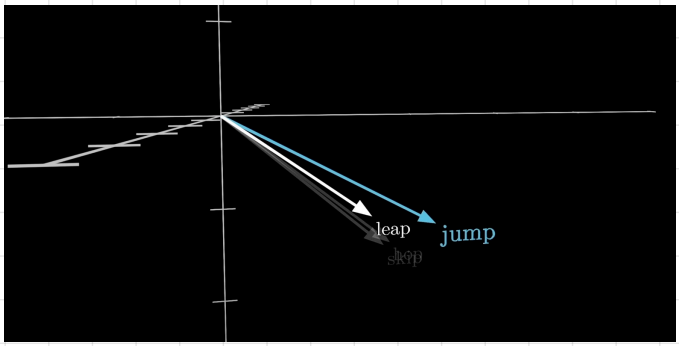
Understanding Transformers - 3B1B video (04/01/24)

- GPT - generative pre-trained transformer
- The way chatgpt works is via a "what comes next" prediction
  - Suppose we have a half-done story, then gpt produces a probability distribution on what the next word could be (it takes the highest one)

(1) The input data/text is first broken up into bunch of little pieces → they are called tokens
  - If data is text, then tokens are words, if it's audio, then it is small samples or small patches if it's an image

  - Each token is associated with a vector, we can think about them as coordinates in a very high dimensional space
    - Similar words ⇒ similar vectors



(2) These sequence of vectors are then passed through an "Attention" block

  - This allows vectors to talk to each other & pass info. back & forth to update their values

Come back to this

  - It also responsible for figuring out which words in context are relevant to updating the meanings of which other words

* note that the word "meaning" means the vector *

(3) After this, it is passed through an MLP (multi-layer perceptron), also referred to as feed-forward layer

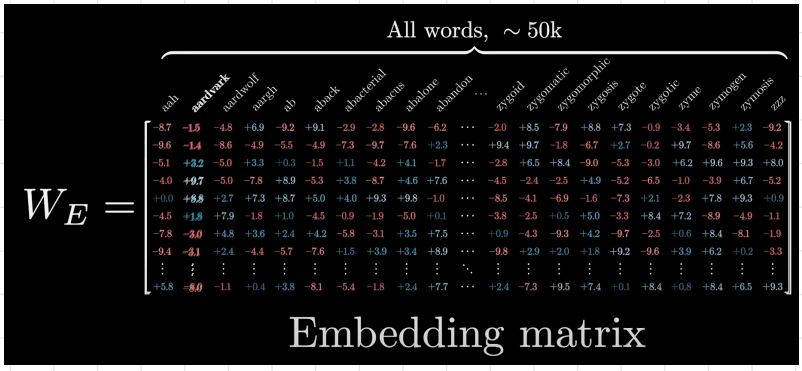
  - Each vector goes through the operation in parallel
  - Think about this step as asking the vector a long list of questions & then updating based on answers

(4) This process is then repeated a bunch of times. After all that, the last vector produced will create a prob. distribution to predict next word

---

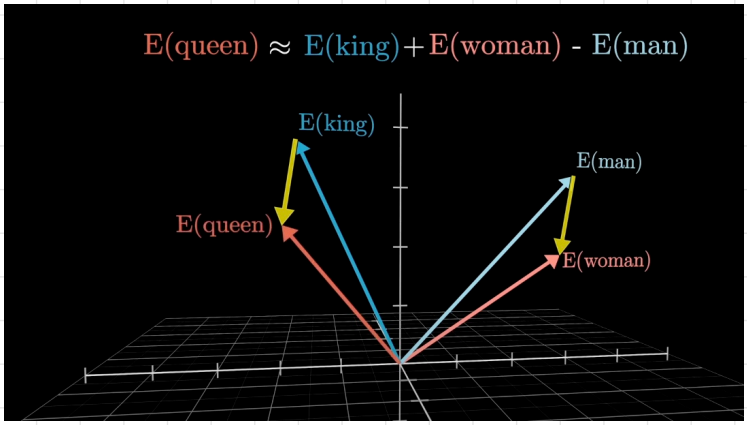
Word Embeddings:

- For NLP models, they have a predefined vocabulary (all the words)
- An embedding matrix is made, each column represents the vector for a word



- When we assign these words a vector, it is called "embedding" them
- Directions in vector space carry semantic meaning
  - Similar vector ⇒ similar meanings

- Difference between vectors can be used to find other words



- We can dot 2 vectors & figure out similarity
  - ↳ If positive, similar, if negative, not similar
- The vectors contain information beyond the word, for example, it contains information about what they do/did, position in the text, etc.
- Context size are the number of vectors the model can process at 1 time
  - ↳ For GPT-3, it was 2048

How the output is produced

- Once the last vector is produced, we want to turn it into a probability dist.
  - ↳ So, we [matrix multiply all tokens in the vocab] with the final vector
    - we call this the unembedding matrix, has a row for each word in vocab & each row has same # of elements as the embedding dim. (12,288)
- However, we still need to normalize the output & this is done via SoftMax

Understanding Softmax:

- Recall for prob. distributions, each value $\in [0, 1]$ and sum(all elements) = 1
- When we do matrix mul, we get values that are negative or above 1 & it almost never sums up to 1
- Softmax takes in an arbitrary list and converts to a valid distribution
  - ↳ large values end up close to 1 & smaller vals. end up closer to 0
- Here is how it works:

Softmax

$$\underbrace{\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}}_{\text{Logits}} \longrightarrow \begin{bmatrix} e^{z_1} \div \sum_{n=0}^{N-1} e^{z_n} \\ \vdots \\ e^{z_n} \div \sum_{n=0}^{N-1} e^{z_n} \end{bmatrix}$$

  - ↳ e^{z_n} converts value to a non-negative # & dividing by the sum normalizes to a value between 0 and 1

- GPT models use an extra parameter called temperature: $e^{z_n/T} \Big/ \sum_{n=0}^{N-1} e^{z_n/T}$ (Temperature)
  - ↳ When T is big, smaller values benefit, but when T is small, bigger values dominate
- Side note: gpt won't let you pick T > 2

Understanding Self-Attention

- We can measure similarity between words in 3 ways:

  1) dot product
  2) cosine similarity
  3) scaled dot product → we divide by $\sqrt{n}$, where n is # of dimensions

  $a \cdot b = |a| \cdot |b| \cdot \cos\theta$

- Consider: "an apple and an orange"

  Apple (2,2)
  Orange (0,3)
  Tech (4,0)

  Similarity table (determined via cosine sim. with arbitrary vectors)

	Orange	Apple	And	An
Orange	1	0.71	0	0
Apple	0.71	1	0	0
And	0	0	1	1
An	0	0	1	1

$$\text{Orange} = 1 \cdot \text{orange} + 0.71 \cdot \text{apple}$$
$$\text{Apple} = 0.71 \cdot \text{orange} + 1 \cdot \text{apple}$$
$$\text{And} = 1 \cdot \text{And} + 1 \cdot \text{An}$$
$$\text{An} = 1 \cdot \text{And} + 1 \cdot \text{An}$$

Put in matrix form, they are also called LOGITS

- Now we normalize values by applying softmax

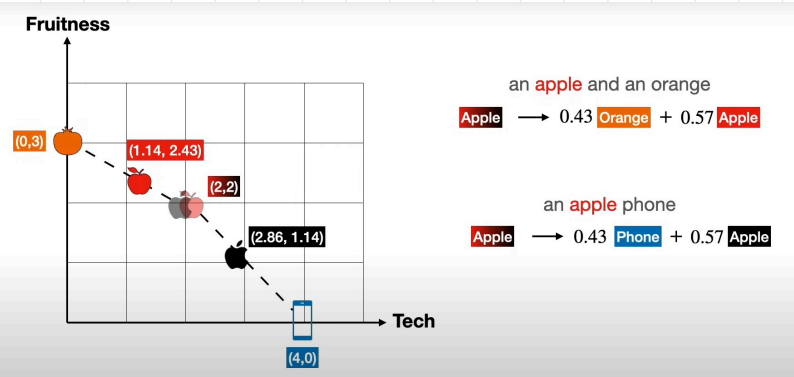
$$\text{Orange} = \frac{e^{1} \cdot \text{orange} + e^{0.71} \cdot \text{apple} + e^{0}\,\text{And} + e^{0}\,\text{An}}{e^{1} + e^{0.71} + e^{0} + e^{0}} = 0.4 \cdot \text{Orange} + 0.3 \cdot \text{Apple} + 0.15\,\text{And} + 0.15\,\text{An}$$

↳ This is how it really should be, but since we are only working with few words, let's not focus on the 0's

$$\text{Orange} = 0.57 \cdot \text{Orange} + 0.43 \cdot \text{Apple}$$
$$\text{Apple} = 0.57 \cdot \text{Apple} + 0.43 \cdot \text{Orange}$$

$$\text{and} = 0.5\,\text{An} + 0.5\,\text{And}$$
$$\text{an} = 0.5 \cdot \text{An} + 0.5 \cdot \text{And}$$

- This is the result:

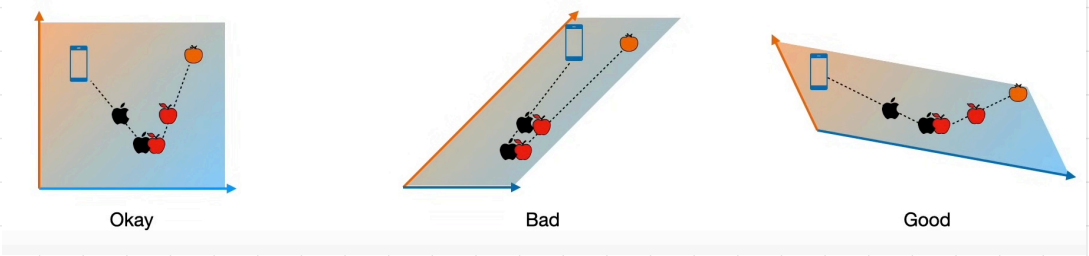


↳ we move the red apple 0.43 towards Orange
↳ we move black apple 0.43 towards tech

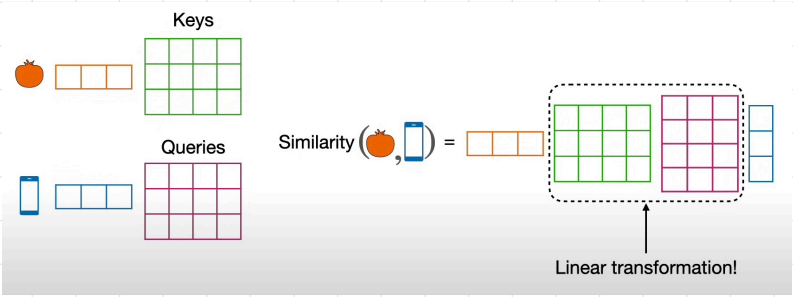
- Now we have a better vector to represent the words, which are better aligned with their meaning
  ↳ remember this process is repeated many times

Understanding Key, Query & Value Matrices

- We can apply linear transformations to our original word embedding matrix to get a matrix that works better when we apply attention:



- The keys & queries matrix are the linear transformation matrix, which takes the original word embedding & transforms it
- Initially key & query matrix is initialized to something random, W^Q & W^K



Setup:

1. **Input Embeddings:** Suppose we have 2 input tokens, and after embedding, each token is represented as a 4-dimensional vector. So, our input embeddings might look like this (let's call this matrix X):

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

Here, X is a matrix with dimensions 2×4 (2 tokens, each represented by a 4-dimensional vector).

2. **Key Matrix W^K:** Assume our transformer model is designed such that the dimensionality for keys (d_k) is 3. The weight matrix W^K for transforming input embeddings into keys will have dimensions 4×3 (since we're mapping from 4-dimensional embeddings to 3-dimensional keys). Let's suppose W^K is as follows:

$$W^K = \begin{bmatrix} 0.5 & -1 & 0 \\ 1 & 0 & 0.5 \\ -0.5 & 1 & -1 \\ 0 & 0.5 & 1 \end{bmatrix}$$

Calculation:

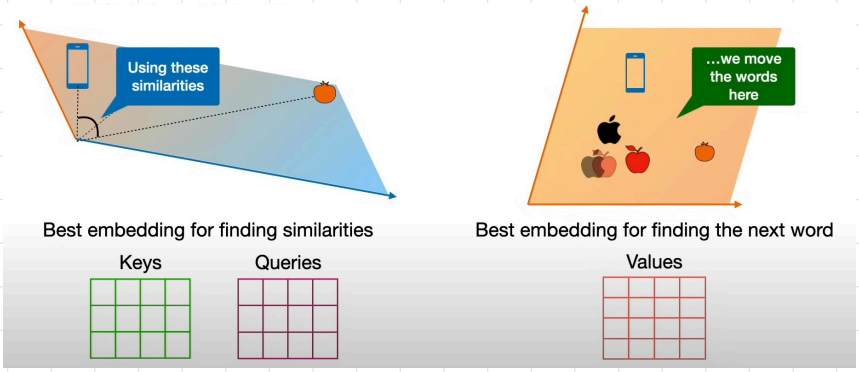
To determine the key matrix K, we multiply the input embeddings matrix X by the weight matrix W^K:

$$K = X \times W^K$$

- More intuitive way of understanding key matrix
- We do $Q \cdot K^T$ to compute similarity matrix and scale it by $\frac{1}{\sqrt{n}}$

Values matrix

- We use $Q \cdot K^T$ to find similarities, but then we update the vectors in only the value matrix:



- We use $K \cdot Q^T$ to find features of the words
- We use values to find next best word given the context
  - ↳ more specifically, we do $Q \cdot K^T \cdot V$, this produces the output of the attention layer
    - ↳ each row represents self-attention output for a token in the sequence

an apple and an orange

	Orange	Apple	And	An
Orange	0.4	0.3	0.15	0.15
Apple	0.3	0.4	0.15	0.15
And	0.15	0.15	0.5	0.5
An	0.15	0.15	0.5	0.5

Value matrix

=

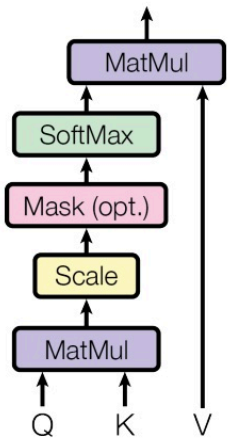
	Orange	Apple	And	An
Orange	v_{11}	v_{12}	v_{13}	v_{14}
Apple	v_{21}	v_{22}	v_{23}	v_{24}
And	v_{31}	v_{32}	v_{33}	v_{34}
An	v_{41}	v_{42}	v_{43}	v_{44}

apple ⟶ $0.3 \cdot$ orange $+0.4 \cdot$ apple $+0.15 \cdot$ and $+0.15 \cdot$ an

apple ⟶ $v_{21} \cdot$ orange $+v_{22} \cdot$ apple $+v_{23} \cdot$ and $+v_{24} \cdot$ an

- Putting everything together, we have: $\text{Attention}(Q,K,V) = \text{softmax}\left[\frac{Q \cdot K^T}{\sqrt{d^K}}\right] \cdot V$

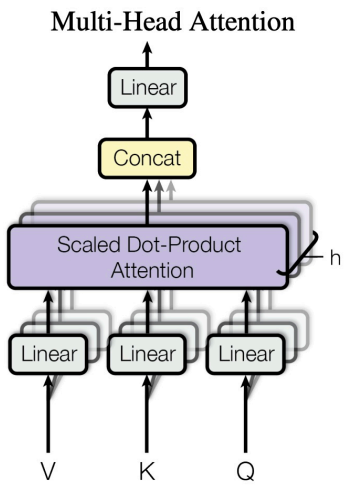
Scaled Dot-Product Attention



↳ To put visually, this is what is happening with QVK matrices

Multihead Attention:

- When we say "multihead", it means we have multiple key, query, & value matrices:



- After performing scaled dot product, we concatenate the matrices:
  - ↳ Suppose we have model dimension is 512 and we have 8 heads, then $\frac{512}{8} = 64$, which is the dim. of each Q, K, V matrix
  - ↳ Each head will produce a $N \times 64$ matrix, thus when we concatenate all heads, we get $N \times 512$.

- Following concatenation, we have a higher dimension output, so we do a linear transformation which maps it down to a lower dimension
  - ↳ We also scale up the better matrices for $Q \cdot K^T$ & scale down the not so good ones

* understand encoder & decoder in transform