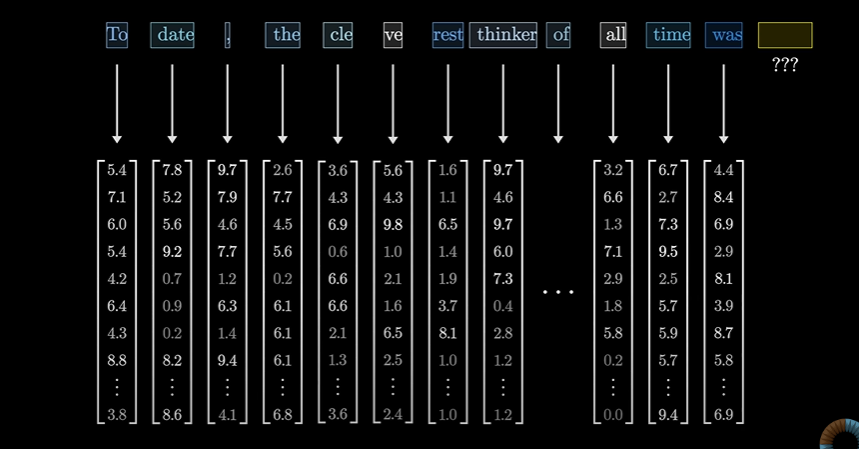
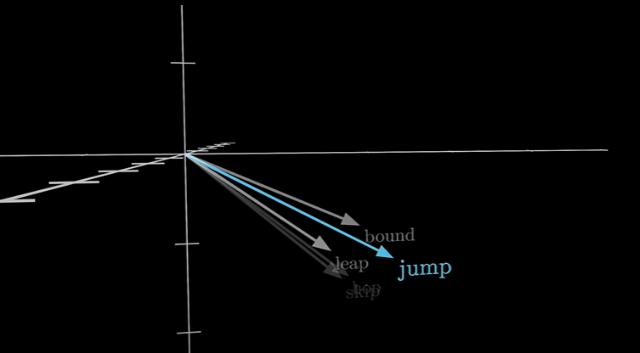
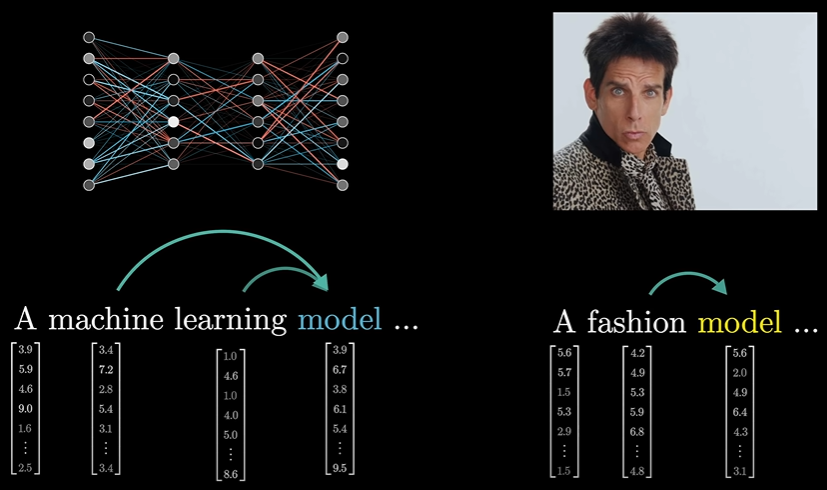
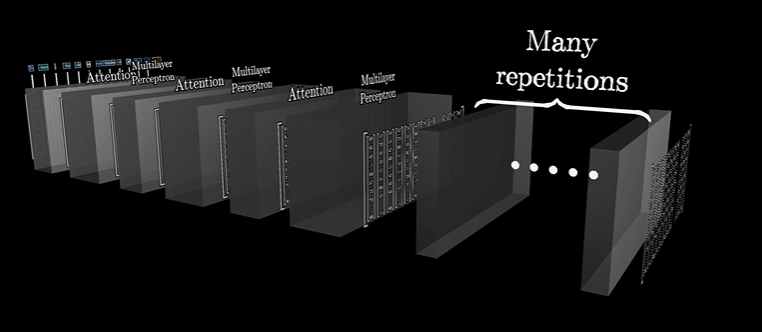
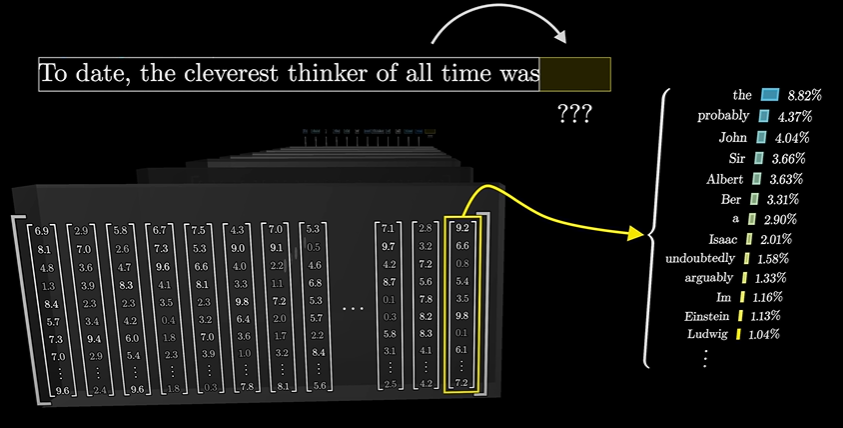
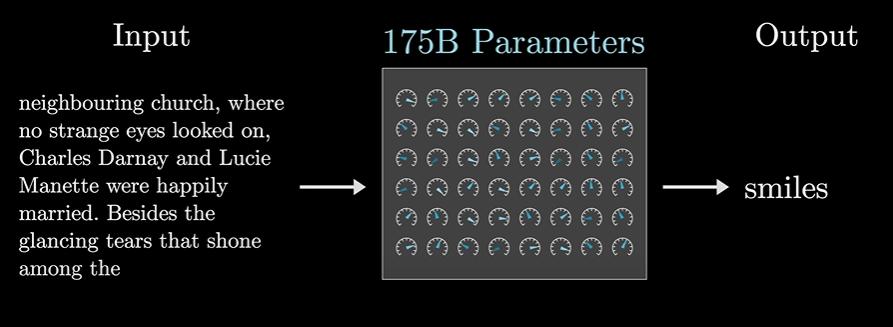
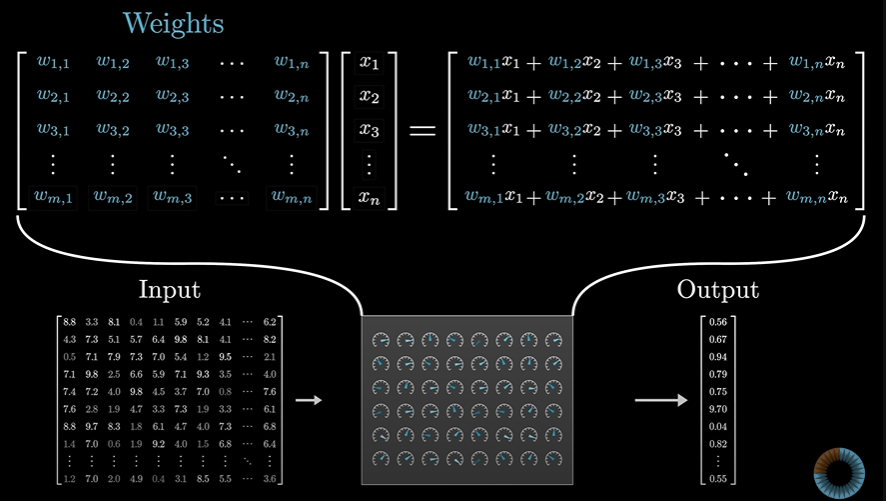
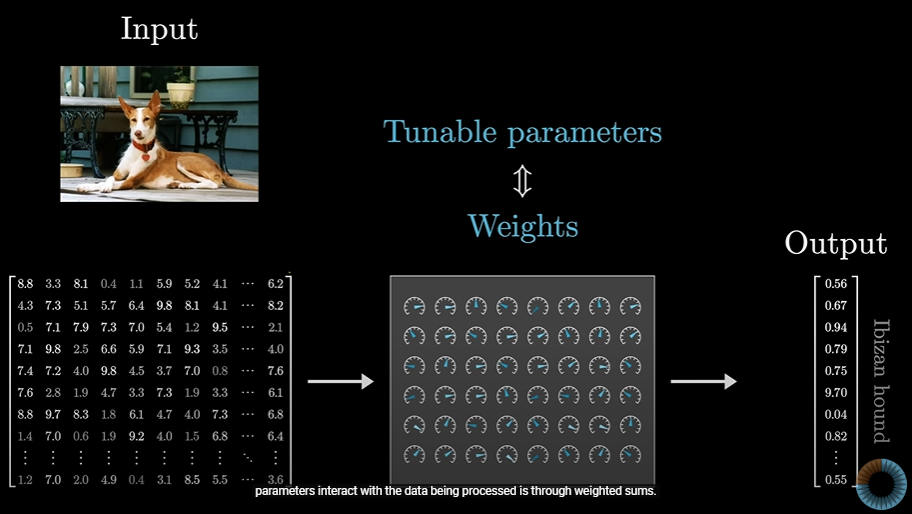
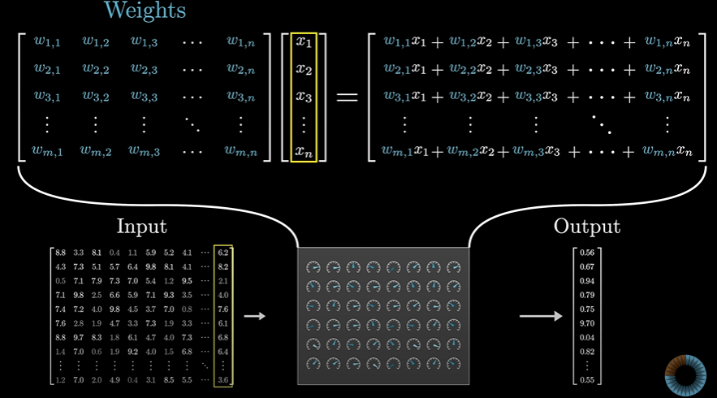
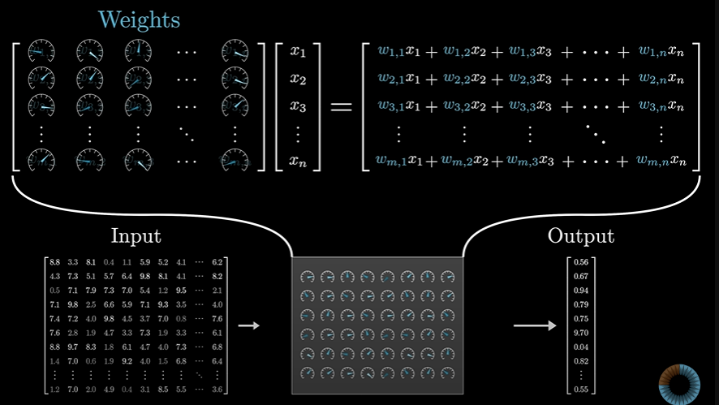
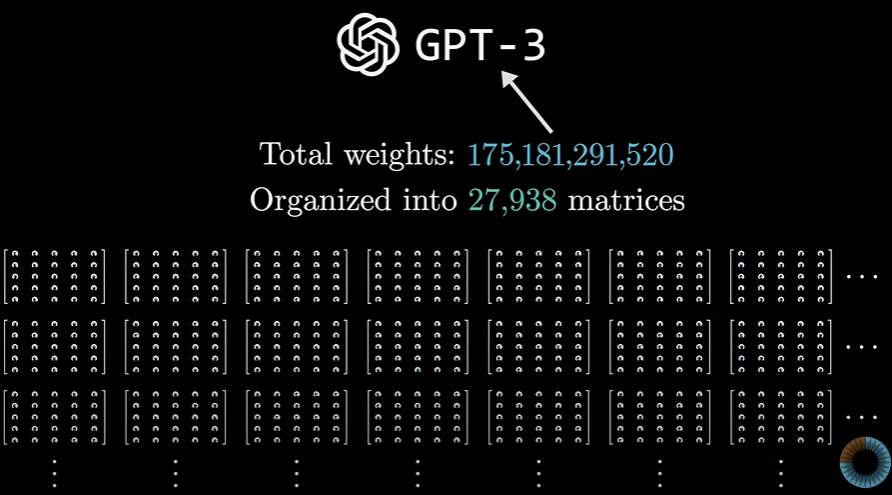
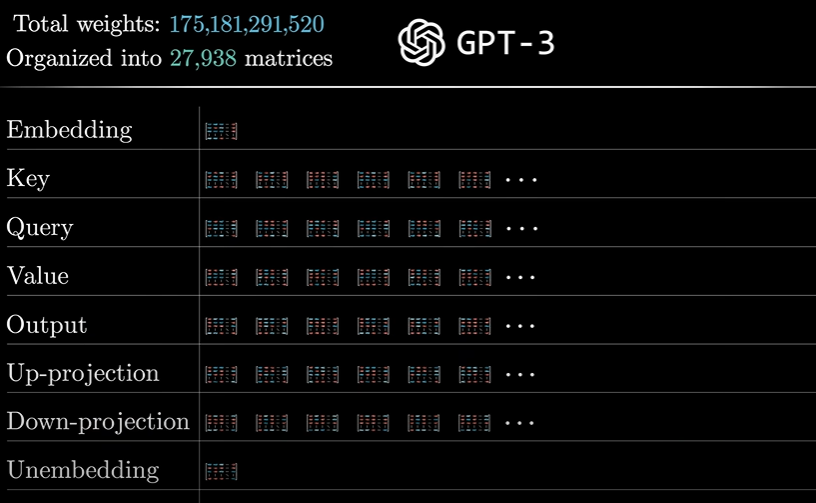
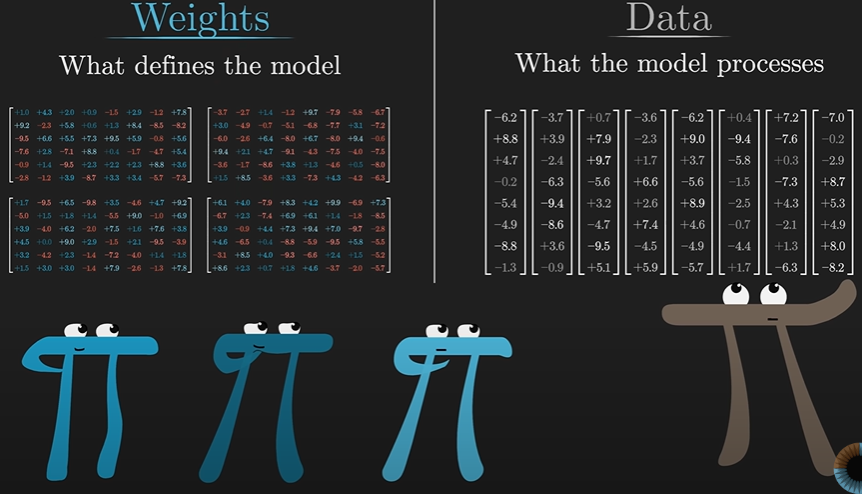
day 1

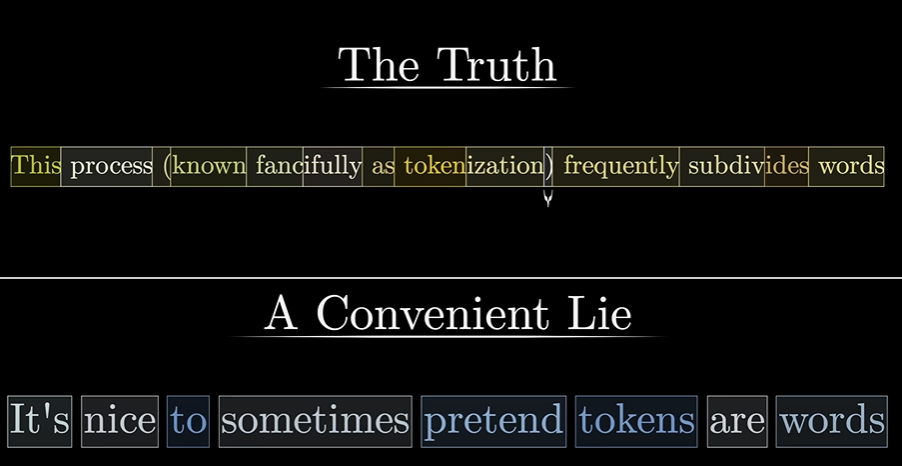
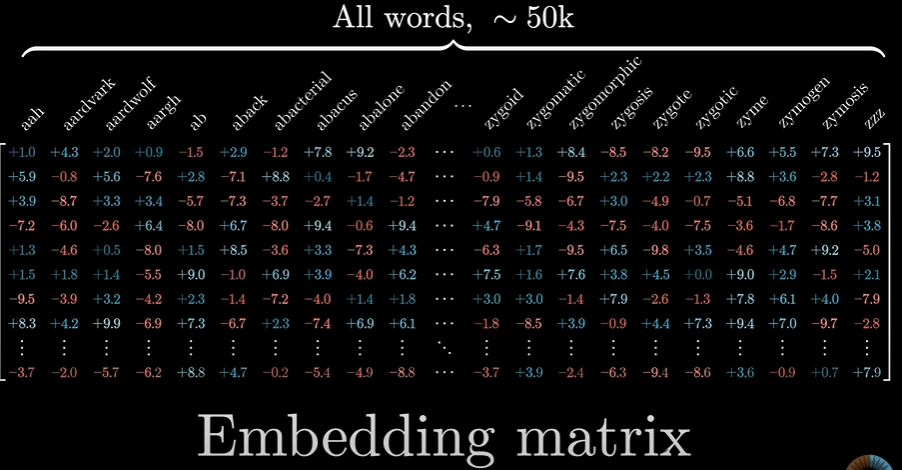
* I'll be trying to make a LLM(large language models) aggregator
* Let's see how much deep I get into this project.
* (llm works according to how we train the language, e.g., human intervention or RLHF(Reinforcement learning with human feedback))
* Providing a single interface to talk to any of the multiple 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)

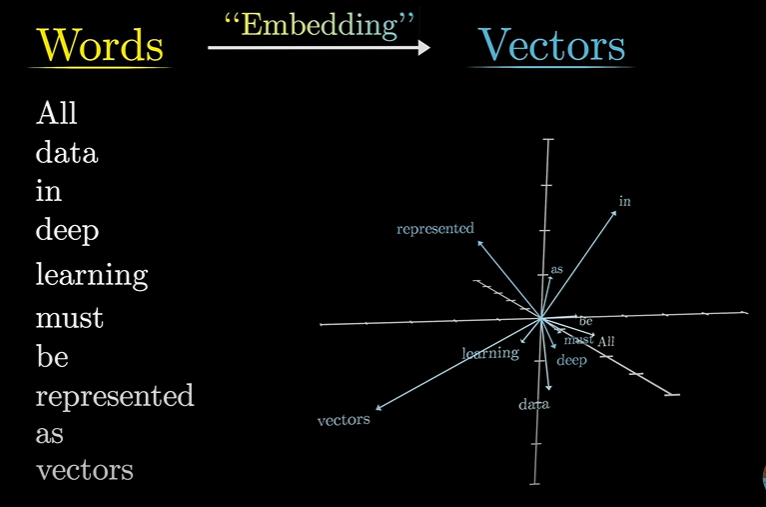


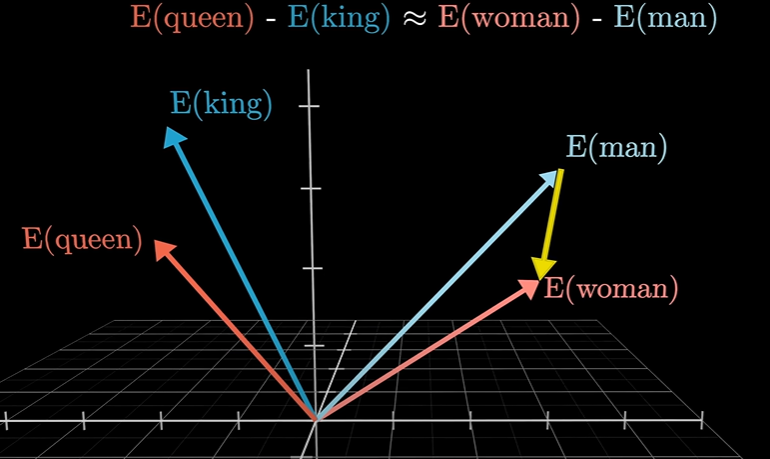
* These matrices are divided into categories:
* Now we will break this down until we list down all the matrices in GPT-3

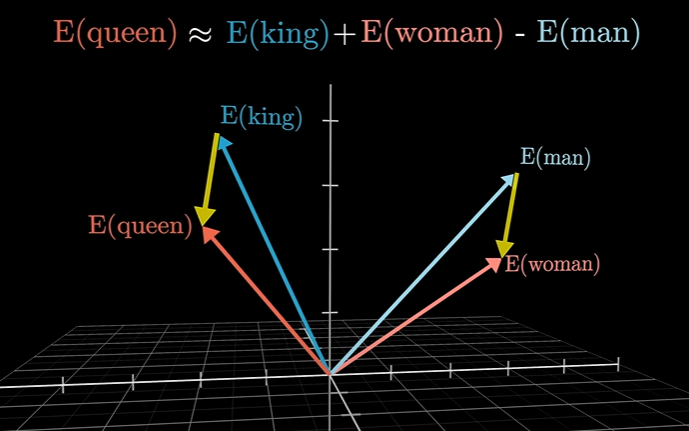


WORD EMBEDDING:

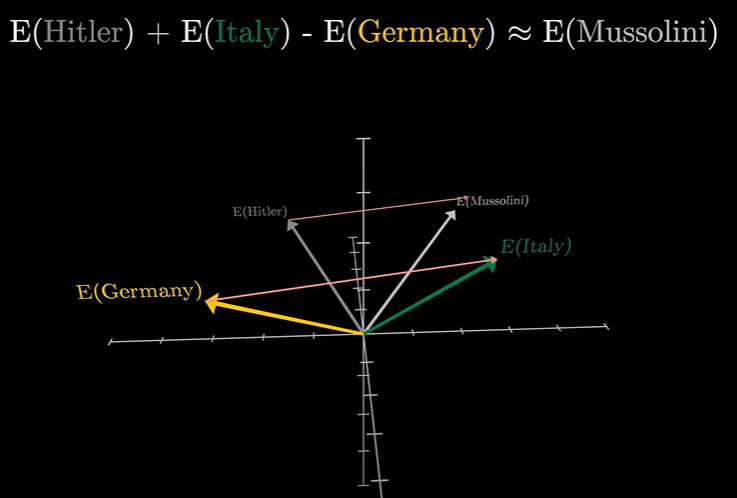
* Model has an inbuild dictionary of all possible words, around 50k.
* The first matrix we’ll encounter is the Embedding matrix:
* It has columns for each of the possible word, it decides which vector each word turns into , the vector will start random and will be trained to attain nearly accurate vector.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

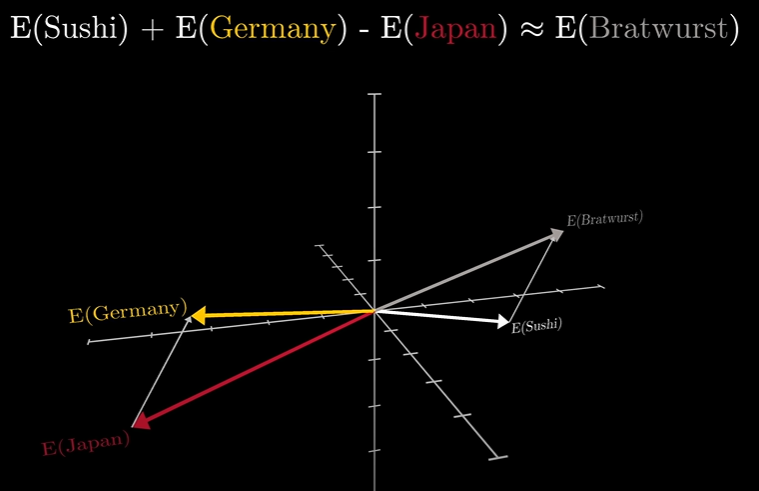


* Let’s assume you want to search the word for a female monarch, you can use this equation to derive or find the most similar words:
* and:



* 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.