UNITED STATES MILITARY ACADEMY

PRELIMINARY WRITTEN PRODUCT

MA389: INDIV STUDY IN MATHEMATICS

SECTION N/A

COL JAMIE BLUMAN

BY

CADET IRELAND MCCAULEY '26, CO F2

WEST POINT, NEW YORK

17 SEPTEMBER 2024

_____ I CERTIFY THAT I HAVE COMPLETELY DOCUMENTED ALL SOURCES THAT I USED TO COMPLETE THIS ASSIGNMENT AND THAT I ACKNOWLEDGED ALL ASSISTANCE I RECEIVED IN THE COMPLETION OF THIS ASSIGNMENT.

MCCAULEY.IRELAND.CALLULA-EVE.1611155456 Digitally signed by MCCAULEY.IRELAND.CALLULA-EVE.1611155456 Date: 2024.09.17 21:07:32 -04'00'

_____ I CERTIFY THAT I DID NOT USE ANY SOURCES OR RECEIVE ANY ASSISTANCE REQUIRING DOCUMENTATION WHILE COMPLETING THIS ASSIGNMENT.

SIGNATURE: MCCAULEY.IRELAND.CALLULA-EVE.1611155456 Digitally signed by MCCAULEY.IRELAND.CALLULA-EVE.1611155456 Date: 2024.09.17 21:08:03 -04'00' DATE: 17SEP2024

MA389 Preliminary Written Product

To gain a strong foundation on neural networks, I watched and took notes on the 3Blue1Brown videos covering all topics related to neural networks, such as what they are, how they learn and gradient descents, backpropagation, and applications of how these models work on a larger scale such as ChatGPT. First, I learned the basic structure of a neural network; there are neurons that are activated on a scale of zero to one. These neuron activations in one layer will influence it in the next layer, and so on an so forth. Next, the component of weight is added in, which is an indication of how a neuron in one layer is correlated with a new neuron in the second layer. Weights are often stored in a weight matrix, and are multiplied to the activation column which stores the neurons. From here, the sigmoid and ReLU functions are two common ways to normalize all of the weighted sums between zero and one. Depending on how one wants the neurons to light up, a second component can be added to the product called bias. This is a constant tells you how high the weighted sum needs to be before the neurons start getting meaningfully active.

Neural networks learn through training data, and by trying to minimize the cost function. At the lowest level, this is done by adding the squares of the differences between each of the "wrong" choices and the value you wanted the computer to output. Using the negative gradient vector we can see which weights need to adjust, by how much, and in which direction. To obtain these gradients, it is very computationally detailed and a slow process. A stochastic gradient descent is obtained by computing the gradient descent for each randomly assigned mini group of data, rather than the entire dataset. While it is not exact, it is a approximate gradient. The calculus that goes into the backpropagation of gradients involves the activations of each neuron which in itself depends on the weight, bias, and previous neuron activation. From there, you

want to know how sensitive is the cost function to small changes in the weights, where the chain rule is applied. You repeat this same ideology but with bias and previous neuron activation

Lastly, some higher level usages of neural networks can be used on natural language processing models like ChatGPT. The 3Blue1Brown videos went into great depth of how this specific model takes in text, and produces a prediction for what should come next using a these networks along with probability distributions and vast amounts of data to train with.

Annex A:

3Blue1Brown PowerPoint Notes (see attached PDF)

Neural Networks

Ireland McCauley

Structure

Brief overview of the structure of neural networks.

What is a neural network?

- Neurons
  - A function that take in all the outputs from the previous layer and spits out a number between 0 and 1
  - “activation”
  - Hidden layers
    - Activations in one layer influence it in the next layer
    - Broken down into smaller components
      - Curved edges, long/straight edges may activate certain numbers
  - Take activations from first layer and add in a weighted component
    - weight is an indication of how its neuron in the first layer is correlated with new neuron in the second layer

What is a neural network?

- Sigmoid and ReLU function
    - Bias for inactivity
        - Bias tells you how high the weighted sum needs to be before the neuron starts getting meaningfully active

- Learning
    - Fine tuning the weights and bias's

[Weight matrix] [activation column] + [bias column]

$$W\vec{v} + \vec{b}$$

[784 x 16] [16 x 1] + [784 x 1]

$$a^{(1)} = \sigma\left(Wa^{(0)} + b\right)$$

How they learn

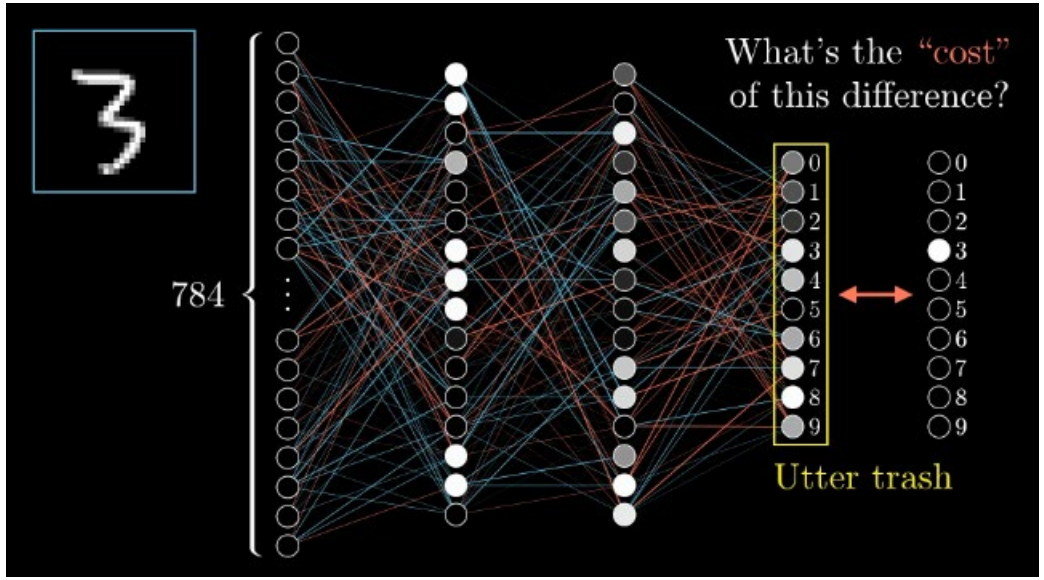
Learning

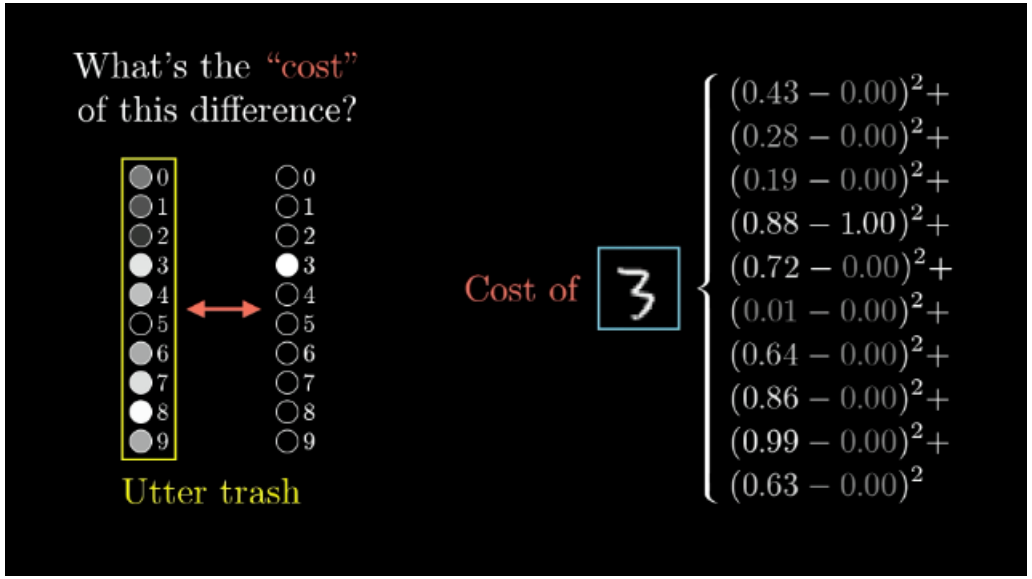
- **Give the algorithm training data with numbers and the answer of what they are so it can practice**
- **See how accurate the model can get with new data**
- **When the machine gives a wrong/bad answer, you can add the squares of the differences between each of the bad choices and the value you wanted the computer to choose**
  - The “cost”
  - The average cost from all training data shows how well the machine performed

Cost function

- input→ weights/ biases
- Output→ the cost (how well/poor it performed)
- Parameters→ lots of training data

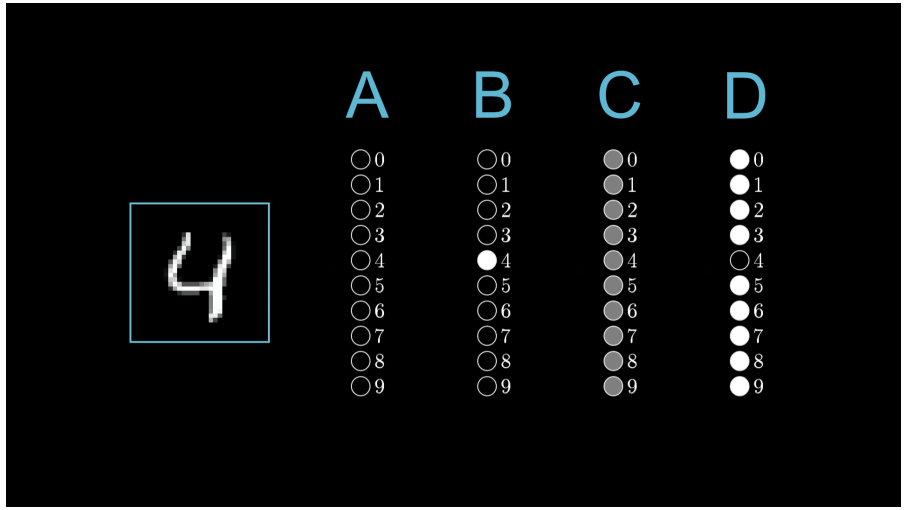
- **Finding local minimum of cost function to tell the computer how to adjust**
- **Using gradients, we can see which weights need to adjust, by how much, and in which direction to make the model more accurate**
- **Learning = minimizing cost function**





https://www.3blue1brown.com/lessons/gradient-descent

Rank the cost associated with each output



https://www.3blue1brown.com/lessons/gradient-descent

D→ C→A→B

B = lowest cost

- Ideal output,

A = smaller cost

- 9/10 output values are correct
- Only the output neuron corresponding to the 4 is wrong

C = higher cost than A

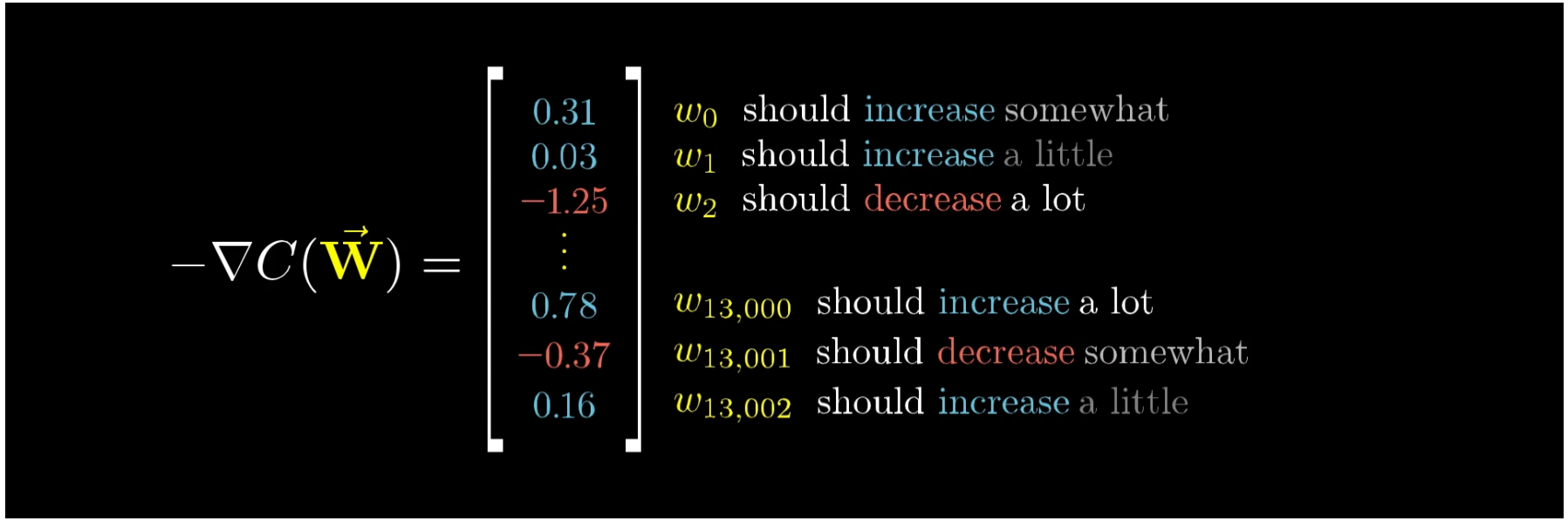
- All output neurons are wrong

D = exact opposite of desired output

- Every neuron is wrong

What does the gradient do?

The negative gradient is a vector direction → tells you what nudges to those 13 thousand numbers would cause the most rapid decrease to the cost function



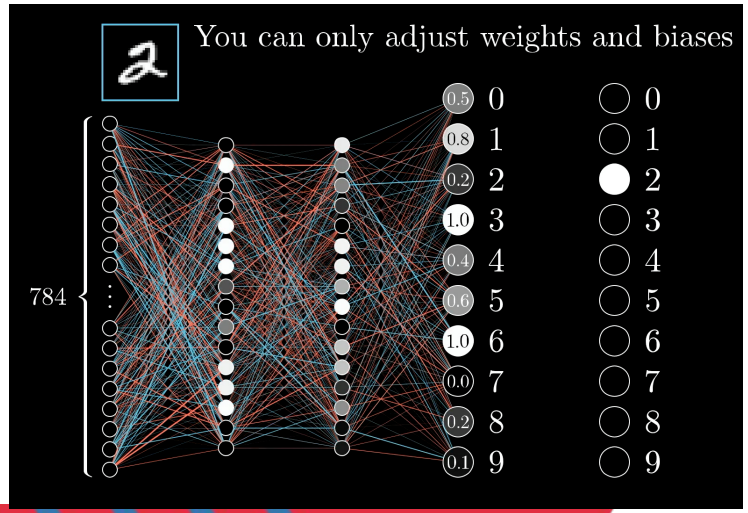
3Blue1Brown - Gradient descent, how neural networks learn

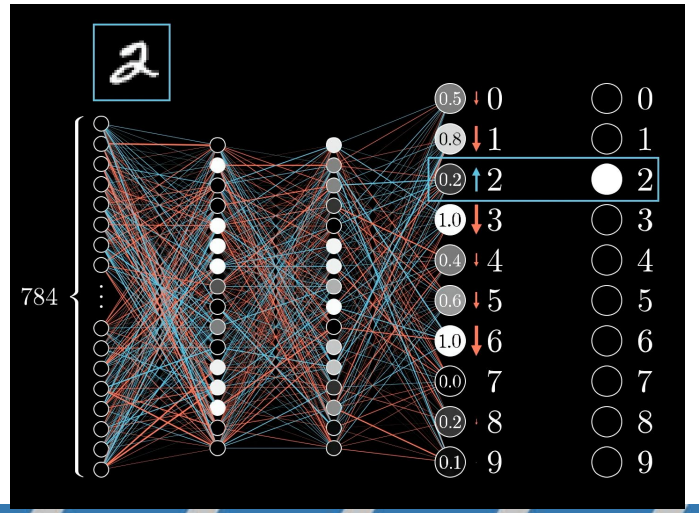
Backpropagation

Gradient

- The negative gradient
  - magnitude of each component of the gradient tells you how sensitive the cost function is to each corresponding weight and bias

- Whichever numbers activation you wish to increase and which other numbers you wish to decrease should be proportional to the amount the model gave you

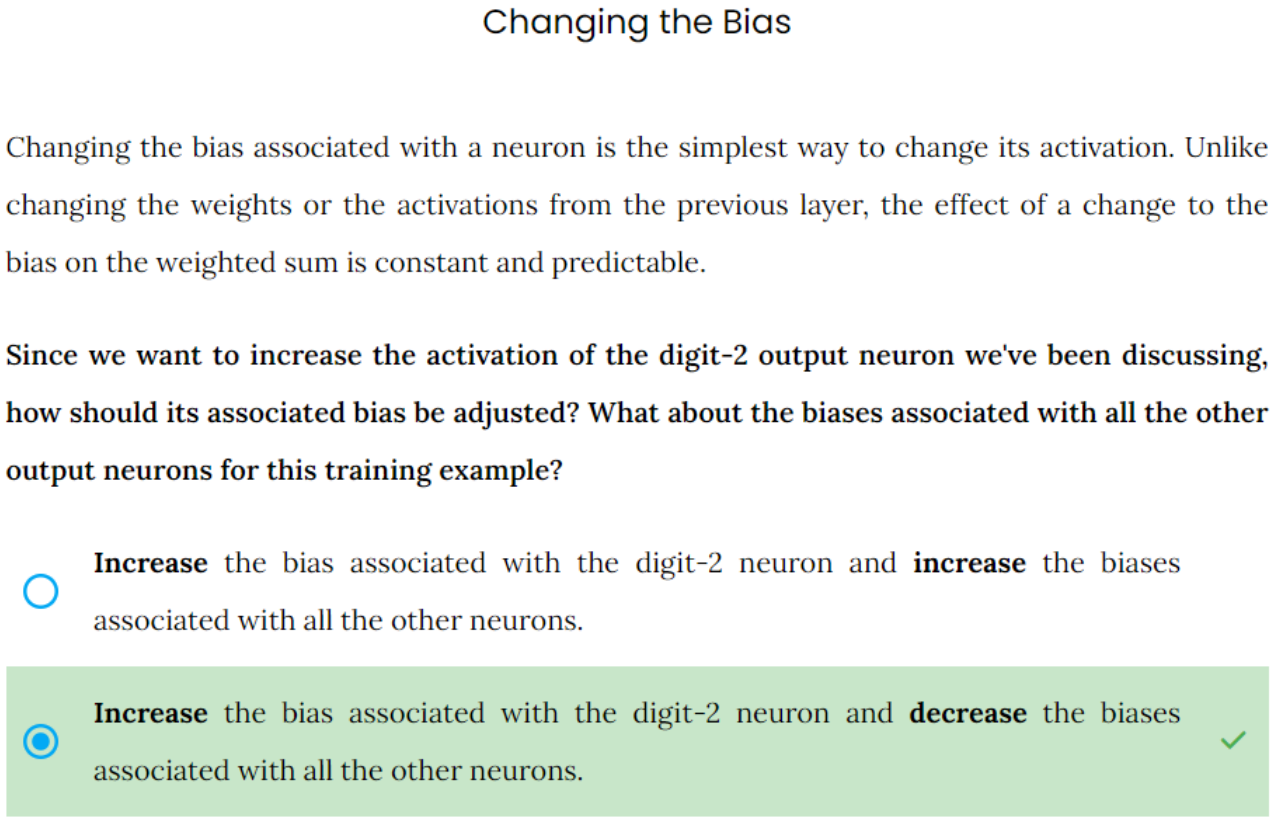




https://www.3blue1brown.com/lessons/backpropagation -

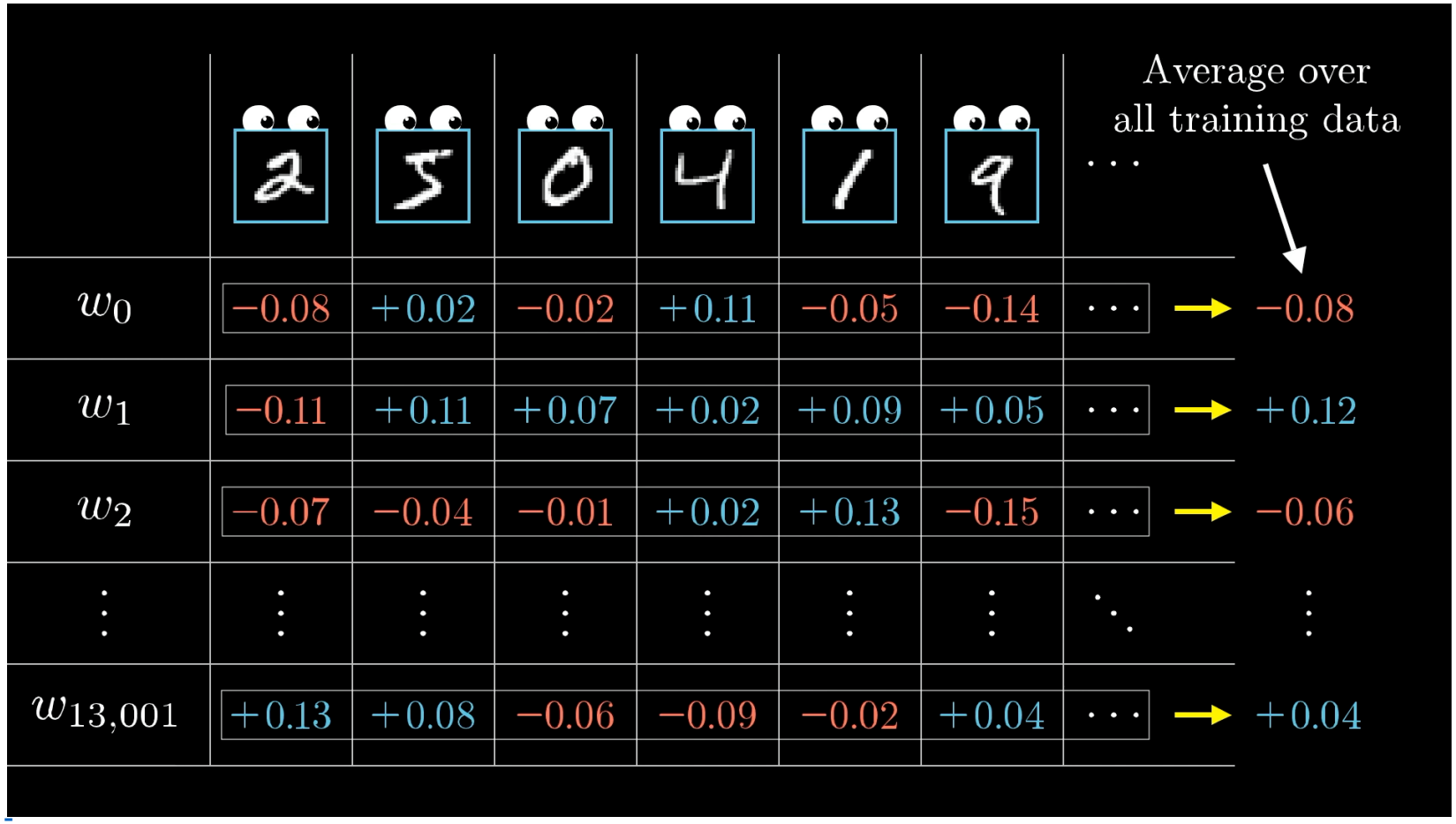
How to increase activation

- Increasing the weights
  - Increasing the weights on the already activated nodes will have a greater effect than changing it on the poorly lit nodes. This is because the activation is multiplied by the weight (so a .9 times weight will have more effect than a .3 times a weight)
- Increase the bias
  - Constant and predictable
- Change the activation
  - Add activation to nodes that thought the number was the correct one, and decrease the ones that did not
  - Most effective = changes in accordance with weights



Calculating Gradient

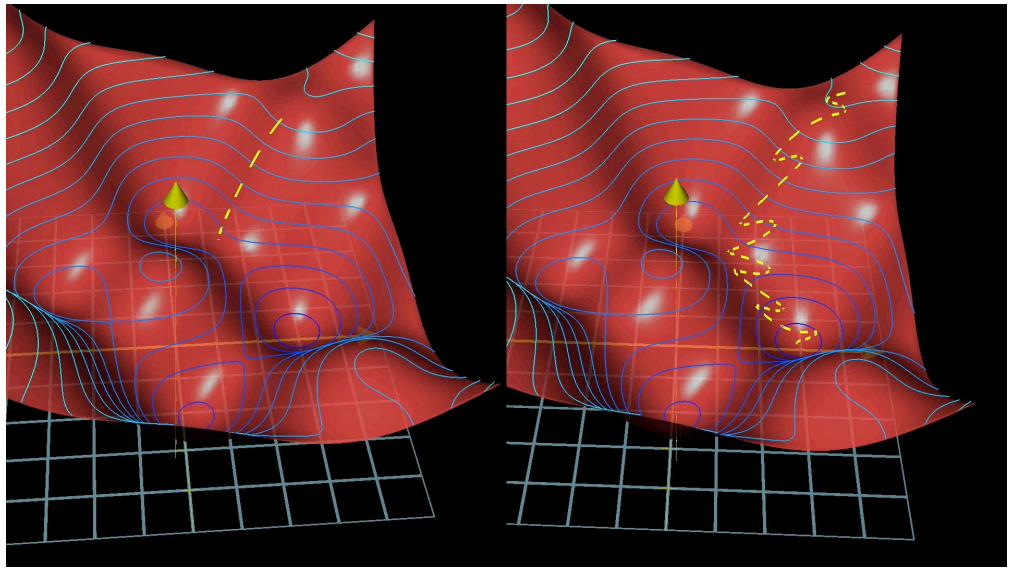
- You repeat this for every other training example, recording how each of them would change the weights and bias
- Take the average of changes
- This is the negative gradient of the cost function
- However, this is extremely computationally slow



https://www.3blue1brown.com/lessons/backpropagation -

Mini Batches and Stochastic Gradient Descent

- Randomly put training data into smaller groups
- Compute gradient descent for each mini batch instead of the entire dataset
- Not exact, but good representation
- Approximate gradient



https://www.3blue1brown.com/lessons/backpropagation -

Backpropagation Calculus

Tree of neuron activation

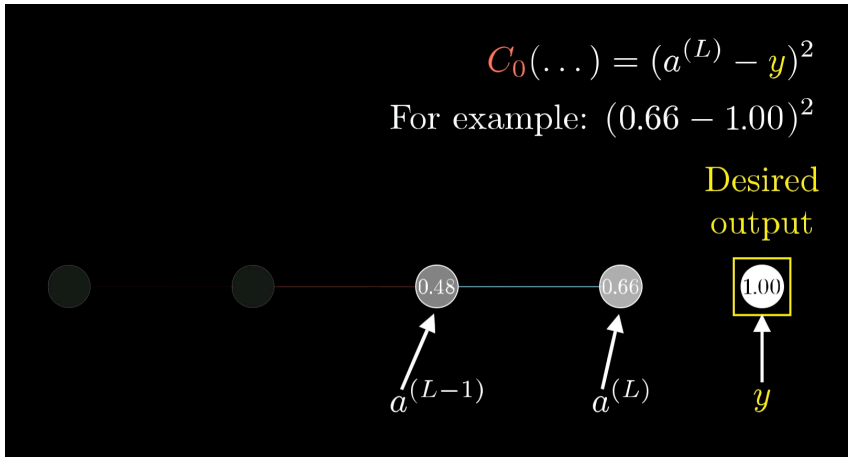
Activation is determined by weight, bias and previous neuron activation

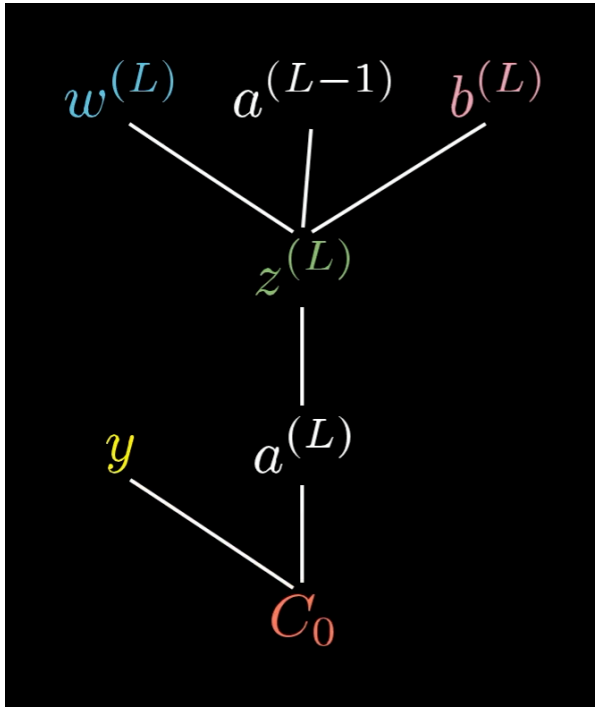
$a(L)=\sigma(w(L)a(L-1)+b(L))$

To make it easier to understand, we can call the inner components, z, and rewrite it.

$$z(L)=w(L)a(L-1)+b(L)$$

$$A(L) = \sigma(z(L))$$





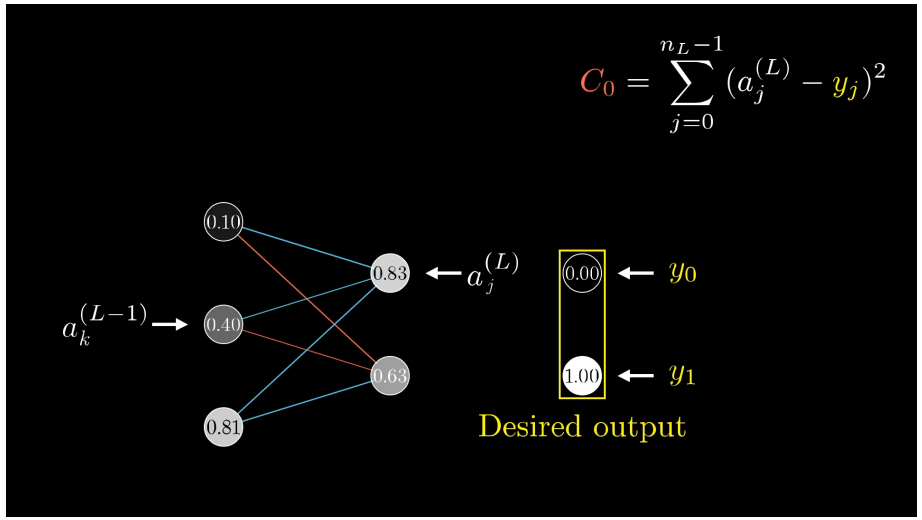
https://www.3blue1brown.com/lessons/backpropagation-calculus -

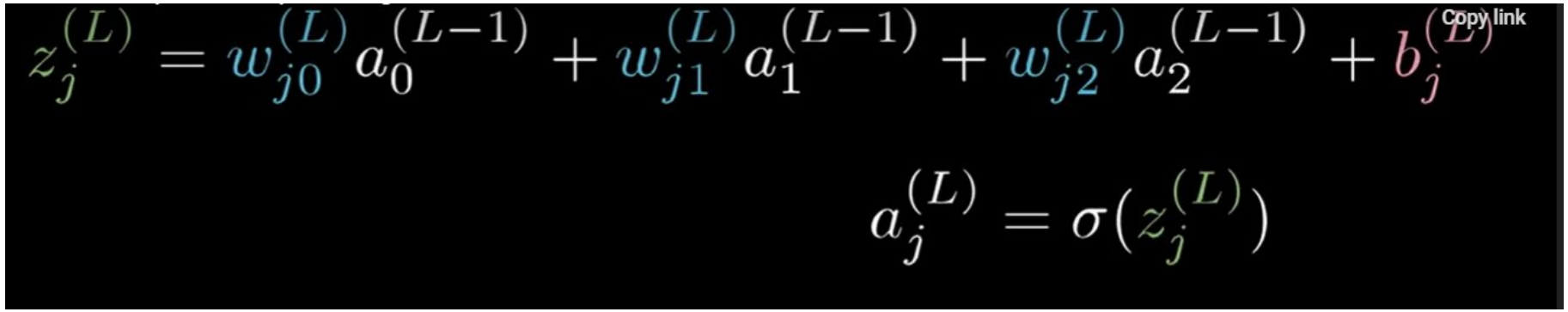
Partial Derivatives

- What is the derivative of C, with respect of w (L)
- Easily understood as how sensitive is the cost function, C to small changes in the weights
- We use the chain rule to find this
  - This tiny nudge to $w(L)$ causes some nudge to $z(L)$, which in turn causes some change to $a(L)$, which directly influences the cost, $C0$.
  - You repeat these steps but with respect to bias, 1, and previous activation.
    - when the activation in the second-to-last layer is changed, the effect on $z(L)$ will be proportional to the weight.

More Complex Data

- Use subscripts to indicate layer, and which neuron it is
- To find cost, add the sum of squares of the differences between the last activation layers and the desired result →→→
- Also add subscripts and keep track of weight associated with those neurons





https://www.3blue1brown.com/lessons/backpropagation-calculus -

More Complex Data

- Because this previous-layer neuron influences the cost along multiple different paths, the derivative of the cost with respect to this neuron, the sensitivity of C0 to changes, and involves adding up multiple different chain rule expressions corresponding to each path of influence.

- Sum over layer L

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}}$$

https://www.3blue1brown.com/lessons/backpropagation-calculus -

ChatGPT

Transformers and Deep Learning

Transformers

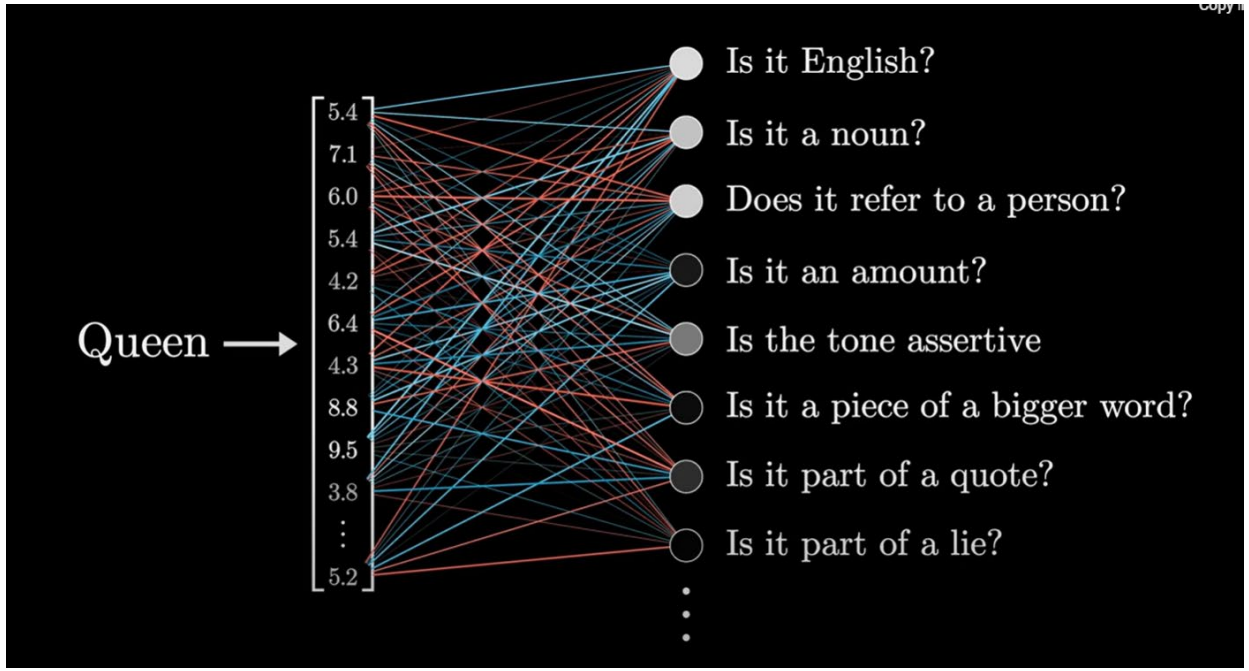
Specific neural network

Models trained to take in text, and produce a prediction for what comes next in the passage

- Probability distribution
- Repeated prediction and sampling with vast amounts of data it trains on

Tokens

- Tokens can be a word, a piece of an image or chunk of a recording
    - Tokens are then associated with a list of vectors which encodes their “meaning”
    - If you were to graph these vectors, words with similar meanings would be near each other in that space
- They then go into an attention block where all of the words and vectors can talk to each other and update their values
- Different meanings of words within the sentence
- Then they all go through the same operation in parallel called the multilayer perceptron (photo)
- The process repeats between attention blocks and perceptron blocks until the last step where the probability distribution is created on all possible tokens to get the word with the highest chance of coming next.



https://www.3blue1brown.com/lessons/gpt

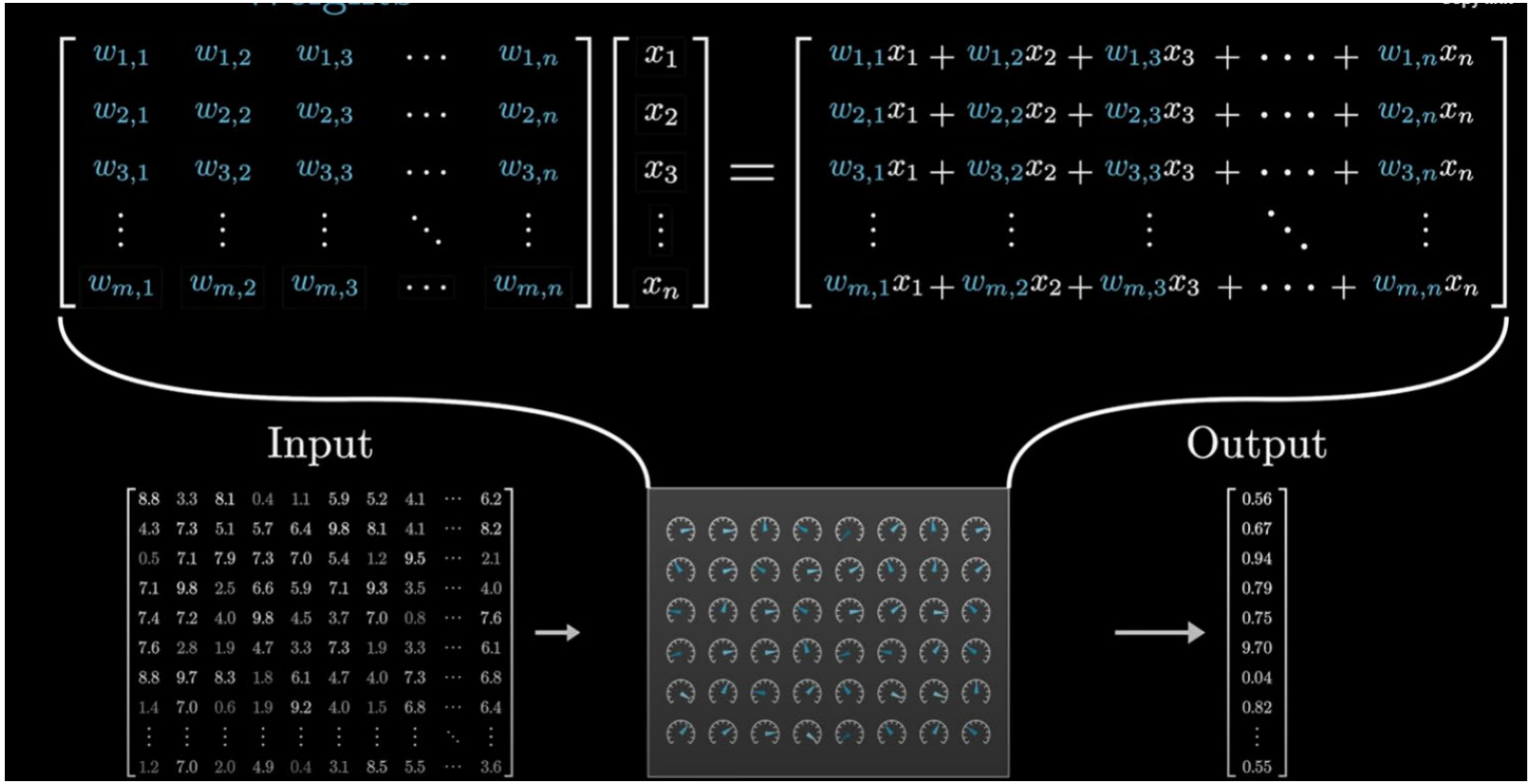
Machine Learning

- Uses DATA
- Flexible structure with lots of dials and ability to tweak and adapt

https://www.3blue1brown.com/lessons/gpt

Deep learning

- Backpropagation
- Input must be formatted as an array of real numbers
- Then transformed into many layers
- Final layer is output (ex. Probability distribution of what next word is)
- The adaptable parameters are known as “weights”
- Usually kept as a matrix, and the output is the weighted sum

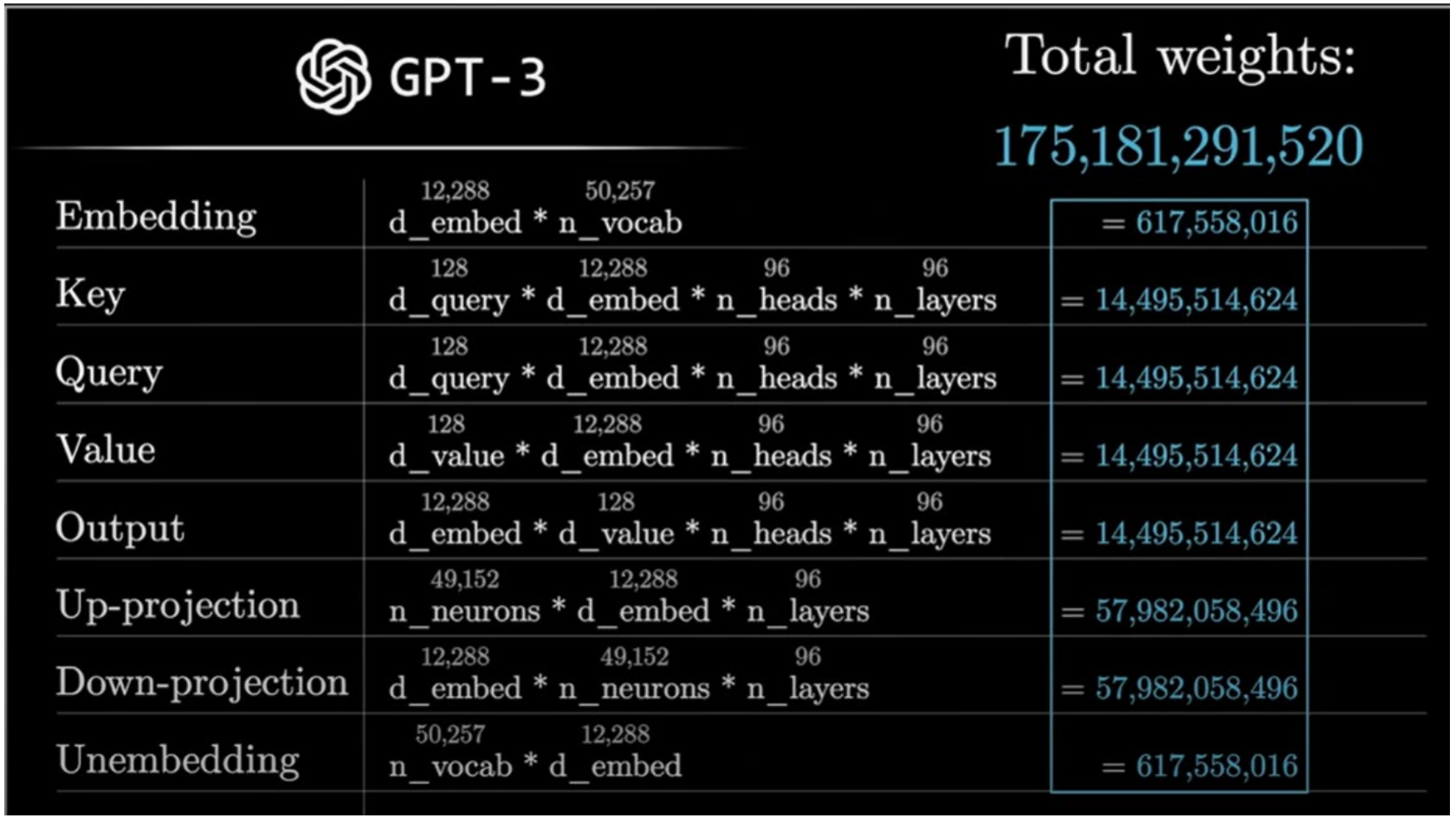


Matrices

- 175 billion weights
- Organized into 27k matrices
  - Embedding, key, query, value, output, up projection, down projection, unembedding

Weights define the model

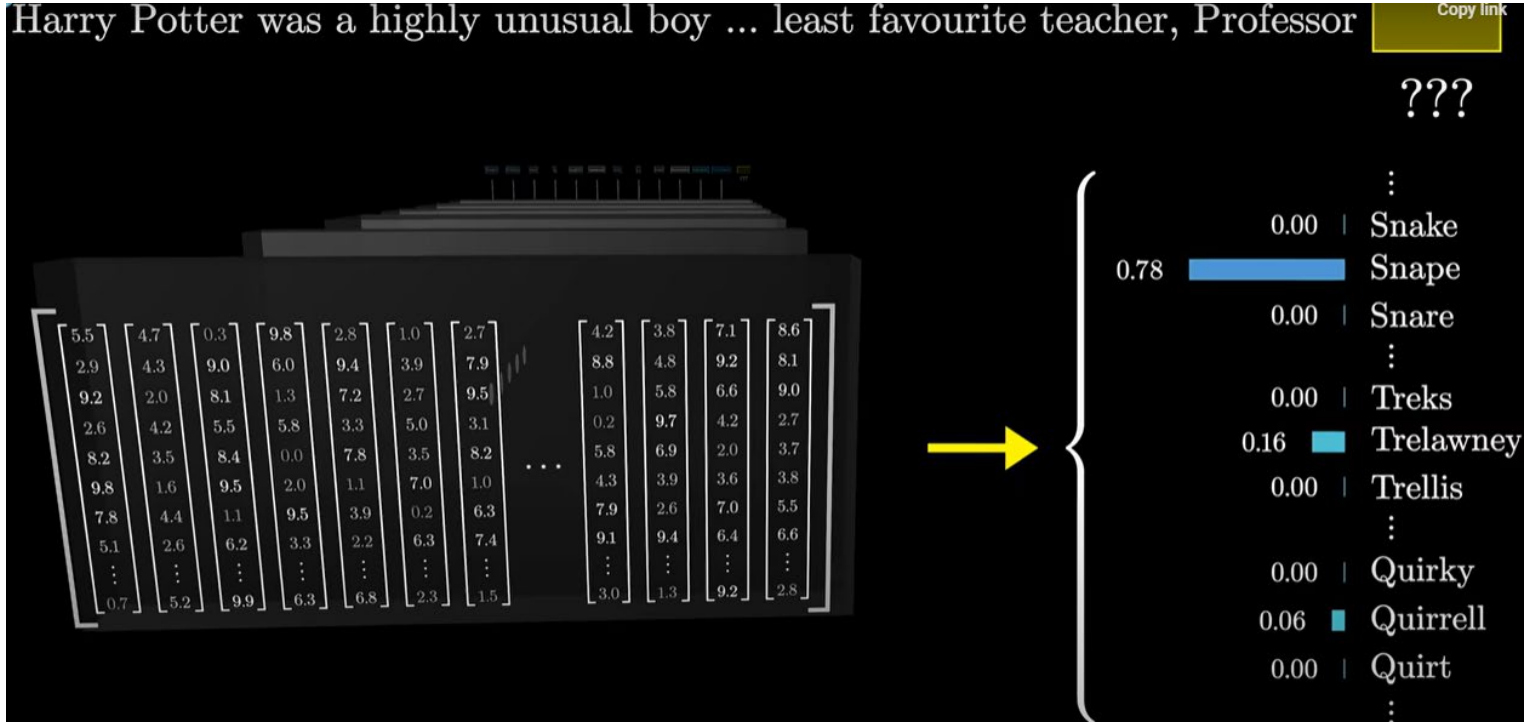
Data is what the model processes



https://www.3blue1brown.com/lessons/gpt

Embedding

- Turns words into vectors
- Lots of weights for each word
- Large dimensional space
- Capacity to soak in context
- Meaning is changed by surrounding words
- Account for about 617k weights



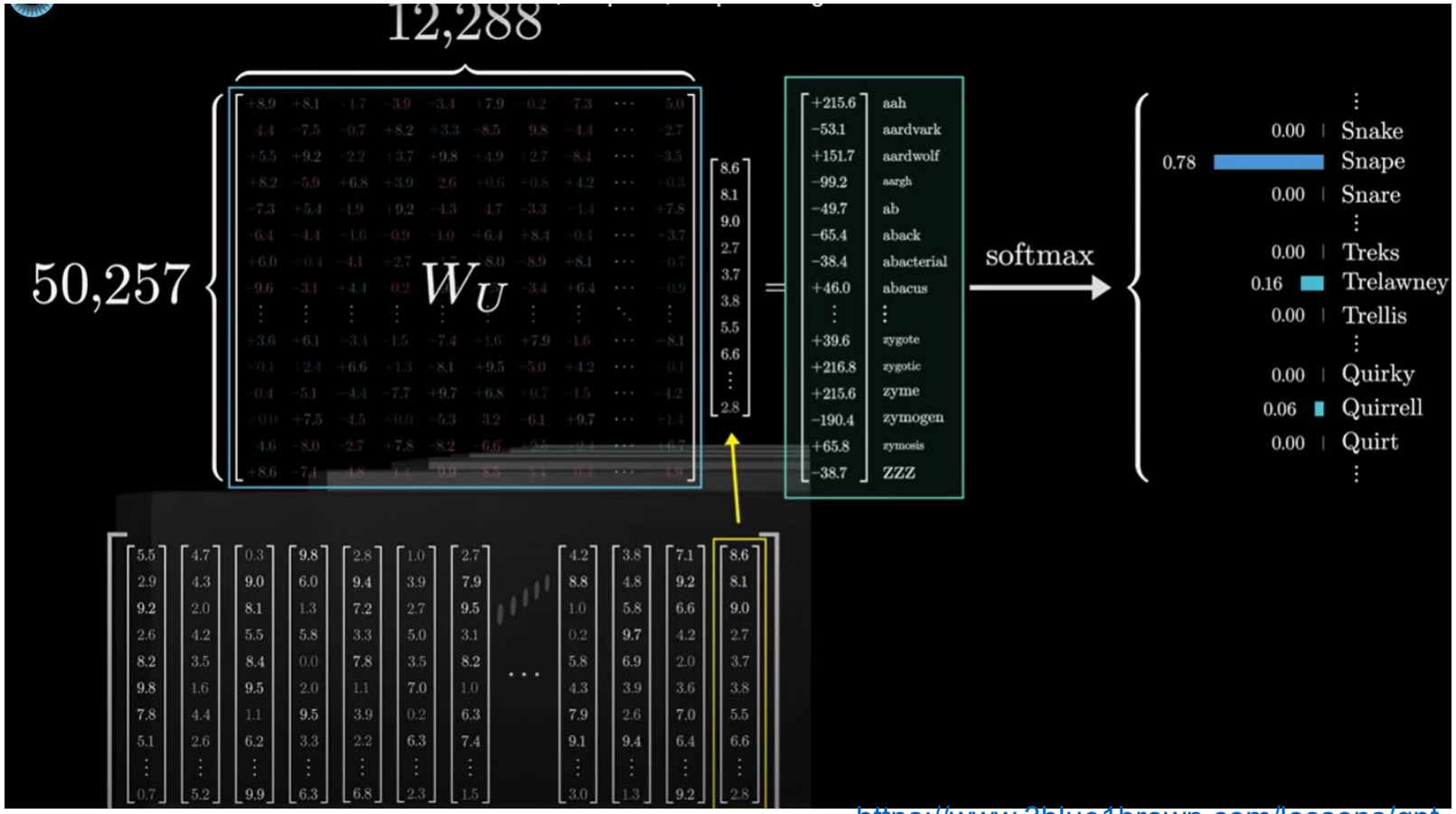
https://www.3blue1brown.com/lessons/gpt

Dot product

- Positive when they point in similar direction
- Zero if perpendicular
- Negative when opposite directions

Unembedding and probability distribution

- Maps last output vector to another weight matrix
  - Each row for a word in the vocabulary (50k)
  - Same number of columns as the embedding dimension (the output)
- Then SoftMax normalizes this into a probability distribution
- Accounts for another 617k weights



https://www.3blue1brown.com/lessons/gpt

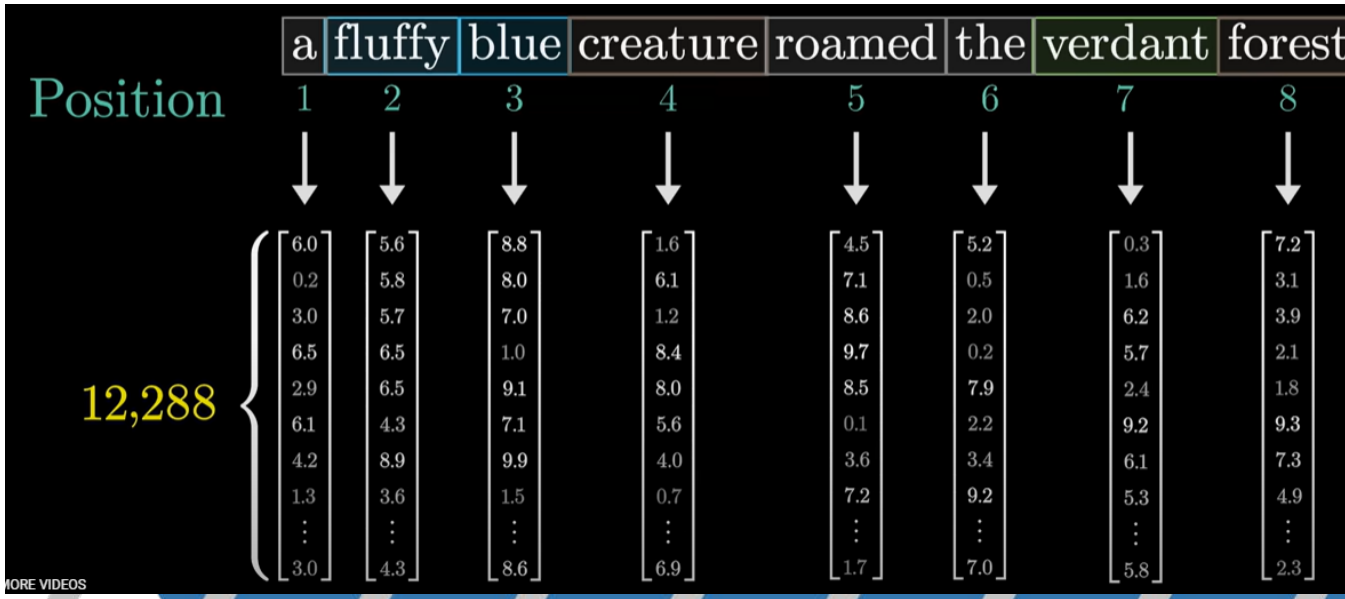
Attention

The Attention Matrix

- The same word with different meaning will have the same original embedding vector
- It is not till the attention step that it will adjust
- Attention calculates what you need to add to the original vector

- Move information encoded in one vector to another

- The first step is the encoding for the word, as is with the position



https://www.3blue1brown.com/lessons/attention

- Imagine each noun asking the adjectives in front of it how to encode it in another vector
  - Called query vector
  - Smaller dimension than embedding vector
  - W q times the embedding
- Key matrices
  - Maps adjectives that are closely aligned with the query
  - Compute dot product between each key query to see if they align (large positive number)
  - Score of how relevant each word is to updating the meaning of every other word

Tell me more

	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 $\vec{K}_1$	+0.7	−83.7	−24.7	−27.8	−5.2	−89.3	−45.2	−36.1
fluffy $\vec{E}_2$ W_K $\vec{K}_2$	−73.4	+2.9	−5.4	+93.0	−48.2	−87.3	−49.7	+7.8
blue $\vec{E}_3$ W_K $\vec{K}_3$	−53.4	−5.7	+1.8	+93.4	−55.6	−56.0	−26.1	−62.1
creature $\vec{E}_4$ W_K $\vec{K}_4$	−21.5	−29.7	−56.1	+4.9	−32.4	−92.3	−9.5	−28.1
roamed $\vec{E}_5$ W_K $\vec{K}_5$	−20.1	−40.9	−87.8	−55.4	+0.6	−64.7	−96.7	−18.9
the $\vec{E}_6$ W_K $\vec{K}_6$	−87.9	−33.3	−22.6	−31.4	+5.5	+0.6	−4.6	−96.8

https://www.3blue1brown.com/lessons/attention

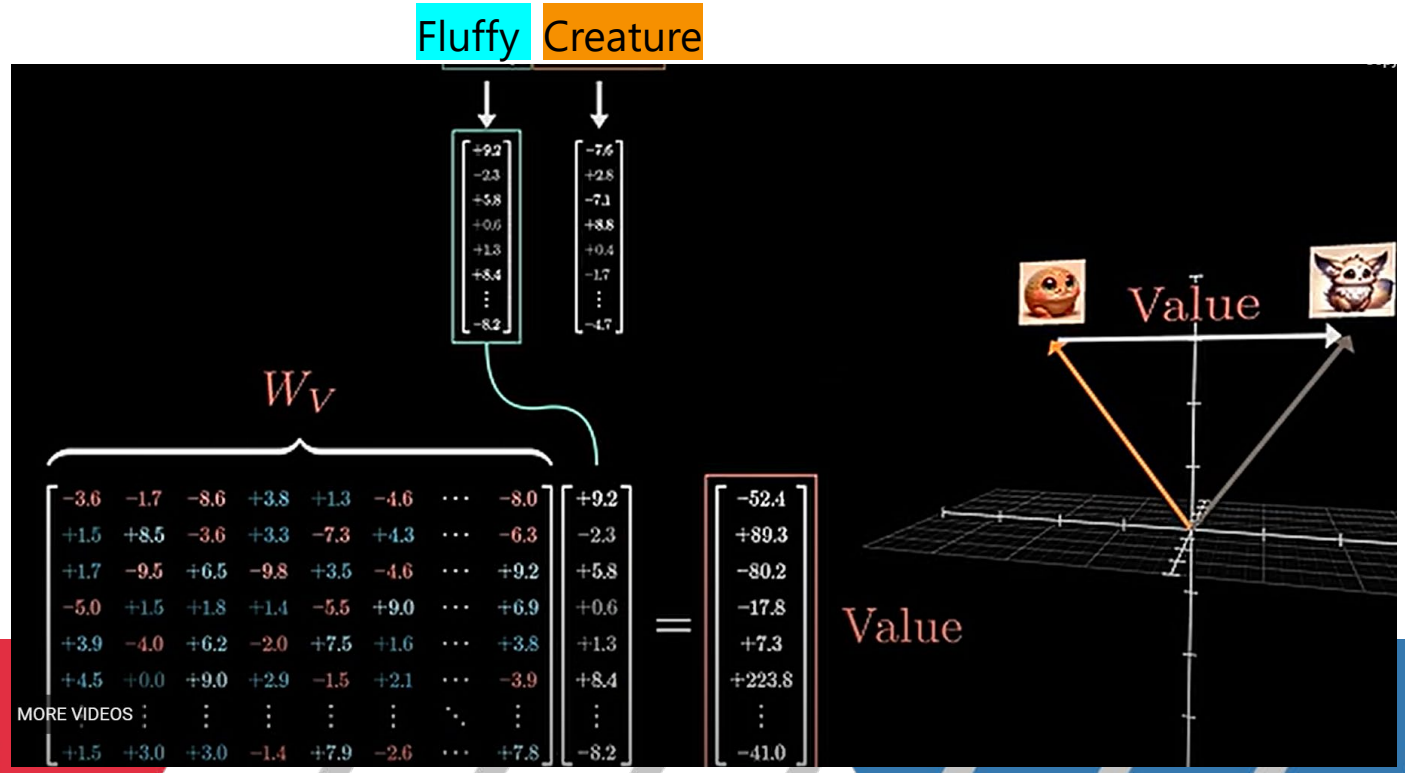
Embedding space → 12,288 dimensional
Query/key space→ 128 dimensional

- The next step is taking a SoftMax from each column to normalize the values and get a probability function
- This gives a weight to how relevant the word on the left is to the corresponding word/value at the top
- This is called an attention pattern
- You never want later words to influence earlier words
  - You can adjust this by setting their values to zero
  - Before SoftMax, set it to negative infinity
  - “Masking”

- Next, the embeddings need to be updated
- For example, you need the word “fluffy” to somehow cause a change to “creature” that moves it to a different part of the embedding space that more specifically encodes a fluffy creature

Ways of changing embedding

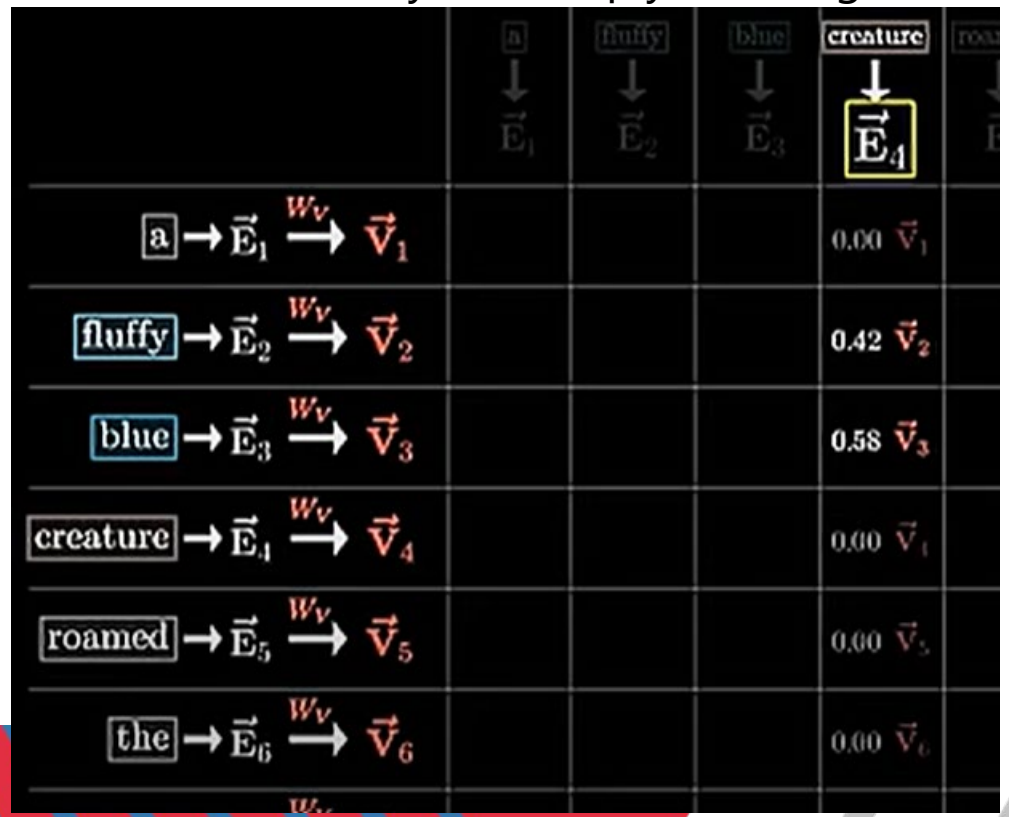
- Use a 3rd matrix called the value weight matrix
- You multiply this matrix to the first word, “fluffy”
- The result is the value vector, and then you add this to the second word, “creature”



https://www.3blue1brown.com/lessons/attention

One head of attention

- You multiply each weight value matrix to each embedding word
- Then that created a value vector
- From there you multiply the weight associated with each word by that value vector



- The way you actually change the embedding of the specific word from here, is by adding together all of the rescaled values in the column, called Δ E, to the original context free embedding of “creature”

https://www.3blue1brown.com/lessons/attention

Query, Key, and Value

Query:

12,288 columns x 128 rows

= 1.5 million added parameters per head

Key:

12,288 columns x 128 rows

= 1.5 million added parameters per head

Values: (linear map)

Value up: [12,288 rows x 128 columns]

Value down: [128 rows x 12,288 columns]

= 1.5 million added parameters each per head

Large Language Model

Multi Layer Perceptron's

MLPs

Assumptions

- Supposed one direction = first name Michael
- Perpendicular direction = last name Jordan
- Another direction = basketball

Each vector goes through operations→ the output is the same dimension as the original→ they get added together as the sum going out

For this example, a vector that encodes First Name = Michael and Last Name = Jordan goes in, and the output is basketball.

Operations

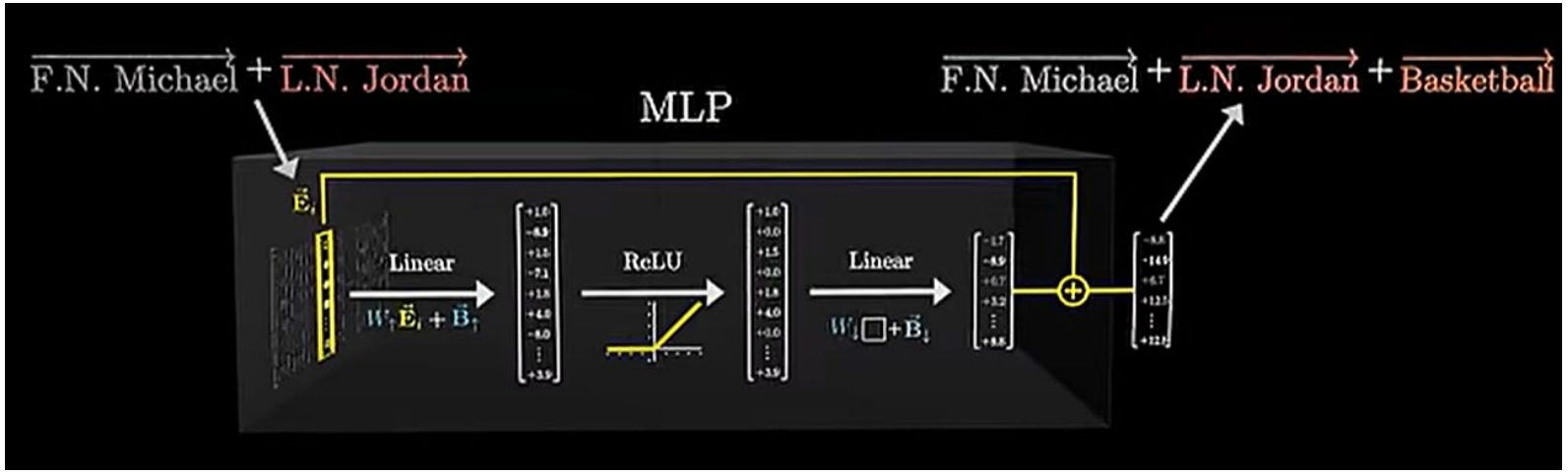
Linear (weight up)

- Multiply vector by a matrix full of parameters (weights) and add in bias
- The number of rows in the weight matrix is 4 x 12,288 (embedding dimension)

ReLU

- Pass vector through non linear function
- Leaves as positive values as is, and makes negative values 0
- Values are the neurons

https://www.3blue1brown.com/lessons/mlp



Linear (weight down)

- Multiply by another matrix and add bias
- This time the output is equal to the dimension space of the embedding (12,288)