2025-04-14/16

TRANSFORMERS

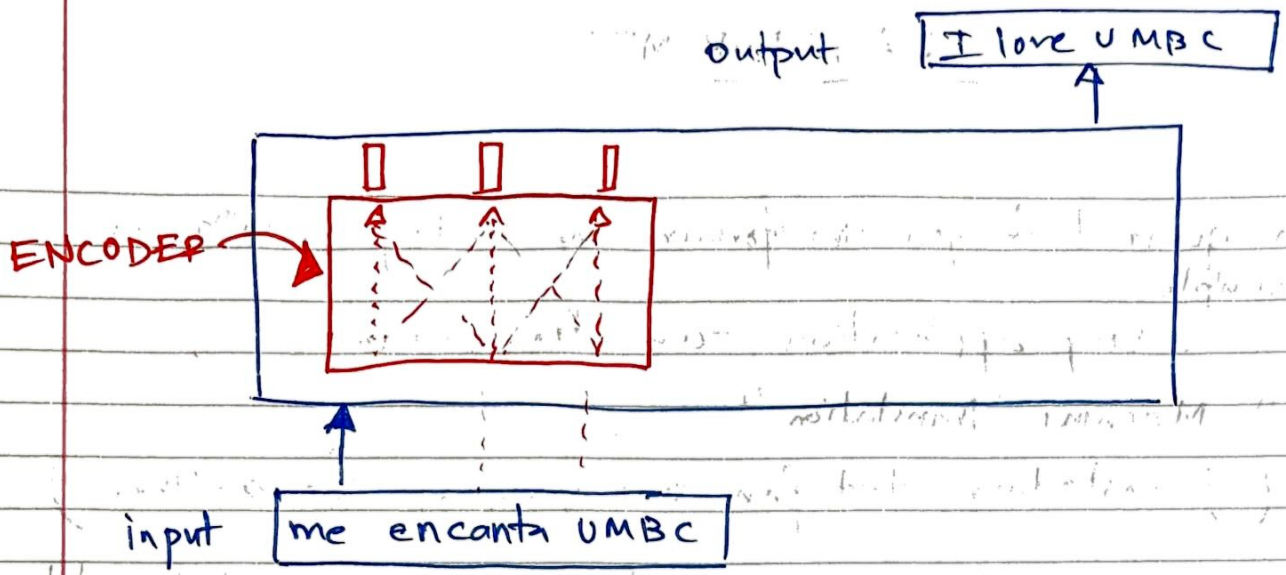
CMSC 475/675 Neural Networks

(TEJAS GOKHALE)

- "Transformers" is a neural network (ENCODER-DECODER) architecture
  - ↳ initially developed for language modeling (but also has other applications)
  - ↳ huge impact: the workhorse behind "LLM's" and AI chatbots that we all use
- This lecture:
  - ↳ a mathematical (linear algebra?) approach to explaining how a transformer works
  - ↳ Many blogs, videos, slides etc. on the web about transformers with good visualizations (use them to accompany the math or to build a better intuition from an application perspective)
  - ↳ My notes are derived from Soheil Feizi (UMD)

* EXAMPLE APPLICATION: Machine Translation
  - ↳ want to translate from Spanish to English

"I love UMBC" ⟷ "Me encanta UMBC"



we have an Encoder – Decoder architecture

what is the job of the encoder ?

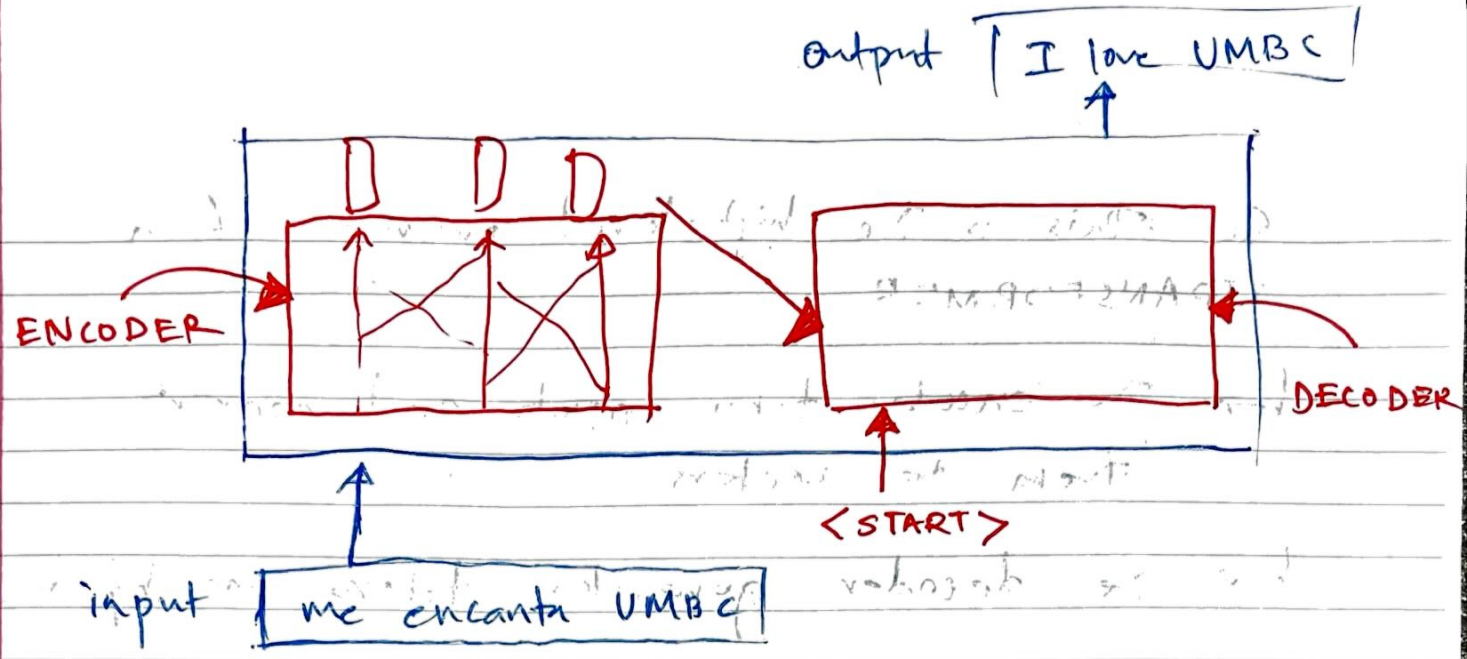
↳ Encode information for each word / token based on the context

↳ for each word, the output of the encoder will be a VECTOR

↳ the vectors will have a particular length (a design choice)

(we have seen word embeddings before. we will ~~test it~~ see a new way (transformers) to obtain these embeddings)

↳ we don't want single-word embeddings we want to integrate context info *



- the encoded vectors are then going to be sent to the DECODER
- the DECODER's job is to ~~translate~~ (do the task)
- The DECODER gets the <start> token as input and generate the next token

Input	Output	
<START>	"I"	STEP ①
<START> I	"love"	
<START> I love	"UMBC"	
<START> I love UMBC	"<STOP> <EOS>"	

- the decoder keeps generating the next token one by
  - ↳ until the <STOP> token is generated

- So, this is the high-level overview of a TRANSFORMER

↳ the encoder takes inputs and converts them to vectors

↳ the decoder generates tokens one by one

↳ every step of the decoder depends on the previously generated token

↳ * that is why these models are also called AUTO REGRESSIVE MODELS

So that's Encoding - Decoding at a high level

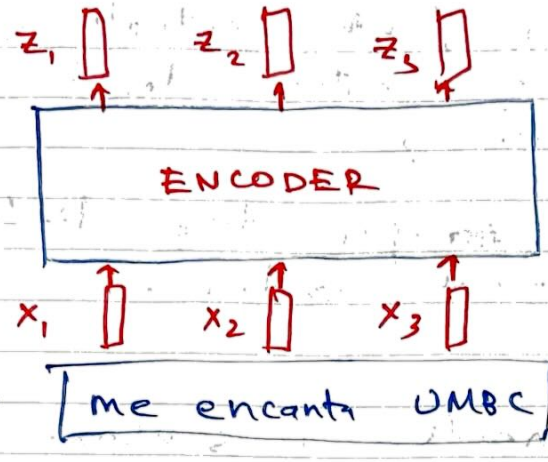
This lecture :

(1) HOW CAN WE encode information (with content) into vectors ?

(2) HOW CAN WE send that informations to the decoder ?

(3) HOW CAN WE use it in autoregressive fashion to decode and generate output tokens.

* LET'S START WITH THE ENCODER



- by now, we have seen that we can't directly input words to a neural network
  - ↳ we need to convert them to vectors
  - ↳ "embedding"

- input of the encoder: 512x1 word embeddings x
- output of the encoder: 512x1 word embeddings z

.... ok SO WHY DO WE NEED THE ENCODER?

↳ to encode CONTEXT !

Q: what if I change the order of $x_1 x_2 x_3$?

A: z_1, z_2, z_3 should change
(the vectors should be different)

Tejas loves UMBC
UMBC loves Tejas
} different meaning!
∴ $z_1 z_2 z_3$ should be different!

- ~~Current~~ architectures / LM methods we have seen until now either don't consider the order,

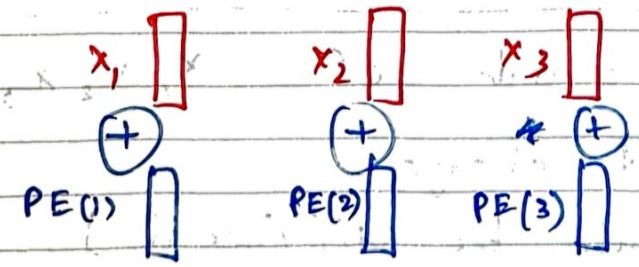
  or are bad at generating different vectors for different orders.

- This problem needs a solution.
- the Transformers Paper (2017) Vaswani et al. provided that answer in an effective way.

* IDEA: add "position vectors" to the input

> POSITION EMBEDDINGS
> some vectors unique to each position

↳ instead of $x_1\ x_2\ x_3$ as input



size: 512 x 1 (so that we can add)

what should the PE vectors be?

↳ vectors that are unique for each position

↳ Vaswani et al. used sine function

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

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

pos : position in the input

d : dimension of vector (512)

i : index over vector dimension

(if $d = 512$, $i \in [0, 255]$)

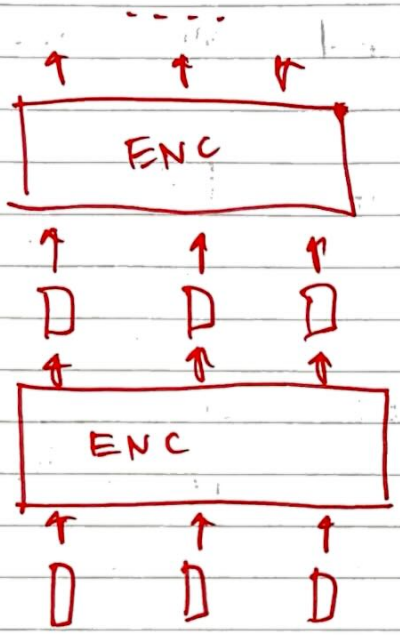
10000 : a large base

(Wavelengths : geom progression from 2π to 20000π

→ smaller i : more oscillations
larger i : less oscillations

* WE CAN STACK ENCODERS

(just like we can stack conv/ layers in a CNN linear)



(more layers)

(more parameters)

(model very complex functions)

Vaswani et al. stack 6 encoders

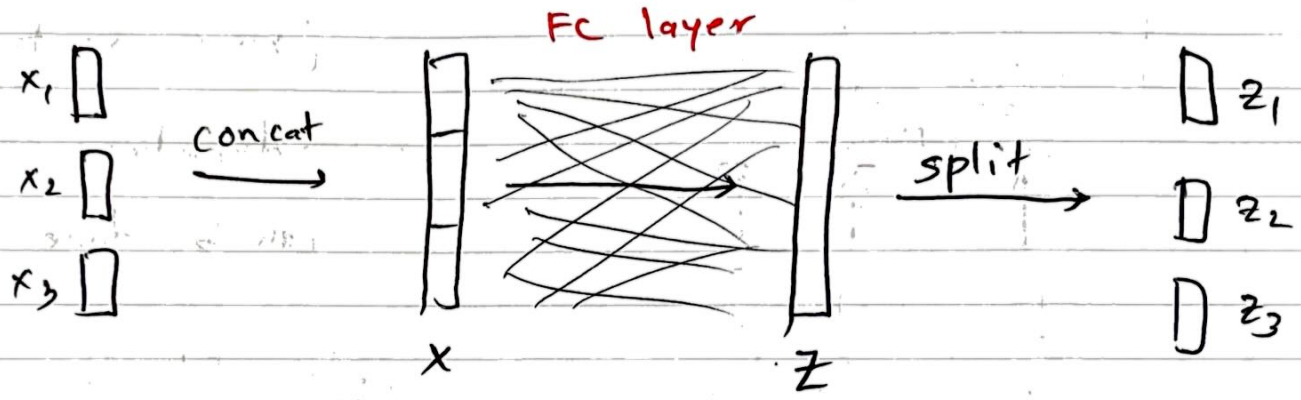
(hyper parameter)

* OK FINE, but how does the ENCODER actually work ?

how can we incorporate context ?

* ENCODER DETAILS :

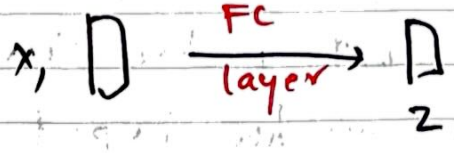
- what is the simplest NN layer we know ?
  - ↳ Fully-connected layer
- one idea



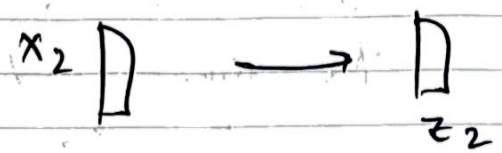
$(3 \times 512) \times (3 \times 512) = 9 \cdot 2^{18}$ weights

- another idea

ONE (SMALL N.N.) per token



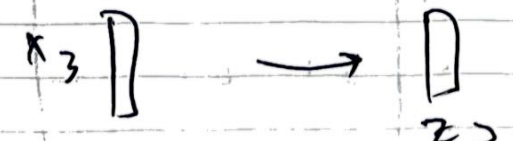
~~ok but~~ $3 \cdot 2^{18}$ weights



ok but context ?

this is dumb ?

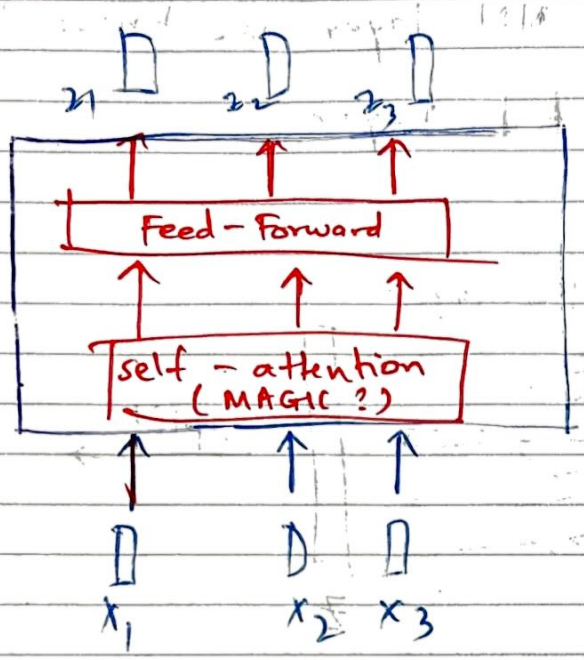
what can we do ?



ATTENTION LAYER →

(SELF)

* ATTENTION LAYER allows us to ENCODE CONTEXT



Feed Forward

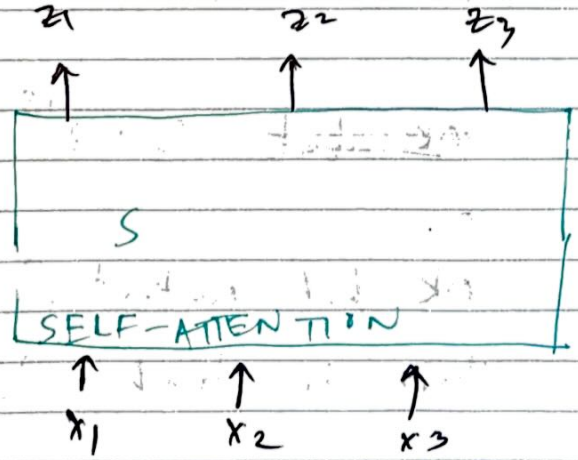
- applied to each vector separately (no cross-talk)
- advantage: can be done in parallel

Self-Attention

- something that incorporates context information

What are some choices for the Self-Attention Layer

OPTION ①



Consider a matrix P with probabilities of how important each word is to the other word

example:

$$P = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0.1 & 0.7 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix} \end{array}$$

* P is the attention matrix

↳ every row denotes how important each token is to one of the words

(a probability vector that sums to 1)

Output:

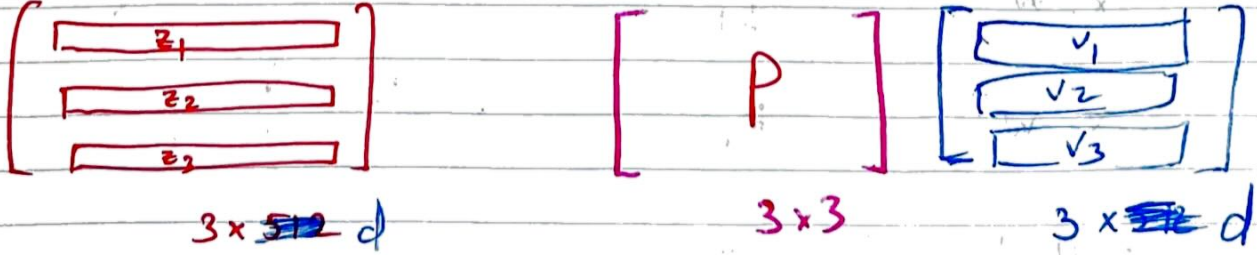
$$\begin{cases} z_1 = 0.8\, v_1 + 0.2\, v_2 + 0.0\, v_3 \\ z_2 = 0.1\, v_1 + 0.7\, v_2 + 0.2\, v_3 \\ z_3 = 0.3\, v_1 + 0.3\, v_2 + 0.4\, v_3 \end{cases}$$

where

$$v_1 = x_1 W^v$$
$$v_2 = x_2 W^v$$
$$v_3 = x_3 W^v$$

F-C- layer

$$\therefore Z = P \cdot V$$



$3 \times d$ $\quad$ 3×3 $\quad$ $3 \times d$

* Learnable parameters : W^v , P

Any issues ?

P is a $N \times N$ matrix

↳ N is the number of input tokens

↳ if $N = 10000$, P has 10^8 parameters

↳ N is variable length (depending on input length

NOT GOOD.

need a better idea

IDEA

we two intermediate matrices

W^k , W^q

W^k → keys

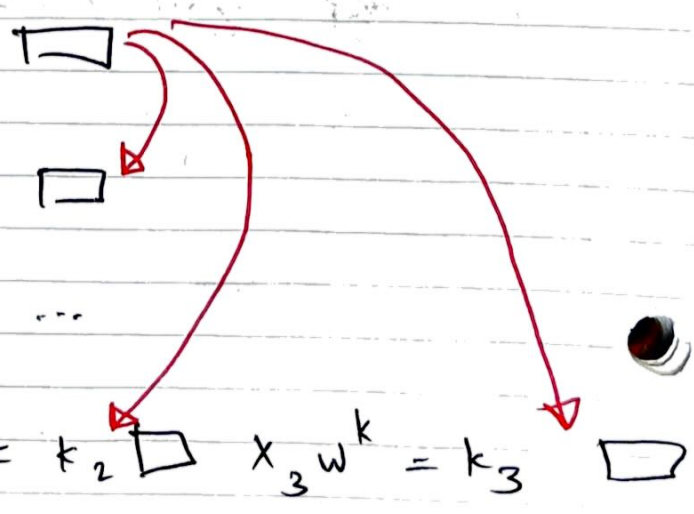
W^q → queries

- ~~Idea~~ Multiple x_1 with W^q and so on

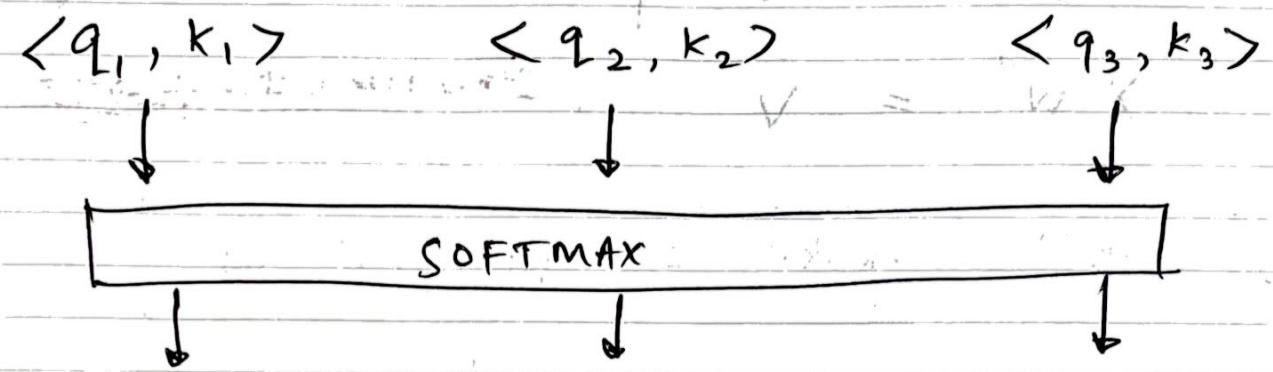
$x_1 W^q = q_1$

$x_1 W^k = k_1$

similarly for $x_2, x_3, \ldots$



- find similarity between ~~each~~ a "query embedding" and each "key embedding"
- and convert to a probability



first row of P (corresponding to q_1)

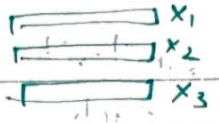
Practical : scale inner product with sqrt of key dimension d_k before taking softmax

~~DEFINE: Attention Softmax~~

$$\text{SOFTMAX}\left(\left[\frac{\langle q_1, k_1 \rangle}{\sqrt{d_k}}, \frac{\langle q_1, k_2 \rangle}{\sqrt{d_k}}, \frac{\langle q_1, k_3 \rangle}{\sqrt{d_k}}\right]\right)$$

and similarly for q_2, q_3.

Combined :



$$X W^Q = Q$$
$$X W^K = K$$
$$X W^V = V$$

$$P = \text{softmax}\left(\frac{Q K^T}{\sqrt{d_{key}}}\right)$$

[redacted]

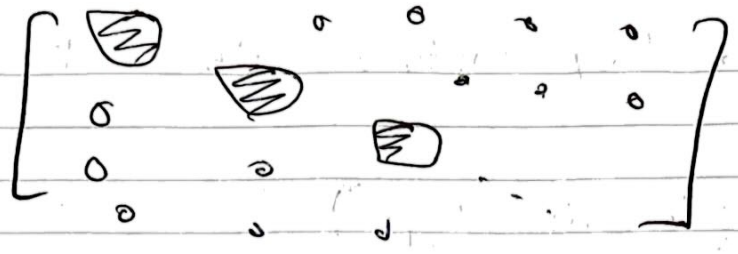
Learnable Parameters : W^Q, W^K, W^V

no dependence on # tokens

$$\therefore \quad Z = P \cdot V$$
$$= \text{softmax}\left(\frac{Q K^T}{\sqrt{d_{key}}}\right) V$$

$$Z \doteq \text{ATTENTION}(Q, K, V)$$

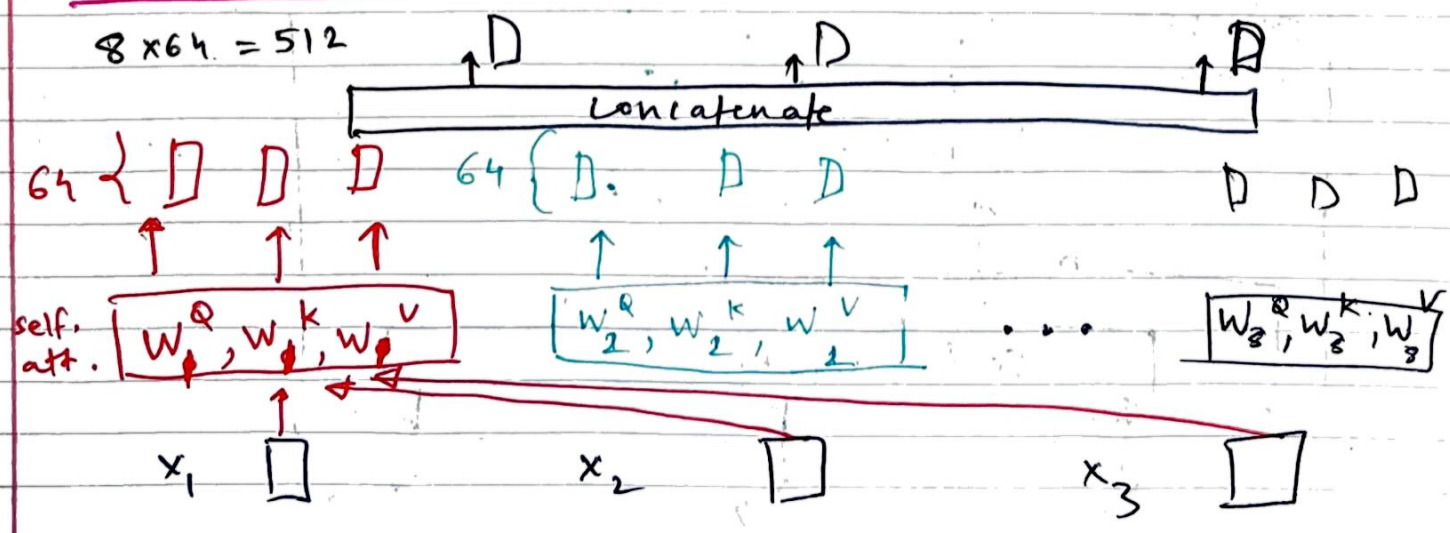
What if P is diagonal (or close to diagonal)?

$P =$ 

↳ This means there is no cross-talk

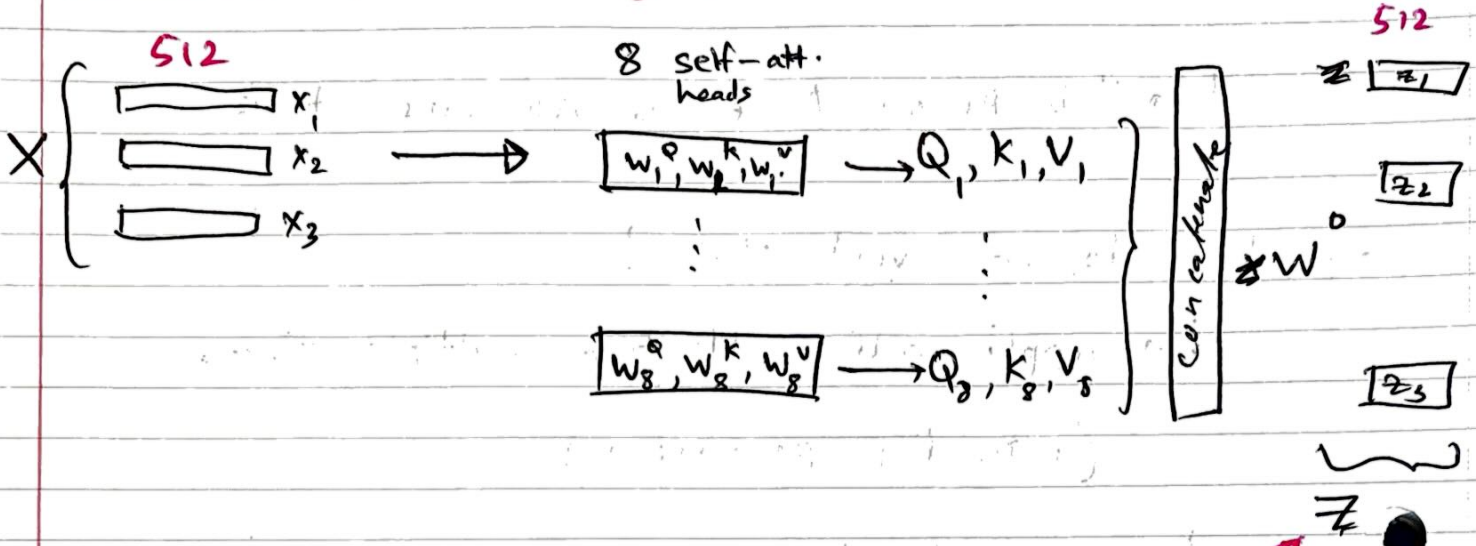
- how do we avoid this?
  - ↳ apply self-attention multiple times (with diff. parameters)
  - ↳ concatenate
- instead of output 512 for self-attention create 8 ~~64~~ self-attention "heads" of 64 dimension

MULTI-HEAD ATTENTION



in the paper they multiply with another matrix W^0

Summary (Overview Figure)



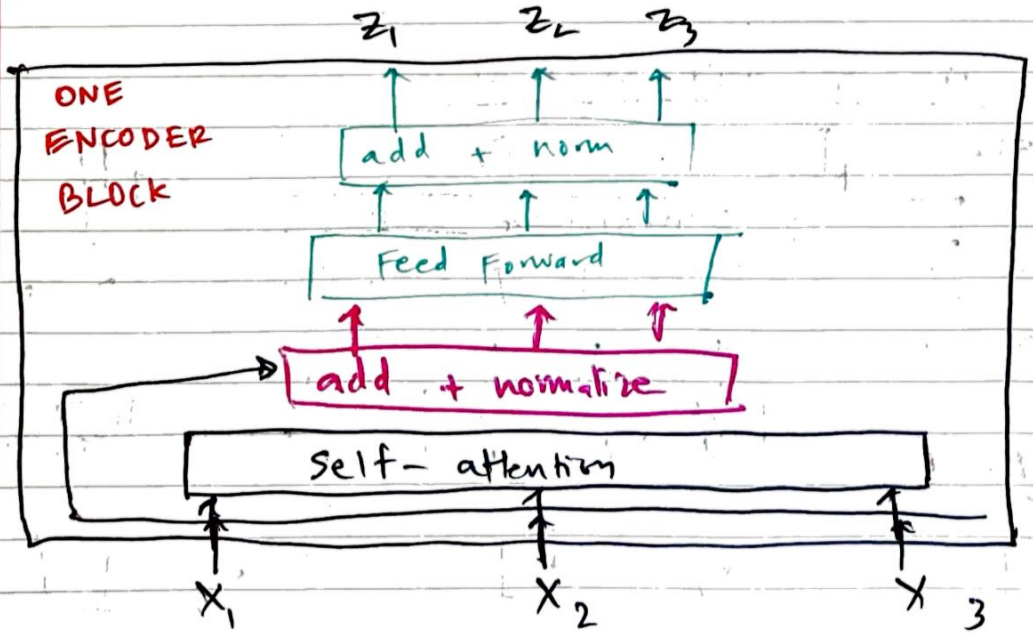
one final design element:

add $X + Z$

(like resnet skip connection)

apply layer normalization

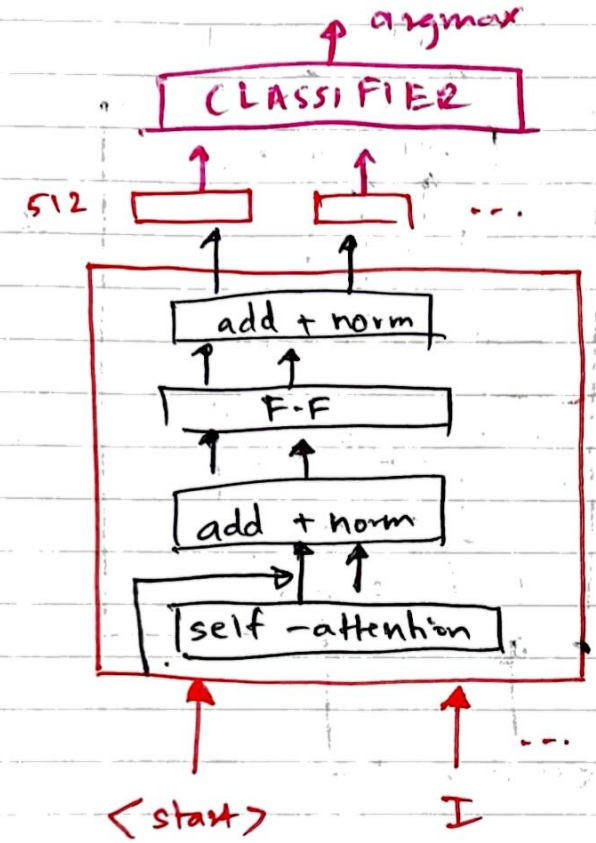
(make each row 0 mean, unit var)



multiple such encoder blocks stacked on top of each other

* NOW THE DECODER

- The decoder works in an autoregressive fashion
- ↳ previously generated tokens are the inputs for generating the next word



We're going to use the same architecture as the encoder

- the output will be two vectors (as many vectors as the input)
- put them through a classifier (over the entire vocab)

* ANY ISSUES ?

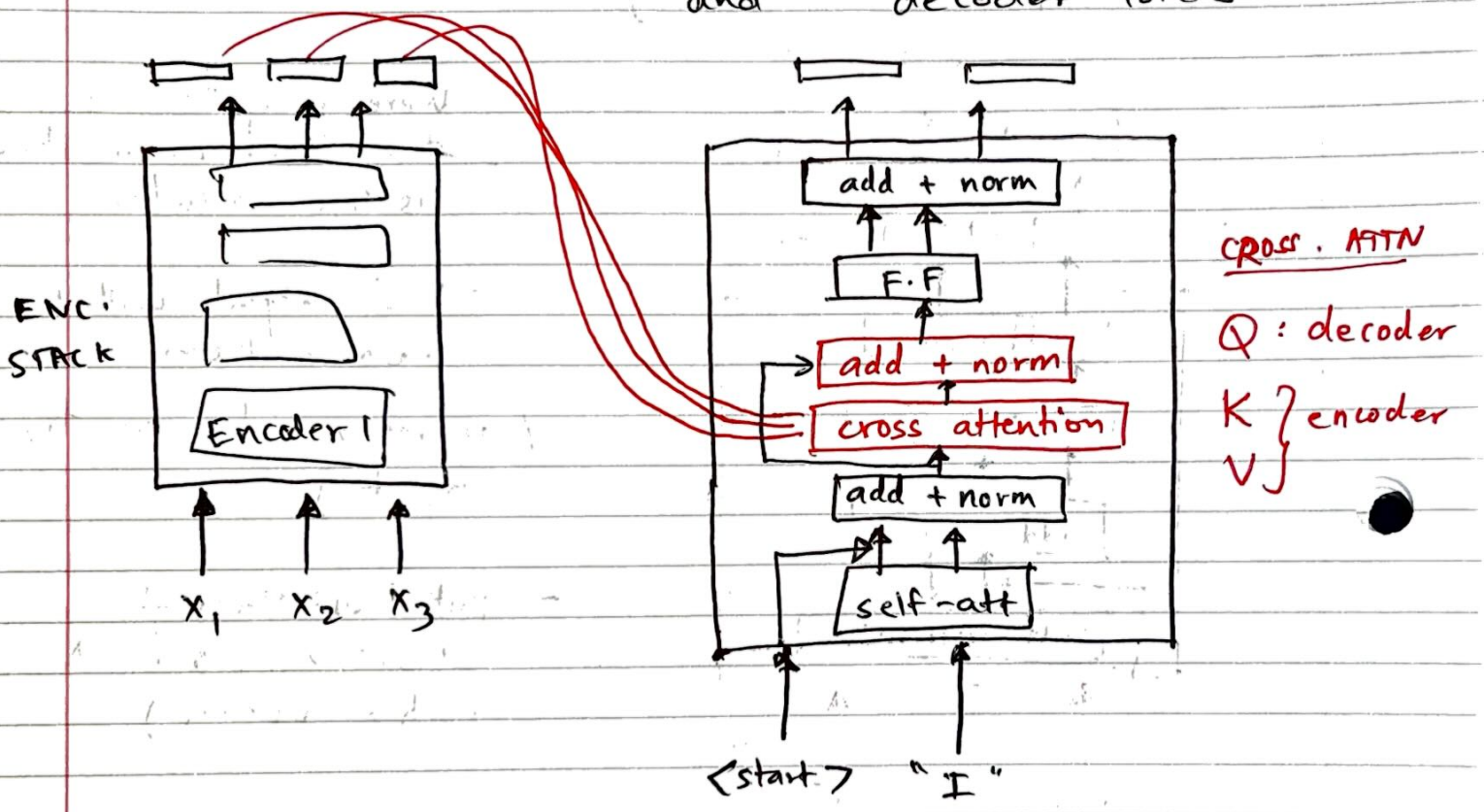
(1) in self-attention, maybe don't consider a very long context
↳ easy solution (zero out top ~~right~~ diagonal of * P matrix)

(2) what about the task ?
how are we ensuring that the output of "decoder" is correct ?

~~(3)~~ (3) We are not using the encoder outputs !

- We should get the encoder outputs to talk to my decoder.

"CROSS-ATTENTION" between encoder tokens and decoder tokens



* LINEAR ATTENTION

If you have N input, what is the size of the attention matrix ? $N \times N$

↳ for large N, this is really expensive.

$$O(N^2)$$

Q: How can we make this computation cheaper. Linear in N ? $O(N)$

Katharopoulos ICML 2020

NOTATION

$X \in \mathbb{R}^{N \times d}$ (N tokens, each token $d \times 1$)
eg. 512

$T : \mathbb{R}^{N \times d} \longrightarrow \mathbb{R}^{N \times d}$

the transformer is a mapping from N tokens to N token

– defined by L layers $T_1, \cdots, T_L$

$$T_\ell(x) = f_\ell\left(A_\ell(x) + x\right)$$

f_ℓ ↑ F.F and layer norm

$A_\ell(x)$ ↑ self-attn

$+ x$ ↑ residual

and formula for Attention :

$Q = XW^Q \rightarrow \in \mathbb{R}^{d \times D}$

$K = XW^K$

$V = XW^V$

$\in \mathbb{R}^{N \times d}$

$\in \mathbb{R}^{N \times D}$

$$P = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)$$

$\in \mathbb{R}^{N \times N}$

$$A(x)_\ell = P \cdot V \equiv V' \quad (1)$$

we should improve the formula for P

(use some other similarity measure instead of inner product)

— let's look at the i^{th} row of V' (embedding for i^{th} token)

$$V'_i \equiv \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)\, V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)} \quad (2)$$

Q_i: query for i^{th} token

(1) ≡ (2) if $\text{sim}(q,k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$

but we don't have to use this formula

↳ we can use some other formula

↳ what other similarity / distance measure can we use?

"KERNELS"

instead of inner-product bet^n vectors, use inner-product bet^n some transformation of vectors.

$$\text{Kernel}(\vec{q}, \vec{k}) = \phi(q)^T \phi(k)$$

↳ transformation.

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(k_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(k_j)}$$

↳ we have decomposed similarity as a product

$$= \frac{\phi(Q_i)^T \left(\sum_{j=1}^{N} \phi(k_j) V_j\right)}{\phi(Q_i)^T \left(\sum_{j=1}^{N} \phi(k_j)\right)}$$

do it once and store (doesn't depend on i)

Same for every token

$O(N)$

Computational Complexity?

(for computing attention matrix)

goes does from $O(N^2)$ to $O(N)$

but what is the dimensionality of $\phi()$?

features $\phi()$ usually have bigger dimensions than the vector

(eg. polynomial kernel of degree 2 increases dimension from r to r^2)

so we should use suitable kernels

(if we want to reduce complexity)