Artificial Intelligence Strategies, Paths and Dangers

Artificial intelligence (AI), the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. AI is defined as machine intelligence or intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans. The term AI is often used to describe machines that mimic human cognitive functions such as learning, understanding, reasoning or problem-solving (Russell and Norvig 2016). A field of science that studies about Human cognitive function is called Cognitive Science. **It is** the **scientific** study of the human mind. The study of thinking, language, consciousness, learning, and mental representations. It **is** a highly interdisciplinary field, combining ideas and methods from psychology, computer **science**, linguistics, philosophy, and neuroscience.

Artificial intelligence (AI), refers to the replication or simulation(inspired) of human intelligence in machines that are programmed to think like humans and mimic their actions. It's a way of making machines mimic the BRAIN. The human brain does so many fascinating things , the brain can learn to see , can learn to process images , can learn to hear , can learn to do Math and Calculus. So the brain does so many amazing things , so if we can fully understand how the really the brain works , then there might be things in there for us to copy and being able to model it inside a computer to end up with human level intelligence and use it for our machine learning systems in order to solve complex problems of the universe.

Question:-

1) How do we really understand how the human brain fully works before we model it in to this computer systems ?

2) How do I model the human's brain in to this computer systems ?

3) A) What is the use of mimicking the human's brain

B) Or do we have these much Complex problems that needs much effort and science that the traditional approaches can provide?

C) Why is it really hard to use Traditional Programming Paradigms(bunch of if-else codes) to solve these complex problems ?

Answer:-

1) If we are going to say that a given program thinks like a human, we must have first understand how humans think , how the human brain works ? We need to get inside the actual workings of human minds , we need to understand the cognitive ability of humans ? There are three ways to do this, these are: -

i) Through introspection – trying to catch our own thoughts as they go by , it is self-examination, analyzing yourself, looking at your own personality and actions, and considering your own motivations. An **example of introspection** is when you meditate to try to understand your feelings.

ii) Through psychological experiments – observing a person in action. Experimental psychology is concerned with testing theories of human thoughts, feelings, actions, and beyond – any aspect of being human that involves the mind.

iii) Through brain imaging – observing the brain in action. **Brain imaging** methods allow neuroscientists to see inside the living **brain**. These methods help neuroscientists: Understand the relationships between specific areas of the **brain** and what **function** they serve. Locate the areas of the **brain** that are affected by neurological disorders.

2) How can we fully model the human's brain on to these computer systems ?

Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. The Multidisciplinary field called cognitive science(computer science + neuroscience + psychology + Philosophy) help us to model the human brain to the computer systems by making experts in neuroscience working together with computer scientists to develop certain simulated standard of the human brain. Example:- Artificial Neural Network developed to work on the same manner of the Biological Neural networks.

Now a days cognitive science are now trying to understand how the human brain really works by using Artificial Intelligence themselves by brings together computer models from AI(Like ANN) and experimental techniques from psychology or neuroscientists to try to construct more and more precise and testable theories of the workings of the human mind than ever. I mean Neuroscientists have been working for so many years to understand how the human brain really works ? But why are we seeing so much progress recently ? This is because Neuroscientists are making Artificial Intelligence better and at the same time Artificial Intelligence are making Neuroscience better.

3) A) why do we need to mimic the Human Brain ?

let's take a quick look at a biological neuron. Neurons are un-usual looking cells mostly found in animals brain which has a number of input wires called Dendrite(accept input from other locations) and also has an output wire called Axons(to send signals or messages to other neurons) with a cell body containing the nucleus. Neurons are generally a computational units that get a number of input wires with their dendrites , so some computations with their Nucleus and send outputs with various Axons to other nucleus. Near its extremity the axon splits off into many branches called telodendria, and at the tip of these branches are minuscule structures called synaptic terminals (or simply synapses), which are connected to the dendrites or cell bodies of other neurons. One output of the Axons will connect to the input wire Dendrites of other Neurons.

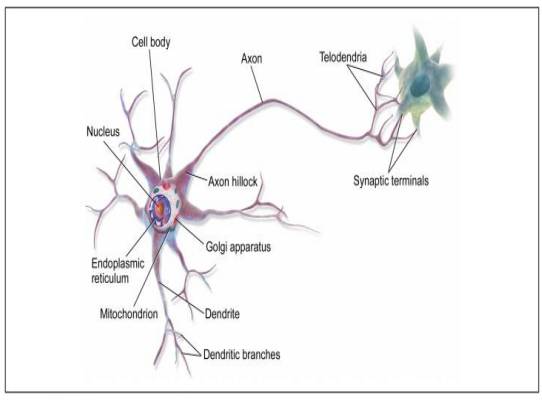


Figure 10-1. Biological neuron[4]

There are about 100billion(10^11) Neurons on our brain. Each neuron is typically connected to thousands of other neurons, so that it is estimated that there are about 100 trillion (= 10^14) synapses within the brain. Biological neurons produce short electrical impulses called action potentials (APs, or just signals) in order to communicate with each other and travel along the axons and make the synapses release chemical signals called neurotransmitters. What is the advantage of Synaptic Terminals ? A bulb at the end of an axon in which neurotransmitter molecules are stored and released to the Dendrite of the next neuron When a neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires its own electrical impulses (actually, it depends on the neurotransmitters, as some of them inhibit the neuron from firing).. Each neuron can be viewed as a separate processor, performing a very simple computation: This makes the brain a massively parallel computer made up of 10^11 processing elements(The parallelism refers to the fact that each node(Neurons) is conceived of as operating independently and concurrently (in parallel with) the others, and the "knowledge" in the network is distributed over the entire set of weights, rather than focused in a few memory locations as in a conventional computer). If that is all there is to the brain, then we should be able to model it inside a computer and end up with human intelligence or capacity inside a computer.

One thing that is probably fairly obvious is that one neuron isn't that interesting. It doesn't do very much, except fire or not fire when we give it inputs. In fact, it doesn't even learn. If we feed in the same set of inputs over and over again, the output of the neuron never varies—it either fires or does not. So to make the neuron a little more interesting we need to work out how to make it learn, and then we need to put sets of neurons together into neural networks so that they can do something useful.

Thus, individual biological neurons seem to behave in a rather simple way, but they are organized in a vast network of billions, with each neuron typically connected to thousands of other neurons. Highly complex computations can be performed by a network of fairly simple neurons. In the context of Machine Learning, the phrase "neural networks" generally refers to ANNs, not BNNs

B) Do we have that much Complex problems that needs much effort and Science than the Traditional approaches can provide ?

I think we probably agree that we are living an entirely exciting time of science , I like to say we are moving from the discovery of the building blocks of genes and atoms and bits to building first Quantum Computers , Gene Technologies and of course AI and Machine Learning. All of these technological advancements have been made by Humans those who used only 10% of their brains capacity (cause an average person uses 10% of his brains capacity.) All of these Highly complex computations performed on this earth is achieved by a network of fairly simple neurons in the brain of Humans (only 10% network of neurons from the entire 10^11).

So By Using These Does Science achieved all of the complex problems that we faced ? No , All of the scientific Knowledge that we have currently don't even know one atom in it's entirety , we know how to use them , we know how to break them , we know how to fuse them , we know how to make them explode but we don't know what the hell atom is by itself . we don't know nothing about the source for all the matters on the entire universe. We are just using them but we don't know them by themselves. Why do we need to know ? Because the Goal of Scientists is to make Humans the dominant creature on the entire universe , The Goal Of Scientists is to make this world a better place to live. So if we can't understand the universe well , then we can't Guarantee Humans and the world in General. So Now why do we I need Artificial Intelligence ? Because Some Scientists believe that the clear manifestation of this universe is not in Mars or any other inner / outer Space , they guessed that it is found on the Humans brain , so if we understand how the brain works then there might be things in there for us to copy and being able to model it inside a computer to figure out complex patterns of the universe. Why we are modeling it? Because Humans by themselves can not utilize their brains Fully.

C) Why is it really hard to use Traditional Programming Paradigms(bunch of if-else codes) to solve these complex problems ?

It's a way of making machines mimic the BRAIN. The human brain does so many fascinating things , the brain can learn to see , can learn to process images , can learn to hear , can learn to do Math and Calculus. So the brain does so many amazing things , if we want to mimic the brain we need to write a lots of pieces of software's to all of its features , so writing many lines of code does not yield a good hypothesis or prediction. Why ?

Consider how you would write a spam filter using traditional programming techniques

1) First you would consider what spam typically looks like. You might notice that some words or phrases (such as "4U," "credit card," "free," and "amazing") tend to come up a lot in the subject line. Perhaps you would also notice a few other pat- terns in the sender's name, the email's body, and other parts of the email.

2) You would write a detection algorithm for each of the patterns that you noticed, and your program would flag emails as spam if a number of these patterns were detected.

3) You would test your program and repeat steps 1 and 2 until it was good enough to launch.

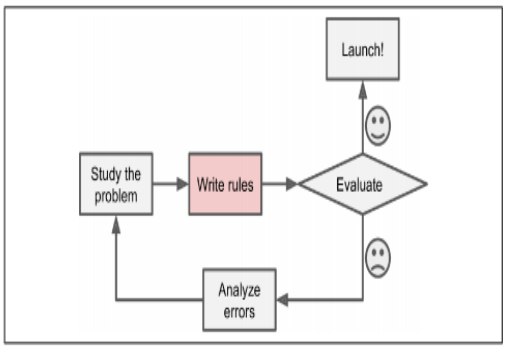


Figure 1-1. The traditional approach

Since the problem is difficult, your program will likely become a long list of complex rules—pretty hard to maintain. In contrast, a spam filter based on Machine Learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples (Figure 1-2). The program is much shorter, easier to maintain, and most likely more accurate.

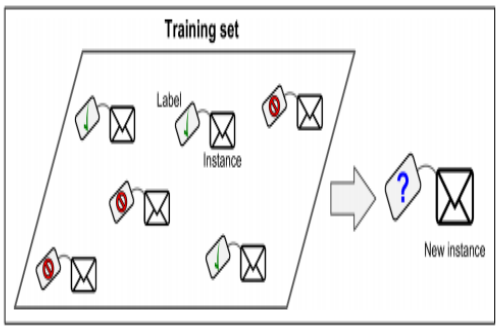


Figure 1-5. A labeled training set for spam classification (an example of supervised learning)

What if spammers notice that all their emails containing "4U" are blocked? They might start writing "For U" instead. A spam filter using traditional programming techniques would need to be updated to flag "For U" emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.

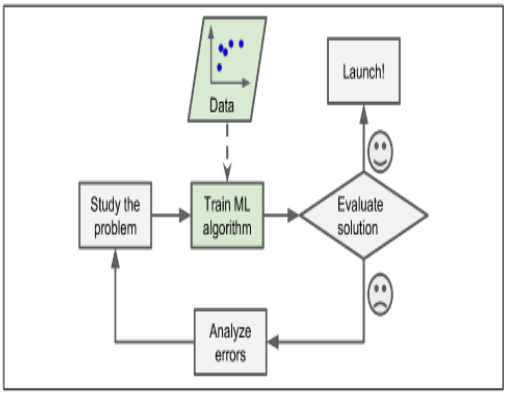


Figure 1-2. The Machine Learning approach

In contrast, a spam filter based on Machine Learning techniques automatically noti- ces that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention.

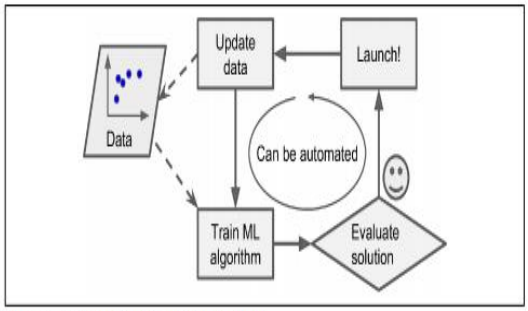


Figure 1-3. Automatically adapting to change

How can the machine Detect unusual Spams and it starts flagging them without your intervention ? There is a technique in machine learning called Anomaly Detection

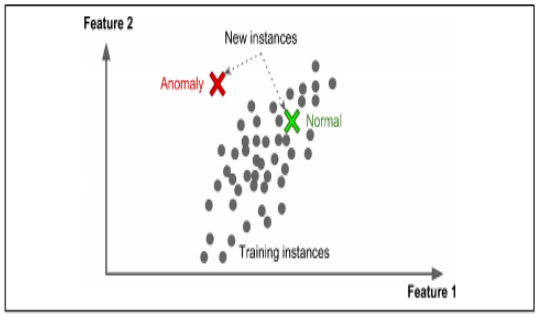


Figure 1-10. Anomaly detection

When these Algorithm sees a new instance, it can tell whether it looks like a normal one or whether it is likely an anomaly.

Another area where Machine Learning shines is for problems that either are too complex for traditional approaches or have no known algorithm. For example, consider speech recognition. Say you want to start simple and write a program capable of distinguishing the words “one” and “two.” You might notice that the word “two” starts with a high-pitch sound (“T”), so you could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos—but obviously this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages. The best solution (at least today) is to write an algorithm that learns by itself, given many example recordings for each word.

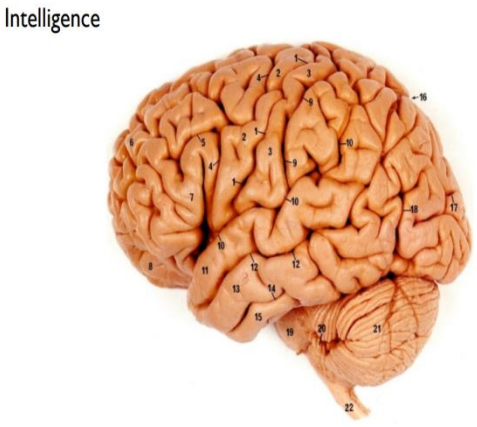
Using traditional programming you may build capabilities to copy, rename, delete the file but it’s hard to know the content in that file. I mean there may not be a known algorithm to solve this kind of complex problems , even if there is it will be a long list of rules which will be hard to maintain.

Machine Learning/Deep Learning are now today understanding the content in the image , Natural Language , Speech ranging from simple phones to the autonomous self driving cars. And these can be achieved by giving row inputs(like pixels in the image) , transform them in to useful representations and then extract

higher level features (such as shapes and edges of the image) that capture the complex concepts of combining smaller and smaller pieces of information to solve challenging tasks such as image classification.

Since the development of the digital computer in the 1940s, it has been demonstrated that computers can be programmed to carry out very complex tasks—as, for example, discovering proofs for mathematical theorems or playing chess—with great proficiency. Still, despite continuing advances in computer processing speed and memory capacity, there are as yet **no** programs that can match human flexibility(Human ability) over **wider domains** or in tasks requiring much everyday knowledge. AI is also about developing computer programs to solve complex problems by applications of process that are analogous to human reasoning process. Some of these programs have attained the performance levels of human experts and professionals in performing certain **specific** tasks(on performing single tasks not combinations).

What Is Intelligence?



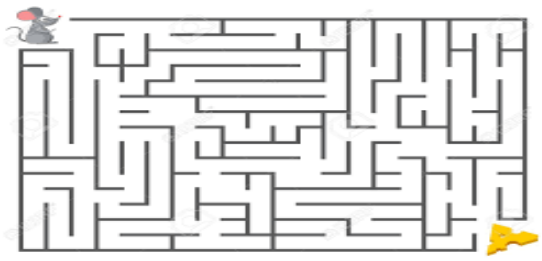
Understanding of AI requires understanding of related terms like intelligence and knowledge, for example, what's the term "Intelligence" refers to. Many of us exactly know what it is, but we won't exactly able to define it. Is it something tangible? We all know that it does exist, but what actually it is. Some of us will attribute intelligence to living beings and would be of the view that all living species are intelligent. So, can we say that Intelligence is a trait of some living species? But, how about plants and trees, they are living species and can we say they are intelligent too ?

Intelligence is All but the simplest human behavior is ascribed to intelligence , Psychologists generally do not characterize human intelligence by just one trait but by the combination of many diverse abilities. Research in AI has focused chiefly on the following components of intelligence :learning, reasoning, problem solving, perception ,and using language , the ability to give inference. Let's try to understand this phenomena of intelligence one by one.

Problem solving

Problem solving, particularly in artificial intelligence, may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution.(Mostly Reinforcement Learnig) Problem-solving methods divide into special purpose and general purpose. A special-purpose method is tailor-made for a particular problem and often exploits very specific features of the situation in which the problem is embedded. In contrast, a general-purpose method is applicable to a wide variety of problems. One general-purpose technique used in AI is means-end analysis—a step-by-step, or **incremental**, reduction of the difference between the current state and the final goal. The program selects actions from a list of means—in the case of a simple **robot** this might consist of PICKUP, PUTDOWN, **MOVEFORWARD, MOVEBACK, MOVELEFT, and MOVERIGHT—until the goal is reached.**

Consider Figure 1.1, a maze search by mouse to find a piece of cheese located at the bottom right corner, in which it starts from Top left.



This problem can be considered as a common real life problem which we deal with many times in our life, that is finding a path may be to a university, to a friend house, to a market or in the example to the piece of cheese. The mouse tries various paths as shown by arrows and can reach to the cheese through more than one paths, in other words the mouse finds more than one solutions to the problem. Now the mouse was intelligent enough to find a solution to the problem at hand, hence the ability of problem solving demonstrates intelligence.

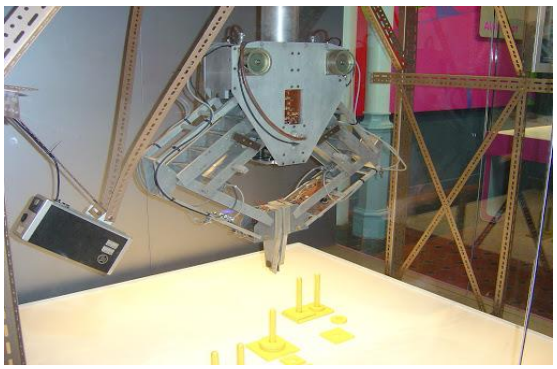
Perception

Perception in Artificial Intelligence is the way of perceiving the environment. It is a process of interpreting vision, sounds, smell, and touch. Perception helps to build machines or robots that react like humans. Perception is a process to interpret, acquire, select, and then organize the sensory information from the physical world to make actions like humans. The main difference between AI and robot is that the robot makes actions in the real world.

At present, artificial perception is sufficiently well advanced to enable optical sensors to identify individuals(objects), autonomous vehi cles to drive at moderate speeds on the open road, and robots to roam through buildings collecting empty soda cans.

Perception helps to build machines or robots that react like humans. Perception is a process to interpret, acquire, select, and then organize the sensory information from the physical world to make actions like humans. One of the earliest systems

to integrate perception and action was FREDDY, a stationary robot with a moving television eye and a pincer hand. FREDDY was able to recognize a variety of objects and could be instructed to assemble simple artifacts, such as a toy car, from a random heap of components.



Example :- Let see one last thought, a lot of us regularly watch television and consider that you switch off the volume of your TV set. If you are watching a lecture you will somehow perceive that the person standing in front of you is not singing a song or anchoring a musical show or playing some sport. Then just by observing the sequence of images of the person you are able to perceive meaningful information out of the video, because our intelligence helped you to perceive and understand what was happening on the TV. Hence the ability to understand and perceive demonstrates intelligence.

Thinking , Memory Manipulation , Numerical Processing ability :-

Example :- Let's find the next value in a sequence numbers 1, 3, 7, 13, 21, ? Just to help you out "add the next even number", that is if we add 2 to 1 we get 3, then add 4 to 3 we get 7, then add 6 to 7 we get 13, then add 8 to 13 we get 21 and finally add 10 to 21 we'll get 31 as the answer. This implies process of finding answers to questions requires a little bit intelligence. Or the characteristic of intelligence comes in when we try to solve problems, because we checking various ways and different combinations. All this thinking, memory manipulation capability, numerical processing ability and a lot of other things add to one's intelligence.

The ability to Think, To plan , and To Schedule

Example : An example that you are experiencing, that is to look at the timetable and go to attend the respective class. Not even caring that how that time table was actually developed, it may be simple to develop a timetable for few courses. But it gets tougher and tougher in cases where we have 100s of students studying in different classes, where we have only a few rooms and limited time to schedule all classes. The person who makes the timetable has to look into all the time schedule, availability of the teachers, availability of the rooms, and many other things to fit all the items correctly within a fixed span of time. He has to look into many expressions and thoughts like "If room A is free AND teacher B is ready to take the class AND the students of the class are not studying any other course at that time" THEN "the class can be scheduled". This is a fairly simple one, but the problem gets more and more

complex we add more and more parameters, for example if we were to consider that teacher B might teach more than one course and he might just prefer to teach in room C and many other things like that. We are pretty much sure than none of us had ever realized the complexity that the developer goes through while developing schedules for our classes. All this information has to reside in the developer's brain and his intelligence helps him to create such a schedule; hence the ability to think, to plan and to schedule demonstrate intelligence.

Correct and efficient memory as well as information manipulation

Example 4: Consider a person goes to a doctor and tells him that he is not feeling well, then the doctor asks a few more questions to clarify the patient's situation. Again the doctor takes a few measurements to check the physical status of the patient, these measurements might just include Temperature (T), Blood Pressure (BP), Pulse Rate (PR) and others. For simplicity the doctor only checks these measurements and tries to come up with a diagnosis for the disease. Then he takes these measurements and using his previous knowledge he tries to diagnose the disease. His previous knowledge is based on rules like "If the patient has a high BP and normal T and normal PR then he is not well", "If only the BP is normal then what ever the other measurements may be the person should be healthy" and many such other rules. Here the key thing to notice is that by using such rules the doctor might classify a person to be healthy or ill and might as well prescribe different medicines to him using the information observed from the measurements according to his previous knowledge. Diagnosing a disease has many other complex information and observations involved, we have just mentioned a very toy case here. However, the doctor is actually faced with solving a problem of diagnosis having looked at some specific measurements. It is important to consider that a doctor who would have a better memory to store all this precious knowledge and better ability of retrieving the correct portion of the knowledge that able to classify the patient. Hence, correct and efficient memory as well as information manipulation also counts towards one's intelligence.

The ability to tackle ambiguous and fuzzy problems demonstrates intelligence :-

Example 5: People don't think problems in the same manner, for example what would you say if someone asks a question like "Just say something about your height" or "Are you short, medium or tall?" Even if it's extremely easy question and you might think that you are tall but your friend who is taller than you might say that NO You are not. The point being that some people might have such a distribution in their mind that people having height around 4ft are short, around 5ft are medium and around 6ft are tall. Others might have this distribution that people having height around 4.5ft are short, around 5.5ft are medium and around 6.5ft are tall. Even having the same measurements

different people can get to completely different results as they approach the problem in different fashion. Things can be even more complex when the same person, having observed same measurements solves the same problem in two different ways and reaches different solutions. But we all know that we answer such fuzzy questions very efficiently in our daily lives, our intelligence actually helps us do this. Hence the ability to tackle ambiguous and fuzzy problems demonstrates intelligence.

The ability to learn and recognize demonstrates intelligence

Example 6: Can you recognize a person just by looking at his/her fingerprint? Though we all know that every human has a distinct pattern of his/her fingerprint but just by looking at a fingerprint image a human generally can't just tell that this print must be of person X. On the other hand, having distinct fingerprint is really important information as it serves as a unique ID for all the humans in this world. Let us just consider five different people and ask a sixth one to have a look at different images of their fingerprints. We ask him to somehow learn the patterns, which make the five prints distinct in some manner. After having seen the images a several times, that sixth person might get to find something that is making the prints distinct. Things like one of them has fever lines in the print, the other one has sharply curved lines, some might have larger distance between the lines in the print and some might have smaller displacement between the lines and many such features. The point being that after some time, which may be in hours or days or may be even months, that sixth person will be able to look at a new fingerprint of one of those five persons and he might with some degree of accuracy recognize that which one amongst the five does it belong. Only with five people the problem was hard to solve, his intelligence helped him to learn the features that distinguish one finger print from the other. Hence the ability to learn and recognize demonstrates intelligence.

Dictionaries define intelligence as the ability to acquire, understand and apply knowledge, or the ability to exercise thought and reason. In other words, Intelligence is the ability to reason, to trigger new thoughts, to perceive and learn.

In AI the goal is to develop working computer systems that are truly capable of performing tasks that require high level of intelligence. Finally, a better understanding of AI is gained by looking at the component area of study that makes up the whole. These include topics like robotics, memory organization, knowledge representation, storage and recall, learning models, inference techniques (refers to the process of using a trained *machine learning* algorithm to make a prediction for unseen data's), commonsense reasoning, dealing with uncertainty in reasoning and decision making, understanding natural language, speech recognition and a variety of AI tools. This means that a True AI can be gained by compos

1) Knowledge Representation

2) Automated Reasoning
3) Machine Learning
4) Computer Vision
5) Robotics

As shown in Figure 1.2 below, the underlying thrust force behind every intelligence is knowledge. Base of the knowledge hierarchy are symbols which form means of representation. Data can be defined as a collection of mere symbols. One gets information when data are processed. Knowledge is organized information. One can also define knowledge as a piece of information that helps in decision-making.

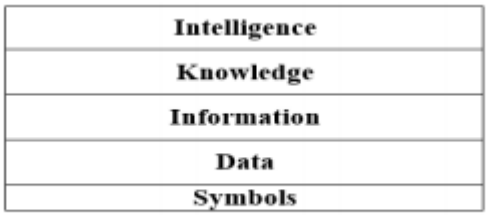


Figure 1.2: Knowledge Hierarchy

In order to achieve the manifestation of intelligence like

- Thinking
- Problem solving
- Perception
- Decision Making(inference)
- able to learn and recognize patterns , we need an organized set of information's called Knowledge , so we can say knowledge is the back bone of intelligence.

1.3. Formal Definition of Artificial Intelligence

1.3.1. Acting Humanly: The Turing Test Approach

The **Turing Test**, proposed by Alan Turing (Turing, 1950), was designed to provide a satisfactory operational definition of intelligence. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator. He suggested a test based on in distinguishability from undeniably intelligent entities-human beings. The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not. Programming a computer to pass the test provides plenty to work on. The computer would need to possess the following capabilities:-

natural language processing to enable it to communicate successfully in English (or some other human language);

knowledge representation to store information provided before or during the interrogation;

automated reasoning to use the stored information to answer questions and to draw new conclusions;

machine learning to adapt to new circumstances and to detect and extrapolate patterns.

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because *physical* simulation of a person is unnecessary for intelligence. However, the so-called **total Turing Test** includes a video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need

- **computer vision** to perceive objects, and
- **robotics** to move them about.

The above six disciplines compose most of AI, and Turing deserves credit for designing a test that remains relevant 60 years later.

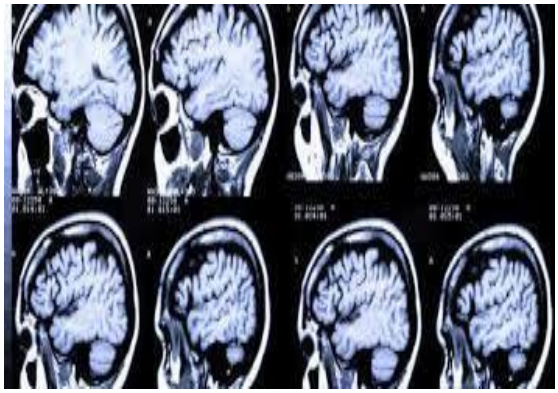
1.3.2. Thinking Humanly:

The Cognitive Modeling Approach If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds. There are three ways to do this, these are: -

i) Through introspection – trying to catch our own thoughts as they go by , it is self-examination, analyzing yourself, looking at your own personality and actions, and considering your own motivations. An **example of introspection** is when you meditate to try to understand your feelings.

ii) Through psychological experiments – observing a person in action. Experimental psychology is concerned with testing theories of human thoughts, feelings, actions, and beyond – any aspect of being human that involves the mind.

iii) Through brain imaging – observing the brain in action. **Brain imaging** methods allow neuroscientists to see inside the living **brain**. These methods help neuroscientists: Understand the relationships between specific areas of the **brain** and what **function** they serve. Locate the areas of the **brain** that are affected by neurological disorders.



Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. How ? The interdisciplinary(computer science and psychology) field of cognitive science brings together computer models from AI(Neural Networks) and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind. This means Neuroscience helped AI to build computer models and Now Ai is also helping Neuroscientist to understand the human brain more.

1.3.3. Thinking Rationally: The “Laws of Thought” Approach The Greek philosopher Aristotle was one of the first to attempt to codify “right thinking”, that is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises, for example “Socrates is a man; all men are mortal; therefore, Socrates is mortal.” These laws of thought were supposed to govern the operation of the mind and their study initiated the field called logic. Logicians in the 19th century developed a precise notation for statements about all kinds of things in the world and about the relations

among them. By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation. The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between being able to solve a problem in principle and doing so in practice. Even problems with just a few dozen facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. Although both of these obstacles apply to any attempt to build computational reasoning systems, they appeared first in the logicist tradition.

1.3.4. Acting Rationally: The Rational Agent Approach An agent is just something that acts. But computer agents are expected to have other attributes that distinguish them from mere programs, such as operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, create and pursue goals, and capable of taking on another's goals. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In the "laws of thought" approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals and then to act on that conclusion. On the other hand, correct inference is not all of rationality, in some situations there is no provably correct thing to do, but something must still be done. There are also ways of acting rationally that cannot be said to involve inference, for example recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation.

All the skills needed for the Turing Test also allow an agent to act rationally. Knowledge representation and reasoning enable agents to reach good decisions. We need to be able to generate comprehensible sentences in natural language to get by in a complex society. We need learning not only for erudition, but also because it improves our ability to generate effective behavior.

The rational-agent approach has two advantages over the other approaches. First, it is more general than the "laws of thought" approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development than are approaches based on human behavior or human thought

Evolution of Artificial Intelligence

AI is a young field and inherited ideas, concepts and techniques from various disciplines like philosophy, mathematics, psychology, linguistics, biology, etc.

- From over a long period of traditions in philosophy theories of reasoning and learning have emerged.

- From over 400 years of mathematics we have formal theories of logic, probability, decisionmaking and computation. Why Math Matters in Machine Learning

Algebra:- Will teach you how to represent and manipulate data

Statistics:- How to extract meaning from the it.

Probability :- How to make decisions under uncertainty

Optimization:- how to measure and improve performance.

- From psychology we have the tools and techniques to investigate the human mind and ways to represent the resulting theories.

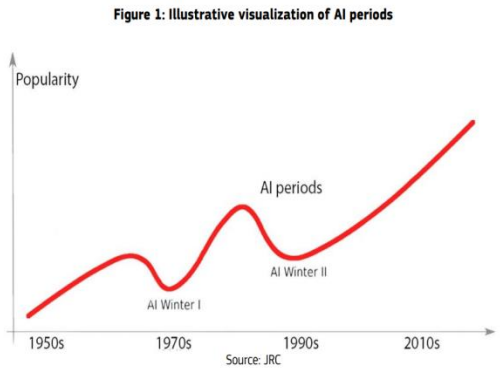
- Linguistics provides us with the theories of structure and meaning of language.

- From biology we have information about the network structure of a human brain and all the theories on functionalities of different human organs.

Finally, from computer science we have tools and concepts to make AI a reality

Brief History Of Artificial Intelligence

Artificial intelligence (AI) can have a major impact on the way modern societies respond to the hard challenges they face. Properly harnessed, AI can create a more fair, healthy, and inclusive society. Today, AI has become a mature technology and an increasingly important part of the modern life fabric. AI is already deployed in different application domains, e.g. recommendation systems, spam filters, image recognition, voice recognition, virtual assistants, etc. It spans across many sectors, from medicine to transportation, and across decades, since the term was introduced in the 1950s. The approaches also evolved, from the foundational AI algorithms of the 1950s, to the paradigm shift in symbolic algorithms and expert system development in the 1970s, the introduction of machine learning in the 1990s and the deep learning algorithms of the 2010s. Starting with the fundamental definitions and building on the historical context, this report summarizes the evolution of AI, it introduces the “seasons” of AI development (i.e. winters for the decline and springs for the growth), describes the current rise of interest in AI, and concludes with the uncertainty on the future of AI, with chances of another AI winter or of an even greater AI spring.



Modern History Of AI

We will start studying the History of Artificial Intelligence from Alan Turing but there was So many philosophy's and Greek Methodologies before that. Who is Alan Turing ? , What did he do that was so important ?

Alan Turing was a brilliant mathematician , philosopher , computer scientist born in London in 1912 , he studied at both Cambridge and Princeton Universities.

Turing took up a full time role at Bletchley park - where top secret work was carried out to decipher the military codes used by Germany and its allies. The main focus of Turing's work at Bletchley was in cracking the "Enigma" code. The Enigma was a type of encrypting machine used by the German armed forces to send message securely.

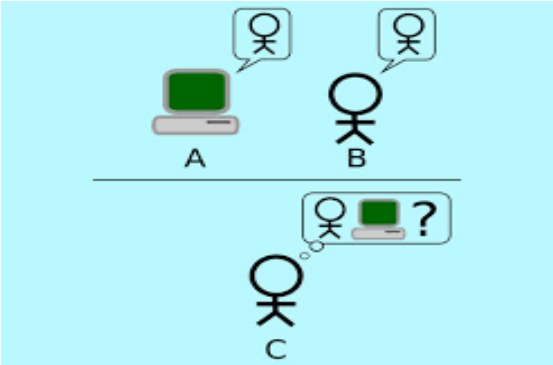
Although polish mathematicians had worked out how to read Enigma messages and had shared this information with the British But the Germans increased its security at the outbreak of the war by changing the encrypt system daily. This made the task of understanding the code even more difficult.

Turing played a key role in this , inventing – along with fellow code breakers – A machine known as the Bombe. Turing helped adapt the device originally developed by Poland to create Bombe. This device helped to significantly reduce the work of the code – breakers. From mid 1940 , German air force signals were being read at Bletchley and the intelligence gained from them was helping the war effort and save the life's of millions of people

Turing pitted machine against machine. The prototype model of his anti-Enigma "bombe", named simply Victory, was installed in the spring of 1940. His bombes turned Bletchley Park into a code breaking factory. As early as 1943 Turing's machines were cracking a staggering total of 84,000 Enigma messages each month - two messages every minute. The work of Bletchley park and Turing's role there in cracking the Enigma code was kept secret until the 1970's and the full story was not known until the 1990's. In 1952 Alan Turing was arrested for homosexuality which was then illegal in Britain. But avoided a prison sentence by accepting chemical castration. In 1954 he was founded dead by chemical Poisoning An inquest(jury that determine the cause of a person's death)ruled that it was a suicide

Alan Turing's Role In Artificial Intelligence

In 1950, Alan Turing published the milestone paper "Computing machinery and intelligence" (Turing 1950), considering the fundamental question "Can machines think?" Turing proposed an imitation game, known as the Turing test afterwards, where if a machine could carry on a conversation indistinguishable from a conversation with a human being, then it is reasonable to say that the machine is intelligent. The Turing test was the first experiment proposed to measure machine intelligence.



He just mentioned that if a system (COMPUTER) and a Human being are in the same room and if they are taking to each other , and if there is a person who is not in that room but listening to their conversation and if he can't differentiate between the system and the Human being then the system passed the test and it's real AI. So there are a lot of field of specialization to satisfy these test:

Natural Language Processing :- A field of science enables to communicate with Human languages successfully mainly English.

Knowledge-Representation:-
These field of specialization helps to store what it knows or hears.

Automated Reasoning:- To use the stored information to answer questions and to draw new conclusions.

Machine Learning :- These is the field of specialization helps to adapt to new unseen circumstances and to detect and extrapolate patterns.

Computer Vision :- Hot field of specialization helps to perceive objects Eg:-Like Tesla self driving cars.

Robotics:- To manipulate objects and move about

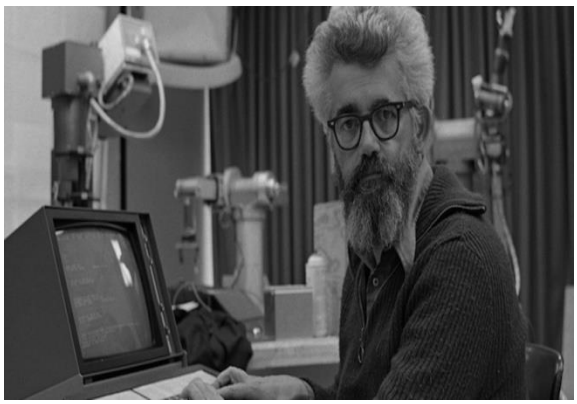
The first “AI period” began with the Dartmouth conference in 1956, where AI got its name and mission.

The Dartmouth Summer Research Project on Artificial Intelligence was a summer workshop widely considered to be the founding moment of artificial intelligence as a field of research. Held for eight weeks in Hanover, New Hampshire in 1956 the conference gathered 20 of the brightest minds in computer- and cognitive science . Arthur Samuel Was there.

Prior to the conference, Assistant Professor of Mathematics , computer scientist and Cognitive scientist at Dartmouth John McCarthy thought to gather scientists to have a summer study analogous to the summer studies that had been held about some defense problems so he thought that If ten people got together and spent the summer that substantial progress might me made and so he got three scientists

- Nathaniel Rochester From IBM
- Claude Shannon from MIT
- Marvin Minsky to join him in submitting a proposal to the Rockfeller Foundation to support a study at Dartmouth for the summer. Then in 1956 the Dartmouth conference gathered 20 of the brightest minds in computer- and cognitive science

John McCarthy was the father of Artificial Intelligence , he was an American computer scientist and Cognitive scientist who coined the name "artificial intelligence," for the first time which became the name of the scientific field.



He developed the Lisp programming language family. LISP was the first language which was widely used as an AI programming language at that time. Though McCarthy had the required tools with him to implement programs in this language but access to scarce and expensive computing resources were also a serious problem. McCarthy was the co founder of the first Ai Lab at MIT and the founder of the AI lab at Stanford university.

The Dartmouth workshop didn’t lead to any new breakthroughs, but it did all the major people who were working in the field to each other. Over the next twenty years these people, their students and colleagues at MIT, CMU, Stanford and IBM, dominated the field of artificial intelligence. The most lasting and memorable thing that came out of that workshop was an

agreement to adopt the new name for the field Artificial Intelligence, so this was when the term was actually coined.

John McCarthy quoted that " Certain things did not happen as planned Rockefeller Foundation gave us only half as much money as we asked for and most of the people who came could only came for short time , I thought we would make substantial progress towards human level artificial intelligence that summer and We didn't , Human level artificial intelligence has turned up to be a very difficult scientific problem and isn't solved yet So people shouldn't be discouraged if it takes a long time".

In 1952 , saw the first computer program which could learn as it run. It was a game which played Checkers created by Arthur Samuel. He joins IBM's Poughkeepsie Laboratory and begin working on some of the very first machine learning program , first creating programs that play checkers.

Arthur Samuel was not well enough in playing chess but playing with the machine Hundred and Thousands of times and enable the machine to learn so many ways and patterns , at last the machine wins Arthur Samuel himself.

Question:- How Arthur Samuel able to program a chess playing program without knowing how to play a chess well ? Because This is How Machine Learning Works.

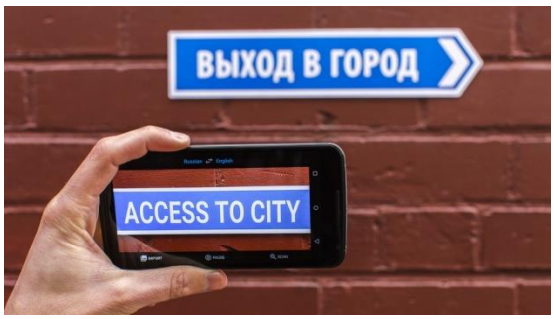
"Machine Learning is a field of Science that gives computers the ability to learn without explicitly programmed" – Arthur Samuel 1959

"Normally Humans Writes code , but on these case optimization writes code , we are creating input output specifications(if it is supervised machine learning), and then we have a lots of examples of it , and the optimization writes code and sometimes it can write code better than us , I thought that was a very new way of thinking about programming " – Andrej Karpathy (Director Of Ai at Tesla , Previous researcher @Openai , @cs Stanford student)

This what Machine Learning Experts called it "Unreasonable Effectiveness Of a data"

In 2006 The National Institute Of Science and Technology (NIST) in the united states run their 5th annual machine translation contest. That year Google Dominated the competition with a program that translated Arabic and Chinese News documents in to English. Their translation program contained no explicit grammatical rules or dictionaries. In fact, nobody on the team that created the program Spoke Arabic or Chinese . This an example of Machine Learning : Software that's trained with data

without explicitly programmed. But what was the reason that makes Google a Winner ?



To understand this , consider the experiment done by Microsoft in 2001. Researchers run a side by side test to evaluate the merits of four different approaches to Machine Learning Translation (Natural Language Disambiguation). They trained each model from scratch with the same input data , running a series of trials with varying dataset size from 100k to 1 billion words.

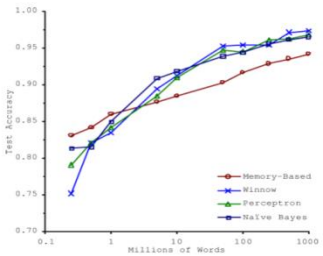


Figure 1-20. The importance of data versus algorithms[9]

As seen in the chart above , they discovered that the size of the dataset used to train the model mattered far more than the choice of ML approach And the performance differences between the models became very small as the dataset grew large. The idea that data matters more than algorithms for complex problems was further popularized by peter Norvig , In a paper titled “The unreasonable Effectiveness of Data” published in 2009.

In fact , this is how Google won the automated translation contest in 2006. They had access to more examples of correct Arabic – to – English and Chinese – to – English (i.e training data) than anyone else , because Google had just spent 8 years cataloging the internet (means collecting a linked internet addresses which talks about a specific criteria)

The first AI winter started in the 1970s, due to unfulfilled promises, vast expectations, and financial difficulties. At the same time, AI encountered impossible-to-overcome technological barriers, mainly on the limitations of computing power, memory, and processing speed.

Machine learning and deep learning (1990s – 2020s) ERA

In the 1990s–2010s, AI had addressed complex problems, providing solutions that were found to be useful in different application domains including data mining, industrial robotics, logistics, business intelligence, banking software, medical diagnosis, recommendation systems, and search engines. AI researchers began to develop and use more sophisticated mathematical tools. There was a widespread realization that many AI problems have already been worked on by researchers in fields like mathematics, economics, or operations research. The shared mathematical language allowed a higher level of collaboration with established fields and made AI a more rigorous scientific discipline.

Artificial intelligence, machine learning, and deep learning

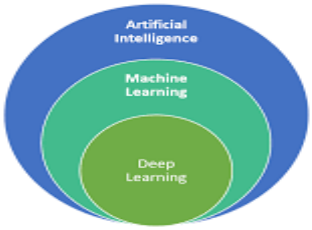


Figure 1: artificial intelligence, machine learning and deep learning Source: Nadia BERCHANE (M2 IESC), 2018)

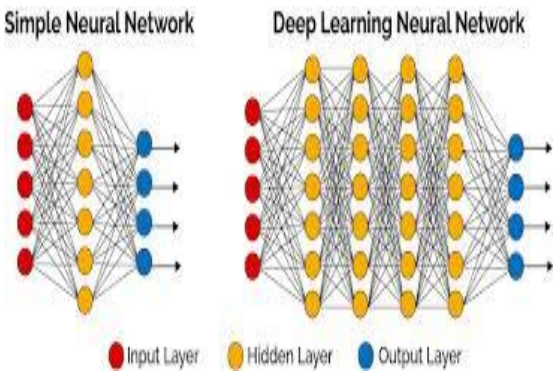
AI (Artificial intelligence) is a branch of computer science in which machines are programmed and given a cognitive ability to think and mimic actions like humans and animals. The benchmark for AI is human intelligence regarding reasoning, speech, learning, vision, and problem solving, which is far off in the future.

ML (Machine Learning) is a type of AI in which a computer is trained to automate tasks that are exhaustive or impossible for human beings. It is the best tool to analyze, understand, and identify patterns in data based on the study of computer algorithms. Machine learning can make decisions with minimal human intervention.

Comparing Artificial Intelligence vs Machine Learning, Machine learning uses data to feed an algorithm that can understand the relationship between the input and the output. When the machine finished learning, it can predict the value or the class of a new data point.

Deep-Learning(Machine Learning Deep Learning) is a computer software that mimics the network of neurons in a brain. It is a subset of machine learning and is called deep learning because it makes use of deep neural networks. The machine uses different layers to learn from the data. The depth of the model is represented by the number of layers in the model. Deep learning is the new state of the art in term of AI. In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other

“Why is it we been seeing so much rapid progress in AI recently? Why is everything suddenly work so well? Why did voice recognition suddenly get so good ? why did medical diagnosis suddenly working awesome ” That Is because of Deep Learning , instead of making our own machine learning Algorithms for solving complex problems we just started mimicking how the Human Brain really works”.



Neural Networks Reinvented

ANNs are at the very core of Deep Learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex Machine Learning tasks. An ANN is a Machine Learning model inspired by the networks of biological neurons found in our brains. However, although planes were inspired by

birds, they don't have to flap their wings. Similarly, ANNs have gradually become quite different from their biological cousins. Some researchers even argue that we should drop the biological analogy altogether (e.g., by saying "units" rather than "neurons"), lest we restrict our creativity to biologically plausible systems.

In 1943 Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, published "A logical calculus of the ideas immanent in nervous activity" in the In this paper McCulloch and Pitts tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected together. These basic brain cells are called neurons, and McCulloch and Pitts gave a highly simplified model of a neuron in their paper. The McCulloch and Pitts model of a neuron, which we will call an MCP neuron for short, has made an important contribution to the development of artificial neural networks -- which model key features of biological neurons. The MCP neuron is not a real neuron; it's only a highly simplified model. We must be very careful in drawing conclusions about real neurons based on properties of MCP neurons.

The early successes of ANNs led to the widespread belief that we would soon be conversing with truly intelligent machines. When it became clear in the 1960s(the 1 layer perceptron) that this promise would go unfulfilled (at least for quite a while), funding flew elsewhere, and ANNs entered a long winter. In the early 1980s, new architectures were invented and better training techniques were developed, sparking a revival of interest in connectionism (the study of neural networks). But progress was slow, and by the 1990s other powerful Machine Learning techniques were invented, such as Support Vector Machines . These techniques seemed to offer better results and stronger theoretical foundations than ANNs, so once again the study of neural networks was put on hold.

In 2006, Geoffrey Hinton et al. published a paper showing how to train a deep neural network capable of recognizing handwritten digits with state-of-the-art precision (>98%). They branded this technique called "Deep Learning."

Training a deep neural net was widely considered impossible at the time, and most researchers had abandoned the idea in the late 1990's(Support Machine Learning Algorithms are the most widely used until 2006). This paper(Geoffrey Hinton Hand written digits recognition) revived the interest of the scientific community, and before long many new papers demonstrated that Deep Learning was not only

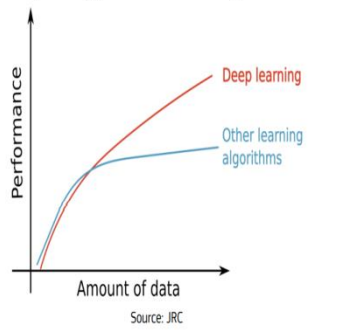
possible, but capable of mind blowing achievements that no other Machine Learning (ML) technique could hope to match (with the help of tremendous computing power and great amounts of data). This enthusiasm soon extended to many other areas of Machine Learning.

A decade or so later, Machine Learning has conquered the industry: it is at the heart of much of the magic in today's high-tech products, ranking your web search results, powering your smartphone's speech recognition, recommending videos, and beating the world champion at the game of Go. Before you know it, it will be driving your car.

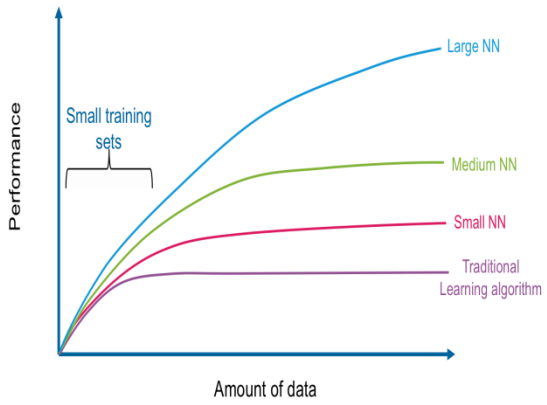
We are now witnessing yet another wave of interest in ANNs. Will this wave die out like the previous ones did? Well, here are a few good reasons to believe that this time is different and that the renewed interest in ANNs will have a much more profound impact on our lives:

There is now a huge quantity of data available to train neural networks(which is one of the largest problem in the late 1990's), and ANNs frequently outperform other ML techniques on very large and complex problems.

Figure 2: Deep Learning performance considering the amount of data



A more good way to express how Neural Networks outperformed other traditional machine learning algorithms is by looking over this depiction.



If we plot the performance of a traditional learning algorithms like support vector machine or logistic regression as a function of the amount of data you might get a curve that looks like above , where the performance improves for a while as you add more and more data but after a while the performance will be pretty much a horizontal plateaus , that means these traditional algorithms didn't know what to do with huge amounts of data. – What neural networks lead turns out that if you train a small neural net then this performance may looks like above – If we train somewhat larger Neural Networks that is called a Medium sized neural net to fall in something a little

bit better and If we train a very large neural net then it's the form and often just keeps getting better and better. So From this depiction we will understood that – if you want to hit this very high level performance then you need this two things

1) To be able to train a big enough neural network in order to take advantage of the huge amount of data.

2) To be able to provide a lot of hidden units , a lot of parameters to end up with a lot of connections.

We are going to use a lower case alphabet to denote the size of my training examples this lower case M .so if we don't have a lot of training data (Look the part on the image that says small training sets) then it will be often up to the skill Feature Engineering that determines the performance , so if someone training SVM is more motivated to hand Engineer Features then the SVM algorithm could do better . so in the let region of the figure the relative ordering between the algorithms is not that well defined and performance depends much more on your skill at Engineering Features and other details of the algorithms and there is only in this big data regime very large training sets in the right that we more consistently see large neural networks dominating the other approaches.

The tremendous increase in computing power since the 1990s now makes it possible to train large neural networks in a reasonable amount of time. This is in part due to Moore's law (the number of components in integrated circuits has doubled about every 2 years over the last 50 years), but also thanks to the gaming industry, which has stimulated the production of powerful GPU cards by the mil- lions. Moreover, cloud platforms have made this power accessible to everyone.

The training algorithms have been improved. To be fair they are only slightly different from the ones used in the 1990's, but these relatively small tweaks have had a huge positive impact.

ANNs seem to have entered a virtuous circle of funding and progress. Amazing products based on ANNs regularly make the headline news, which pulls more and more attention and funding toward them, resulting in more and more progress and even more amazing products.

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples. Other tasks, that can not be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now

Types of artificial intelligence

Artificial Narrow Intelligence (ANI), often referred to as "Weak" AI is the type of AI that mostly exists today. ANI systems can perform one or a few specific tasks and operate within a predefined environment, e.g., those exploited by personal assistants Siri, Alexa, language translations, recommendation systems, image recognition systems, face identification, etc. ANI can process data at lightning speed and boost the overall productivity and efficiency in many practical applications, e.g., translate between 100+ languages simultaneously, identify faces and objects in billions of images with high accuracy, assist users in many data-driven decisions in a quicker way. ANI can perform routine, repetitive, and mundane tasks that humans would prefer to avoid. Why do we mean by ANI can perform mundane tasks of humans ? why do we need to avoid that? I guess the currently the Goal of AI is to make machines cover the daily repetitive , mundane tasks of humans and make ourselves focus on other critical creativities.

While ANI is well known by it's incapability of generalization, i.e. to re-use learned knowledge across domains, e.g., the ANI capable of image recognition cannot transfer its knowledge in the domain of speech recognition. The generalization problem is still an open question (Hernández-Orallo 2017).

ArtificialGeneral Intelligence (AGI) or "Strong" AI refers to machines that exhibit human intelligence. In other words, AGI aims to perform any intellectual task that a human being can. AGI is often illustrated in science fiction movies with situations where humans interact with machines that are conscious, sentient, and driven by emotion and self-awareness. At this moment, there is nothing like an AGI.

Artificial-Superintelligence (ASI) is defined as "any intellect that greatly exceeds the cognitive

performance of humans in virtually all domains of interest" (Bostrom 2016). ASI is supposed to surpass human intelligence in all aspects — such as creativity, general wisdom, and problem-solving. ASI is supposed to be capable of exhibiting intelligence that we have not seen in the brightest thinkers amongst us. Many thinkers are worried about ASI. At this moment, ASI belongs to science fiction.

Technological Advancements Of Deep Learning

ANNs are at the very core of Deep Learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex

- Machine Learning tasks such as classifying billions of images (e.g., Google Images , helps Google for data mining),

powering speech recognition services (e.g., Apple's Siri),

recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube , Netflix),

In 1997 IBM's Deep Blue beats the world chess champion Gary Kasprov. "Good Humans Plus A Machine Is The Best Combination".

In 2011 IBM's Watson(Watson is a question-answering computer system capable of answering questions posed in natural language , Watson was

named after IBM's founder and first CEO, industrialist Thomas J. Watson.) outplayed two humans at the popular television quiz show **Jeopardy**! The computer's opponents weren't merely two humans—they were the two all-time best **Jeopardy**! champions, ever.

Learning to beat the world champion lee sedol(18 times world go champion) at the game of Go(a 3000 years old Chinese game which is more and more complex than chess) (DeepMind's AlphaGo , 2015 , The Go Game got 10 to the power of 170 **possible** board configurations , this is makes Alpha Go to have a combination more than **the number of atoms** in the known universe(10^78 – 10^82). This makes the game of **Go** more complex than chess.) , flagging fake news , self driving cars and predicting Earth Quick and more and more.

Neural Networks And Deep Learning

Objective of the note

Recap To A Logistic Regression

- Sigmoid Function and ReLu Function
- The Relation Between Logistic Regression and Artificial Neural Network
- Why recent ANN's choose ReLu Function over Sigmoid Function

Recap To Logistic Regression

Some Regression Algorithms can be used for classification problems. Logistic Regression is basically a supervised classification algorithm used to estimate the *probability* that a given instance belongs to a *particular class*. Example:- what is the probability of a given email is flagged as a spam ?

-If the estimated probability is greater than 50% , the model predicts that instance belongs to the positive class or the class that is labeled as 1. Otherwise it predicts that it does not (Labeled it to the negative class , label 0). These is actually for *a binary classifier*.

-Since Logistic Regression is a classification Algorithm , the variable y (output) that we want to predict is a *discrete value*.

Some Examples of Classification Problems:-

Email:- Spam / Ham

Online Fraudulent Transaction – Yes / No

Tumor Size :- Malignant and Benign to Breast Cancer

Y element {0 , 1}, if it is Binary Classification

Y = 0 :- Negative Class or absence to something that we are looking for

Y= 1 :- Positive Class or presence to something that we are looking for.

Later we will see Multi Class classification problems :-

Y element {0 , 1 , 2 , 3 , ..}

Generative and Discriminative Classifiers:

The most important difference between *naive Bayes* and *logistic regression* is that logistic regression is a discriminative classifier while naive Bayes is a generative classifier. These are two very different frameworks for how to build a machine learning model.

Story:

A father has two kids, Kid A and Kid B. Kid A has a special character whereas he can learn everything in depth. Kid B have a special character whereas he can only learn the differences between what he saw.

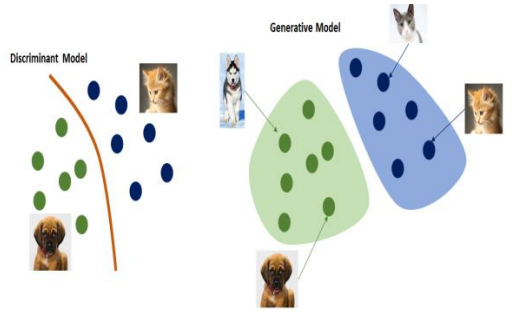
One fine day, The father takes two of his kids (Kid A and Kid B) to a zoo. This zoo is a very small one and has only two kinds of animals say a lion and an elephant. After they came out of the zoo, the father showed them an animal and asked both of them "is this animal a lion or an elephant?"

The Kid A, the kid suddenly draw the image of lion and elephant in a piece of paper based on what he saw inside the zoo. He compared both the images with the animal standing before and answered based on the closest match of image & animal, he answered: "The animal is Lion".

The Kid B knows only the differences, based on different properties learned, he answered: "The animal is a Lion".

Here, we can see both of them is finding the kind of animal, but the way of learning and the way of finding answer is entirely different. In Machine Learning, We generally call Kid A as a Generative Model & Kid B as a Discriminative Model.

Consider a visual metaphor: imagine we're trying to distinguish dog images from cat images. A generative model would have the goal of understanding what dogs look like and what cats look like. You might literally ask such a model to 'generate', i.e., draw, a dog. Given a test image, the system then asks whether it's the cat model or the dog model that better fits (is less surprised by) the image, and chooses that as its label. A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them). So maybe all the dogs in the training data are wearing collars and the cats aren't. If that one feature neatly separates the classes, the model is satisfied. If you ask such a model what it knows about cats all it can say is that they don't wear collars.



More formally, recall that the naive Bayes assigns a class c to a document d not by directly computing P(c|d) but by computing a likelihood and a prior .A generative model(Uses Bayes Theorem) like naive Bayes makes use of this likelihood term, which generative model expresses how to generate the features of a document if we knew it was of class c. By contrast a discriminative model in this text categorization scenario attempts discriminative model to directly compute P(c|d). Perhaps it will learn to assign a high weight to document features that directly improve its ability to discriminate

between possible classes, even if it couldn't generate an example of one of the classes.

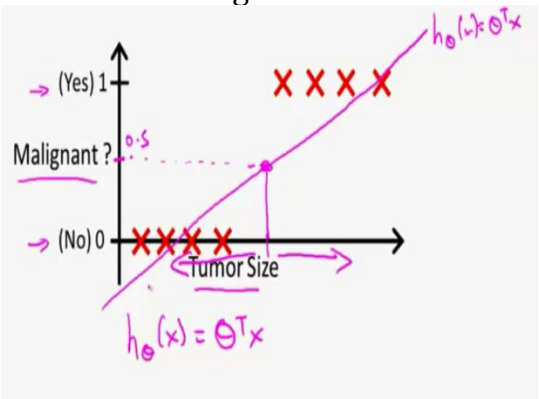
Why Linear Regression Is Not Suitable For Classification Problems

Two reasons why linear regression is: not suitable for classification problems:-

- The predicted value is continuous, not probabilistic
- Sensitive to imbalance data when using linear regression for classification.

Sensitive To Imbalanced Data which makes the Linear Regression to shift its threshold when a new data points are added

The objective of a linear regression model is to find a relationship between the input variables and a target variable. Below is our linear regression model that was trained using some dataset.

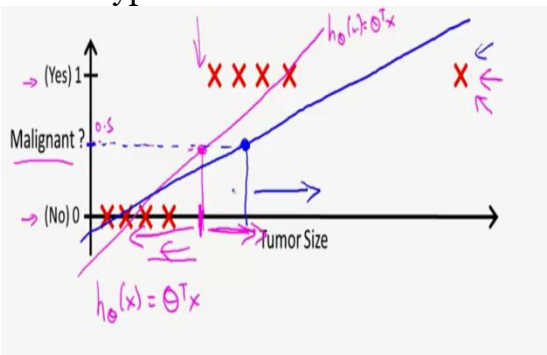


The above graph shows the best fit line for the given points. In order to make a better prediction we need something called *Threshold*. Threshold classifier classifies the output ho(x) at 0.5 (Default).

If ho(x)>=0.5 predicts as 1

If ho(x) <0.5 predicts as 0

This is a simple example and the real-world data is never this simple. So coming back, when we add another point to this dataset, our best fit line and our threshold shifts to fit that point. This makes the straight line position changed and have a new threshold position which causes a worst Hypothesis.



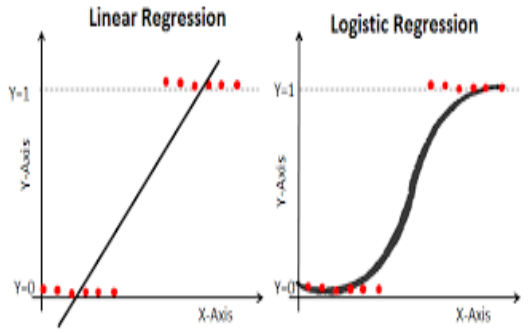
The threshold classifier will enable us to know the exact position of x(Tumor size in these case).

The Predicted Value is Continuous , Not Probabilistic

In a binary classification problem, what we are interested in is the probability of an outcome occurring. Probability is ranged between 0 and 1, where the probability of something certain to happen is 1, and 0 is something unlikely to happen. But in linear regression, we are predicting an absolute number, which can range outside 0 and 1.

In Linear Regression the hypothesis ho(x) will may be predict way more than 1 or too less than 0 even though all of our training examples are between 0 and 1. But for classification problems our probability is between 0 and 1.
But sure, we can limit any value greater than 1 to be 1, and value lower than 0 to be 0. Linear regression can still work, right?

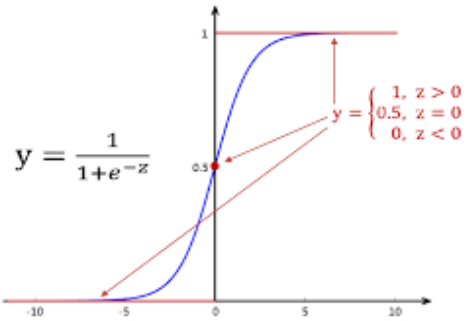
Yes, it might work, but *logistic regression* is more suitable for classification task and we want to prove that logistic regression yields better results than linear regression. So our Algorithm which is Logistic Regression can predict output between 0 <= ho(x) <=1.



How do we know that Logistic Regression is better than Linear regression , By comparing the performance RMSE of both models for same datasets.

-But the Question is , is the function of the Logistic Regression is Linear ? No, Logistic Regression uses a function called Sigmoid. It is a kind of function that comprises the hypothesis between 0 and 1 for any value of x.

.



Classification: the sigmoid

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation. Here we introduce the sigmoid classifier that will help us make this decision.

consider a single input observation x, which we will represent by a vector of features [x1, x2,..., xn] (we'll show sample features in the next subsection). The classifier output y can be 1 (meaning the observation is a member of the class) or 0 (the observation is not a member of the class). We want to know the probability P(y = 1|x) that this observation is a member of the class.

So perhaps the decision is "positive sentiment" versus "negative sentiment", the features represent counts of words in a document, P(y = 1|x) is the probability that the document has positive sentiment, and P(y = 0|x) is the probability that the document has negative sentiment.

Logistic regression solves this task by learning, from a training set, a vector of weights and a bias term. Each weight wi is a real number, and is associated with one of the input features xi . The weight wi represents *how important that input feature is to the classification decision*, and can be positive (providing evidence that the instance being classified belongs in the positive class) or negative (providing evidence that the instance being classified belongs in the negative class). Thus we might expect in a sentiment task the word awesome to have a high positive weight, and bias term abysmal to have a very negative weight. The bias term, also called the intercept, is intercept another real number that's added to the weighted inputs.

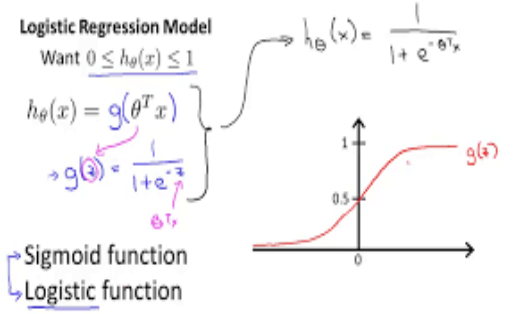
To make a decision on a test instance— after we've learned the weights in training— the classifier first multiplies each xi by its weight wi , sums up the weighted features, and adds the bias term b. The resulting single number z expresses the weighted sum of the evidence for the class.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

In the rest of the book we'll represent such sums using the dot product notation from linear algebra. The dot product of two vectors a and b, written as a · b is the sum of the products of the corresponding elements of each vector. Thus the following is an equivalent formation to

z = w· x+b This is the Hypothesis or prediction for the test set

But note that we need to force z to be a legal probability, that is, to lie between 0 and 1. In fact, since weights are real-valued, the output might even be negative; z ranges from −∞ to ∞. So we will use sigmoid function. The sigmoid function
y = 1/ 1+e ^−z takes a real value and maps it to the range [0,1].



If z -> Positive Infininty , e ^−z ->0
y = 1/ 1+0 , which is 1 //

If z -> Negative Infinity , e ^−z -> very large number
y = 1/ 1+large_Number -> 0//

To create a probability, we'll pass z through the sigmoid function, σ(z). The sigmoid function (named because it looks like an s) is also called the logistic function, and gives logistic regression its name.

$$y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$

The sigmoid has a number of advantages; it takes a real-valued number and maps it into the range of [0,1], which is just what we want for a probability.

We're almost there. If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1. To make it a probability, we just need to make sure that the two cases, p(y = 1) and p(y = 0), sum to 1. We can do this as follows:

$$\begin{aligned} P(y=1) &= \sigma(w\cdot x+b) \\ &= \frac{1}{1+\exp(-(w\cdot x+b))} \\ P(y=0) &= 1-\sigma(w\cdot x+b) \\ &= 1-\frac{1}{1+\exp(-(w\cdot x+b))} \\ &= \frac{\exp(-(w\cdot x+b))}{1+\exp(-(w\cdot x+b))} \end{aligned}$$

The sigmoid function has the property 1−σ(x) = σ(−x) so we could also have expressed P(y = 0) as σ(−(w· x+b)).

Now we have an algorithm that given an instance x computes the probability P(y = 1|x). How do we make a decision? For a test instance x, we say yes if the probability P(y = 1|x) is more than 0.5, and no otherwise. We call 0.5 the decision boundary:

$$\hat{y} = \begin{cases} 1 \text{ if } P(y=1|x) > 0.5 \\ 0 \text{ otherwise} \end{cases}$$

Hypothesis Representation

Interpretation of hypothesis output: $h_\theta(x)$ is the estimated probability that y = 1 on input x.

Formally, $h\theta(x)=P(y=1|x;\theta)$
Example :- X = [x_0 = [1
X_1] Tumor Size]

$h\theta(x)$= 0.7 //

Read us :- Tell patient that 70% chance of tumor being malignant .

$h\theta(x)$ is the estimated probability that y = 1 on input x. Formally,

$h\theta(x)=P(y=1|x;\theta)$
Also, $P(y=0|x;\theta)+P(y=1|x;\theta)=1$

therefore, $P(y=0|x;\theta)=1-h\theta(x)$

Cost Function For Logistic Regression

After training your model, you need to see how well your model is performing. While accuracy functions tell you how well the model is performing (fitness function).

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/prediction.

In the case of Linear Regression, the Cost function is –

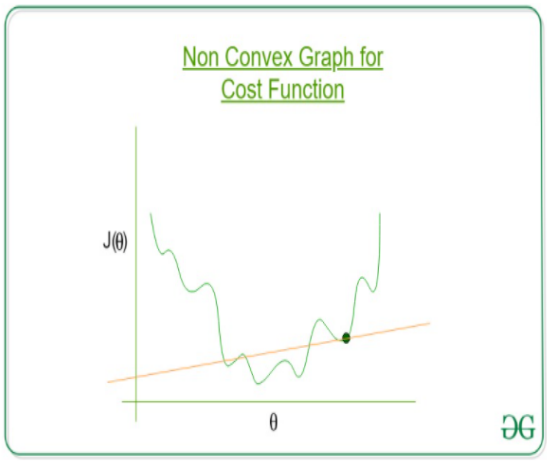
$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}[h_\Theta(x^{(i)}) - y^{(i)}]^2$$

***Cost(h*Θ(x) , y *) = ½[(h*Θ(x) - **y)²** , This is the cost function for one training example.

But for Logistic Regression

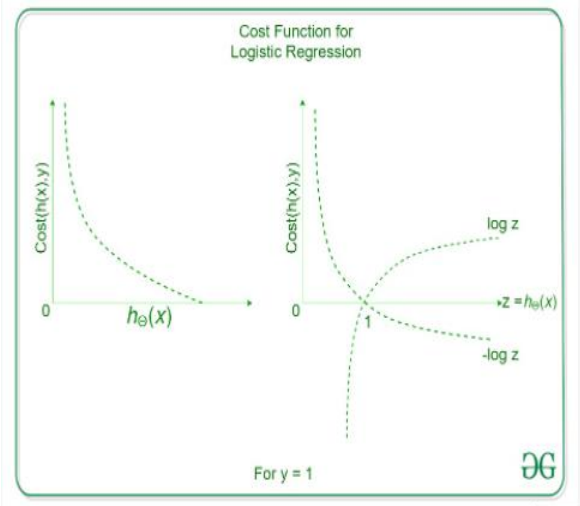
$$h_\Theta(x) = g(\Theta^T x)$$

It will result in a non-convex cost function. This results in cost function with so many local optima's which is a very big problem for Gradient Descent to compute the global optima.



$$\text{Cost}(h_\Theta(x), y) = \begin{cases} -log(h_\Theta(x)) & if \quad y = 1 \\ -log(1 - h_\Theta(x)) & if \quad y = 0 \end{cases}$$

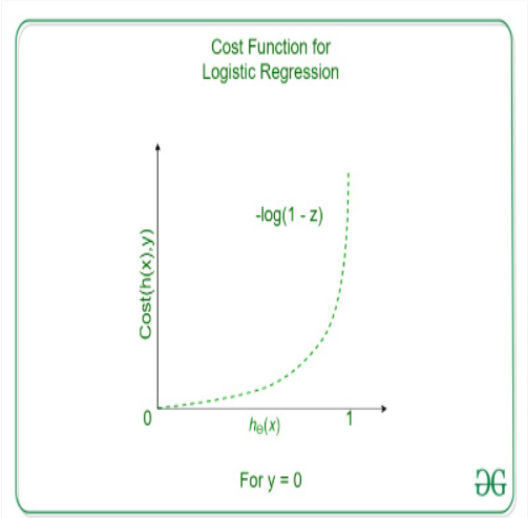
If y = 1



Cost = 0 if y = 1, $h_\theta(x) = 1$
But as,
$h_\theta(x) \to 0$
Cost -> Infinity

Since the label is y = 1 and the hypothesis is zero , so we will penalize the learning algorithm by a very large cost

If y = 0



Cost = 0 if y = 0, $h_\theta(x) = 0$
But as,
$h_\theta(x) \to 1$
Cost -> Infinity

Since the label is y = 0 and the hypothesis is one , so we will penalize the learning algorithm by a very large cost.

Simplified Cost Function and Gradient – Descent

$$\text{Cost}(h_\Theta(x), y) = \begin{cases} -log(h_\Theta(x)) & if \quad y = 1 \\ -log(1 - h_\Theta(x)) & if \quad y = 0 \end{cases}$$

But we can compress these as one

$$\text{Cost}(h_\Theta(x), y) = -y\log(h_\Theta(x)) - (1-y)\log(1-h_\Theta(x))$$

$$J(\Theta) = \frac{-1}{m}\sum_{i=1}^{m} Cost(h_\Theta(x), y)$$

When y = 1
Cost (hΘ (x) , y) = -1log(hΘ (x))

When y = 0
Cost(hΘ (x), y) = -log(1 – hΘ (x))

J(Θ) = -1/m Σ ylog(h Θ (x) + (1 – y)log(1 – h Θ (x)))

Gradient Descent For Logistic Regression

The cost function must be minimized ,which is the distance between the predicted and the true responses.

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

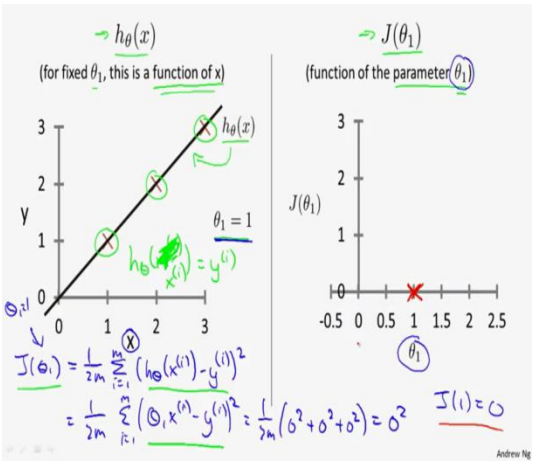
$$\theta_0, \theta_1$$

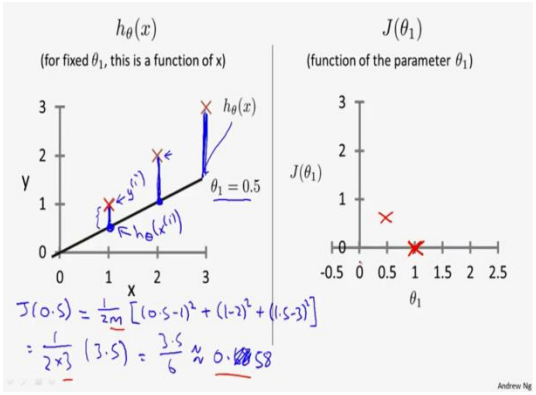
Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

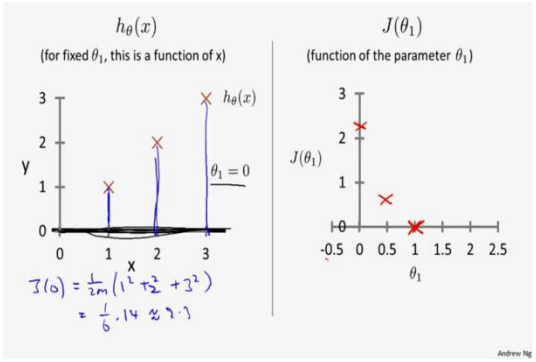
Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Iteratively Finding the parameters or the value of θ which minimizes the RMSE (Cost Function)

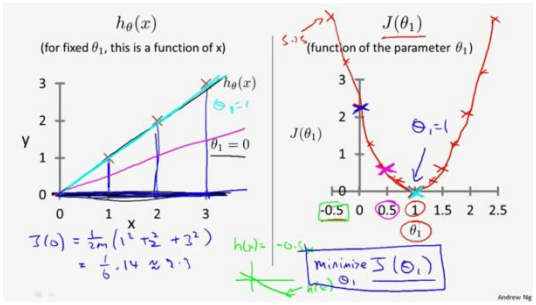




Let's try $\theta_1 = 0$



Our final shape of the cost function will be



Gradient Descent

But it is tedious to find the best cost function by iterating the parameters. So we need an algorithm to find the best parameters which can give a minimum cost function.

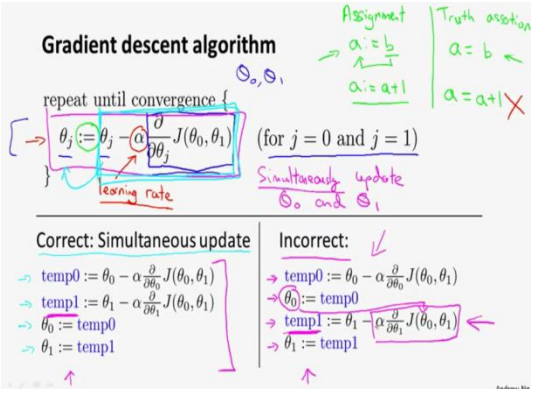
Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

Outline:

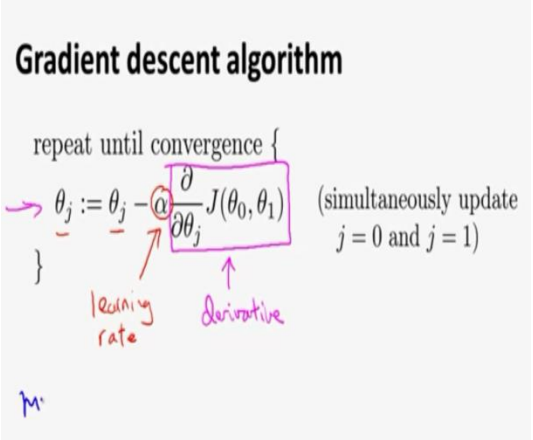
- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum

Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

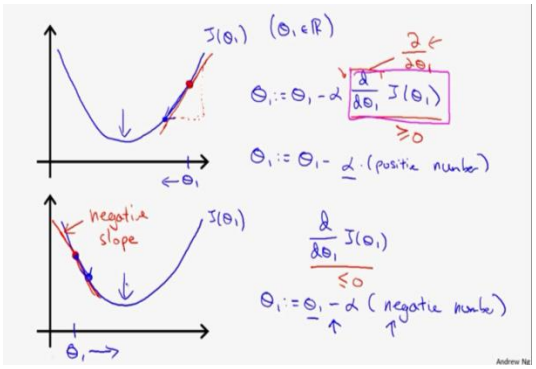


Concretely, you start by filling θ with random values (this is called random initialization). Then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum.

The Derivative Term



The derivative term simply means the slope.



The Alpha

Alpha means about how fast is going when we update our parameters.

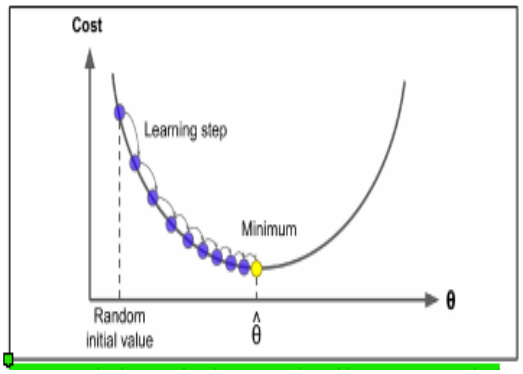


Figure 4-3. In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function; the learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum

An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyper parameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time.

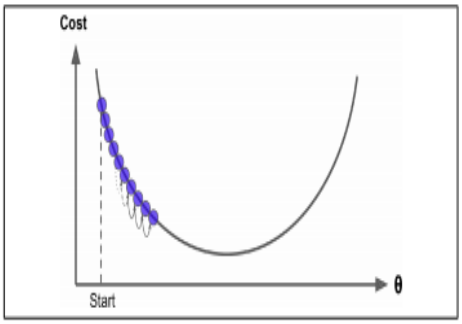


Figure 4-4. The learning rate is too small

On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution.

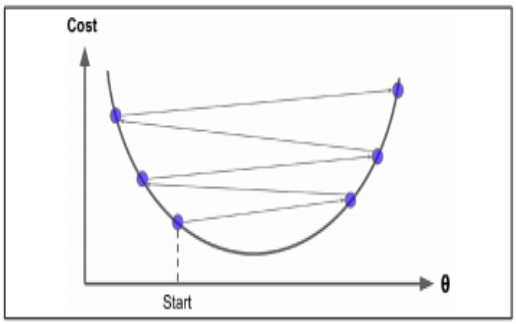


Figure 4-5. The learning rate is too large

The figure shows the two main challenges with Gradient Descent. If the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau. And if you stop too early, you will never reach the global minimum.

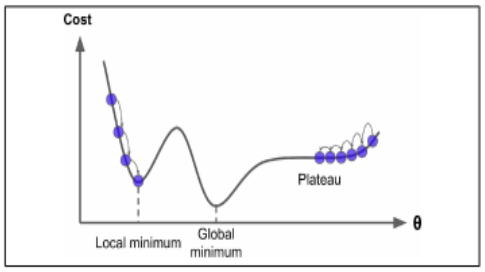


Figure 4-6. Gradient Descent pitfalls

What is the best use of Alpha

...0.001 , 0.01 , 0.1 , 1 , ...

Some range of numbers with 10 fraction difference

0.001 , 0.003 , 0.01 , 0.03 , 0.1 , 0.3 , 1 , ...

Some range of numbers with 3* difference

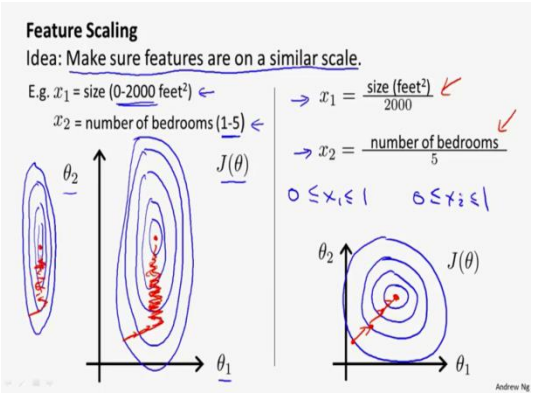
-The point is we start with some range of small numbers and check the gradient descent is converged , then take the smaller alpha which gives us a good gradient descent in that range.

Gradient Descent – Feature Scaling

In the housing price scale example
X1 = size of the house (0 – 2000ft²)

X2 = Number of bed rooms (1 - 5)

There is a big difference between x1 and x2



In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales. The Figure shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has

much smaller values than feature 2 (on the right).

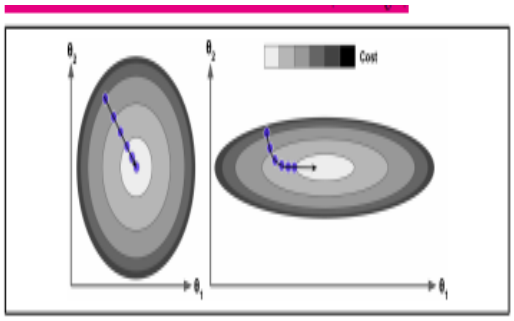


Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

If the features are not scaled , the shape of the cost function will be skewed(The image on the left) , it takes longer time to converge but if the features are scaled , it shortly goes to the minimum.

The best way to scale our features is to use a **mean normalization**
x1 = (1 - 2000ft²) - size of the house
x2 = 1 - 5 - number of the bed rooms
# use x - average instead of x

mean-Normalization = #size of the house - average / # total

mean-normalization = # number of bed rooms - average / # total

mostly features are scaled between -1 <= features(x) <= 1, but it doesn't have to be only this , which means

3 <= x < = 3 --- this is scaled
0.5 < = x < = 0.5 -- this is also scaled
-0.00001 < = x < = 0.00001 --- this is not scaled
-100 < = x < = 100 ------this is also not scaled

Regularization:-Handling Overfitting and Under Fitting

If you perform high-degree Polynomial Regression, you will likely fit the training data much better than with plain Linear Regression. For example, Figure 4-14 applies a 300-degree polynomial model to the preceding training data, and compares the result with a pure linear model and a quadratic model (second-degree polynomial). Notice how the 300-degree polynomial model wiggles around to get as close as possible to the training instances.

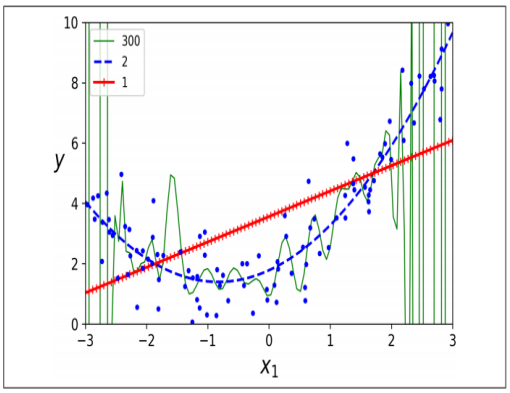
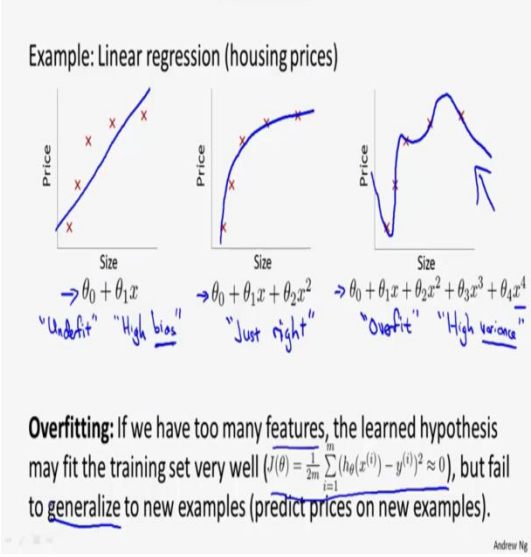
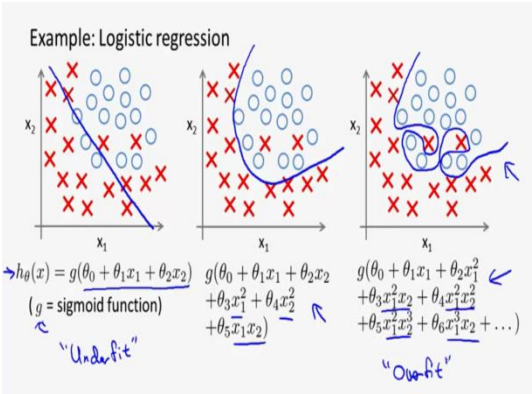


Figure 4-14. High-degree Polynomial Regression

This high-degree Polynomial Regression model is severely overfitting the training data, while the linear model is underfitting it. The model that will generalize best in this case is the quadratic model, which makes sense because the data was generated using a quadratic model. But in general you won't know what function generated the data, so how can you decide how complex your model should be? How can you tell that your model is overfitting or underfitting the data?



Overfitting and Underfitting for Logistic Regression



Before understanding on how to address overfitting and underfitting we must first understand on how to know there is overfitting or underfitting on our model ?

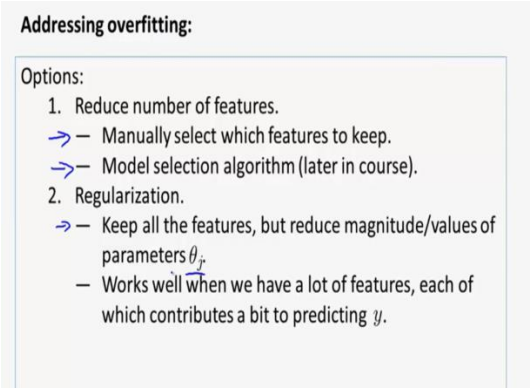
using cross-validation we can get an estimate of a model's generalization performance. If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then your model is overfitting. If it performs poorly on both, then it is underfitting. This is one way to tell when a model is too simple or too complex.

If we think overfitting is occurring what can we do to address it ?

Plotting the hypothesis could be one way to try to decide what degree polynomials to use , but that doesn't always work. In fact there are learning problems that may just have so many features , there is not just a matter of selecting what degree polynomial to use , in fact when we have so many features it also becomes much harder to plot the data , it becomes much harder to visualize it to decide what features to keep or not , and sometimes all of the features might be quite useful.

Addressing Overfitting

:-



The first approach is not always good because sometimes we need all of our features , each may contribute a bit to predicting the hypothesis.

MCP Neurons

In 1943 Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, published "A logical calculus of the ideas immanent in nervous activity" in the In this paper McCulloch and Pitts tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected together. These basic brain cells are called neurons, and McCulloch and Pitts gave a highly simplified model of a neuron in their paper. The McCulloch and Pitts model of a neuron, which we will call an MCP neuron for short, has made an important contribution to the development of artificial neural networks -- which model key features of biological neurons. The MCP neuron is not a real neuron; it's only a highly simplified model. We must be very careful in drawing conclusions about real neurons based on properties of MCP neurons.

In order to understand MCP neurons, let's look at an example. Suppose there is a neuron in a bird's brain that has two receivers, which are connected somehow to the bird's eyes. If the bird sees a round object, a signal is sent to the first receiver. But if any other shape is seen, no signal is sent. So the first receiver is a roundness detector. If the bird sees a purple object, a signal is sent to the second receiver of the neuron. But if the object is any other color, then no signal is sent. So the second receiver is a purple detector. Notice that for either receiver there is a question that can be answered "yes" or "no," and a signal is only sent if the answer is "yes." The first receiver corresponds to the question "Is the object round?" The second receiver corresponds to the question "Is the object purple?" We would like to produce an MCP neuron that will tell the bird to eat a blueberry, but to avoid eating red berries or purple violets. In other words, we want the MCP neuron to send an "eat" signal if the object is both round and purple, but the MCP neuron will send no signal if the object is either not round or not purple, or neither round nor purple. So the bird will only eat an object if the MCP neuron sends a signal. If no signal is sent, then the bird will not eat the object. Here is a table that summarizes how the MCP neuron would work in several cases.

Table 1

Object	Purple?	Round?	Eat?
Blueberry	Yes	Yes	Yes
Golf ball	No	Yes	No
Violet	Yes	No	No
Hot Dog	No	No	No

Notice that all the signals sent to the MCP neuron and the signal that it sends out are all "yes" or "no" signals. This "all or nothing" feature is one of the assumptions that McCulloch and Pitts made about the workings of a real neuron. They also assumed that somehow a real neuron "adds" the signals from all its receivers, and it decides whether to send out a "yes" or "no" signal based on the total of the signals it receives. If the total of the received signals is

high enough, the neuron sends out a "yes" signal; otherwise, the neuron sends a "no" signal. In order to "add" the signals that the MCP neuron is receiving, we will use the number 1 for a "yes" and the number 0 for a "no." Then Table 1 will now look like this.

Table 2

Object	Purple?	Round?	Eat?
Blueberry	1	1	1
Golf ball	0	1	0
Violet	1	0	0
Hot Dog	0	0	0

Now we need a way to decide if the total of the received signals is "high enough." The way McCulloch and Pitts did this is to use a number they called a **threshold**. So what is a threshold, and how does it work? Every MCP neuron has its own threshold that it compares with the total of the signals it has received. If the total is bigger than or equal to the threshold, then the MCP neuron will send out a 1 (i.e. a "yes" signal). If the total is less than the threshold, then the MCP neuron will send out a 0 (i.e. a "no" signal). So the MCP neuron is answering the question "Is the sum of the signals I received greater than or equal to my threshold?"

In order to see how this threshold idea works, let's suppose that we have an MCP neuron with two receivers connected to a bird's eyes. The one receiver is a roundness detector and the other is a purple detector, just as we had in the example above. Since we want the neuron to instruct the bird to eat blueberries but not golf ball, violets or hotdogs we need a threshold high enough so that it requires that *both* of the two properties are present. Let's try a threshold of "2" and see if that doesn't work.

If the bird sees a blueberry, then the purple detector sends a 1 and the roundness detector sends a 1. So our MCP neuron adds these signals to get a **combined input** of 1+1 =2. Now our MCP neuron takes this total input of 2 and compares it to its threshold of 2. Because the total input (= 2) is greater than or equal to the threshold (=2), the MCP neuron will send an output of 1 (which means, "EAT").

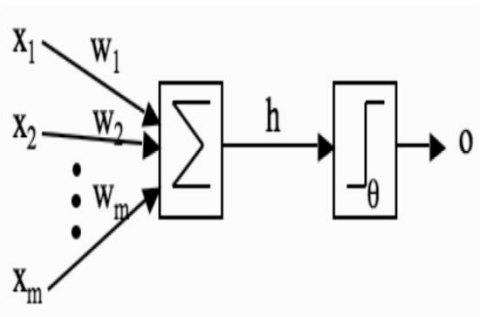
We should now be familiar with the basic workings of an MCP neuron. Your next task is to plug in different values for the **threshold** to produce different types of behavior in the bird. Where would you set the threshold so the bird would eat 3 of the 4 objects? none of the objects? all four of the objects? In the interactive exercises below, you will be asked to set the threshold so as to cause the bird to behave in different ways. There are three separate tasks that you can perform. We encourage you to do all of them.

Let's summarize what we know so far. The behavior of our (artificial) bird is dependent upon (1) the information it receives from the

environment (through it's two sensory "detectors"), and (2) the threshold at which the MCP neuron is set. The detectors can be thought of as "looking for" particular properties. The combined input to the neuron (from the detectors) will be larger or smaller depending upon how many of the "looked-for" properties are detected by the sensors. Whether or not the total input from the sensors will be a number high enough to cause the neuron to "fire" (i.e, to give an output of "1" instead of "0") depends upon how high the threshold is.

Object	Purple?	Round?	Total	Greater than or equal to threshold of 1?	Eat?
Blueberry	1	1	2	Yes	1
Golf ball	0	1	1	Yes	1
Violet	1	0	1	Yes	1
Hot Dog	0	0	0	No	0

FIGURE :-A picture of McCulloch and Pitts' mathematical model of a neuron. The inputs xi are multiplied by the weights wi , and the neurons sum their values. If this sum is greater than the threshold θ then the neuron fires; otherwise it does not.



(1) **a set of weighted inputs** wi that correspond to the synapses

(2) **an adder** that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)

(3) **an activation function** (initially a threshold function) that decides whether the neuron fires ('spikes') for the current inputs.

A picture of their model is given in Figure 3.1, and we'll use the picture to write down a mathematical description. On the left of the picture are a set of input nodes (labeled x1, x2, . . . xm). These are given some values, and as an example we'll assume that there are three inputs, with x1 = 1, x2 = 0, x3 = 0.5. In real neurons those inputs come from the outputs of other neurons. So the 0 means that a neuron didn't fire, the 1 means it did, and the 0.5 has no biological meaning, but never mind. (Actually, this isn't quite fair, but it's a long story and not very relevant.) Each of these other neuronal firings flowed along a synapse to arrive at our neuron, and those synapses have strengths, called weights. The strength of the synapse affects the strength of the signal, so we multiply the input by the weight of the synapse (so we get x1 × w1 and x2 × w2, etc.). Now when all of these signals arrive into our neuron, it adds them up to see if there is enough strength to make it fire. We'll write that as

$$h = \sum_{i=1}^{m} w_i x_i,$$

$$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta. \end{cases}$$

which just means sum (add up) all the inputs multiplied by their synaptic weights. I've assumed that there are m of them, where m = 3 in the example. If the synaptic weights are w1 = 1, w2 = −0.5, w3 = −1, then the inputs to our model neuron are h = 1 × 1 + 0 × −0.5 + 0.5 × −1 = 1 + 0 + −0.5 = 0.5. Now the neuron needs to decide if it is going to fire. For a real neuron, this is a question of whether the membrane potential is above some threshold. We'll pick a threshold value (labeled θ), say θ = 0 as an example. Now, does our neuron fire? Well, h = 0.5 in the example, and 0.5 > 0, so the neuron does fire, and produces output 1. If the neuron did not fire, it would produce output 0.

The McCulloch and Pitts neuron is a binary threshold device. It sums up the inputs (multiplied by the synaptic strengths or weights) and either fires (produces output 1) or does not fire (produces output 0) depending on whether the input is above some threshold. We can write the second half of the work of the neuron(The First One is to add the inputs with the weighted synapses), the decision about whether or not to fire (which is known as an activation function), as:

This is a very simple model, but we are going to use these neurons, or very simple variations on them using slightly different activation functions (that is, we'll replace the threshold function with something else) for most of our study of neural networks. In fact, these neurons might look simple, but as we shall see, a network of such neurons can perform any computation that a normal computer can, provided that the weights wi are chosen correctly. So one of the main things we are going to talk about for the next few chapters is methods of setting these weights.

Excitatory

So far we have only considered signals coming from the bird's receivers that are added to the other signals coming from the other receivers. These types of signals are called excitatory because they excite the neuron toward possibly sending its own signal. The more excitatory signals a neuron receives, the closer the total will be to the neuron's threshold, and so the closer the neuron will be to sending its signal. So as the neuron receives more and more excitatory signals, it gets more and more excited, until the threshold is reached, and the neuron sends out its own signal. But there is another

kind of signal that has the opposite effect on a neuron. These other signals are called inhibitory signals, and they have the effect of inhibiting the neuron from sending a signal. When a neuron receives an inhibitory signal, it becomes less excited, and so it takes more excitatory signals to reach the neuron's threshold. In effect, inhibitory signals subtract from the total of the excitatory signals, making the neuron more relaxed, and moving the neuron away from its threshold.

Inhibitory Signal

Now let's look at an example of an MCP neuron with an inhibitory signal. Let's consider a particular type of bird, say a robin. Now the robin, which has red feathers on its breast, is safe around any red objects, including red creatures such as a cardinal. Suppose our robin's brain has a neuron with two receivers connected to the robin's eyes. Normally our robin will flee from any other creature it sees. If the robin sees another creature, an excitatory signal will be sent to the first receiver, which will try to cause the bird to flee. So the first receiver is a creature detector, and it excites our bird to fleeing. However, if the creature that the robin sees has red on it, an inhibitory signal will be sent to the second receiver, which will prevent the bird from fleeing. So the second receiver is a red detector, and it inhibits our bird from fleeing.

Suppose our robin sees a black cat. What would happen? The creature detector would send an excitatory signal to the neuron, and the red detector would send no signal. So the bird would flee.

Suppose our robin sees a cardinal. The creature detector would send an excitatory signal to the neuron, and the red detector would send an inhibitory signal. So the bird would not flee, because the inhibitory signal would "cancel" the excitatory signal.

Here is a table that summarizes how this MCP neuron with the excitatory and inhibitory signals would work in several cases.

Object	Creature?	Red?	Flee?
Black Cat	Yes	No	Yes
Male Cardinal	Yes	Yes	No
Hot Dog	No	Yes	No

Now we'll see how these new ideas of excitatory and inhibitory signals work when the MCP neuron compares these signals to its threshold. As before, we'll use a 1 if an **excitatory** signal is sent, and a 0 if no excitatory signal is sent. But now we'll use a -1 when an **inhibitory** signal is sent, and a 0 if no inhibitory signal is sent. Because we are using a -1 for an inhibitory signal, when we add an inhibitory signal to the total of all signals received, the effect of the

inhibitory signal on the total is to subtract a 1. (Recall that adding a -1 is the same as subtracting a 1.) So when an MCP neuron computes the total effect of its signals, it will add a 1 to the total for each of the excitatory signals and add a -1 to the total for each of its inhibitory signals. If the total of excitatory signals and inhibitory signals is greater than or equal to the threshold, then the MCP neuron will send a 1. If the total of excitatory signals and inhibitory signals is less than the threshold, then the MCP neuron will send a 0.

Let's look at an example. Suppose we have an MCP neuron connected to a creature detector that sends an excitatory signal and a red detector that sends an inhibitory signal. Let's also suppose the threshold is 1. We could have chosen another number for the threshold. Now for each object in Table 4, we can compute the total of the signals by adding a 1 for each excitatory signal and a -1 for each inhibitory signal. Then we compare the total to the threshold.

If the robin sees a black cat, then the creature detector, which is excitatory, sends a 1, because the cat is a creature. The red detector, which is inhibitory, sends a 0 , because the cat is not red. Because there is one excitatory signal and no inhibitory signal, the total is 1 + 0 = 1 We compare this total of 1 to the threshold. Because the total of 1 is equal to the threshold of 1, the MCP neuron will send a 1, and so the robin will flee.

If the robin sees a male cardinal, then the creature detector, which is excitatory, sends a 1, because the cardinal is a creature. The red detector, which is inhibitory, sends a -1, because the cardinal is red. Because there is one excitatory signal and one inhibitory signal, the total is 1 + -1 = 0. We compare this total of 0 to the threshold. Because 0 is less than the threshold of 1, the MCP neuron will send a 0, and so the robin will not flee.

If the robin sees a hot dog, then the creature detector, which is excitatory, sends a 0, because the hot dog is not a creature. The red detector, which is inhibitory, sends a -1, because the hot dog is red. Because there is no excitatory signal and one inhibitory signal, the total is 0 + -1 = -1. We compare this total of -1 to the threshold. Because -1 is less than the threshold of 1 , the MCP neuron will send a 0, and so the robin will not flee.

We can summarize how this MCP neuron works in the table below.

Table 5

Object	Creature? Excitatory	Red? Inhibitory	Total	Greater than or equal to threshold of 1?	Flee?
Black Cat	1	0	1	Yes	1
Male Cardinal	1	-1	0	No	0
Hot Dog	0	-1	-1	No	0

We now have two ways of adjusting

an MCP neuron. We can make signals either excitatory or inhibitory, and we can change the threshold. By making different choices for these adjustments, we can make MCP neurons that produce a variety of results. For example, suppose we wanted to construct an MCP neuron that signals a bird to eat objects that are not both purple and round. In other words, this is an MCP neuron that does the exact opposite of what we did in our first example of an MCP neuron. (See Table 2.) Here's what we could do. We could make both the purple detector and the roundness detector inhibitory. Then we could set the threshold to 0.

Now if the object is a blueberry, there are two inhibitory signals sent. So the total is -1 + (-1)= -2 . When we compare -2 to the threshold of 0, we see that -2 is less than 0, so the MCP neuron will send a 0, and the bird will not eat the blueberry.

The rest of the results are summarized in the following table.

Table 6

Object	Purple? Inhibitory	Round? Inhibitory	Total	Greater than or equal to threshold of 0?	Eat?
Blueberry	-1	-1	- 2	No	0
Golf ball	0	-1	-1	No	0
Violet	-1	0	- 1	No	0
Hot Dog	0	0	0	Yes	1

As you can see, we can indeed make an MCP neuron that does the exact opposite of the first MCP neuron that we considered. We would now like to see what other results we can produce with an MCP neuron by using various combinations of excitatory and inhibitory signals with various thresholds. For now we will only consider MCP neurons that receive two signals. Is there any limit to the type of MCP neurons we can make?

There are some limits. With only two signals, there are only three possible combinations of excitatory and inhibitory signals: both signals excitatory, both signals inhibitory, and one excitatory signal with one inhibitory signal. We've seen all of these combinations in previous examples.

Of course, the number of possible thresholds is infinite. But there are only a few types of results that an MCP neuron can produce, because it only sends out a 0 or 1. MCP neurons are also limited by the number of signals they receive. If we ignore the types of detectors and just concentrate on whether or not the detector sends a signal, we can see there are only four combinations of two signals: both detectors send a signal, the first sends a signal and the second detector does not , the second detector sends a signal and the first detector does not, and neither detector sends a signal. As a consequence, it can be shown that there are sixteen possible ways(2^4) to produce a 0 or 1 from each of the four pairs of signals. So there are at most sixteen possible types of MCP

neurons with two receivers. We'll now consider a particular one of these sixteen types.

Let's consider a bird with two detectors connected to a neuron. The first detector will send a signal if the object is a creature with four legs, and the second detector sends a signal if the object is green. We want to make an MCP neuron that will signal the bird to flee if the object is either four-legged or green, but not both. We'll start by trying two excitatory signals with a threshold of 1. We'll consider four objects: a frog, a green snake, a black cat, and a violet. Notice that these objects give us the four possible combinations of 0 and 1 for each pair of signals. Also, because all the signals are excitatory, the total is just the sum of the two signals. This will produce the following results.

Table 7

Object	Four Legs? Excitatory	Green? Excitatory	Total	Greater than or equal to threshold of 1?	Flee?
Frog	1	1	2	Yes	1
Green Snake	0	1	1	Yes	1
Black Cat	1	0	1	Yes	1
Violet	0	0	0	No	0

While this is close to what we want, it's not perfect. The bird will flee from an object that is four-legged, such as the cat, or from an object that is green, such as the snake. But the bird will also flee from any object that is both four-legged and green, which we don't want. So this combination of signals and threshold doesn't work. In fact, what we are trying to do is make an MCP neuron that produces a 1 if either signal is a 1, but produces a 0 if both signals are either 0 or 1.

Exercise 9. Make a table like Table 7 for an MCP neuron with one excitatory signal and one inibitory signal and a threshold of 0. Does this MCP neuron produce a 1 if either signal is a 1, but produces a 0 if both signals are either 0 or 1?

The particular type of MCP neuron we are trying to make is called an "exclusive or" MCP neuron, because it sends a 1 if it receives a 1 from one signal or the other, but not both. An "exclusive or" would have 0,1,1,0 in the last column of Table 7. There is also an "inclusive or" which sends a 1 if it receives a 1 from either signal or both. An "inclusive or" would have 1,1,1,0 in the last column of Table 7. In fact, we have already made an "inclusive or" MCP neuron in **Table 3**.

If we try the various combinations of excitatory and inhibitory signals with several thresholds, we will begin to

suspect that it is impossible to build an MCP neuron that produces an "exclusive or." In fact, it can be proved mathematically that no combination of signals and thresholds can produce the "exclusive or." So there is a fundamental limitation on the type of results a single MCP neuron can produce.

In order to produce more complicated results like the "exclusive or" we must start connecting MCP neurons together to form neural networks. It turns out that the "exclusive or" can be made by using three MCP neurons each with two receivers. Two of the MCP neurons each have one receiver attached to the first detector and the other receiver attached to the second detector. So the same signal from each detector is sent to both MCP neurons simultaneously. Each of these MCP neurons then send their signal to one of the receivers of the third MCP neuron. If the various signals and thresholds are chosen properly, this network will accept two signals and send out a signal that produces an "exclusive or." In other words, we can produce a neural network of three MCP neurons that produces an "exclusive or."

Another way to increase the possible results from an MCP neuron is to use more receivers. We could look at MCP neurons with three or more receivers. We could then link these MCP neurons together in a neural network. By doing just this, McCulloch and Pitts showed there is a neural network of MCP neurons that produces whatever combination of 1's and 0's we would like from the possible signals it receives. So while there is a fundamental limitation on a single MCP neuron, this limitation can be overcome by connecting the single MCP neurons together in a neural network. This also shows how very complicated results can be obtained by connecting together a large number of very simple parts (the MCP neurons). Perhaps this explains how our brain can do amazing things, even though it is a collection of a huge number of basic cells (real neurons) that are connected to a very large number of other basic cells.

The Artificial Neural Networks One thing that is probably fairly obvious is that one neuron isn't that interesting. It doesn't do very much, except fire or not fire when we give it inputs. In fact, it doesn't even learn. If we feed in the same set of inputs over and over again, the output of the neuron never varies—it either fires or does not. So to make the neuron a little more interesting we need to work out how to make it learn, and then we need to put sets of neurons together into neural networks so that they can do something useful. The question we need to think about first is how our neurons can learn. We are going to look at supervised learning for the next few chapters, which means that the algorithms will learn by example: the dataset that we learn

from has the correct output values associated with each data point. At first sight this might seem pointless, since if you already know the correct answer, why bother learning at all? The key is in the concept of generalization that we saw in Section 1.2. Assuming that there is some pattern in the data, then by showing the neural network a few examples we hope that it will find the pattern and predict the other examples correctly. This is sometimes known as pattern recognition.

Before we worry too much about this, let's think about what learning is. In the Introduction it was suggested that you learn if you get better at doing something. So if you can't program in the first semester and you can in the second, you have learnt to program. Something has changed (adapted), presumably in your brain, so that you can do a task that you were not able to do previously. Have a look again at the McCulloch and Pitts neuron (e.g., in Figure 3.1) and try to work out what can change in that model. The only things that make up the neuron are the inputs, the weights, and the threshold (and there is only one threshold for each neuron, but lots of inputs). The inputs can't change, since they are external, so we can only change the weights and the threshold, which is interesting since it tells us that most of the learning is in the weights, which aren't part of the neuron at all; they are the model of the synapse! Getting excited about neurons turns out to be missing something important, which is that the learning happens between the neurons, in the way that they are connected together. So in order to make a neuron learn, the question that we need to ask is: How should we change the weights and thresholds of the neurons so that the network gets the right answer more often?

Neural Networks

Before we discuss about artificial neurons, let's take a quick look at a

biological neuron. Neurons are unusual looking cells mostly found in animals brain which has a number of input wires called Dendrite(accept input from other locations) and also has an output wire called Axons(to send signals or messages to other neurons) with a cell body containing the nucleus. Neurons are generally a computational units that get a number of input wires with their dendrites , do some computations with their Nucleus and send outputs with various Axons to other nucleus. Near its extremity the axon splits off into many branches called telodendria, and at the tip of these branches are minuscule structures called synaptic terminals (or simply synapses), which are connected to the dendrites or cell bodies of other neurons. One output of the Axons will connect to the input wire Dendrites of other Neurons. The Input signals can be gained from the external inputs from the environment or the outputs of other neurons. There are about 100billion(10^11) Neurons on our brain. Each neuron is typically connected to thousands of other neurons, so that it is estimated that there are about 100 trillion (= 10^14) synapses within the brain. Biological neurons produce short electrical impulses called action potentials (APs, or just signals) in order to communicate with each other and travel along the axons and make the synapses release chemical signals called neurotransmitters. When a neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires its own electrical impulses (actually, it depends on the neurotransmitters, as some of them inhibit the neuron from firing). Each neuron can be viewed as a separate processor, performing a very simple computation: This makes the brain a massively parallel computer made up of 10^11 processing elements.(The parallelism refers to the fact that each node is conceived of as operating independently and concurrently (in parallel with) the others, and the "knowledge" in the network is distributed over the entire set of weights, rather than focused in a few memory locations as in a conventional computer.) If that is all there is to the brain, then we should be able to model it inside a computer and end up with animal or human intelligence inside a computer. This is the view of strong AI.

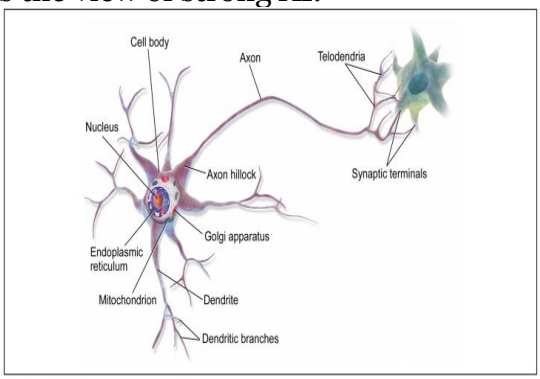
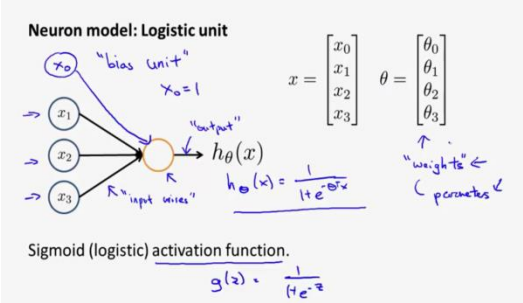


Figure 10-1. Biological neuron[4]

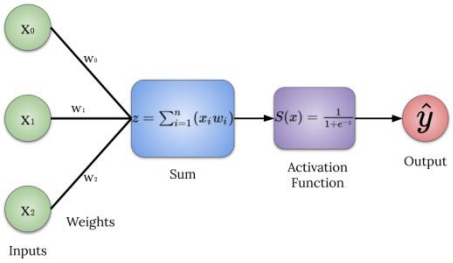
Thus, individual biological neurons seem to behave in a rather simple way, but they are organized in a vast network of billions, with each neuron typically connected to thousands of other neurons. Highly complex computations can be performed by a network of fairly simple neurons. In the context of Machine Learning, the phrase “neural networks” generally refers to ANNs, not BNNs.

One thing that is probably fairly obvious is that one neuron isn’t that interesting. It doesn’t do very much, except fire or not fire when we give it inputs. In fact, it doesn’t even learn. If we feed in the same set of inputs over and over again, the output of the neuron never varies—it either fires or does not. So to make the neuron a little more interesting we need to work out how to make it learn, and

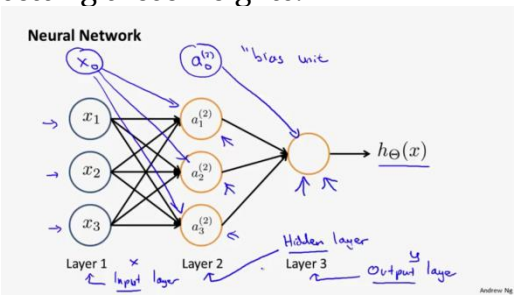
then we need to put sets of neurons together into neural networks so that they can do something useful.



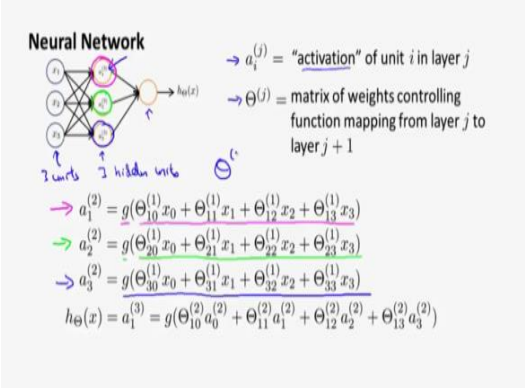
(1) **a set of weighted inputs** wi that correspond to the synapses (according to BNN).
(2) **an adder** that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)
(3) **a Sigmoid(Logistic) activation function** (initially a threshold function) that decides whether the neuron fires ('spikes') for the current inputs.



A picture of their model is given in Figure 3.1, and we'll use the picture to write down a mathematical description. On the left of the picture are a set of input nodes (labeled x1, x2, . . . xm). These are given some values, and as an example we'll assume that there are three inputs, with x1 = 1, x2 = 0, x3 = 0.5. In real neurons those inputs come from the outputs of other neurons. So the 0 means that a neuron didn't fire, the 1 means it did, and the 0.5 has no biological meaning, but never mind. (Actually, this isn't quite fair, but it's a long story and not very relevant.) Each of these other neuronal firings flowed along a synapse to arrive at our neuron, and those synapses have strengths, called weights. The strength of the synapse affects the strength of the signal, so we multiply the input by the weight of the synapse (so we get x1 × w1 and x2 × w2, etc.). Now when all of these signals arrive into our neuron, it adds them up to see if there is enough strength to make it fire.
This is a very simple model, but we are going to use these neurons, or very simple variations on them using slightly different activation functions (that is, we'll replace the threshold function with something else) for most of our study of neural networks. In fact, these neurons might look simple, but as we shall see, a network of such neurons can perform any computation that a normal computer can, provided that the weights wi are chosen correctly. So one of the main things we are going to talk about for the next few chapters is methods of setting these weights.



Question:- Why are we calling the middle layer as a Hidden Layer ?
In supervised Learning , you get to see the input , you get to see the correct output where as the Hidden Layer are values you don't get observed in the training set , they are not in X , they are not in y , so we called them Hidden.

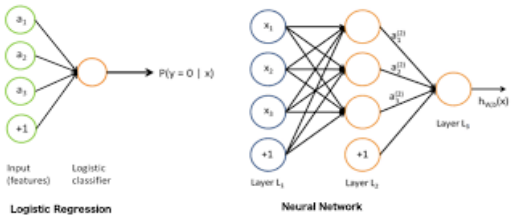


The question we need to think about first is how our neurons can learn. We are going to look at supervised learning for the next few chapters, which means that the algorithms will learn by example: the dataset that we learn from has the correct output values associated with each data point. At first sight this might seem pointless, since if you already know the correct answer, why bother learning at all? The key is in the concept of generalization that we saw in Section 1.2. Assuming that there is some pattern in the data, then by showing the neural network a few examples we hope that it will find the pattern and predict the other examples correctly. This is sometimes known as pattern recognition.

Before we worry too much about this, let's think about what learning is. In the Introduction it was suggested that you learn if you get better at doing something. So if you can't program in the first semester and you can in the second, you have learnt to program. Something has changed (adapted), presumably in your brain, so that you can do a task that you were not able to do previously. Have a look again at the McCulloch and Pitts neuron (e.g., in Figure 3.1) and try to work out what can change in that model. The only things that make up the neuron are the inputs, the weights, and the threshold (and there is only one threshold for each neuron, but lots of inputs). The inputs can't change, since they are external, so we can only change the weights and the threshold, which is interesting since it tells us that most of the learning is in the weights, which aren't part of the neuron at all; they are the model of the synapse! Getting excited about neurons turns out to be missing something important, which is that the learning happens between the neurons, in the way that they are connected together. So in order to make a neuron learn, the question that we need to ask is: How should we change the weights and thresholds of the neurons so that the network gets the right answer more often?

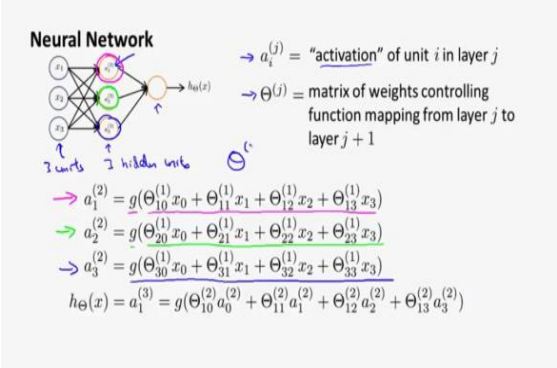
What is the relation between Logistic Regression and Neural Networks

Neural networks are somewhat related to logistic regression. Basically, we can think of logistic regression as a one layer neural network. a neural network can be viewed as a series of logistic regression classifiers stacked on top of each other.



Vectorized Representation Of a Neural Netwrok's

Here we'll look at how to carry out the computation efficiently through a vectorized implementation.



$Z_1^{(2)} = \Theta_{10}^{(1)}X_0 + \Theta_{11}^{(1)}X_1 + \Theta_{12}^{(1)}X_2 + \Theta_{13}^{(1)}X_3$
$Z_2^{(2)} = \Theta_{20}^{(1)}X_0 + \Theta_{21}^{(1)}X_1 + \Theta_{22}^{(1)}X_2 + \Theta_{23}^{(1)}X_3$
$Z_3^{(2)} = \Theta_{30}^{(1)}X_0 + \Theta_{31}^{(1)}X_1 + \Theta_{32}^{(1)}X_2 + \Theta_{33}^{(1)}X_3$

$Z_1^{(3)} = \Theta_{10}^{(2)}a_1^{(2)} + \Theta_{10}^{(2)}a_1^{(2)} + \Theta_{10}^{(2)}a_1^{(2)} + \Theta_{10}^{(2)}a_1^{(2)}$

$a_1^{(2)} = g(Z_1^{(2)})$
$a_2^{(2)} = g(Z_1^{(2)})$
$a_3^{(2)} = g(Z_1^{(2)})$

$a_1^{(3)} = g(Z_1^{(3)})$

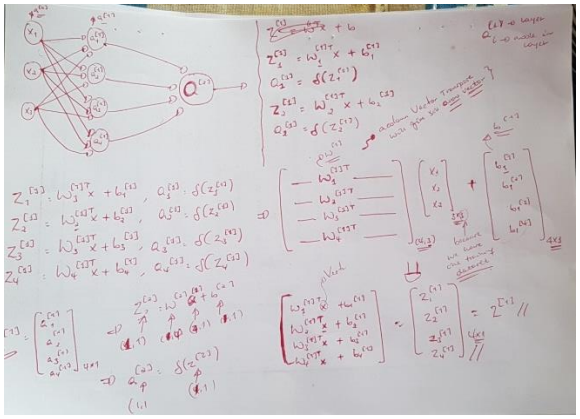
Let's put it in the vectorized form

$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $Z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$ $a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$

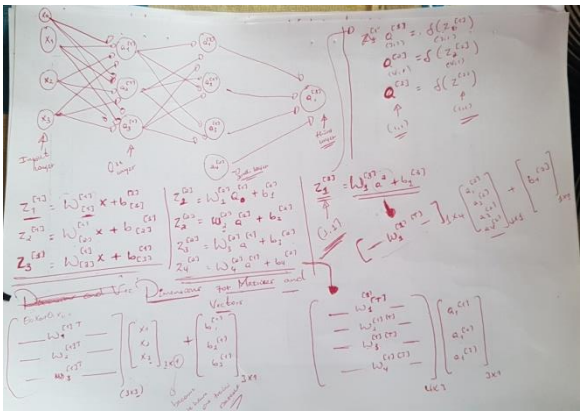
$Z^{(2)} = \Theta^{(1)} X$
$a^{(2)} = g(Z^{(2)})$

Then the hypothesis
$h_\Theta(x) = a^{(3)} = g(Z^{(3)})$ but $Z^{(3)} = \Theta^{(2)} a^{(2)}$

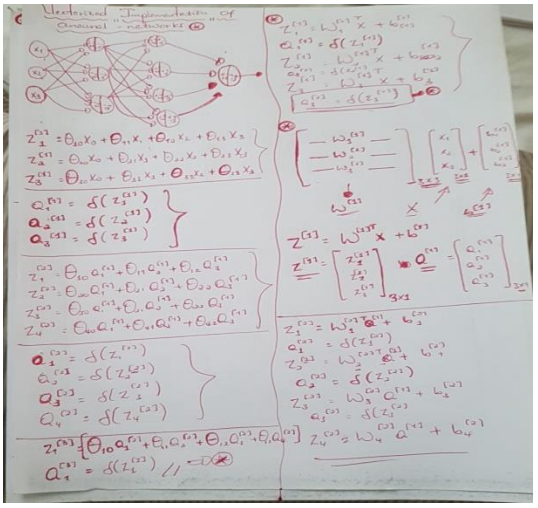
Dimensions Of The Matrices and Vectors vectorized with one training instance

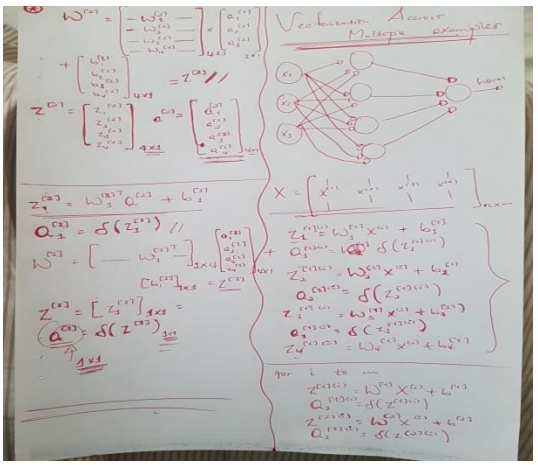


Dimensions of the matrices and the vectors for 3 – Layers neural network vectorized with one training instance

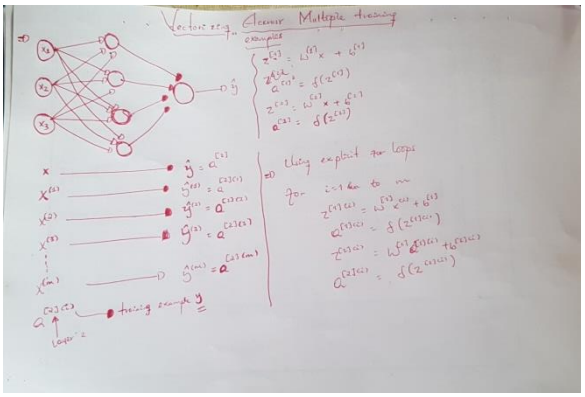


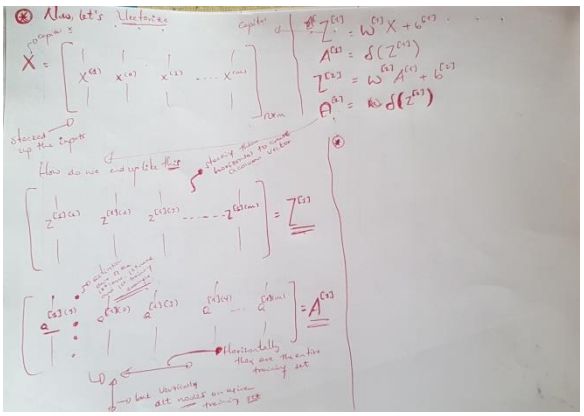
Detail:- Dimensions of the matrices and the vectors for 3 – Layers neural network vectorized with one training instance



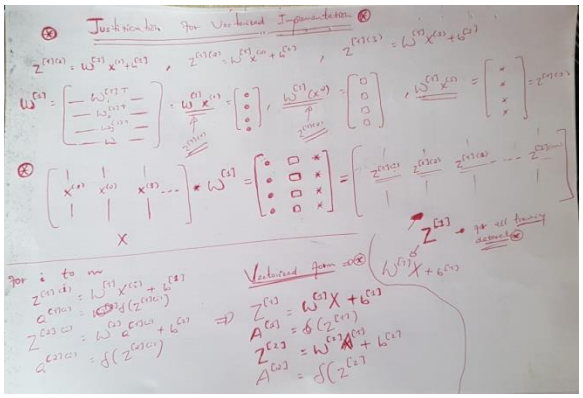


Vectorization Across Multiple Training Examples



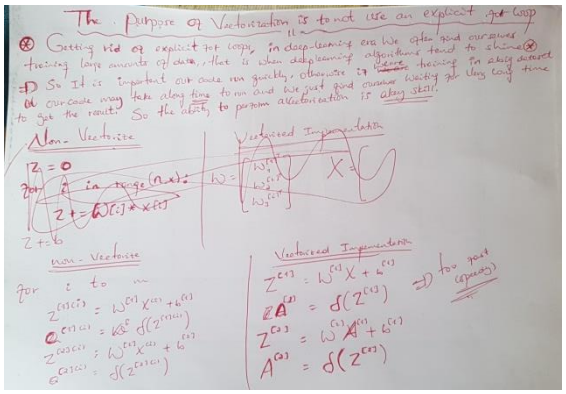


Justification For Vectorized Implementation



Why am I using vectorized representation of a neural network , because it makes the computation to much speedy.

Getting rid of explicit for loops , In deep learning era we often find ourselves training a large amount of data, that is when deep learning algorithms tend to shine. So it is important our code run quickly , otherwise if we are training in a big dataset our code may take along time to run and we just find ourselves waiting for a very long time to get the result , so the ability to perform a vectorization is a key skill.



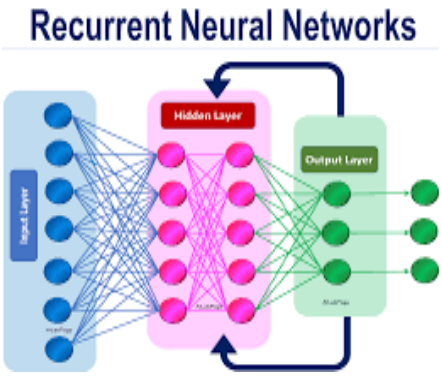
This process of calculating $h_\Theta(x)$ is called *forward propagation*

- Worked out from the first layer
- Starts off with activations of input unit
- Propagate forward and calculate the activation of each layer sequentially

Neural Network Architecture :- The way the neural networks are connected are called *Architectures*.

There are two main categories of network architectures depending on the type of the connections between the neurons . 1) Feed Forward neural Networks and 2) Recurrent Neural Networks . If there is no feed back

from the outputs of the neurons towards the inputs throughout the network , then the network is referred as a “Feed Forward Neural Network ” , Otherwise , if there exists a feed back (cylce) i.e a synaptic connection from the outputs towards the inputs , then the networks is called *Recurrent Neural Networks*.

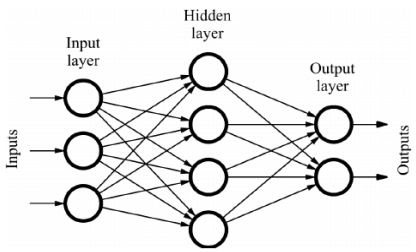


Feed Forward Neural Networks fall in to two categories depending on the number of the layers , either “Single Layer ” or “Multi Layer”

Feed Forward Neural Networks

A Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The opposite of a feed forward neural network is a recurrent neural network, in which certain pathways are cycled. The feed forward model is the simplest form of neural network as information is only processed in one direction.

While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.



Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. There is no feedback between layers. This is why they are called feed forward neural networks.

Feed-forward networks have the following characteristics:

1. Perceptron’s are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.
2. Each perceptron in one layer is connected to every perceptron on the next layer fairly in a straight forward. Hence information is constantly "fed forward" from one layer to the next., and this explains why these networks are called feed-forward networks.
3. There is no connection among perceptron’s in the same layer.;..

Finally, there are terms used to describe the shape and capability of a neural network; for example:

Size :- The total number of nodes in the network

Width :- The total number of nodes in a specific layer

Depth :- The total number of Layers that we will use in the Network including the output layer.

Architecture :- The Specific arrangement of the layers and the nodes in the networks.

Questions :- Why can't we connect neurons of the same layer ?

Is it necessary for artificial Neurons to always be connected fully ?

When do I use partially connected neural networks and when do I use fully connected neural networks ?

We have so many raw inputs then the output , so what is the advantage of the network of neurons ?

What does the Layers represent in a neural network ?

How to choose the number of hidden layers in my network ?

How many collection of neurons are grouped as a layer ?

What is the value of having so many hidden layers in the network?

Conceptually, what does it mean to have multiple neurons in a single layer of a neural network?

Where should I start counting the layers , from the input layers or from the first hidden layer ?

Are neurons in the same layer of an Artificial neural network connected to each other ?

In a feedforward artificial neural network, you assume that your inputs are all ***independant*** to each other, this means that no information shared among the inputs would in any way benefit the performance on the output (classification). This is the reason why the input layer has no intra-layer connections.

The hidden layers perform a feature extraction on these inputs, and we still assume that these extracted features are not dependent to each other.

We have so many raw inputs then the output , so what is the advantage of the network of neurons ?

The deep neural networks take raw inputs (such as pixel values in an image) and transform them into useful representations, extracting higher-level features (such as shapes and edges in images) that capture complex concepts by combining smaller and smaller pieces of information to solve challenging tasks such as image classification.

What does the Layers represent in a neural network ?

Artificial Neural Networks inspired from the Biological Neural networks
But we used the term Layer for grouping nodes together for simplicity for computation

How to choose the number of hidden layers in my network ?

Depends on the problem but usually from 1 - 5

How many collection of neurons are grouped as a layer ?

It depends on the problem but usually from 10 - 100

Where should I start counting the layers , from the input layers or from the first hidden layer ?

Traditionally, there is some disagreement about how to count the number of layers.

Some researchers argue that we need to start counting our layers starting from the input Layers. Because the whole idea of the Layer is to group nodes together for computation simplicity , so we are now grouping the input features together in a Layer , so we need to count it as out First Layer.

Other researchers argue that , it Should not be counted because the input features are not active , they are simply input variables , the didn't extract features. So by these reason it should not counted , hence for this reason we called it the input Layer as a zero layer.

This convenient notation summarizes both the number of layers and the number of nodes in each layer. The number of nodes in each layer is specified as an integer, in order from the input layer to the output layer, with the size of each layer separated by a forward-slash character ("/").

For example, a network with two variables in the input layer, one hidden layer with eight nodes, and an output layer with one node would be described using the notation: 2/8/1.

Conceptually, what does it mean to have multiple neurons in asingle layer of a neural network?

There is a difference, each neuron in a layer is a feature detector and having 1 vs 100 can be a big difference. The idea of having multiple neurons in a single layer is to make those neurons learn compact representations of the training data.

So the multiple nodes in a layer make it possible for that layer to learn a small set of features that can be useful for the network to make decisions.

What is the value of adding so many hidden layers in the network.

The **hidden layer(s)** are where the *black magic* happens in neural networks. Each layer is trying to *learn* different aspects about the data by minimizing an error/cost function. The most intuitive way to understand these layers is in the context of 'image recognition' such as a face. The first layer may learn edge detection, the second may detect eyes, third a nose, etc. This is not exactly what is happening but the idea is to break the problem up in to components that different levels of abstraction can piece together much like our own brains work (hence the name 'neural networks').

A single-layer neural network can only be used to represent linearly separable functions. This means very

simple problems where, say, the two classes in a classification problem can be neatly separated by a line. If your problem is relatively simple, perhaps a single layer network would be sufficient.

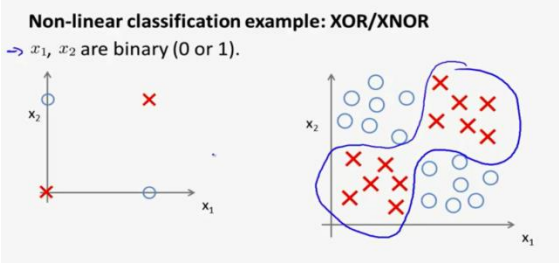
Most problems that we are interested in solving are not linearly separable.

A Multilayer Perceptron can be used to represent convex regions. This means that in effect, they can learn to draw shapes around examples in some high-dimensional space that can separate and classify them, overcoming the limitation of linear separability.

In fact, there is a theoretical finding by Lippmann in the 1987 paper "An introduction to computing with neural nets" that shows that an MLP with two hidden layers is sufficient for creating classification regions of any desired shape. This is instructive, although it should be noted that no indication of how many nodes to use in each layer or how to learn the weights is given.

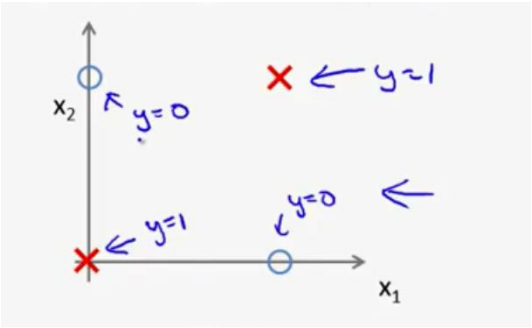
Implementing Logic Gates using Neural Networks

x1 , x2 element of (0 , 1)



X1 XOR X2 , it will be True(1) either x1 or x2 is True
X1 XNOR X2 or NOT(X1 XOR X2)

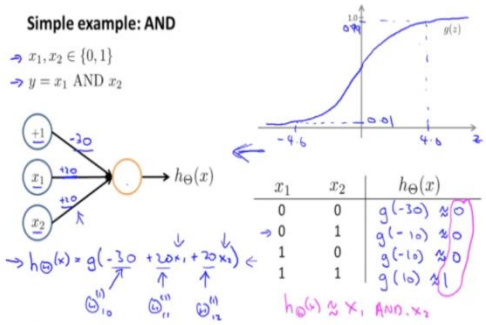
-Let's look the XNOR for the above non – linear classification example
So it will be True when both of them are 1 or when both of them are 0.



Let's check this

Implementing AND gate

AND gate operation is a simple multiplication operation between the inputs. If any of the input is 0, the output is 0. In order to achieve 1 as the output, both the inputs should be 1.



All this boils down to one question and that is given we have the input vector x, what should be the value of ***weight vector*** to achieve this task?

(0,0) case

Consider a situation in which the input or the x vector is (0,0). The value of Z, in that case, will be nothing but W0. Now, W0 will have to be less than 0 so that Z is less than 0.5 and the output or ŷ is 0 and the definition of the AND gate is satisfied. If it is above 0, then the value after Z has passed through the sigmoid function will be 1 which violates the AND gate condition. Hence, we can say with a resolution that W0 has to be a negative value. But what value of W0?

(0,1) case

Now, consider a situation in which the input or the x vector is (0,1). Here the value of Z will be W0+0+W2*1. This being the input to the sigmoid function should have a value less than 0 so that the output is less than 0.5 and is classified as 0. Henceforth, W0+W2<0. If we take the value of W0 as -30(remember the value of W0 has to be negative) and the value of W2 as +20, the result comes out to be -3+20 and that is -10 which seems to satisfy the above inequality and is at par with the condition of AND gate.

(1,0) case

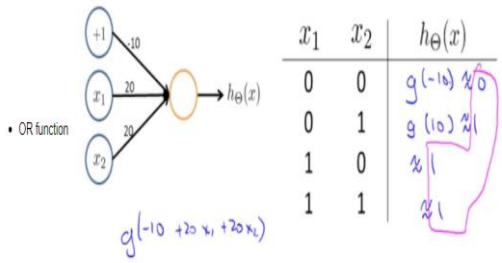
Similarly, for the (1,0) case, the value of W0 will be -30 and that of W1 can be +20. Remember you can take any values of the weights W0, W1, and W2 as long as the inequality is preserved.

(1,1) case

In this case, the input or the x vector is (1,1). The value of Z, in that case, will be nothing but W0+W1+W2. Now, the overall output has to be greater than 0 so that the output is 1 and the definition of the AND gate is satisfied. From previous scenarios, we had found the values of W0, W1, W2 to be -30,20,20 respectively. Placing these values in the Z equation yields an output-30+20+20 which is 10 and greater than 0. This will, therefore, be classified as 1 after passing through the sigmoid function.

Implementing OR gate

The table on the left depicts the truth table of the OR gate. For the given two inputs, if any of the input is 1, the output or ***y*** is 1. The graph on the right shows the plotting of all the four pairs of input i.e (0,0), (0,1), (1,0), and (1,1)



$h_\Theta(x)$ = X1 OR X2

(0,0) case

Consider a situation in which the input or the x vector is (0,0). The value of Z, in that case, will be nothing but W0. Now, W0 will have to be less than 0 so that the output or ŷ is 0 and the definition of the OR gate holds water. Hence, we can say with a resolution that W0 has to be a negative value.

(0,1) case

Now, consider a situation in which the input or the x vector is (0,1). Here

the value of Z will be W0+0+W2*1. This being the input to the sigmoid function should have a value greater than 0 so that the output is greater than 0.5 and is classified as 1. Henceforth, W0+W2>0. if we take the value of W0 as -10(remember the value of W0 has to be negative) and the value of W2 as +20, the result comes out to be -10+20 and that is 1 which seems to satisfy the above inequality.

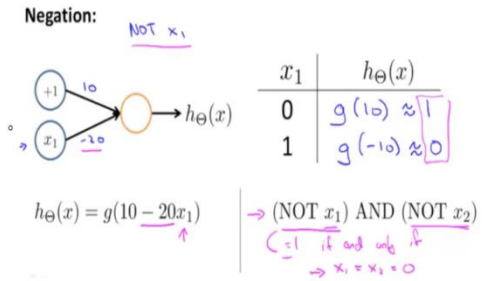
(1,0) case

Similarly, for the (1,0) case, the value of W0 will be -10 and that of W1 can be +20.

(1,1) case

Similarly, for the (1,1) case, the value of W0 will be -10 and that of W1 can be +20 and the value of the W2 can be +20.

Implementing NOT gate



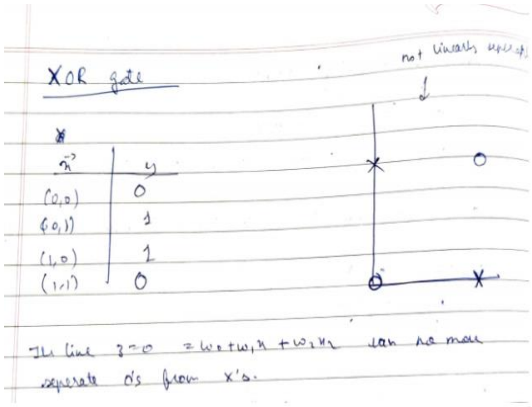
In order to end up with a negation we need to add a large negative weight to the variable you want to negate.

-Negating more than one variable

(NOT X1) AND (NOT X2) , outputs 1 if and only if both of the variables are 0.

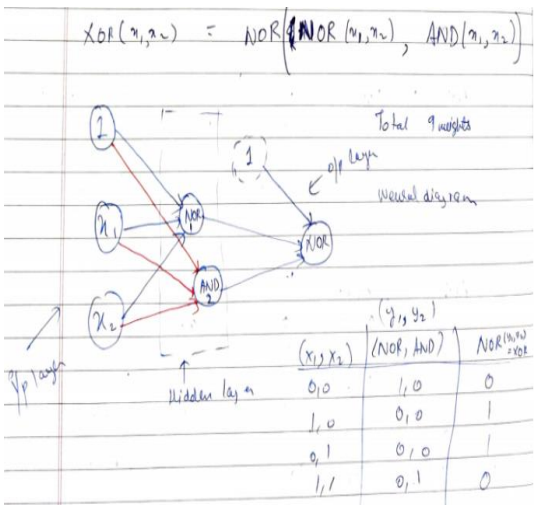
Implementing XOR gate

For the XOR gate, the truth table on the left side of the image below depicts that if there are two complement inputs, only then the output will be 1. If the input is the same(0,0 or 1,1), then the output will be 0. The points when plotted in the x-y plane on the right gives us the information that they are not linearly separable like in the case of OR and AND gates(at least in two dimensions).



To solve the above problem of separability, adding extra layers also known as Deep network is used.

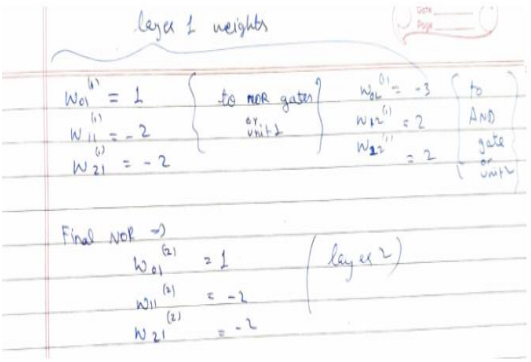
XOR (x1,x2) can be thought of as NOR(NOR(x1,x2),AND(x1,x2))



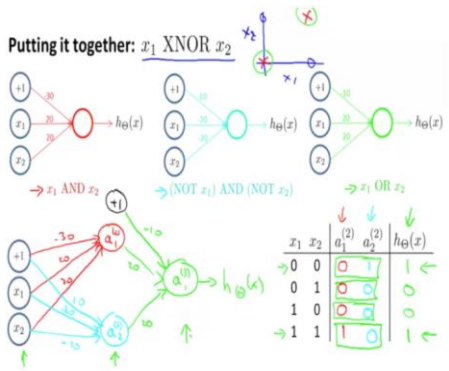
Weights of the XOR network

Here we can see that the layer has increased from 2 to 3 as we have added a layer where AND and NOR operation is being computed. The inputs remain the same with an additional bias input of 1. The table

on the right below displays the output of the 4 inputs taken as the input. An interesting thing to notice here is that the total number of weights has increased to 9. A total of 6 weights from the input layer to the 2nd layer and a total of 3 weights from the 2nd layer to the output layer. The 2nd layer is also termed as a hidden layer.



Implementing XNOR gate



Multi Class Classification : Neural Network Representation

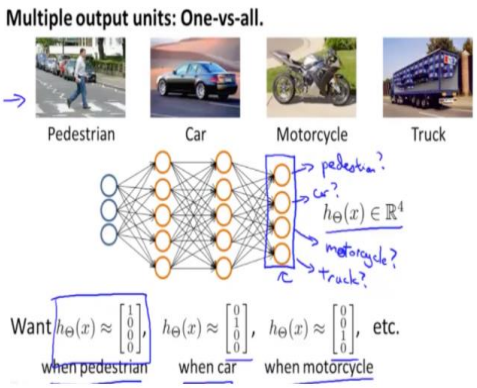
Multi-label classification involves predicting zero or more class labels. Deep learning neural networks are an example of an algorithm that natively supports multi-label classification problems.

Classification is a predictive modeling problem that involves outputting a class label given some input.

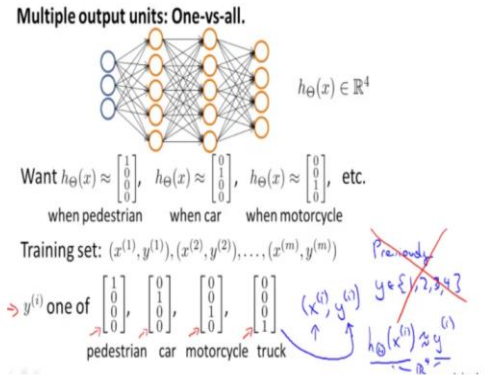
Typically, a classification task involves predicting a single label. Alternately, it might involve predicting the likelihood across two or more class labels.

Neural network models can be configured to support multi-label classification and can perform well, depending on the specifics of the classification task.

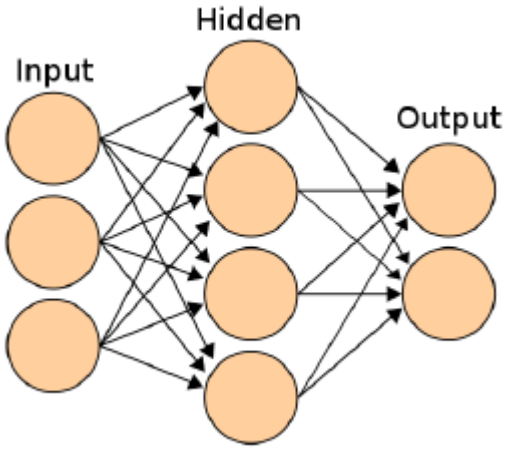
Multi-label classification can be supported directly by neural networks simply by specifying the number of target labels there is in the problem as the number of nodes in the output layer. For example, a task that has three output labels (classes) will require a neural network output layer with three nodes in the output layer.



Each node in the output layer must use the sigmoid activation. This will predict a probability of class membership for the label, a value between 0 and 1. Finally, the model must be fit with the binary cross-entropy loss function.



Multi Class Classification only works for the neural networks with three or more classes because with two classes we can't do the one versus all method , if we have two classes it is binary classifier so a neural network with one output will work for that . But the question is if we used one output for two class labels and multi classification for classes with three or more than three class labels , what is the point of having a neural networks with two output nodes?



Difference Between Multi-Class and Multi-Label Classification Problem Statements in Deep Learning

While solving the classification problem statements using Deep Learning, we may come up with mainly the following two types of classification tasks:

- Multi-Class Classification
- Multi-Label Classification

As a short introduction, In multi-class classification, each input will have only **one output class**, but in multi-label classification, each input can have **multi-output classes**.

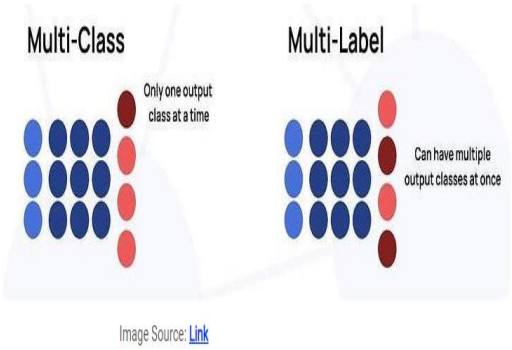


Image Source: Link

What is Binary Classification?

In binary classification problem statements, any of the samples from the dataset takes only one label out of two classes.

For example, Let's see an example of small data taken from amazon reviews data set.

Review	sentiment
Very good quality though	Positive
The design is very odd	Negative
I advise EVERYONE DO NOT BE FOOLED!	Negative
So Far So Good!	Positive
Works great!	positive

If we carefully look into the table, we will see that we can only classify the review as either positive or negative i.e, only two possible target outcomes. So, this is an example of a

binary classification problem statement.

What is Multi-Label Classification?

Multi-label vs. single-label is the matter of how many classes an object or example can belong to. In neural networks, when single-label is required, we use a single softmax layer as the last layer, learning a single probability distribution that ranges over all classes. In the case where multi-label classification is needed, we use multiple sigmoids on the last layer and thus learn a separate distribution for each class.

In certain problems, each input can have multiple, or even none, of the designated output classes. In these cases, we go for the multi-label classification problem approach.

For example, If we are building a model which predicts all the clothing articles a person is wearing, we can use a multi-label classification model since there can be **more than one possible option at once**.

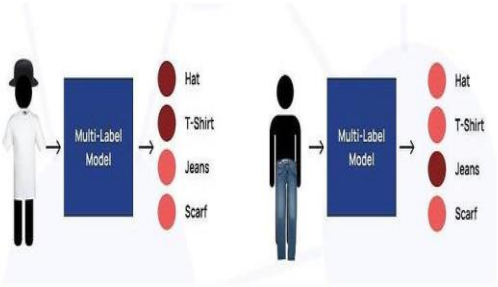


Image Source: Link

If we carefully look into the table, we will see that the movie may take more than one genre i.e, the movie could be comedy and Fantasy at the same time. This means samples can have more than two possible labels.

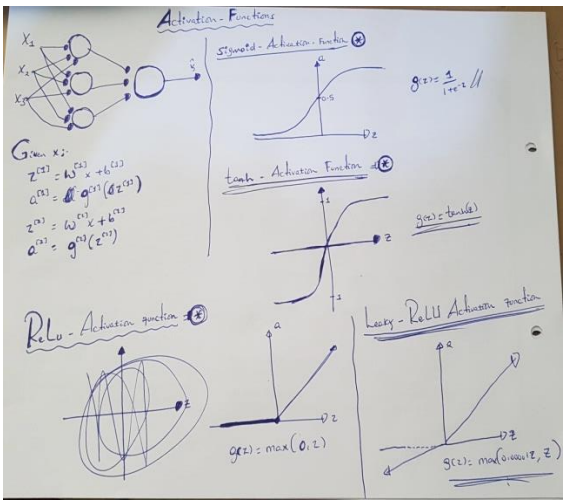
So, this is an example of a multi-label classification problem statement.

Cost Function For Neural Networks

Activation Functions For Neural Networks

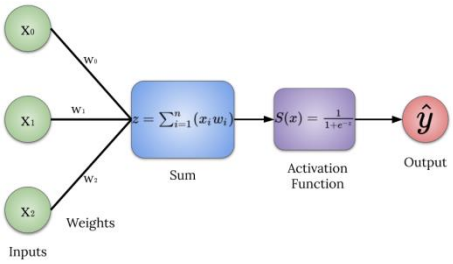
The Objective

- The Limitation of Sigmoid and Tanh Activation Functions
- What is ReLu and How ReLu Overcome the Limitations of Sigmoid
- How ReLu Activation Functions Work
- The Advantages Of ReLu Activation Functions
- How to choose activation function for our neural networks
- Why non-linear activation functions



Limitations of Sigmoid and Tanh Activation Functions

A neural network is comprised of layers of nodes and learns to map examples of inputs to outputs
In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

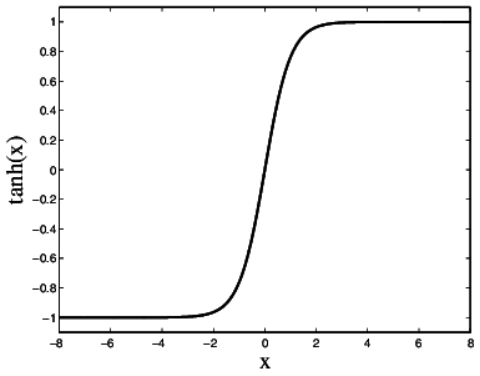


For a given node, the inputs are multiplied by the weights in a node and summed together. This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output or "activation" of the node.

The simplest activation function is referred to as the *linear activation*, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity (e.g. regression problems).

Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the ***sigmoid*** and ***hyperbolic tangent*** activation functions.
The sigmoid activation function, also called the logistic function, is traditionally a very popular activation function for neural networks. The input to the function is transformed into a value between 0.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0. The shape of the function for all possible inputs is an S-shape from zero up through 0.5 to 1.0. For a long time, through the early 1990s, it was the default activation used on neural networks.

The hyperbolic tangent function, or tanh for short, is a similar shaped nonlinear activation function that outputs values between -1.0 and 1.0. In the later 1990s and through the 2000s, the tanh function was preferred over the sigmoid activation function as models that used it were easier to train and often had better predictive performance.



The input to the function is transformed into a value b/n -1.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than -1.0 are snapped to -1.0. The hyperbolic tangent activation function typically performs better than the logistic sigmoid. Why? *lways mean of tanh function would be closer to zero when compared to sigmoid.* It can also be said that data is centered around zero for tanh (centered around zero is nothing but mean of the input data is around zero. This approach(normalization) should be applied at all the layers in the network, which means that the average of the *outputs* of a node to be close to zero because these outputs are the inputs to the next layer.
Convergence is usually faster if the average of each input variable over the training set is close to zero.
The network training converges faster if its inputs are whitened — i.e., linearly transformed to have zero means As each layer observes the inputs produced by their previous layers, it would be advantageous to achieve the same whitening of the inputs of each layer.

Limitation of both Tanh and sigmoid activation function

At a very large value of z on either side of z , the slope will tend to be zero , these slows down the gradient descent , because the learning rate is proportional to the slope.

The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

Although the use of nonlinear activation functions allows neural networks to learn complex mapping functions, they effectively prevent the learning algorithm from working with deep networks.

The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

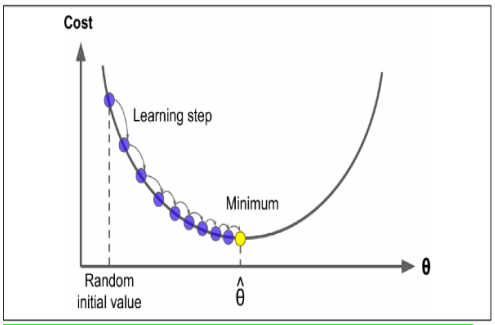


Figure 4-3. In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function; the learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum

The **rectified linear activation function** or **ReLU** for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

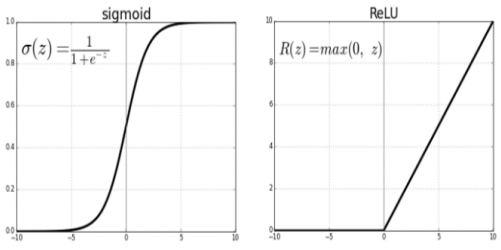


Fig: ReLU v/s Logistic Sigmoid

The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

```
if input > 0:
    return input
else:
    return 0
```

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. But how , if it acts like a linear function , how can that able us to understand complex structures ?

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero. Since it is linear for the value of z>0 , so the gradient will be very fast because linear function has a convex gradient descent which we can converge to the global optima rapidly.

We use ReLu function if we our hypothesis is positive , the slope for z<0 is 0 but almost all of the units in our neural network has a value greater than zero , because our hypothesis representation.

The slope for the value of z>0 is 1 because the first derivative of the linear function is a constant.

But the question is , for the value z>0 it is linear function , so our gradient descent will converge faster l, that is god but , since it is linear there will never be any transformation , the next layer will never learn from the previous layer , because the sum of two linear function is also another linear function , so having one layer stacked up or many layers doesn't make any difference , how do we handle that ?

Problem with ReLu

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

Leak ReLu

It is an attempt to solve the dying ReLU problem

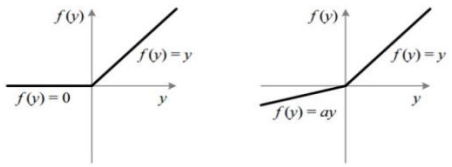
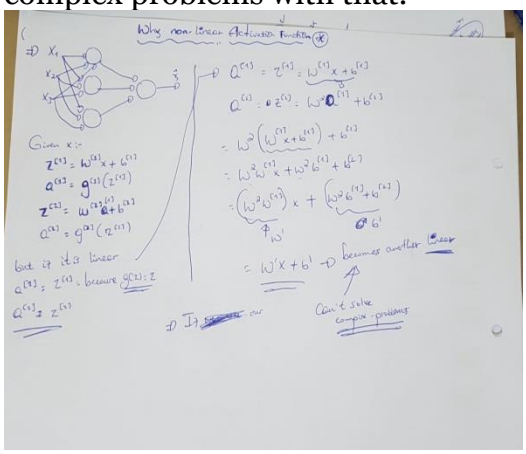


Fig : ReLU v/s Leaky ReLU

Why non-linear activation functions

Why do we need a non linear activation functions in the first place? Well, if you chain several linear transformations, all you get is a linear transformation. For example, if $f(x) = 2x + 3$ and $g(x) = 5x - 1$, then chaining these two linear functions gives you another linear function: $f(g(x)) = 2(5x - 1) + 3 = 10x + 1$. So if you don't have some nonlinearity between layers, then even a deep stack of layers is equivalent to a single layer, and you can't solve very complex problems with that.



How to choose activation function for our neural networks

Activation functions are a critical part of the design of a neural network.
The choice of activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make.

As such, a careful choice of activation function must be made for each deep learning neural network project.

Activation Functions

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

Sometimes the activation function is called a "*transfer function*." If the output range of the activation function is limited, then it may be called a "*squashing function*." Many activation functions are nonlinear and may be referred to as the "*nonlinearity*" in the layer or the network design.

The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

A network may have three types of layers: input layers that take raw input from the domain, **hidden layers** that take input from another layer and pass output to another layer, and **output layers** that make a prediction.

All hidden layers typically use the same activation function. The output layer will typically use a different activation function from the hidden layers and is dependent upon the

type of prediction required by the model.

Activation functions are also typically differentiable, meaning the first-order derivative can be calculated for a given input value. This is required given that neural networks are typically trained using the backpropagation of error algorithm that requires the derivative of prediction error in order to update the weights of the model.

There are many different types of activation functions used in neural networks, although perhaps only a small number of functions used in practice for hidden and output layers.

Activation for Hidden Layers

A hidden layer in a neural network is a layer that receives input from another layer (such as another hidden layer or an input layer) and provides output to another layer (such as another hidden layer or an output layer).

A hidden layer does not directly contact input data or produce outputs for a model, at least in general.

A neural network may have zero or more hidden layers.

Typically, a differentiable nonlinear activation function is used in the hidden layers of a neural network. This allows the model to learn more complex functions than a network trained using a linear activation function.

There are perhaps three activation functions you may want to consider for use in hidden layers; they are:

- Rectified Linear Activation (**ReLU**)
- Logistic (**Sigmoid**)
- Hyperbolic Tangent (**Tanh**)

ReLU Hidden Layer Activation Function

The rectified linear activation function, or ReLU activation function, is perhaps the most common function used for hidden layers.
It is common because it is both simple to implement and effective at overcoming the limitations of other previously popular activation functions, such as Sigmoid and Tanh. Specifically, it is less susceptible to vanishing gradients that prevent deep models from being trained, although it can suffer from other problems like saturated or "*dead*" units.

The ReLU function is calculated as follows:

- max(0.0, x)

This means that if the input value (x) is negative, then a value 0.0 is returned, otherwise, the value is returned.

Sigmoid Hidden Layer Activation Function

The sigmoid activation function is also called the logistic function.

It is the same function used in the logistic regression classification algorithm.

The function takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

The sigmoid activation function is calculated as follows:

- 1.0 / (1.0 + e^-x)

Tanh Hidden Layer Activation Function

The hyperbolic tangent activation function is also referred to simply as the Tanh (also "*tanh*" and "*TanH*") function.
It is very similar to the sigmoid activation function and even has the same S-shape.

The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

The Tanh activation function is calculated as follows:

- (e^x – e^-x) / (e^x + e^-x)

How to Choose a Hidden Layer Activation Function

A neural network will almost always have the same activation function in all hidden layers.

It is most unusual to vary the activation function through a network model.

Traditionally, the sigmoid activation function was the default activation function in the 1990s. Perhaps through the mid to late 1990s to 2010s, the Tanh function was the default activation function for hidden layers.

The hyperbolic tangent activation function typically performs better than the logistic sigmoid.

Both the sigmoid and Tanh functions can make the model more susceptible to problems during training, via the so-called vanishing gradients problem. The activation

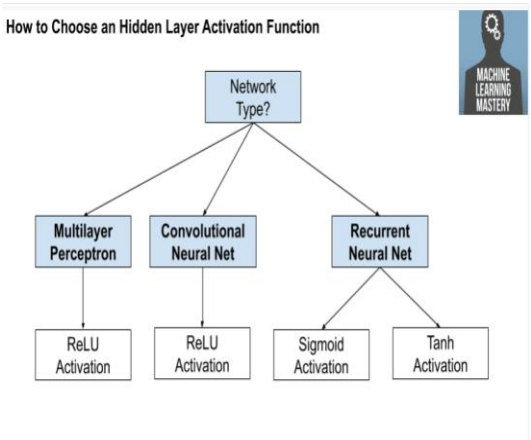
function used in hidden layers is typically chosen based on the type of neural network architecture.

Modern neural network models with common architectures, such as MLP and CNN, will make use of the ReLU activation function, or extensions.

In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU ...

Recurrent networks still commonly use Tanh or sigmoid activation functions, or even both. For example, the LSTM commonly uses the Sigmoid activation for recurrent connections and the Tanh activation for output.

Multilayer Perceptron (MLP): ReLU activation function.
Convolutional Neural Network (CNN): ReLU activation function.
Recurrent Neural Network: Tanh and/or Sigmoid activation function.



Activation for Output Layers

The output layer is the layer in a neural network model that directly outputs a prediction.

All feed-forward neural network models have an output layer.

There are perhaps three activation functions you may want to consider for use in the output layer; they are:

- Linear
- Logistic (Sigmoid)
- Softmax

Linear Output Activation Function

The linear activation function is also called "*identity*" (multiplied by 1.0) or "*no activation*."
This is because the linear activation function does not change the weighted sum of the input in any way and instead returns the value directly.

Sigmoid Output Activation Function

Target labels used to train a model with a sigmoid activation function in the output layer will have the values 0 or 1.

Softmax Output Activation Function

The softmax function outputs a vector of values that sum to 1.0 that

can be interpreted as probabilities of class membership.
Softmax outputs a 0 for all options and 1 for the chosen option. that allows a probability-like output of a winner-take-all function(One versus all method).
As such, the input to the function is a vector of real values and the output is a vector of the same length with values that sum to 1.0 like probabilities.

How to Choose an Output Activation Function

You must choose the activation function for your output layer based on the type of prediction problem that you are solving.

Specifically, the type of variable that is being predicted.

For example, you may divide prediction problems into two main groups, predicting a categorical variable (*classification*) and predicting a numerical variable (*regression*).

Regression MLP(Multi Layer Perceptron)

First, MLPs can be used for regression tasks. If you want to predict a single value (e.g.,
the price of a house, given many of its features), then you just need a single output
neuron: its output is the predicted value. For multivariate regression (i.e., to predict multiple values at once), you need one output neuron per output dimension. Forexample, to locate the center of an object in an image, you need to predict 2D coordinates, so you need two output neurons. If you also want to place a bounding box around the object, then you need two more numbers: the width and the height of the object. So, you end up with four output neurons.

In general, when building an MLP for regression, you do not want to use any activation function for the output neurons, so they are free to output any range of values. So you should use a linear activation function , directly output the input .

But we can also use ReLu for positive values , for regression problems the summed inputs(Z) are always greater than zero , so we can use ReLu activation function as output.

Regression problems can not have a linear activation function as their hidden layers.

Loss function for regression problems :- The loss function to use during training is typically the mean squared error, but if you have a lot of outliers in the training set, you may prefer to use the mean absolute error instead. Alternatively, you can use the Huber loss, which is a combination of both.

Table 10-1. Typical regression MLP architecture

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Classification MLPs

Predicting a probability is not a regression problem; it is classification. In all cases of classification, your model will predict the probability of class membership (e.g. probability that an example belongs to each class) that you can convert to a crisp class label by rounding (for sigmoid) or argmax (for softmax).

If there are two mutually exclusive classes (binary classification), then your output layer will have one node and a sigmoid activation function should be used. If there are more than two mutually exclusive classes (multiclass classification), then your output layer will have one node per class and a softmax activation should be used. If there are two or more mutually inclusive classes (multilabel classification), then your output layer will have one node for each class and a sigmoid activation function is used.

MLPs can also be used for classification tasks. For a binary classification problem, you just need a single output neuron using the logistic activation function: the output will be a number between 0 and 1, which you can interpret as the estimated probability of the positive class. The estimated probability of the negative class is equal to one minus that number.

MLPs can also easily handle multilabel binary classification tasks (see Chapter 3). For example, you could have an email classification system that predicts whether each incoming email is ham or spam, and simultaneously predicts whether it is an urgent or non-urgent email. In this case, you would need two output neurons, both using the logistic activation function: the first would output the probability that the email is spam, and the second would output the probability that it is urgent. More generally, you would dedicate one output neuron for each positive class. Note that the output probabilities do not necessarily add up to 1. This lets the model output any combination of labels: you can have non-urgent ham, urgent ham, non-urgent spam, and perhaps even urgent spam (although that would probably be an error).

If each instance can belong only to a single class, out of three or more possible classes (e.g., classes 0 through 9 for digit image classification), then you need to have one output neuron per class, and you should use the softmax activation function for the whole output layer (see Figure 10-9). The softmax function (introduced in Chapter 4) will ensure that all the estimated probabilities are between 0 and 1 and that they add up to 1 (which is required if the classes are exclusive). This is called multiclass Classification.

- Binary Classification: One node, sigmoid activation.
- Multiclass Classification: One node per class, softmax activation.
- Multilabel Classification: One node per class, sigmoid activation.

Regarding the loss function, since we are predicting probability distributions, the cross-entropy loss (also called the log loss, see Chapter 4) is generally a good choice.

Table 10-2. Typical classification MLP architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

Derivation For Deep Learning and Neural Networks

Derivation For Deep Learning and Neural Networks

Objective Of This Note :-

What is Derivation of a Function

Why do I need to study a Derivative Function

What is Chain Rule ?

Why do I need to study a Chain Rule ?

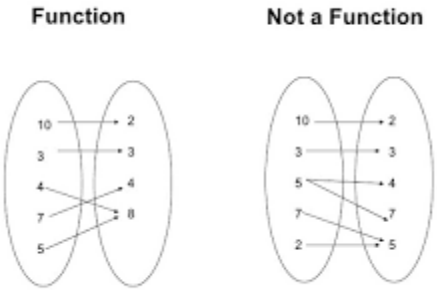
What is the n^{th} Derivative ?

Why do I need to calculate higher order derivatives?

What is Function

A Function is mapping from a set of inputs (The domain) to a set of possible outputs(The co Domain). A Function takes a number as its input and producing another number as its output.

A Technical Definition of a function is : A Relation from a set of inputs to a set of possible outputs where each input is related to exactly one output.



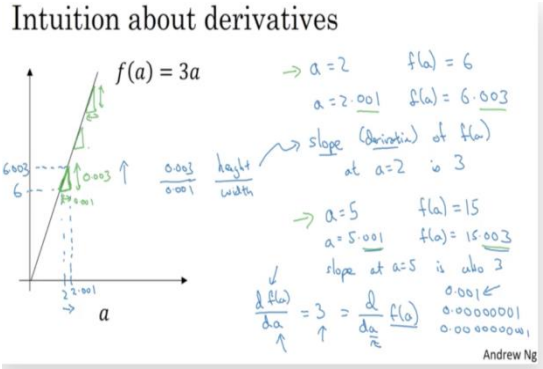
This means that if the object X is in the set of inputs (Called the domain) then a Function f will map the object X to exactly one object f(x) in the set of possible outputs(Called the Co Domain).

A Function is more formally given a set of inputs X (Domain) and a set of possible outputs Y (Co domain).

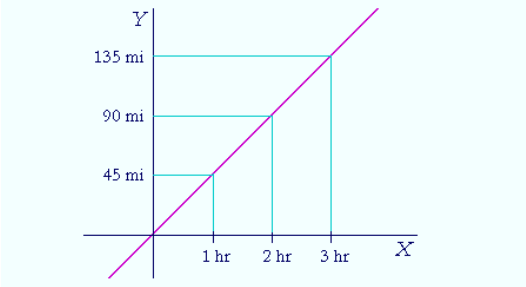
What is Derivation

Consider X to be independent variable and y the dependent variable , then any change ΔX in the value of the x will result in a change ΔY in the value of the y.

- So the whole idea of derivation is to find the rate of change – How many unit of Y for a tiny unit of nudge in X.



-In a straight line , the rate of change is constant for any value of X , if you nudge the value of by the same amount.



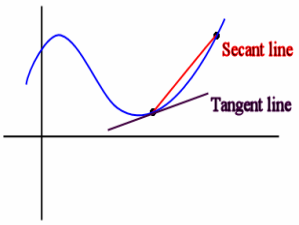
If this line represents distance y versus time x then the rate f change is speed , So for any value of X(time) if you nudge is the same then the speed will also be constant.

Secant Lines, Tangent Lines, and Limit Definition of a Derivative

A secant line is a straight line joining two points on a function , it is also equivalent to average rate of change or simply a slope between two points.

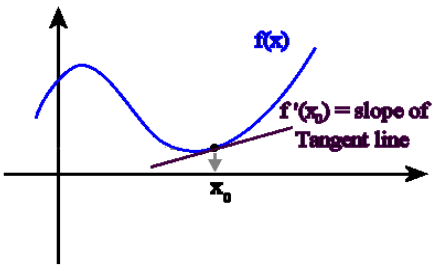
$$slope = m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(t_1) - f(t_0)}{t_1 - t_0} = Average\ Rate\ of\ Change$$

Secant line = Average Rate of Change = Slope



A tangent line is a straight line that touches a function at only one point. The tangent line represents Instantaneous Rate of change of a function at that one point. The slope of a tangent line at a point on the function is equal to the derivative of the function at the same point.

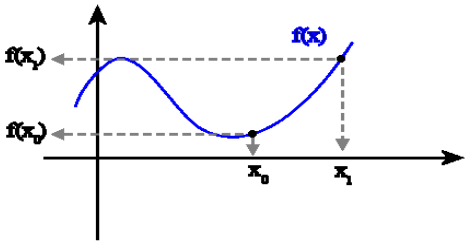
Why are we calling the slope of a tangent line as **instantaneous rate** of change ? Because the change in the **rate** is done at a particular instant of point, and it is same as the change in the derivative value at a specific point.



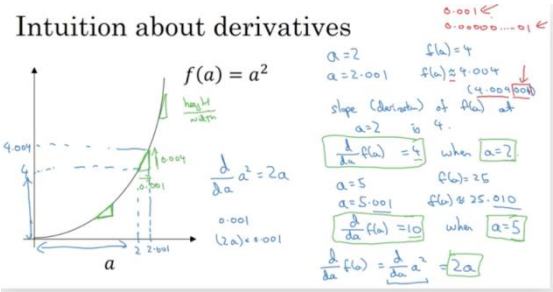
Tangent Line = Instantaneous Rate of Change = Derivative

Let's see what happens as the two points used for the secant line get closer to one another. Let Dx represent the distant between the two points along the x-axis and determine the limit as Dx approaches zero.

$$f'(x_0) = \lim_{(x_1 - x_0) \to 0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}, \text{where: } x_1 = (x_0 + \Delta x), \text{and: } \Delta x = (x_1 - x_0)$$

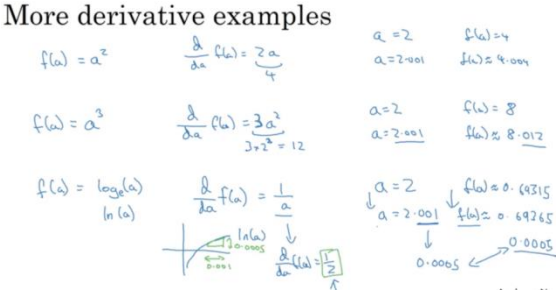


As the two points used for the secant line get closer to one other , the average rate of change becomes Instantaneous rate of change and the secant line becomes the tangent line.



The value of f(a) changes by the rate of 10* as you change a=5 a little bit.

At different points of the curve , the slope is different even there is the same nudge.



In practice, we have to simplify the difference quotient before letting *h* approach 0. We have to express the numerator --

$f(x + h) - f(x)$

Example :- let $y = x^2 + 3x$

a) The average rate of change of y with respect to a nudge in the x[1,3]
b) The Instantaneous rate of change of y with respect at x = 1
c) The slope of secant line passing through (-1 , -2) and (2 , 10)

Solution :-

a) $X_0 = 1$ and $x1 = 3$ -> $f(x_0) = 4$ and $f(x1) = 18$

Slope = $f(x_1) - f(x_0) / x_1 - x_0$

b) $\lim \frac{f(x_1) - f(x_0)}{x_1 - x_0} = x^2 + 3x - 4 / x - 1 =$

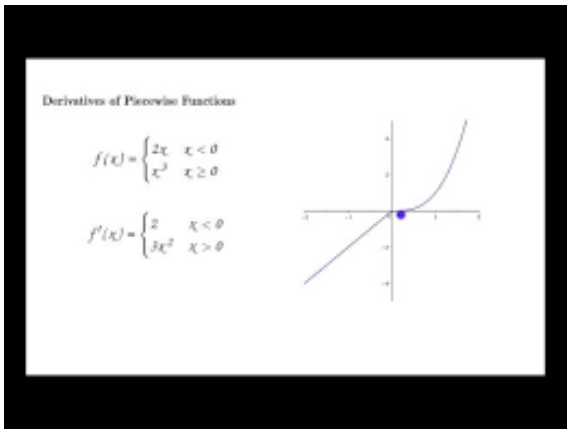
$x_1 \to x_0$

c) The slope of the secant line passing through (-1 , -2) and (2 , 10)

slope = $f(x_1) - f(x_0) / x_1 - x_0 = 10 - (-2) / 3 = 4$ //

Calculating the derivative of a piecewise functions

A piece wise function is commonly made up of two smooth piece joined together at one point.



The definition of the derivative is to find the derivative function of each smooth piece and check whether these functions agree at that chosen point. Showing that this strategy works together with investigating discontinuities.

What is Differentiable

Differentiable means that the function has a derivative . In simple terms , it means there is a slope (one that you can calculate) . The slope will tell you about the rate of change.

The Derivative must exist for all points in the domain , otherwise the function is not differentiable . This might happen when you have a hole in the graph , there's no slope.

In general , a function is not differentiable t three reasons

- Vertical Tangent Line
- Sharp Corner
- Discontinuity

How to check a function is differentiable at a given point x.

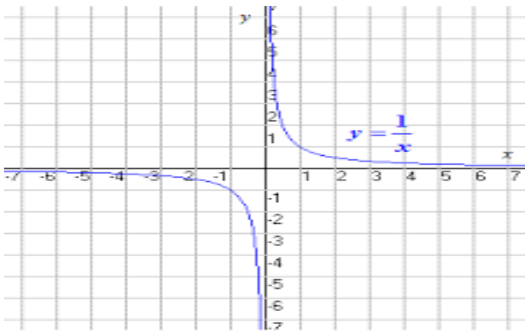
- Check the limit exists at x = a

Lim $f(x_1) - f(x_0) / x_1 - x_0$, as $x_1 \Rightarrow x_0$ if these exist

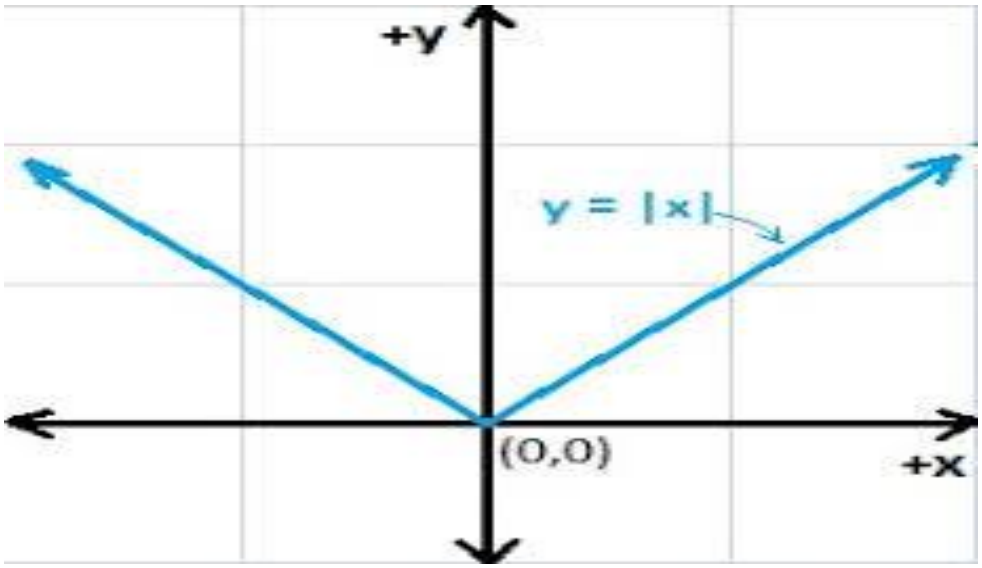
- If the left and the right limit are equal

Lim $f(x) - f(a) / x - a$ as $x \to a^+$

Lim $f(x) - f(a) / x - a$ as $x \to a^-$



The limit does not have a derivative at x = 0 because the function is discontinuous there at x = 0.



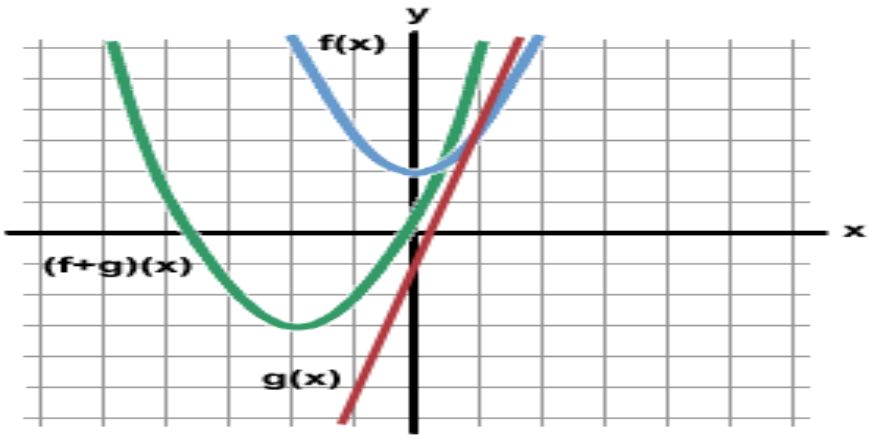
Example :- $f(x) = x^2 + 2$ $g(x) = 4x - 1$, then $(f+g)(1)$

$f(1) = 1 + 2 = 3$ // $g(1) = 4(1) - 1 = 3$ //

$(f + g)(x) = f(x) + g(x)$

$= (x^2 + 2) + (4x - 1)$

$= x^2 + 4x + 1$ //



The y – coordinate of each point on the graph of y = (f+g)(x) is the result of adding the y –coordinate of g(x) to the y – coordinate of f(x). Example:- f(1) = 3 , g(1) = 3 (f+g)(1) = 3 + 3 = 6 //

Derivative of sum and Difference

If f and g are differentiable at x_o , then f + g and f – g are also differentiable at x_o. so that :-

(f + g)'(x_o) = f'(xo) + g'(xo) => Sum rule

(f-g)'(x_o)=f'(xo)-g'(xo)=>Differencerule

Example:- find a
f(x) = $x^3 - 5x$

f(a) = $a^3 - 5a$
f'(a) = $3a^2 - 5$
$13 + 5 = 3a^2$
$a^2 = 18 / 3$
$a^2 = 6$ //

Derivatives of product and quotient of function

If f and g are differentiable at x then the product of fg is differentiable at x and it's derivative is given by

(fg)'(x) = f'(x)g(x) + f(x)g'(x)

(fgh)'(x) = f'(x)g(x)h(x) + f(x)g'(x)h(x) + f(x)g(x)h'(x)

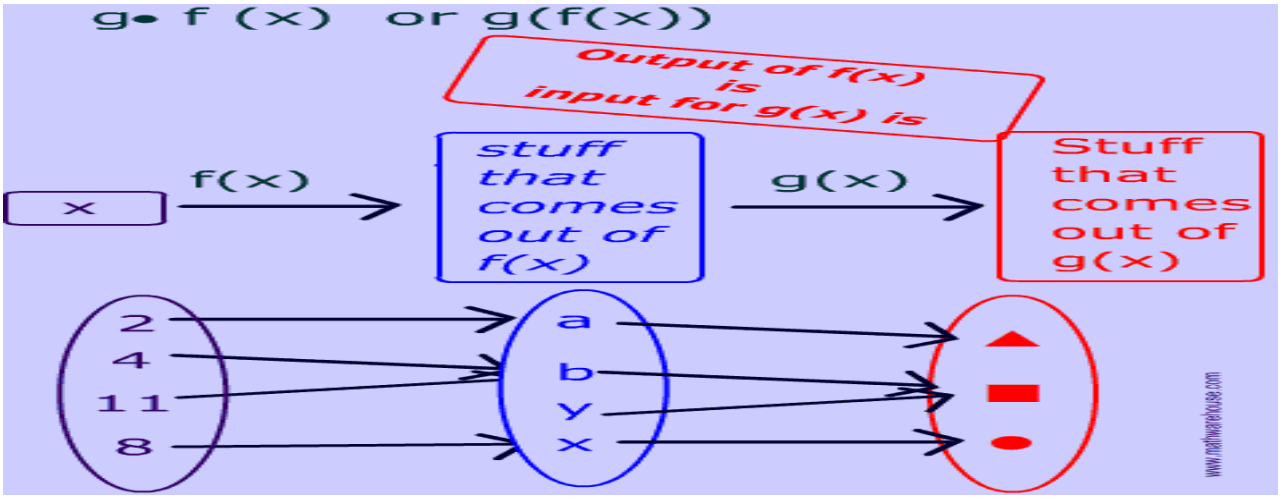
Quotient Rule :- (f/g)'(x) = g(x)f'(x) – f(x)g'(x) /$(g(x))^2$

The chain Rule

The chain rule provides a technique for finding the derivative of composite functions.

What are composite functions ?

-It is a kind of functions made of other functions the output of one function is the input to the other.



Chain Rule :- If g is differentiable at x and f is differentiable at g(x) then f(g(x)) is differentiable at x

Y = f(g(x))' = f'(g(x))g'(x)

It is sometimes easier to think of the functions f and g as "Layers" of a problem . Function f is the outer layer and function g is the inner layer , Thus the chain rule tells us to first differentiate the outer layer unchanged , leaving the inner layer unchanged (the term f'(g(x))) , then differentiate the inner layer (the term g'(x)).

Y = f(g(x))

let g(x) = u , du/dx -> the inner layer

Y = f(u) = d(y)/d(u) -> the outer layer

dy/dx = dy/du.du/dx -> the chain rule

Example:- y = 1/ x+4 , let u = x+4

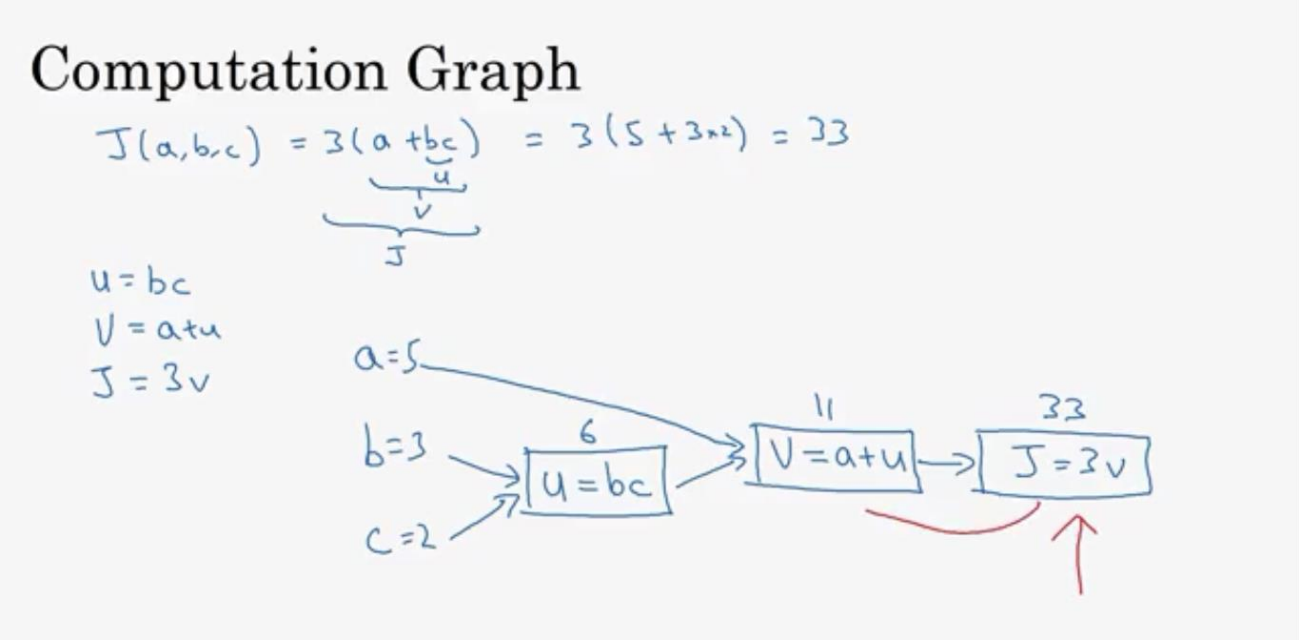
dy/du = 1/u and du/dx = 1

dy/dx = dy/du . du/dx = -1 / (x+4) ^2 //

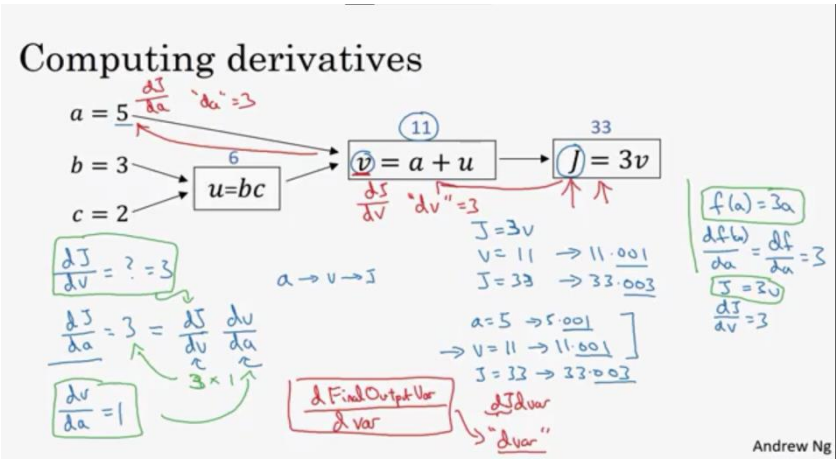
Computation Graphs :- The computation of a neural network organizes in terms of forward path (forward propagation step) which compute the output of the neural network followed by a backward path (a backward propagation step) which is used to compute gradients or derivatives.

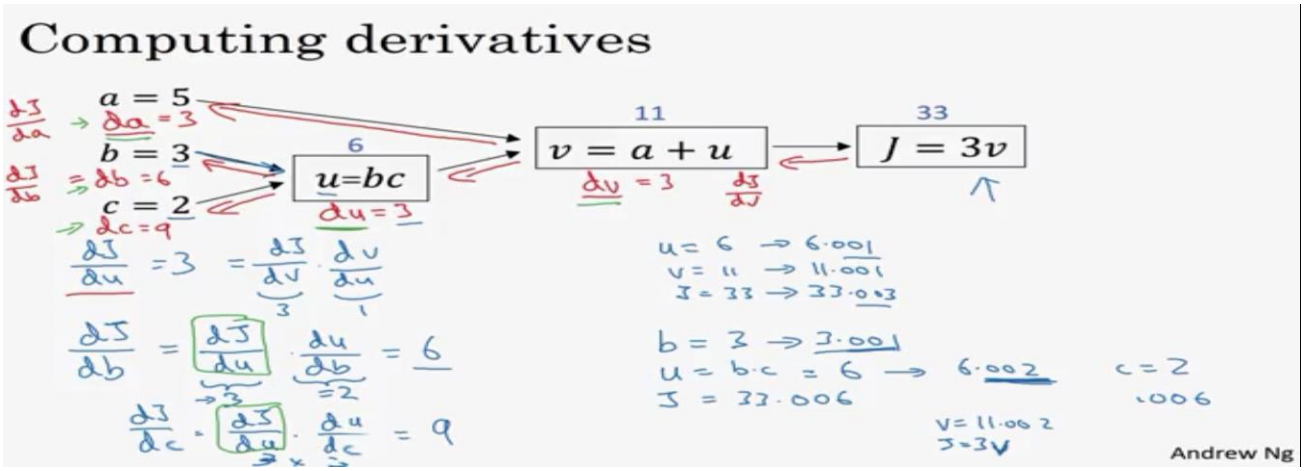
The computation graph explains why it organizes these way. Example :- Let's a function J(a , b , c) = 3(a+bc)

Computing these function got three distinct steps , first computing bc , then (a+bc) then J(a+bc) , three layers f(g(h(x))) , f(x) , g(x) , h(x)



Derivatives Of A Computation Graph





That was the computation graph , go from the left to the right to compute the cost function , such as J you want to optimize and backward from the right to the left to calculate derivation.

BackPropagation

An MLP is composed of one (pass-through) *input layer*, one or more layers of TLUs, called *hidden layers*, and one final layer of TLUs called the *output layer* (see Figure 10-7). The layers close to the input layer are usually called the *lower layers*, and the ones close to the outputs are usually called the *upper layers*. Every layer except the output layer includes a bias neuron and is fully connected to the next layer.

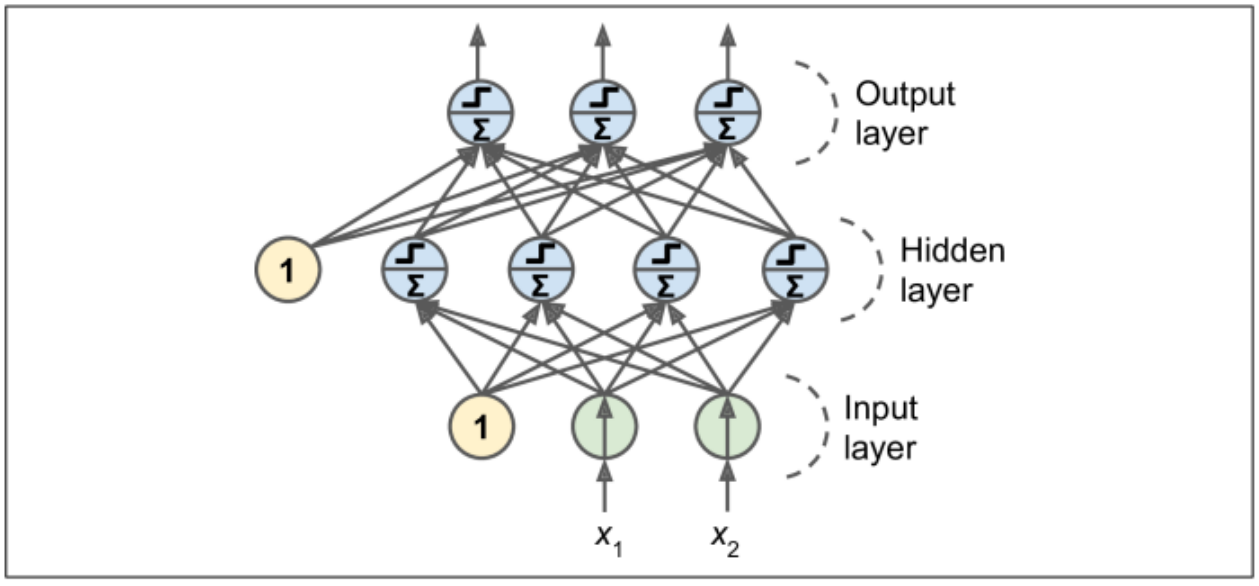


Figure 10-7. Architecture of a Multilayer Perceptron with two inputs, one hidden layer of four neurons, and three output neurons (the bias neurons are shown here, but usually they are implicit)

The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a *feed-forward neural network* (FNN).

When an ANN contains a deep stack of hidden layers,9 it is called a *deep neural net- work* (DNN). The field of Deep Learning studies DNNs, and more generally models containing deep stacks of computations. Even so, many people talk about Deep Learning whenever neural networks are involved (even shallow ones). For many years researchers struggled to find a way to train MLPs, without success. But in 1986, David Rumelhart, Geoffrey Hinton, and Ronald Williams published agroundbreaking paper10 that introduced the *backpropagation* training algorithm, which is still used today. In short, it is Gradient Descent (introduced in Chapter 4) using an efficient technique for computing the gradients automatically:11 in just two passes through the network (one forward, one backward), the backpropagation algo- rithm is able to compute the gradient of the network's error with regard to every sin- gle model parameter. In other words, it can find out how each connection weight and each bias term should be tweaked in order to reduce the error. Once it has these gradients, it just performs a regular Gradient Descent step, and the whole process is repeated until the network converges to the solution.

Let's run through this algorithm in a bit more detail:

• It handles one mini-batch at a time (for example, containing 32 instances each), and it goes through the full training set multiple times. Each pass is called an *epoch*.

• Each mini-batch is passed to the network's input layer, which sends it to the first hidden layer. The algorithm then computes the output of all the neurons in this layer (for every instance in the mini-batch). The result is passed on to the next-layer, its output is computed and passed to the next layer, and so on until we get the output of the last layer, the output layer. This is the *forward pass*: it is exactly like making predictions, except all intermediate results are preserved since they are needed for the backward pass.

• Next, the algorithm measures the network's output error (i.e., it uses a loss function that compares the desired output and the actual output of the network, and returns some measure of the error).

• Then it computes how much each output connection contributed to the error. This is done analytically by applying the *chain rule* (perhaps the most fundamental rule in calculus), which makes this step fast and precise.

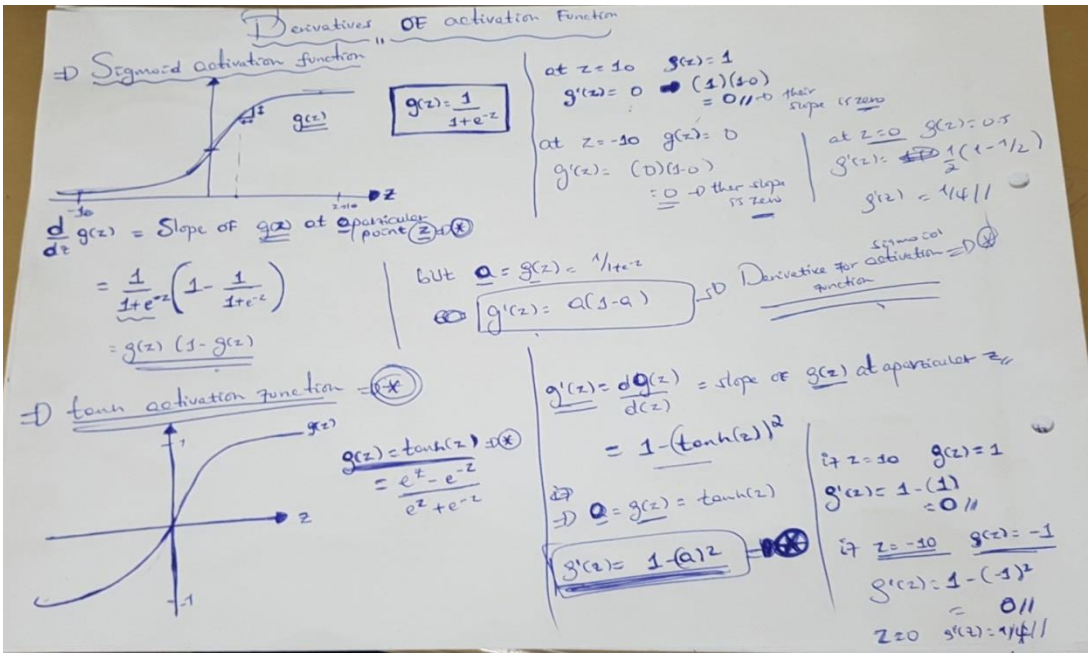
The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule, working backward until the algorithm reaches the input layer. As explained earlier, this reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).

• Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.

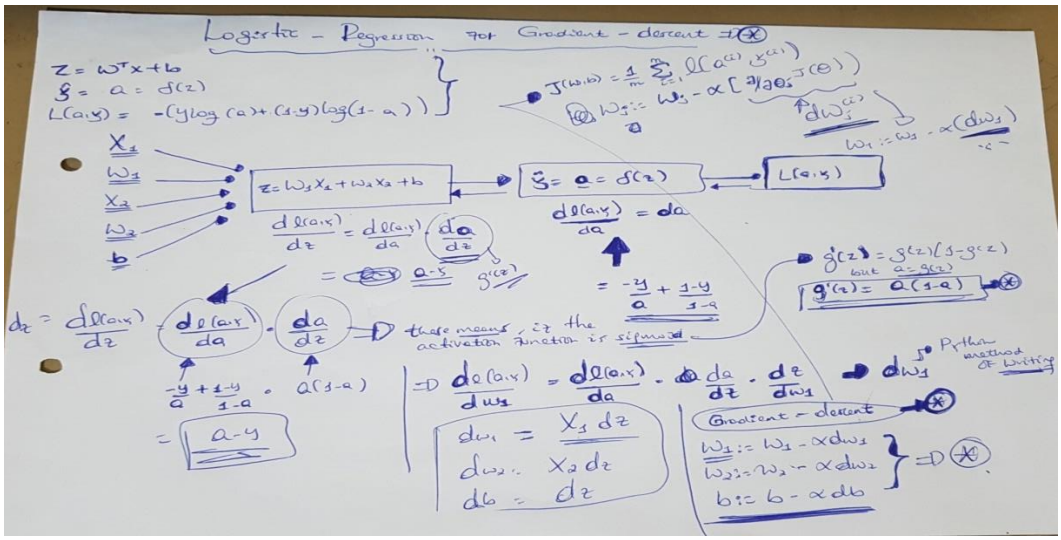
This algorithm is so important that it's worth summarizing it again: for each training instance, the back-propagation algorithm first makes a prediction (forward pass) and measures the error, then goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and finally tweaks the connection weights to reduce the error (Gradient Descent step).

It is important to initialize all the hidden layers' connection weights randomly, or else training will fail. For example, if you initialize all weights and biases to zero, then all neurons in a given layer will be perfectly identical, and thus back-propagation will affect them in exactly the same way, so they will remain identical. In other words, despite having hundreds of neurons per layer, your model will act as if it had only one neuron per layer: it won't be too smart. If instead you randomly initialize the weights, you *break the symmetry* and allow - to train a diverse team of neurons.

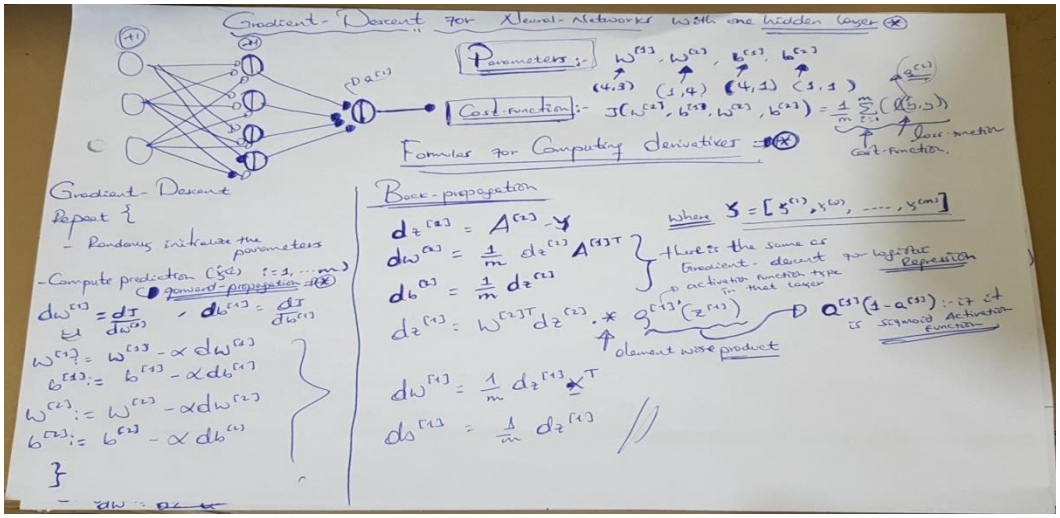
Derivatives Of Activation Function



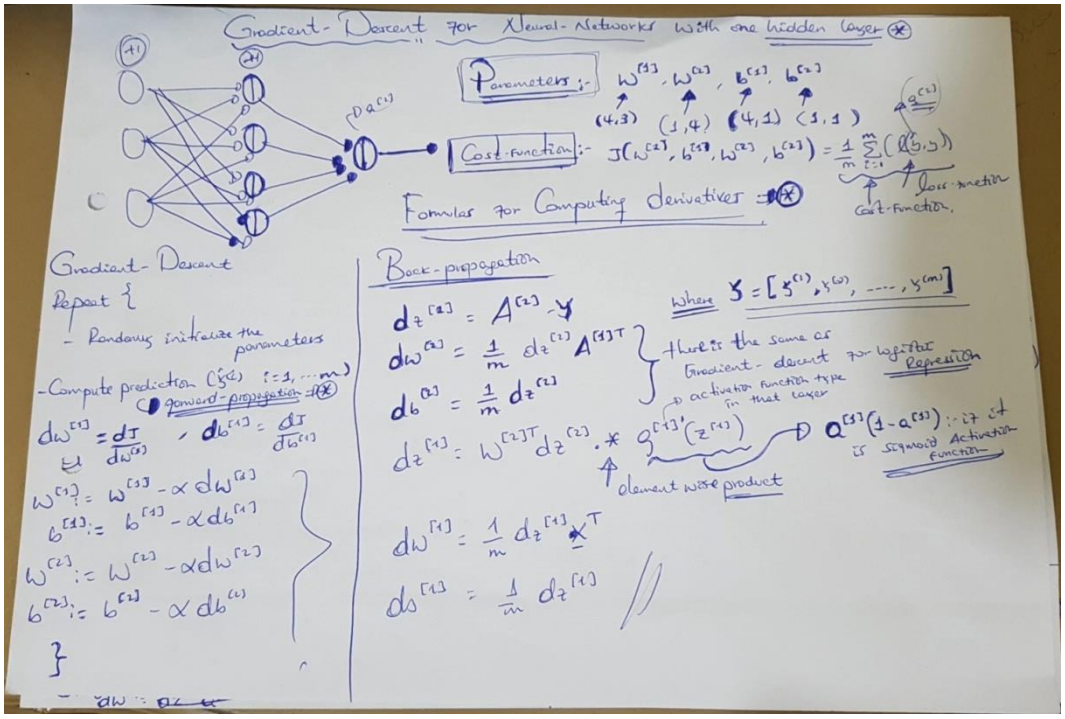
Logistic Regression For Gradient Descent



Vectorized Implementation Of Logisitc Regression With the entire dataset



Gradient Descent For Neural Networks With One Hidden Layer



How Really does BackPropagation Works ?