Understanding AI

and Its Limits

By: Abd Al Rahman Bassim Hasan

Table of content

Abstract…………...…..…..…..…..…..…..…..…..…..…..…..….…….....…..…..…..4

Introduction…...……....…..…..…..…..…..…..…..…..…..…..…...…….....…..…5

Chapter 1 Definition
- 1.1 AI Definition and types…...……....…..…..…..…..…..…..…..…..…..……6
- 1.2- what is a function and why They're important…...……....…..…..…..……8
- 1.3- Examples of AI…...……....…..…..…..…..…..…..…..…..…..…..……9

Chapter 2 Evolution
- 2.1 Origins…...……....…..…..…..…..…..…..…..…..…..…..……………10
- 2.2 Early days before AI winter…...……....…..…..…..…..…..…..…..…...11
- 2.3 After AI winter…...……....…..…..…..…..…..…..…..…..…..……13
- 2.4 The start of the boom…...……....…..…..…..…..…..…..…..…..……16
- 2.5 Today…...……....…..…..…..…..…..…..…..…..…..……………...19
- 2.6 Quick overview of reasoning LLMs vs LLMs…...……....…..…..……20

Chapter 3 Perception
- 3.1 The Inception………....…..…..…..…..…..…..…..…..…..…..…...21
- 3.2 The AI Early Teen Years…...……....…..…..…..…..…..…..…..…..21
- 3.3 Late Teen Years………....…..…..…..…..…..…..…..…..…..……21
- 3.4 Early Adult Hood…...……....…..…..…..…..…..…..…..…..…..…22
- 3.5 Adult hood………....…..…..…..…..…..…..…..…..…..…..……22
- 3.6 Post adult hood………....…..…..…..…..…..…..…..…..…..…23

Chapter 4 Beneath The Algorithm
- 4.1 Chapter Intro……....…..…..…..…..…..…..…..…..…..…..………24
- 4.2 Flower Problem.....……....…..…..…..…..…..…..…..…..…..……24
- 4.3 Linear....……....…..…..…..…..…..…..…..…..…..…..…………25
- 4.4 Activation Functions...…....…..…..…..…..…..…..…..…..…..…...28
- 4.5 Implementation of a Linear Model flower problem…...…..…..…..…..29
- 4.6 Cost Function…...…..…..…..…..…..…..……………………...30
- 4.7 Slopes…...…..……………………..…..…………………...…32
- 4.9 Final Implementations To Flower Problem…...…..…………………36
- 4.10 Results of Flower Problem …...…..……………………………...39
- 4.11 Terminology…...…..……………………………….…….…...40

Chapter 5 The Limits And How To Break Them (if they can be)
5.1 Hardware Dependency..……………………………………………..………42
5.2 Scaling Issues ..…………………………………………………………...43
5.3 LLMs Resource Consumption..…………………………………………48
5.4 Diminishing Returns ……………………………………………………49

Conclusion……………………………………………………………………...51

Further Development or Research……………..…………………………………...52

Abstract

This beginner-friendly report aims to demystify Artificial Intelligence (AI) for the average person without a technical background. Recognizing the growing popularity of AI and the accompanying misinformation around it, this report will explain how AI works in simple terms, clarify its current capabilities, and address its current limitations. Through hands-on problem-solving and project-building exercises, readers will gain a practical understanding of AI's potential and boundaries., we will also delve into AI's current limitations and see if and how we can overcome them.

Introduction: Decoding the AI Revolution

Artificial Intelligence (AI) has surged in popularity in recent years, and with it has come a wave of misinformation and false promises from some companies. The majority of people still don't know how AI really works and what its current limitations are. Every other day, you hear someone babbling about how AGI is just around the corner, but it never arrives.

This report aims to cut through the hype and demystify AI for the average reader, regardless of their technical background. We will embark on a beginner-friendly journey to explore the fundamental concepts of AI, explaining how these intelligent systems learn and operate. Through clear definitions, illustrative examples, and hands-on problem-solving exercises, you will gain a practical grasp of AI's current capabilities and the boundaries that define its reach.

We will begin by establishing a foundational understanding of AI and its subfields, as well as the crucial role of mathematical functions in its operation. Tracing the historical evolution of AI, we will highlight key milestones and the waves of optimism and skepticism that have shaped its development. We will also examine how public perception of AI has evolved alongside its technological advancements, from early anxieties about job displacement to the more recent concerns surrounding LLMs (Large Language Models).

A significant portion of this report will be dedicated to unraveling the "black box" of AI algorithms. Through practical, step-by-step examples, we will illustrate how neural networks learn to identify patterns and make predictions. This hands-on approach will provide an intuitive understanding of the core principles behind machine learning and function approximation.

Finally, we will confront the crucial topic of AI's limitations. By exploring the inherent constraints of current AI technologies, we aim to foster a more realistic perspective on its potential and prevent the pitfalls of hype and over-expectations.

By the end of this report, you will have a solid foundational understanding of AI, its capabilities, and its limits, empowering you to navigate the ongoing AI revolution with greater clarity and an informed perspective.

Chapter 1

What Is AI, Really?

(1.1) So what is AI ?

According to IBM AI or Artificial intelligence is technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy AKA match or surpass human intelligence.

Some subsets of AI include:

ML (machine learning):

It's basically developing a system that can learn without explicit

Instructions. It's how AI learns, and we will be focusing heavily on it.

NLP (natural language processing):

Systems that can understand human language and respond to it.

Computer Vision :

very self explanatory a system that can see and recognize
the world around it

Speech Recognition:

a system that can understand human speech

Robotics:

Robotics involves integrating AI to enable robots to perform complex tasks.

Expert Systems:

Expert systems are computer programs designed to mimic the behavior and judgment of human experts.

That's the most common dumbed down definition of AI but it doesn't tell you a lot

to get a better idea of how AI work lets understand how neural networks learn:

Basically the way neural networks learn is universal function approximation

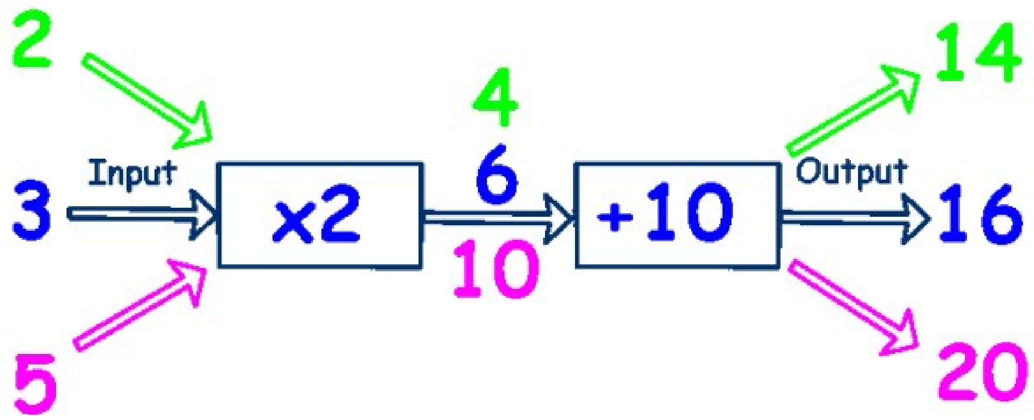
So what does that mean? First we need to understand Functions

(1.2) Functions :

Functions are input-output machines that take in a value and output a value they also describe the relation. Between the input and output values

So basically, a way to describe a relation between an input value and an output value,

for example The set of numbers [2, 3, 5] goes into a function as input and comes out as output [14, 16, 20]. So what's the relation or function between the input set and output set? Well, we are multiplying by 2 and adding 10 as shown in the figure below. So we would define that relation between them as f(input) = (input x 2) + 10



Well now that we understand what functions are **but how is approximating them useful?**

Well, because functions describe the world, everything from the sound you hear to the light that you see All of It can be described by functions. mathematical ones The whole world can be described by numbers and the relationship between those numbers. We can predict, model, and simulate anything by functions!

What AI helps us with is coming up with the functions that describe everything, for example, students attendance rates to their final exam grade results or the width and length of flower peddles to their color, AI can help us approximate those functions.

Even today's LLMs generate texts and images by using functions that a trained NN came up with

We will understand how AI learns and produces these functions in later chapters. For now, you should know that NNs (Neural Networks) learn by providing a specific algorithm with a massive dataset of input and output data. This allows the NN to develop a function that describes the relationship between that input and output data.

With function approximation, AI can predict things based on how much data you feed it, so it can create a better approximation. The more data, the more accurate it becomes. Companies like Google, IBM, Facebook, and much, much more have been using AI for years to maximize revenue from users.

They would feed the NN user information and activity as input and output, and the MM would approximate a function that predicts users activity based on their information like What is they most likely going to buy or watch

It can also predict the next word in a sentence or the next pixel in an image, which is how LLMs, as well as AI image and video generation, work as previously stated

(1.3) Examples of AI use:

As previously mentioned, AI is heavily used in social media apps for targeted ads, but It's also used in smartphones. voice assistance, chatbots, GPS navigation systems for picking the most optimal path, and self-driving cars. Its also heavily used in insurance companies to determine a plan for the client

Chapter 2

The Rise of AI

(2.1) When did it start origins of AI?

The origins of artificial intelligence can be traced back to the 1950s. In fact, in 1956, the Dartmouth Conference was held to bring together experts from various fields, including mathematics, computer science, physics, and more, to explore the possibility of "Synthetic Intelligence." At the time, the term *Artificial Intelligence* (AI) had not yet been coined. During the conference, participants discussed wide range of topics related to AI, such as natural language processing, problem-solving, and machine learning. They also outlined a roadmap for AI research, which included the development of programming languages and algorithms to create intelligent machines. This conference not only marked the birth of the field but also the moment when the term *Artificial Intelligence* was officially coined.

Hello Word! :

In the 1950s, a paper by Alan Turing titled *Computing Machinery and Intelligence* introduced the concept of the Turing Test. The test involved humans and a robot, where one human would act as the judge. The judge would engage in conversations with other humans, but one of those "humans" would actually be a robot. If the judge was fooled into thinking the robot was human, it would indicate that the robot possessed artificial intelligence. Turing predicted that by the year 2000, computers would have 1 MB of memory and that this test would be easily passed. Unfortunately, even with significantly more memory than 1 MB, computers still have not fully passed the test. Instead, they often rely on clever strategies to fool judges rather than demonstrating true intelligence.

(2.2) Eliza And Parry:

In the 1960s, the first chatbot was created. It was called Eliza and was designed to impersonate a therapist. It would ask open-ended questions, allowing the user to do most of the talking. It was a very simple script with pre-programmed responses. Some people claimed that it passed the Turing Test, those people are morons. It was a static program with pre-generated responses that didn’t leverage the full power of computers. Instead, it exploited human naivety. In fact, people were so naive back then that it was often difficult to convince them they had been conversing with a program rather than a real person. Similarly, the Parry chatbot, which impersonated a paranoid schizophrenic, also dictated conversations rather than engaging in genuine interaction.

```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZZ   AAAAA
                    EE      LL       II        ZZ   AA   AA
                    EEEEE   LL       II      ZZZ    AAAAAAA
                    EE      LL       II     ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Shakey the Robot:

The grandfather of drones and self-driving cars was developed in the late 1960s as an autonomous system designed to function independently of humans. Its capabilities were relatively weak and limited, but it paved the way for the advanced autonomous robots we see today. One of its key contributions was the popularization of the A* algorithm, which remains widely used for navigational tasks, whether in self-driving cars, drones, or even video games. A* was a significant improvement over Dijkstra's algorithm, providing a more efficient pathfinding solution. To enhance its functionality, the developers built a rangefinder for the system. Back in the day, digital cameras didn't exist, so they constructed their own analog-to-digital converter. The heavy mathematical computations and processing were not performed on the robot's body itself (often referred to as "Shakey"); instead, they were carried out on a computer located in the next room. Shakey's body was equipped primarily with sensors, leaving the computational workload to the external system.This invention was groundbreaking for its time and laid the foundation for many of the autonomous technologies we take for granted today.

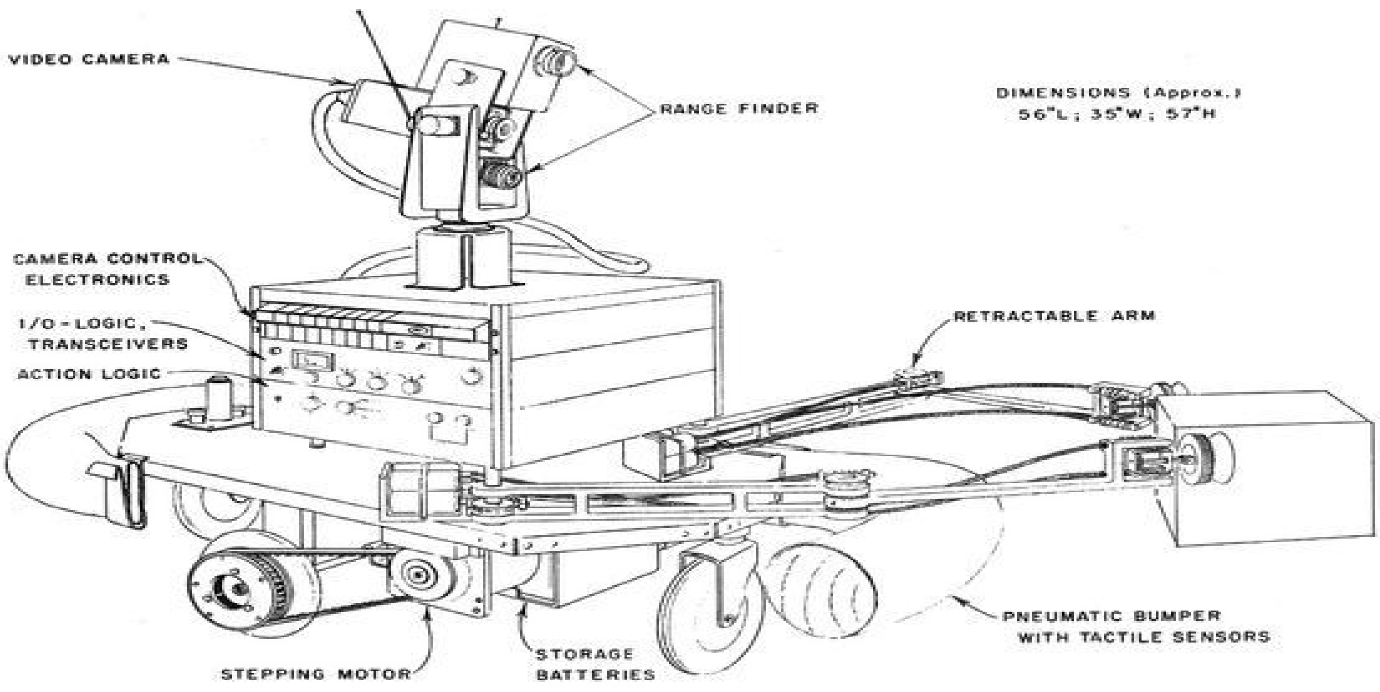


FIG. 2 AUTOMATON VEHICLE

The AI Winter;

In 1974, the applied mathematician Sir James Lighthill published a critical report on academic AI research. In his report, he argued that researchers had significantly over-promised and under-delivered on the potential of AI. This critique led to widespread skepticism about the field and resulted in substantial funding cuts for AI research and development.

(2.3) Hope:

after the brutal AI winter finally after big leaps the field managed to secure more funding and the advancement in hardware which helped a lot interests was still very low in AI but due to the following projects it started increasing

NavLab 1:

NavLab 1 was the first self-driving car made in 1986, equipped with television cameras and computers to digitize the visual data. The preparation process involved driving the car around a neighborhood to build a map of the area. Once the map was created, you could instruct the car to navigate using the pre-built map. However, it was more of a programmable car than the self-driving vehicles we see today. Users had to program specific instructions for how to use the route, when to speed up or slow down, and which modules to activate at particular times.Despite its limitations, NavLab 1 featured a trained neural network, which was a significant leap forward for the time. It could traverse intersections and stop when objects were detected in front of the car. This innovation was groundbreaking and brought much-needed funding and attention to the neglected field of AI, though not to the same extent as some later advancements.

Deep Blue:

Where do I even start? I think it was one of the most significant events in securing funding for AI. is shrouded in so many controversies that I could write 100 pages about it. Deep Blue was an developed by IBM to defeat chess champion Garry Kasparov—and it did in 1997. After that, I like to think the floodgates of modern AI were opened. This event was extremely popularized, and the wider population became aware of AI and its capabilities. However, the competition itself was shrouded in many controversies. I recommend the documentary "*Deep Blue Down the Rabbit Hole"* by Fredrik Knudsen, which is free on YouTube. It's extremely Interesting.

(2.4) WE ARE SO BACK!:

The 2000s were a very turbulent decade for many reasons—including my own birth But while world was doing its thing, AI researchers were hard at work, making breakthroughs. Hardware was also advancing at an incredible rate. In fact, at the time, Moore's Law stated that the number of transistors in a chip would double each year with an extremely minimal cost increase. As a result, t he AI boom was beginning, and we saw old concepts being pushed to new heights

Roombas:

The grandchild of Shakey the Robot, it uses an array of sensors to take in data and make decisions based on that data to clean and vacuum the space around It. It saw great commercial success in the early 2000s.

A.L.I.C.E ChatBot:

The granddaughter version of ELIZA and that other bad chatbot that doesn't deserve remembering. And by improvement, I mean massive. You can still test all of them today, and I advise you to do so. To this day, it still holds up amazingly and could easily fool people into thinking they're

talking to a human.

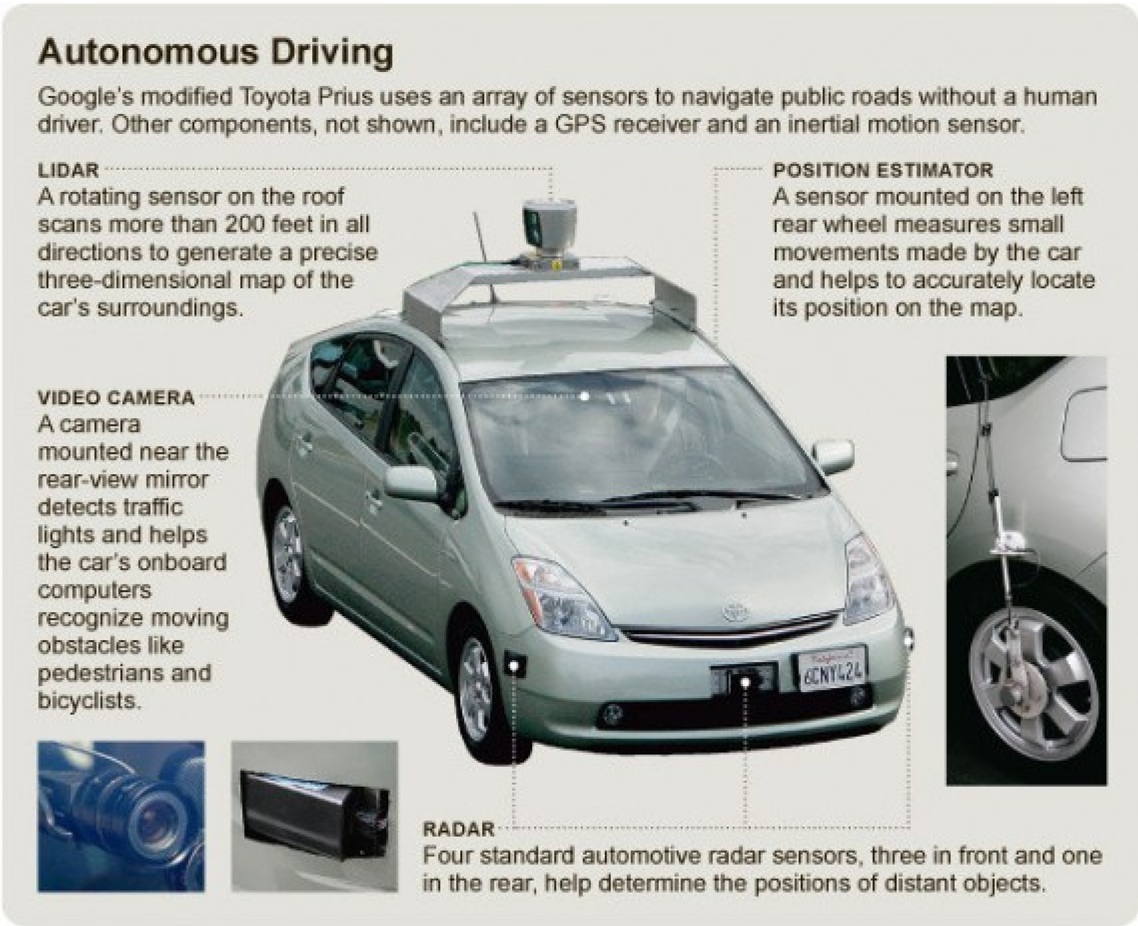
Chat Bot – ALICE (AI Foundation)

- *Human: hi my name is Carla*
  ALICE: *How are you doing, Carla.*
  Human: what is your name?
  ALICE: *My name is ALICE.*
  Human: what is your full name?
  ALICE: *ALICE = Artificial Linguistic Internet Computer Entity.*
  Human: when were you born?
  ALICE: *I was connected to the net on November 23, 1995.*
- *Human: Have you passed the Loebner Prize?*
  ALICE: *The Loebner Prize is an annual Turing Test sponsored by New York philanthropist Hugh Loebner. Dr. Wallace and ALICE won it in 2000. The ALICE program was ranked "most human computer" by the panel of judges.*
- *Human: who is Dr. Wallace?*
  ALICE: *He is a famous computer scientist, the author of ALICE, and a graduate of Carnegie Mellon.*
- *Human: Where does Dr. Wallace work?*
  ALICE: *he works from his home in San Francisco.*

Carla P. Gomes
INFO372

Google Routes and Self-Driving Cars:

The grandchildren of NAVLAB. Google Maps dynamically sorts the shortest path to a destination, and Google's self-driving cars are self-explanatory.



Source: Google

THE NEW YORK TIMES; PHOTOGRAPHS BY RAMIN RAHIMIAN FOR THE NEW YORK TIMES

You should be noticing a trend: old technologies were improved and pushed to their limits in the early 2000s due to advancements in hardware and software but mostly hardware.

SIRI :

The first major product to popularize natural language processing (NLP) was developed by SRI researchers. At the time, it was impressive because it was able to understand human language (natural language understanding - NLU), translate language, and respond (natural language generation - NLG) and NLP in general started to take off which lead to the creation of LLMs.

The Gold Rush:

After OpenAI's release of GPT-3.5 in 2022, the floodgates opened. Large language models (LLMs) started appearing everywhere; you couldn't escape them. They were also forcibly injected into everything, for better or worse, and anything with the name "AI" received extra funding, regardless of its merit. The peak of the AI bubble is now upon us.

(2.5) Today (early 2025):

The gold rush continues in 2025, but the hype is waning, particularly with the emergence of models like DeepSeek. DeepSeek, a reasoning LLM that, reportedly outperforms ChatGPT while being cheaper to train. How much cheaper you ask? **training cost around $100 million, DeepSeek's was approximately $5 million** thats still a lot, but a massive improvement. These past five years will definitely be remembered as the era of LLMs, and they're only going to grow. Companies are even building nuclear plants to satisfy the immense power demands of AI. **It's estimated that a single ChatGPT prompt consumes the same amount of energy as a light bulb burning for 20 minutes, and 10 times the power of a Google search.** Keep I n mind, that's just for using the *trained* model; training these models consumes waaaay more power. Most AI companies are still spending far more than they're earning. However, one thing is clear: AI is here to stay.

(2.6) Reasoning LLMs vs LLMs:

Reasoning LLMs represent a massive, yet somewhat underrated, breakthrough in AI. Non-reasoning LLMs are essentially glorified autocorrect; they're impressive, but they struggle with any task of moderate complexity. Reasoning LLMs, on the other hand, break down complex tasks into simpler problems and solve them one by one. This change has drastically increased the proficiency of AI models. While they are far from perfect, reasoning models are a significant leap forward compared to standard AI models it does take a lot more time and power though.

Chapter 3

How did the people React?

(3.1) The Inception:

Back in the 1960s, the term "AI" was coined. At that time, AI could create terrible music, recognize voices, and identify shapes. then, fears of mass unemployment are on the rise which is a recurring theme throughout AI's history. People worried that computers would replace them and excel at their jobs, leaving them with alternatives. They were partly right, as computers did take some jobs, but they also created many more. That was the common perception. But what about researchers and scientists? They predicted that within 10 years, AI would be the chess champion, create amazing music, and outwit humans. They were completely wrong about the timeline, but some of their predictions did eventually come true.

(3.2) The AI Early Teen Years:

AI had a hopeful childhood, and like many of us, had difficult adolescence where a lot went wrong. But it managed to pull through in the end. Due to the "AI winter," over-promising and under-delivering led people to forget about AI, though the concern of machines taking their jobs never entirely disappeared. Funding was scarce, breakthroughs were sparse, and scientists and researchers criticized the over-hyping of AI and the outlandish claims

(3.3) Late Teen Years:

A major breakthrough occurred, reigniting the AI hype. Several things happened, but in my opinion the biggest and most important was Deep Blue becoming a chess champion. All the hype returned, and many more people became aware of AI and its capabilities. AI was also used to prove a mathematical theorem. However, it was still bad at making music.

(3.4) Early Adult Hood:

AI received a massive boost due to advancements hardware, leading to wider adoption. While most people were still unaware of AI's presence, the minority who were saw its great potential coming to fruition. This was also a time when researchers were learning more about and developing deep learning techniques. deep learning

(3.5) Adult hood:

The best way to **describe AIs early adult hood is fear and misunderstanding.** After further advancements in software and hardware, we witnessed the emergence of LLMs (large language models), and the general public experienced them after the release of ChatGPT 3.5. AI's popularity was reaching new heights. It transformed from a niche subject to the most sensationalized topic imaginable. People feared for their jobs and even for humanity's future. Scientists were repeatedly claimed that AGI (artificial general intelligence) was imminent and posed an existential threat if not handled correctly. The AI hype peaked in this era, with any company involving anything that has to do with "AI" getting billions of dollars in investment. Worst of all, AI remains largely a black box; the general public doesn't understand its inner workings of, which a significant problem. People began rejecting AI for various reasons, including fear of the unknown. This was particularly true in the art world, which is understandable. Art, by definition, is human creativity, so forcing AI into it felt tasteless. These AI-generated art forms were heavily and understandably scrutinized and rejected. Major security issues also arose from AI's ability to generate highly convincing videos and photos from prompts. This only increase the rampant misinformation and scams already everywhere on social media.

(3.6) Post adult hood:

The AI hype is dying down, but AI itself is here stay. AI companies made false claims to attract investment, and people are starting to realize that. Public perception is becoming more grounded. The fear of AI taking jobs is still present and will likely remain. However, people have also begun to embrace and use AI more. There's a growing acceptance that AI is here to stay, even though major problems with AI remain largely unsolved. These include its immense power consumption, the use of copyrighted material to train algorithms, security concerns surrounding deepfakes, and the very real job lose and layoffs that are occurring.

Chapter 4
Behind The Algorithm

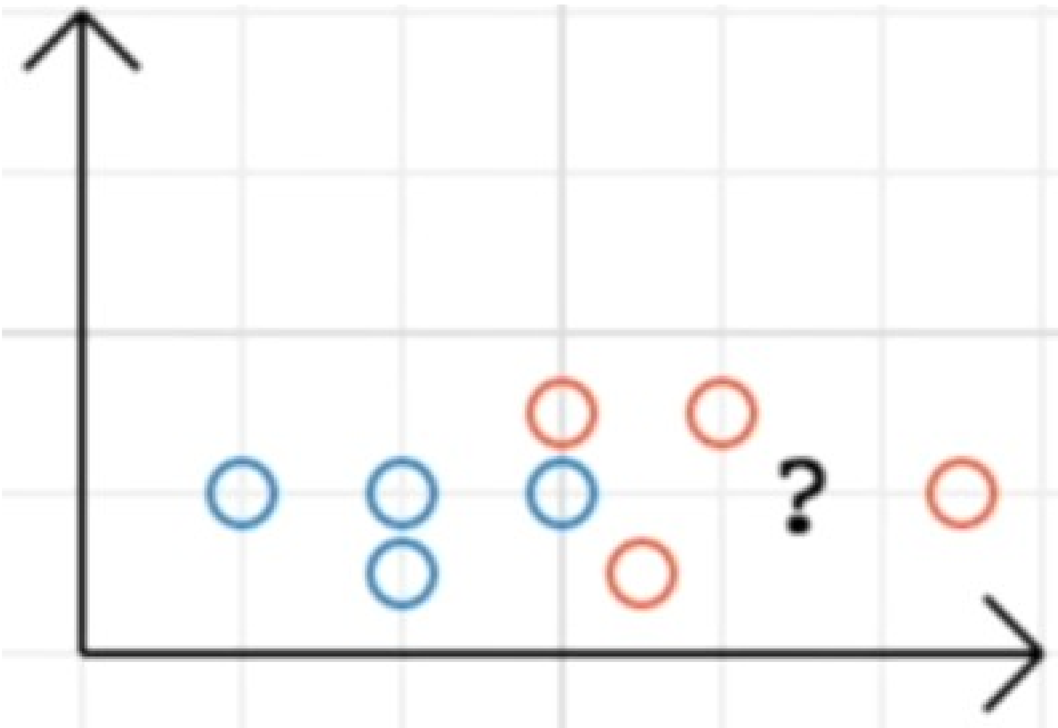
(4.1) In this chapter, we will be learning how neural networks train and learn! As a quick refresher to what we said in the first chapter, we defined NNs as general function approximators that generate functions based on the data we feed them. We also learned why functions are super important because with them we can model and predict anything. In this chapter, we will be able to build an AI model that does these things!

(4.2) First, let's start off with a problem. Let's say you're a farmer and you sell two types of flowers: red and blue. Red flowers are used for decoration because they have a long life, and blue flowers are used for perfume because they have a very short life. Each flower blooms randomly; it could take them between 1 to 2 weeks. So, you won't know for sure what flower it is until they bloom, which can cause you a lot of financial damage and a bunch of headaches. So, you, being the smart farmer you are, get a bunch of bloomed flowers and start measuring them, and your data set looks like this:

color	○	○	○	○	○	○	○	○	
length	3	2	4	3	3.5	2	5.5	1	4.5
width	1.5	1	1.5	1	.5	.5	1	1	1

Aw, shucks! You accidentally measured the length and width of a flower petal, but you forgot to measure its color. These darn flowers and the headache they cause you!

But, being the smart farmer you are, you graph the data you have to see if there is any relationship between the data that will lead you to predicting the missing flowers' color. You got this, and it looks like this:



X-axis: flower petal Length; Y-axis: petal flower width

You start to see it a lot more clearly now. There's clearly a relationship between the flowers' petal length, width, and their color. We see here that the more the flower's length increases, the more likely its color is red. Therefore, you can confidently estimate that the missing flower's color is red. We describe this relationship as a linear one.

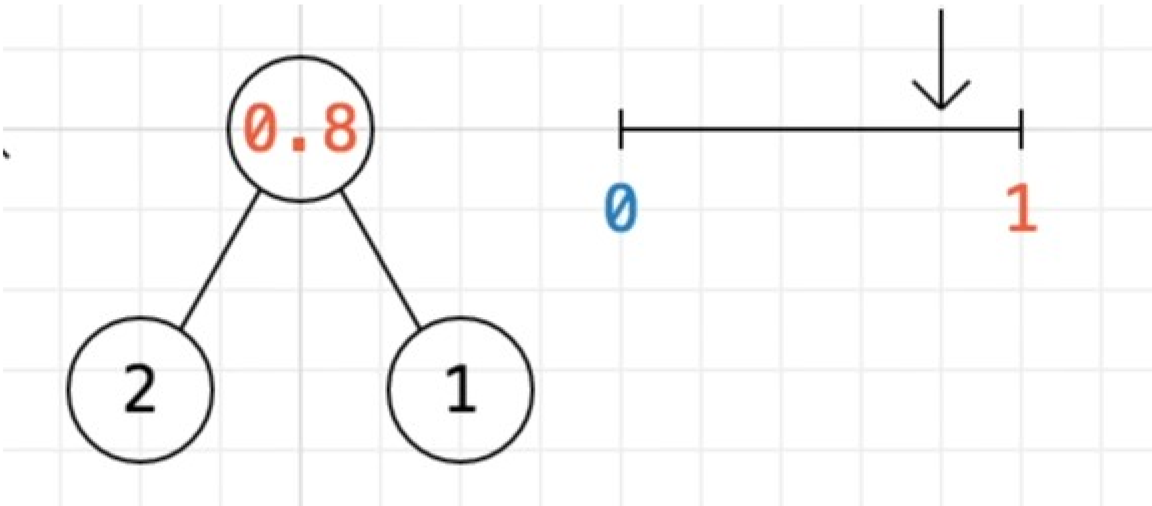
(4.3) A linear relation means that one variable changes in amount proportionally to another variable.

A good example of a linear relation is someone's hourly wage. Let's say you get paid 15 dollars per hour. The more hours you work, the more you're going to get paid. If the hours increase, the money will also increase, and vice versa.
The same could be said for our flowers: the more the length and width increased, the more likely it is for the flowers to be red."

So, well done! You managed to solve this problem, but it took you a lot of effort, and it's not the most accurate. Plus, you had to work with just 10 data points and 3 variables. What if you took the data of 1,000 or 10,000 flowers? And what if it wasn't a relation between 3 variables, but 10 or 20? That would be impossible for a human to do, but that's precisely why computers were made. In fact, increasing the number of data points and the parameters will only increase the accuracy of our NN.

So, now we need to find an algorithm that takes those measurements and finds the relation between them. Wait, finding the relationship between data points? Isn't that just another name for finding a function? And, as we learned previously, finding the relationship between input and output means finding a function between input and output data. NNs are approximate function generators! So, that sounds like the perfect use for a NN!

So how can we make a NN a computer algorithm do that ?



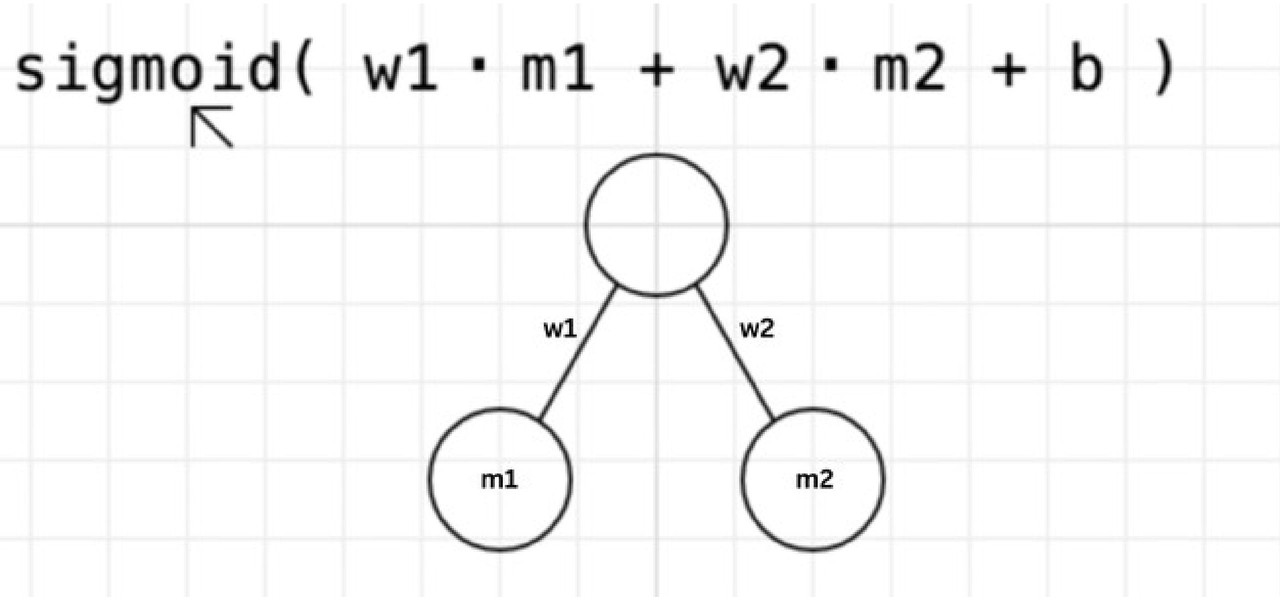
Here's how our model is going to work. We will keep it simple for this project. We will have two layers: one output and one input, with no hidden layers.

Hidden layers are the layers between the input and output layers, acting as intermediate processing stages. They're where the network performs complex calculations and transformations on the input data, gradually extracting meaningful features and patterns. Think of them as the 'thinking' part of the network, where it learns to understand the relationships within the data before producing a final result. We will get more into them in the next project

Since the data has a linear relationship we will be using a linear equation to represent the model model

a line can be represented by the following functionality

f(x) = wx + b



w1 = weight 1, associated with our first parameter.

w2 = weight 2, associated with our second parameter.

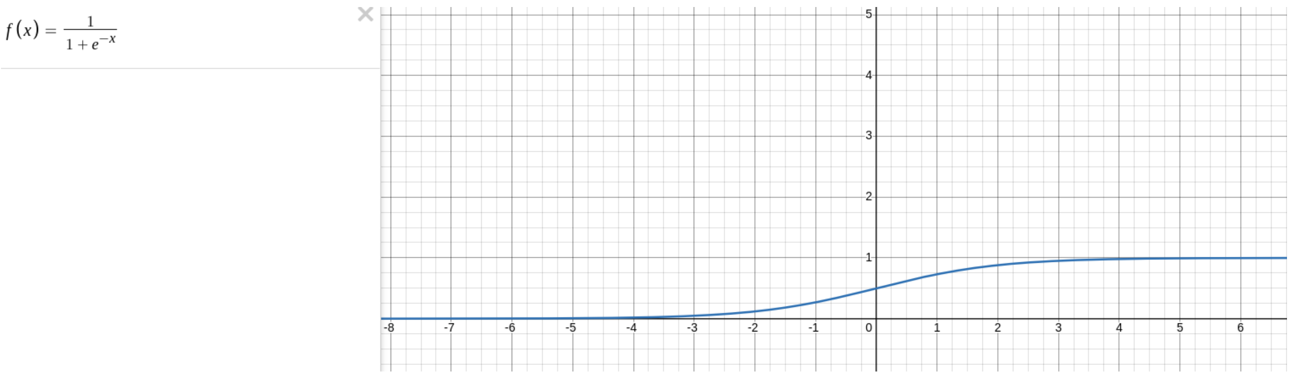
b = bias of the function.

(4.4) Sigmoid function: it's an activation function that takes in a value and squishes it between 0 and 1.

So, essentially, we would be squishing the result we got from our linear equation to a value between 0 and 1. But why, and how, you ask? And what is an activation function?

Well, why we do it is because we are trying to predict a binary value, a value that is either 1 for red or 0 for blue flowers. This function will show us how certain our model is in the flower being one or the other. If it's 1 or close to 1, it means it's certain it's red. If it's 0 or close to 0, it means it's sure it's blue. Remember, the sigmoid function will not just return 0 or 1, but a spectrum including 0 and 1

how is by using this equation



F(x) = 1 / 1+e^{x}

Activation functions: are essentially functions that transform the output of a neuron, often into a value that can be interpreted as a probability or a decision. They take the numerical result produced by our model's function and convert it into a meaningful output.

For example, in our flower model, an activation function like the sigmoid would take a number like 0.8 and indicate a high probability of the flower being red, while 0.2 would indicate a low probability, suggesting it's likely blue.

(4.5) Were Gonna be Using the python programming language but this guide will work with any programming language

first we import our data set it should look like this

```
import math #for math operations
import random # for generating random numbers
```

```
#flower length, flower width, flower type (0 blue, 1 red)
data = [[3, 1.5, 1],
[2, 1, 0],
[4, 1.5, 1],
[3, 1, 0],
[3.5, .5, 1],
[2, .5, 0],
[5.5, 1, 1],
[1, 1, 0]]
```

```
mystery_flower = [4.5, 1]
```

We defined our data matrix. It's called a matrix because it's two-dimensional.1 We have rows and columns. If it's only rows or only columns, it's called a vector. If it's one number/element, it's called a scalar. If it's higher than two dimensions, it's called a tensor.

After we implement our activation function and neural network

```
def sigmoid(x):
return 1/(1 + math.exp(-x))
```

```
def NN(v1, w1, v2, w2, b):
z = v1 * w1 + v2 * w2 + b
return sigmoid(z)
```

Now lets initialize our weights and biases we do that by picking a random number from -2 to 2

```
w1 = random.uniform(-2, 2)
w2 = random.uniform(-2, 2)
b = random.uniform(-2, 2)
```

now lets try it !

```
for point in data:
v1 = point[0]
v2 = point[0]
print(NN(v1, w1, v2, w2, b))

#output the closer to red the more our NN is certain that its red and vise versa for blue
# 0.9994938209555344
# 0.9944522240876219
# 0.9999540280269855
# 0.9994938209555344
# 0.9998474359125414
# 0.9944522240876219
# 0.9999987425342811
# 0.9421044754357956
```

well our NN failed to predict anything and of course it failed the wieghts and biases were totally random and they didn't change we need to train our NN in a way to change the wieghts and the biases so that it can be better at predicting one way we can do this is

(4.6) Cost Functions:

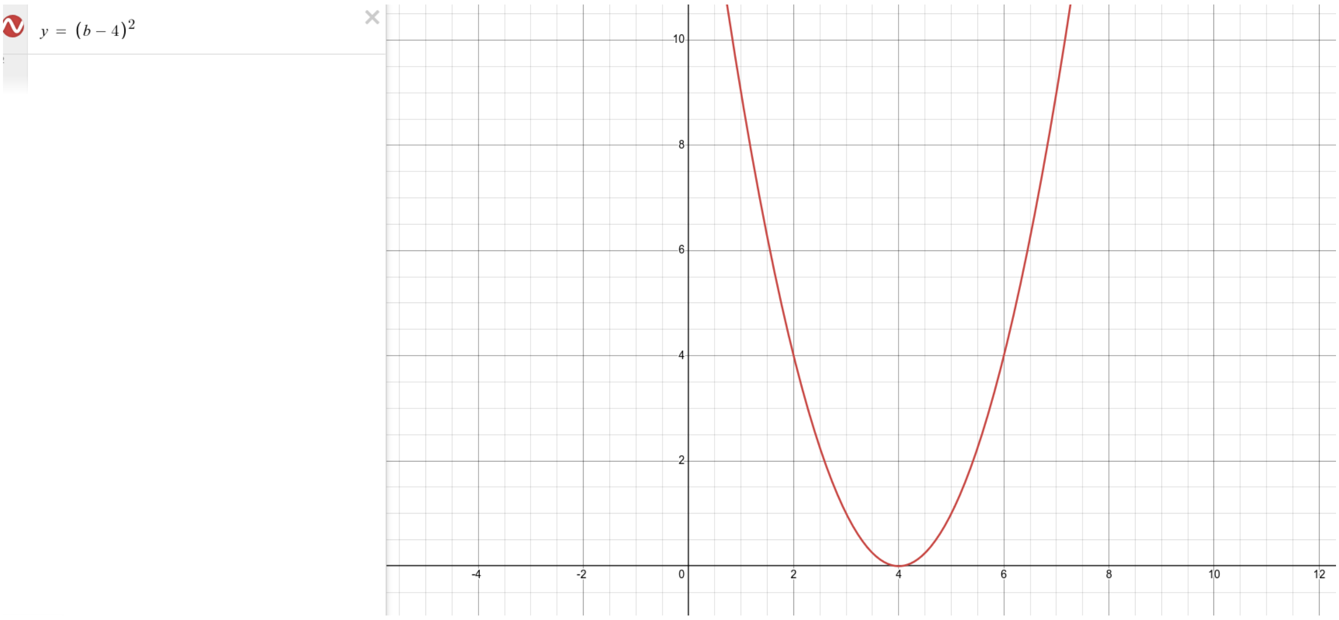
is a function that basically tells the computer how bad its got its predictions its the squared difference between the predicted value and the actual value and gives us a number that represents how wrong the computer guess is so our goal is to minimize this number because the less it is the more right our guess is
and if its 0 it means the computer guess is correct **they're a great measurement of our NN performance**

Cost = (predicted value – target value $)^2$

we square it to penalize the effect of larger errors

lets try an example now:

lets say we have a simple function
f(x) = x
all it does take in number x and outputs it
lets try x = 4
f(4) = 4
our cost function would be (predicted value – 4 the real value$)^2$



As we can see the cost function is a Parabola (our cost function) that has a value of 0 at 4 which is the real value

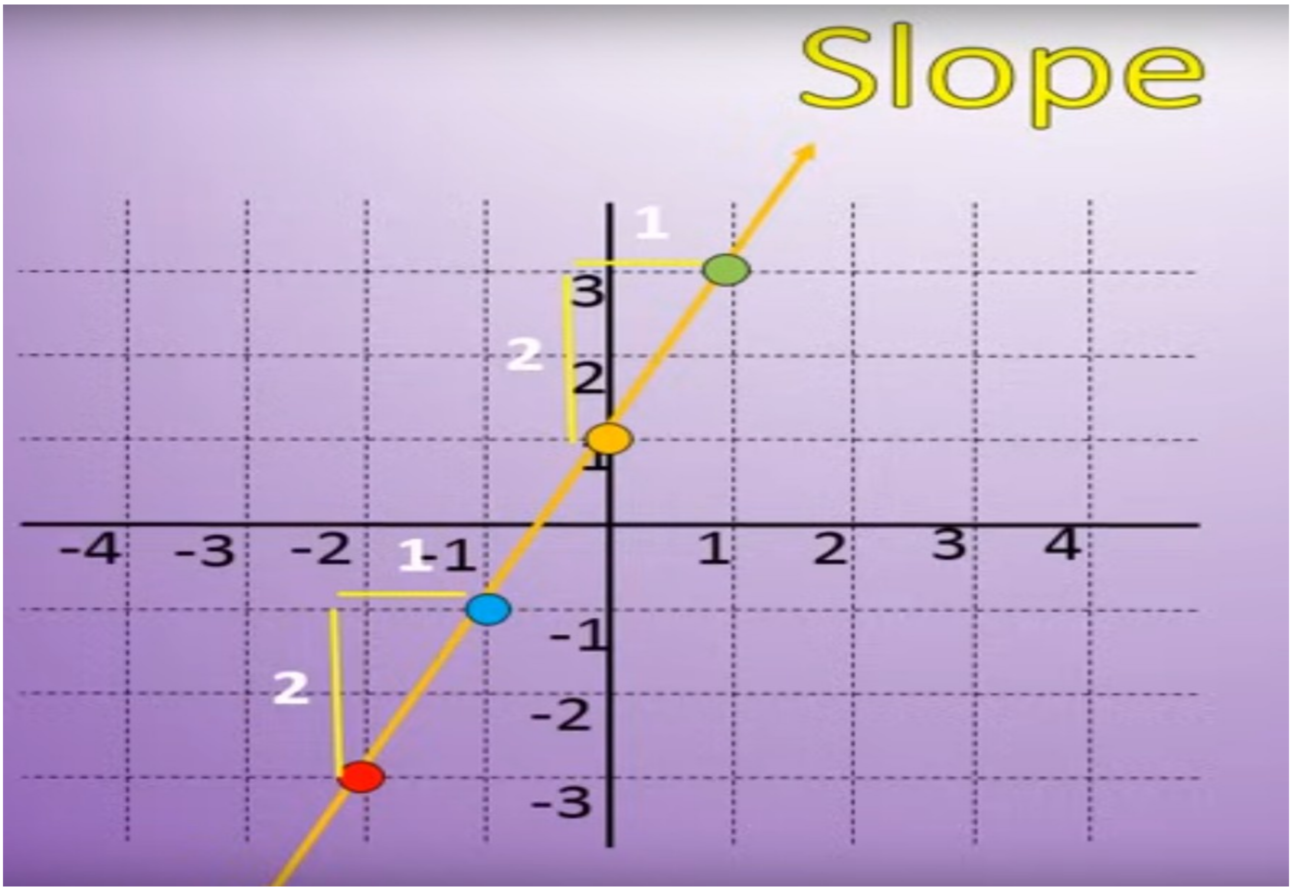
so now we can predict how wrong our NN is but how do we make it self correct ?

Well we do that by subtracting from the slope of the function but what is a slope ?

(4.7) Slope:

A slope is basically how steep a line is. There can be many slopes, and a slope always remains consistent; you can pick any two points on a line, and they should give you the same slope. It can also be defined as the rate of change.

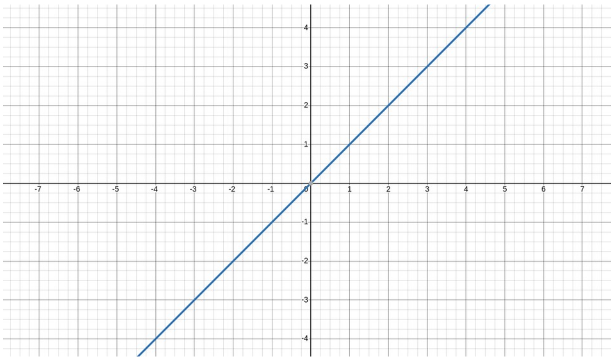
so its the rise/run

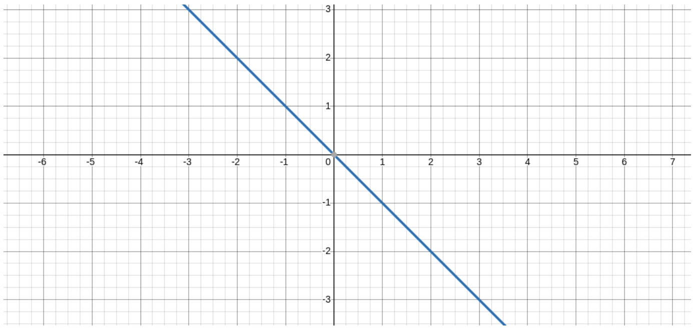


for example:

if we pick 2 points red and blue we will get
rise = 2(y), run = 1(x)
slope = y/x = 2/1 = 2
if we pick point red and yellow we should get the same answer rise = 4 y run = 2x
slope = y/x = 4/2 = 2/1 = 2
so a slope is a always consistent no matter what point and it
defines how steep a line is

There are many types of slopes, including negative, positive, and zero slopes.



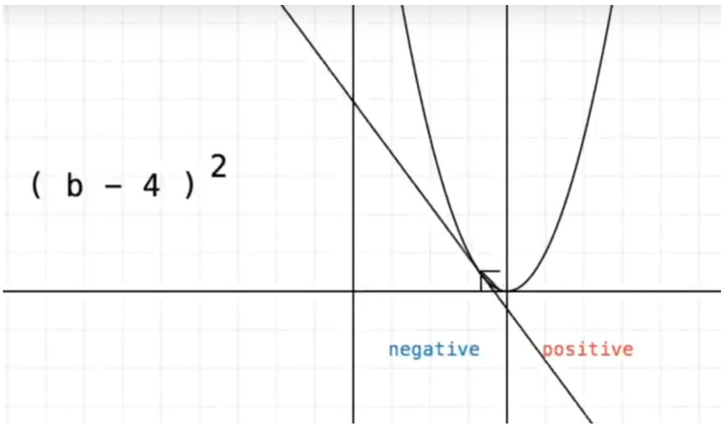


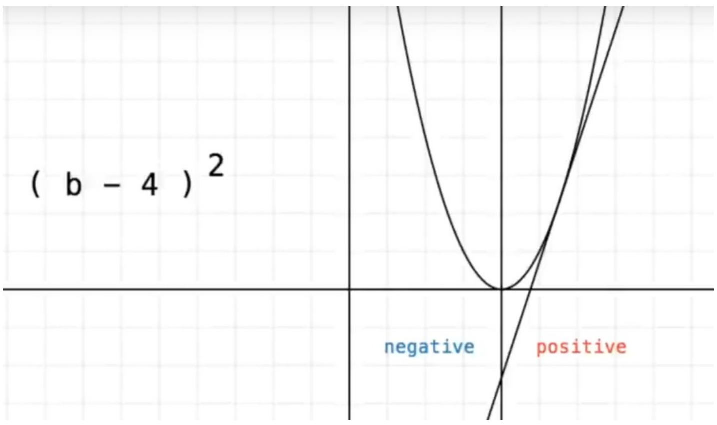
A positive slope is where, from left to right, a line goes in a positive direction.

And **a negative slope** is the opposite, where it goes from a positive direction on the left to a negative direction on the right.

There are many ways to classify a slope.

So, how does a slope help us self-correct? Well, in our cost function, there are two slopes: negative and positive, depending on our predicted value. And if we subtract by that slope, we can get closer to 0.





The closer our value is to 0, the lower the slope gets, until we get to 0 and our slope will also be 0. So how do we get the slope of a function? We learned how to get the slope of a line, but how **do we get the slope of a function**?

Well, we derive it with respect to the target value!

The slope of the cost function is:

cost = (b - 4)^2

Slope is approximately (cost(b + h) - cost(b)) / h

= ((b + h - 4)^2 - (b - 4)^2) / h

= (h*(h + 2b - 2 * 4))/ h

= h + 2b - 2 * 4

= 2(b - 4) + h

The smaller the h, the better the approximation.

So, the slope is approximately:

= 2(b - 4)

= 2 * (predicted value - target value)

And we can test this by substituting this into our previous equation.

Target value = 4 Predicted value = 3

2 * (3 - 4) = -2

Target value = 4 Predicted value = 5

2 * (5 - 4) = 2

So, we can see here that if we are above the real value 4, we get a positive slope, and if we're under, we get a negative slope.

So, we would get closer to 4 if we take the difference between our prediction and the target value

So now we can see how wrong our model is and how to correct it by taking the difference between our models predicted value and the real value and if we take the slope (rate of change) for our model function we can see how to fix our models error and get the cost down!

(4.8) Lets test this in our program

```
def function_slope(predictedValue, realValue):
return 2 * (predictedValue – realValue)
```

```
predictedValue = 10
realValue = 4
learning_rate = 0.1
```

Our predicted value is what our model got, and the learning rate is how much we want our predicted value to self-correct in the direction of the real value. We don't want it to be too much because our algorithm will overshoot in the other direction. We will see how the learning rate affects our example here.

```
for i in range(5):
predictedValue = predictedValue - (learning_rate * function_slope(predictedValue, realValue))
print(predictedValue)
```

```
# output learning rate (0.1):
# 8.8
# 7.840000000000001
# 7.072000000000001
# 6.457600000000001
# 5.966080000000001
```

```
# output with learning rate (0.3)
# 6.4
# 4.96
# 4.384
# 4.1536
# 4.06144
```

```
# 0.3999999999999986
# 6.160000000000001
# 2.7039999999999993
# 4.7776000000000005
# 3.5334399999999997
```

we can see with our very low number if iterations (5) 0.1 learning rate was too small 0.3 was better and 0,8 was too big now you can play around with these for optimal results this was just to show case how the learning rate works one thing we also have is sum of costs where we will take the sum of all costs for each point to get a better general idea of how our NN is doing

costs = $\sum$ (Predicted Value – Real Value) ^{2}

$\sum$ means sum

lets say we have

predicted values = [1, 2]

real values = [1, 3]

costs = $(1 - 1)^2 + (2-3)^2$

and for costs average = 1/length of all data points $\sum$ (Predicted Value – Real Value) ^{2}

(4.9)now lets implement this :

```
import math
import random
from matplotlib import pyplot as plt
```

matplot lib is for plotting data

```
#flower length, flower width, flower type (0 blue, 1 red)
data = [[3, 1.5, 1],
[2, 1, 0],
[4, 1.5, 1],
[3, 1, 0],
[3.5, .5, 1],
[3, 1.5, 1],
[3.5, 2.5, 1],
[2, .5, 0],
[5.5, 1, 1],
[1, 1, 0]]
mystery_flower = [4.5, 1]
```

Our data set and mystery flower that is red

```
def sigmoid(x):
return 1/(1+ math.exp(-x))
```

```
def dsigmoid(x):
return sigmoid(x) * (1-sigmoid(x))
```

sigmoid derived sigmoid function

```
w1 = random.uniform(-2, 2)
w2 = random.uniform(-2, 2)
b = random.uniform(-2, 2)
```

initialize weights and bias

```
total_costs = []
iterations = 1000000
learning_rate = 0.1
```

```
for i in range(iterations):
        randomPoint = random.choice(data)
        length = randomPoint[0]
        width = randomPoint[1]
```

we chose a random point in the data and we train our model with it

```
        z = length * w1 + width * w2 + b
        predicted_value = sigmoid(z)
```

we get the value that our model predicted

```
        real_value = randomPoint[2]
        cost = (predicted_value - real_value) ** 2
```

we get the cost for that prediction

```
        total_costs.append(cost)
```

we add that cost to the other total costs so we would now our total costs

```
#deriving
        dcost_rpredicted = 2 * (predicted_value - real_value)
        dpredicted_value_rz = dsigmoid(z)
```

we derive the cost function and the sigmoid function with respect to the predicted value

```
        #deriving z it has 3 variables w1 w2 and b
        dz_rw1 = length
        dz_rw2 = width
        dz_rb = 1
```

we derive our function with repect to each variable

```
        #now to derive cost we ussing the chain rule
        dcost_rw1 = dcost_rpredicted * dpredicted_value_rz * dz_rw1
        dcost_rw2 = dcost_rpredicted * dpredicted_value_rz * dz_rw2
        dcost_rb = dcost_rpredicted * dpredicted_value_rz * dz_rb
```

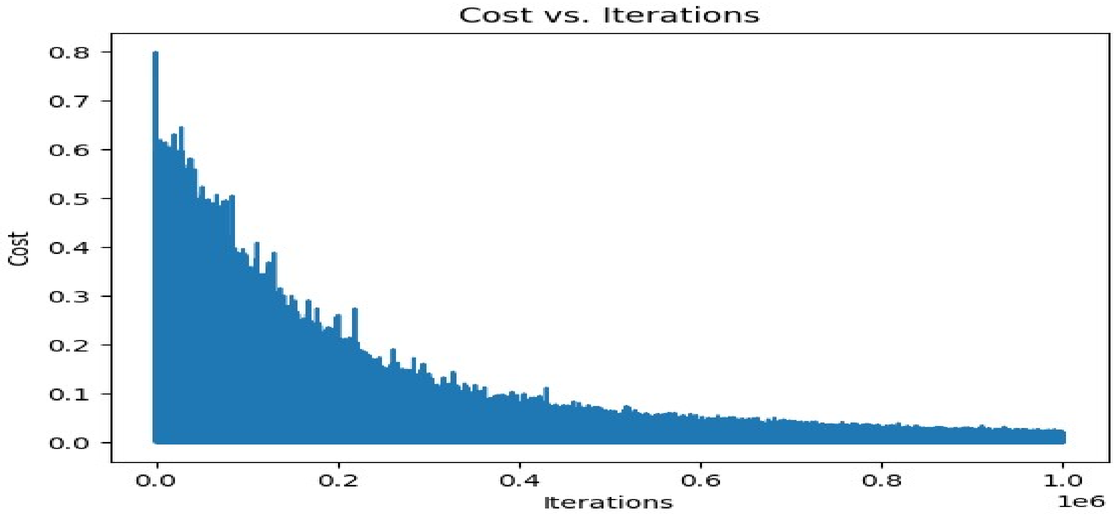
and now we derive the costs with repect to every variable

```
        w1 = w1 - learning_rate * dcost_rw1
        w2 = w2 - learning_rate * dcost_rw2
        b = b - learning_rate * dcost_rb
```

we change the wieghts based of the the cost and learning rate

```
plt.plot(total_costs)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost vs. Iterations")
plt.savefig('plots/cost_plot.png')
```

(4.10) Results:



We can clearly see a trend of the cost going down with each iteration. The data we have is too little and not ideal, so in a real-life scenario, this wouldn't be acceptable. However, this model does its job for our small purpose. In fact, it's about time to test it out! But have fun adding/removing new data points and changing the number of iterations to minimize the cost!

```
def predict_flower(w1, length, w2, height, b):
if sigmoid(w1 * length + w2 * height + b) > 0.5:
return "red"
else:
return "blue"

flowerType = predict_flower(w1, mystery_flower[0], w2, mystery_flower[1], b)

print(flowerType)
#output
# red
```

WORKS FINE!

(4.11) Terminology:

Forward propagation:

It describes the act of our neural network (NN) making a random guess. This involves initializing the weights and biases, and our NN guesses based on these initial weights and biases.

Backwards propagation:

As you can tell from the name, it's the opposite of forward propagation. It's where our NN self-corrects and learns by comparing the actual value to the value it guessed and making the necessary changes to the weights and biases so that its guess is closer to the actual value

Gradient Descent:

is a way to find the lowest point of something by taking small steps in the direction that decreases it the most. It's like a smart way of feeling your way down a hill without being able to see the whole thing. And as we explained before, the slope is at its lowest or 0 when it's at the target or real value, so the lower we get, the more accurate our NN is.

And with that, you now pretty much have a great understanding of how AI works on a high level. It's just scaling these simple principles until you get what you want. For example, with LLMs, instead of feeding them flower data, we would feed them millions of texts from books and websites, and instead of them predicting a red or blue flower, they try to predict the next word in a sentence. For example:

You say,

"What's 1 + 1?"

The AI model would take that input, break it down into tokens, and try to predict what the most likely word is to come after "1 + 1," and it turns out to be "2."

The same goes for image prediction. For example, let's say we want to train our NN to spot the difference between images of red and blue flowers. We would need thousands of pictures of blue or red flowers. We would turn these flowers into CSVs where their label and every pixel value would be there, and we would feed that to our NN. Our NN will find the relation between pixel values and the image content, and it will be able to guess it too. But that's not all sunshine and rainbows. AI has some serious limitations when it's scaled up like that, and we will find out more about them in:

Limits of AI
And of they can be broken

AI isn't perfect right now; in fact, it's far from it. But in this chapter, we will list those limitations and try our best to mitigate them!

(5.1) Hardware dependency:

If you were paying attention in Chapter 2, you should've realized that AI is the carriage, while hardware is the horse. So, the AI field is extremely dependent on advancements in hardware, and we're kind of reaching a limit. The biggest reason why advancements happened in the 2000s with AI is because the hardware was strong enough to

make those ambitious ideas a reality. But now, sadly, we've hit a wall with hardware. Our

transistors are so small now they don’t even go by regular physics;

they follow quantum physics. So, for now AGI (artifical general intelegince) is not even close

(5.2) Scaling issues :

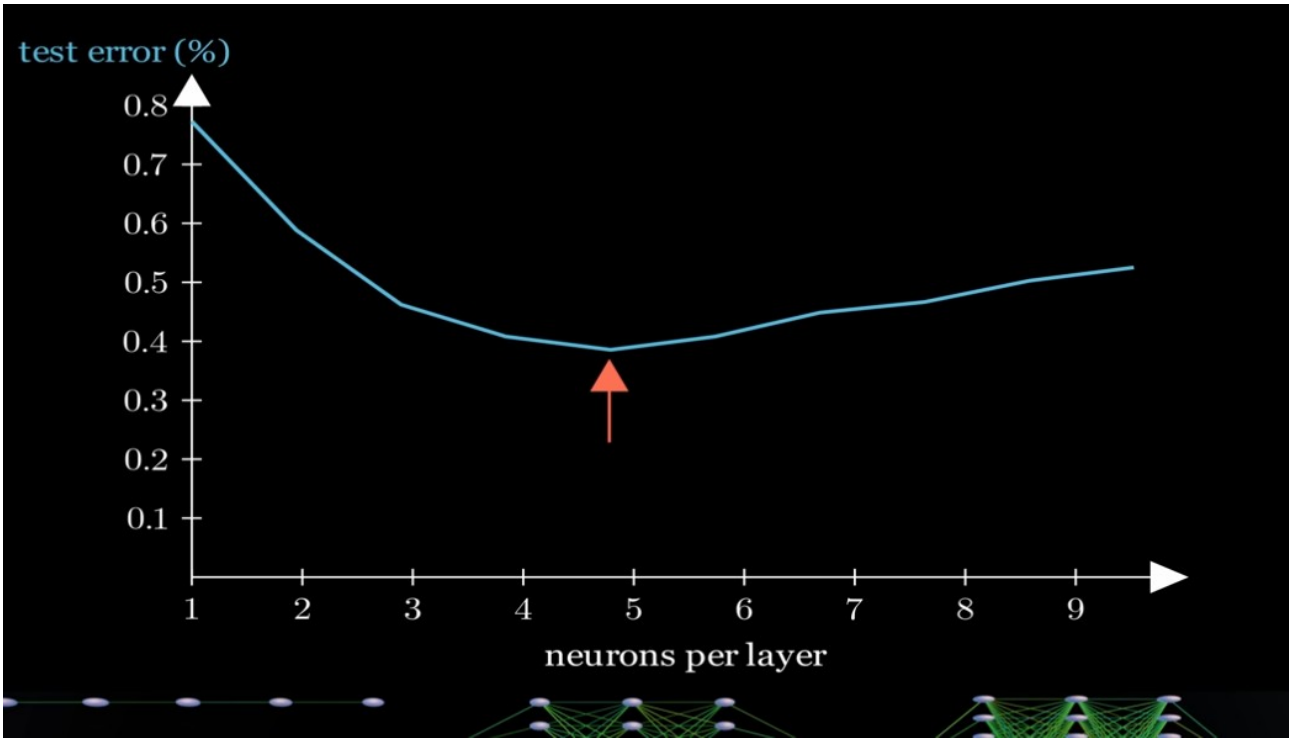
AI scaling has got to be one of the most unintuitive things ever. It seems simple: by adding more data to your LLM and more compute, it should be better, right? Well, no, of course it's not that simple. It all comes **down to over fitting.**

as we learned, the primary goal of our AI model is to get closest to the target value and minimize cost. So, in some cases, our AI model can just memorize the dataset, be 100% accurate on our training sample, but in real-world application, it's as good as useless because it didn’t learn anything; it only memorized the data in our training set without really understanding the relationship between the variables. So, if you have a model with a lot of weights and biases, there is a very good chance that it won't even learn anything it will just memorize your data set.

Overfitting:

Imagine you're studying for a multiple-choice test. Instead of actually learning the material, you simply memorize the correct letter answer for each specific question in the practice test. You might get 100% on that practice test. However, if the real test has slightly different questions, even if they cover the same topics, you won't know the answers because you never truly understood the concepts. Your "learning" was specific to the practice test and didn't generalize to new situations. That's what overfitting is like for an AI model – it performs very well on the data it was trained on but fails to perform well on new, unseen data because it just memorized the training examples instead of learning the underlying patterns.

for example

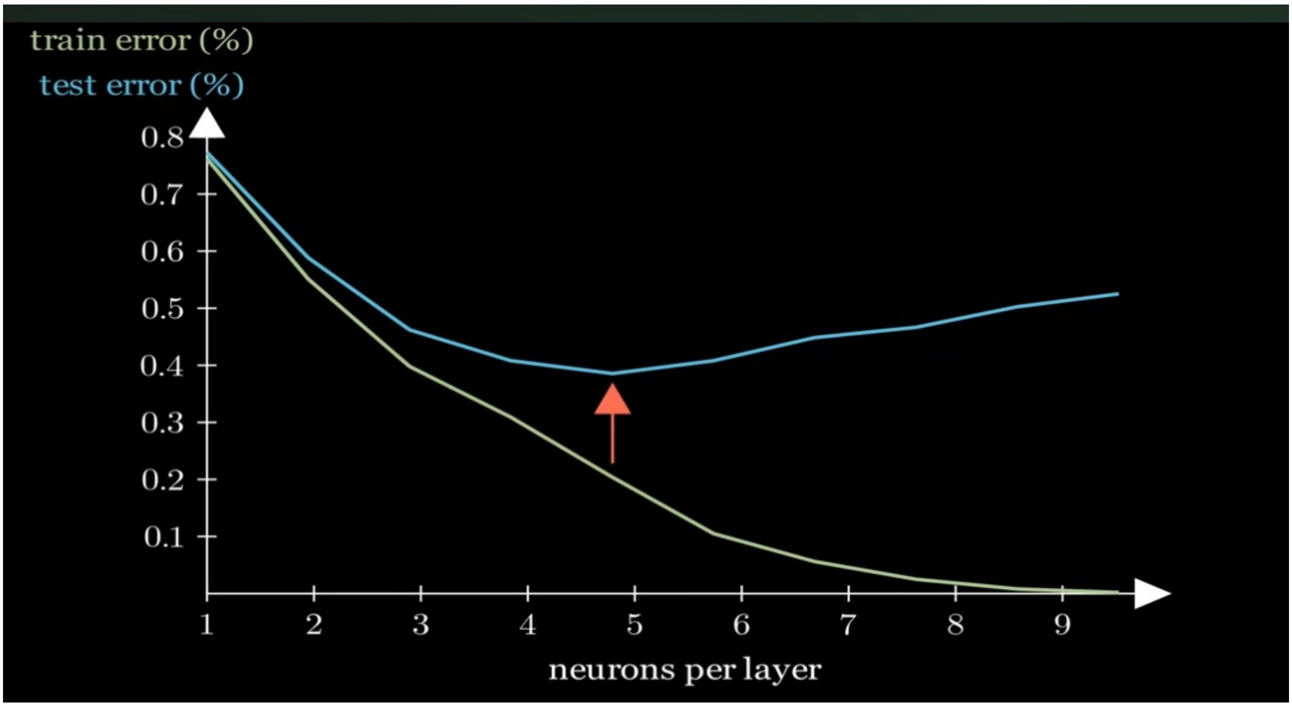


In this model, we will see the error rate decreasing with the number of neurons per layer. Then, it starts to increase again as we increase our neurons. But why? Well, again, it's because of overfitting, where instead of the model learning the dataset, it just memorizes it. So how do we solve this?

Solution 1 (for small scale models):

Stop just throwing more neurons at the problem

use the correct amount and always use two datasets: one for testing and another for learning.

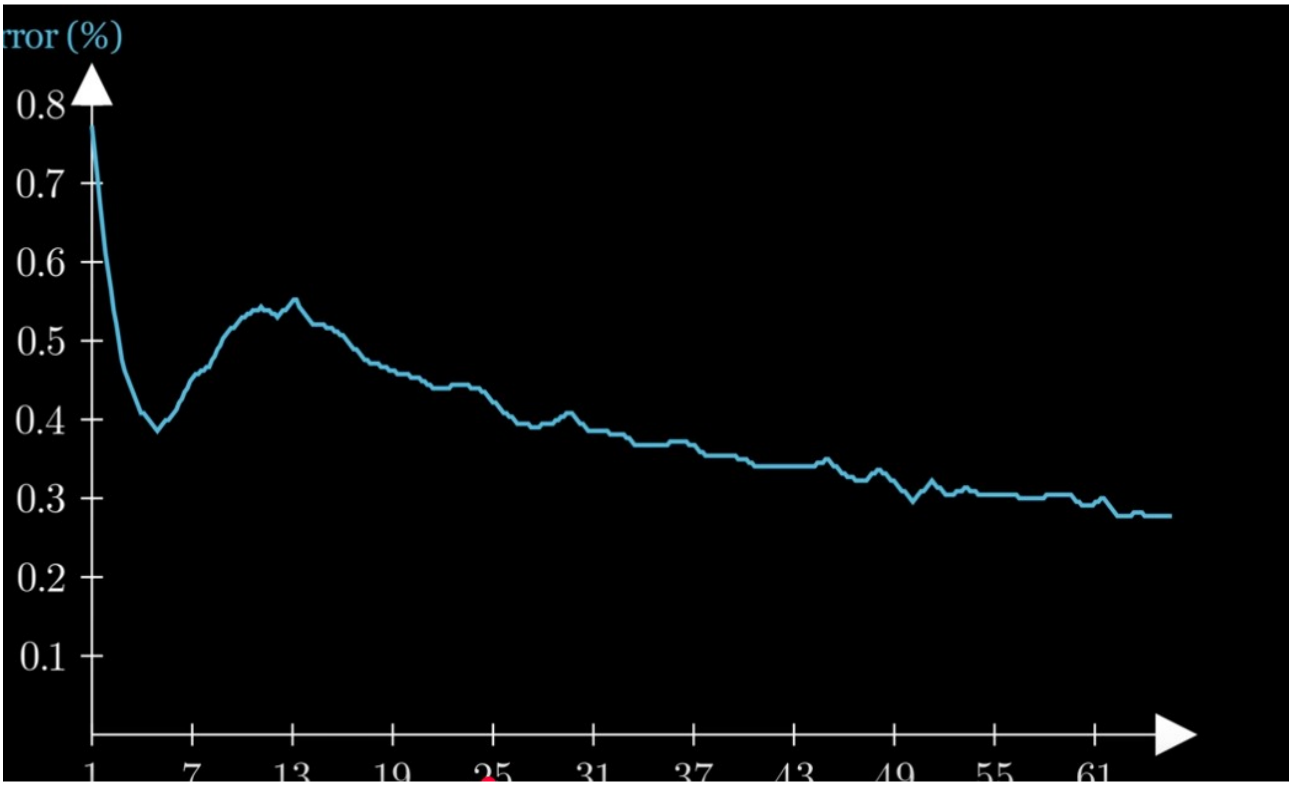


As we can see, our training data error reaches 0. That's because our model has memorized the entire dataset. But our test data error tells us a whole different story.

Our testing error didn’t just stay the same it increased, which means our model is way worse. So by using 2 different data sets we can determine when does over fitting occurs and we can avoid it

Solution 2 (for LLMs):

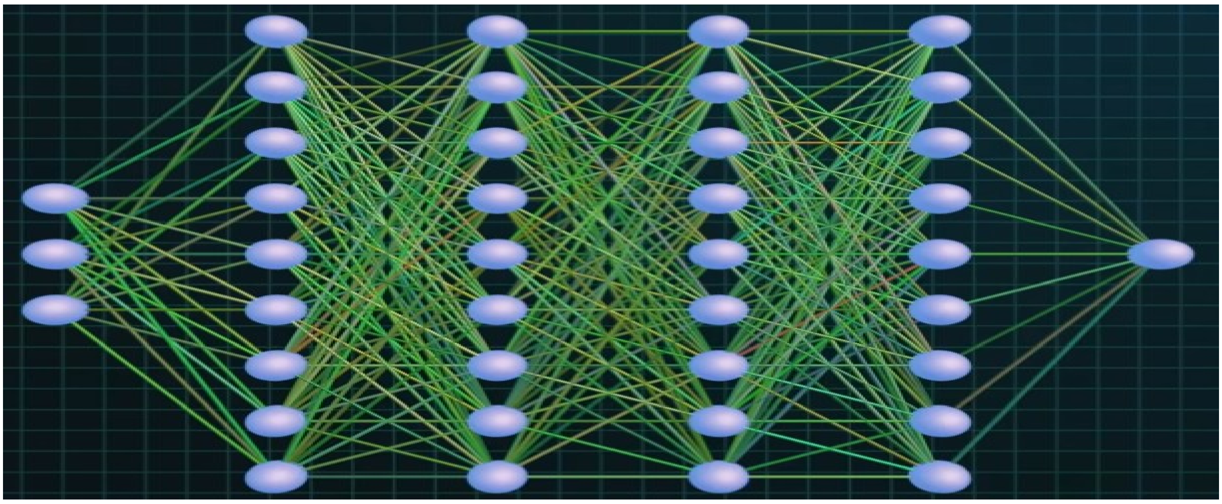
Well, by just throwing more neurons at the problem – I know, I know, I just told you throwing more neurons at the problem could make it a lot worse but I already warned you, AI scaling is very unintuitive.



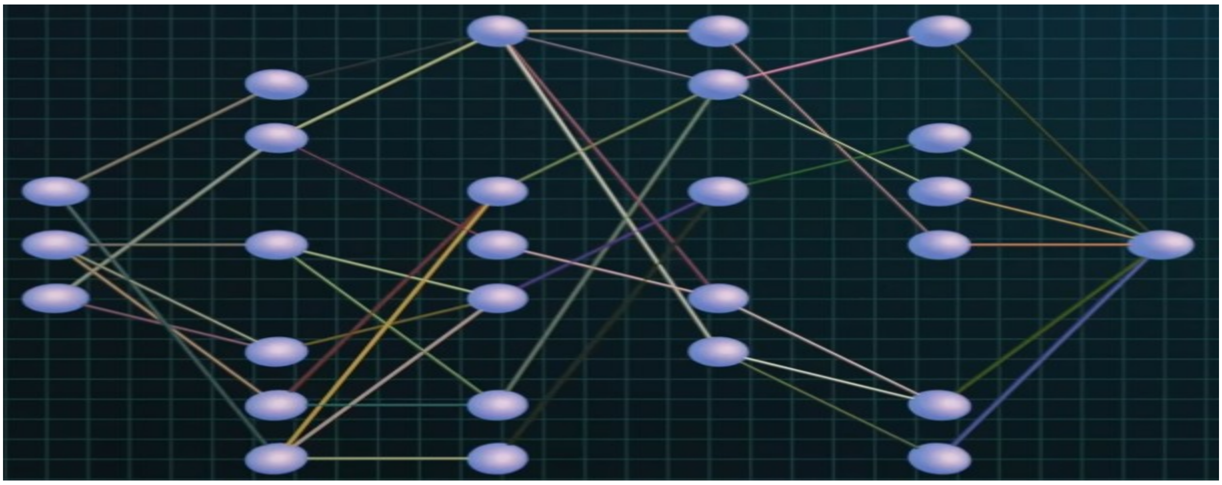
Y is the error % **X is the amount of neurons per layer**

But why? Why would the accuracy increase out of nowhere? Well, it's because most of our large NNs are useless. Most of our neurons are useless. Most of the weights in LLMs that are close to 0 don’t really do much, which means they are neurons that are used to memorize the data. If we strip all weights and neurons that have a value that is very close to 0, we can remove up to **90% of the neurons without affecting the testing error %.** Yes, **more than 90% of current LLMs' neurons are completely useless!** So, when we reach peak testing error, it means our model has memorized the full dataset, and the extra neurons we add will start learning the dataset instead of memorizing it. And the more neurons we add from that point, the bigger the likely hood of learning neurons that will learn the dataset instead of memorizing it!

lets take this LLM



if we strip it down to



It will still have the same accuracy and testing error % as the model Above !

(5.3) LLMs resource consumption:

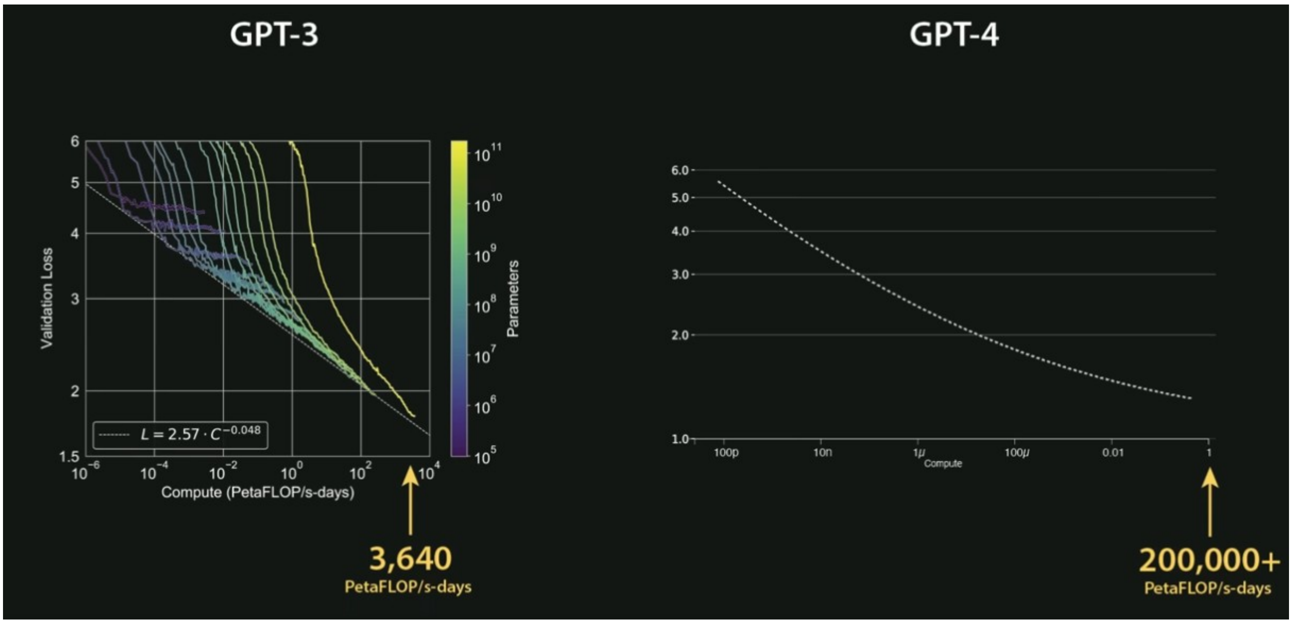
Well, if you try to train an LLM or even run one, you'd realize how resource intensive they are. They need terabytes of training data and terabytes of memory and terawatts of power. Which will only increase with their size

Solution:

The best thing to do is instead of training one large LLM that does everything, train smaller LLMs that are good at a particular thing. This way, you can get a way better, cheaper, and more accurate LLM. It's kind of like you have the choice between training 1 person to be an Olympic math whiz, wrestling champion, and also the best violinist, or training 3 people: one in math to become a math champion, one in wrestling, and the other in violin. Of course, training 3 people at 3 different things would be better. The same could be said for AI. You would need way fewer resources, way less power and data to train them. DeepSeek already tested this with great results, but this concept could be scaled a lot more!

(5.4) Diminishing Returns:

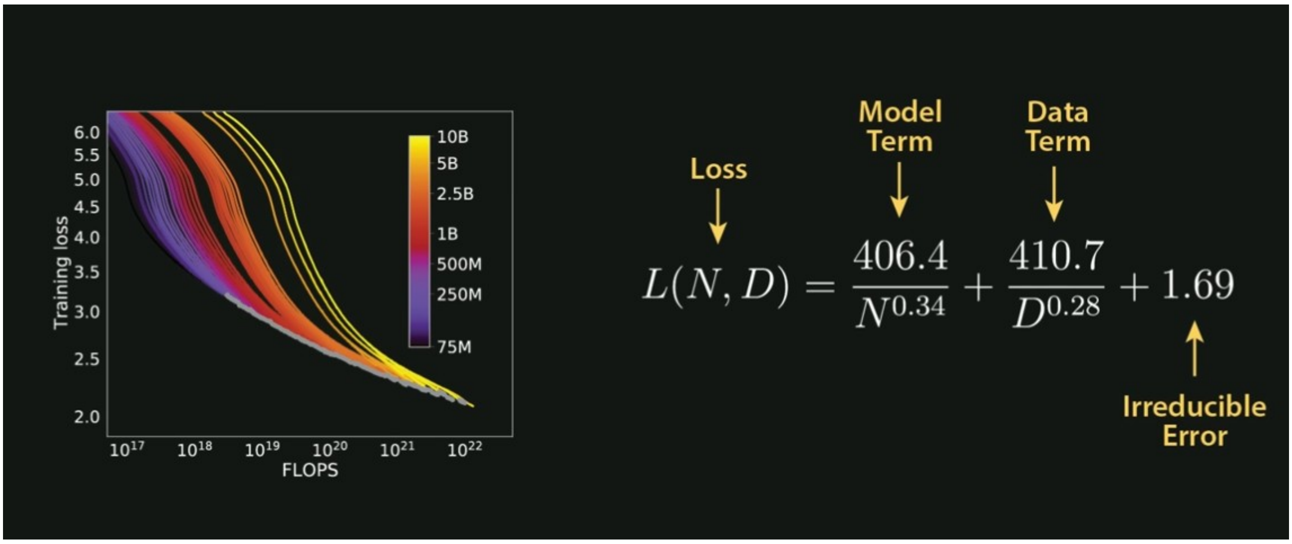
it's hard to get the exact number since OpenAI, ironically, is a closed-source company. However, it has been estimated that ChatGPT 3.5 was trained on 175 billion parameters, while ChatGPT 4 was trained on around 1.8 trillion. So, with approximately ten times the parameters, it must have been at least ten times better, right? **Well, in fact, it was barely better.**



Y is losses (less is better) **X is how much resources the model took**

A computer with a speed of one petaflop can perform one quadrillion (1,000,000,000,000,000) basic arithmetic operations on floating-point numbers every single second. So, training ChatGPT 4 required 25,000 Nvidia GPUs running for more than three months, and each one of those GPUs costs around $20,000!

which is insane for such small improvement



Y is losses (less is better) **X is how much resources the model took**

This is the scaling law of AI. Using this law, OpenAI was able to predict AI model performance ranging from 10−8 petaflop-days to 200,000 petaflop-days.

This is the scaling law of AI. Using this law, OpenAI was able to predict AI model performance ranging from 10−8 petaflop-days to 200,000 petaflop-days

As we can see in the graph, going from 75 million to 2.5 billion gave us very good results but going from 2.5 billion to 10 bullion parameters didn't lower the y-axis losses by a significant amount, even though it took far more resources to train. This really puts into perspective how significant the diminishing returns problem is in LLMs.

The scaling law presented in the image aims to predict the **training loss** (L(N,D)) of a large language model based on two key parameters:

- **N (Model Size):** This represents the number of **parameters** in the neural
- **D (Dataset Size):** This represents the **amount of training data**
- **Irreducible Error (1.69):** This constant term represents a baseline level of error that the model is predicted to never go below, no matter how large the model or how much data it's trained on. **It suggests there are inherent limitations to the learning process or the data itself.**

Conclusion

NNs are universal function approximators. All they do is take input and output data and try to find the relationship between them. They can go very far; we can scale this concept to make LLMs, image generation, and classification models, etc.

LLMs will also get a lot smaller with time. They won't get more accurate by a huge margin, but they will be a lot easier and cheaper to train and run.

With Dimensioning returns AGI isn't around the corner or anywhere close in the next 10 years.

Hardware advancements are a major driver in AI advancement, so if we remain stagnant on the hardware front, the same thing will happen on the AI front.

Further Development or Research

1. **Delving deeper into the mathematics of AI, especially matrix and tensor operations.**
2. **Testing out activation functions and exploring them in greater detail.**
3. **getting a large language model (LLM) and attempting to shrink it down by eliminating useless neurons and weights.**
4. **Training small LLMs and combining them into one very efficient and cheap LLM (similar to DeepSeek's R1).**
5. **Investigating the optimal number of neurons for general neural networks and identifying when overfitting typically begins to occur (getting more in depth about overfitting).**