RECURRENT NEURAL NETWORK.

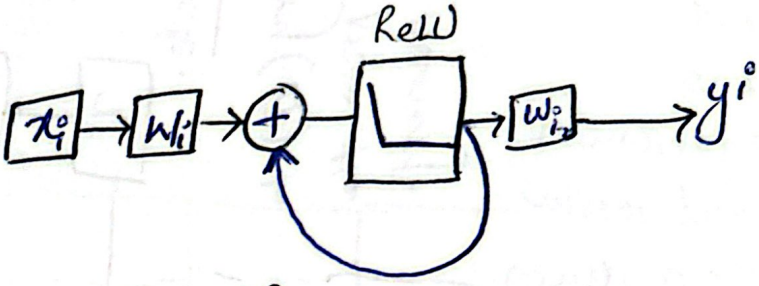
→ is a class of Neural network designed to model sequential data. It has a memory mechanism mean its output depend on current state plus previous state

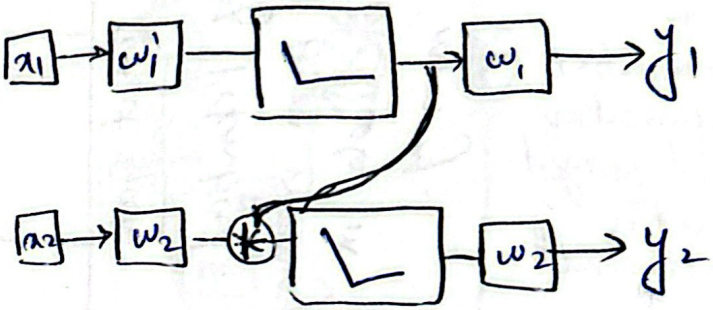
Hidden state update.

$$h_t = f(W_x h_t + W_{hh} h_{t-1} + b_h)$$

Output.

$$y = g(W_{hy} h_t + b_y)$$





ISSUES :-

→ Vanishing gradient: too low steps cause reach max steps without finding optimum

→ Exploding gradient >1 : too big steps cause divert from optimum.

→ Difficulty with long-term dependencies.
captures short term context
struggles over long sequences.

→ Training Insufficiency
sequential in nature
can't use GPU etc.

* The LSTM / GRU provides a solution.

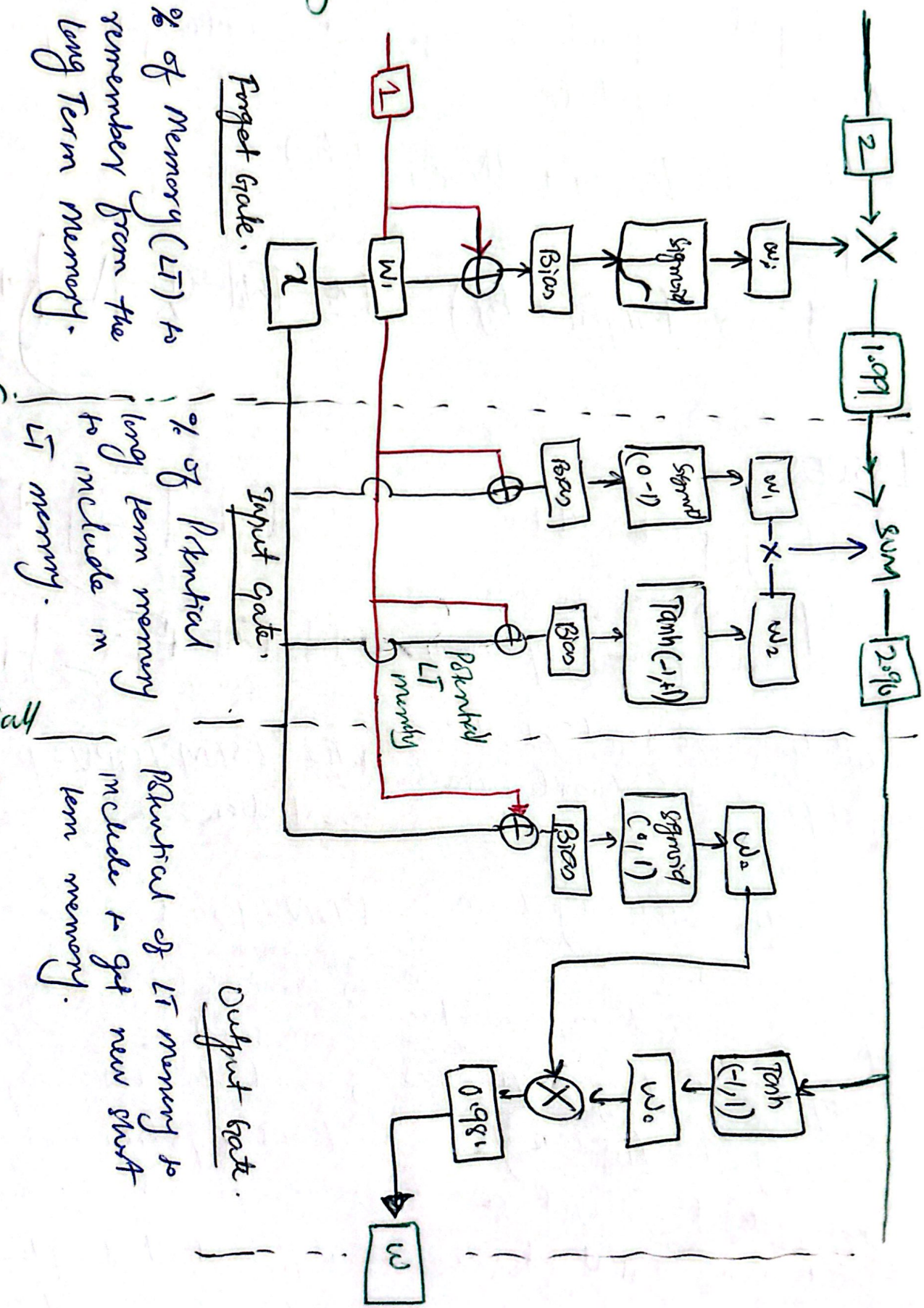
RNN Pro's

- same set of weights & biases are used.
- good for time dependent tasks (speech, text etc)
- can process sequences of differential length
- can be configured for different Input/output mappings

LONG - SHORT TERM MEMORY

Sigmoid.
Take x and fixed it to y in b/w (
Tanh
Take x → y (-1, 1)

cell - state → long term memory
hidden state → short term memory



- LSTM learns long term dependencies and resolve RNN issues.
- can remember information and forget what is irrelevant
- still sequential mean no GPU,
- more complex then RNN
- Memory intensive due to gates.

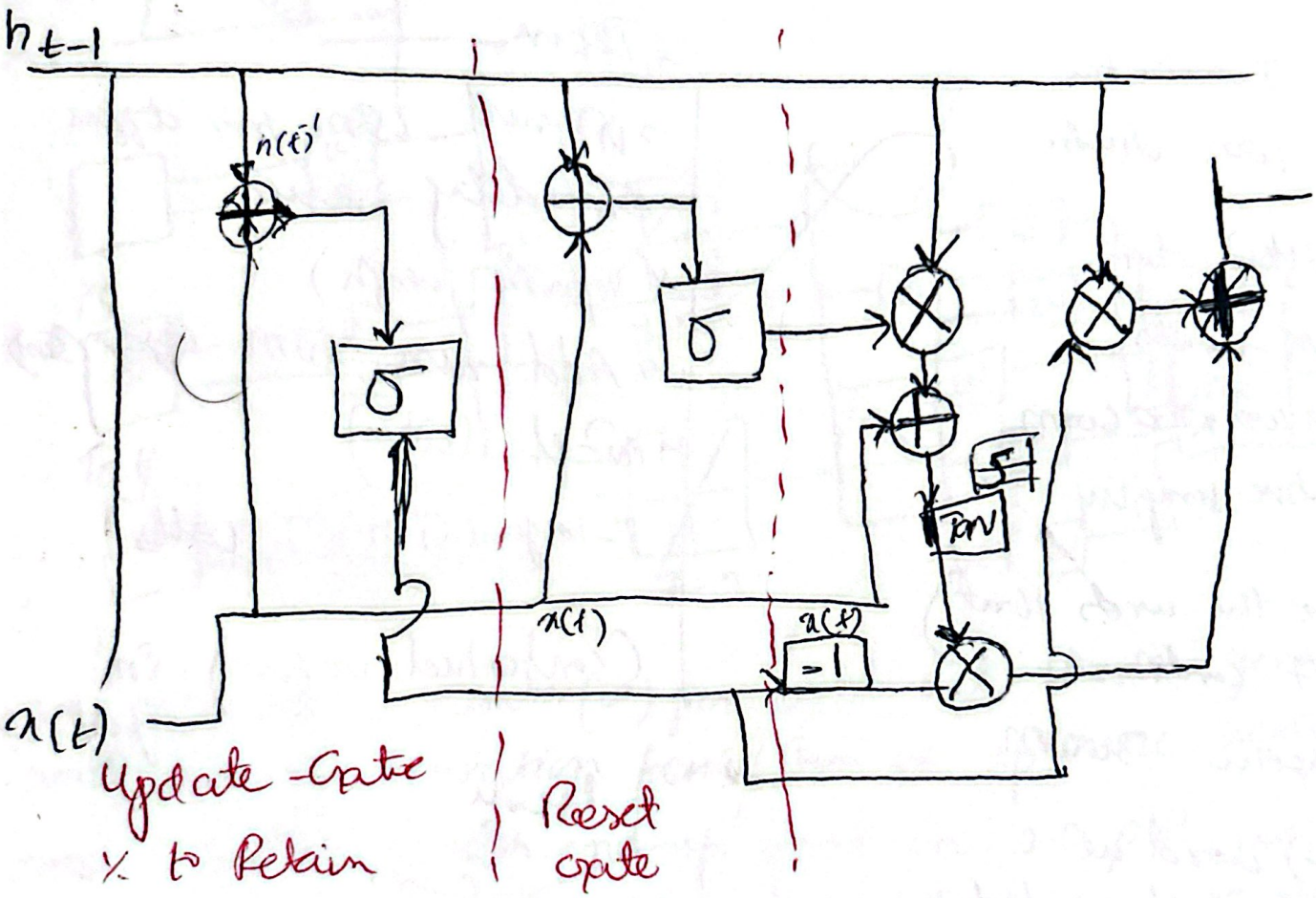
Gated - Recurrent Unit (GRU)

Type of RNN designed to avoid gradient issues and learn long-term dependencies in sequential data. → Similar to LSTM but faster. (best for small sequence) LSTM for long.

→ combines long & short term memory

Update → How much past memory to keep

Reset → How much to forget



Deva

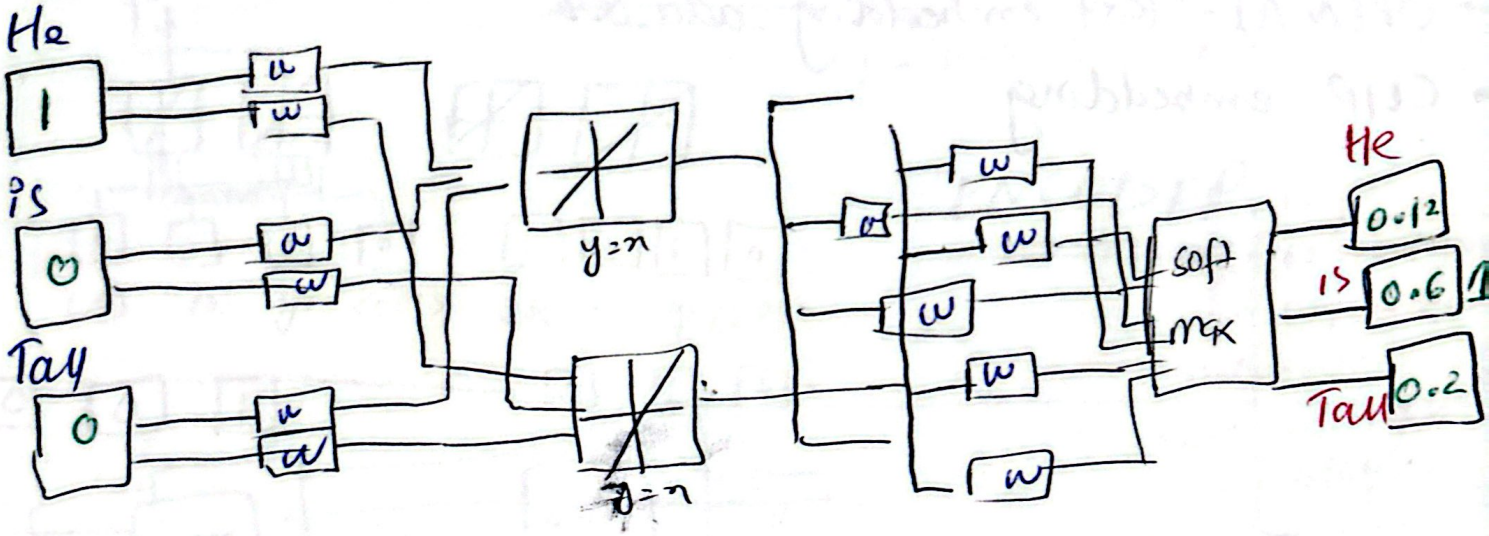
WORD - EMBEDDINGS

Word embedding is a vector representation of word in a continious vector space.

Prose-

- Dense and low-dimensional representaton
- captures semantic/syntactic meaning.
- Enable transfer learning, better generalizaton vs one hot vector

Example by NN:



→ We associate 2 number(w) per word but 100 in general and 100 of activation fcn's. Then we optimize weight and similar words end-up near in 2-D plot. we associated various to capture the context (+ve, -ve, etc)

word 2Vec:

Usually, we train over whole Wikipedia etc

1) CBOW:

we give surrounding words to predict middle ones mean predict current word given context words.

2) Skip-gram:

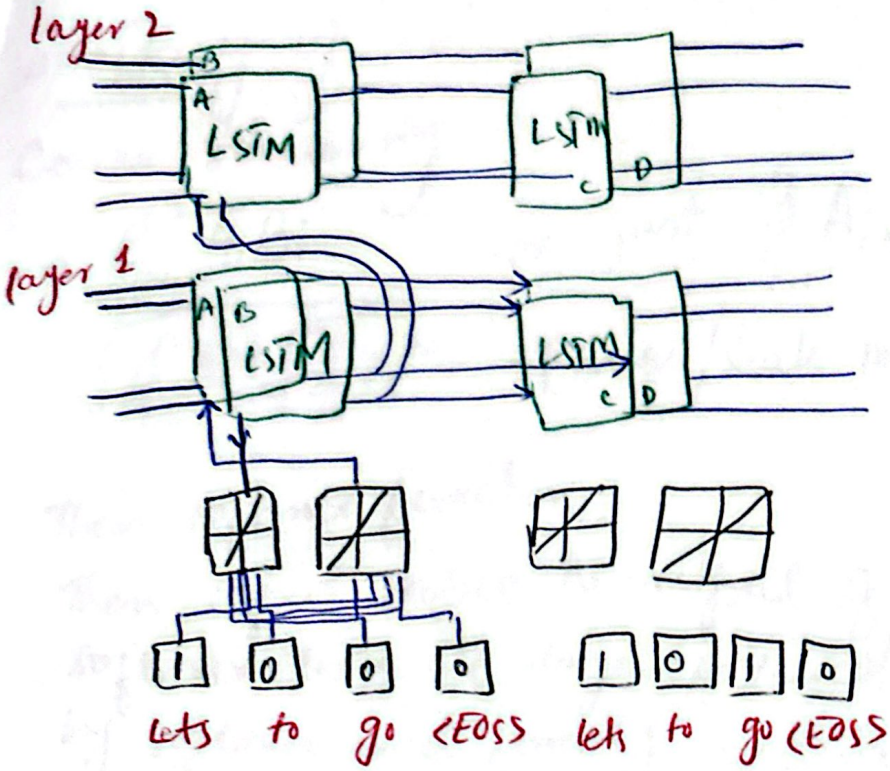
Vice-versa

SEQ-2-SEQ: (Encoder-Decoder)

s a NN design to transform one sequence into another.

Eg for Translation.

(TEACHER-FORCING)



contextual vector

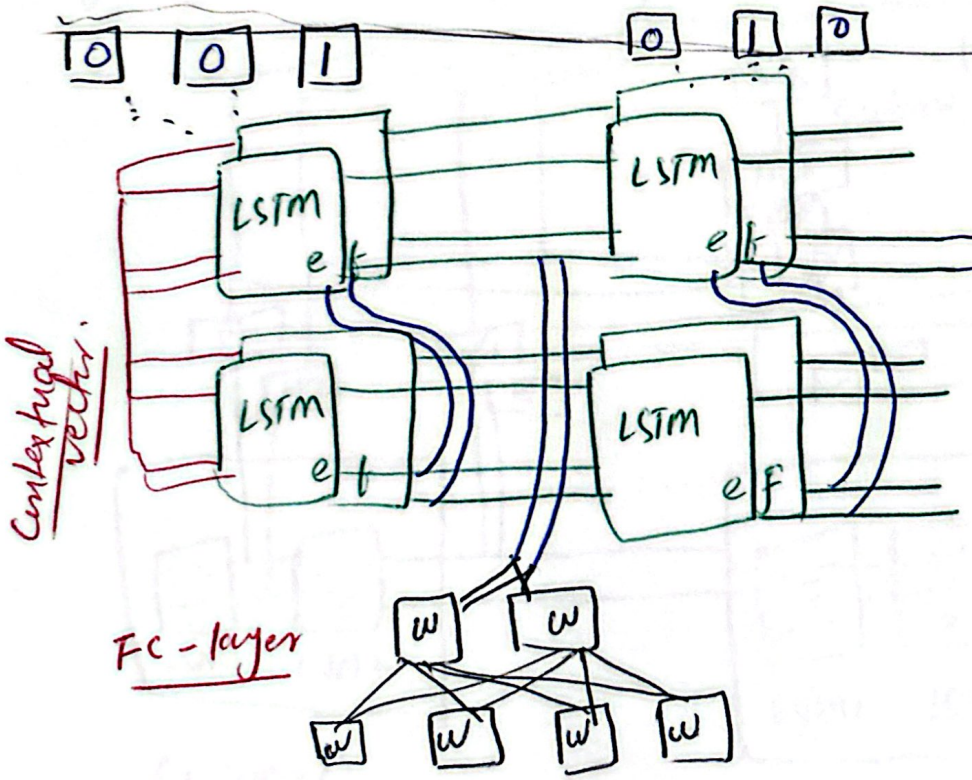
→ 2 cells - 2layer LSTM

→ Each cells have same weight/Biase

→ but differed in layer

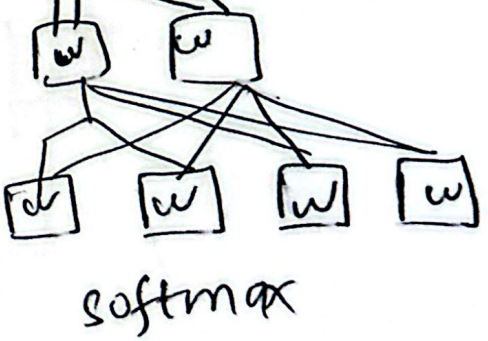
ENCODER: Paper:

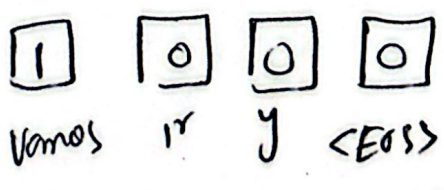
input	160,000 token
output	80,000 token
embedd	1000
LSTM	1000 cells 4 layers

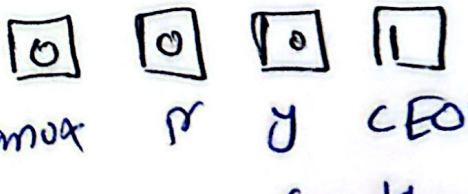


DECODER:

384 M weights/Biases







(output)

ATTENTION FOR NEURAL NETWORKS :-

A adding an additional path for each input value so that each step of decoder can directly access these values

Similarity of series.

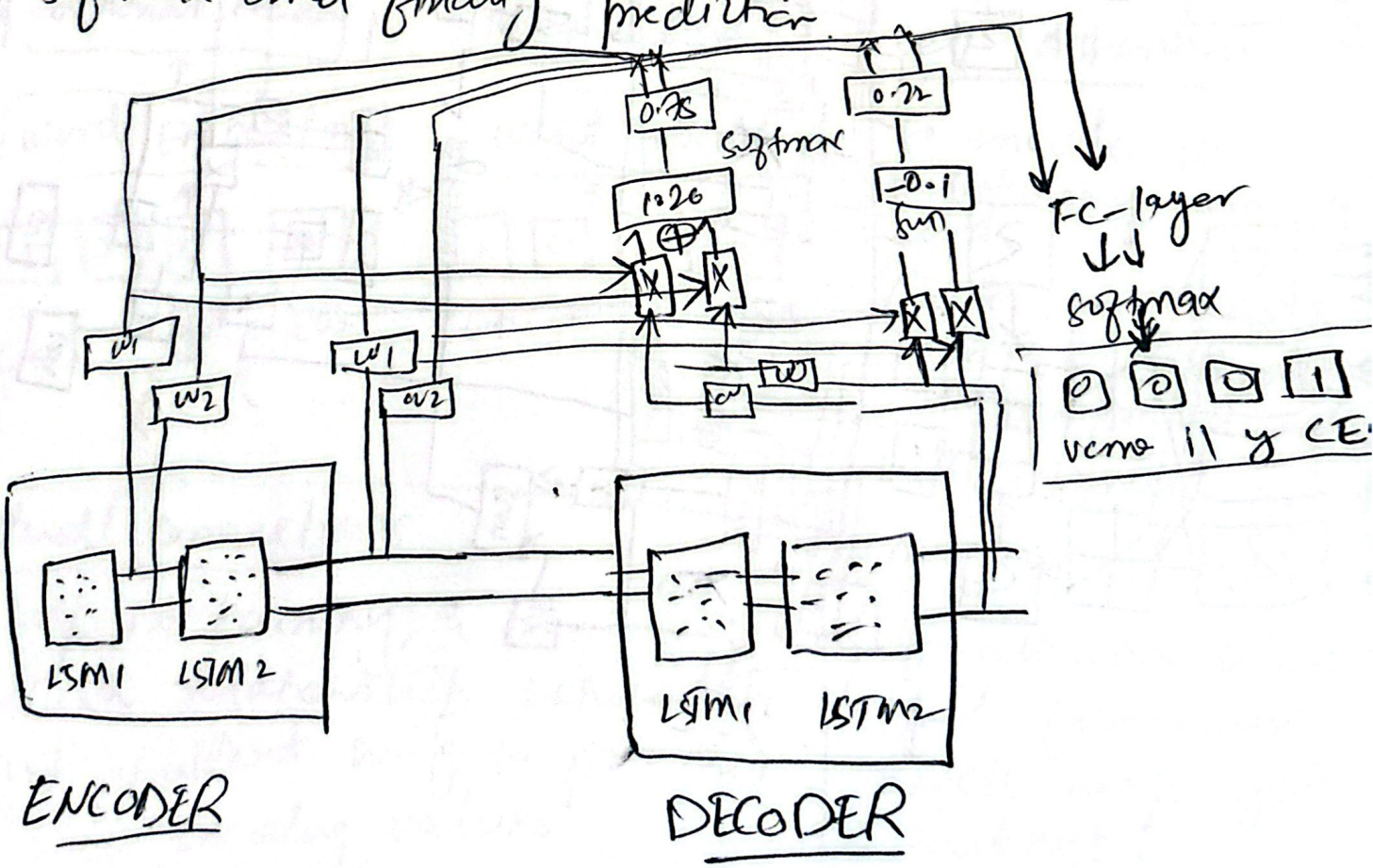
Cosine-Similarity

⇒ $\sum_{i=1}^{n} A_i B_i$ or just $\sum_{i=1}^{n} A_i \cdot B_i$ {dot product

$\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}$ → [Range / Scale to (-1, 1)]

Then softmax function.

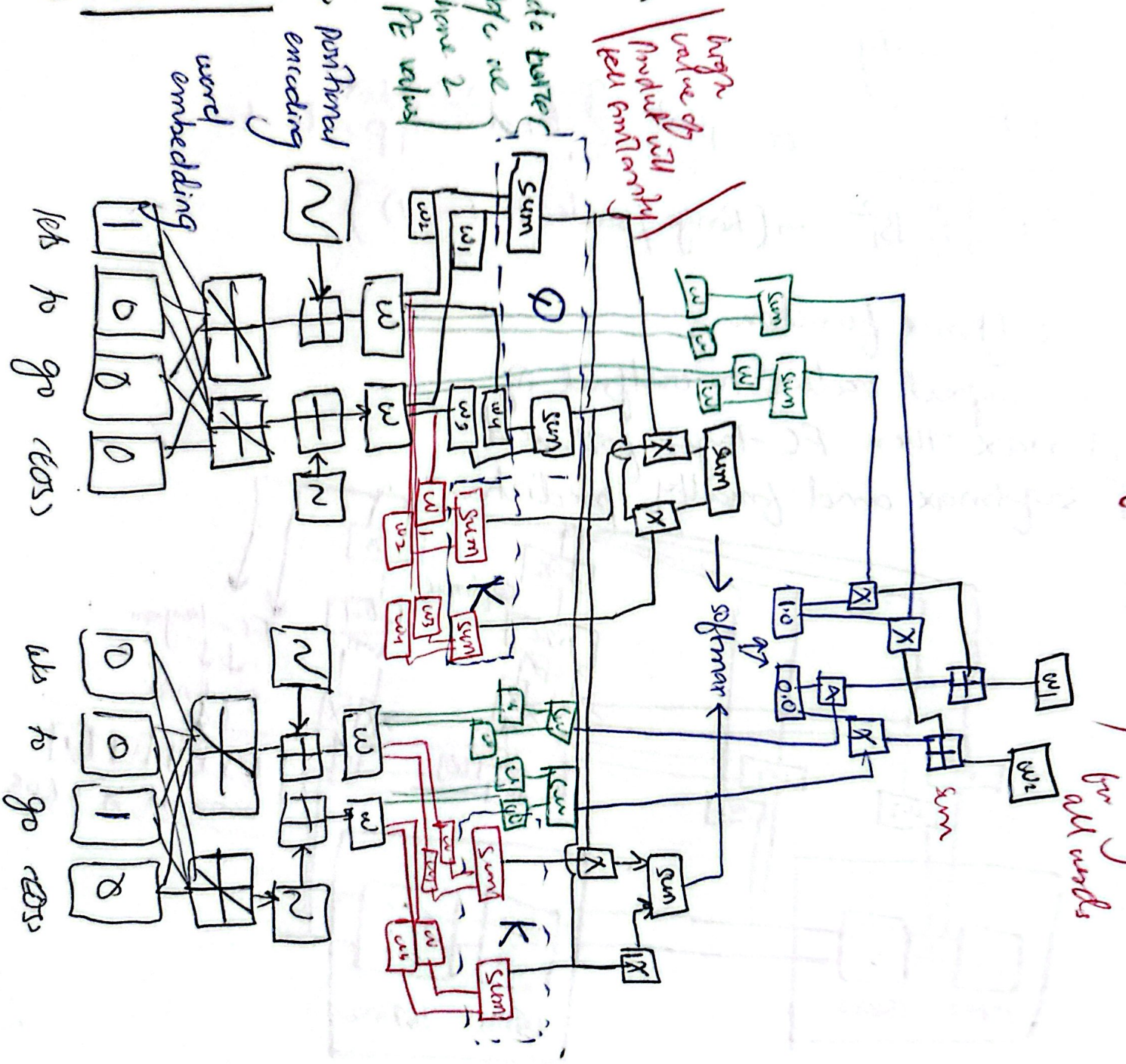
Then input value to output of softmax then FC-layer followed by softmax and finally prediction



TRANSFORMER ARCHITECTURE

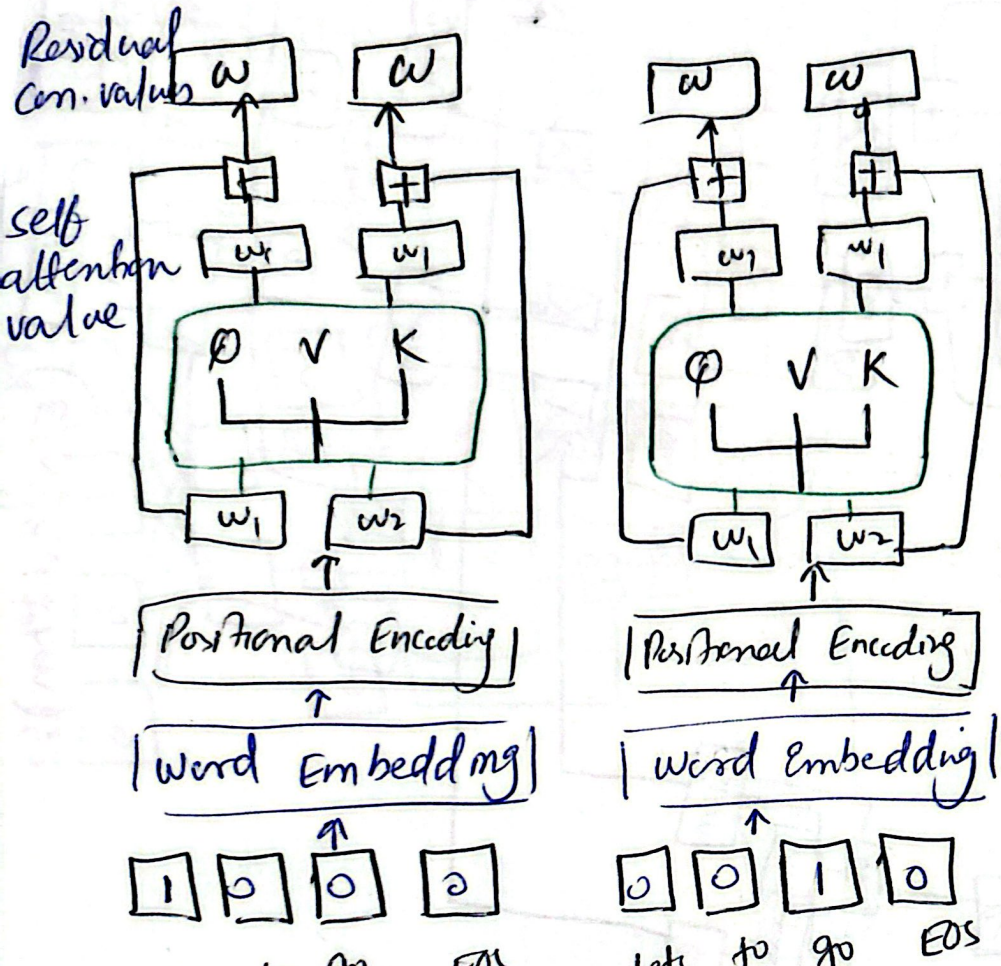
By Statquest. (Example, Lets go ↔ Vamos)

ENCODER (SELF-ATTENTION) for each words



2- ARCHITECTURE (CONTINUED)

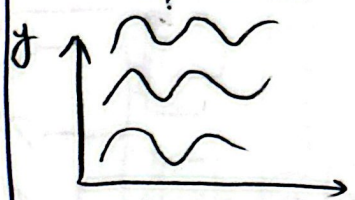
ENCODER:



Residuall connections

To help in training, & to establish relationship between input - (without having to preserve) Encoding values

FOR Positional Encoding:



for n (words), we pick y value and add them to encode.

(To keep track of the order of words)

Self Attention -

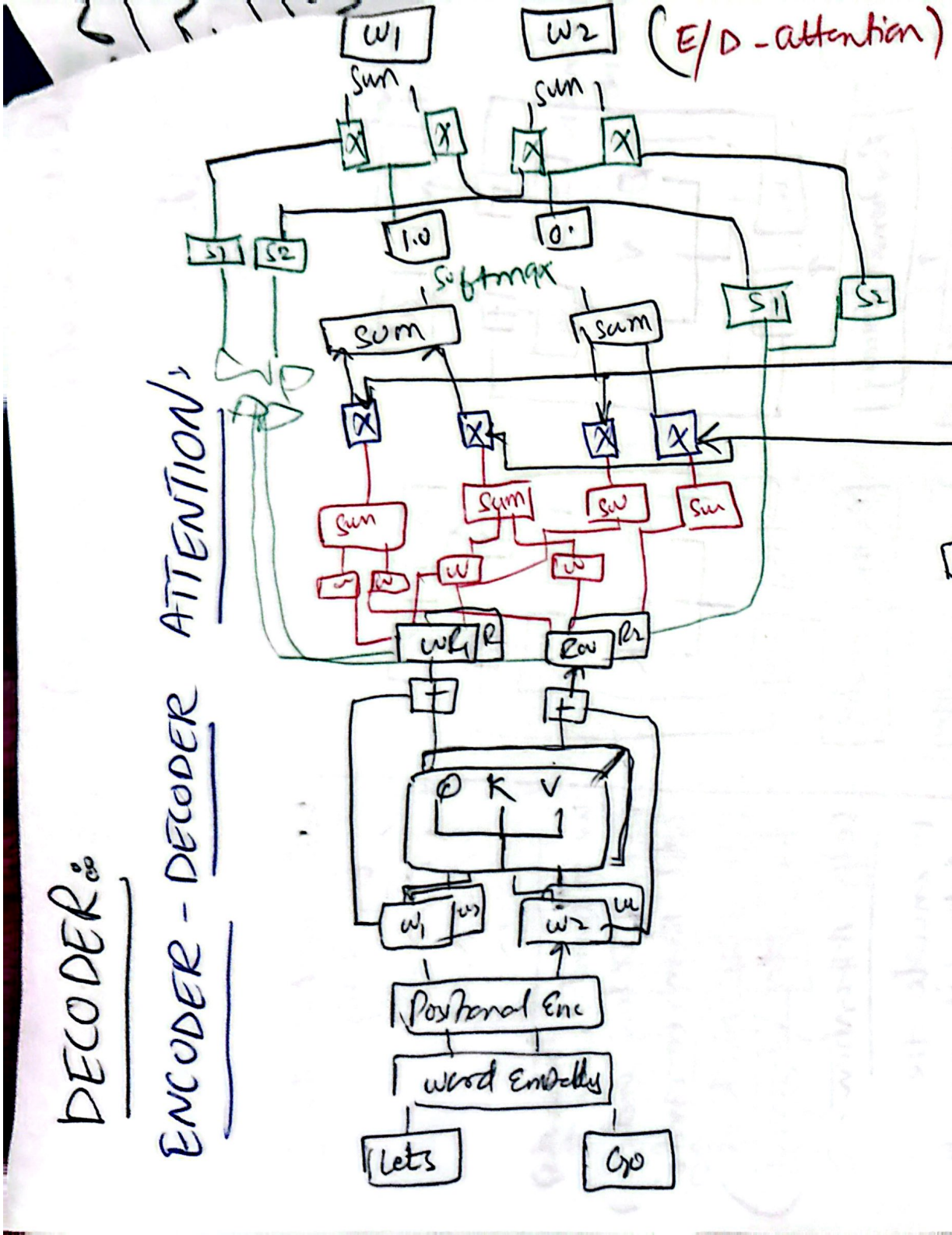
to encode the relationship among words.

(used same weights of Q, V, K to get self attention for all words

But different for st

(many stack = Multihea

self attention have input from other words help get context.



Decoder.

- Starts at (EOS).
- Decoder Q, K, V are different.
- E/D Q, K, V are also different but can be stacked.
- E/D allows to keep track of significant words of input and output.

ONTINUED :-

1076

IDS → Embedding 512 (learned) → PE (fixed not learned)

PE: even = sin pos / 10000 2i/dmodel (T/Inference)

odd: cos. —

(i = vector i)

why sin/cos ?? our model can recognize as continuous and see the pattern

self-attention. (each word to other in a sentence)

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

Eg = 6 word sentence $d_k = 512$

6 [E || 512] Q, × 512 [(K)]^T / √512 then softmax [6×6]

each value represent dA product of 1st Row with 1st Col

each Row sum to 1.

[6×6] × [V 6,512] = [6, 512]

each row capture meaning, position & relation with other words.

Same dimension as we started

STATQUEST.

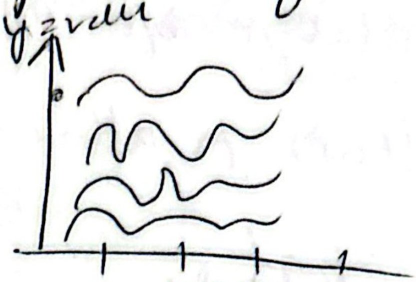
Attension is all you need!

(1) Word Embedding

(2) Positional encoding (To keep track of word order)

we add these values in embedding

even if it similar, we'll get unique values.



0.2 + 0.9 + ~ = 0.11

even if we exchange the position of words the positional encoding value will be same, but result will change

3) Transformer (dependency)

by - self attention. (similarity with others and itself) for all

Score weight of (Q, K, V)

Residual

help bring,

encode

word to no,
position of words,
relation ong words,
easy to run in parallel
E-D attention to keep track of things imbetue I/P → O/P

Self attention

have input from other word

help context.

Similarity = Dot Product

Embed values per ...

~~to establish~~ b/w ~~input and~~ & output also preserve the unchanged PE input

8-self attention self

Multihead to (words are viewed in parallel)

with own weights

Refer 1.

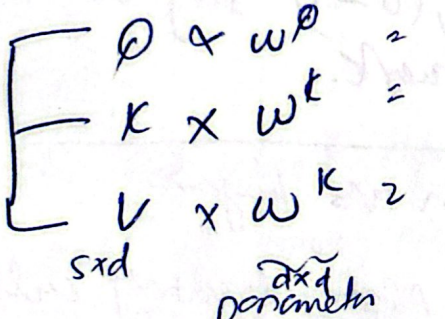
1) Permutation invariant (if we ignore PE) { mean if we change position of words there value will remain exact, just changes position

2) Self attention needs no parameter (except embeddings)

3) we expect value along diagonal to be max (as it's dot with itself) and if we want word not to interact, before softmax we replace it with $(-\infty)$.

Multi-head attention:-

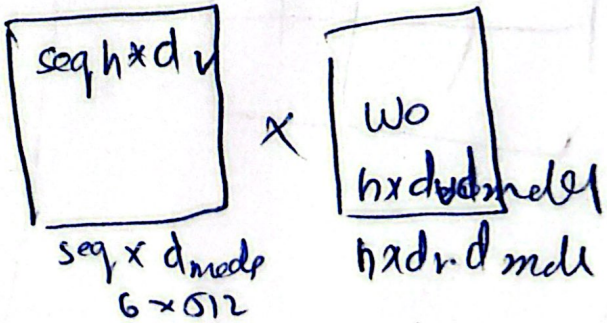
Encoder · (6 × 512) input



each part will see sentence but some embeddings

attention (Q1, K1, V1) different aspect

⇒ concat (head1, .. headn) W^O →



⇒

→ OK → softmax — visualize attention.

Q, K, V:

Add & Norm:

layer Normalization (we calculate mean & variance of each item independently) & replace by

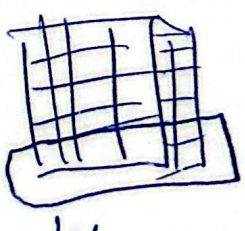
$$\hat{a}_j = \frac{x_j - \mu}{\sqrt{\sigma_j^2 + \epsilon}} \quad (\epsilon \text{ to large } (0-1))$$

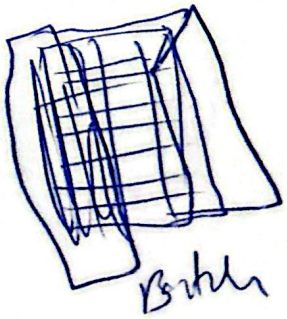
we also multiply by (gama) & add beta. (& γ & β are learnable // to learn these value and introduce fluctuations (amplified or not) when needed, (0-1) may be not resistive for network.

Batch vs layer

layer Norm, treating each item independently

- Batch = using for same parameter





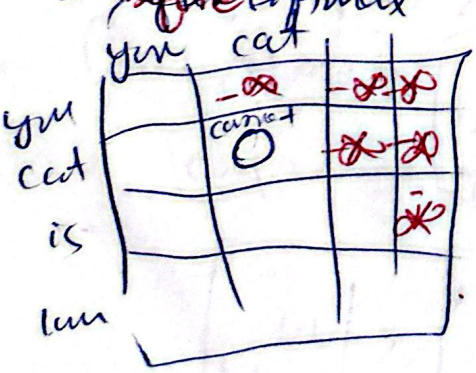
Decoder:

(Cross-MHA) KV from Encoder

masked MHAs

want to make model casual. (the output should only depend on current/previous not future)

so before softmax

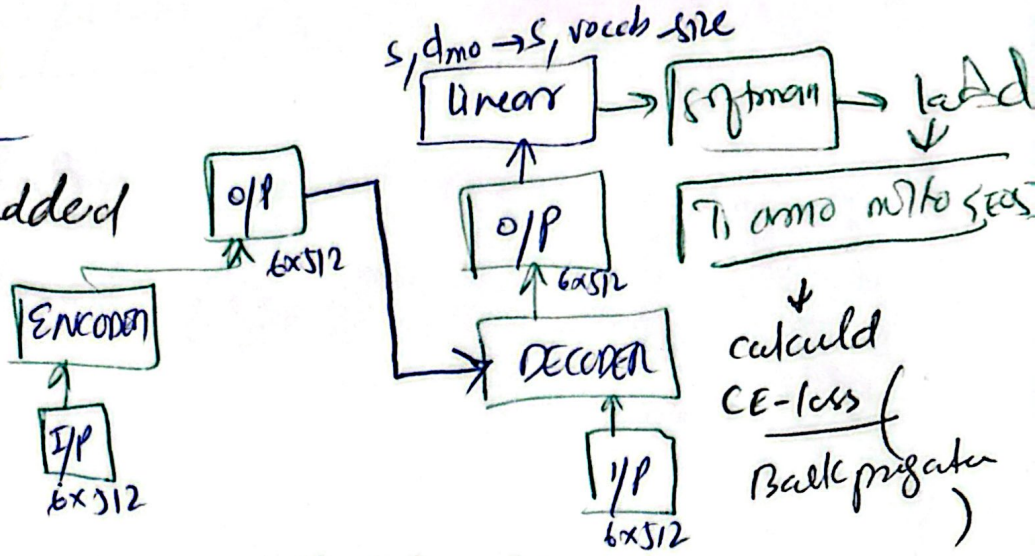


Inference & Training

Decoder input is padded

<SOS> — <EOS>

<SO> ----



This all happen in all 1-time step while RNN - N-step for N-length sequence

Inference:

Same → (output = logits)

But not in one-pass, by token-token (ti = 1, amo = 2,)

Greedy: at every time step we select word with max softmax value

Beam Search:

at each step select top B words & evaluate all possible next words for each of them and at each step keeping the top B most probable sequences. This performs better.