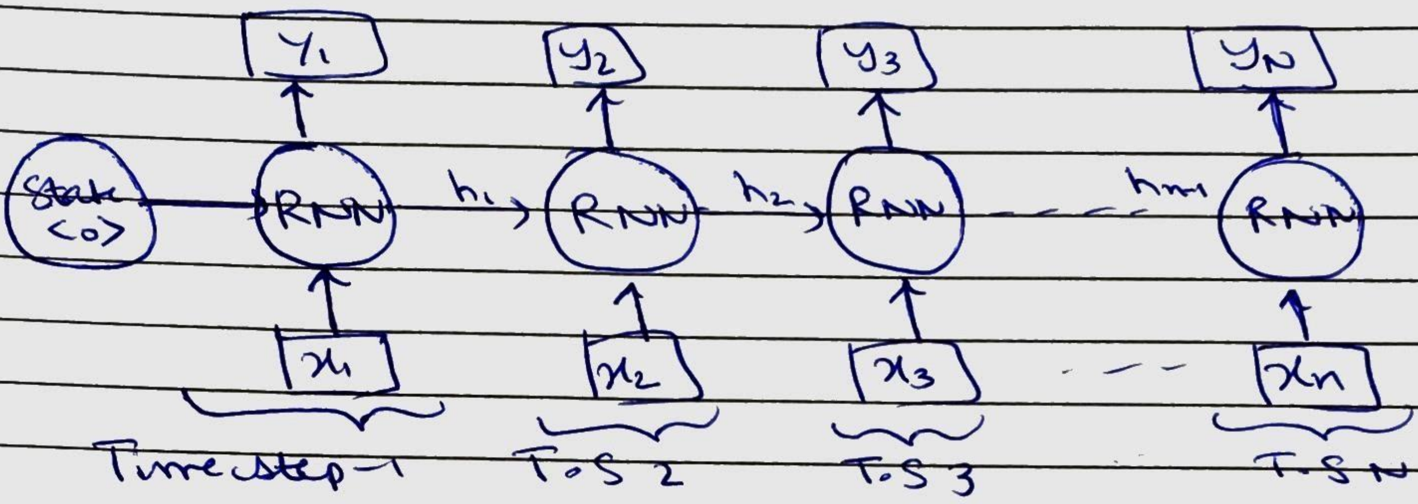
ATTENTION IS ALL YOU NEED

#RNNs and their problems:→

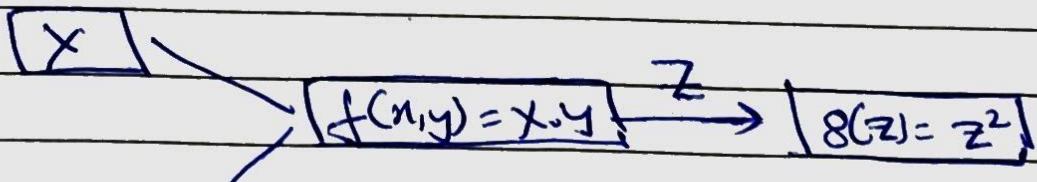


→ ~~For~~ Seq2 Seq model (map x to Y)
→ For n token, there will be n time stamps
→ Starts with 0, and each RNN has two inputs, x_i and h_{i-1} (hidden state from previous computation)

Problems:→

(1) Slow computation for long sequences (since it is sequential)

(2) Vanishing and exploding gradients:
eg. assume a simple graph



Here, $\frac{dy}{dx} = \frac{dy}{df} \cdot \frac{df}{dx}$

→ Thus, the longer the chain the more chances of vanishing / exploding gradients which is not desirable!!

(3) Difficulty in accessing information from long ago
→ First token will have very less or negligible impact on the last token (loss of context)
→ This is undesirable!!

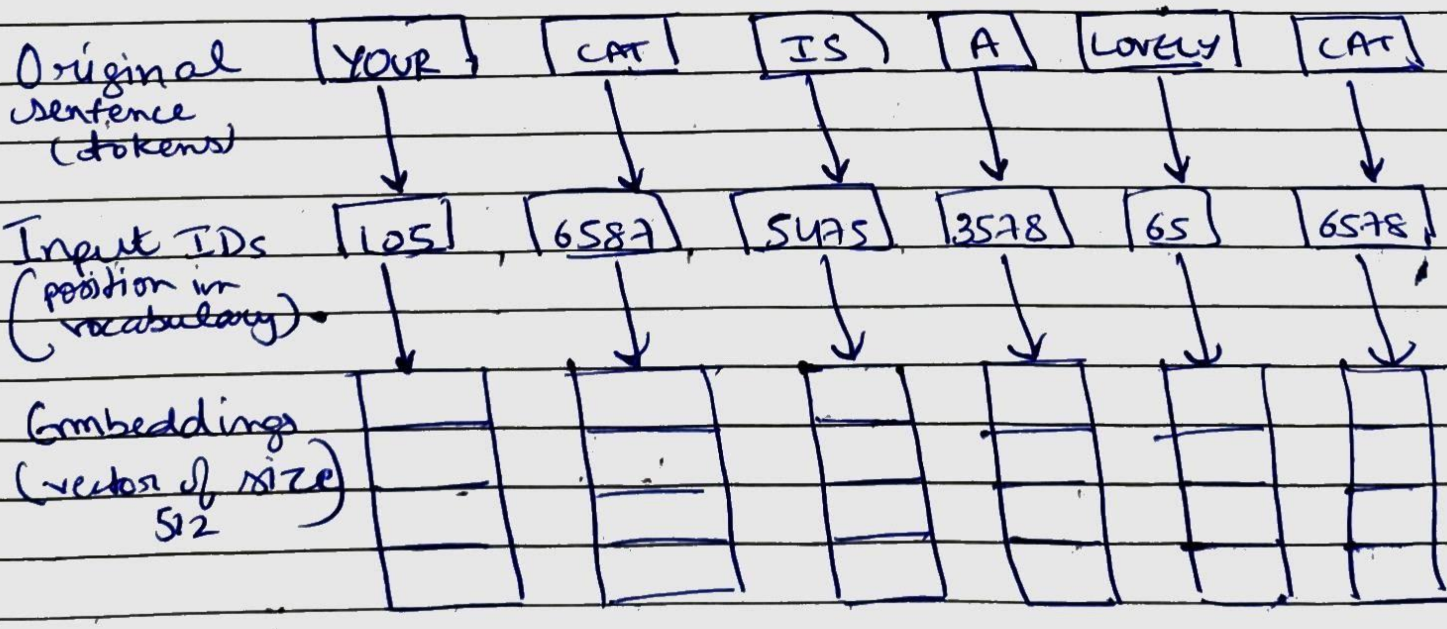
ENCODER: →

STEP 1: Input embeddings: →

eg. Your cat is a lovely cat

→ Tokenize it (i.e split into individual words)

→ Map these tokens into numbers which represent their position in our vocabulary


Original sentence (tokens)
YOUR
CAT
IS
A
LOVELY
CAT
Input IDs (position in vocabulary)
105
6587
5475
3578
65
6578
Embeddings (vector of size 512)

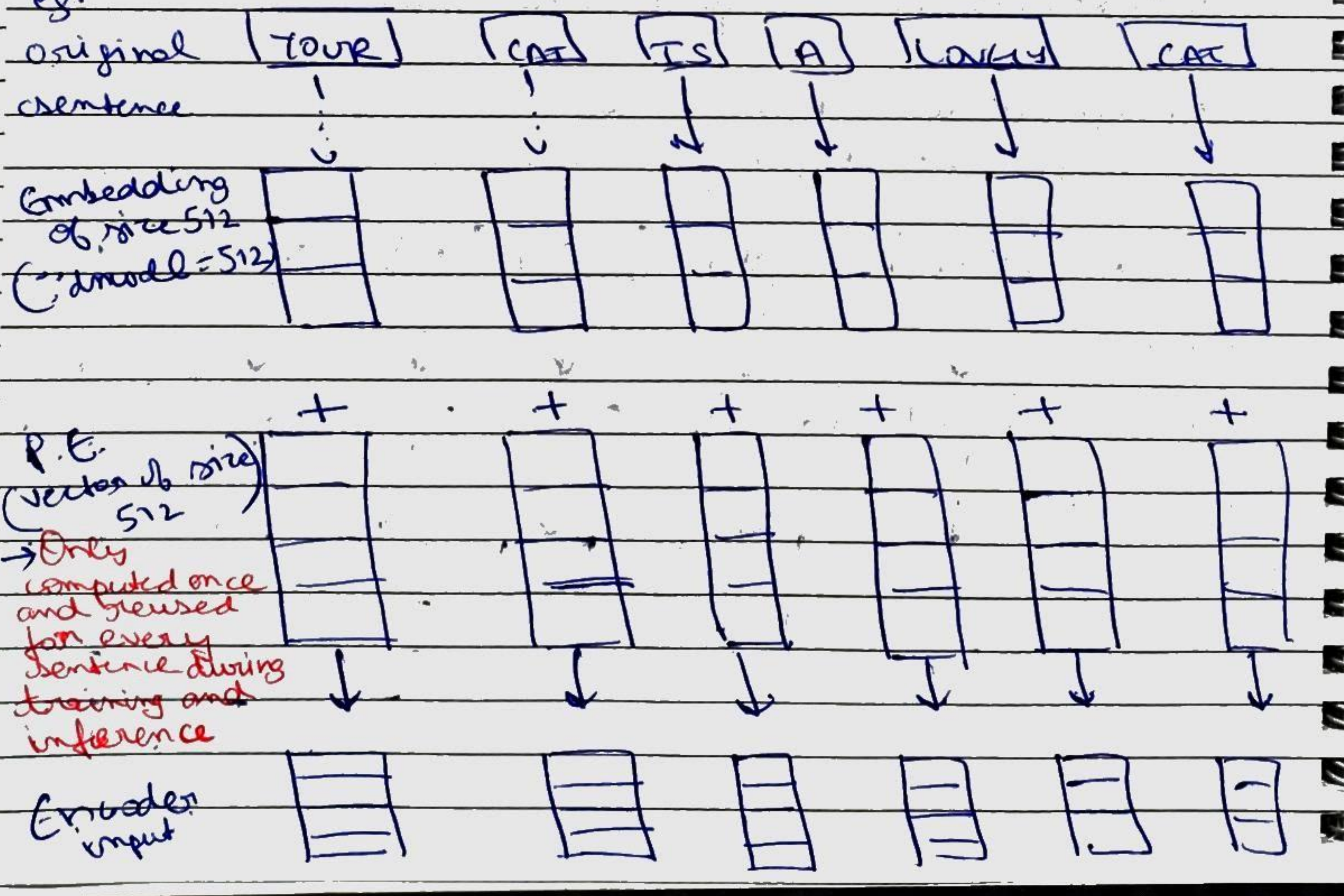
→ However, note that these embeddings are learnable parameters and can change with time during training in order to satisfy the loss function!!

→ Also, we define $\boxed{dmodel = 512}$ which represents the size of embedding vector of each word.

STEP 2: POSITIONAL ENCODINGS :→

→ We want each word to carry some information about it's spatial encoding in the sentence.
→ We want model to treat words that appear 'close' to each other as 'close' and those that are distant as 'distant'.
→ We want the PE to represent a pattern that can be learned by the model!! (HOW??)

eg.

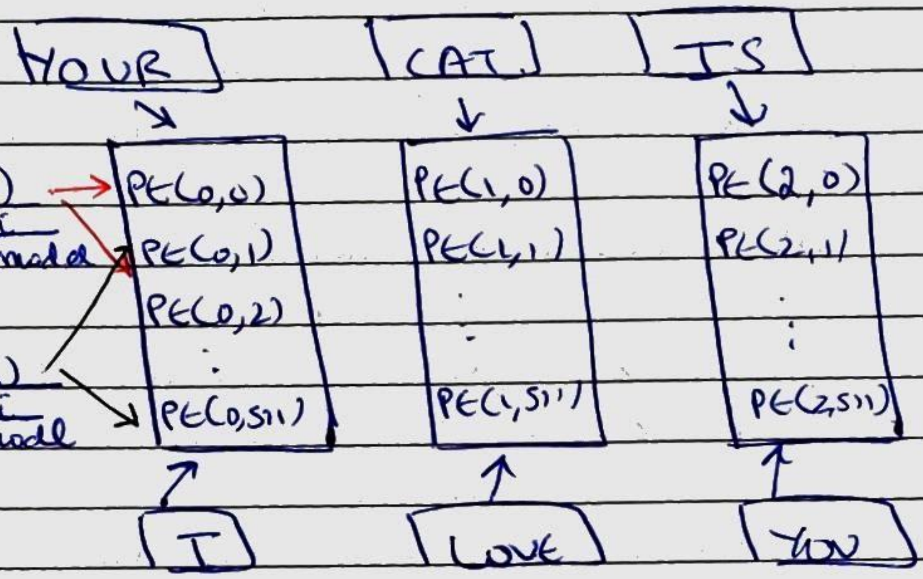

original sentence
YOUR
CAT
IS
A
LOVELY
CAT
Embedding of size 512 (∵ dmodel = 512)
+
P.E. (vector of size 512)
→ Only computed once and reused for every sentence during training and inference
Encoder input

How is PE calculated?

eg.

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

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



→ We can use same PE for diff sentences (as it is just a way of storing position of words) and is thus not learned or unique to diff. sentences.

→ Thus, they are just computed once and can be used again and again!!

Q=) Why trignometric functions??

A=> ″ ″ like sin and cos naturally represent a pattern that the model can recognize as <u>continuous</u>, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.

STEP-3 SELF-ATTENTION (with single head)

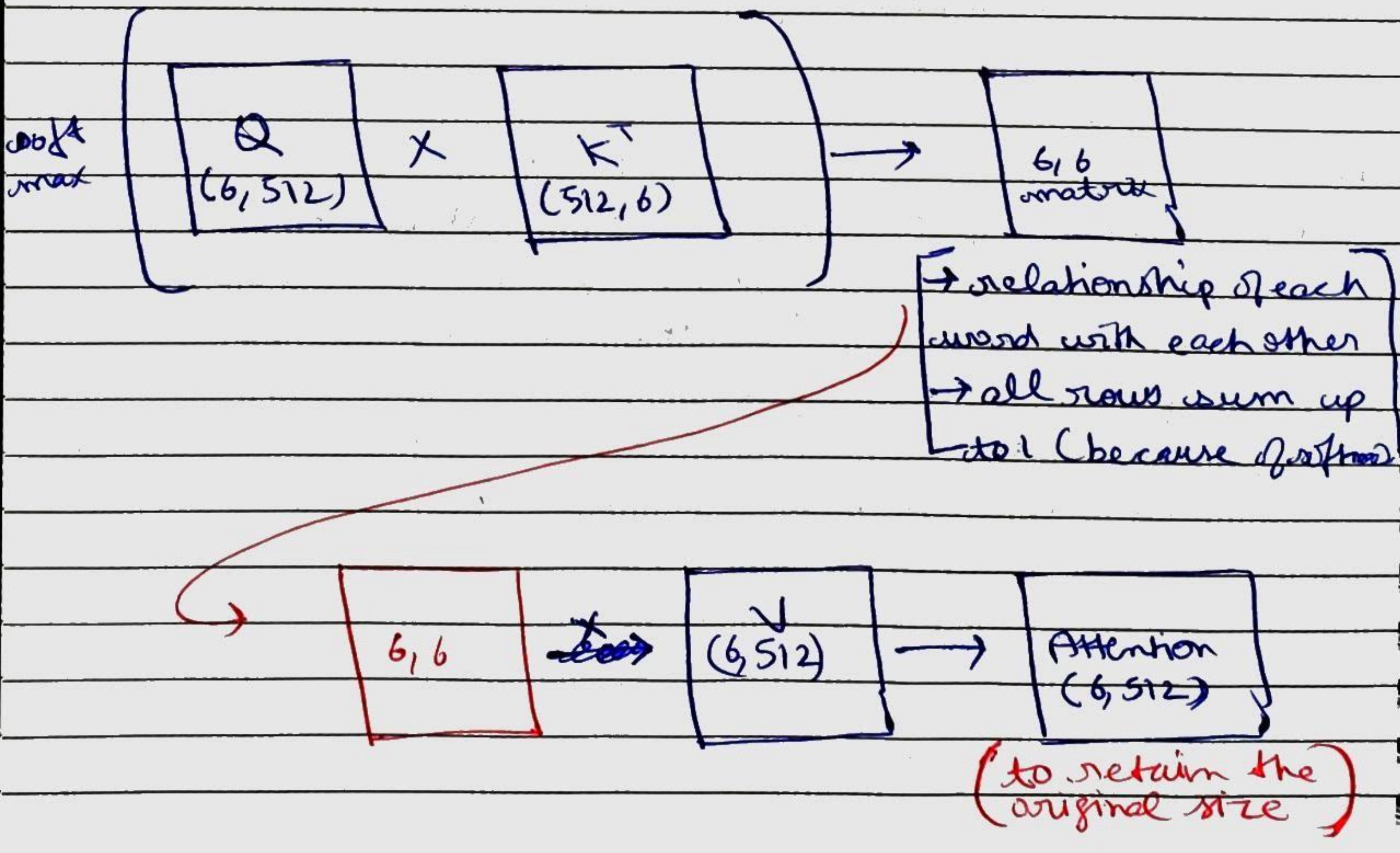
→ It existed before Transformers. The authors just changed it to Multi-Head!!

→ SA allows the model to relate words to each other

eg. IE → captures meaning of words
PE → give them info of position of words inside sentence
SA → relates words to each other.

$$\text{Attention}(Q,K,V) \rightarrow \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V$$

Here, Q, K, V are just the input sentence embeddings (after PE of course) of size (6, 512)
∵ our sequence is of length = 6, and dmodel = 512

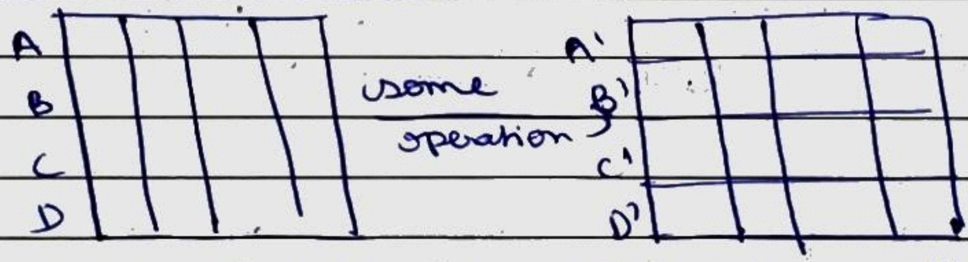


→ relationship of each word with each other
→ all rows sum up to 1 (because of softmax)

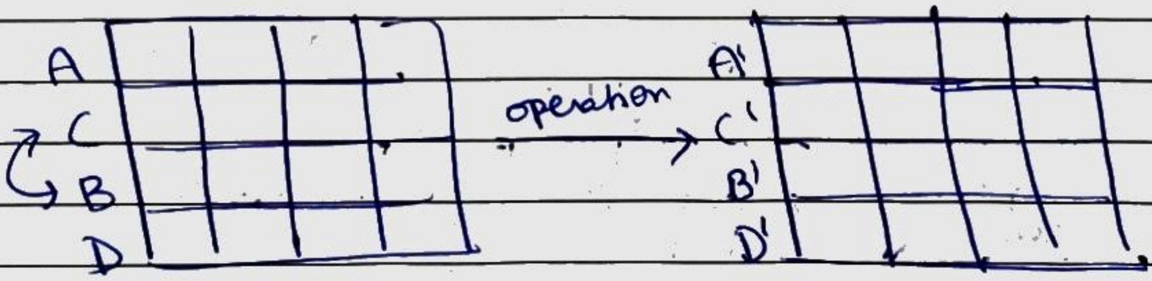
→ Each row in this new attention matrix captures not only the meaning (given by the embeddings) or their position in sentence (given by P(·)) but now also represents each words interactions / relationship with other words!!

Properties of self-attention :→

(1) Self attention is permutation-invariant
(i.e swapping orders does not change the computed value)
→ This is desirable

eg.


if we swap B & C, it won't change the calculated value of B' and C'. It will just change their position in the matrix.

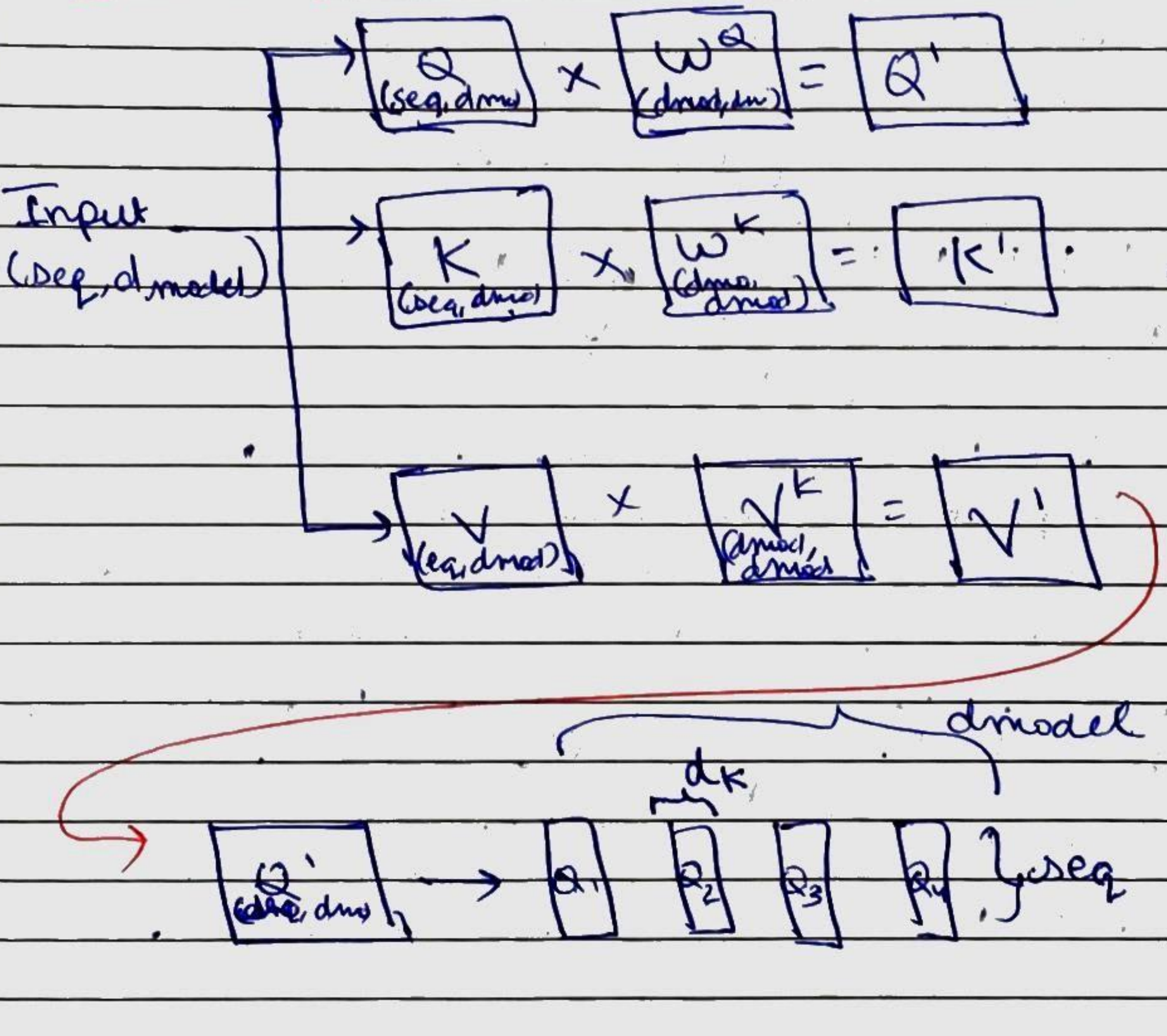


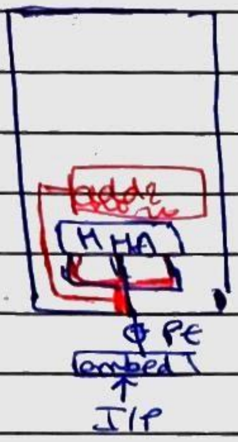
(2) It requires no parameters as of now
(will change later when we discuss multi head attention)

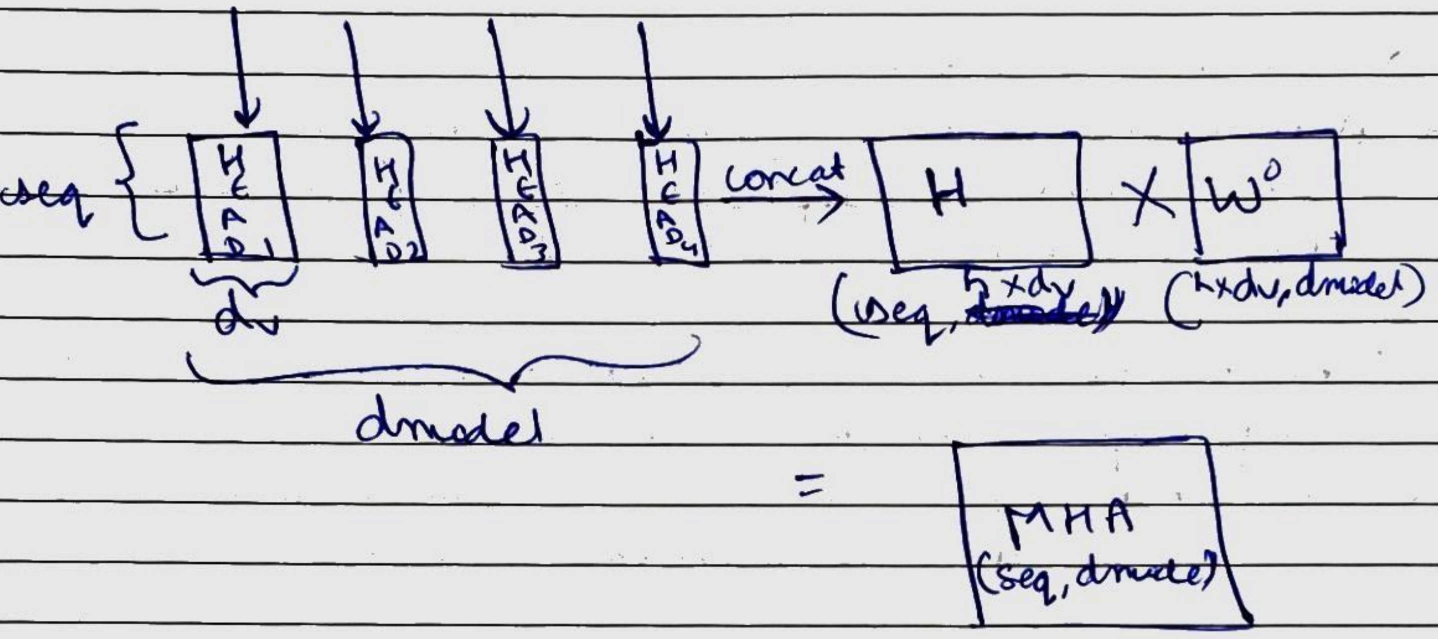
(3) Values along diagonals are expected to be highest
(∵ a word will have strongest relationship with itself)

(4) If we don't want some positions to interact, we can always set their ~~position~~ value to $-\infty$ before applying softmax operation and the model will not learn those interactions }
($\because e^{-\infty} = 0$). This is desirable when we study decoders!!.

STEP-4: MULTI-HEAD ATTENTION:







→ Here, initially 3 copies of the input embeddings are made as Q, K, V (as previous)
→ These Q, K, V are multiplied by parameterized W^Q, W^K, W^V to form Q', K' and V'
→ These Q', K' and V' are split into multiple heads and then attention is performed on each head as per usual.

$$\boxed{\begin{aligned}\text{Attention}(Q,K,V) &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\cdot V \\ \text{and head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}}$$

→ These heads are then concatenated to form H matrix and then multiplied by W^o to give our final output.

$$\boxed{\text{Multihead}(Q,K,V) = \text{concat}(\text{head}1, \text{head}2, \text{hed}\ldots h)W^o}$$

Q→ But why split???

→ Here, each head$_i$ is watching the full sentence but a different aspect of ~~each word~~ the embedding of each word.

→ This is because, we want each head to watch diff. aspect of the same word.
eg. In chinese, a word can be a noun, verb, adverb, etc depending on the context.

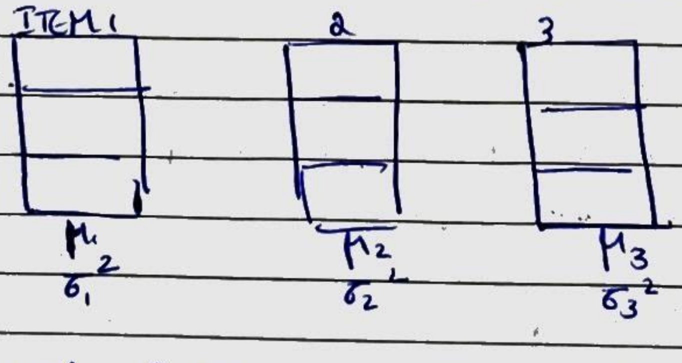
→ So, what we want is that maybe one head relates it as a noun, another as verb and so on..
i.e, we want to look at all diff possibilities.

→ This attention can also be visualized!!

Q⇒ Why Q, K, V?
⇒ Simple, it comes from python dictionary!!

STEP-5: ADD & NORM: →

→ But, first we need to learn about layer normalisation.
→ Consider, three items in a batch.



→ We calculate mean & variance for each item separately and replace each value in item by

$$x_i = \frac{x - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$

→ We also introduce two parameters, gamma and beta, that introduce some fluctuations in the data, because maybe having all values between 0 and 1 may be too restrictive for model. The network will learn to tune these two parameters to introduce fluctuations when necessary.

DECODER :→

→ We have output embeddings (similar to IE in Encoder) and PE (similar to Encoder)

→ Next, we have, masked multihead attention

→ The K and V come from encoder whereas the query comes from decoder. Thus, it is no longer a self-attention but a cross-attention!!

STEP-1 : MASKED MULTIHEAD ATTENTION :→

→ Our goal is to make the model causal; ie; it means the output at a certain position can only depend on the words on the previous positions. !

→ The model must not be able to see future words. (HOW ??)

→ We do this by simply replacing the future words with $-\infty$.

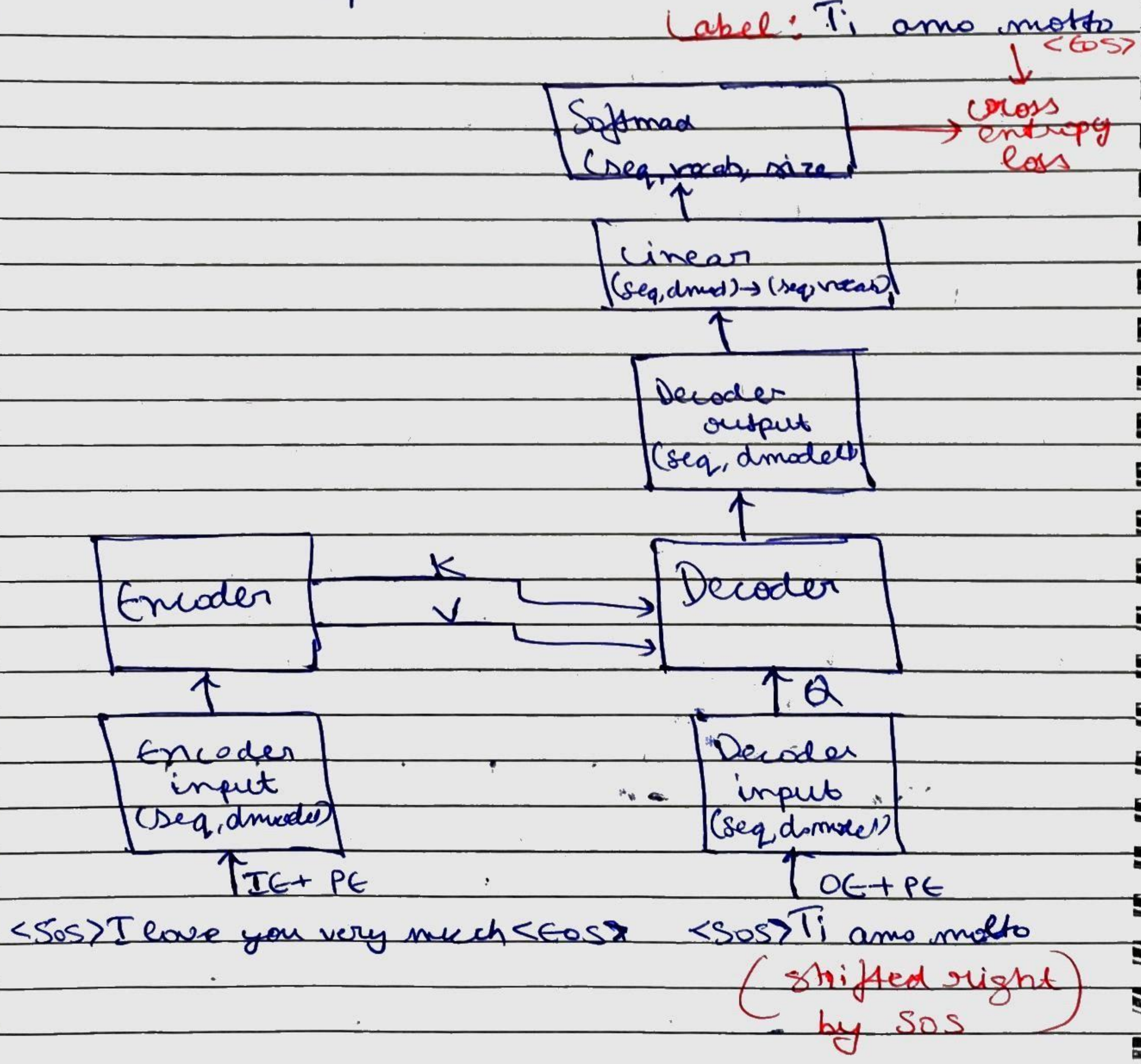
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR		$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
CAT			$-\infty$	$-\infty$	$-\infty$	$-\infty$
IS				$-\infty$	$-\infty$	$-\infty$
A					$-\infty$	$-\infty$
LOVELY						$-\infty$
CAT						

#TRAINING:→

→ We will consider a translation task.

eg. I love you very much (ENG)
↓
Ti Amo Malto (ITALIAN)

→ We add special tokens to our sentence.


Label: Ti amo motto <EOS>
cross entropy loss
Softmax (seq, vocab, size)
Linear (seq, dmod) → (seq, vocab)
Decoder output (seq, dmodel)
Encoder
K
V
Decoder
Q
Encoder input (seq, dmodel)
Decoder input (seq, dmodel)
IE + PE
OE + PE
<SOS> I love you very much <EOS>
<SOS> Ti amo motto
(shifted right by SOS)

→ Next, we do IE + PE to create input embeddings for our encoder. Illy for decoder.

→ We get K and V from encoder and use Q from decoder (as it is masked) to get our decoder output

→ This output needs to be mapped to our vocabulary with a linear layer

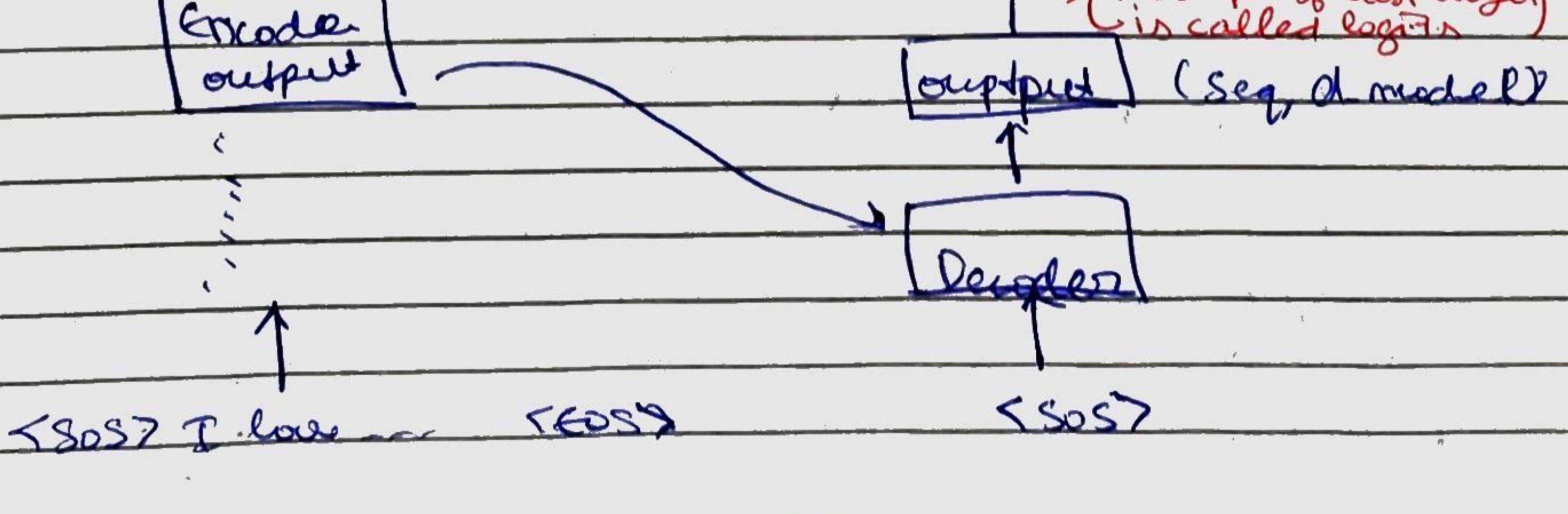
→ Finally, we take the softmax to get our word predictions.

→ Once, we have our predicted label and the true label, we use cross entropy loss to update the weights.

→ This all happens in 1 time step
(Time step = 1)
i.e all in one pass.

INFERENCE :→

TS1



→ We start with <SOS> and compute the logits as per decoder model.
→ We then perform softwarm on these logits to get the next word which is Ti

Time step 2 :→
<SOS> Ti
→ We don't compute encoder part again as our English sentence hasn't changed.
→ However, for decoder we go from
<SOS> Ti ⟶ ~~<~~ Amo

TS 3 :→
<SOS> Ti ⟶ Molto

TS 4 :→
<SOS> Ti Amo Molto ⟶ <EOS>

Thus; it is token by token until we generate <EOS>!!

INFERENCE STRATEGY :→

→ We selected, at every step, the word with the maximum probability / softmax value. This strategy is called greedy and usually does not perform well.

→ A better strategy is to use top B words and evaluate all possible next words for each of them and at each step, keeping the top B most probable sequences. This is Beam Search and generally performs better!!