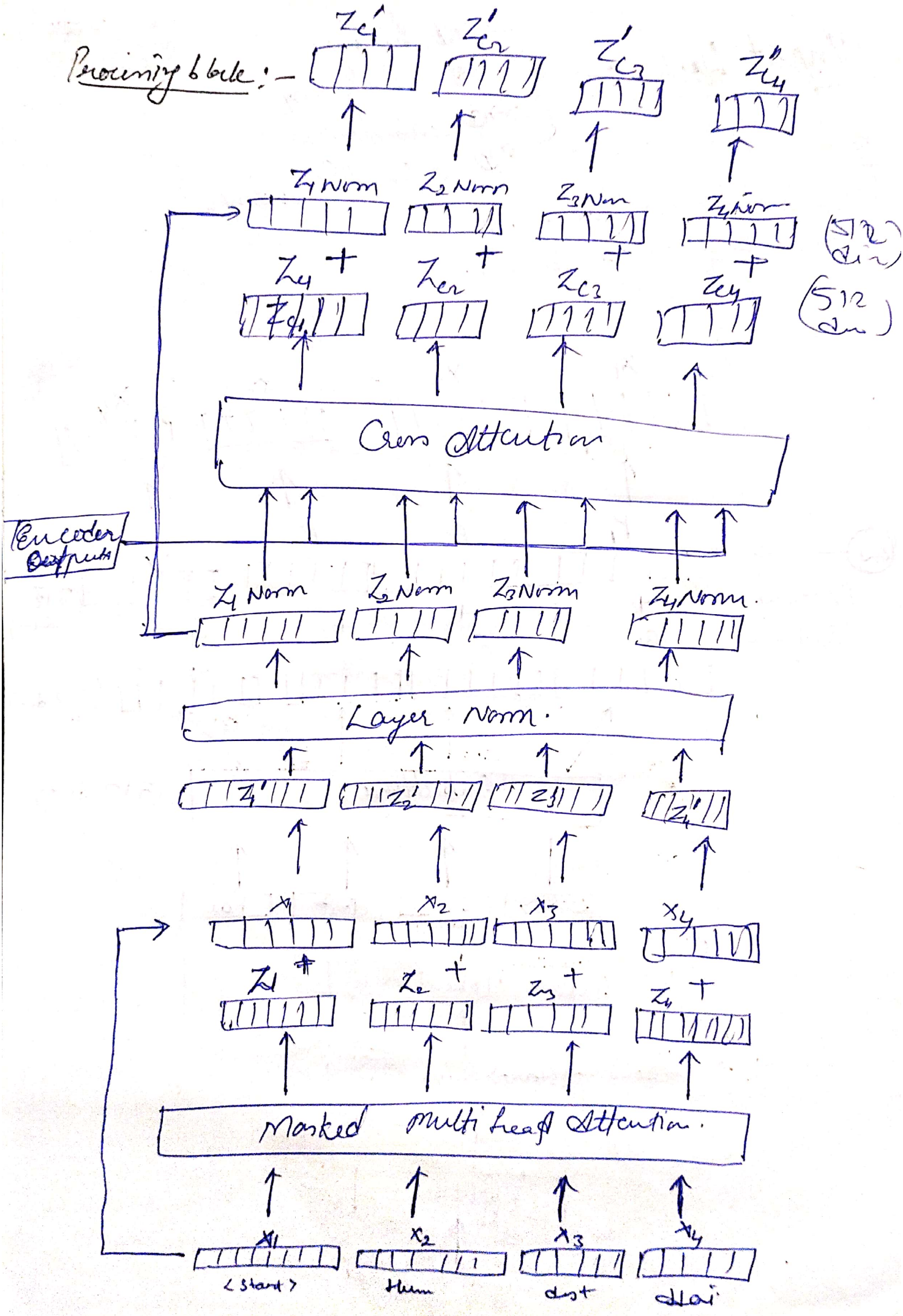
Eg:-

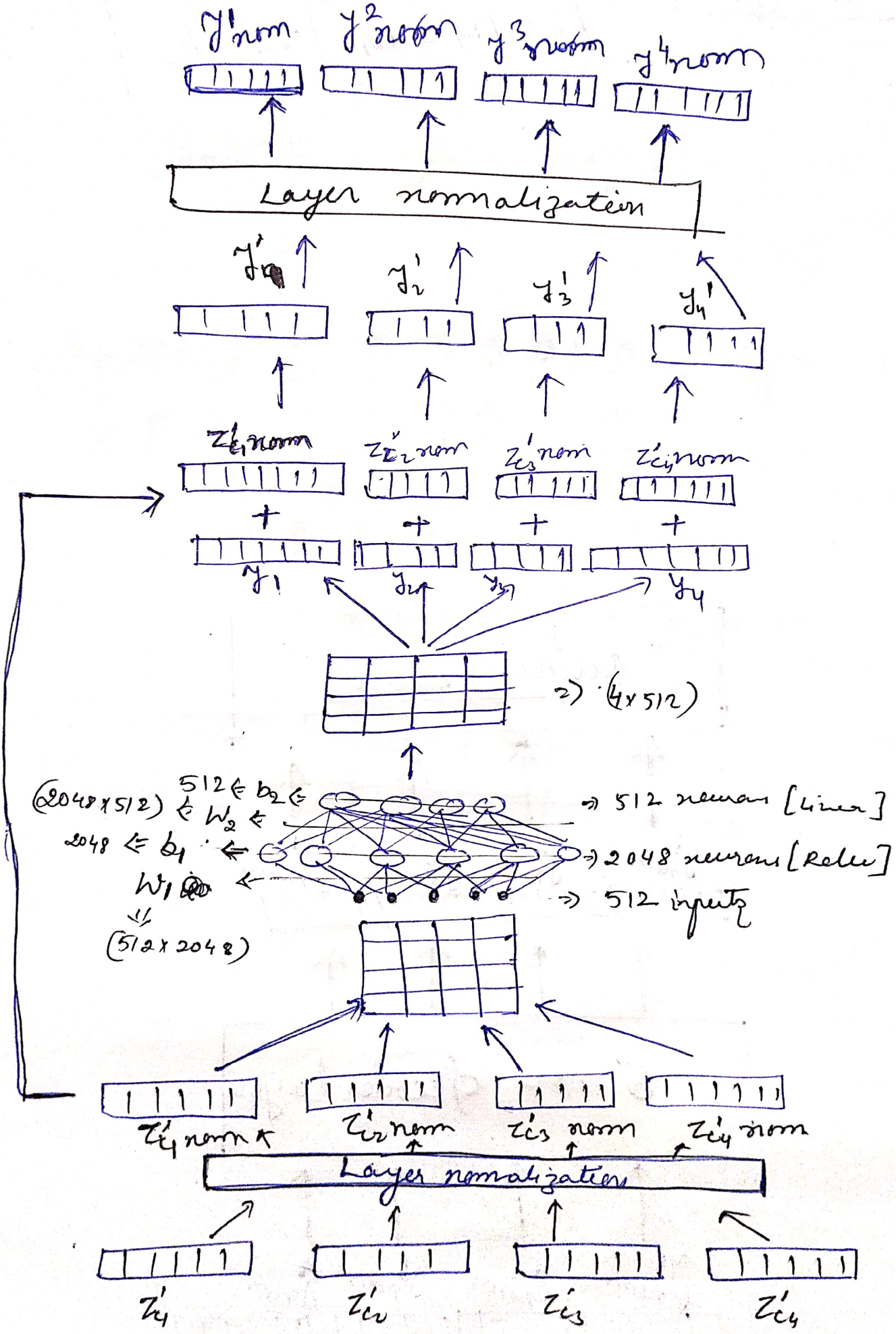
English sentence :- We are friends.

Hindi sentence :- Hum dost hai.



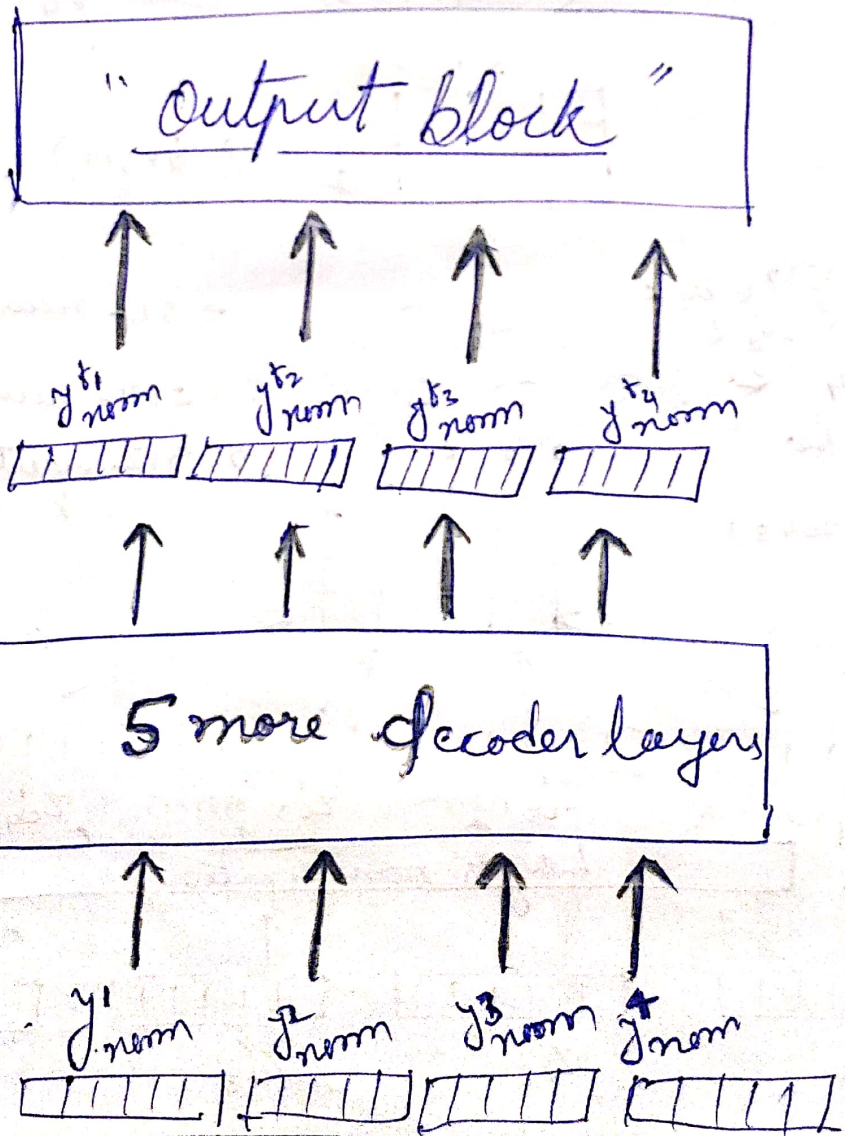
Processing block :-



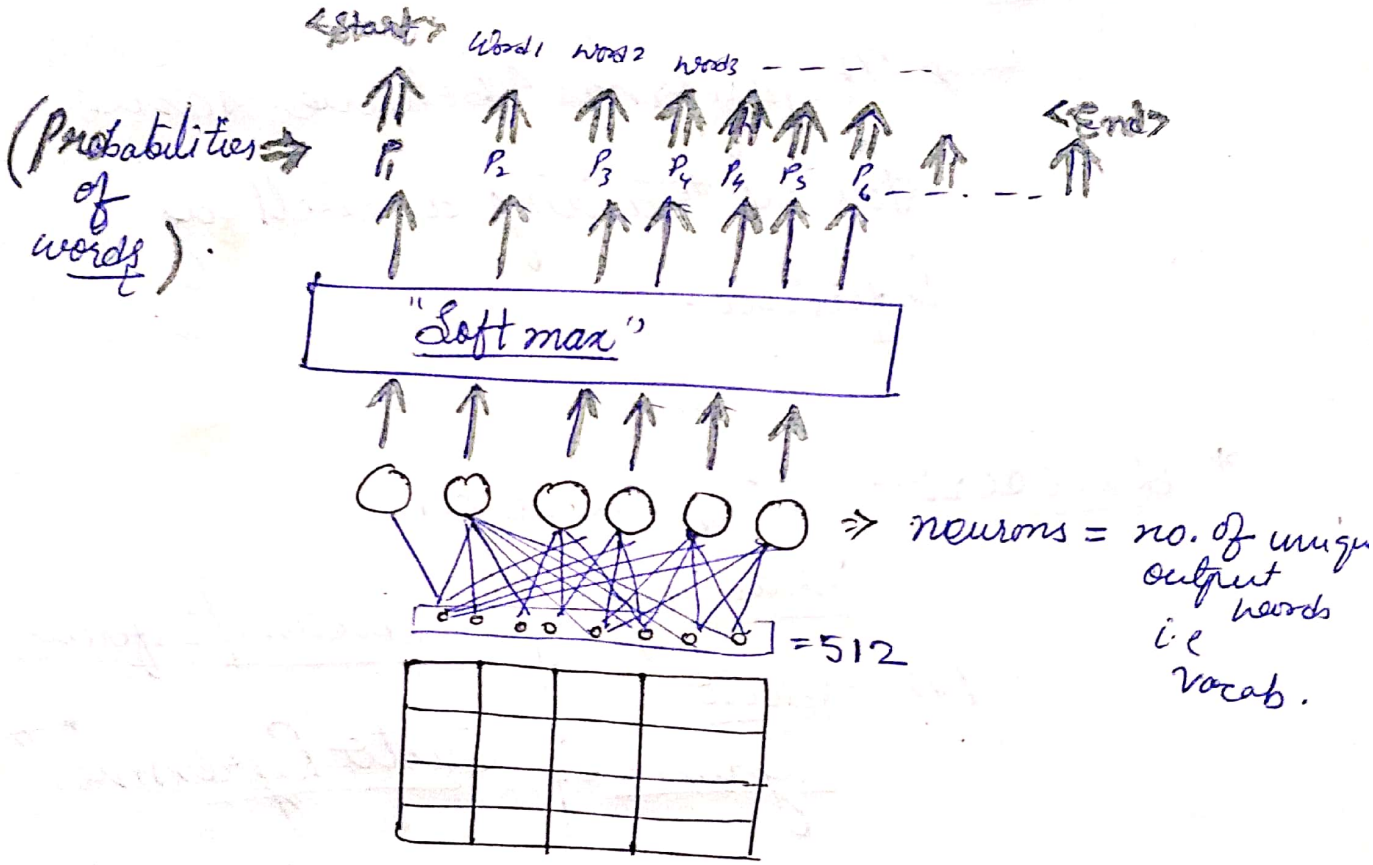

Layer normalization
Layer normalization
=> (4x512)
512 ← b2 ←
(2048x512) ← W2 ←
2048 ← b1 ←
W1
(512x2048)
=> 512 neuron [Linear]
=> 2048 neurons [Relu]
=> 512 inputs

ffNN Equⁿ :-

$$[\text{relu}(W_1 \cdot z + b_1)] \cdot W_2 + b_2$$



Output block :-



Transformer During Inference [Prediction]

* Encoder :-

  ↳ It performs & behave same during training as well as inference.

* Decoder :- It & behave as :-

Training	Prediction / Inference
"Non-Auto Regressive"	"Auto Regressive."