

Transformers

↓
BERT

- * BERT is special type of transformers.

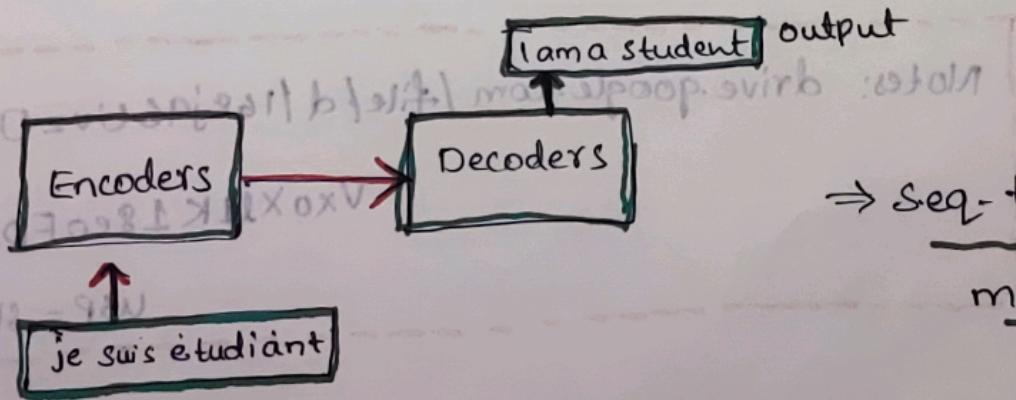
BERT → Bi-directional Encoder Representation from

Transformers

- * To understand transformers, we need to understand attention, to understand attention, we need to understand encoder-decoder, models.

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

- * Transformers: top-down Approach:



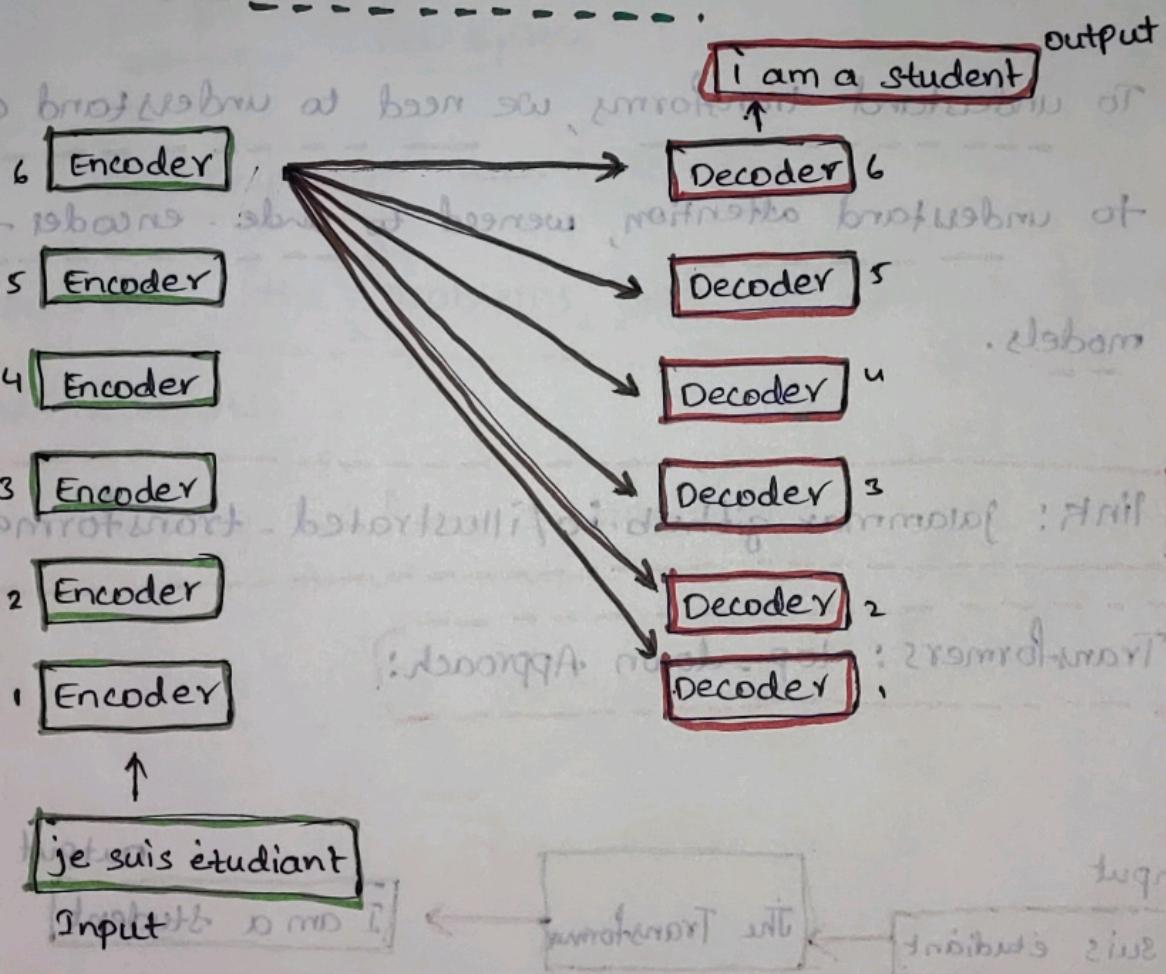
→ This is the top-down approach.

→ 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)



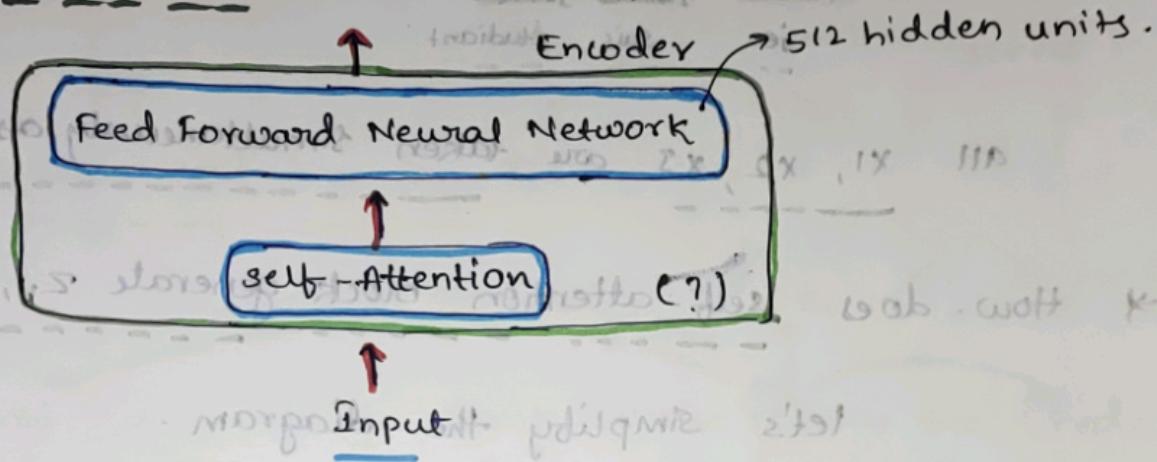
Notes: drive.google.com/file/d/168j96UvzOkwvbrhUsVx0Xl1K18eoFDSKb/view?

USP = Sharing

* each of these encoders are identical

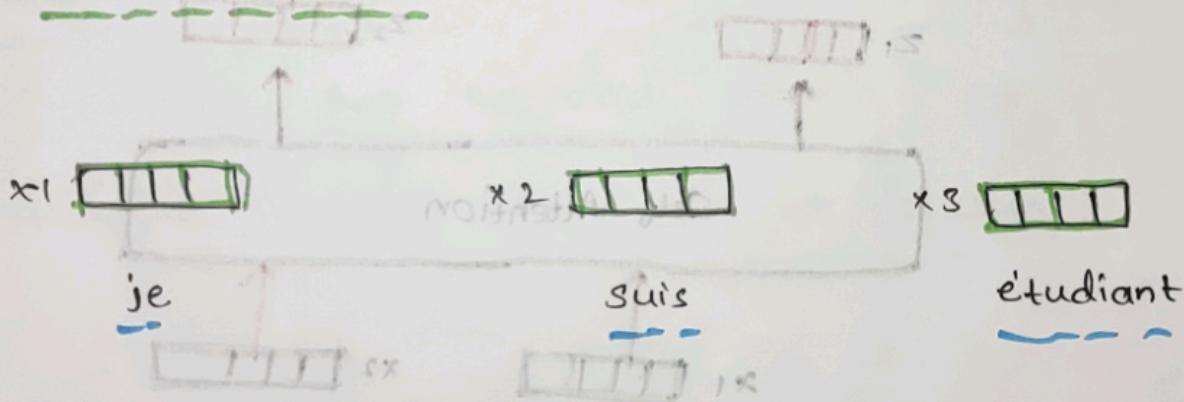
* Here, the output of last encoder is fed to all the decoders.

* Structure of encoder:

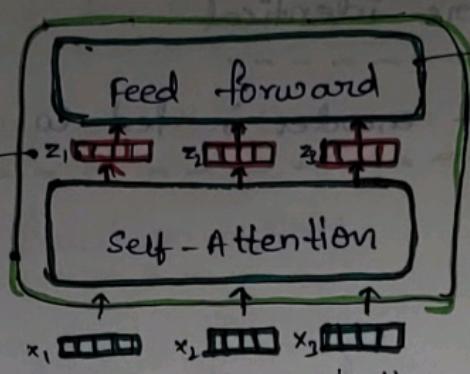


input: word-vectors/tensors.

→ take Jlp as non-english word, convert it into 512 dimensional vector (w_2vec)



each of these are 512-dimensional

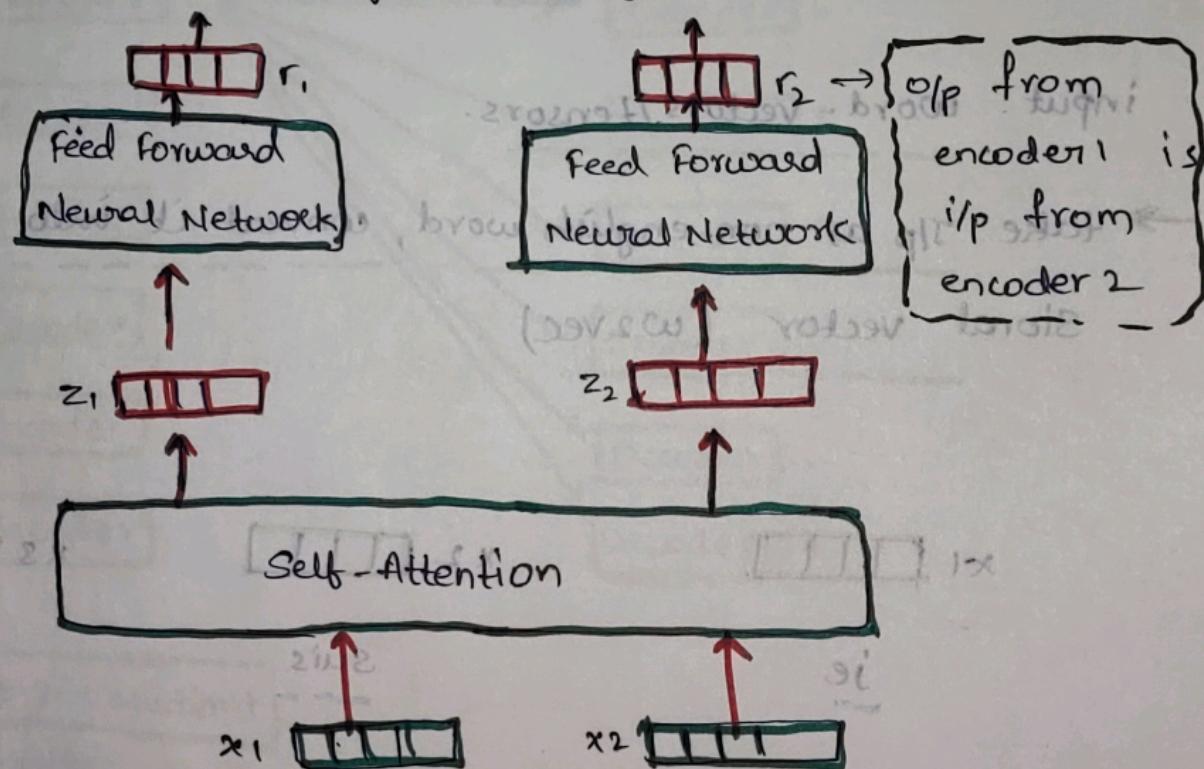


%p
corresponding
to x_1

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.

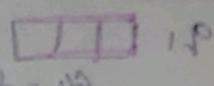
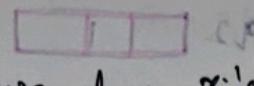
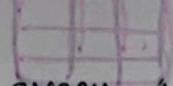
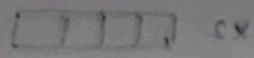
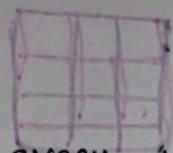
Here, o/p from encoder-1 goes as i/p to encoder-2
(r_1, r_2)

* what is self-attention?

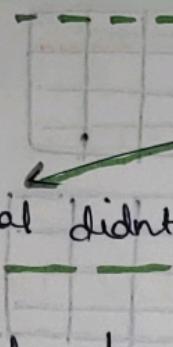
input: "The animal didn't cross the street because it was

"too tired"

refers to animal or
street?

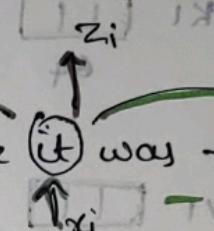


→ for every word, we have x_i 's



should be connected to this

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



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

$$p_0 = x_b$$

(animal) \rightarrow (tired) \rightarrow BCN

→ We should pay attention to the word animal & tired, more than the word 'street'

attention refers to where we are focusing.

self-attention BLOCK

position feature sets BCN, CW, PW for word

entity and prior most often seen test

(repeat multiple times for word entity)

How z_i created from x_i ??

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

2nd step: 1x of .280 is dotprod of inputs.

Step-2
multiply
 k_i with q_i

input

a big item

Thinking

from just
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 $8 (\sqrt{dk} = 64)$

$$112 \Rightarrow 14$$

$$96 \Rightarrow 12$$

4th step:

Apply softmax.

$$14 \Rightarrow 0.88$$

$$96 \Rightarrow 0.12$$

Step-5:-

The softmax we get, multiply with values.

$$v_1 \quad \boxed{\text{1111}}$$

(we are trying to generate z_i corr. to x_i (x_i 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]

① \Rightarrow

$$x \cdot w^Q \quad Q$$

$$\begin{matrix} x_1 \\ x_2 \end{matrix} \quad \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} * \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} = \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} \quad Q_1 \quad Q_2$$

$$x \quad \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} * \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} = \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} \quad K_1 \quad K_2$$

$$x \quad \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} * \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} = \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} \quad \leftarrow v_1 \quad \leftarrow v_2$$

② \Downarrow

$$JP = \underbrace{\left(\begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} * \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} \right)}_{\sqrt{dk}} \quad K^T \quad V$$

softmax $\left(\frac{\boxed{1111} * \boxed{1111}}{\sqrt{dk}} \right)$

$$P = \begin{matrix} \boxed{1111} \\ \boxed{1111} \end{matrix} \rightarrow 21 \quad \rightarrow 22$$

- * → 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 tiredness 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.

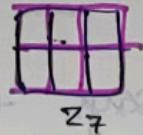
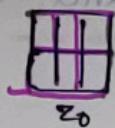
(more hyper-parameters encodes more Info)

8-headed attention mechanism:

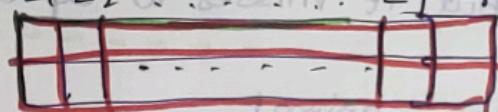
Thinking
machines



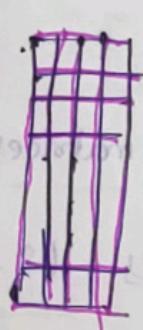
calculating attention separately in
eight different attention heads



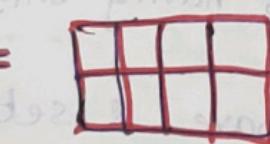
i) concatenate all the attention heads



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



→ another weight matrix (W^o)

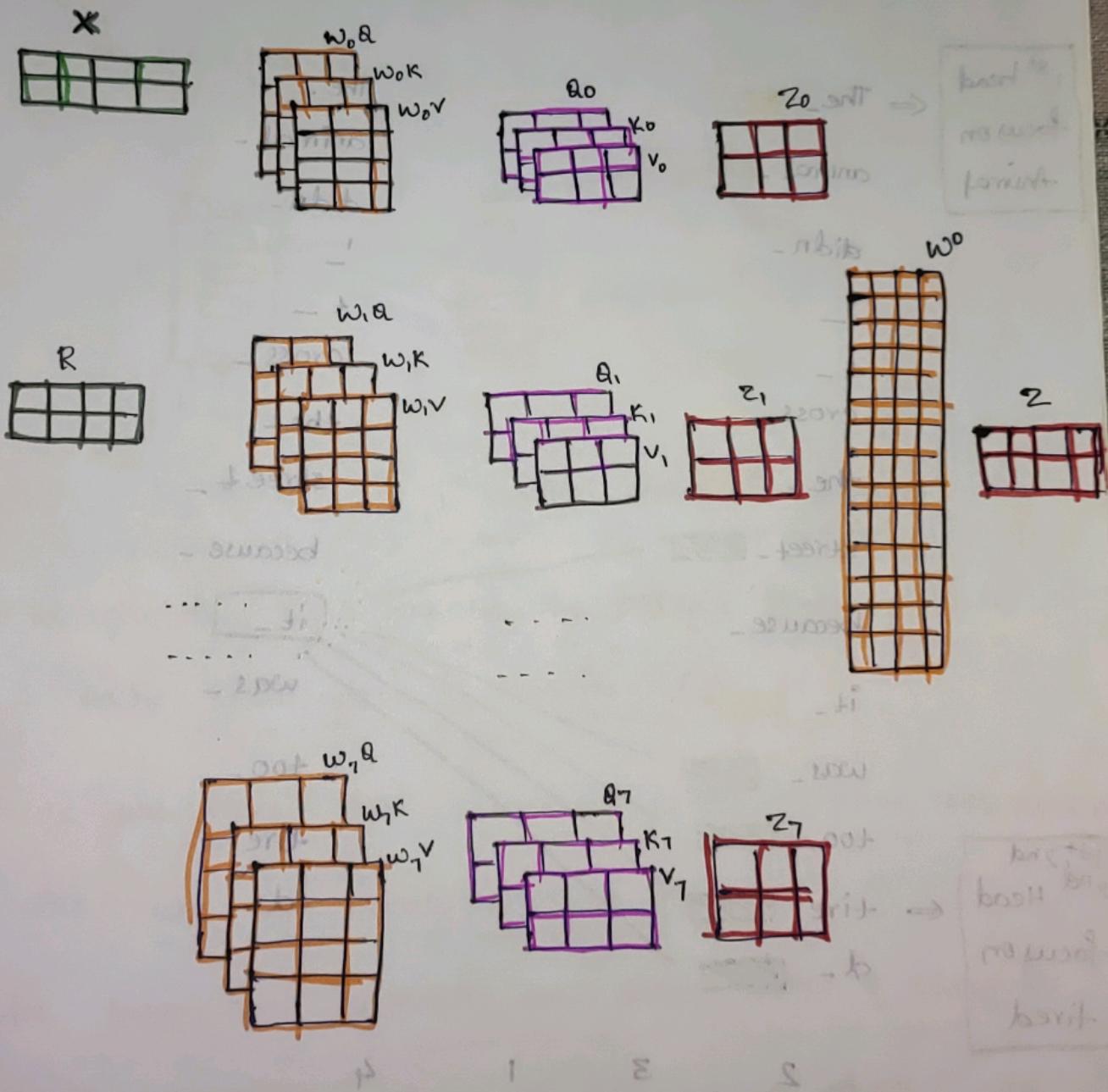


* now, how do we translate into one single head?

and these encoders are all identical with the
same dimensionalities.

here, w_0 multiplies with (z_1, z_2, \dots, z_8) , to create final z , which will be passed further to feed forward NN & that will be sent to the next encoder.

Architecture:



$X \Rightarrow$ Input sentence \Rightarrow we embed each word \Rightarrow split into 8 heads.

we multiply X or R with weight matrices \Rightarrow calculate attention

using the resulting Q/K/V matrices \Rightarrow concatenate the resulting

Z matrices then multiply with weight matrix w^o to

produce the o/p of layer.

* here, $w_i, Q, w_i K, w_i V$ are trainable matrices & w^o also

----- trainable -----

Run

1st head
focus on
Animal

The animal

B, C, W
V, C, W
The animal didn't

X

didn't cross the street because it was

B, C, W
V, C, W
cross the street

X

2nd Head
focus on
tired

tire d-

B, C, W
V, C, W
too tired

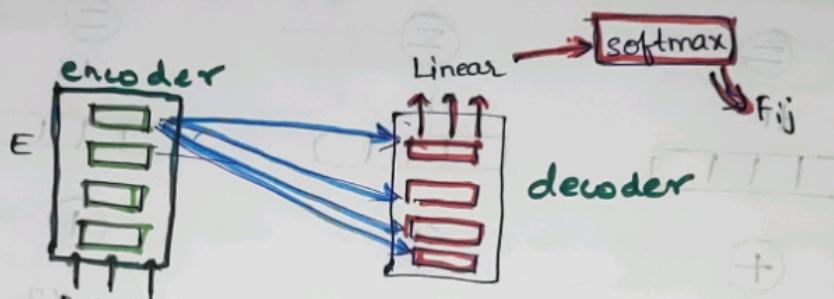
2 3 1 4

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.

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

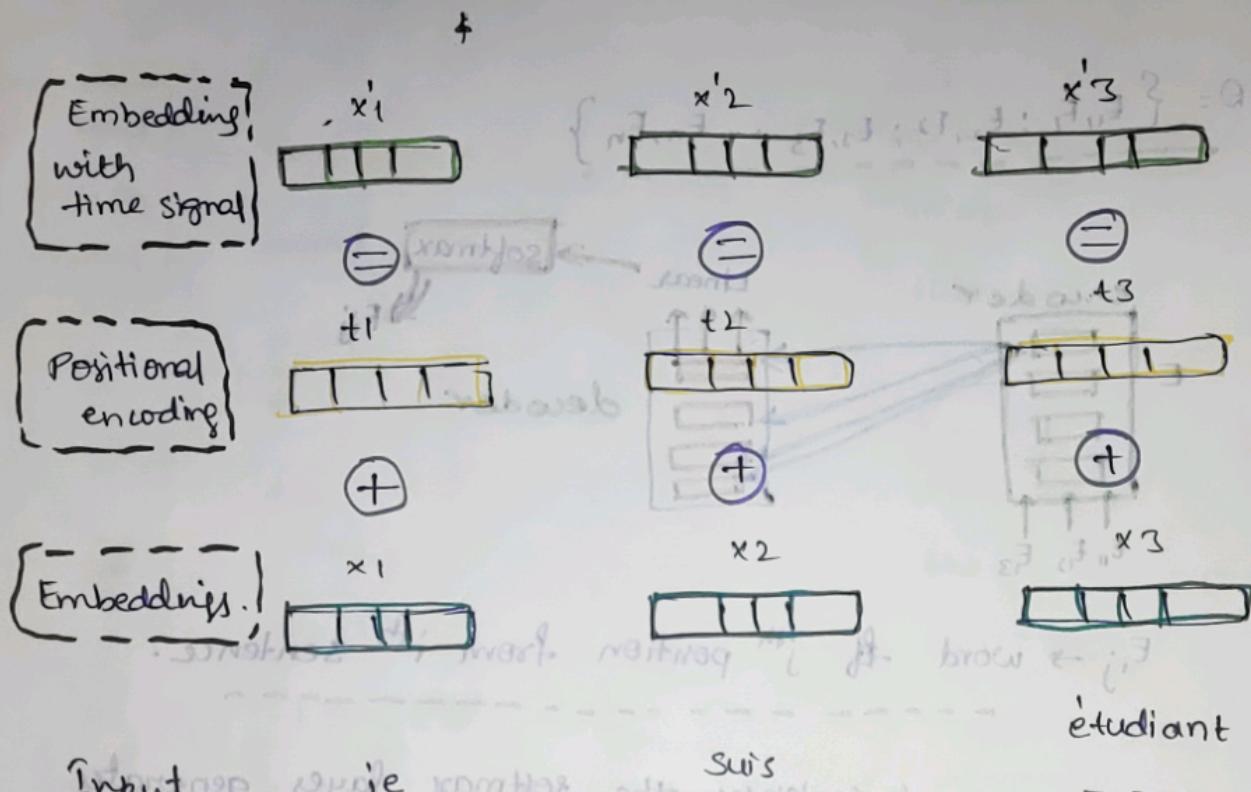


$F_{ij} \rightarrow$ word f_j j^{th} position from i^{th} sentence.

- 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 backpropagate them.
- for just one encoder, we have bunch of weights where we can backpropagate & adjust them.
- The model will learn all the weights w.r.t 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?



Input

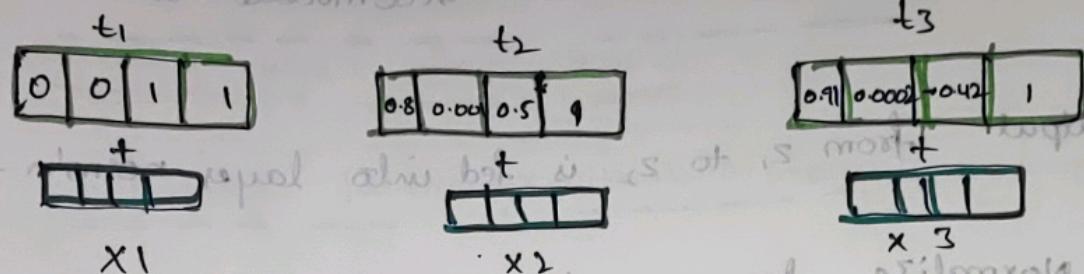
x_1, x_2, x_3 are the word embeddings.

* we will add a vector of same size (t_1, t_2, t_3), the final vector is (x'_1, x'_2, x'_3)

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

* 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 .

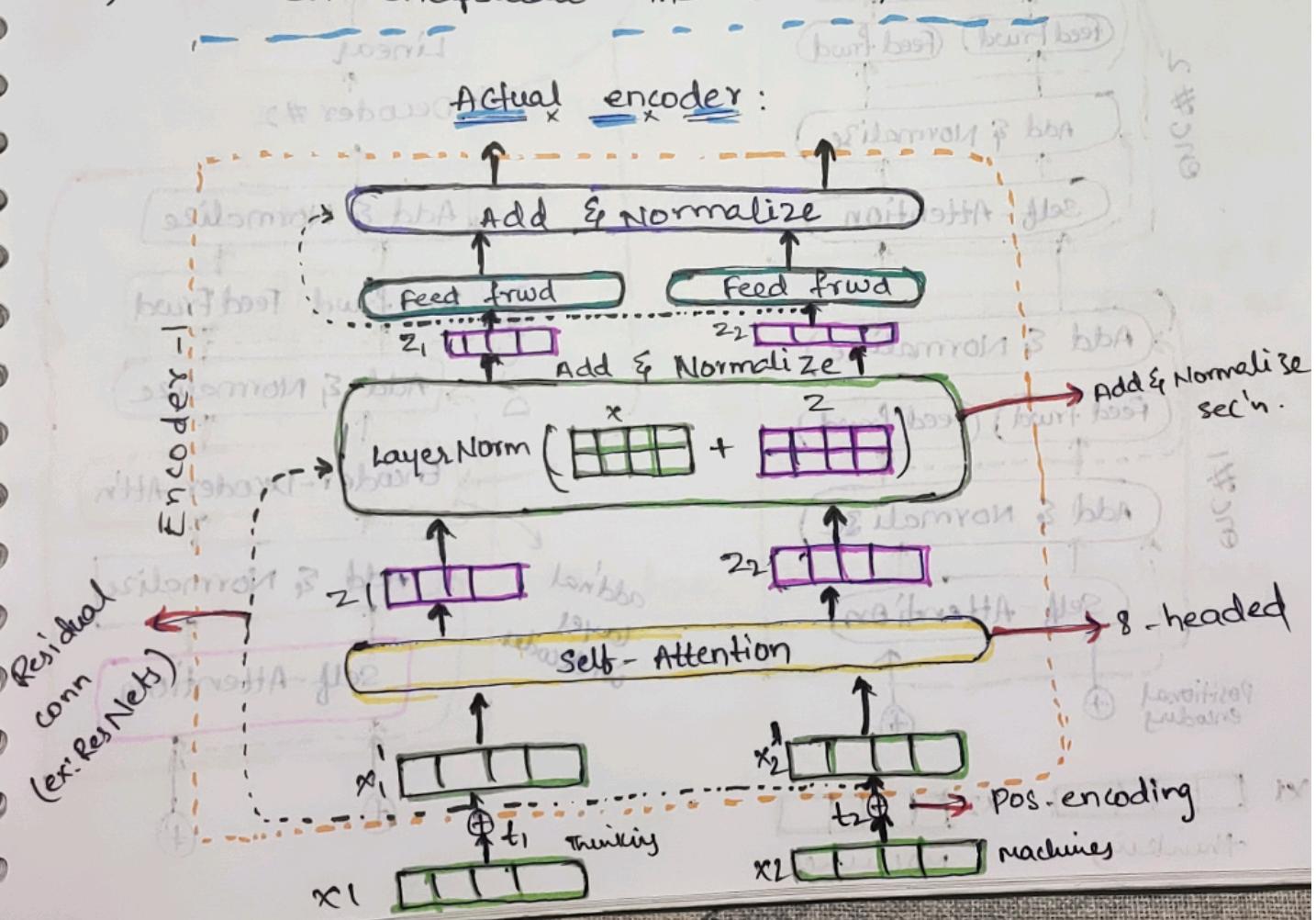
* How to design t's?



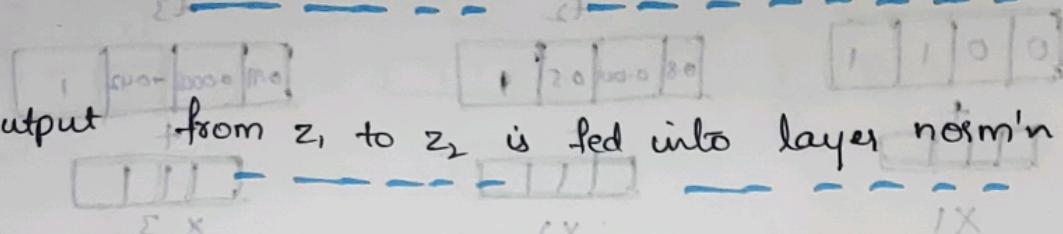
these vectors are designed in such a way that,

the dist b/w t_1 & t_2 is less than dist b/w t_1 & t_3 .

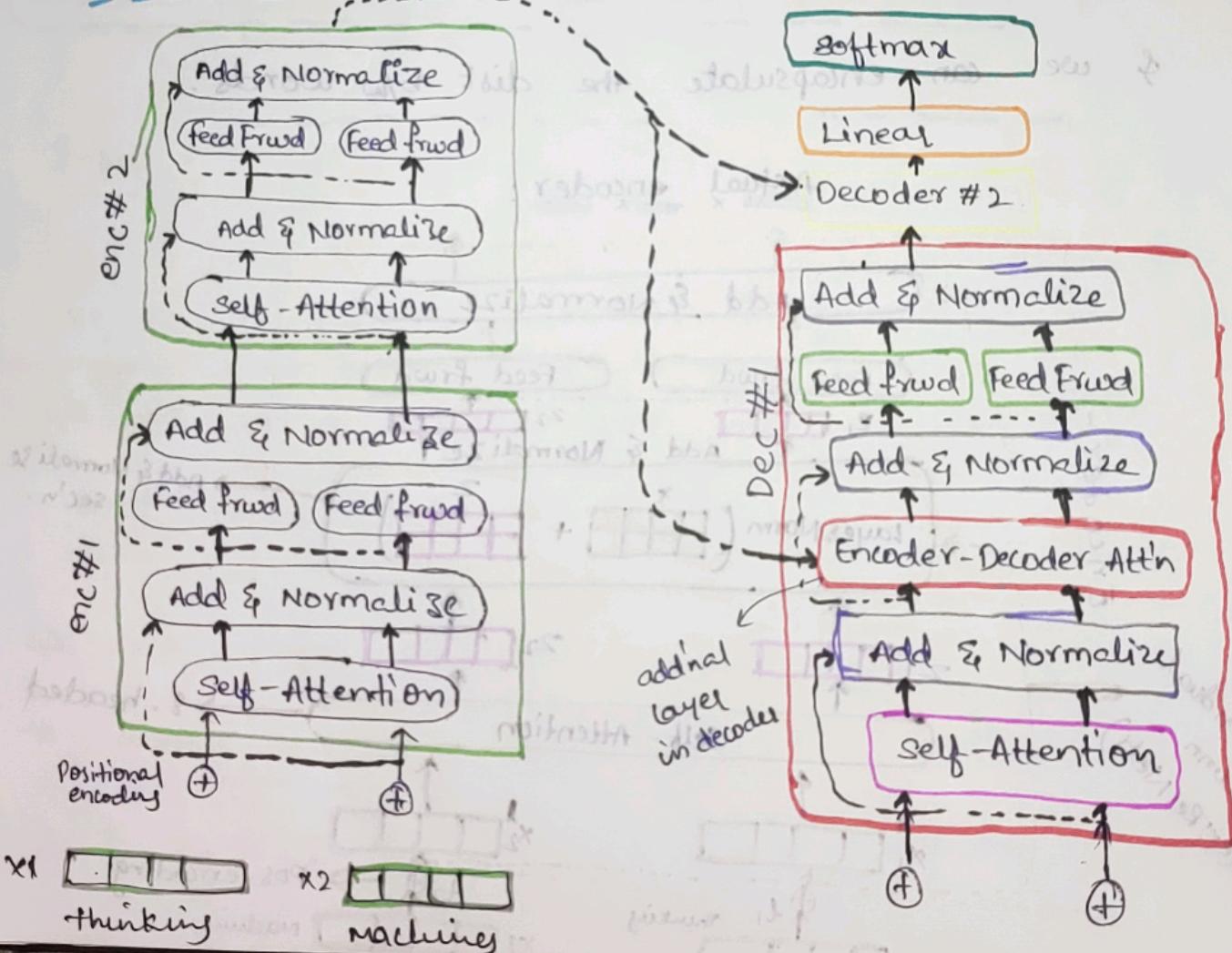
& we can encapsulate the dist b/w words.



- * 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 vector is taken in such a way that it can accommodate 200 words

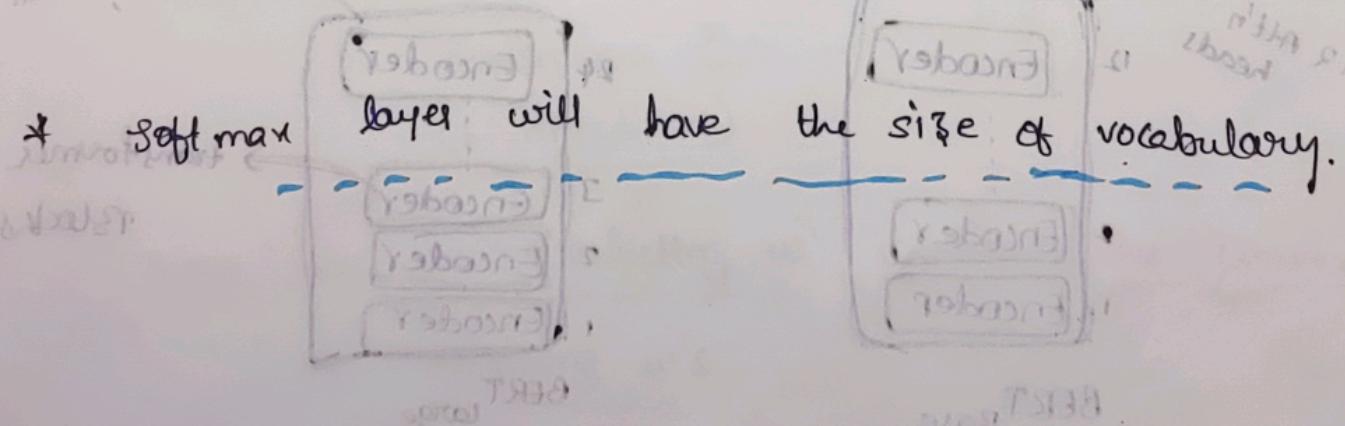


- * The output from z_1 to z_2 is fed into layer norm'n.
- * Add & Normalize layer sends o/p to next encoder.
- * Res-Nets are implemented if we feel certain layers are useless.



- * 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: https://jalammar.github.io/images/transformer_decoding_1.gif
https://jalammar.github.io/images/transformer_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 on.



- * Google - translate uses BERT
- BERT: Bi-directional encoder representation for from Transformers.
Transfer learning in NLP.

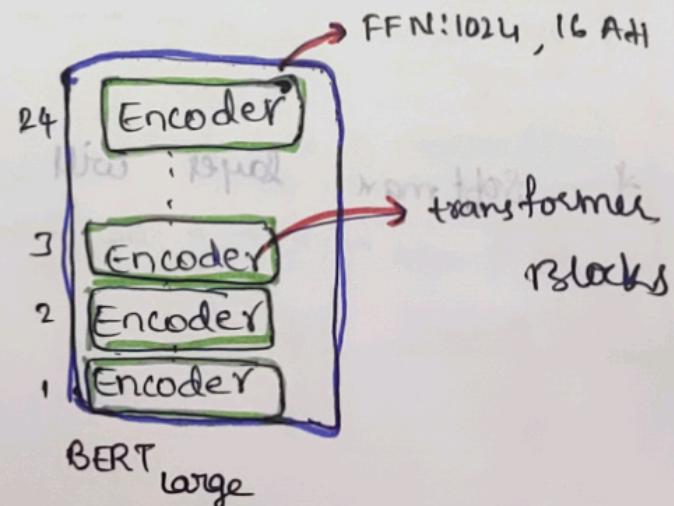
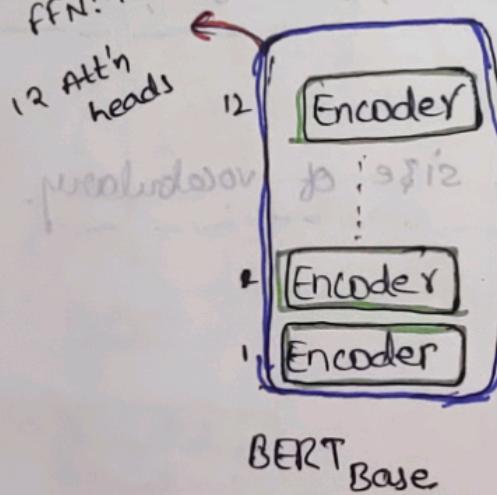
PDF on Transformers

arxiv.org/pdf/1706.03762.pdf

- * Text summarization \rightarrow Transformers work well.
- * Attention layers are implemented by default in Keras
- * BERT is a special type of transformer.
- * BERT \rightarrow sent & vec.

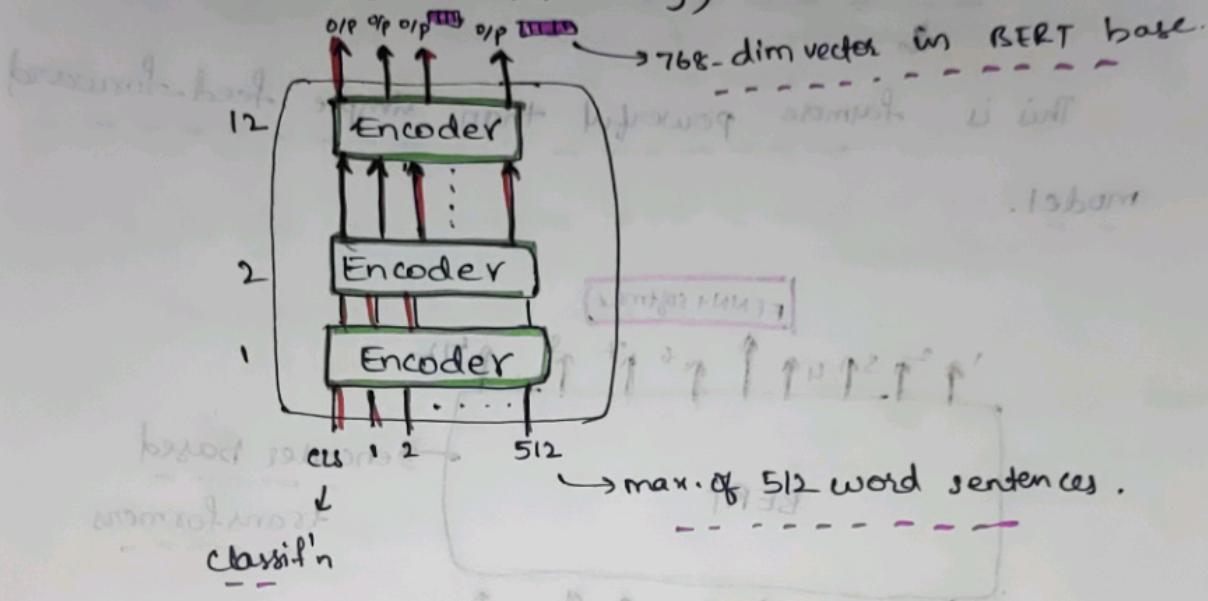
* BERT large \rightarrow comparable to humans on multiple NLP tasks.

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-directional model.

looking at words before & after.

w₁, w₂, w₃, **w₄**, w₅, w₆

(bottom up arrows)

↓

←

flow of logic

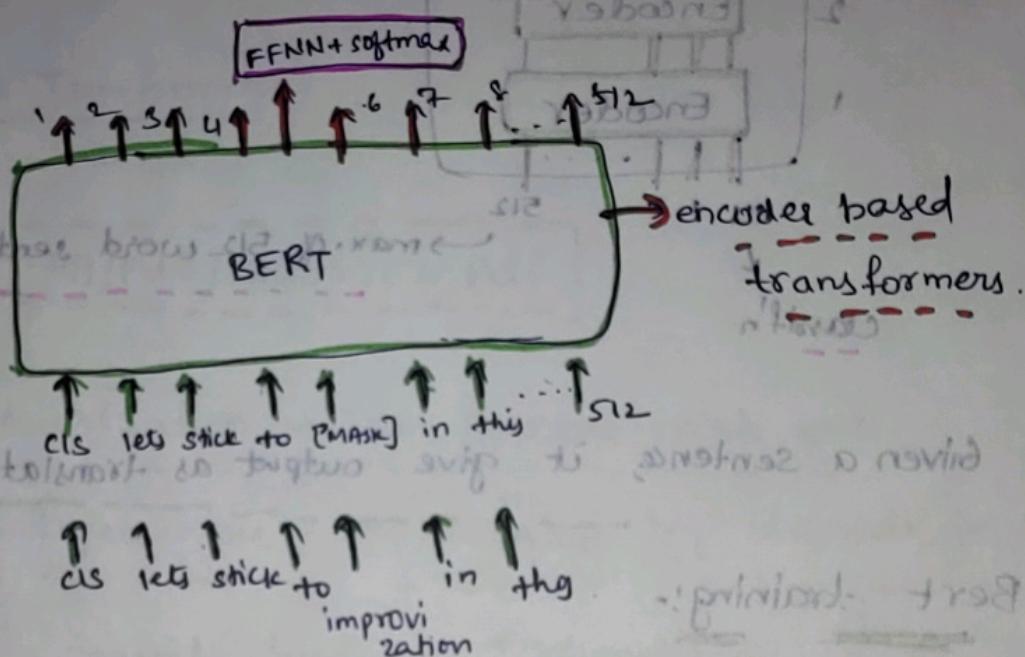
(bi-directional)

here, we are predicting w₄, 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 these masked words.

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

- 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 classif'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/