Reading 12

☆ Transformers : the tech behind LLMs

- GPT : Generative Pretrained Transformer
  - ↳ Transformer is a specific type of neural network
  - ↳ the reason behind the current AI boom

- Remember chatbots are simply predicting the next word. They choose from a probability distribution of the next word.

- How is the next word produced?
  - ↳ 1. The input is broken up into small pieces called "tokens"
  - 2. Each token is associated with a vector that encodes the meaning of that token.
    - ↳ words with similar meanings tend to have the vector end up is the same ballpark area in the high dimensional space.
  - 3. Then the sequence of vectors passes through an operation called an attention block
    - ↳ the attention block allows the vectors to talk to each other and update their values
    - ↳ for example the meaning "model" can be different depending on context.
  - 4. Then the vectors pass through a MLP or feed-forward layer.
    - ↳ vectors go through same operation in parallel
    - ↳ vectors do not talk to each other.

Note! there are normalization steps in between and each operation in attention and MLP is a matrix multiplication.

Note! A higher dimensional array is called a tensor.

- The model has a predefined vocabulary.
  - ↳ the embedding matrix has a single column for each one of these words
  - ↳ the columns determine what vector each token turns into
  - ↳ the embedding matrix is denoted W_E
  - ↳ the values begin random, but then are learned based on data.

- Word embeddings turn tokens into vectors in a high dimensional space and each dimension in that space tends to carry some semantic meaning.
  - ↳ Ex $E(\text{man}) - E(\text{woman}) \approx E(\text{king}) - E(\text{queen})$
    - ↳ so this implies one dimension in the space encodes gender information.

- The dot product of two vectors can be thought of as a way to measure how well they align.
  - ↳ it is positive when vectors point in similar directions
  - ↳ it is 0 if they are perpendicular
  - ↳ it is negative if they point in opposite directions.

- The embedding matrix is the first set of weights in our model
- These "word vectors" also carry information about the position of that token
  - ↳ they have the capacity to soak in context

- Initially each word vector is just pulled from the embedding matrix, but the goal is to modify it to be more specific given the context.

- Network can only process a fixed number of vectors at a given time, which is known as context size.
  - ↳ this limits how much text the transformer can incorporate when its making its prediction of the next word.

- At the very end remember the desired output is a probability distribution over all tokens that might come next.
  - ↳ In order to do this we need another matrix that maps the very last vector in the context to a list the size of the models vocabulary. (remember matrices are just linear transformations)
    - ↳ this is the unembedding matrix denoted W_U
    - ↳ its W_E is $M \times N$ then W_U is $N \times M$.
    - ↳ so it accounts for the same number of weights in the model as W_E.
  - ↳ Then Softmax is run on the output to normalize into a probability distribution.
    - ↳ Remember Sofmax is a function that maps an arbitrary list of numbers to a probability distribution.
      - ↳ higher numbers map closer to 1, lower numbers map closer to 0.
    - ↳ the inputs to softmax are referred to as logits.
    - ↳ the outputs of Softmax are probabilities

- Temperature T parameter is often thrown into softmax.
  - ↳ when T is higher, lower values get more weight in comparison to when T is low
    - ↳ results in a more uniform distribution
  - ↳ when T is smaller, the bigger values tend to dominate.
    - ↳ when $T=0$, all weight goes to maximum value.

✯ Attention in transformers:

- Remember a transformer has many attention blocks.

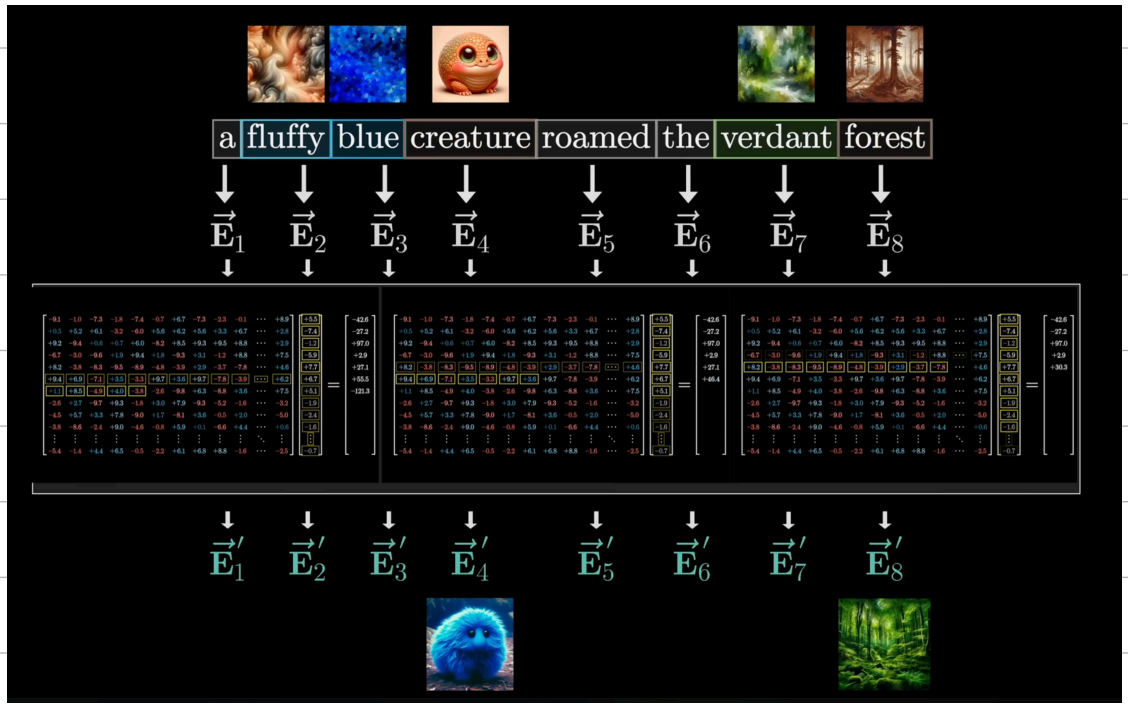
- Ex "... therefore, the player was ???"

  ↳ the next token is only predicted based on the vector for "was."

  ↳ the "was" embedding started as just the corresponding column from the embedding matrix W_E

  ↳ but by the end of all the attention blocks its been updated to carry all the relevant info for predicting the next token from the context window

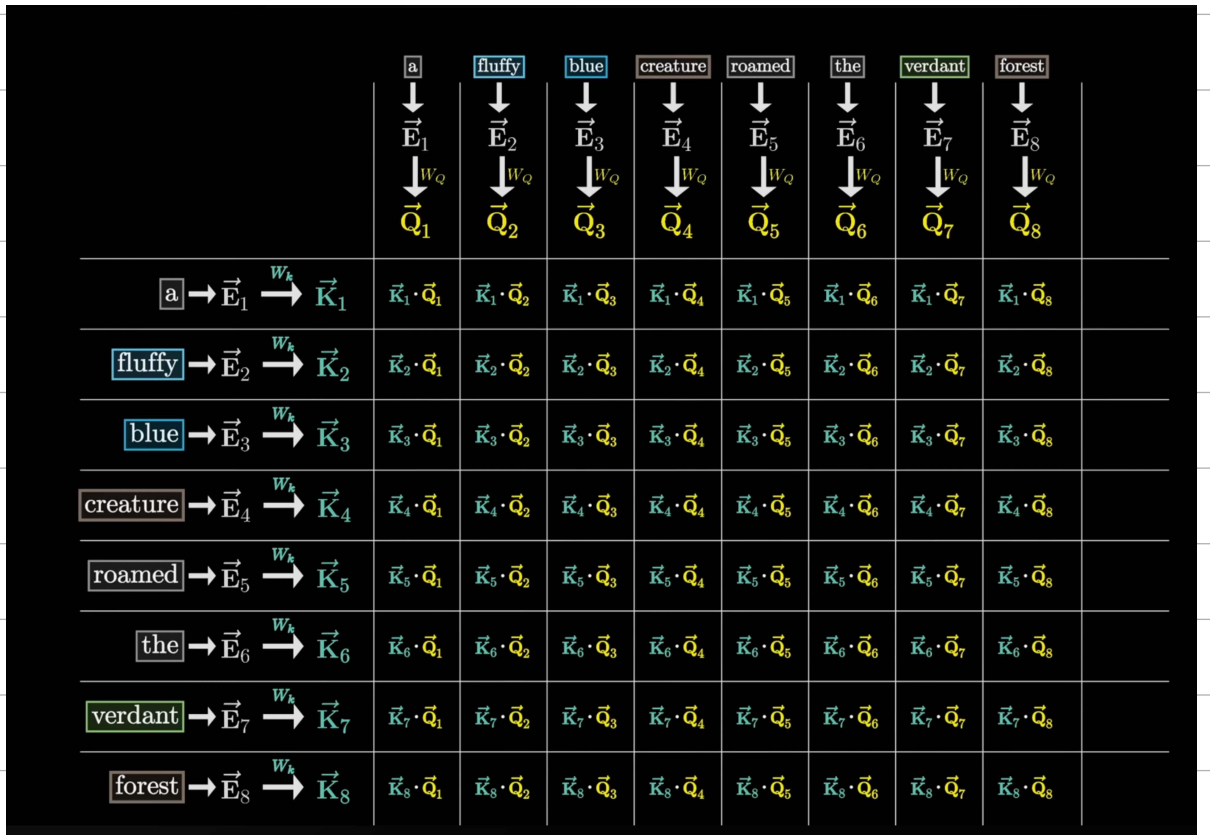
- Ex Only focused on updating the nouns vector $\vec{E}_i$ with its corresponding adjectives (this is a single head of attention):



- Each token has "asks a question" embedded in a query vector.
  - ↳ in our Ex the question would be "Are there any adjectives in front of me?" (nouns ask this question).
    - ↳ query vec has a smaller dimensionality than the embedding vector.
    - ↳ computing the query is multiplying a matrix W_Q with the embedding vec.
      - ↳ this computation produces a query vector.
      - ↳ do this for each token.
    - ↳ W_Q is a matrix of weights in our model that are learned. (like W_E and W_u)

↳ there's also a key matrix W_K that is multiplied by each embedding vector $\vec{E}_i$.
- ↳ think of this operation as answering the query
- ↳ W_K is also full of tunable weights.
- ↳ the keys match the query when the dot product of resulting vecs is higher

- Ex:



	a ($\vec{E}_1$, W_Q → $\vec{Q}_1$)	fluffy ($\vec{E}_2$, W_Q → $\vec{Q}_2$)	blue ($\vec{E}_3$, W_Q → $\vec{Q}_3$)	creature ($\vec{E}_4$, W_Q → $\vec{Q}_4$)	roamed ($\vec{E}_5$, W_Q → $\vec{Q}_5$)	the ($\vec{E}_6$, W_Q → $\vec{Q}_6$)	verdant ($\vec{E}_7$, W_Q → $\vec{Q}_7$)	forest ($\vec{E}_8$, W_Q → $\vec{Q}_8$)
a → $\vec{E}_1$ → (W_K) $\vec{K}_1$	$\vec{K}_1 \cdot \vec{Q}_1$	$\vec{K}_1 \cdot \vec{Q}_2$	$\vec{K}_1 \cdot \vec{Q}_3$	$\vec{K}_1 \cdot \vec{Q}_4$	$\vec{K}_1 \cdot \vec{Q}_5$	$\vec{K}_1 \cdot \vec{Q}_6$	$\vec{K}_1 \cdot \vec{Q}_7$	$\vec{K}_1 \cdot \vec{Q}_8$
fluffy → $\vec{E}_2$ → (W_K) $\vec{K}_2$	$\vec{K}_2 \cdot \vec{Q}_1$	$\vec{K}_2 \cdot \vec{Q}_2$	$\vec{K}_2 \cdot \vec{Q}_3$	$\vec{K}_2 \cdot \vec{Q}_4$	$\vec{K}_2 \cdot \vec{Q}_5$	$\vec{K}_2 \cdot \vec{Q}_6$	$\vec{K}_2 \cdot \vec{Q}_7$	$\vec{K}_2 \cdot \vec{Q}_8$
blue → $\vec{E}_3$ → (W_K) $\vec{K}_3$	$\vec{K}_3 \cdot \vec{Q}_1$	$\vec{K}_3 \cdot \vec{Q}_2$	$\vec{K}_3 \cdot \vec{Q}_3$	$\vec{K}_3 \cdot \vec{Q}_4$	$\vec{K}_3 \cdot \vec{Q}_5$	$\vec{K}_3 \cdot \vec{Q}_6$	$\vec{K}_3 \cdot \vec{Q}_7$	$\vec{K}_3 \cdot \vec{Q}_8$
creature → $\vec{E}_4$ → (W_K) $\vec{K}_4$	$\vec{K}_4 \cdot \vec{Q}_1$	$\vec{K}_4 \cdot \vec{Q}_2$	$\vec{K}_4 \cdot \vec{Q}_3$	$\vec{K}_4 \cdot \vec{Q}_4$	$\vec{K}_4 \cdot \vec{Q}_5$	$\vec{K}_4 \cdot \vec{Q}_6$	$\vec{K}_4 \cdot \vec{Q}_7$	$\vec{K}_4 \cdot \vec{Q}_8$
roamed → $\vec{E}_5$ → (W_K) $\vec{K}_5$	$\vec{K}_5 \cdot \vec{Q}_1$	$\vec{K}_5 \cdot \vec{Q}_2$	$\vec{K}_5 \cdot \vec{Q}_3$	$\vec{K}_5 \cdot \vec{Q}_4$	$\vec{K}_5 \cdot \vec{Q}_5$	$\vec{K}_5 \cdot \vec{Q}_6$	$\vec{K}_5 \cdot \vec{Q}_7$	$\vec{K}_5 \cdot \vec{Q}_8$
the → $\vec{E}_6$ → (W_K) $\vec{K}_6$	$\vec{K}_6 \cdot \vec{Q}_1$	$\vec{K}_6 \cdot \vec{Q}_2$	$\vec{K}_6 \cdot \vec{Q}_3$	$\vec{K}_6 \cdot \vec{Q}_4$	$\vec{K}_6 \cdot \vec{Q}_5$	$\vec{K}_6 \cdot \vec{Q}_6$	$\vec{K}_6 \cdot \vec{Q}_7$	$\vec{K}_6 \cdot \vec{Q}_8$
verdant → $\vec{E}_7$ → (W_K) $\vec{K}_7$	$\vec{K}_7 \cdot \vec{Q}_1$	$\vec{K}_7 \cdot \vec{Q}_2$	$\vec{K}_7 \cdot \vec{Q}_3$	$\vec{K}_7 \cdot \vec{Q}_4$	$\vec{K}_7 \cdot \vec{Q}_5$	$\vec{K}_7 \cdot \vec{Q}_6$	$\vec{K}_7 \cdot \vec{Q}_7$	$\vec{K}_7 \cdot \vec{Q}_8$
forest → $\vec{E}_8$ → (W_K) $\vec{K}_8$	$\vec{K}_8 \cdot \vec{Q}_1$	$\vec{K}_8 \cdot \vec{Q}_2$	$\vec{K}_8 \cdot \vec{Q}_3$	$\vec{K}_8 \cdot \vec{Q}_4$	$\vec{K}_8 \cdot \vec{Q}_5$	$\vec{K}_8 \cdot \vec{Q}_6$	$\vec{K}_8 \cdot \vec{Q}_7$	$\vec{K}_8 \cdot \vec{Q}_8$

- where $\vec{K}_i \cdot \vec{Q}_j \in \mathbb{R}$

- If the dot product of "fluffy" and "creature" is high, then "fluffy" attends to "creature"

- We use all the dot products to take a weighted sum along each column, weighted by the relevance.
  - ↳ instead of having the values in the columns range from $(-\infty, \infty)$ we want them to range from $[0,1]$
    - ↳ and add to 1.
  - ↳ So we compute a softmax along each column to normalize the values.
  - ↳ So then the matrix looks like this:

	a $\vec{E}_1 \downarrow W_Q \vec{Q}_1$	fluffy $\vec{E}_2 \downarrow W_Q \vec{Q}_2$	blue $\vec{E}_3 \downarrow W_Q \vec{Q}_3$	creature $\vec{E}_4 \downarrow W_Q \vec{Q}_4$	roamed $\vec{E}_5 \downarrow W_Q \vec{Q}_5$	the $\vec{E}_6 \downarrow W_Q \vec{Q}_6$	verdant $\vec{E}_7 \downarrow W_Q \vec{Q}_7$	forest $\vec{E}_8 \downarrow W_Q \vec{Q}_8$
a $\vec{E}_1 \xrightarrow{W_K} \vec{K}_1$	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fluffy $\vec{E}_2 \xrightarrow{W_K} \vec{K}_2$	0.00	1.00	0.00	0.42	0.00	0.00	0.00	0.00
blue $\vec{E}_3 \xrightarrow{W_K} \vec{K}_3$	0.00	0.00	1.00	0.58	0.00	0.00	0.00	0.00
creature $\vec{E}_4 \xrightarrow{W_K} \vec{K}_4$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
roamed $\vec{E}_5 \xrightarrow{W_K} \vec{K}_5$	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
the $\vec{E}_6 \xrightarrow{W_K} \vec{K}_6$	0.00	0.00	0.00	0.00	0.99	1.00	0.00	0.00
verdant $\vec{E}_7 \xrightarrow{W_K} \vec{K}_7$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00
forest $\vec{E}_8 \xrightarrow{W_K} \vec{K}_8$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

- Think of each value denoting how important the word on the left is to the word on the right.
- We call this grid an attention pattern.

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

- You also don't want later tokens to affect earlier ones.
  - ↳ so you set all entries below diagonal to $-\infty$, and then apply softmax.

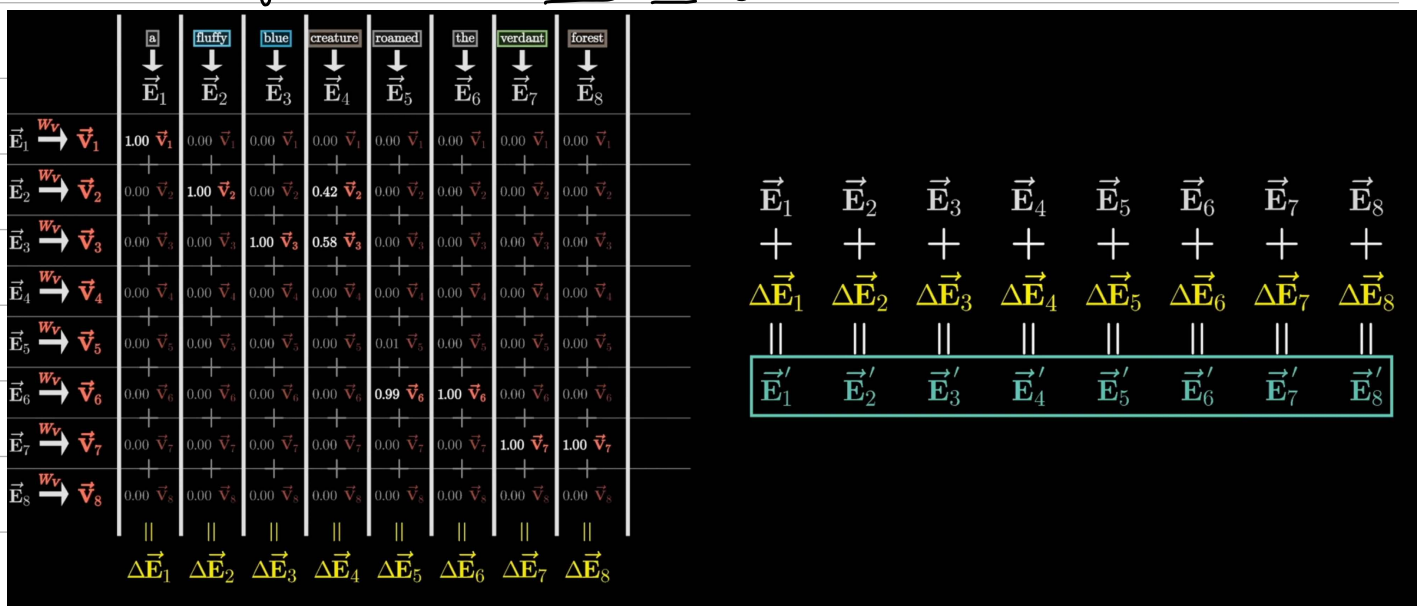
Unnormalized Attention Pattern							Normalized Attention Pattern					
+3.53	+0.80	+1.96	+4.48	+3.74	−1.95	softmax →	1.00	0.75	0.69	0.92	0.46	0.00
−∞	−0.30	−0.21	+0.82	+0.29	+2.91		0.00	0.25	0.08	0.02	0.01	0.46
−∞	−∞	+0.89	+0.67	+2.99	−0.41		0.00	0.00	0.24	0.02	0.22	0.02
−∞	−∞	−∞	+1.31	+1.73	−1.48		0.00	0.00	0.00	0.04	0.06	0.01
−∞	−∞	−∞	−∞	+3.07	+2.94		0.00	0.00	0.00	0.00	0.24	0.48
−∞	−∞	−∞	−∞	−∞	+0.31		0.00	0.00	0.00	0.00	0.00	0.03

- This process is called masking. (not letting later tokens affect earlier ones)

- Note that the size of the attention grid is equal to the square of the context size.
  - ↳ So context size can be a bottleneck for models.

- To affect other tokens ("fluffy creature"), the vector for "fluffy" is multiplied by the value matrix W_v which produces a value vector.
  - ↳ You then add the value vector to the embedding of "creature".
  - ↳ W_v is full of tunable weights

- This is the process (of a single head of attention):



- In practice you want number of parameters devoted to $W_Q + W_K$ to equal number of parameters devoted to W_V.
  - ↳ so the value map is split into two smaller matrices
    - ↳ so you're composing 2 linear transformations to get to the same output
      - ↳ but this requires less parameters.
    - ↳ these two matrices are:
      - ↳ Value down matrix: $W_{V\downarrow}$, maps embedding vecs to smaller space.
      - ↳ Value up matrix: $W_{V\uparrow}$, maps from the smaller space back up to embedding space.
    - ↳ $W_{V\downarrow}$ and $W_{V\uparrow}$ are full of tunable parameters and replace W_V in practice.

- What we've looked at is self attention.
- There is also cross attention that process two distinct types of data:
  - ↳ like text in one language and text in another language.
  - ↳ the only difference is that W_K and W_Q map to different datasets.
    - ↳ there is typically no masking here since there is no notion of later tokens affecting earlier tokens.

- A full attention block inside a transformer is consisted of multi-headed attention.
  - ↳ a lot of attention operations running in parallel.
    - ↳ each operation having its own W_Q, W_K, $W_{V\downarrow}$, $W_{V\uparrow}$
  - ↳ so each operation produces a proposed changed to a given token.
  - ↳ so you simply add the sum of all the changes from each operation to the original embedding.

- By running many attention heads in parallel, you are giving the model many distinct ways that contains meaning.

- In practice all the $W_{v\uparrow}$ matrices for a attention block appear stapled or concatenated together into something called the output matrix.
  - ↳ This matrix is associated with the entire multi-headed attention block.

- And in practice when people refer to the value matrix of a given attention head, they are only referring to the $W_{v\downarrow}$ matrix we mentioned earlier.

- Attention accounts for about 1/3 of the total parameters in the model.
- Also attention is highly parallelizable
  - ↳ so you can run a huge amount of computations at once on a GPU.