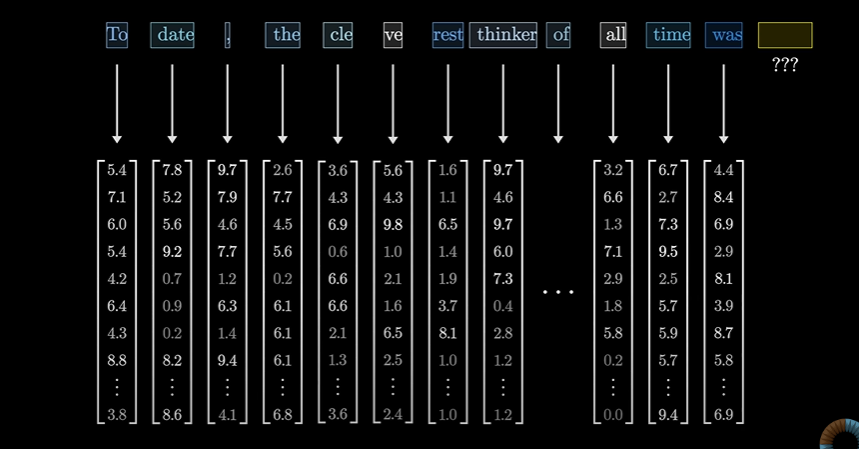
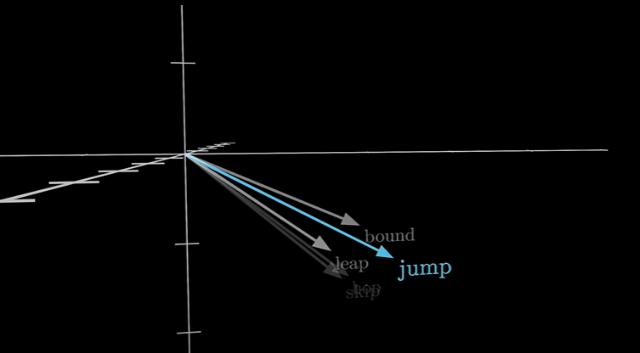
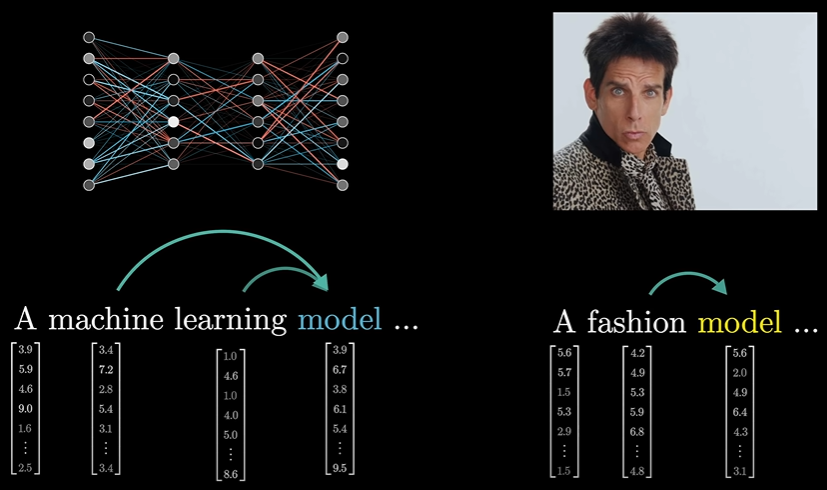
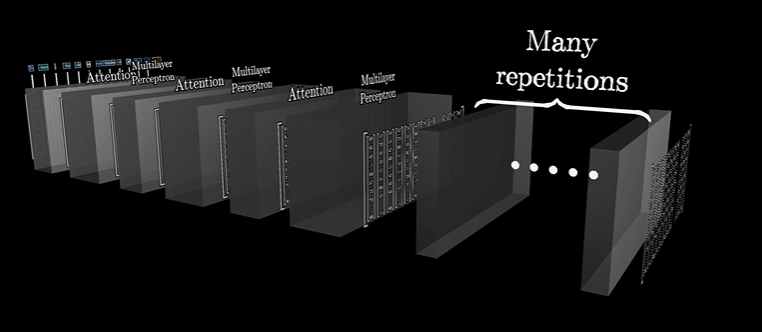
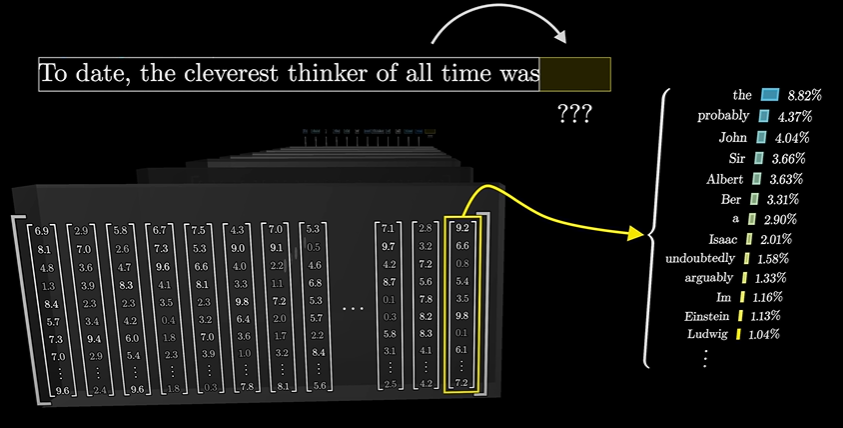
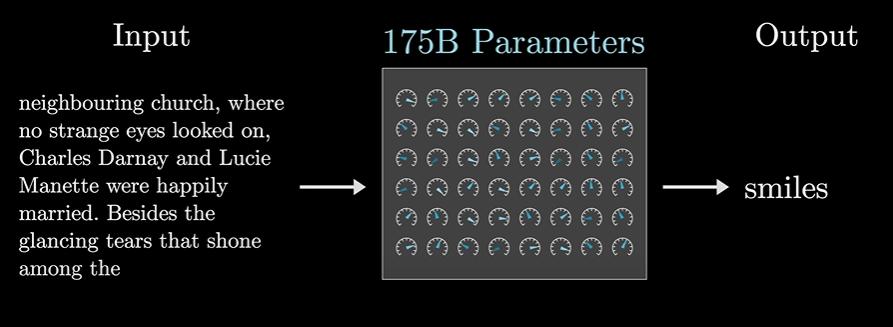
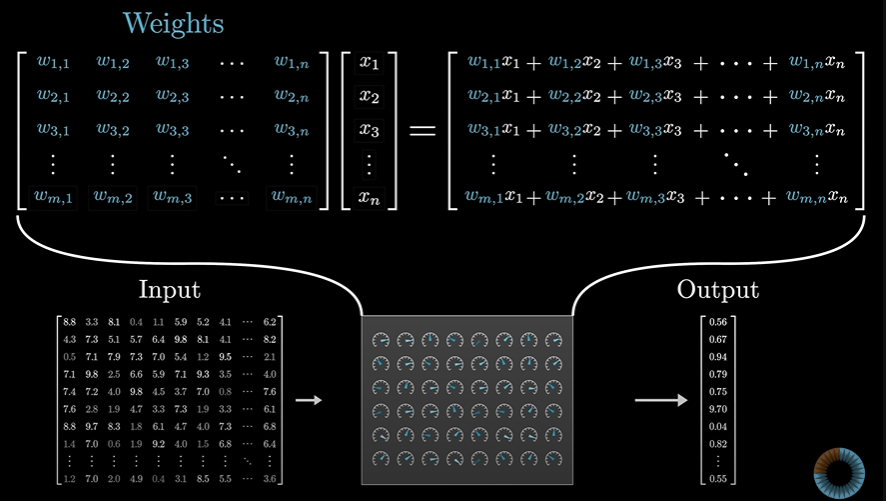
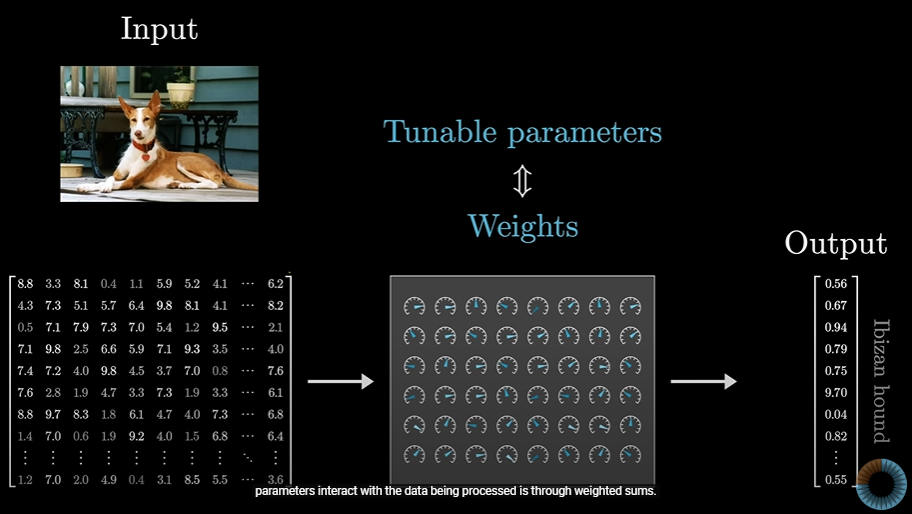
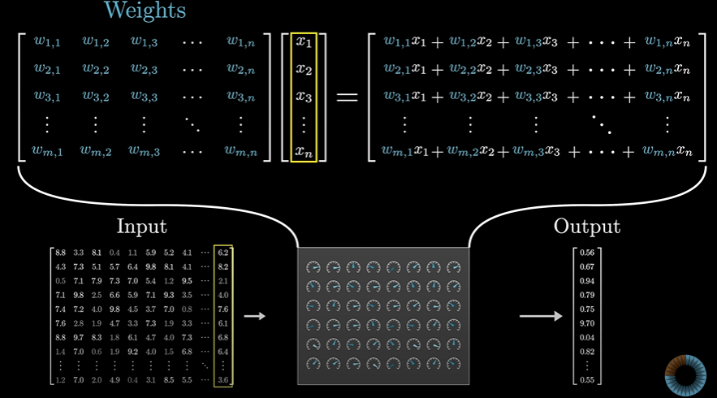
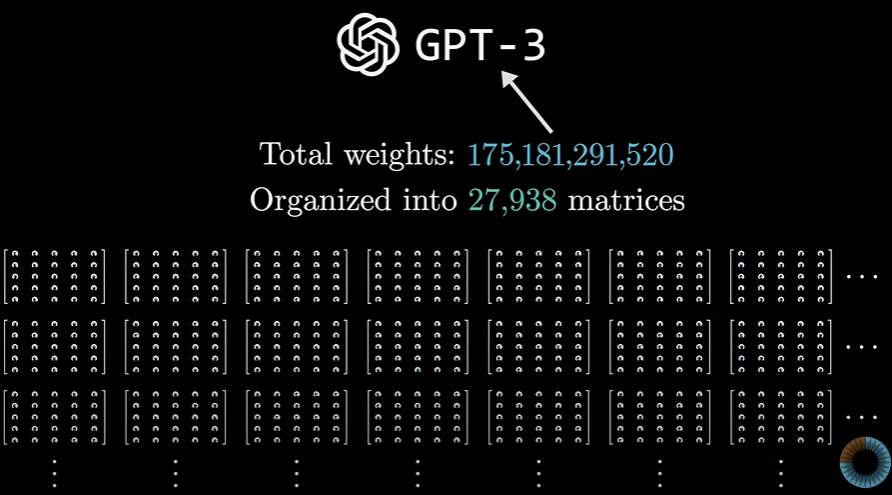
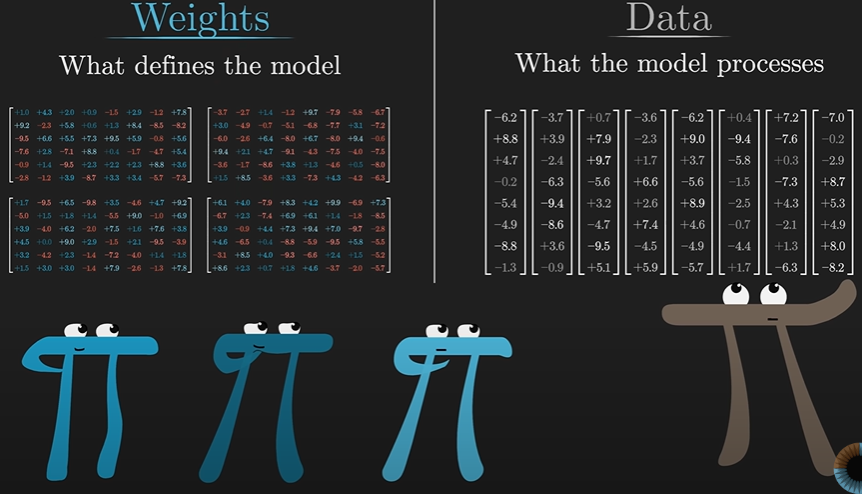
Day 1

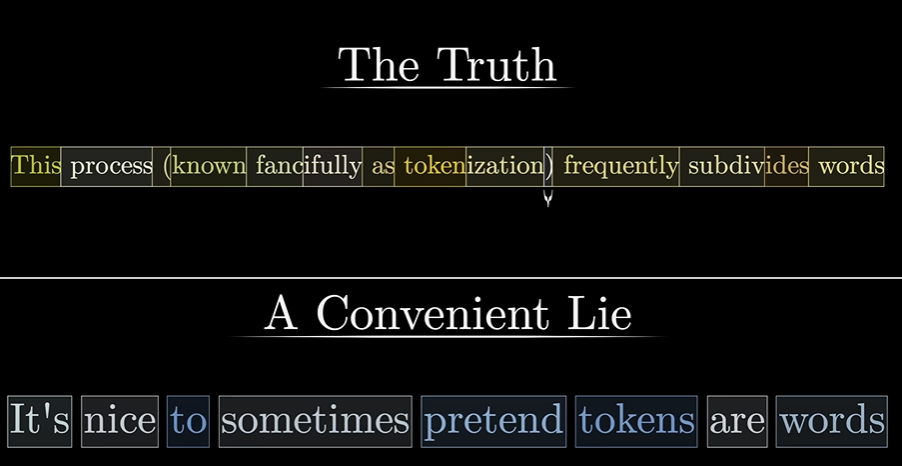
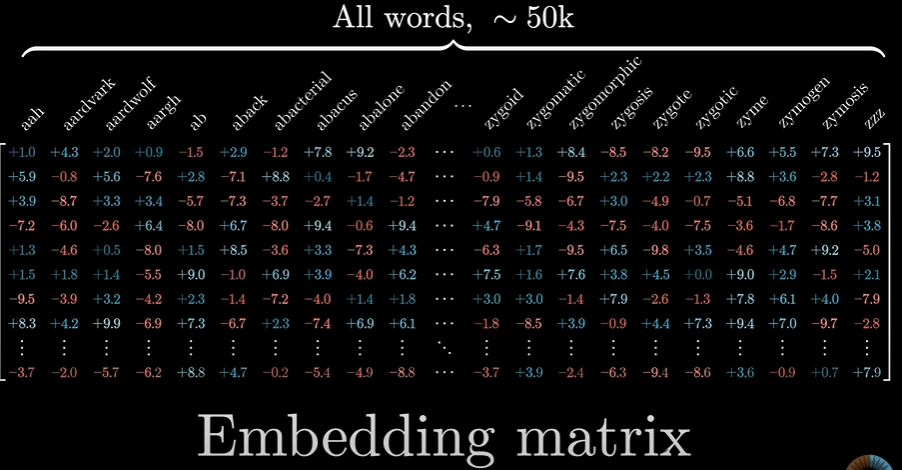
* I'll be trying to make a LLM(large language models) aggregator
* Let's see how much deep I get into this project.
* (llm works according to how we train the language, e.g., human intervention or RLHF(Reinforcement learning with human feedback))
* Providing a single interface to talk to any of the multiple LLM.
* Create a library where I expose some endpoints and users can hit those endpoints to get responses.
* Inspiration: ollama
* Day 1 will include learning about the specific topic like llm
* Usually AI’s uses transformer, e.g., text to image transformer, text to audio transformer.
* The prompt is broken or converted into tokens, could be symbols words, in case of audio or image it could be small portions of those audio or image.
* Each token Is associated with a vector:
* Words with similar meanings tend to land on vectors that are close to each other in that space
* These sequence of vectors passes through an operation called Attention block, this allows the vector to talk to each to other and pass info to each other to update their values
* Attention block is responsible for figuring out which words in context are relevant to updating the meanings of which other words and how exactly those meanings should be updated.
* After that these vectors are passed to multiplayer perceptron(somewhat same process in parallel)
* Now these both operations are done multiple times:
* Until at the very end the hope is that all of the essential meaning of the passage has somehow been baked into the very last vector in the sequence.
* We then perform certain operations on that last vector that produces a probability distribution over all possible tokens, all possible little chunks of text that might come next.
* now, once you have a tool that predicts what comes next given a snippet of text, you can feed it a little bit of seed text and have it repeatedly play this game of predicting what comes next, sampling from the distribution, appending it, and then repeating over and over.
* Rather than using a program or code to process the input and give out output, we use set of multiple parameters, we then play with the parameters to get the closest match.
* Like in this above pic, (btw chatgpt uses 175 billion parameters), all these parameters are tweaked to find out the best parameters which gives out “smiles” as the output.
* These parameters are called as Weights
* These weights perform sum(matrix)

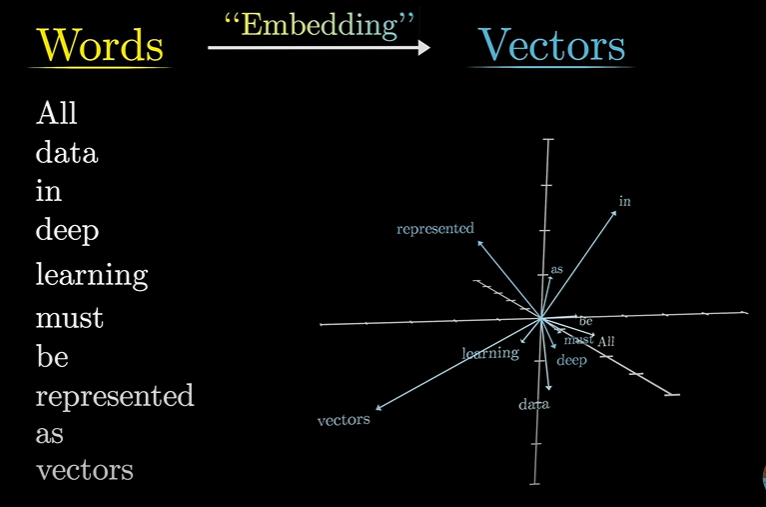


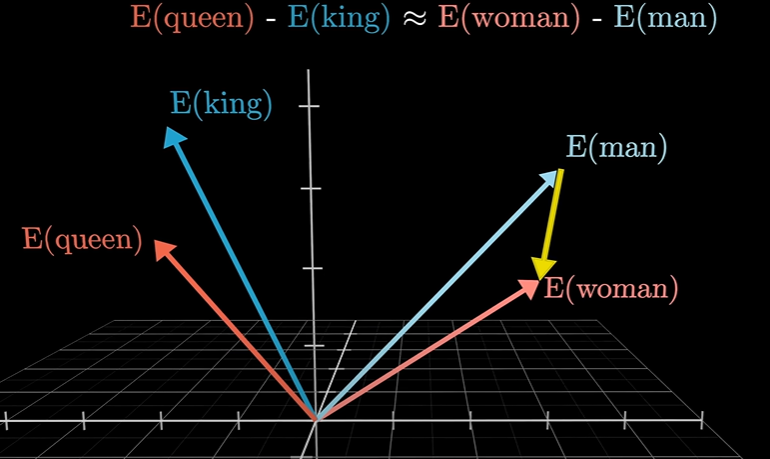
* These matrices are divided into categories:
* Now we will break this down until we list down all the matrices in GPT-3

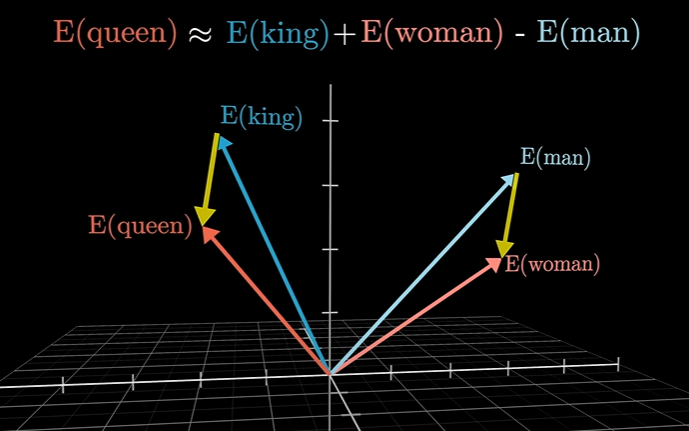


WORD EMBEDDING:

* Model has an inbuild dictionary of all possible words, around 50k.
* The first matrix we’ll encounter is the Embedding matrix:
* It has columns for each of the possible word, it decides which vector each word turns into , the vector will start random and will be trained to attain nearly accurate vector.
* This process is called embedding a word:

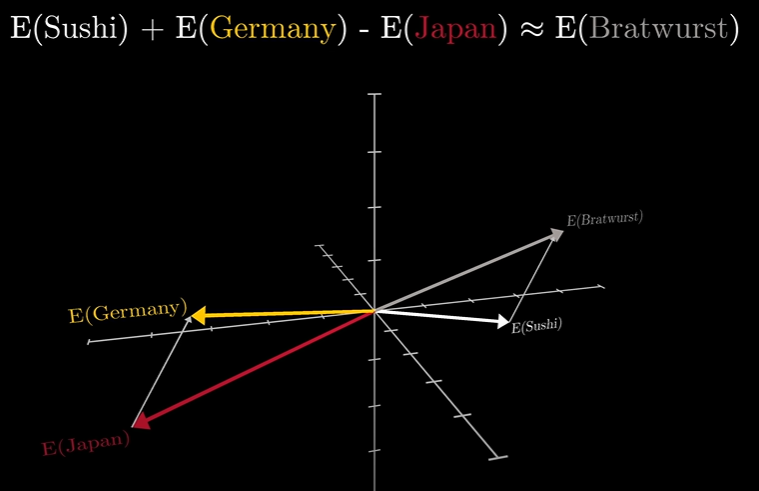


* Let’s assume you want to search the word for a female monarch, you can use this equation to derive or find the most similar words:
* and:



* As the man and woman ~= king and queen we can figure out how to find a particular word related to any of the 4 words.
* Rather than king and queen, we can also relate:
  + Father and mother
  + Niece and nephew
  + Uncle and Aunt
* This same approach can be used to find many related words,few more examples would be:



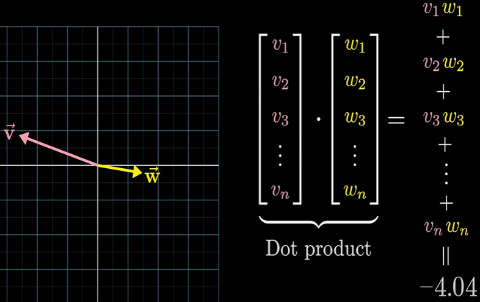
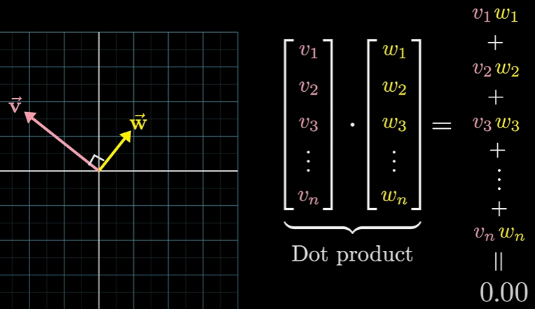
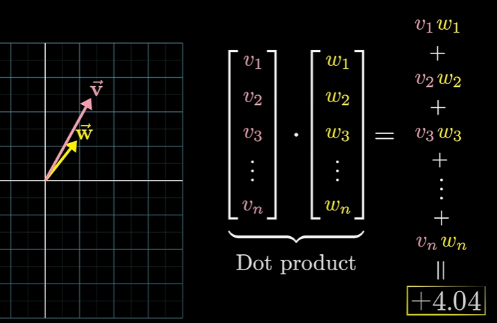


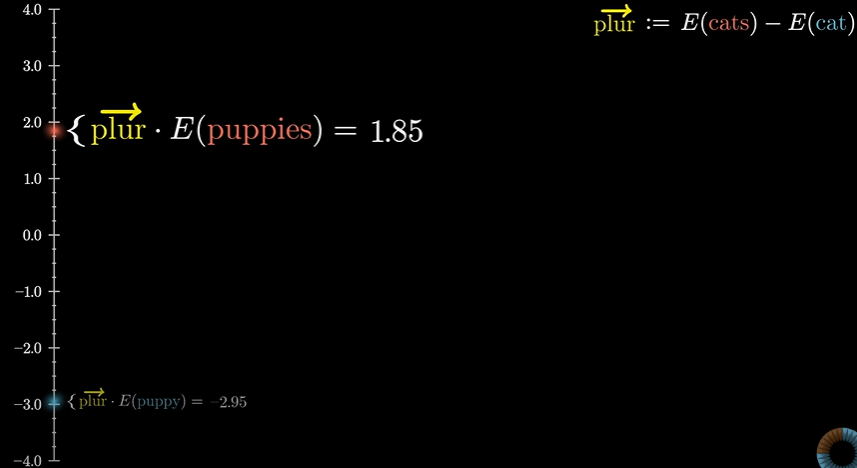
* Our next goal would be understanding how vector values work, all about dot product and when is the value -ve or +ve or simply 0.

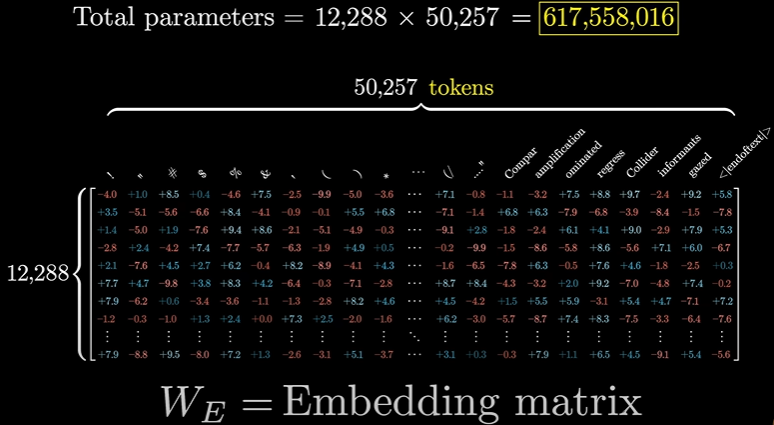
Day 2

* Basically, we’ll learn how the vectors work or the value assigned to them work.
* If both the vectors are at the same quadrant or side:
  + The dot product is +ve
* If the vectors are perpendicular to each other:
  + The dot product is 0
* If the vectors point to the opposite direction:
  + The dot product is -ve

As shown below:



* A fun fact:
  + All the plural words or nouns are placed higher than the singular ones
  + And all the singular words or nouns are placed lower than plural and the dot product is usually -ve.
* Let say the difference between cat and cats gives us a plurality direction in the space.
* And hence we can see the difference in the singular and plural word or noun.
* This also applies to numbers, where 1’s dot product is -ve and all the numbers from the 2 and beyond are in the increasing order.



* Now recall the gpt’s total weight or parameters, which was around 175B
* From 175B, W­E = Embedding matrix’s total parameters are around 617 million.

(Look out the total weight table of gpt)

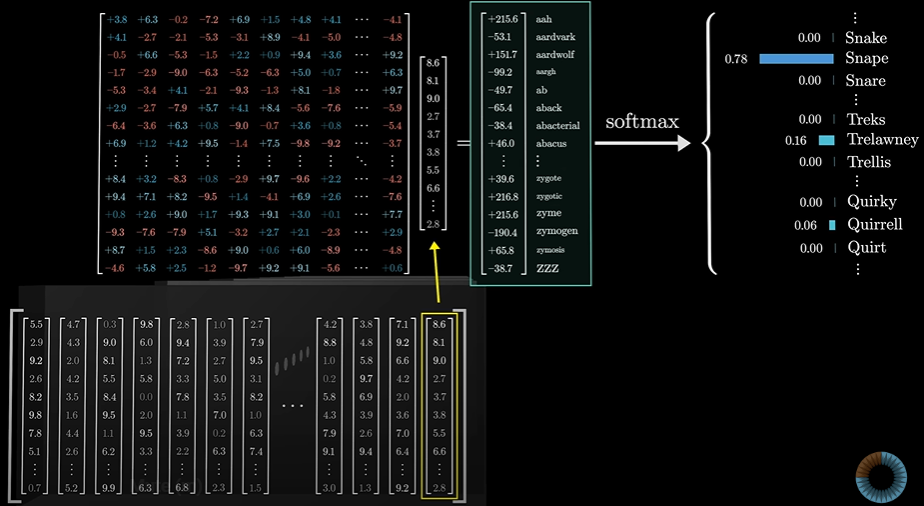
* Now we explored the WE, next let’s zoom into unembedding.
* Remember, the desired output is a probability distribution over all tokens that might come next:

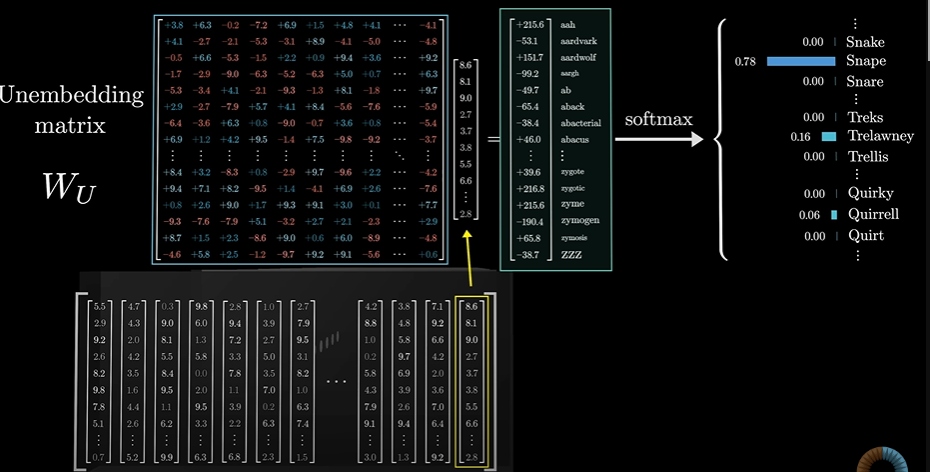
Diagram shown below:

* As the last word in the above phrase is Professor, and first words are Harry Potter, and also includes least favourite teacher, then the well-trained network which has build up Harry Potter knowledge will assign Snape,

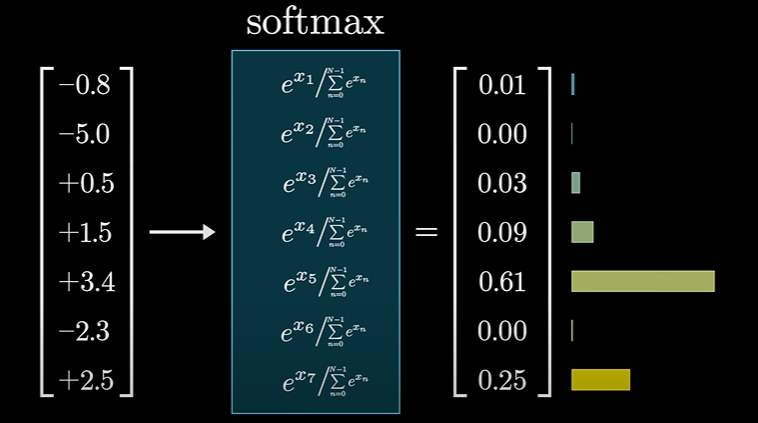
This assignation has 2 steps:

1. Use another matrix that maps the very last vector in that context to a list of 50k values, one for each token in the vocabulary, then there is a function that normalizes this into a probability distribution and it’s called **softmax**

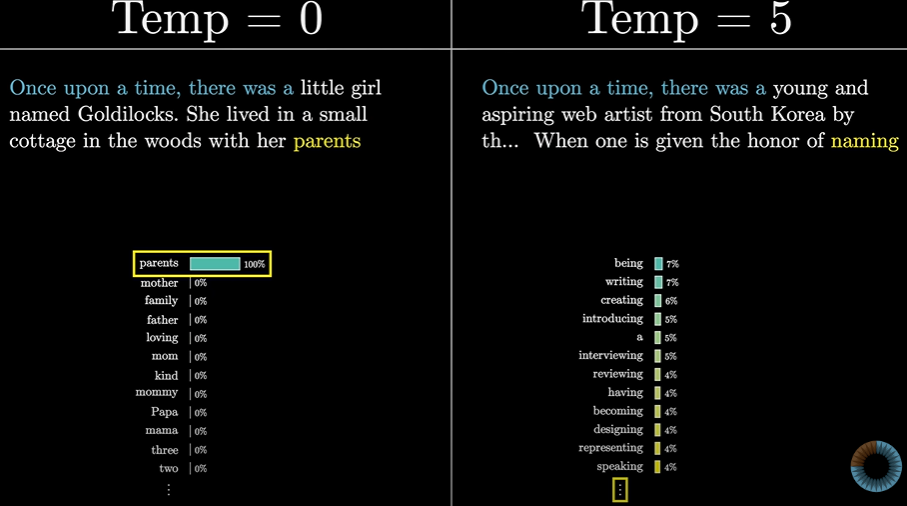
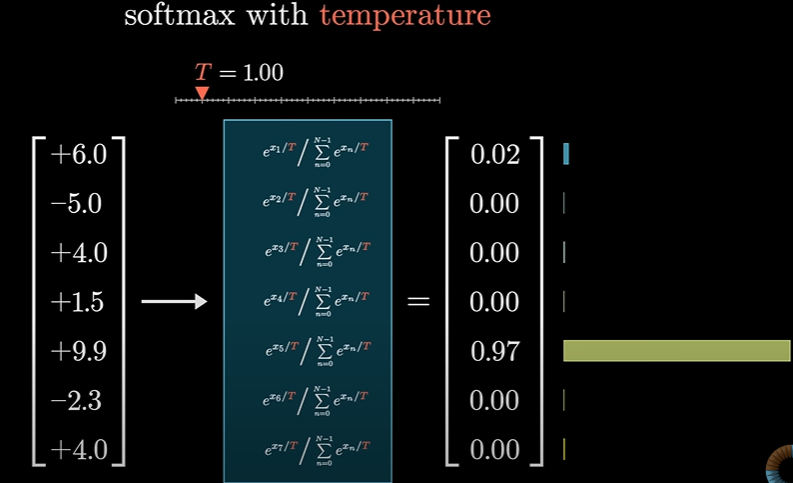
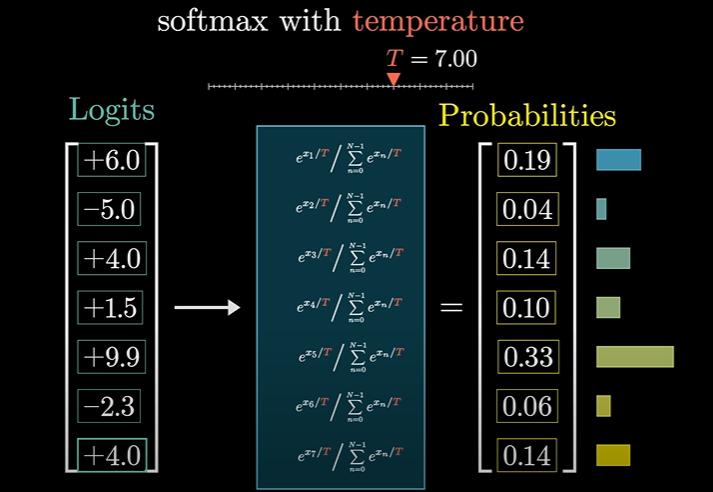


* But if you noticed, it might be a bit weird how it uses only the last embedding and ignore all the 1000’s of the vectors.
* In the training process it turns out to be much more efficient if you use each one of those vectors in the final layer to simultaneously make a prediction for what would come immediately after it.
* This particular matric is called unembedding matrix WU.
* Even WU learns during training process
* Now WU also contains around 617 million weights, updating the total weights table of gpt:

SOFTMAX:

* We know the probability distribution table each value should be between 0 and 1 and they always add up to 1, or else the distribution is considered wrong.
* But in the outputs of the deep learning, matrix does not add up to 1, but they are -ve or greater than 1.
* Softmax is the standard way to turn an arbitrary list of numbers into a valid distribution in such a way that the largest values end up closest to 1 and the smaller values end up closer to 0

Softmax with temperature: (called as temperature bcs it vaguely represents temp in thermodynamics)

* To add up extra spice the model, T (temperature) is used
* Denominator T is used in the softmax formula:
* When T is larger, you give more weight to the lower values, meaning the distribution is a little bit more uniform
* And if T is smaller, then the bigger values will dominate more aggressively.
* And if T is set to 0, all the weight to goes to maximum value.
* When is T is 0-1 the gpt will give us more sensible auto completion story and complete the phrase is more sensible manner
* When T is larger than 2, gpt will give us bogus auto completion which will make no sense at all.
* Usually, the model won’t let us choose T value greater than 2, bcs then it will give very vague words and sentences.
* The input is called Logits

END OF LLM.