April 19, Transformers

Tokens — Patches of sound / text / picture
↓
Vectors → Pass through Attention. ①
- talk to other vectors.
- update their values

e.g. model < Language Model / Fusion Model
update it's meaning (vector)

② feedforward
- don't talk to each other (same operation in parrellel)
- kinda like asking question, each vector. updating them. based on answer to those question

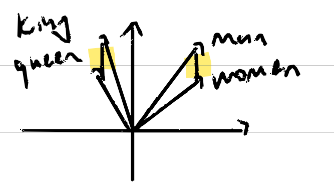
③ repeats. → until end, essential meanings had been baked of the prev token into the "last token" of the sequence

Sound natural? | apple | banana... | and | bread | are | ??? |

↓ we kept doing, new token as seed to repeat calculation

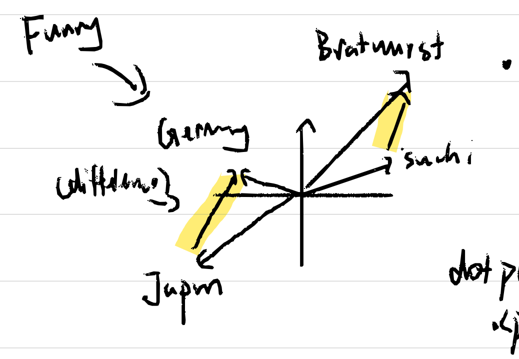
System prompt
[You're a apple.
Prediction – User: Give me a bum
Apple: ???

Tokenizing (vector)
- train tends to settle vector in a space, where direction have a sementic meaning



E(queen) - E(king) = E(women) - E(man)
E(queen) ≈ E(king) + E(women) - E(man)

- e.g. demostrate, demostrate → gender
- E(Hitler) + E(Germany) = E(Mussolini) + E(Italy)



dot product:
- (plur) w/ dogs } dot prod
- (plur) w/ cats }

∨ (larger)

- (plur) w/ dogs } dot prod.
- (plur) w/ cats }

Fun: You can find how plural a word is

Cats - cat ≈ plur?

* embeddings[token] = vector

N words/tokens

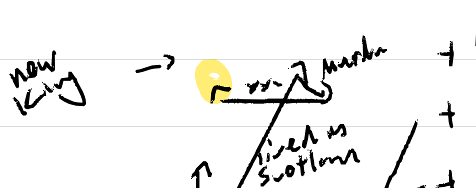
$$\begin{array}{c|cccc} & \text{apple} & \text{banana} & \text{cats} & \text{dogs} \\ \text{dimension} & +8.2 & -4.2 & -2.8 & -8.6 \\ & +7.2 & +6.4 & -1.7 & -4.2 \\ & -8.1 & +3.8 & -4.9 & +1.8 \\ & -8.7 & -1.7 & +1.2 & 8.8 \end{array} = W_E$$

(Embedding matrix)

(only for Embedding)
Parameter = dimention · N words

* vector are updated based on context

* In transformer, these aren't just representing
+ "meaning" of these word
+ positions of that word
+ context

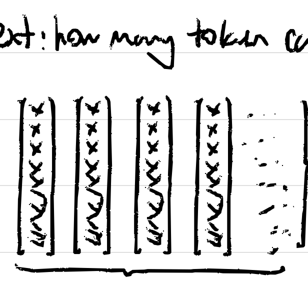


can be forever
⊥

- the direction changes by other "blocks" of the sequence, resulting pointing into a more specific, nuanced direction

* think how you shaped a word. in a sentence.

Context: how many token can it process (how much can it incorporate, when perdict of next word)



Context size: 2048

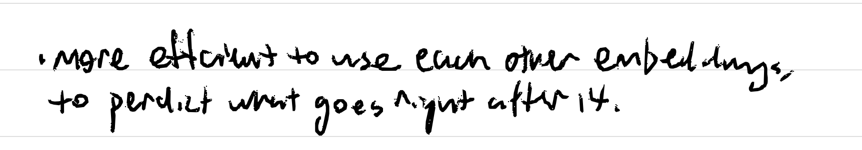
What happends in very end? : Unembedding

Harry potter, least favorite, Professor [ ]
HOW

①: for the last vector
- use a matrix, match the last vector, to a list, of 50,000 vocabulary
  → normalize it with softmax

why only use last embedding to predict?
Q: why other vectors are just sitting there, with it's own context rich meaning?

- more efficient to use each other embeddings, to perdict what goes right after it.
- last token is the summarization of previous tokens.



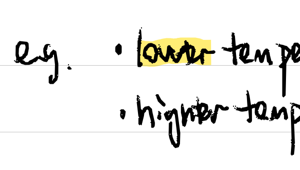
Unembedding Matrix
W_u

(vocab)
W_u = word · dimension
(swapped of embedding)

* unembedding is a classifier.
use the query to find dot product, which is most similar.
- the output is a semantic (mean, you need to find the best fit for the meaning
- Vector is meaning, W_u is a mapper to possibilities, to tokens (words)

Softmax

- Normalizing it, add up to one
- Temperature: throwing a T in the softmax process
  → it's like temperature, because it resembles temperature in therodynamic equation;
  - T is larger, more weight to lower value
    → distribution is more uniformed



Smaller T   Larger T

e.g.
- lower temperature: always most predictable word
- higher temperature: chance to choose less likely words

trite derivative of Goldilocks

Logits —	softmax →	Probabilities
+6.0	$e^{x_i/t} / \sum_{n=0}^{N-1} e^{x_n/T}$	0.19
-5.0	$e^{x_i/t} / \sum_{n=0}^{N-1} e^{x_n/T}$	0.04
+4.0		0.14
+1.5	$e^{x_i/t} / \sum_{n=0}^{N-1} e^{x_n/T}$	0.10
+9.9	$e^{x_i/t} / \sum_{n=0}^{N-1} e^{x_n/T}$	0.35

- logits : raw, unnormalized output

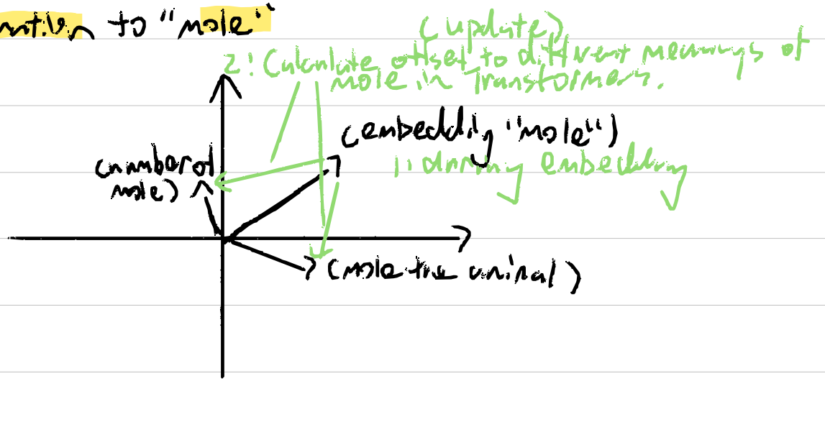
April 29th. Attention in Transformers, DL6

vectors: semantics in space
· Transformers: through adjusting the embeddings.
  · don't merely encode each individual words (vector)
  + but bake them in some much richer contextual meanings

e.g.
· American shrew mole
· One mole of carbon dioxide
· biopsy of a mole

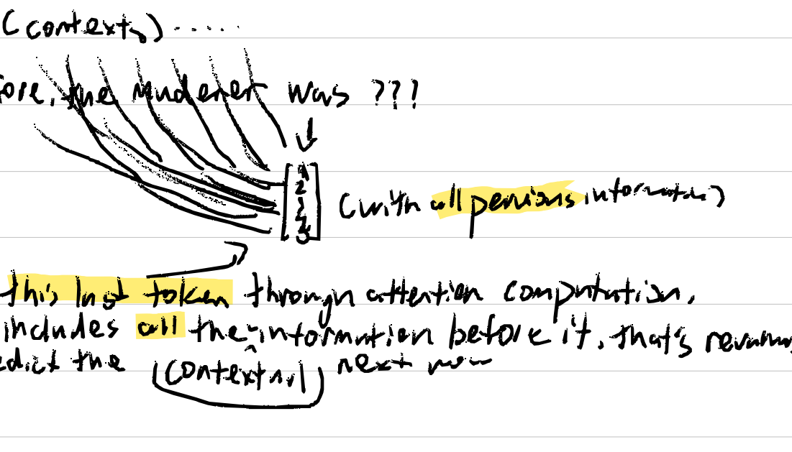
1. After first step of transformers, the "mole" would mean the same:
   · Because first step of token-embeddings is just a lookup table

2. It's the next step of Transformers, the surrounding embedding have the chances to pass the contextual information to "mole"



Other than just refining a meaning of a word, the attention block allows to move information of one embedding, to another
· information far away
· information richer than just a word

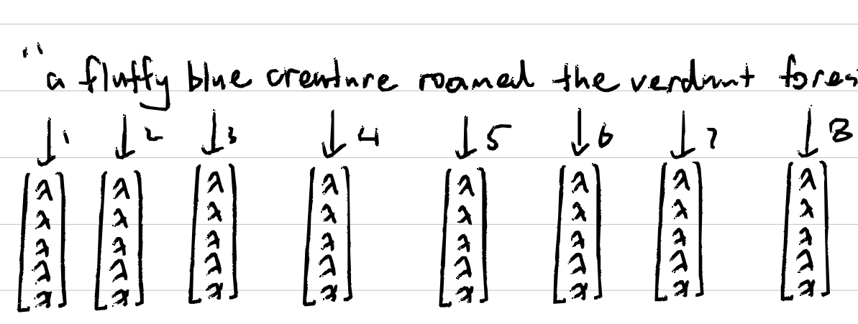
· After all the vectors flow through the networks, different attention block:
· the computations:
→ predict the last token
↳ a function of the last vector in sequence.
existing
······ (contexts) ·····
Therefore, the murderer was ???



this last token through attention computation, will includes all the information before it, that's relevant to predict the (contextual) next word

"attention head" example

"a fluffy blue creature roamed the verdant forest"



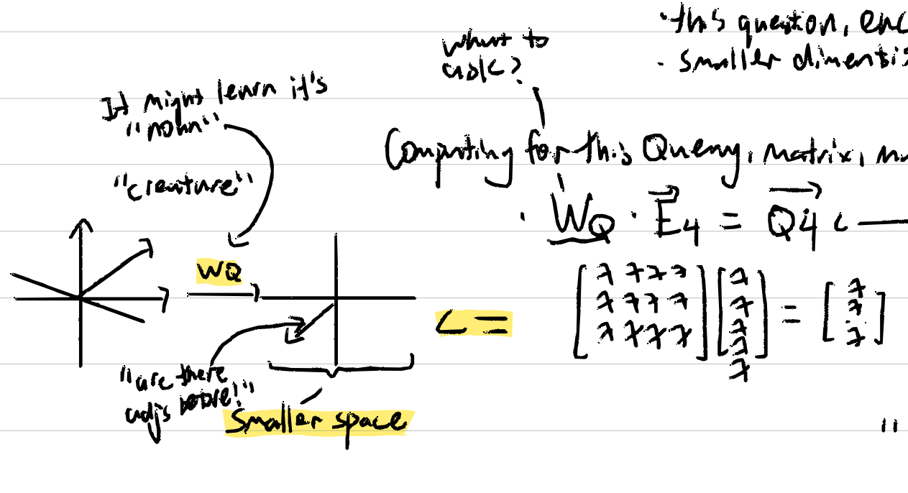
(also, position of the word is also updating the vector)

$\vec{E}_1, \vec{E}_2, \vec{E}_3, \vec{E}_4$ ← should looks like matrix-vector produce w/ tunable weights
$\Delta\vec{E}_1 \Delta\vec{E}_2 \Delta\vec{E}_3 \Delta\vec{E}_4$
updated "creature"

Questions

1: $\vec{E}_4$ (creature) ask, e.g. [are there any adj before me?]
Query
($\vec{E}_2, \vec{E}_3$) → I am!
· this question, encoded in vector
· smaller dimension

what to ask?
Computing for this Query, matrix multiply to embedding
· $\vec{W_Q} \cdot \vec{E}_4 = \vec{Q}_4$ ← Question query



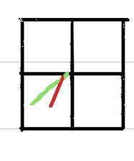
$E_4 \xrightarrow{W_Q} Q_4$
"Any adjectives before me?"

"a fluffy blue creature roamed the verdant forest"

Questions?	$\vec{E}_1 \xrightarrow{W_Q} \vec{Q}_1$	$\vec{E}_2 \xrightarrow{W_Q} \vec{Q}_2$	$\vec{E}_3 \xrightarrow{W_Q} \vec{Q}_3$	$\vec{E}_4 \xrightarrow{W_Q} \vec{Q}_4$	$\vec{E}_5 \xrightarrow{W_Q} \vec{Q}_5$	$\vec{E}_6 \xrightarrow{W_Q} \vec{Q}_6$	$\vec{E}_7 \xrightarrow{W_Q} \vec{Q}_7$	$\vec{E}_8 \xrightarrow{W_Q} \vec{Q}_8$
Potential Answers?								
"a" $\to \vec{E} \xrightarrow{W_k} \vec{K}_1$	K_1Q_1	K_1Q_2	K_1Q_3 ···					
"fluffy" $\to \vec{E} \xrightarrow{W_k} \vec{K}_2$	K_2Q_1	K_2Q_2	K_2Q_3 ···					
"blue" $\to \vec{E} \xrightarrow{W_k} \vec{K}_3$	K_3Q_1	K_3Q_2	K_3Q_3 ···					
"creature" $\to \vec{E} \xrightarrow{W_k} \vec{K}_4$	K_4Q_1	K_4Q_2	K_4Q_3 ···					
"roamed" $\to \vec{E} \xrightarrow{W_k} \vec{K}_5$	K_5Q_1	K_5Q_2	K_5Q_3 ···					

(radius as dot product size)

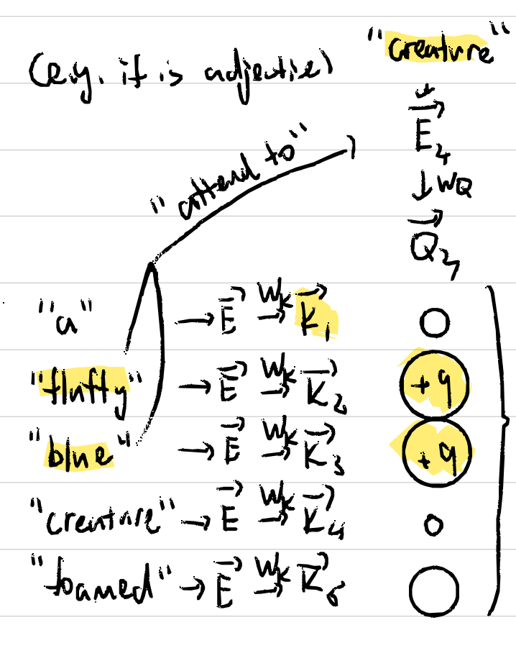
· when the key matches the query: alignes together.



🟥 = Q_1
🟩 = K_2

- e.g. Key (fluffy / blue)
matches
Query (are you adj?)

(e.g. if is adjective) "creature" $\vec{E}_4 \xrightarrow{W_Q} \vec{Q}_4$
"fluffy and blue, is attend to creature"



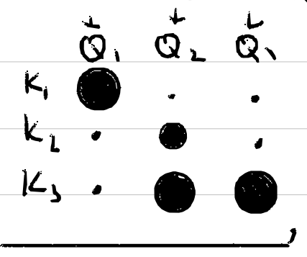
the score for K·Q
* How relevant each word, is to update the meaning of each other words *
(How much should a word change on this word)

(score)
· How this relevant is going to be used,
· Calculate the weighted sum along each column,
· weighted by the relevance

(weighted sum) "creature" $\vec{E}_4$
$\vec{E} \to V_1$ $0.x \vec{V}_1$
$\vec{E} \to V_1$ $0.x \vec{V}_2$
$\vec{E} \to V_1$ $0.x \vec{V}_3$
$\vec{E} \to V_1$ $0.x \vec{V}_4$
"the $\vec{K} \cdot V$"

· Between 0~1, softmax to normalize.

	Q_1	Q_2	Q_3
K_1	1.00	0.07	0.00
K_2	0.00	0.42	0.01
K_3	0.00	0.58	0.99



"attention pattern"

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{K^TQ}{\sqrt{d_k}}\right)V$$

(the grid of dot product)
$K^1 \cdot Q^1$ $K^1 \cdot Q^2$ $K^1 \cdot Q^3$
$K^2 \cdot Q^1$ $K^2 \cdot Q^2$ $K^2 \cdot Q^3$

· array of Qs, Ks
· multiply embedding w/ W_K/W_Q

this is only to help numerical stability

applied column-by-column

Masking

blue creature roamed ???
blue creature roamed the ???
blue creature roamed the verdant ???
a ~~fluff~~ creature roamed the verdant forest ???

· Predict every possible next token. (subsequence)
· to enhance training
· single train ⇒ many training

... during the training process ...
· Because we also train on subsequences (word we already know)
· Never allow later words, to influence earlier words.
· so they can cheat on what's next.

"a fluffy blue creature

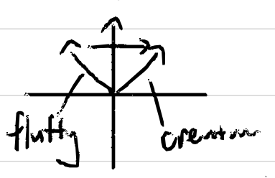
Questions?	$\vec{E}_1 \xrightarrow{W_Q} \vec{Q}_1$	$\vec{E}_2 \xrightarrow{W_Q} \vec{Q}_2$	$\vec{E}_3 \xrightarrow{W_Q} \vec{Q}_3$	$\vec{E}_4 \xrightarrow{W_Q} \vec{Q}_4$
Potential Answers?				
"a" $\to \vec{E} \xrightarrow{W_k} \vec{K}_1$	K_1Q_1	K_1Q_2	K_1Q_3 ···	
"fluffy" $\to \vec{E} \xrightarrow{W_k} \vec{K}_2$	$-\infty$	K_2Q_2	K_2Q_3 ···	
"blue" $\to \vec{E} \xrightarrow{W_k} \vec{K}_3$	$-\infty$	$-\infty$	K_3Q_3 ···	
"creature" $\to \vec{E} \xrightarrow{W_k} \vec{K}_4$	$-\infty$	$-\infty$	$-\infty$	
"roamed" $\to \vec{E} \xrightarrow{W_k} \vec{K}_5$	$-\infty$	$-\infty$	$-\infty$	

· you can't set to zero, or softmax won't add up
· since sometime lower than 0

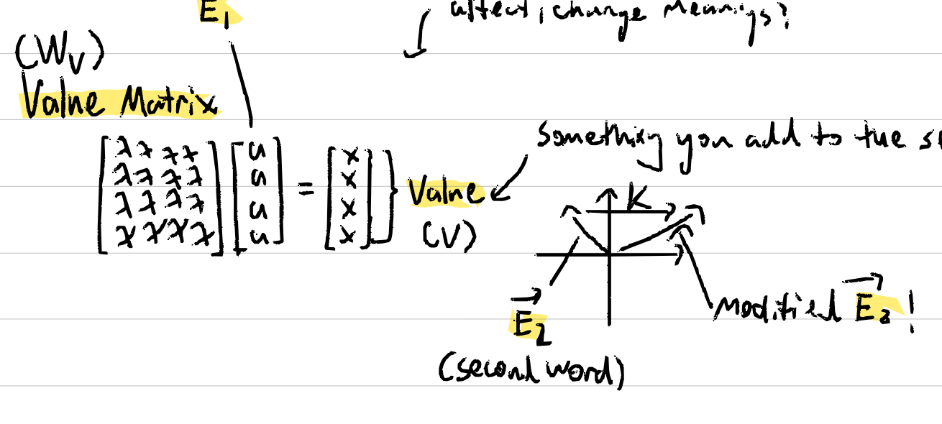
Fun fact: this size (attention pattern) = $(\text{Context-size})^2$

---

this tells us which words are relevant to which other words
· now we need to actually update the embeddings,
· pass information to which ever words they're relevant to
e.g.



· fluffy, affect creature,
↓ · moves it to a different part of this space, that encodes "fluffy"
? How to make one affect, change meanings?



· multiplying is like saying:
· if this word is relevant to adjusting (modifier) the meaning of something else,
then ∴ what exactly should be added to the embeddings of that something else, in order to reflect this word?

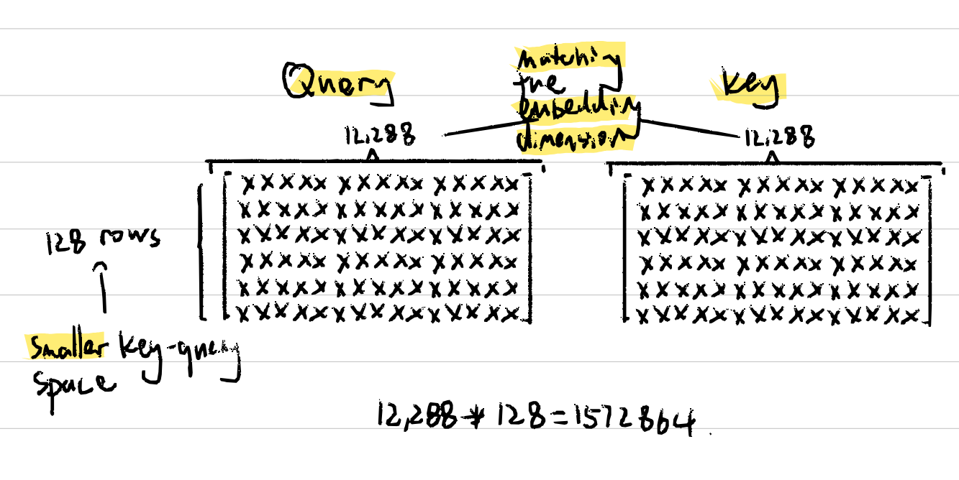
creature
$\vec{E}_4$

"a" $\to \vec{V}$ $0.00 \cdot \vec{V}$
"fluffy" $\to \vec{V}$ $+ 0.42 \cdot \vec{V}$
"blue" $\to \vec{V}$ $+ 0.58 \cdot \vec{V}$
"creature" $\to \vec{V}$ $+ 0.00 \cdot \vec{V}$
"roamed" $\to \vec{V}$ $+ 0.00 \cdot \vec{V}$
$= \Delta\vec{E}_4$

$$\begin{matrix} \vec{E}_1 & + \vec{E}_2 & + \vec{E}_3 & + \vec{E}_4 & + \cdots \\ + & + & + & + & \\ \Delta\vec{E}_1 & \Delta\vec{E}_2 & \Delta\vec{E}_3 & \Delta\vec{E}_4 & + \cdots \\ \| & \| & \| & \| & \\ \vec{E}_1' & \vec{E}_2' & \vec{E}_3' & \vec{E}_4' & \end{matrix}$$

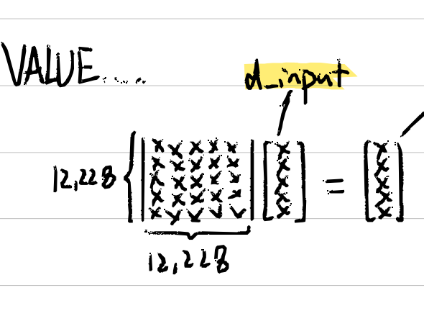
· Single head of attention
← updated vectors!

dimensions:



12,288 * 128 = 1572864

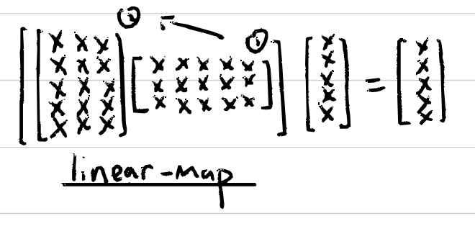
VALUE....

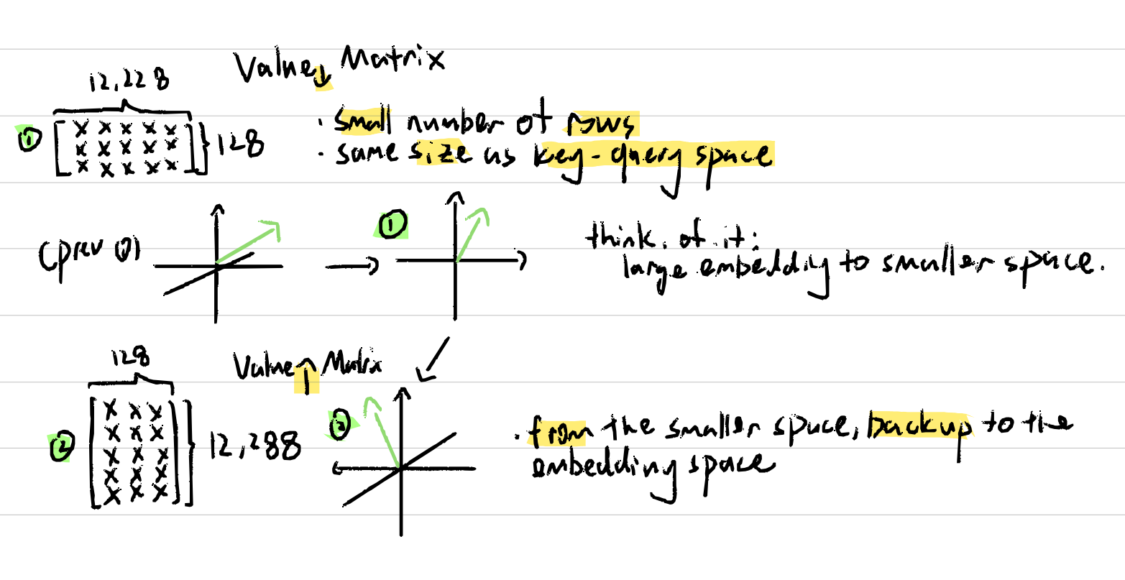


12,228 · 12,228 = 150994944
· You can do this.....

· But usually #Query + #Key = #value parameter
- especially important in multi-head attention

What this means is that: value map is factored as
→ product of two smaller matrices





· Small number of rows
· same size as key-query space

think of it: large embedding to smaller space.

· from the smaller space, backup to the embedding space

Linear-map: constraining overall value map
↓
low rank transformation

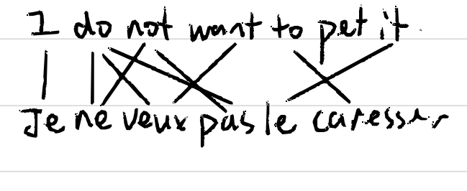
Total parameters....

Embeddings	d_embed (12,288) * d_vocab (50,257)	= 617,558,016	
Key	d_query (12,888) * d_embed (12,288)	= 1,572,864	per head
Query	d_query (12,888) * d_embed (12,288)	+ = 1,572,864	per head
Value↓	d_value (128) * d_embed (12,288)	+ = 1,572,864	per head
Value↑	d_embed (12,288) * d_value (128)	+ = 1,572,864	per head
		= 6,291,456 (per attention head)	

· All described above: self-attention head

Cross-Attention:

· Two types of data: e.g. one language w/ another language.
or audio input w/ speech



I do not want to pet it.
↓ ↓ ↓ ↓ ↓ ↓ ↓
E E E E E E E

Je → $\vec{F} \xrightarrow{W^K} K'$ QK QK ......
ne → $\vec{F} \xrightarrow{W^K} K'$ QK QK ......
veux → $\vec{F} \xrightarrow{W^K} K'$ QK QK ......

· the key might come from one language, query from another
· attention can tell which words from which language, correspond to which words in another
· no masking, since no notion of
· later token affecting earlier ones

Multi-head Attention

· How can we do this 10,000 times?
+ Prev e.g. we focus adj on nouns, & different ways context can influence the meanings of the word.

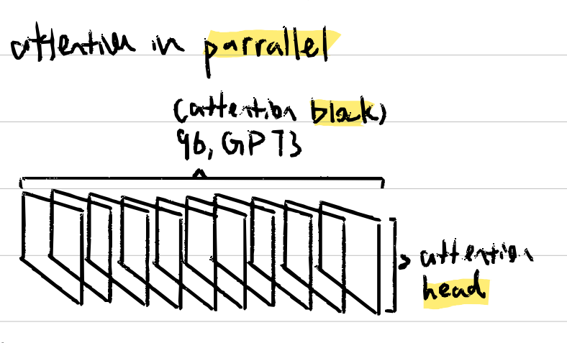
e.g. 1: when sudden they crash the car into a tree
all on "car"

2: when wizard... Hogwarts... Hermione... Harry
(Harry potter)
comparing to:
Queen... Sussex... william... Harry

· For every different type of contextual update, the parameters of these query/key will be different!
· Different value map (parameter): based on what should be added
· weight set whatever the best to be, to predict the next token

· Prev: Single-HEAD-of attention,
· full attention block in Transformer, multi-headed attention.
· Running attention in parallel

(attention block)
96, GPT3



· each with it's own W_Q, W_K, $\downarrow W_V$, $\uparrow W_V$
(attention head)

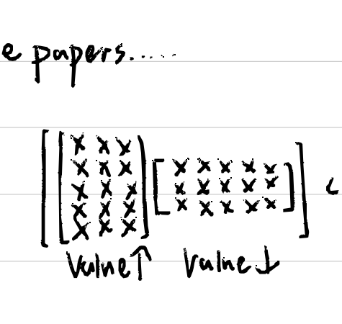
+ 96 W_Q, W_K
+ 96 attention pattern (distinct)
+ 96 $\downarrow W_V$, $\uparrow W_V$
• each head produces a proposed change to the embedding in that position

Original embedding $\vec{E}_i + \Delta\vec{E}_i^{(1)} + \Delta\vec{E}_i^{(2)} + \Delta\vec{E}_i^{(3)} + \Delta\vec{E}_i^{(4)} + \Delta\vec{E}_i^{(5)} + \cdots$

Sum-ups: running distinct heads in parallel, giving the model the capacity to learn distinct ways of that context
· distinct ways that context changes meanings

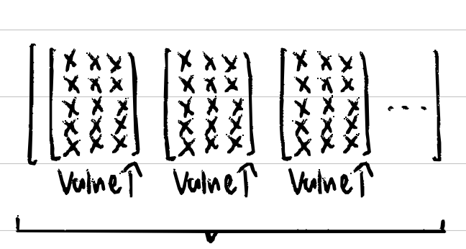
Embeddings	d_embed (12,288) * d_vocab (50,257)		
Key	d_query (12,888) * d_embed (12,288) * n_heads (96)	= 150,994,944	(layer)
Query	d_query (12,888) * d_embed (12,288) * n_heads (96)	+ = 150,994,944	(layer)
Value↓	d_value (128) * d_embed (12,288) * n_heads (96)	+ = 150,994,944	(layer)
Value↑	d_embed (12,288) * d_value (128) * n_heads (96)	+ = 150,994,944	(layer)
		= 603,979,776	

in the papers.....



← instead of that (this structure in each attention head)

more as... →



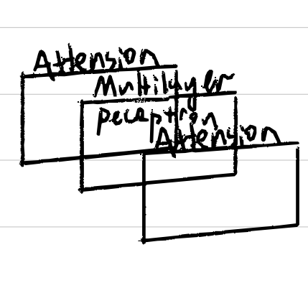
Output matrix
↓
associated with entire multi-headed attention block

· when people refering to value matrix, usually only Value↓ (first step)
Because: you can save computation on running weighted sums on smaller intermediate output produce by the Value↓
* embeddings → Value↓ (128 dimensions) → times with weights (K·Q) for weighted sum *
↓
then times up with Value↑

ppl love to compress into single matrix multiplication

because ppl love single matrix opperations:
· each position, you put together all the 128 dimension vectors, to a big vector with length of (96 * 128)
· then apply the Output Matrix

Going deeper...

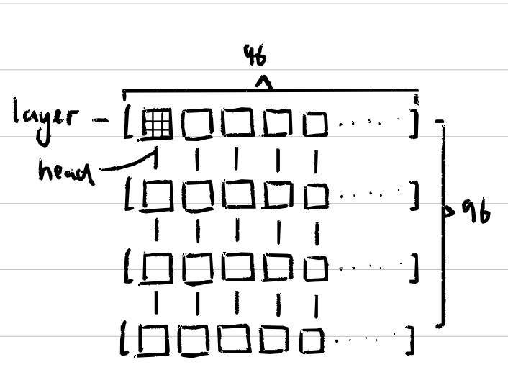


Attention
↓
Multilayer perceptron
↓
Attention
↓
Multilayer perceptron
↓
.....
(kept going)

"After a word with it's context, more chances of this more nuanced embedding, to be influen by it more nuanced surroundings!
· taking more meanings from more embeddings
→ themselves more nuanced

· Hope to encode higher level & more abstract ideas of input
· beyond descriptors and grammatic inputs
e.g. sentiment, tone, underlining scientific piece.....

Embeddings	d_embed (12,288) * d_vocab (50,257)		
Key	d_query (12,888) * d_embed (12,288) * n_heads (96) * n_layer (96)	= 14,495,514,624	
Query	d_query (12,888) * d_embed (12,288) * n_heads (96) * n_layer (96)	+ = 14,495,514,624	
Value↓	d_value (128) * d_embed (12,288) * n_heads (96) * n_layer (96)	+ = 14,495,514,624	
Value↑	d_embed (12,288) * d_value (128) * n_heads (96) * n_layer (96)	+ = 14,495,514,624	
		= 57,982,058,496	



About 1/3
Attention is ~~All~~ You Need
+ in between blocks
+ training processes..

the success
- enables specific kind of behavior
+ the fact that it's extremely parallelizable