Transformers & BERT

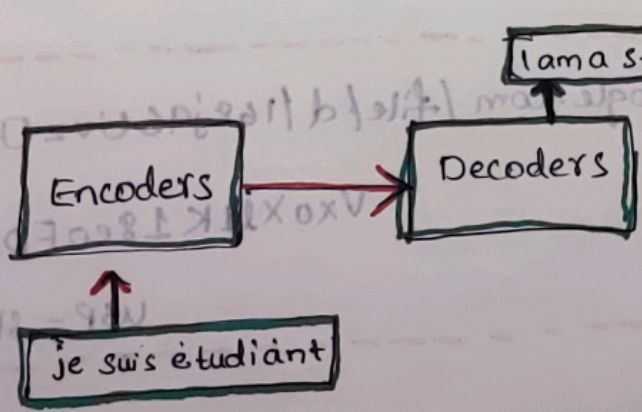
* BERT is special type of transformers.

BERT ⇒ Bi-directional Encoder Representation from Transformers

* To understand transforms, we need to understand attention, to understand attention, weneed to unde. encoder-decoder, models.

link: jalammar.github.io/illustrated-transformer/

* Transformers: top-down Approach:



⇒ Seq-to-Seq model.

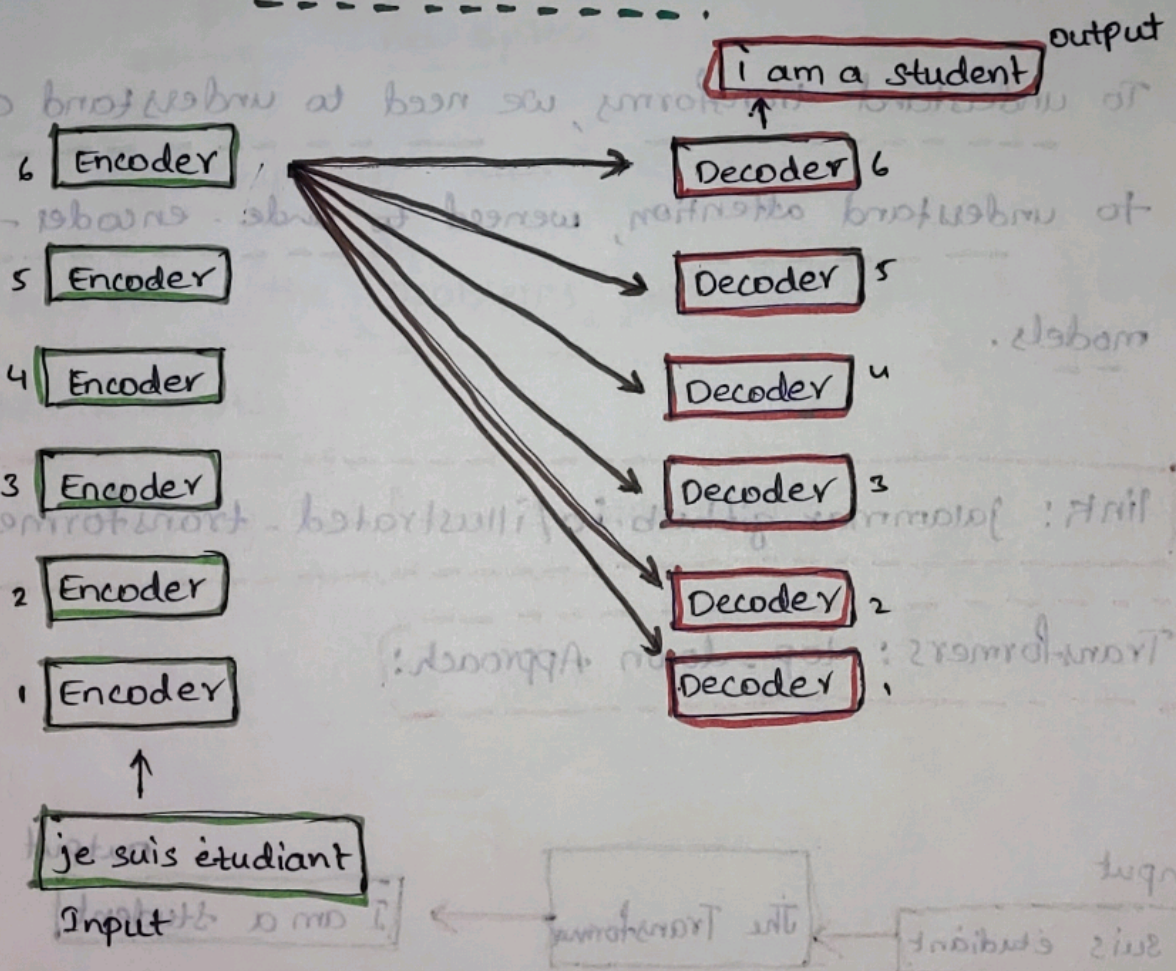
→ This is the top-down approach.

→ This encoder-decoders are based on Attention and not based on RNN

→ Attention is all we need.

↳ A stack of 6 encoders & decoders are used

(6 is a hyper parameter)

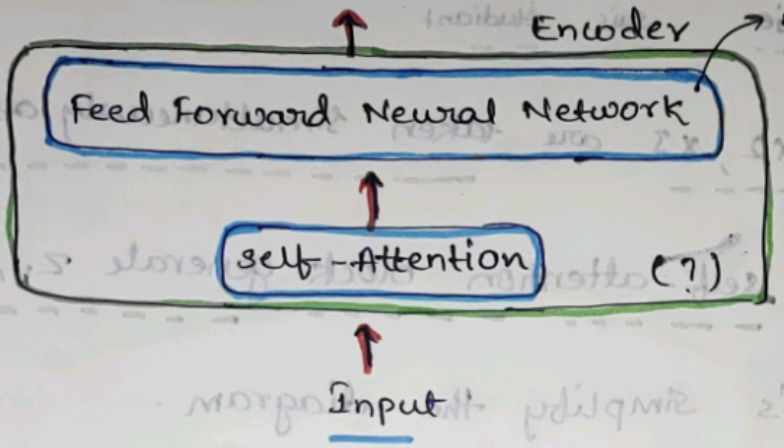


Notes: drive.google.com/file/d/168j96UvzOWvbrhU5Vx0X11K18eoFDSKb/view?usp=sharing

* each of these encoders are identical

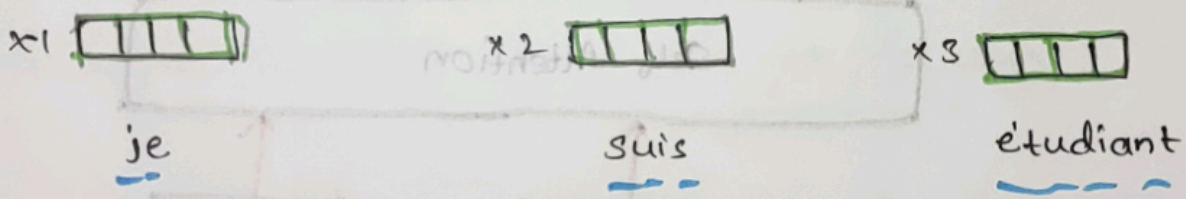
* Here, the output of last encoder is fed to all the decoders.

* Structure of encoder:

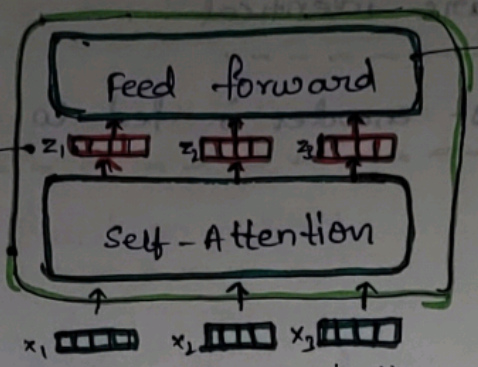


input: word-vectors/tensors.

→ take I/p as non-english word, convert it into 512 dimensional vector (w2vec)



each of these are 512-dimensional

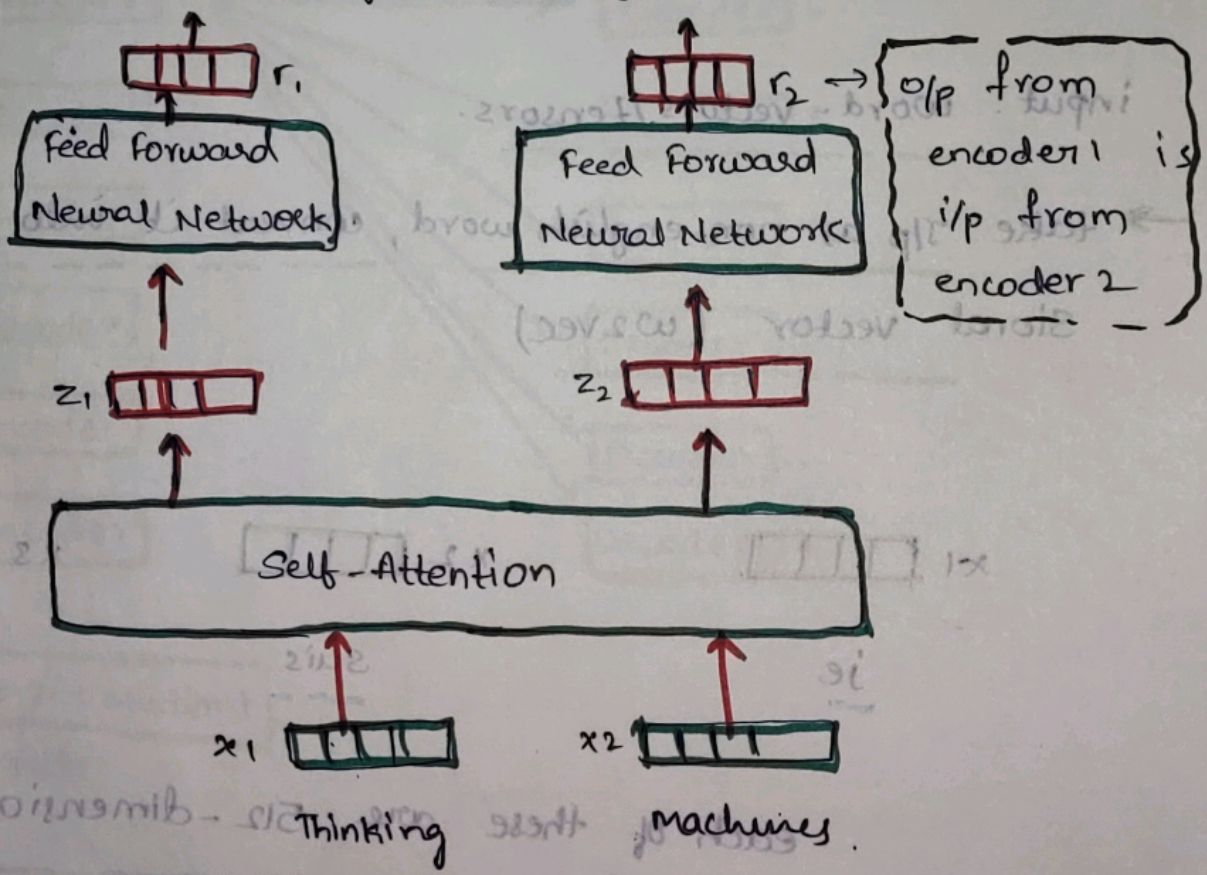


o/p corresponding to x_1

all x_1, x_2, x_3 are taken simultaneously as i/p.

* How does self-attention block generate z_1, given x_1

let's simplify the diagram.



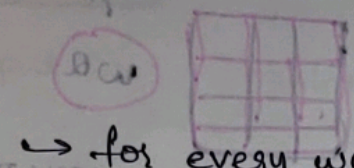
Encoder - 1.

Here, o/p from encoder-1 goes as i/p to encoder-2 (r_1, r_2)

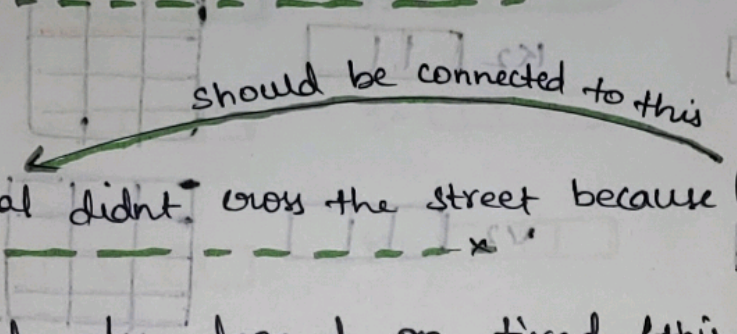
* what is self-attention?

input: "The animal didnt cross the street because it was too tired"

refers to animal or street?



↳ for every word, we have x_i's



⇒ It should also depend on tired (this is an action by animal, not the street)

⇒ we should pay attention to the word animal & tired, more than the word 'street'.

& attention refers to where we are focussing.

(PTO)

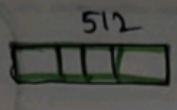
FOR

Self-attention BLOCK.

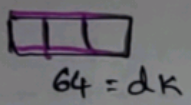
input " Thinking Machines "

Step-1
compute
W^Q, W^K, W^V.

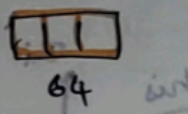
Embedding



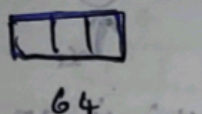
Queries

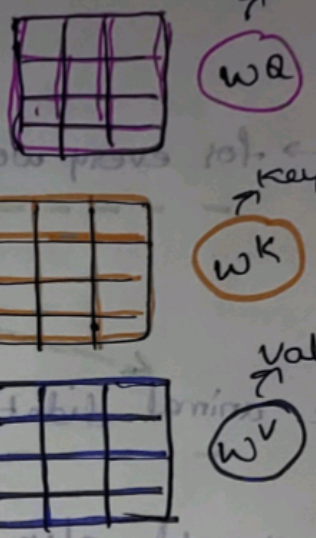


Keys



Values





W^Q ⇒ Query weights (Matrix)

Given x_1 & x_2, we compute q_1 & q_2

$q_i = x_i W^Q$

"W^Q" ⇒ parameters of self-attention module

Given x_i, we are multipl computing q_i by $x_i * W^Q$.

this is simple matrix multiplication & no activ'n function.

$d_k = 64$

64 is a hyp-param

Values of W^Q, W^K, W^V ⇒ all the values of matrices that we need to train using loss func.

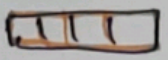
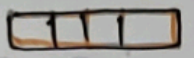
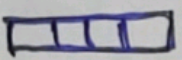
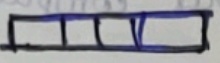
(trainable matrices of attention layer)

§ How z_i created from x_i ??

⇒ from x_1, we get q_i, k_i, v_i by mat-mul. then, we create score (2nd step), Dot prod.

step-2
multiply
Ki with qi

2nd step:

input	Thinking	Machines
Embedding	x_1 	x_2 
Queries	q_1 	q_2 
Keys	k_1 	k_2 
values	v_1 	v_2 
score	$q_1 * k_1 = 112$	
	$q_2 * k_2 = 96$	

3rd step:

divide the values we got with 8 ($\sqrt{dk=64}$)

112 ⇒ 14

96 ⇒ 12

4th step:

Apply softmax.

14 ⇒ 0.88

96 ⇒ 0.12

Step-5:-

The softmax we get, multiply with values.

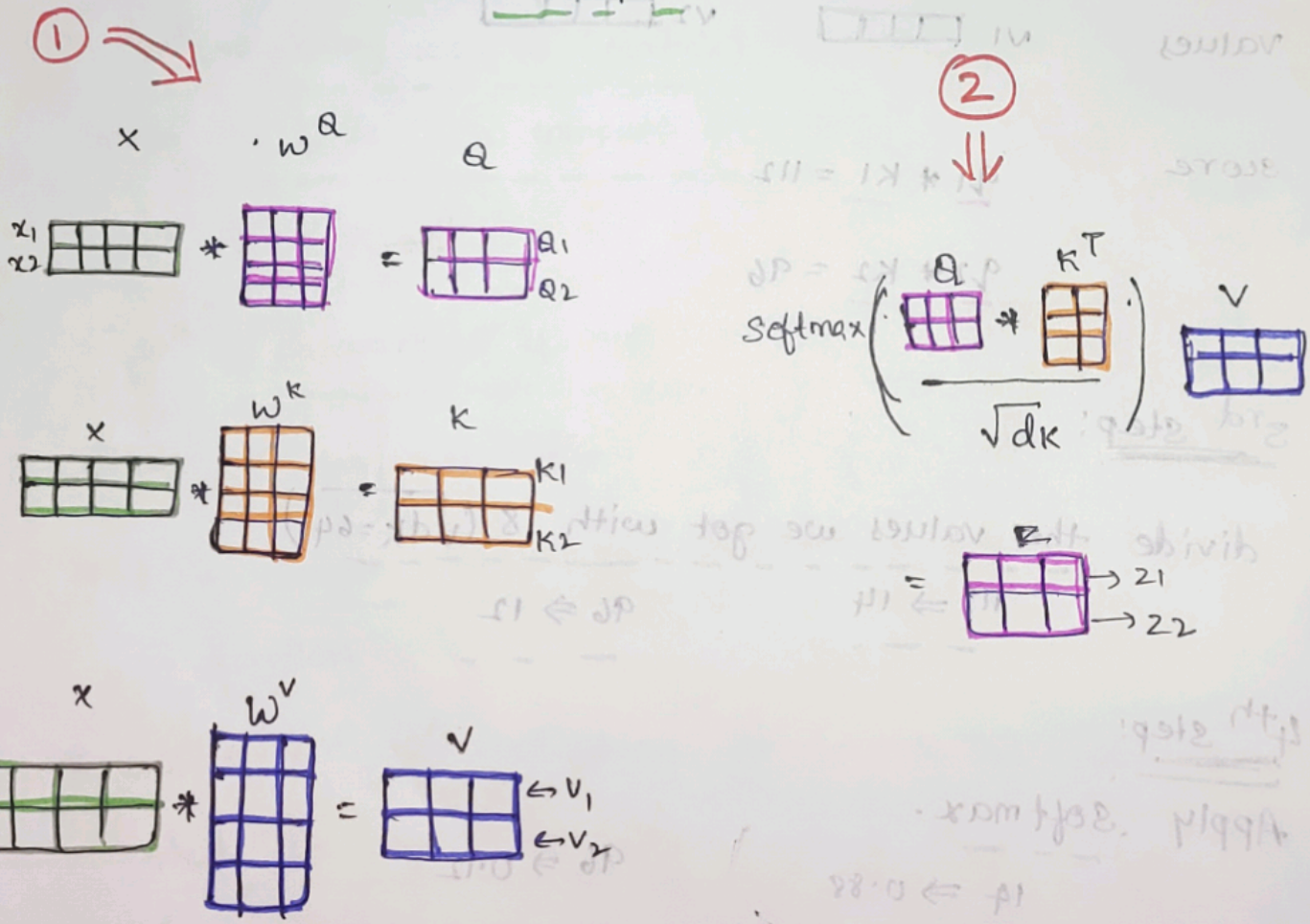
(we are trying to generate z_1 corr. to x_1 (x_1 will focus on itself more).

so, v_1 is larger than v_2 (softmax multiplied is larger).

Step-6:-

Obtain z_i by summing all v_i's.

Matrix computation. [32:31]



* ⇒ here, we've ~~real~~ created z_i's using x_i's.
* here, we have 5 multiplications.
* Attention ~~focus~~ helps us to focus on some of the words in the vicinity which matter.
* There are lot of similarities b/w w2vec and BERT.
* (w2vec says, ~~to encode~~)*
* ~~Single~~ Multi-headed attention:

↳ The single-headed attention capture the significance that "it" is an animal but, it misses to capture the fact that tired ness of the animal.

"The animal didn't cross the street because it was too tired."

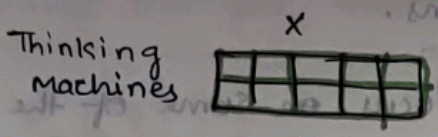
↳ instead of having only one set of matrices W^Q, W^K & W^V, we have 8 sets of matrices (8 - hyperparam)

⇓

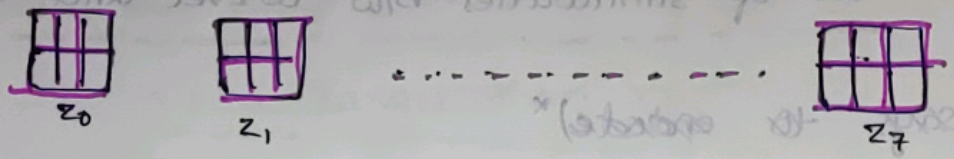
This increases the representation power of our model.

(more hyper-parameters encodes more Info)

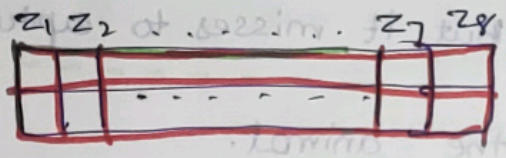
8. headed attention mechanism:-



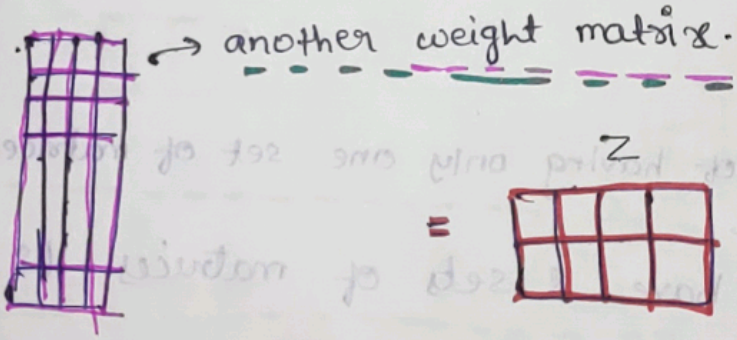
↓ calculating attention seperately in eight different attention heads



1) concatenate all the attention heads.



2) multiply with a weight matrix W^o that was trained jointly with the model.

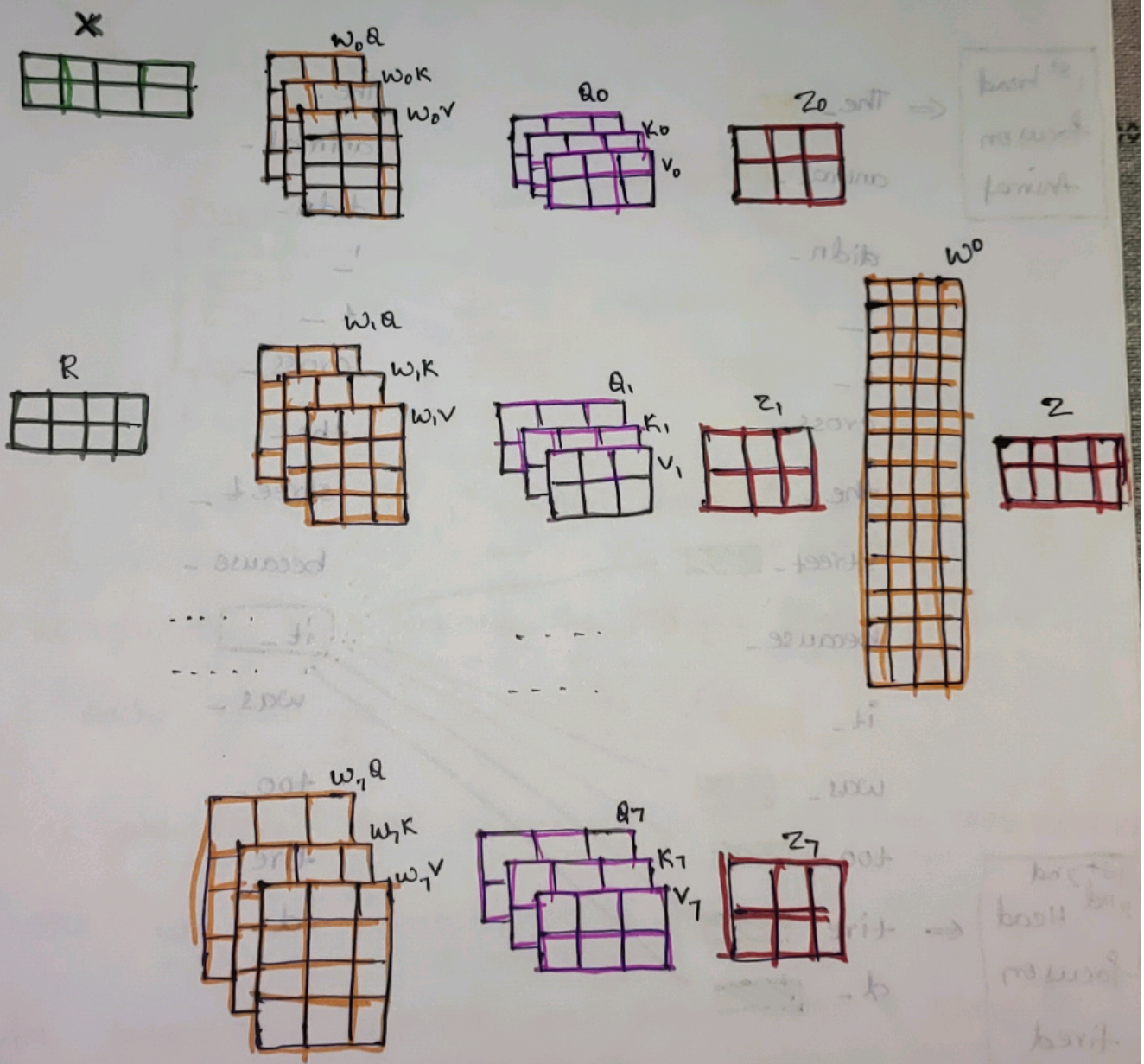


* now, how do we translate into one single head? Z

and these encoders are all Identical with the same dimensionalities.

here, W^0 multiplies with $(z_1, z_2 \ldots z_8)$ to create final Z, which will be passed further to feed forward NN & that will be sent to the next encoder.

Architecture:-



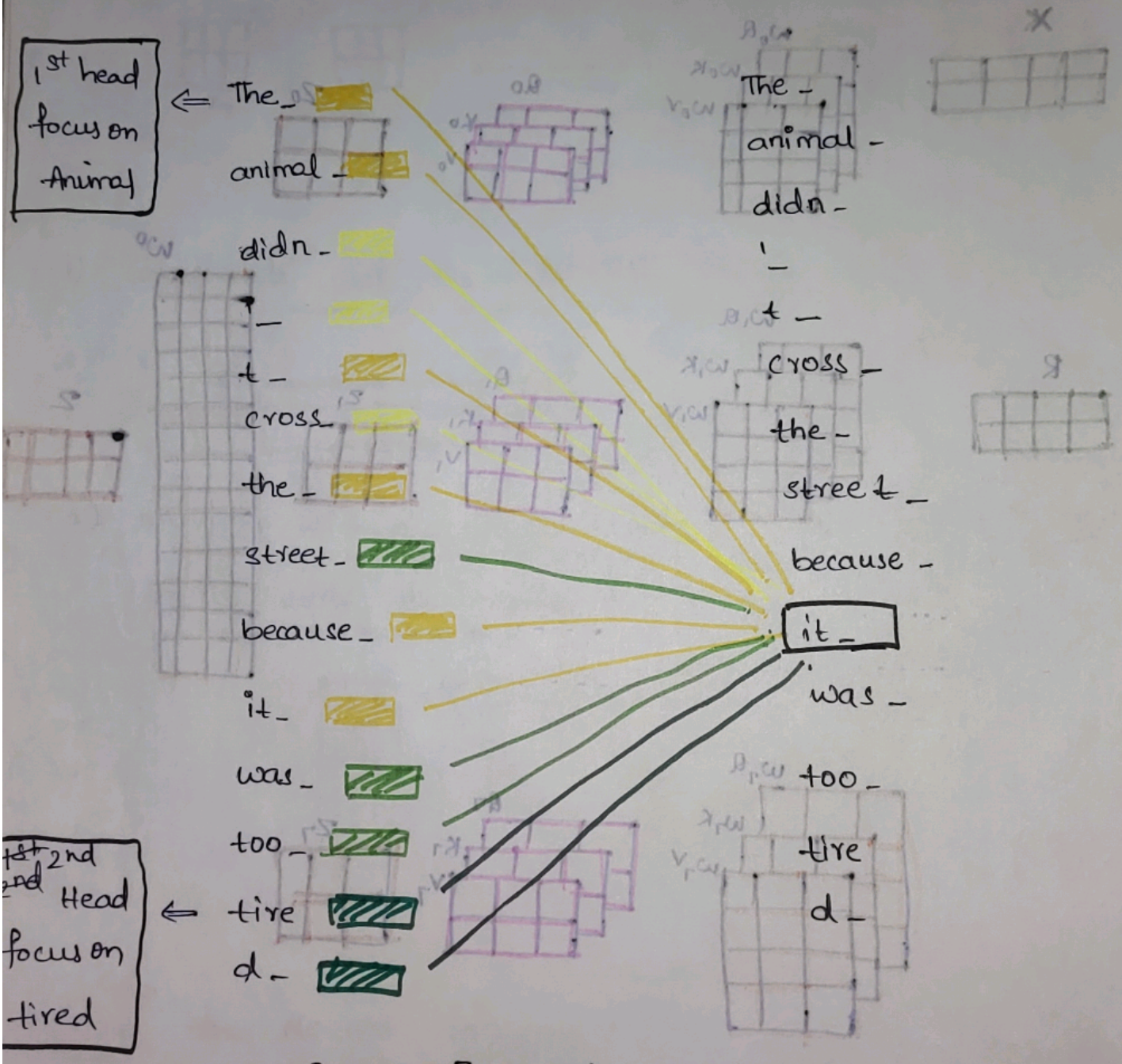
X ⇒ Input sentence ⇒ we embed each word ⇒ split into 8 heads.

We multiply X or R with weight matrices ⇒ calculate attention using the resulting Q/K/V matrices ⇒ concatenate the resulting

z matrices then multiply with weight matrix W^0 to produce the o/p of layer.

* here, W_iQ, W_iK, W_iV are trainable matrices & W^0 also trainable.

fino

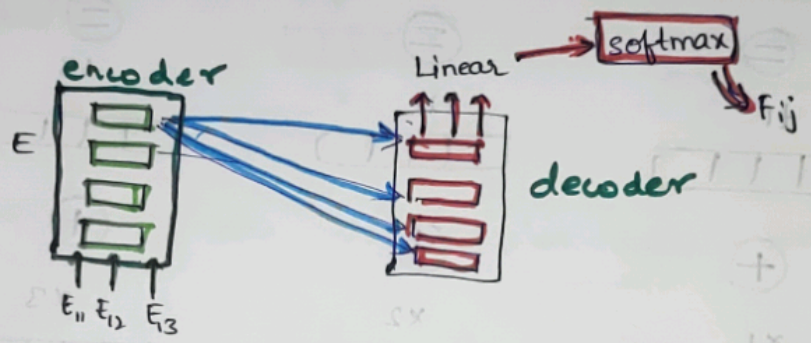


decreasing order of weights.

* by having multi-headed attention models, we can leverage the whole potential of model.

D → Dataset, consists of English & French words.

$D = \{ E_1, F_1 ; E_2, F_2 ; E_3, F_3 \ldots E_n, F_n \}$



E_{ij} → word of j^{th} position from i^{th} sentence.

→ we give the input sentences, the softmax layer generates each French word for sentence. (F_{ij})

→ here, we have loss associated with it. (like cross entropy) and we can back propogate them.

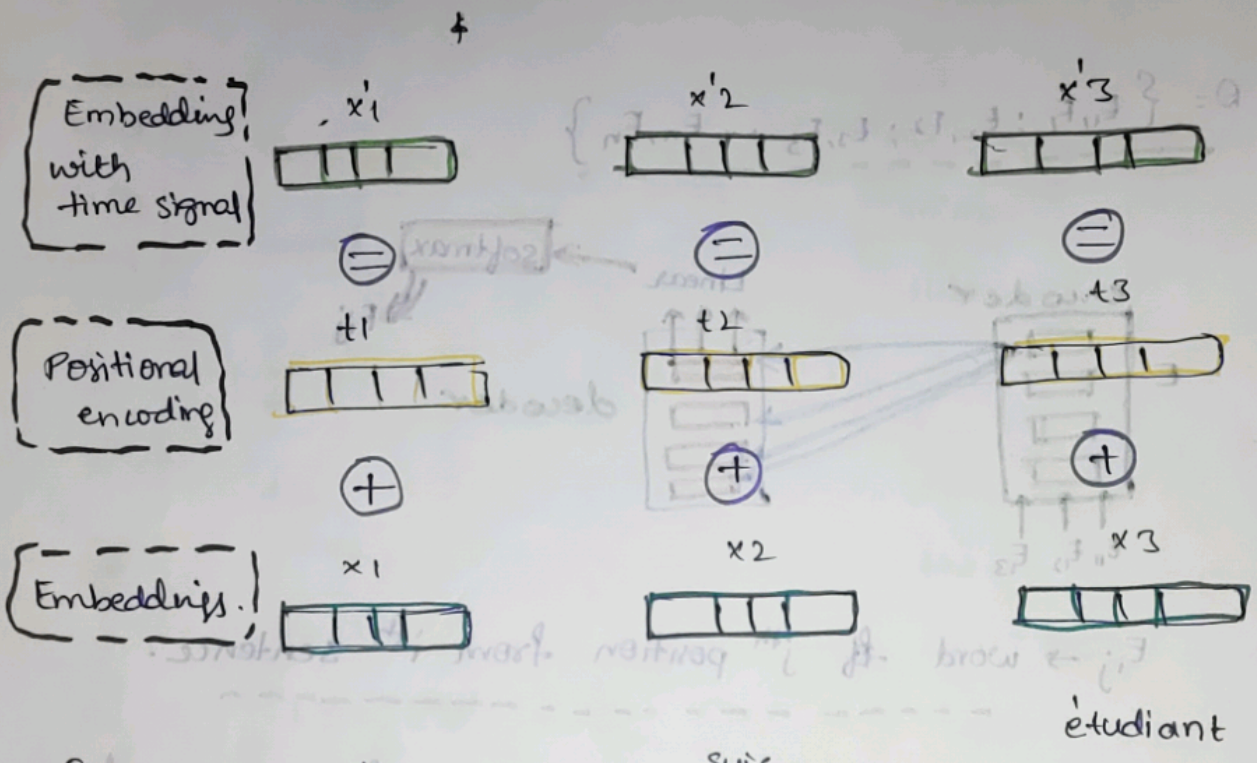
→ for just one encoder, we have bunch of weights where we can backpropagate & adjust them.

→ The model will learn all the weights wrt given in the dataset using back-prop.

→ The weights are initialized randomly.

→ These weights are creating attention.

But here, how do we encode the order of words?



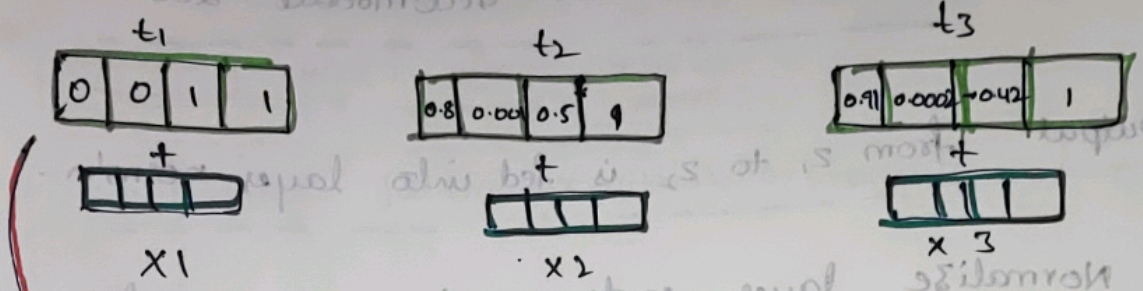
x1, x2, x3 are the word embeddings.

we will add a vector of same size (t1, t2, t3), the final vector is (x1', x2', x3')

we are doing positional encoding, we are trying to encode the fact that 'je' is before to 'suis' & 'suis' is before to "etudiant"

* x_1, x_3 are farther away, which implies the diff b/w x_1' & x_3' is more than diff b/w x_1' & x_2'.

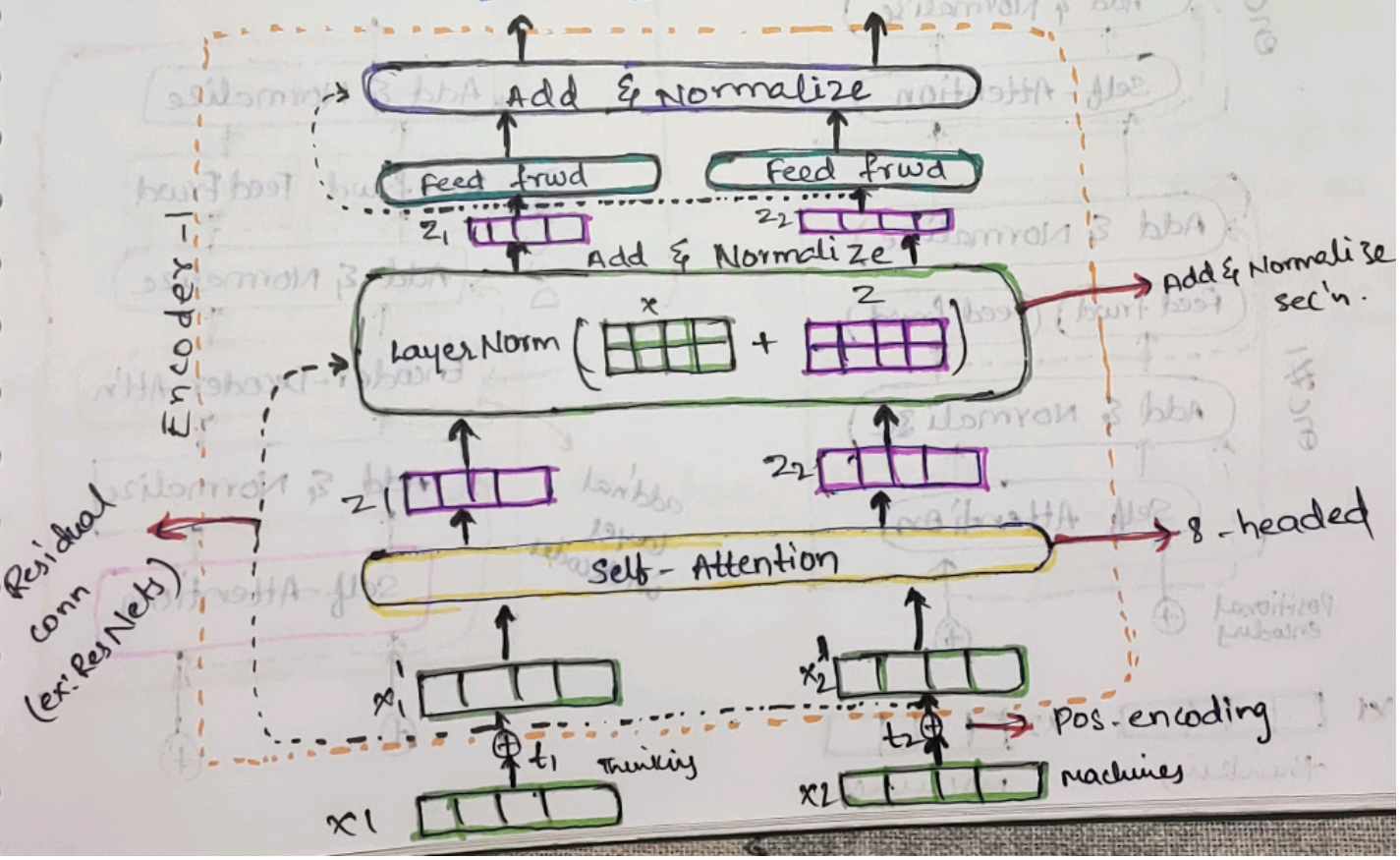
* How to design t_i's.??



these vectors are designed in such a way that, the dist b/w t_1 & t_2 is less than dist b/w t_1 & t_3.

& we can encapsulate the dist b/w words!

Actual encoder:



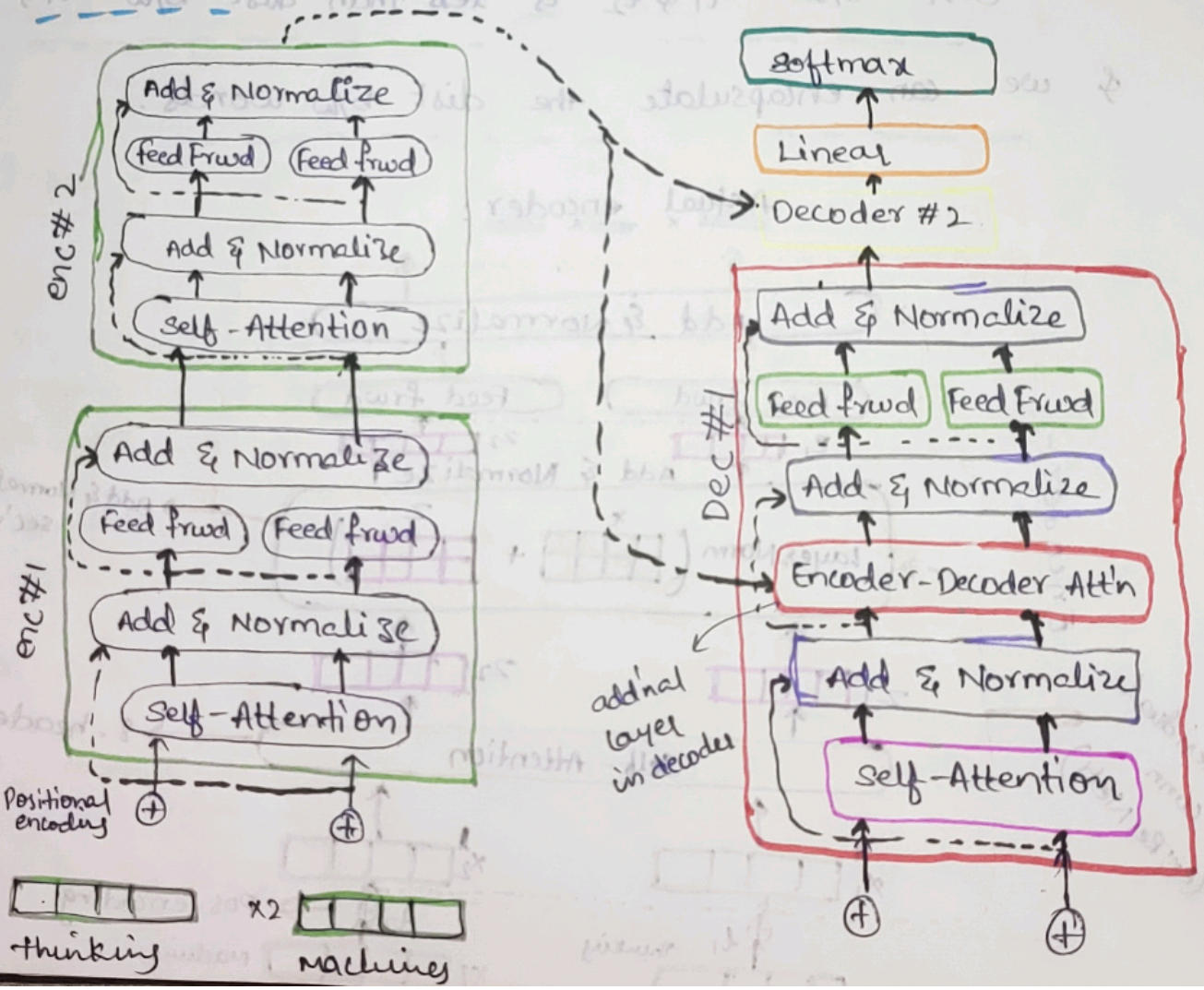
* How do we take in diff. length vectors i/p??

↳ take whole corpus, take the longest Eng. sentence that we have (ex: 200 words). The to vector is taken in such a way that it can accomodate 200 words

* The output from z_1 to z_2 is fed into layer norm'n.

* Add & Normalize layer sends o/p to next encoder.

* Res-Nets are Implemented If we feel certain layers are useless.



* The additional layer (encoder-decoder Attention) gets input from encoder.

* The encoder modules capture all the info from the i/p. the info will pass as a vector to decoder.

* Decoding w.r.t time steps:-

Links: jalammar.github.io/images/transformer_decoding_1.gif

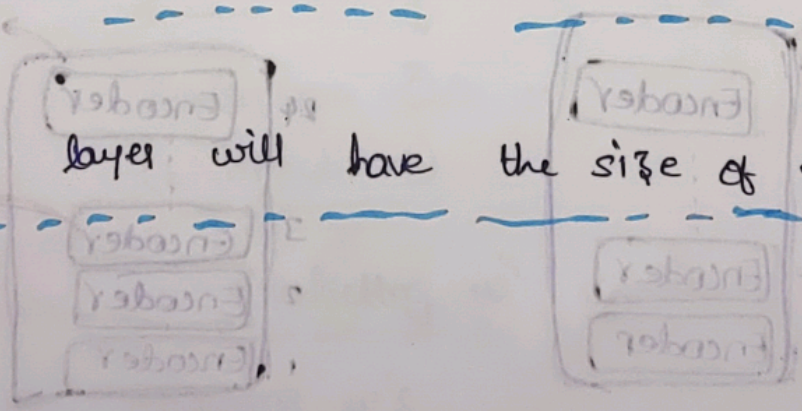
" " _decoding_2.gif

* Note: we send the whole input in 1st time step itself (to the encoder)

* Encoding → 1-time thing

* decoding → time-by-time.

* generating 2nd o/p in decoder → take 1st o/p, send it as i/p from encoders to decoders, then 3rd, 4th.. so



* Softmax layer will have the size of vocabulary.

* Google - translate uses BERT

BERT: Bi-directional encoder representation for from Transformers.

⇓

Transfer learning in NLP.

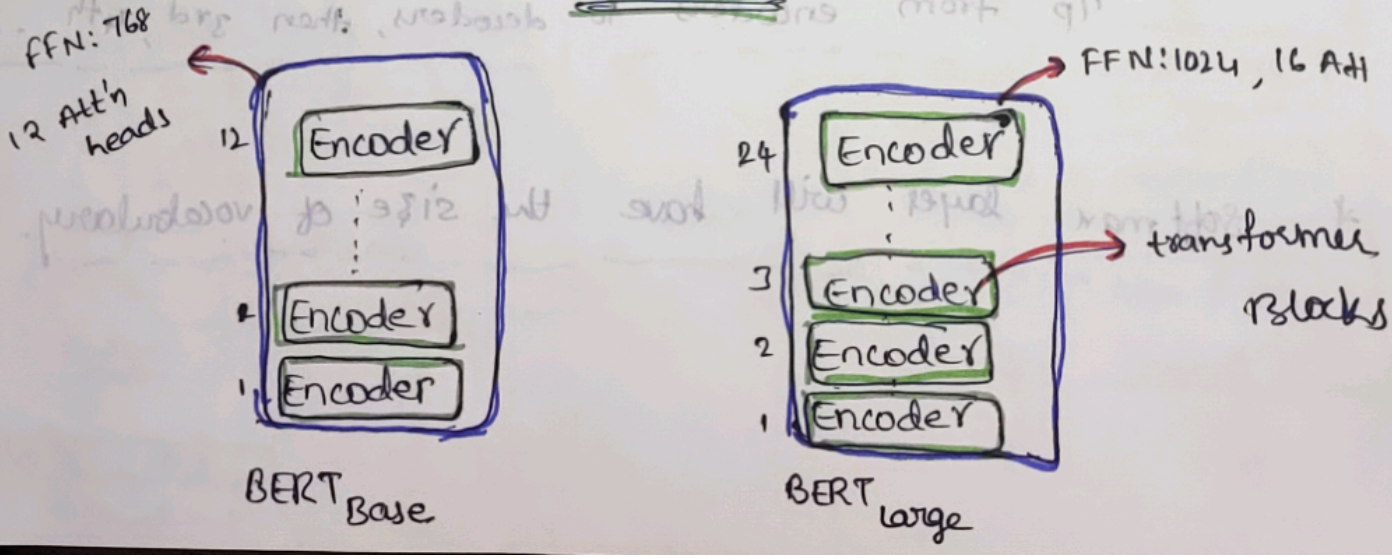
Pdf on Transformers

⇓

arxiv.org/pdf/1706.03762.pdf

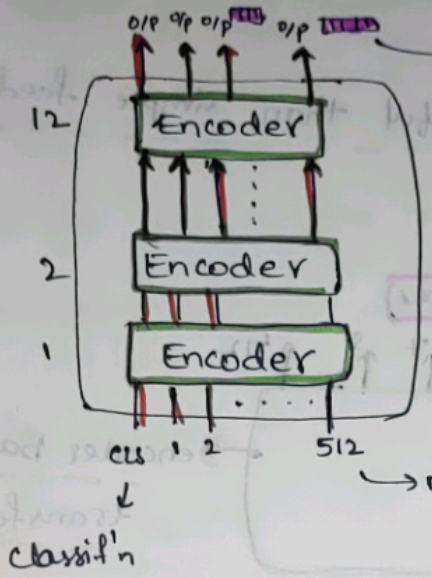
* Text summarization ⇒ Transformers work well.
* Attention layers are implemented by default in Keras
* BERT is a special type of transformers.
* BERT → sent 2 vec.
* BERT large ⇒ comparable to Humans on multiple NLP tasks.

Encoder-stack (No decoders)



* Max of 512 words in a sentence can be given to BERT.

Spam/Not (Transfer learning)



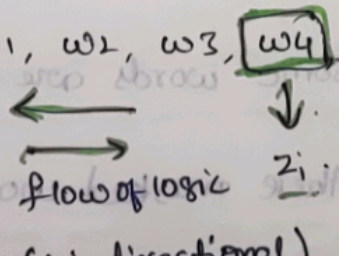
⇒ Given a sentence, it give output as translated sentence.

* Bert training:-

↳ A Bi-dirnal model.

⇓

looking at words before & after.

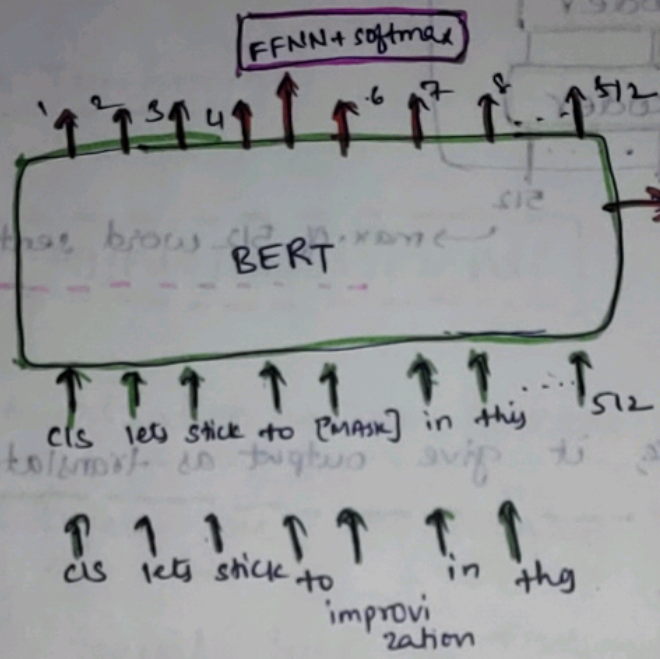


here, we are predicting w^4, based on the context of remaining w's.

⇒ If we have a big sentence, we do something called Masked language model.

⇓

This is far more powerful than simple feed-forward model.



⇒ we mask a word here and try to predict what a word should be.

⇒ Input → seq of words (some words are masked)

o/p → prediction of their masked words.

⇒ we randomly mask 15% of the tokens & label it as mask.

→ Based on all the other words, the o/p of mask will be Predicted.

→ now, what if we have tasks, where we input 2 sentences ??

* → If we give two sentences A&B & we've to predict the Prob that B comes after A.

→ start with CLS, use [SEP] b/w the sentences that we take as input.

* BERT → sentence-pair classif'n tasks

→ Single-sentence class'n task.

→ Q&A system.

→ parts of speech tagging.

→ feature extraction.

Ref code: jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/