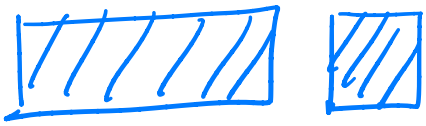
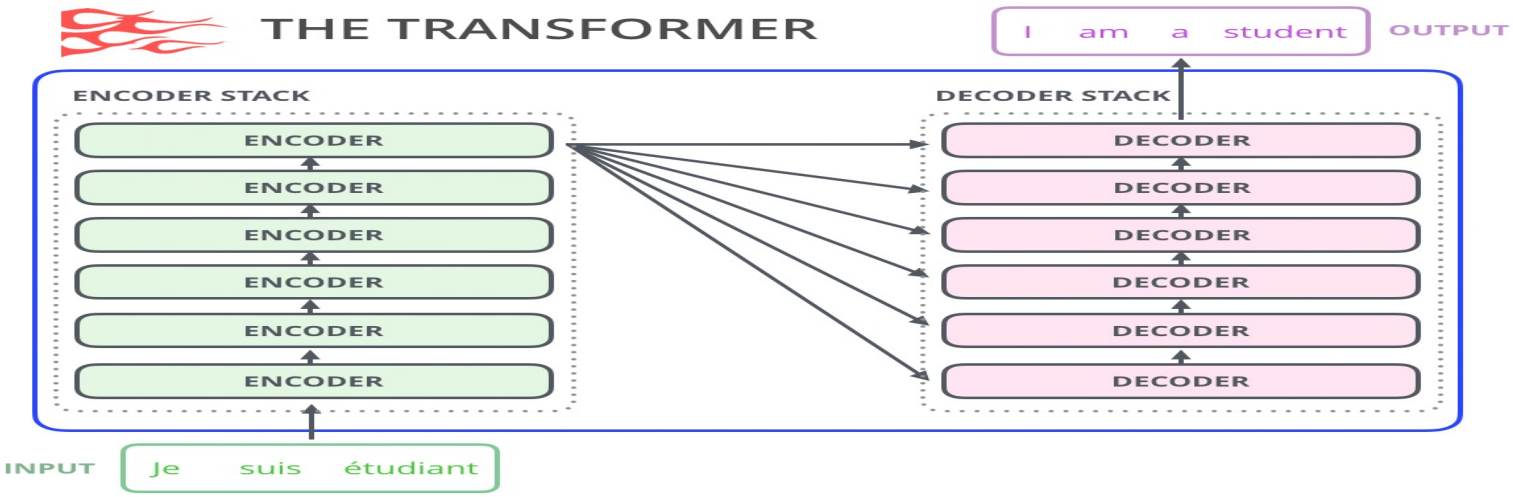

15/May/21

(*) For language modeling data need to
a) tokenized ⇒ into word
b) Converted into word embedding
c) Positional embedding
d) Segment/Sentence embedding.

(*) GPT-2 ⇒ Language model that Can be used for next word Prediction.

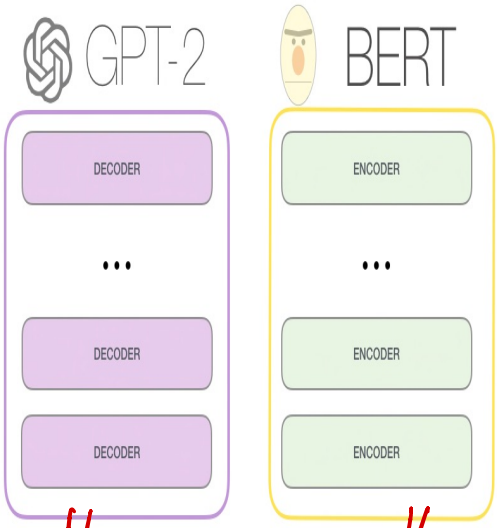


given the sequence so far | Prediction



Transformer have an encoder & decoder architecture to tackle machine translation.

* Subsequent models utilize either encoder/decoder or their variants.



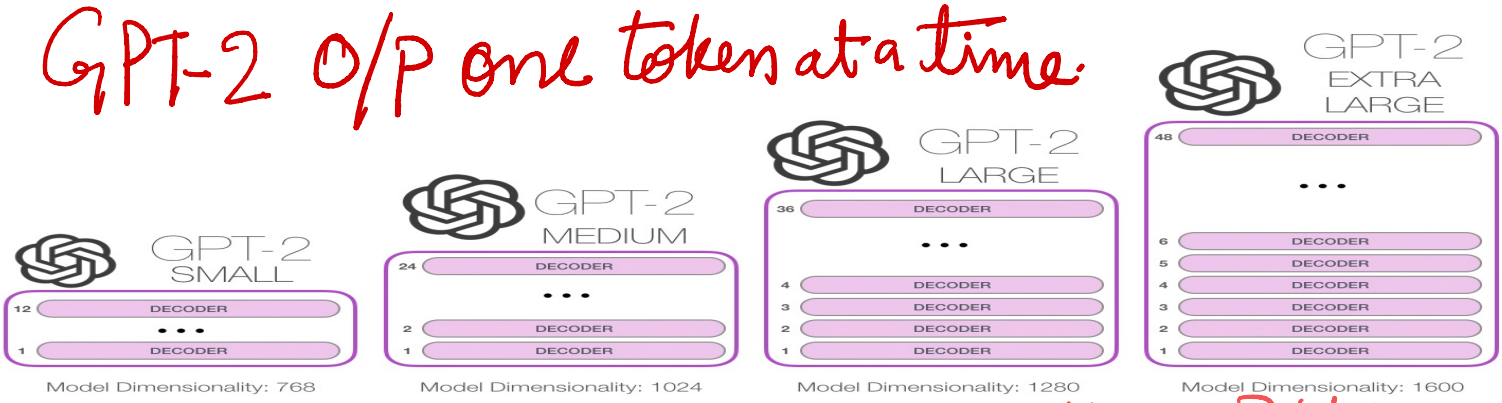
uses Decoder

uses Encoder

* Bert can be used for better features for any downstream task.

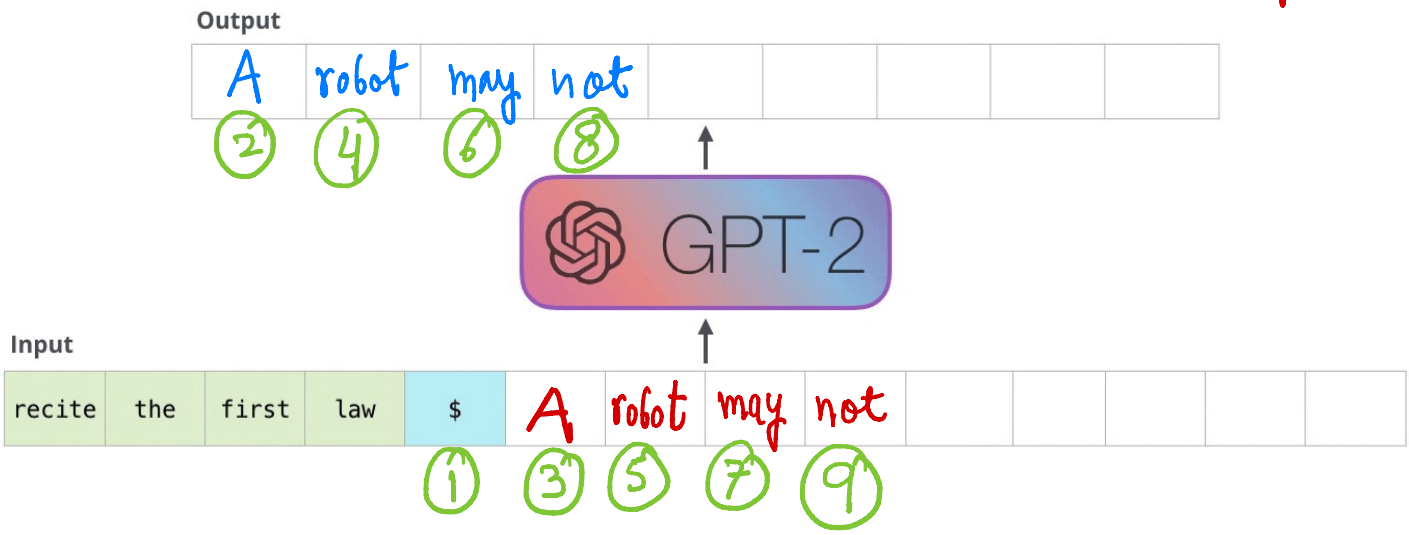
* GPT-2 can be used for language generation.

GPT-2 O/P one token at a time.

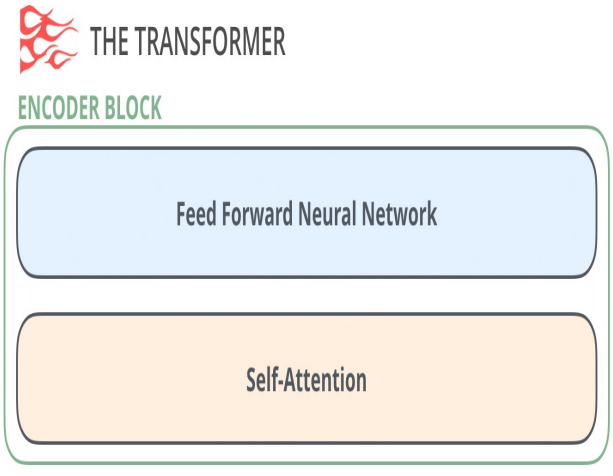


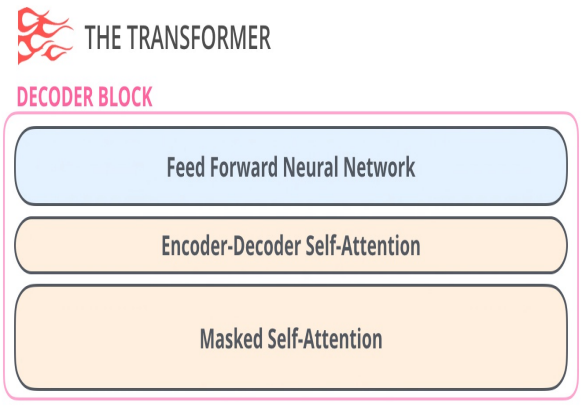
Bert looks/produce all at once ⇒ Hence Bidirectional.

Token after production got added to i/p: Auto-regression



(*) Transformers En/Decoder

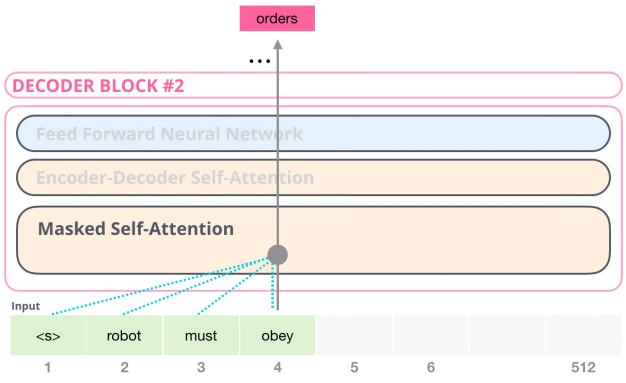


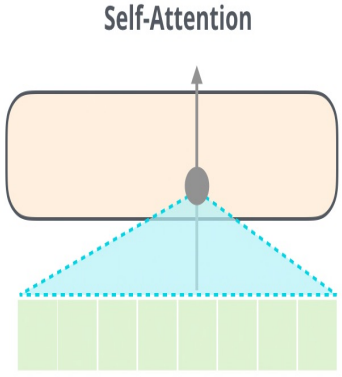


512 (size) for original transformer

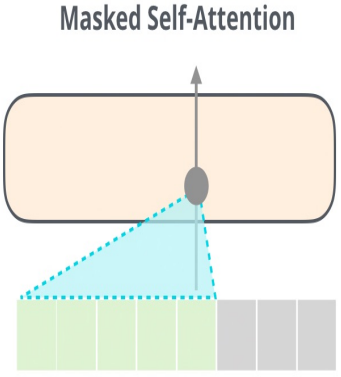
Decoder: ⑥ ⊗ En/De SA layer allows to learn attention to specific segments from the encoder.

(*) Also decoder's (SA) layer is implemented as Masked SA.

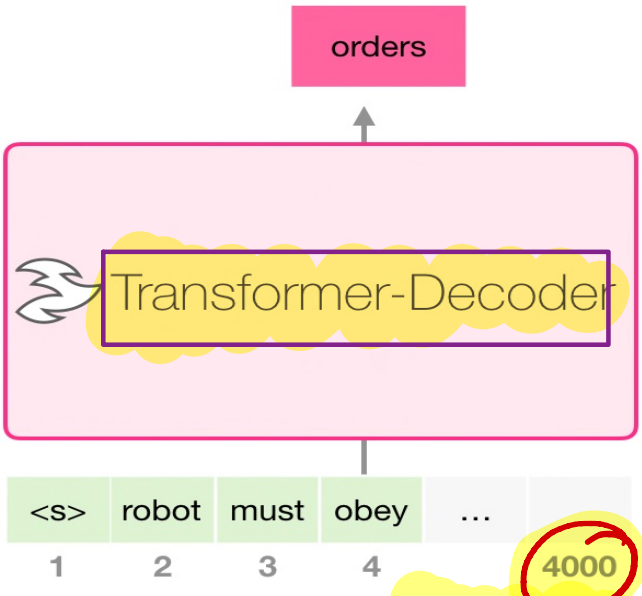




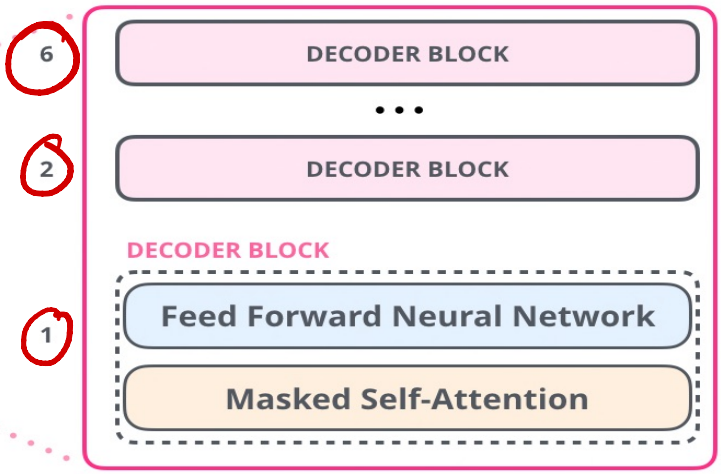
BERT (Encoder)



GPT-2 (Decoder)



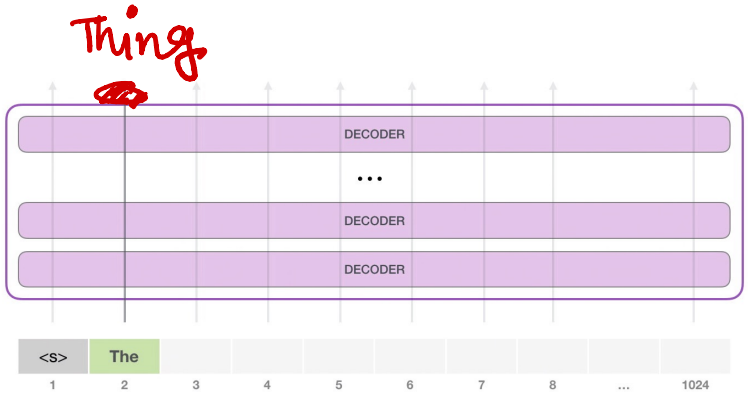
512 → 4000



(Also 2nd CrossAttention is dropped)

Subsequently Transformer Decoders are used in GPT-2 to create a language model that predicts character by character (word by word)

⊛ GPT-2 (Working) {Trained model}



⊛ Give it <s> and it will start spelling next word one by one.

⊛ final o/p vector is scored against model vocabulary (50k)

⊛ Instead of choosing (Max-Prob) we've a (top-k) & sample from it.

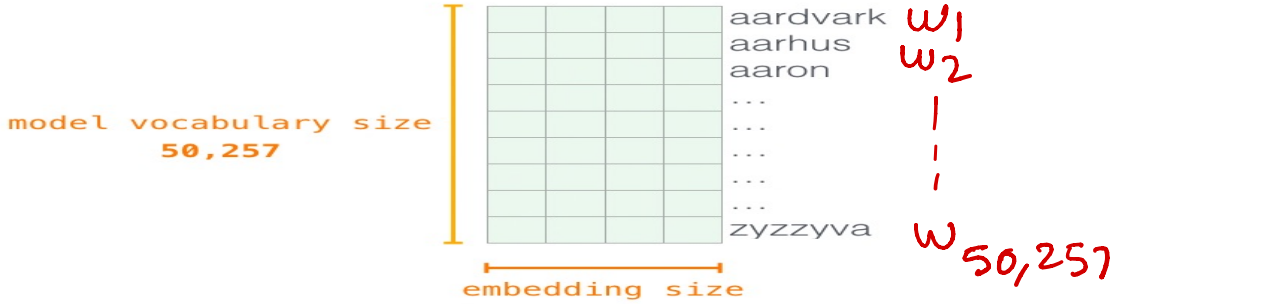
⇓⇓

This will help to generate new data (variety)

Each layer of GPT-2 has retained its own interpretation of the first token and will use it in processing the second token (we'll get into more detail about this in the following section about self-attention). GPT-2 does not re-interpret the first token in light of the second token

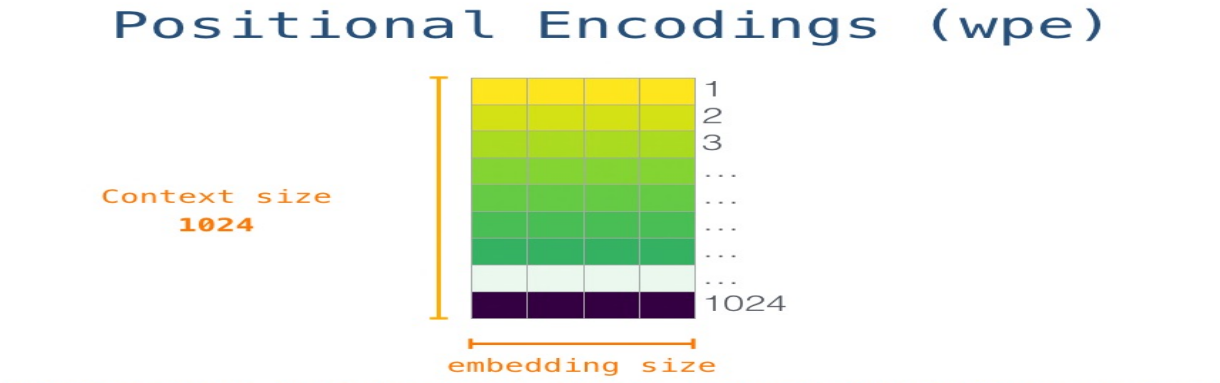
(I) Embeddings/Encoding

(*) Input Encoding



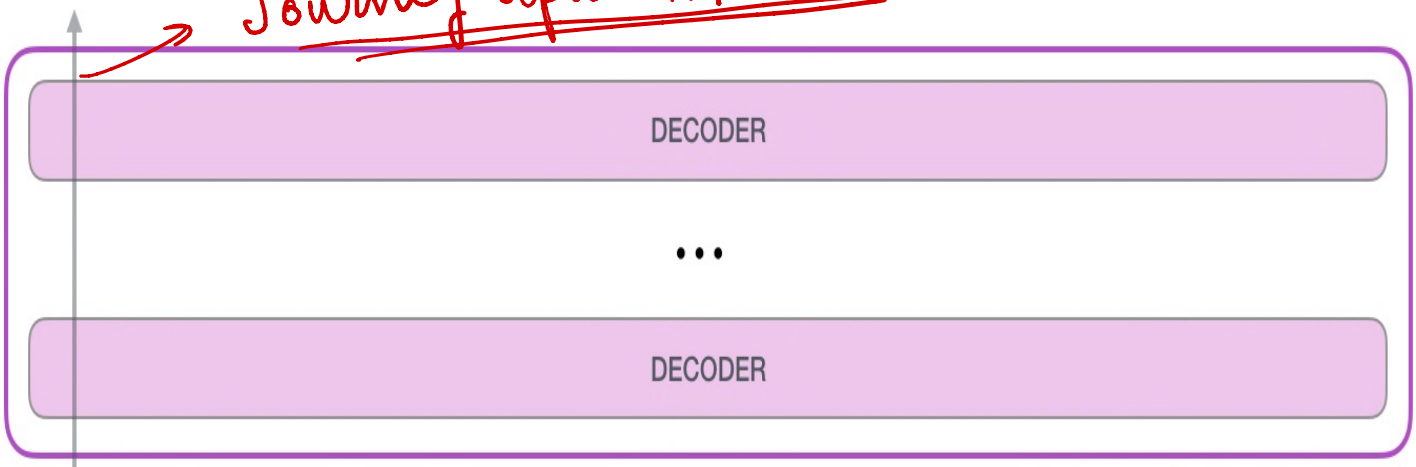
Embedding size for different models.

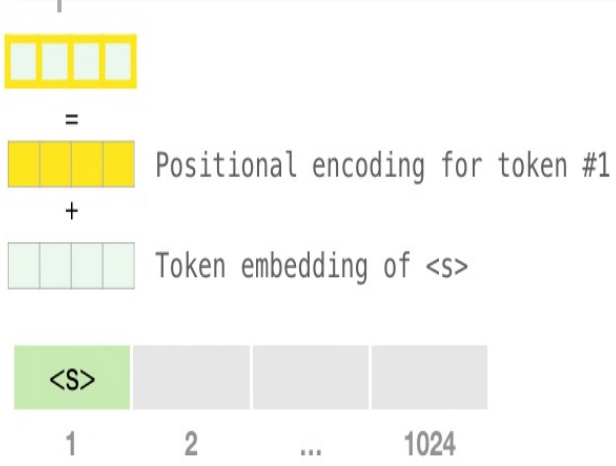
(*) Positional Encoding

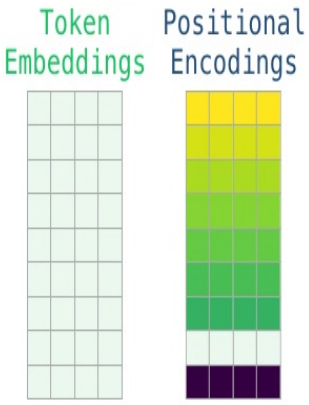


Both are added to get final i/p encoding.

Journey up the stack:





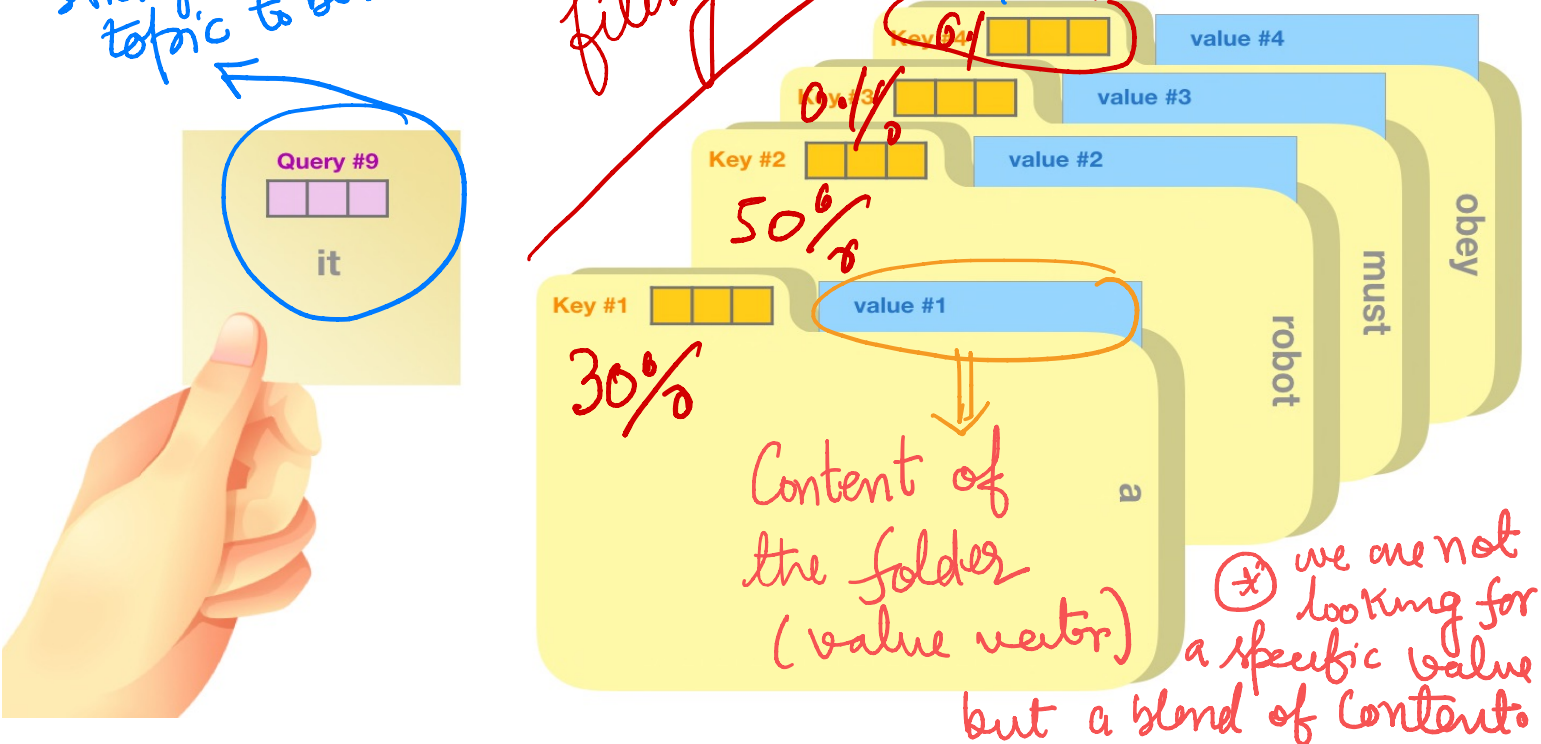


II Self attention process

Sticky note with topic to be searched

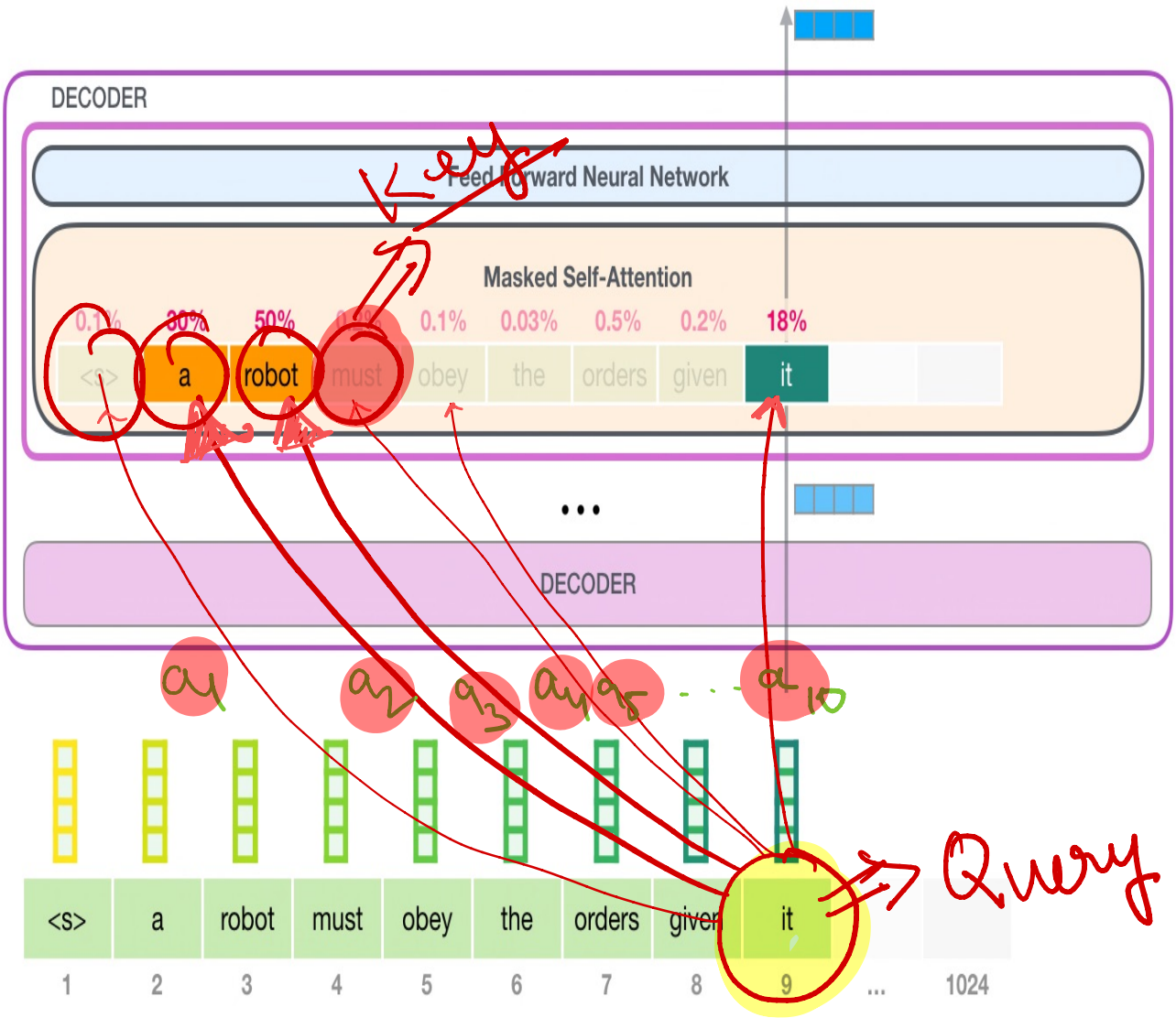
filing Cabinet

labels of folders inside the Cabinet



Content of the folder (value vector)

* we are not looking for a specific value but a blend of contents



for some (i th) word (it), for all (j th) word

$$a_{ij} = \text{Query}_i * \text{key}_j$$

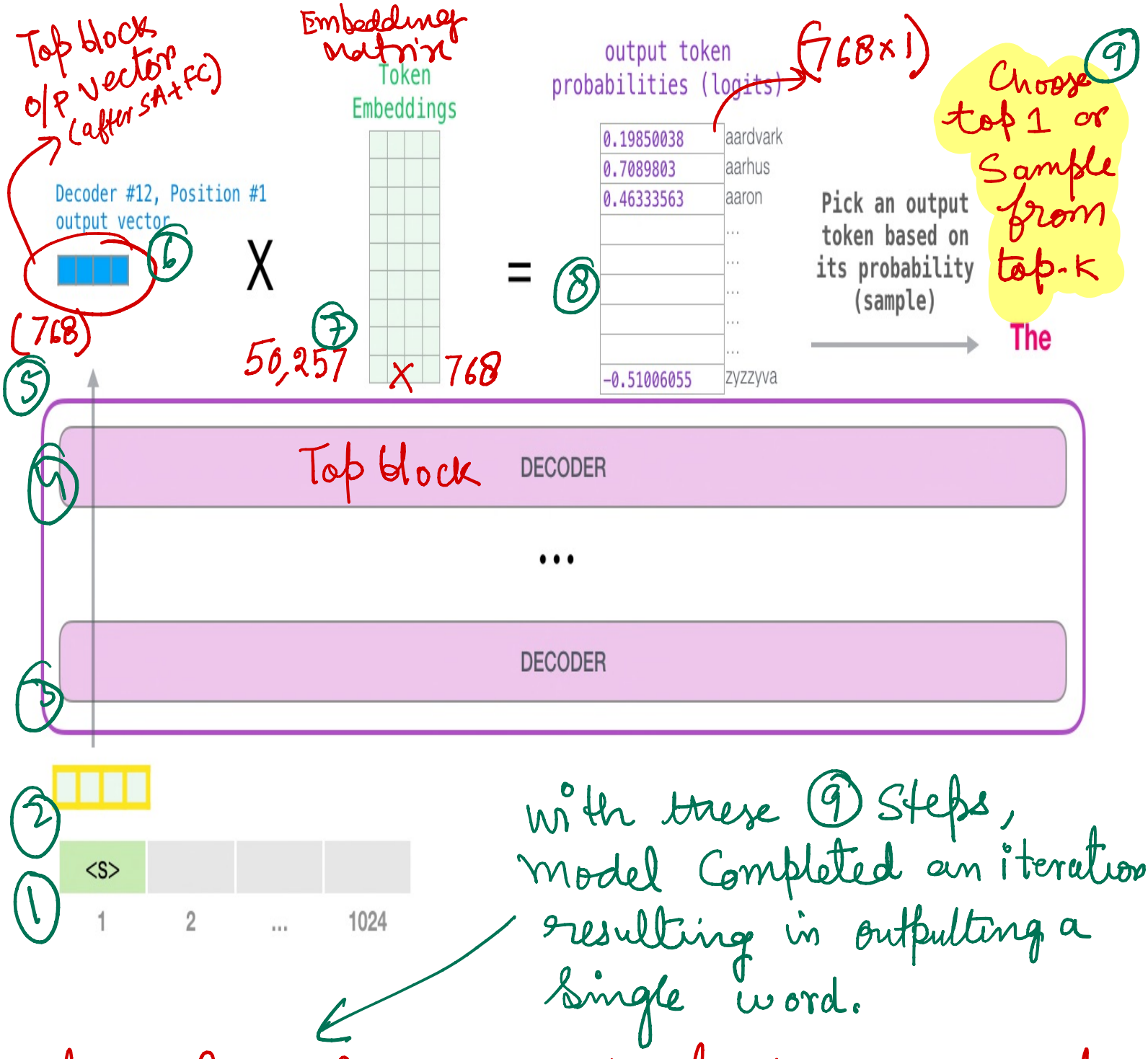
$$\text{O/P } i = a_{ij} * \text{Value}_j$$

Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

(They got most of the weight)

O/P ith Word

Why it is not very high.

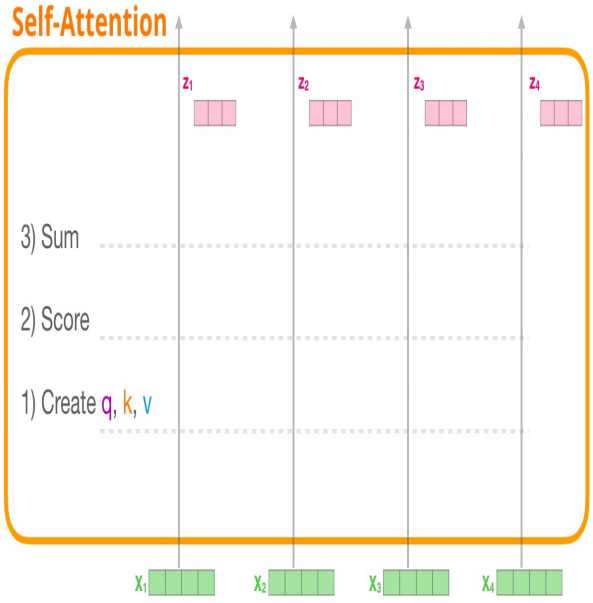


With these (9) steps, model completed an iteration resulting in outputting a single word.

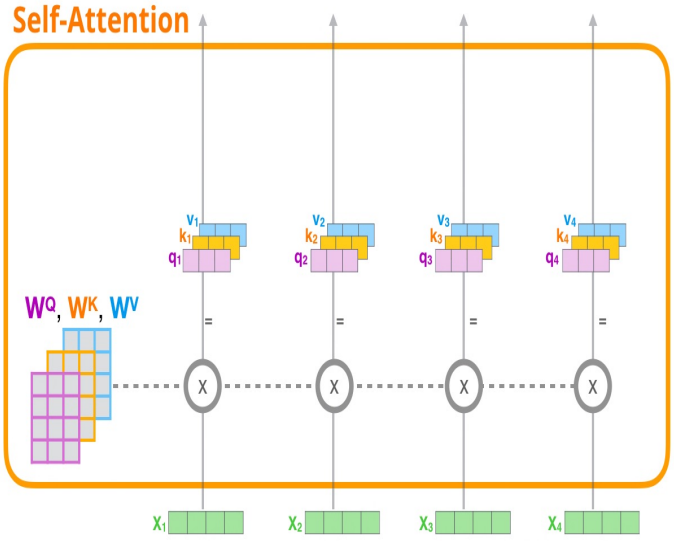
Model continues to iterate till it generate (1024) entire context or <EOS>.

⊛ GPT-2 uses Byte pair encoding (BPE) ⟹ to create tokens in its vocabulary.

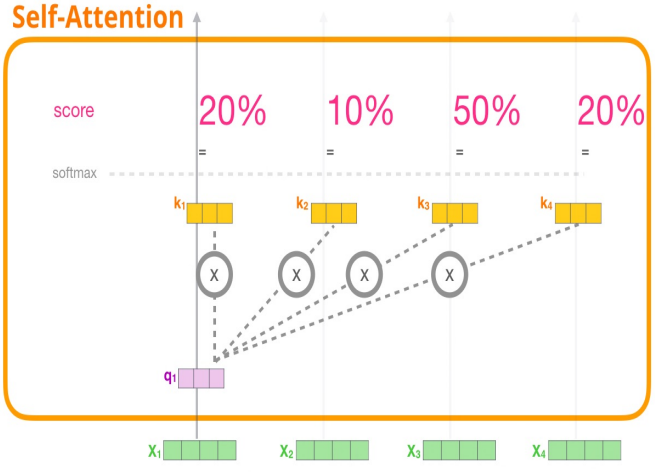
Visual Self Attention (Recap)



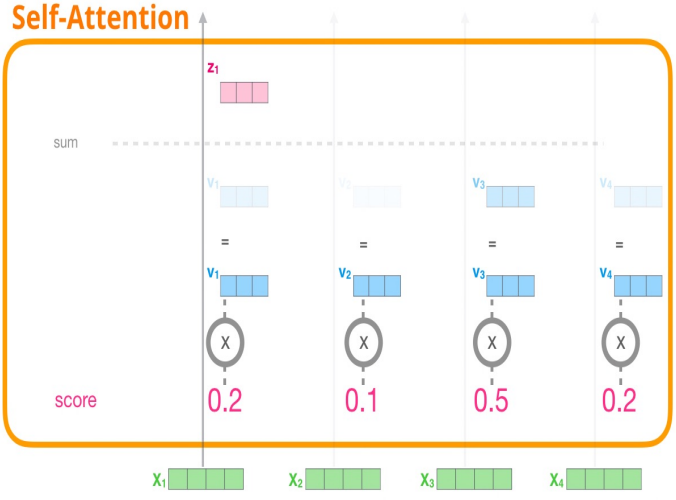
1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V



2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match

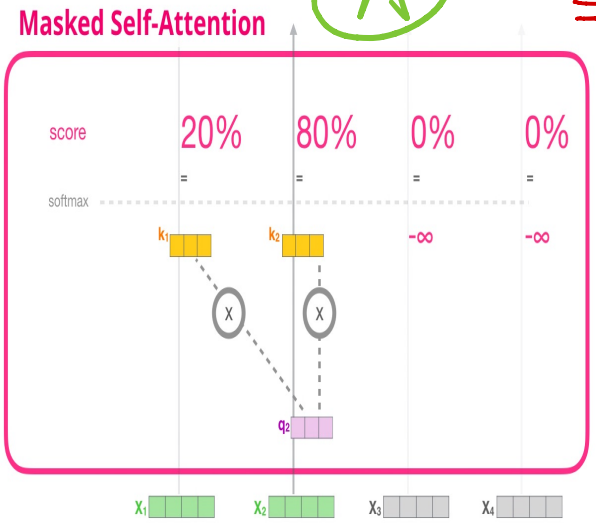


3) Multiply the value vectors by the scores, then sum up



Masked Self Attention

A



⇒ robot must obey orders.

B

Features

Labels

	position: 1	2	3	4	
Example: 1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

Model has (2) tokens and we observe 2nd one.

(*) This sequence will be absorbed in (4) steps. BatchSize = 4 (for this toy example)

robot must obey orders.

k_1 k_2 k_3 k_4

q_1 q_2 q_3 q_4

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Not Matrix multiplication

C

$q_1 k_1$	$q_1 k_2$	$q_1 k_3$	$q_1 k_4$
$q_2 k_1$	$q_2 k_2$	$q_2 k_3$	$q_2 k_4$
$q_3 k_1$	$q_3 k_2$	$q_3 k_3$	$q_3 k_4$
$q_4 k_1$	$q_4 k_2$	$q_4 k_3$	$q_4 k_4$

Scores
(before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax
(along rows)
(why?)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

When model process the 2nd word (input is "robot must") ["Must"]

48% attention is on "robot" & 52% attention is on "must"

While processing first word ("robot") full attention was on that word itself