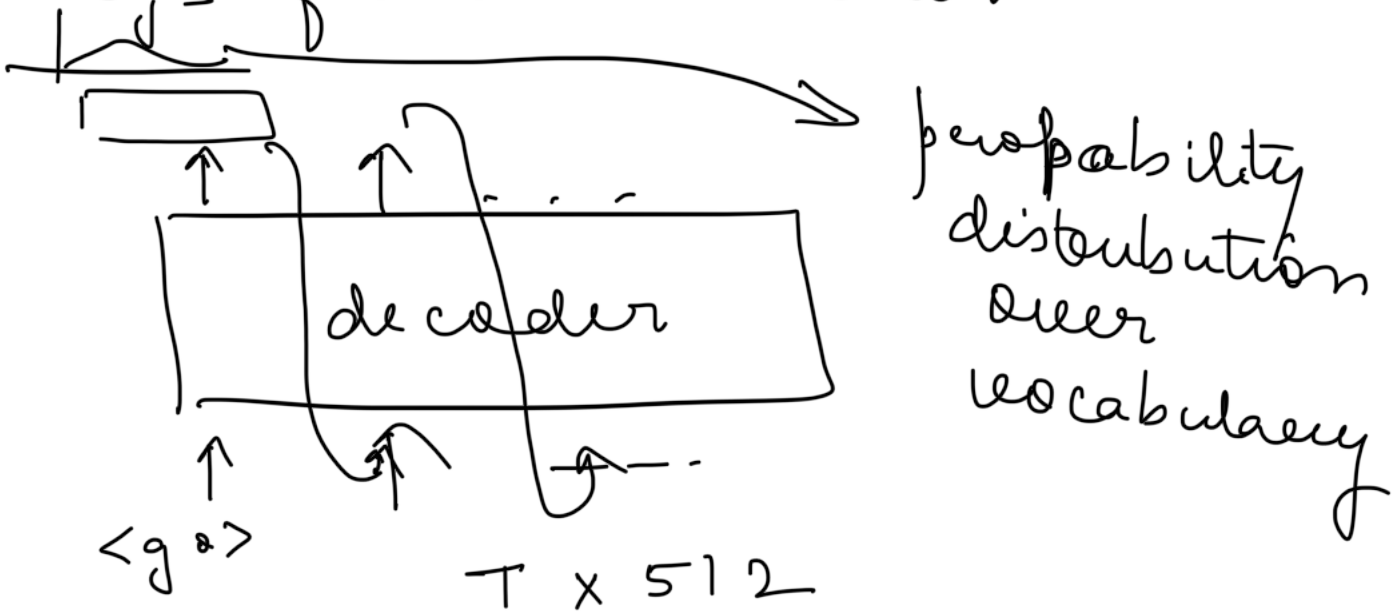
LLM

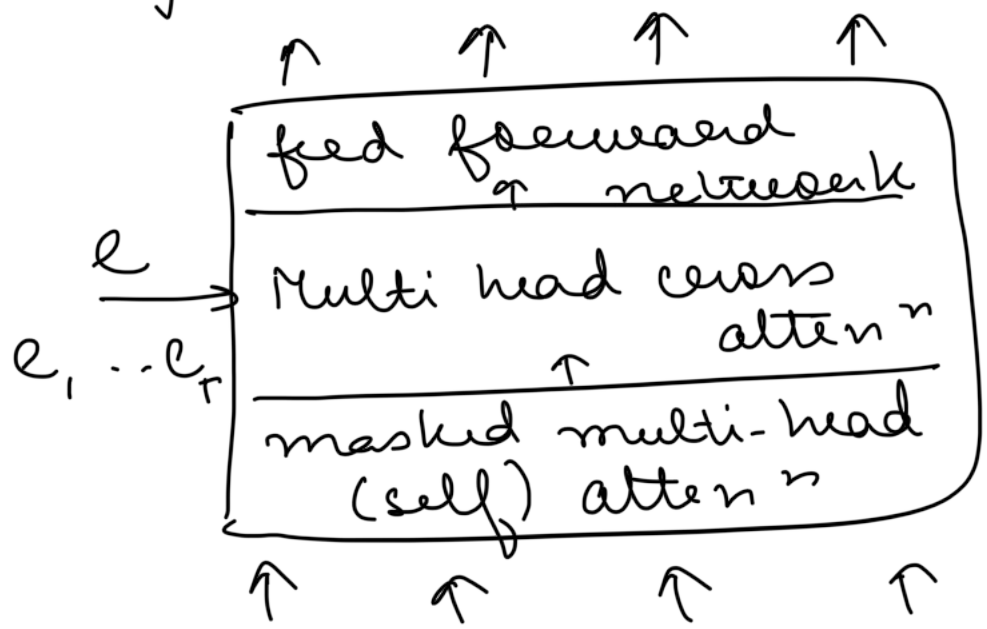
Week-2

L1 Teacher Forcing & Masked Attn^{n}

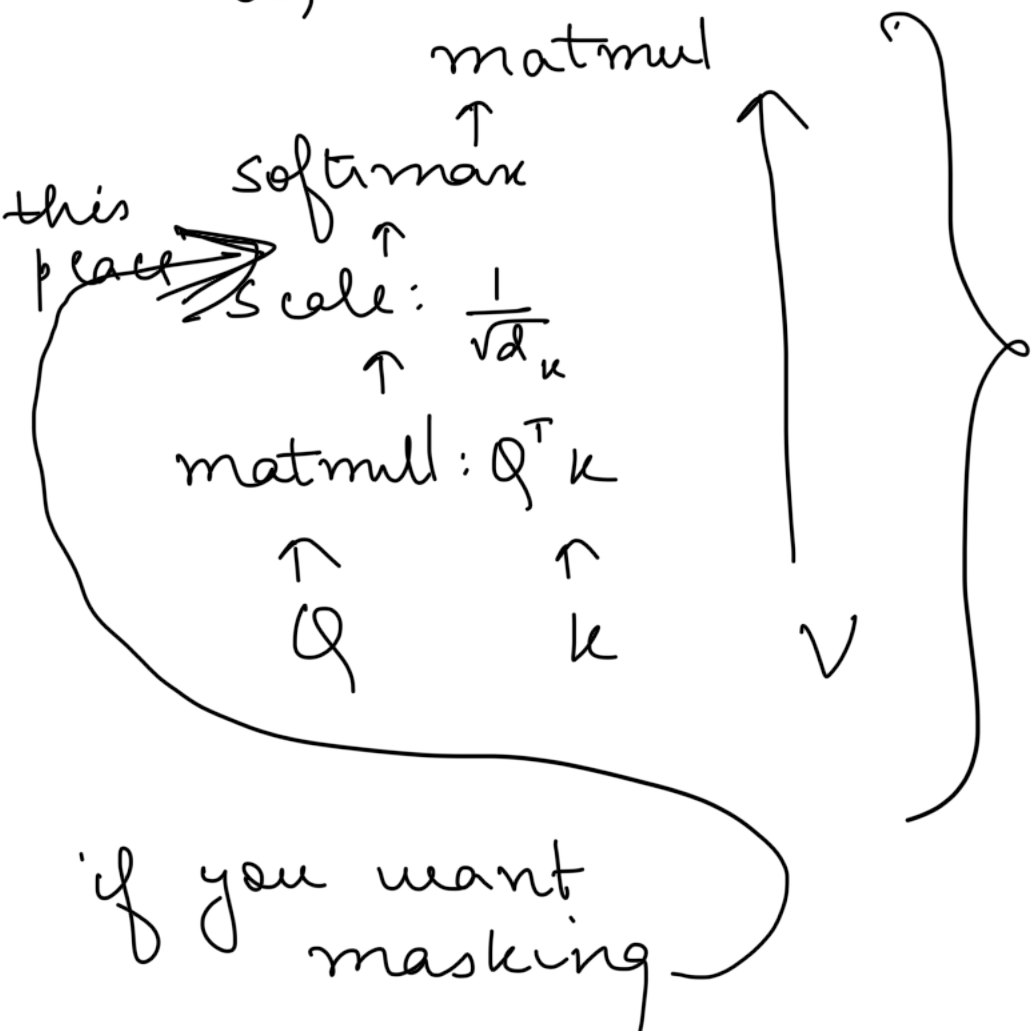
- decoder → we had received the e_i's from the encoder



- again decoder is a stack of N layers. Each layer has 3 sub-layers.

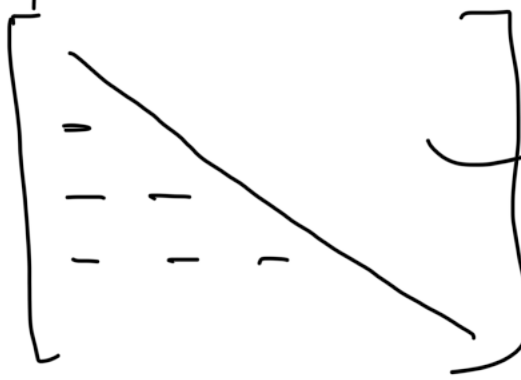


- if you see there is problem, there is a possibility that your first word itself is wrong. Then further, thing will be surely garbage.
- Again, you have both eng. & tamil labelled data, feed it. Then you don't need any encoder.
- feed actual outputs (teacher forcing)
- don not feed everything, do masking on other objects.
- Recall,



- masking to implement the teacher forcing approach during training.

Remember, never use teacher forcing during inference. Then, the decoder directly acts as a auto-regressor.



→ it does masking on the other words.

$$Q_1 = W_{Q_1} H \;,\; K_1 = W_{K_1} H \;,\; V_1 = W_{V_1} H$$

$$Q_1^T K = A$$

Assign 0 wts. $a_{ij} = 0$ for the value vectors v_j to be masked in eq.

$$\begin{bmatrix} & A & \\ & & \end{bmatrix}_{T \times T} \Rightarrow \begin{bmatrix} -\infty & & \\ & M & \\ 0 & & \end{bmatrix}_{T \times T}$$

$$Z = \text{softmax}(A + M)\, V^T$$

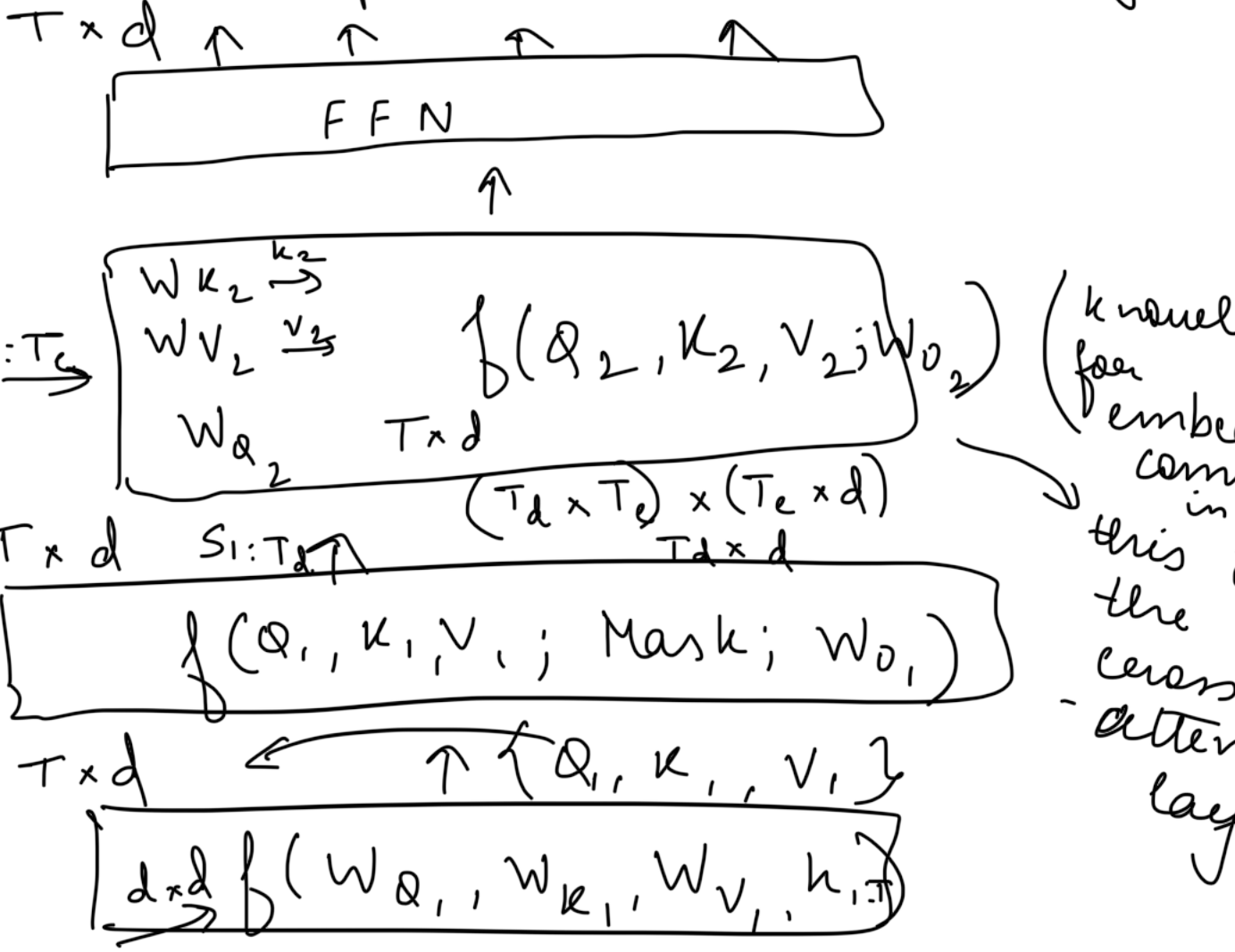
$A + M$

$\Downarrow$

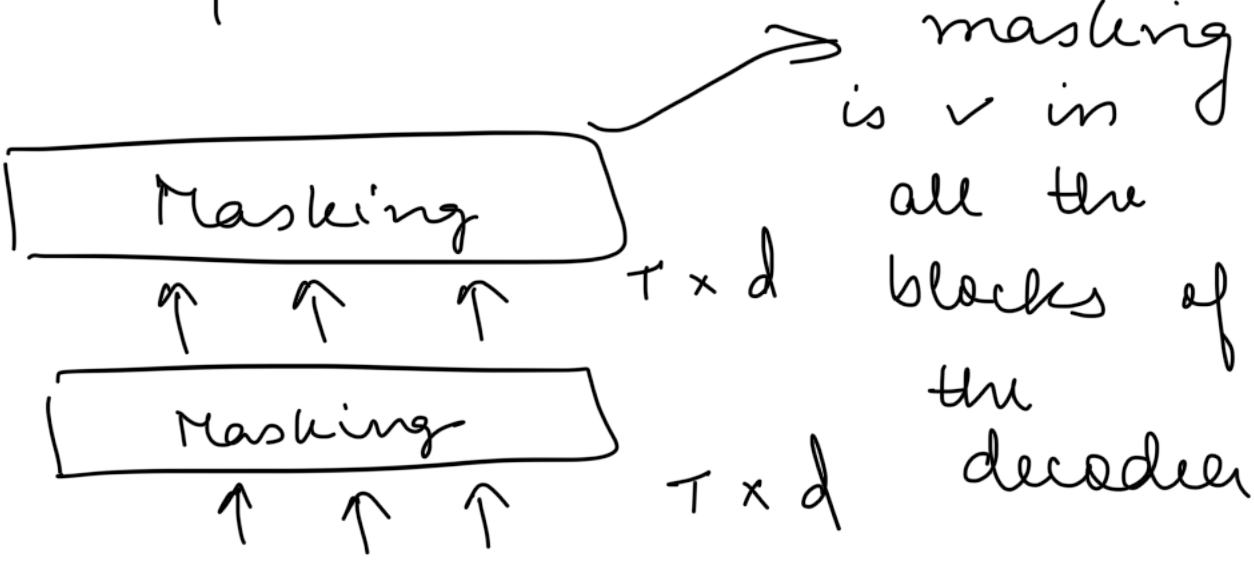
$$T \begin{bmatrix} q_1 k_1 & -\infty & -\infty \\ q_2 k_1 & q_2 k_2 & -\infty \\ \vdots & \vdots & \vdots \end{bmatrix}$$

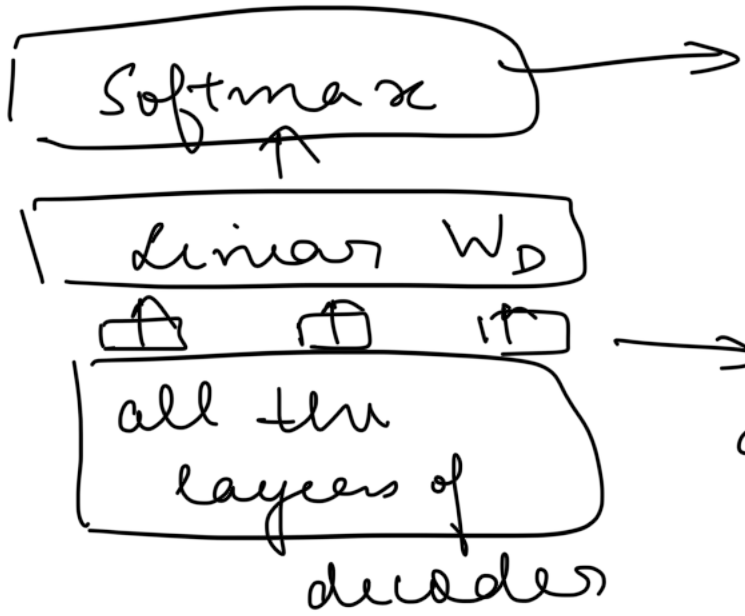
T

L2 Zooming into the decoder layer



- similarly it goes





$$\underset{|V| \times d}{[W_D]} \underset{d \times 1}{[\ ]} = \underset{|V| \times 1}{[\ ]}$$

- this is how the output layer will be. This calculation is meant for linear W_D layer.

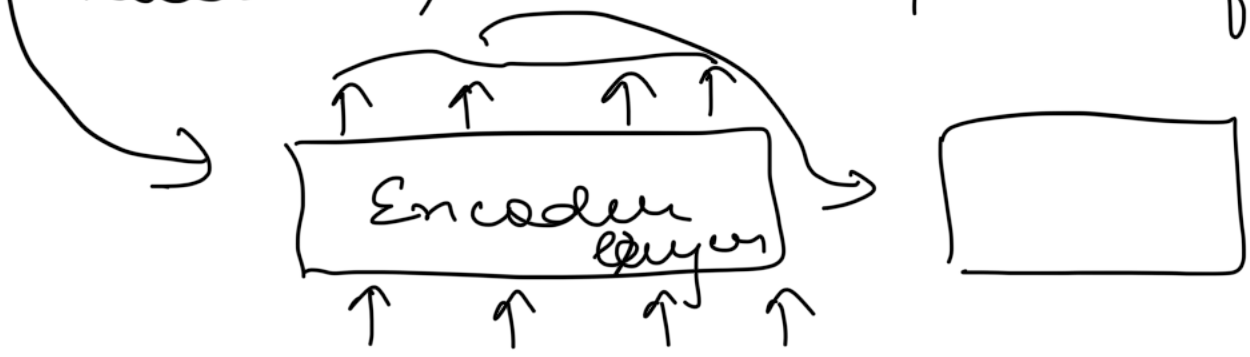
But we don't need a $\{|V| \times 1\}$ kind of a vector. We want probability distribu^n..., I'll use softmax at the last layer.

L3 Positional Encoding

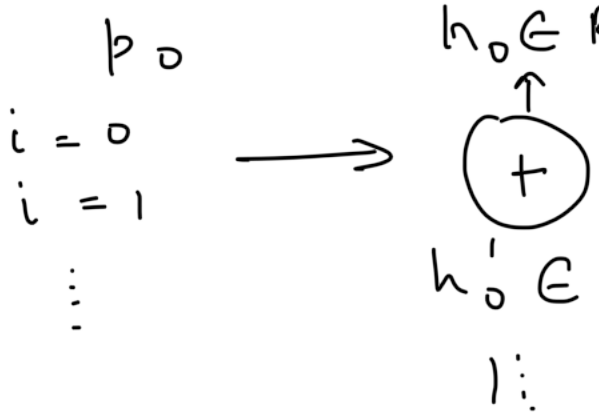
- The pos^{n} of words in a sentence was encoded in the hidden states of RNN-based seq-to-seq. models.

$$z_i = \sum_{j=1}^{4} \left(q_i^T k_j \right) v_j$$

- currently, the words does not have any sense of positions. But relative position of words is needed.
- Output from the self-attn^{n} is permutation invariant. So it is necessary to encode posi^{n} info.



- how do we do this?



it could be a constant vector (all elements are of constant value)

for p_j)

these nos. are typically very large for initialization

- can we use one-hot encoding ?

  $j = 0, 1 \ldots T$

  ↳ may loose info., problem of sparse matrix.

- learn all embeddings for all possible positions ?
- these methods are not suitable for dynamic sentence lens.
- in one hot-coding all 1s will have a 'just same' distance of $\sqrt{2}$. But, this should not be so.

L4 Sinusoidal Encoding

- embed a unique pattern of features for each pos^n j and the model will learn to attend by the relative pos^n

but how do we generate such features?

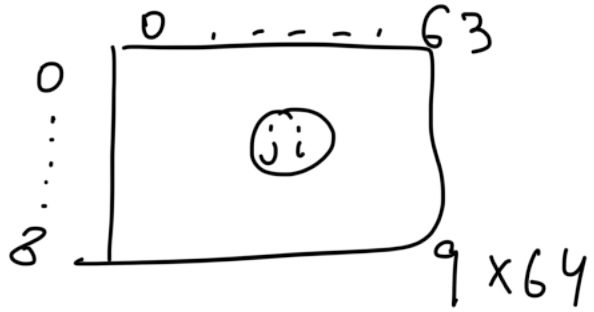
$$PE_{(j,i)} = \begin{cases} \sin\left(\frac{j}{10000^{2i/d_{model}}}\right) & i = 0 \ldots 255 \\ \cos\left(\frac{j}{10000^{(2i+1)/d_{model}}}\right) & i = 0 \ldots 255 \end{cases}$$

pos^n ∈

d_{model} = dim of transformer model = 512

- for a fixed pos^n j, we use sin func^n when i is even, & cos if i is odd

$j = 0 \ldots 8 \qquad i = 0 \ldots 63$



- some obs. —
  1. dist. increases on the left & right of 0.
  2. sym. abt centre of sentence
- PE func^n solves our problem.

L5 Batch Normaliza^n

- roughly, we can say transformer architecture is composed of atten^n layers & hidden layers.
- in such a network, how will the gradient flow will take place?
  by using residual connec^{ns}
- how to sped up? → Normaliza^n

- Batch Normalizaⁿ at l^{th} layer

$x_i^j \rightarrow$ activaⁿ of i^{th} neuron for j^{th} training sample.

accumulator with l^{th} layer stores the activaⁿ of batch inputs

$$\begin{matrix} x_1^m & \cdots & x_1^2 & x_1^1 \\ \vdots & & \vdots & \vdots \\ x_5^m & \cdots & x_5^2 & x_5^1 \end{matrix}$$

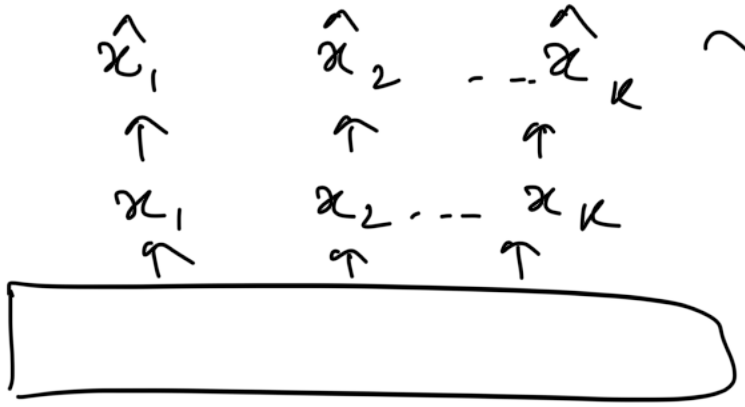
$\longleftrightarrow$ accumulator

- it is generally known that we do standardizaⁿ on the features while in ML or DL.

$$\mu_{i=2} = \frac{1}{m} \sum_{j=1}^{m} x_2^j$$

$$\sigma_i^2 = \frac{1}{m} \sum_{j=1}^{m} (x_2^j - \mu_i)^2$$

$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$ } these are the standardized values.



→ these quantities are also differentiable (no extra wts. or parameters are added here.)

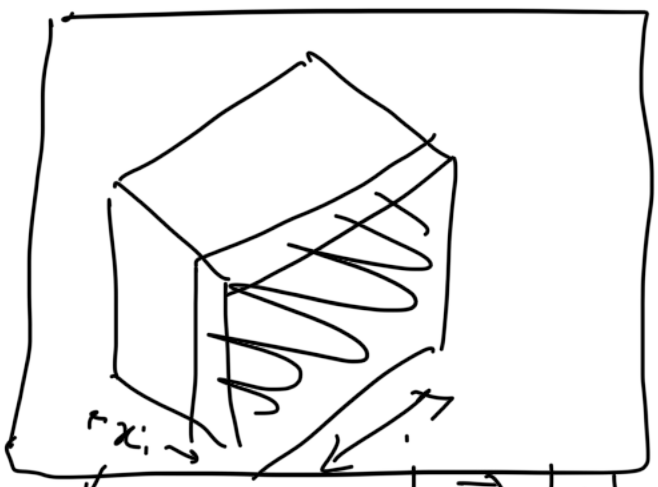
- we are calculating the μ & σ from a batch (m).
  Why? we just passed 32 or 64 samples ⇒ hence, batch normalize.
- why did you standardize?

$$\hat{y}_i = \gamma \hat{x}_i + \beta \rightarrow \mu$$

γ, β ⇒ new params

σ ⇒ then standardized (older one)

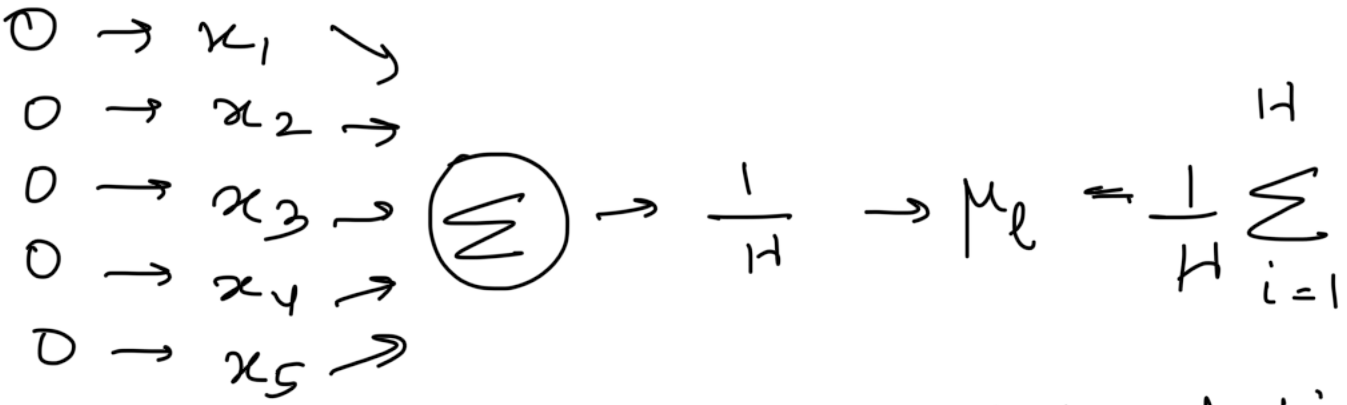
- there 3 variables ⇒ l, i, j



⇒ easier for visualize

L6 Layer Normalizn

- yes we can apply batch normalizn to transformers. The accuracy depends on the size of m.
- Naive use of BN leads to performance degradn in NLP tasks.
- layer normalizn at l^{th} layer.



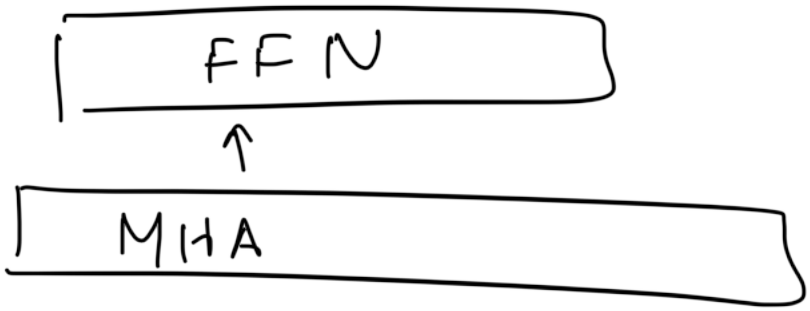
take the avg. across outputs of hidden units in the layer. ∴, the normalizn is indep. to no. of samples in a batch.

$$\sigma_l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i - \mu_l)^2}$$

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \varepsilon}}, \quad \forall\, i$$

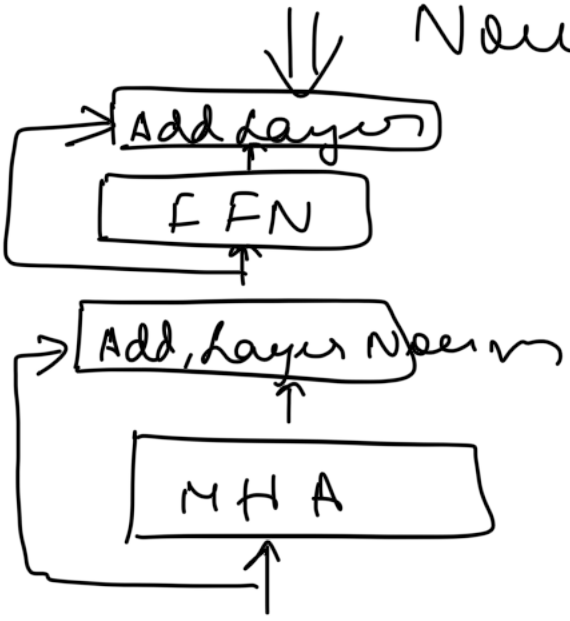
$$\hat{y}_i = \gamma\, \hat{x}_i + \beta$$

Encoder



MHA
⇓
multi-head attenⁿ

Now



add a residual connectⁿ & layer norm block after every MHA, FFN, cross attenⁿ block.

$\hat{x} = x + \hat{h}$ ⇒ in the add & normalize error.

- similarly, we have these connections in the decoder as well.