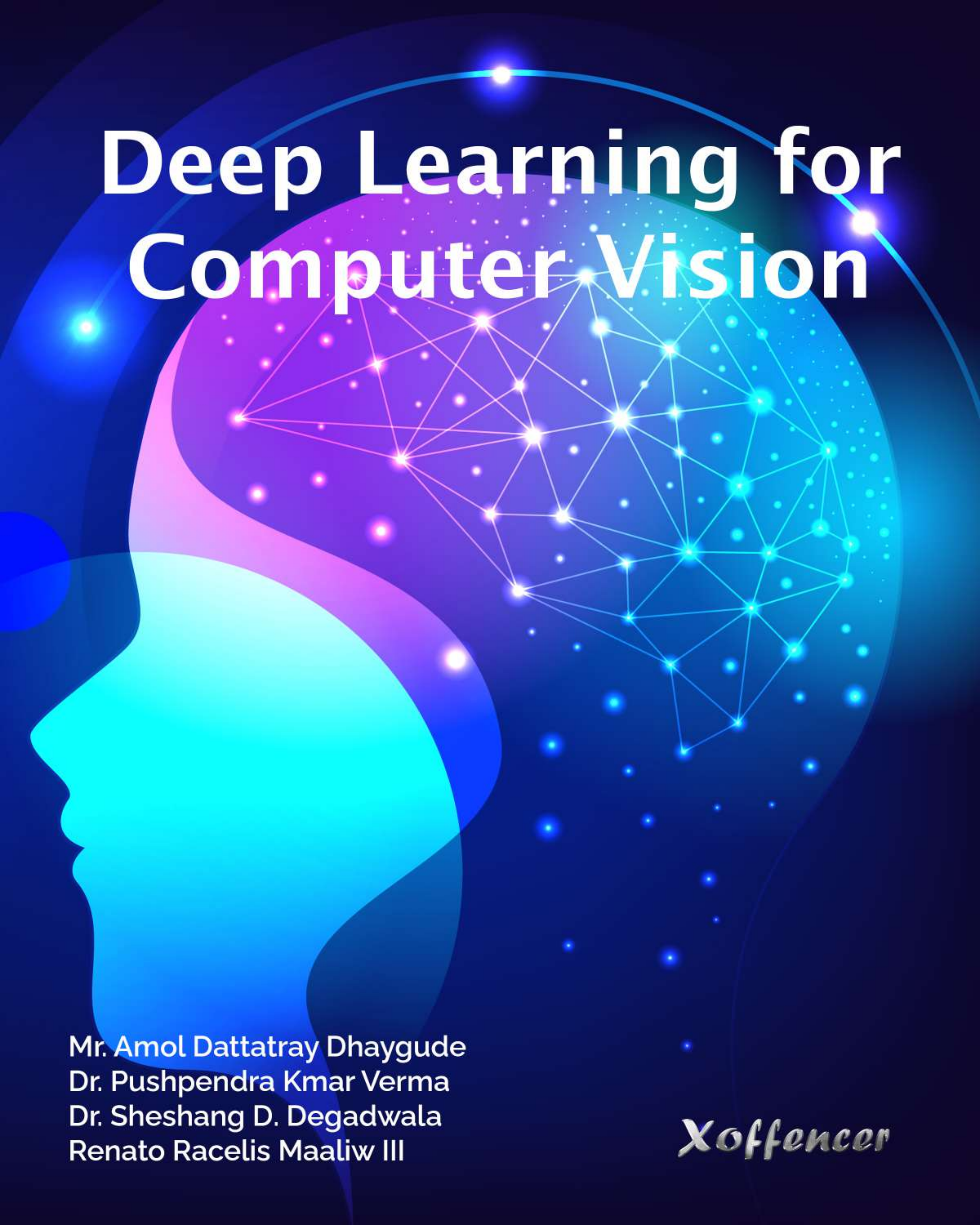
Deep Learning for Computer Vision



Mr. Amol Dattatray Dhaygude
Dr. Pushpendra Kmar Verma
Dr. Sheshang D. Degadwala
Renato Racelis Maaliw III

Xoffencer

DEEP LEARNING FOR COMPUTER VISION

Authors:

Mr. Amol Dattatray Dhaygude

Dr. Pushpendra Kmar Verma

Dr. Sheshang D. Degadwala

Renato Racelis Maaliw III

Xoffencer

www.xoffencerpublication.in

ISBN-13: 978-93-94707-79-5 (paperback)

Publication Date: 24 April 2023

MRP: ₹450/-

ISBN

9 789394 707795

Published by:

Xoffencer International Publication

Shyam Vihar Vatika, Laxmi Colony

Dabra, Gwalior, M.P. – 475110

Cover Page Designed by:

Satyam soni

Contact us:

Email: xoffencer@gmail.com

Visit us: www.xofferncerpublication.in

Author Details

Mr. Amol Dattatray Dhaygude

Mr. Amol Dattatray Dhaygude is renowned professional in field of Machine Learning, Artificial Intelligence, Data Science and Computer Science. He is alumni of University of Washington, Seattle, USA with Master of Science in Data Science and specialization in Machine Learning. Amol has 16 years of software industry experience in top tier organizations including IBM, Cognizant and Microsoft Corporation. He is currently employed at Microsoft Corporation for last 10 years in role of Senior Data & Applied Lead at Redmond, Washington. He is inspired to make use of cutting edge technological advancements in field of Machine Learning and Artificial Intelligence to solve real world practical problems making a difference to world. He has strong techno business acumen to formulate and solve business problems with applications of Data Science, Machine Learning and Artificial Intelligence. He is well versed in Deep Learning, Natural Language Processing, and Computer Vision fields of Artificial Intelligence. Amol celebrates growth mindset with continuous learning, embracing challenges, experimentation, fail fast, feedback and continuous improvement principles. He believes in learning from community and at the same time giving back to community by sharing his knowledge through various avenues such as research publications, blogs, patent publications, book publishing etc. Amol has continuously served as editor for books and journals in his areas of expertise.

Dr. Pushpendra Kmar Verma

Dr. Pushpendra Kmar Verma, is an Associate Professor, School of Computer Science and Applications, IIMT University, Meerut, UP, India. He has done MCA, MTech-CSE. MPhil (CS) and Doctorate in Computer Science. His area of research is in Cryptography and Network and Security/Cyber Security, and his other areas of specialization include Artificial Intelligence and Machine Learning. He has Convener, Keynote Speakers and participated in several high-profile conferences, seminars and workshops. The author has many research papers in international journals, review articles in various Scopus Indexed Journals, international journals and interested in academics and research. He has !7+years extensive teaching and research experience in various Universities as well as colleges across India. He also reviewed the Reputed International Journal Papers. He is member of Vidwan, Springer, International Association of Engineers (IAENG),Society of Digital Information and Wireless Communications (SDIWC) etc.

Dr. Sheshang D. Degadwala

Dr. Sheshang D. Degadwala is presently working as Associate Professor and Head of Computer Engineering Department, Sigma University , Vadodara. He has published 235 research papers in reputed international journals and conferences including IEEE, Elsevier and Springer. His main research work focuses on Image Processing, Computer Vision, Information Security, Theory of Computation and Data Mining. He is also Microsoft Certified in Python Programming and Excel. He has published 18 books and he got grant for 3 patents. He has published 125 Indian Patent. He has received 50 awards for academic and research achievement.

Renato Racelis Maaliw III

Renato Racelis Maaliw III is an Associate Professor and currently the Dean of the College of Engineering in Southern Luzon State University, Lucban, Quezon, Philippines. He has a doctorate degree in Information Technology with specialization in Machine Learning, a Master's degree in Information Technology with specialization in Web Technologies, and a Bachelor's degree in Computer Engineering. His area of interest is in artificial intelligence, computer engineering, web technologies, software engineering, data mining, machine learning, and analytics. He has published original researches, a multiple time best paper awardee for various IEEE sanctioned conferences; served as technical program committee for world-class conferences, author, editor and peer reviewer for reputable high-impact research journals.

Preface

The text has been written in simple language and style in well organized and systematic way and utmost care has been taken to cover the entire prescribed procedures for the general audience.

We express our sincere gratitude to the authors not only for their effort in preparing the procedures for the present volume, but also their patience in waiting to see their work in print. Finally, we are also thankful to our publishers **Xoffencer Publishers, Gwalior, Madhya Pradesh** for taking all the efforts in bringing out this book in due time.

Abstract

The first chapter serves as an introduction to our subject matter and elucidates the reasons why it is pertinent to the society of today. At the same time, it sheds light on the issue that we are going to try to solve. In the beginning of the second section, we will finish completing our broad research objectives, and then we will choose the exact goal that we want to accomplish. Next, provide an explanation of the relationship between our overall research aims and our overall research goals. Second, it will investigate a wide range of computer vision and deep learning approaches, all of which have the possibility of being included into the solution in some way. In the third chapter, the emphasis is placed not only on how the answer is implemented and used in reality, but also on how the theory offered in the second chapter contributes to the solution. In this part of the article, we also look at how the answer contributes to the solution. In spite of the fact that these findings are discussed in the fourth part, a more in-depth analysis will be provided in the next section. In the sixth chapter, we address which methods and implementation strategies should be prioritised in the work that will be done in the future. Those who work in the area of innovation have long had the ambition of developing machines that are capable of thought. There is evidence of this requirement dating back to ancient Greece at the very least. Pygmalion, Daedalus, and Hephaestus are just a few of the characters from Greek mythology who have been presented throughout history as famous pioneers in their respective areas. In Greek mythology, there are a number of other characters, such as Galatea, Talos, and Pandora, who are said to symbolise fabricated life. Ever since the idea of programmable computers was first imagined, people have speculated on the likelihood that programmable computers would one day be able to achieve a high level of accuracy.

Contents

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In the most recent few years, tremendous technical progress has been made in the creation of high-throughput graphics processing units in addition to parallel processing. Processing in parallel has allowed for the realisation of these recent advancements. (GPUs). The amount of computational power that is now accessible has significantly risen, yet the needed amount of power consumption has stayed the same. High-performance parallel processing units are now accessible at a price that is affordable for almost everyone. This is because many of these systems are developed for the consumer market to deliver high-definition gaming experiences.

Although they have been considerably altered to make graphical calculations more effective, they are broad enough to be utilised in a range of different jobs that may be completed concurrently. This is despite the fact that they have been adjusted to make graphical calculations more effective. This new advancement will have a tremendous impact on the whole area of study that focuses on deep learning. At this point in time, it is feasible for anybody to use the most recent techniques of deep learning to their work, regardless of whether they are doing their study in conventional laboratories or at home.

Deep learning is a subfield of machine learning that has shown its usefulness for a variety of activities that are deemed simple for humans but too tough for computers to handle on their own. Image recognition, natural language processing, and voice recognition are all examples of the tasks that fall under this category. Natural language processing and image analysis are two examples of this kind of technology. Both of these include the categorization, identification, and segmentation of various items inside pictures.

This paves the way for the development of autonomous systems, which in turn paves the way for an infinite number of additional possibilities. In the beginning, people usually tried to grasp the present condition of the environment by using a mix of very basic sensory inputs, such as laser and radar scanners. On the other hand, if you do not have access to a more extensive detection method, it may be challenging to design a health evaluation that is both comprehensive and accurate. A more capable system is able to comprehend the pictures that are sent to it by a number of different cameras. These pictures may have originated from any number of different places. If a human driver is provided with a live video feed from a 360-degree radius, then the human driver will have access to practically all of the information necessary to operate most automobiles in an appropriate manner.

Because of this, the human pilot will be able to handle the majority of cars in a manner that is both safer and more efficient. The most recent developments in the field of deep learning have made it feasible to develop an approach to image processing that is not only more reliable but also more all-encompassing. These findings also made it feasible, at least in part, to simulate human pattern recognition abilities in machines. A solution that is based only on sensory input may not give as accurate of a forecast of the environment as would be possible with a software system that is capable of capturing the world around it via the use of live images.

This concept finally became a reality as a result of deep learning and the increasing availability of powerful graphics processors, which sparked a revolution in the process of developing autonomous cars. Also seeing a revolution is deep learning, which has been made possible by the widespread availability of powerful graphics processing units. (GPUs). When it comes to pursuing drones, the sky provides some really interesting new locations. It has been shown beyond a reasonable doubt that a variety of multicopters, which are more usually referred to as "drones" since they operate in areas that are not under human control, are capable of being piloted by an automated system.

Because of the atmosphere in which they travel, this is the case. Drones, in comparison to bigger vehicles such as automobiles, have a size that is substantially more manageable, have a significantly reduced danger of causing injury, and are subject to a significantly smaller number of legal constraints on how they may be used in the community. In addition to these benefits, the applications for drones are far less constrained than other technologies. Drones have a far shorter set of needs to complete, and as a result, they only require a fraction of the computational power that is necessary to properly run an autonomous car. In point of fact, drones are far less complicated than automobiles.

It is now feasible, as a result of recent developments in GPU technology and deep learning, to acquire embedded systems little larger than a credit card that are dedicated to the implementation of deep learning algorithms exclusively. These systems, about the size of a credit card, are able to carry out all of the activities necessary to implement deep learning methods. These may be fastened to the drone, where they replace the onboard computer's duties and enhance the aircraft's capacity to carry out responsibilities associated with computing. In 2010, when radio-controlled drones were readily accessible to customers, a boom occurred in the field of research pertaining to autonomous drones. Since then, a number of improvements with capabilities for autonomous flying have been available.

Many of these enhancements are geared at taking photos from an aerial viewpoint. On the other hand, there are more attempts being made towards autonomous flying that are simply intellectual in their character. The International Competition of Aerial Robots (IARC), which took place in 2017, is one example of such an endeavour. The mission for the participating drones is to make precise evaluations of the state of a number of ground robots that are moving in a field around twenty metres distant. .This is also used in order to traverse the sand in an appropriate manner.

1.2 RESEARCH OBJECTIVES AND QUESTIONS

In order to allow the drone to fly autonomously during the IARC flight competition, the objective is to investigate the many methods in which deep learning and computer vision may be utilised to extract robot position data from live raw images acquired on a drone. This will be done using live photos obtained from the drone. In order to acquire location information from raw images, one of the methods that is being researched is the identification of objects and the positioning of these components inside photographs. This category of surgical procedures encompasses a variety of recently discovered methods, including all of them.

Deep learning has been shown to be the most successful solution to such challenges, and it is the first step in building this study on the most recent and cutting-edge approaches for developing neural networks that are both accurate and effective when detecting things. Constructing and running a specialised sensor network in the current world requires a significant investment of time, the majority of which is spent on the creation of a sensor dataset. As a direct consequence of this, our primary objective will be to gather trustworthy training data before we actually begin the process of building the network of locations. This will take place before we begin the process of constructing the network.

Even though the emphasis of this discussion is on the implementation of deep learning in response to the challenge of object recognition, there will be a need for a sizeable quantity of more conventional computer vision techniques to be used in order to prepare the way for the deep learning solution to operate at its absolute best. In point of fact, the objective is to demonstrate that deep learning may be used to find a solution to the challenge of object identification. Once the necessary training set has been assembled, our attention will be directed towards the investigation of several deep learning algorithms, with the goal of identifying the one that provides the optimal balance of speed and accuracy for an integrated detection system that will be utilised during the IARC Competition. Following the completion of the mandatory training package, a determination like this one will be made. In conclusion, one of our primary goals is to seek for answers to the following study questions:

1.3 STRUCTURE

This chapter serves as an introduction to our subject matter and elucidates the reasons why it is pertinent to the society of today. At the same time, it sheds light on the issue that we are going to try to solve. The second section will begin by determining the particular goal that we want to accomplish by finishing it, as well as the manner in which that goal is directly tied to the general objectives of our study. Next, provide an explanation of the relationship between our overall research aims and our overall research goals. Second, it will investigate a wide range of computer vision and deep learning approaches, all of which have the possibility of being included into the solution in some way.

In the third chapter, the emphasis is placed not only on how the answer is implemented and used in reality, but also on how the theory offered in the second chapter contributes to the solution. In this part of the article, we also look at how the answer contributes to the solution. A more in-depth discussion of these findings will follow the presentation of them for your consideration in the fourth part; in the next section, we will give an analysis of these findings in more depth. In the sixth chapter, this results in a happy ending for everyone involved. In this part, we explore which implementation methods and approaches ought to get the most attention in the subsequent work. Those who work in the area of innovation have long had the ambition of developing machines that are capable of thought.

There is evidence of this requirement dating back to ancient Greece at the very least. Pygmalion, Daedalus, and Hephaestus are just a few of the characters from Greek mythology who have been presented throughout history as famous pioneers in their respective areas. In Greek mythology, there are a number of other characters, such as Galatea, Talos, and Pandora, who are said to symbolise fabricated life. Ever since the idea of programmable computers was first imagined, people have speculated on the likelihood that programmable computers would one day be able to achieve a high level of accuracy. This occurred more than a century before to the development of the first computer. The discipline of artificial intelligence (AI) is now a developing area that has a wide range of applications in the actual world as well as ongoing research problems.

The branch of computer science that deals with artificial intelligence is called AI. Intelligent software is used for a range of purposes, including the automation of regular operations, the analysis of voice or pictures, the formation of medical diagnoses, and the assistance in the investigation of scientific subjects. In the early days of artificial intelligence, the field swiftly confronted and solved difficulties that were cognitively tough for humans but relatively simple for computers. This allowed the field to make significant progress in a short amount of time. These are issues that can be solved by using a predetermined set of principles from formal mathematics.

These issues included the following: Because of this, the field was able to develop quite quickly. The ultimate test of artificial intelligence has been shown to be problems to which people respond instantly and apparently effortlessly, such as recognising spoken words or faces in photographs. Examples of this include the identification of spoken words and the recognition of faces in photographs. This competition requires participants to complete a series of tasks that are simple to carry out but challenging to articulate in the proper manner. This book, when read, will reveal to the reader that it has the solutions to the problems that are most readily apparent to him. It is advocated that the problem be solved by endowing computers with the capacity to learn from experience and comprehend the world in accordance with a hierarchy of ideas, in which each thought is defined by its link to more basic concepts.

This strategy does away with the necessity for human operators to precisely specify any information that the computer requires, instead supplying them with the essential information based on their prior experiences. Because of this, there is no longer a need for human operators. Using the concept of a hierarchy of ideas, the computer is able to comprehend more complicated concepts by first developing from more fundamental concepts. This makes it possible for the computer to acquire new concepts more quickly. If we were to develop a graph that explains how each of these concepts builds on the previous one, it would be pretty complicated and have several levels.

Because of this, the strategy for developing artificial intelligence has been given the moniker "deep learning." Researchers did not need computers to acquire a significant amount of information about the outside world in order to accomplish their aims while they were working on the early triumphs of artificial intelligence. These successes took place in relatively sterile and formal environments. For instance, in 1997, IBM's Deep Blue business was victorious against Garry Kasparov, who was the reigning world chess champion at the time. IBM is responsible for the creation of Deep Blue. (Hsu, 2002). Chess is played on a board that has just sixty-four squares, and each of the game's 32 pieces may only move in one of the many distinct ways that are defined.

The game is played with two people. It should come as no surprise that chess is not a particularly difficult game. The development of a chess strategy that is successful is a remarkable accomplishment; yet, the challenge does not lie in the fact that it is difficult for a computer to recognise chess pieces and lawful movements. The game of chess may be reduced to a very small set of rules that are strictly formal, and these rules can be readily preset by the programmer. This gives you the ability to play the game on the computer instead. In a twist of irony, one of the mental challenges that is the most challenging for a human being to conquer, abstract and formal activities, is one of the mental hurdles that is the easiest for a computer to overcome.

But it wasn't until very recently that computers were able to mimic at least part of the abilities of the typical human to recognise words or objects. Since a long time ago, computers have had the ability to defeat even the most skilled human chess players. However, these features were not present in computers until quite recently. In order to successfully do the activities that are required of them each day, a person has to have a comprehensive understanding of the wider world. It is difficult to express in a more formal manner a significant portion of this information due to the fact that it is very intuitive and subjective in nature.

It is necessary for computers to have the same capacity for storing information as people have if they are to perform intelligently. The act of putting unstructured data into a computer is one of the most difficult obstacles that AI researchers must overcome. A number of research have been conducted in an effort to investigate the feasibility of encoding the knowledge that artificial intelligence systems acquire about the world in official languages. By applying principles of logical reasoning to the data, a computer is able to automatically reason about phrases that are supplied in these formal languages. An method to artificial intelligence that has seen rising levels of adoption over the last several years is known as the knowledge base approach.

To this far, none of these efforts have led to the achievement of a significant milestone. One of the most well-known instances of such an effort coming from the general public is the creation of Cyc. (Lenat and Guha, 1989). A claims database and an inference engine are both included in the Cyc system as individual components. CycL is the name of the programming language that was used to construct the approval database. These observations were made by a team of human auditors who are now engaged in evaluating the data behind the scenes. The procedure is laborious and takes a considerable amount of time.

The issue that mankind has is coming up with formal rules that are sufficiently complicated to offer an accurate account of the cosmos. To give you just one example, Cyc was unable of comprehending the narrative of a guy called Fred who shaved first thing in the morning. Fred's belief that the inference engine "FredWhileShaving" entity comprised electrical components was based on the fact that he owned an electric shaver. In spite of the fact that he was aware people do not include any electrical components, he arrived to the same conclusion. This error in the story logic was discovered by the inference engine, which said that the tale told a contradictory story. As a consequence of this, the question arose as to whether or not Fred could still be regarded as a person despite the fact that he was in the midst of shaving. The difficulties encountered by systems that depend on encoded information demonstrate that AI systems need to be able to gain their knowledge by identifying patterns in raw input.

This is important due to issues that have arisen with systems that depend on information that has been encrypted. This must be done if AI systems are ever going to realise their full potential. The term "machine learning" is often used to refer to this capacity since it better accurately describes its technological nature. The advancement of this science made it possible for computers to solve issues that needed knowledge of the actual world and make choices that seemed to be subjective. This was made possible by the development of machine learning. A simple method of machine learning known as logistic regression may be used to establish whether or not a caesarean section is to be advised.

A straightforward approach to machine learning known as Naive Bayes can differentiate between authentic email and spam. The data format that is presented to these rudimentary machine learning algorithms has a significant impact on how well the algorithms carry out the tasks that they are given. When utilising logistic regression, for instance, the artificial intelligence system does not really examine the patient before making a decision on whether or not to perform a caesarean section. Instead, the physician supplies the system with a variety of fundamental facts, such as whether or not the patient's body bears any signs of a womb scar.

After gathering all of this data, the machine will proceed to establish a diagnostic. The term "feature" refers to any information inside the patient picture that is distinctive from the others. This may include information about you, such as your name, gender, age, ethnicity, and so on. It is feasible to identify how each of these patient attributes connects to a set of outcomes by using a statistical method known as patient logistic regression. This will allow for the proper interpretation of the data. However, it is impossible for it to influence in any manner how features are reported, and as a result, it is impossible for it to influence how features are reported. If the patient's MRI had been included in the logistic regression analysis instead of the physician's coded report, then the results of the analysis would not have been correct about the patient.

The individual pixels on an MRI have a very tenuous relationship to the problems that may arise throughout the process of labour and delivery. This dependence on representations is a common occurrence that may be observed in a range of different computer domains as well as in life in general. Operations in computer science, such as searching a collection of data, have the potential to move much quicker if the collection is organised and indexed in a way that demonstrates intelligent thinking.

Performing mathematical operations using Arabic numerals is often thought of as being very straightforward, but performing the same mathematical operations using Roman numerals may be a time-consuming endeavour. It shouldn't come as much of a surprise to realise that the representation that is used has a significant influence on the outcomes that are generated by the algorithms that are used in machine learning. Instead, it is reasonable to suppose that this is the situation.

Check out this fast and straightforward graphic example. Many issues that need AI may be handled by first determining the suitable collection of features to extract from the current state and then feeding those features into a simple machine learning algorithm. This can be done by finding the relevant set of features to extract from the current state. An estimate of the size of the speaker's vocal tract is one example of a feature that may be helpful in identifying a speaker based purely on pitch. This feature is an example of a feature that can be beneficial.

This is only an illustration, however. Because of this, it is easy to ascertain with absolute certainty whether the person speaking is a man, a woman, or a kid. On the other side, when there are so many different kinds of occupations, it might be challenging to decide which features should be included. Think about a goal for the construction of software that, when given with a frozen framework, can determine whether or not a tool already exists. Since we are aware that rims are a distinguishing feature of cars, we can leverage the fact that our theme also has wheels as a way to differentiate it from other themes that also include vehicles.

In terms of the individual pixel values, it is unfortunately impossible to offer an accurate description of what a wheel looks like as a whole. A wheel's portrayal may be made more complicated in a variety of ways, such as shadows falling on the wheel, the sun reflecting off the metal components of the wheel, the vehicle's fender, or any other component. Despite the fact that a wheel has a straightforward geometric shape, this does not preclude it from having an intricate appearance. wheel; the close-up obscures a portion of the wheel; etc. Machine learning is one of the strategies that may be used to tackle this issue.

Specifically, it can be used to determine not just the mapping of the representation to the output, but also the representation itself. This is one way that machine learning can be used to solve this problem. This would be an illustration of one possible use of machine learning. The strategy in issue is known as representational learning, and its name pretty well sums up all you need to know about it. When compared to the performance that can be achieved with representations that have been manually constructed, the performance that can be achieved with representations that have been learnt is often much greater.

Additionally, it enables AI systems to rapidly adapt to new jobs with relatively minimal intervention from human operators. The ever-increasing sophistication of these technologies makes the achievement of this goal more feasible. Within a few short minutes, the representation learning method is able to locate a solid collection of features that are relevant to a straightforward job. On the other hand, the time it takes for the algorithm to uncover a sufficient collection of characteristics for a challenging job might range anywhere from a few hours to many months. It takes a significant amount of time and effort to go through the process of manually generating new features for the hard work.

It is possible that a full community of academics will need many decades to finish the procedure. The autoencoder serves as the most illustrative illustration possible of how a representational learning algorithm really works when it is put into practise. An autoencoder is the product of the combination of two functions: an encoder function and a decoder function. The encoder function transforms the input data into a different representation, and the decoder function transforms the new representation back into the format that was first utilised.

These two processes, when combined, result in a new representation of the data that was first entered. Autoencoders are taught to retain as much information as they can while it travels through an input encoder and then through the decoder, but they also learn to give the new representation a set of attributes that are wanted. This training takes place before the autoencoder is used for the first time.

Because of this, autoencoders are able to be used in a wide number of contexts. As a result of this, autoencoders are now capable of being used in applications that include machine learning. The many distinct kinds of autoencoders each make a contribution, although in their own particular manner, to the accomplishment of a broad range of desired features. When we generate new features or new learning algorithms for features, our primary objective is often to discover the sources of variation that are responsible for explaining the data that we see. This is the case whether we are developing new features or new learning algorithms for features.

This is true regardless of whether we are developing new techniques for learning functions or new functions themselves. The term "factors" only refers to certain domains in this context; the multipliers themselves are not often coupled via multiplication. These characteristics are not always capable of being measured and are not always readily apparent to an observer.

These entities may be seen of as existing in the physical world either as unknown items or as unseen forces that influence visible values. Either way, they have the potential to be influential. Despite this, the possibility of their existence in the physical world cannot be ruled out. They could also exist as creations of the human mind, providing insightful interpretations or speculative justifications for things that are already known to be true.

There is also the chance that they have always been there, right from the beginning of time. These hypotheses and hypotheses about probable causes may be of use in simplifying an otherwise complicated group of situations. One way to think about them is as notions or abstractions that let us grasp a broad range of facts. Taking this approach is one way to address the problem. It's a perspective on how to think about things. When doing an analysis of previously recorded speech, some of the characteristics that may be taken into consideration as potential causes of variability include the age of the speaker, the speaker's gender, the speaker's accent, and the uttered words.

When examining a photograph of a vehicle, some of the elements that contribute to the variance include the location of the automobile inside the frame, the colour of the car, the direction that the sun is facing, and the amount of sunshine that is there. In many applications of AI in the real world, one of the primary obstacles is the fact that many different variables of variation impact every piece of data that may be visualised.

This is a significant contributor to the overall level of volatility. When captured at night from a moving vehicle, the individual red pixels in a photograph might look incredibly dark, practically black. Something that has a chance of occurring. The outlines of the automobile silhouette will change depending on the angle from which you look at the car; this is because the car may be seen in three dimensions. In the great majority of our programs, we need to disentangle a number of the components that lead to variation and get rid of those that are not pertinent to the objectives we are attempting to accomplish.

Obviously, there will be instances when the process of extracting these high-level and abstract properties from raw data would be quite challenging. With a level of data comprehension that is near to that of humans, it is feasible to discern many of these particular variational qualities, such as a speaker's accent. At first look, it does not seem to us that learning with pictures is going to be particularly successful. This is despite the fact that acquiring an image is almost as hard as conquering the first obstacle. This basic challenge of learning representations may be addressed and conquered via the use of deep learning.

This is accomplished by include the representations that are explained in terms of other representations that are more understandable. The term "deep learning" refers to the process wherein a computer may develop more complex ideas from simpler ones by utilising information that it has already learnt. Figure 1.2 illustrates how a system that employs deep learning may communicate the idea of a picture of a person by merging more fundamental ideas such as vertices and edges, both of which are represented as edges. This is shown inside the framework of an image of a person. This argument is made very obvious by the illustration.

A great number of people believe that the feedforward deep web, which is also referred to as the multilayer perceptron, is the most prototypical illustration of the deep learning paradigm. (MLP). A multilayer perceptron is nothing more than a mathematical function that maps one set of input values to another set of output values. This function transfers one set of input values to another set of output values. This function converts one set of input values into a separate set of output values based on the input values that are provided. The actual function is the product of many additional, smaller functions being combined into a single one.

Because the application of a distinct mathematical function always results in a new representation of the input, we may conceive of it as a new type of representation. Deep learning may be approached from a number of different directions, one of which being the concept of "learning to accurately represent data." Additionally, deep learning may be used in a wide number of settings. It is also possible to think of deep learning as the process of teaching a computer to learn complicated computer programmes by breaking them down into many smaller stages.

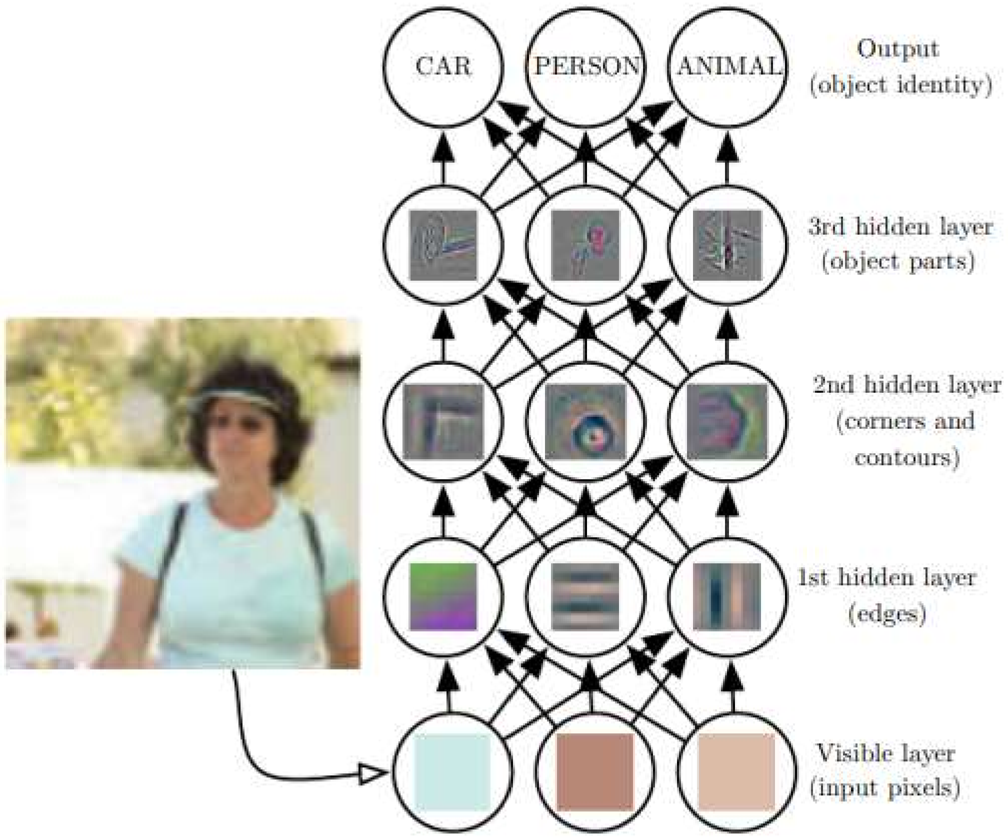


Figure 1.1: Representation of a deep learning model.

You may be able to see the state that the computer's memory is in after a distinct set of instructions has been carried out in parallel at each subsequent level of representation. To put it another way, you may be wondering what condition the memory is in after the computer has done processing the information that it contains. The greater the depth of a network, the greater the number of orders that may be carried out in quick succession by that network. It is difficult to overstate how helpful it is to provide the instructions in the appropriate sequence.

Statements are able to make references to the outcomes of other, previously executed statements. According to this understanding of how deep learning works, not all of the information that is involved in the activations of a level has to contain variable variables that define the input. Because the notation stores information about the status of the system, it is simple to execute a programme that gives the input meaningful context. One may think of this state information as being analogous to a counter or pointer in a more conventional kind of computer programme.

It has nothing anything to do with the information that was supplied as part of the input, but it does assist the model in organising the rendering that it carries out. There are primarily two approaches that may be used for determining the depth of a pattern. The problem is analysed from a number of different perspectives, the first of which is the amount of sequential instructions that need to be carried out in order to assess the design. We can think of this as the length of the longest route along a flowchart that explains how the outputs of the model are computed from each of its inputs. We can think of this as the length of the longest path along a flowchart. Taking these steps will take more time.

The same function may be shown as a flowchart at various depths depending on the functions we use as steps for persons in the organisational chart. In the same way that two equivalent computer programmes can have different lengths depending on the language in which the programme is written, the same function can be represented as a flowchart at different depths. The same function may be represented as a flowchart of varying depths, much as the length of two identical computer programmes might vary depending on the programming language that was used to write the programmes. Because of the choice of language, a single picture might have two distinct measurements, as seen in the image.

This argument is shown in figure 1.1. One further way that is used by deep probability models is the presumption that the depth of a model is the depth of the diagram. This depth refers to the model itself, and not the depth of the diagram that describes how concepts are connected to one another. In this specific situation, the level of detail included within the flowchart that discusses the calculations necessary to compute the representation of each concept may be far more than the level of detail contained within the flowchart that discusses the ideas themselves. This is due to the fact that the system's comprehension of the basic ideas may be enhanced by supplying it with knowledge on the more sophisticated ideas.

For instance, if an artificial intelligence system is examining a picture of a face, but one eye is obscured, it will initially only be able to perceive the other eye. Following the realisation that there is a face sample, he may very certainly draw the conclusion that there is also a second eye sample there. However, the concept graph only has two levels: one for eyes and one for faces. If we go through our estimate about each concept another n times, the computational graph would have 2n levels, but the concept graph will only have two levels. The conceptual network consists of only two levels when applied to this specific case.

Because it is not always apparent which of these two viewpoints (computational graph depth or probabilistic modelling graph depth) is more significant, and because various individuals use different tiny collections of things to create their graphs, there is no one accurate number. There is no one proper value for the length of a computer program, and similarly, there is no one perfect number for the depth of an architecture. This issue is comparable to the one in which it is not always obvious which of these two points of view, computational graph depth or probabilistic depth, is also not in agreement on what depth a model should at least label "deep."

Deep learning, on the other hand, may be understood more accurately as the study of models that comprise a mixture of learned functions or learned concepts at a deeper level than standard machine learning. Deep learning is, in point of fact, an expansion of more conventional forms of machine learning. In a nutshell, artificial intelligence (AI) is a method that is centred on deep learning, and it is the topic of discussion in this book. To be more exact, it is a kind of machine learning, which is a method that enables computer systems to become more effective when more data and experience are added to them.

The authors of this book suggest that the use of machine learning is the only realistic method that can be used to construct artificial intelligence systems that can function well under the challenging circumstances that are representative of the real world. Deep learning is a specialised kind of machine learning that achieves more power and adaptability via the process of mastering the representation of the world as a hierarchical network of interconnected concepts. This indicates that each word is described using simpler terms, and that more complex representations are computed using representations that are more straightforward. Figure 1.1 illustrates the connection that exists between all of these many facets of AI. Figure 1.1 presents an overarching schematic that illustrates how each component functions as a whole.

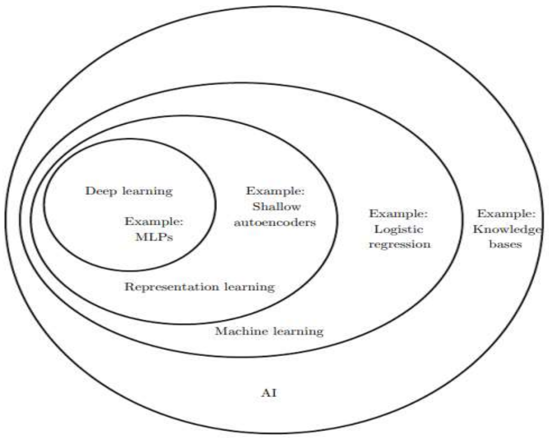


Figure 1.2: A Venn diagram showing how representative deep learning is.

Although it will be of value to a large number of people, the writers of this book primarily had two groups of readers in mind when they wrote it. Among these audiences are college students (either at the undergraduate or graduate level) who are interested in acquiring further knowledge about machine learning. Students who are beginning their careers in deep learning and AI research are included below. The second set of people who would benefit from taking this course are software engineers who do not have a background in machine learning or statistics but who want to acquire these abilities as rapidly as possible in order to begin integrating deep learning on their product or platform. The advantages of engaging in in-depth study.

There are many different types of software domains, including computer vision, voice and sound processing, natural language processing, robotics, biology and chemistry, video games, search engines, Internet advertising, and finance. This book has been broken up into three sections in order to appeal to the greatest number of readers. The first part of this chapter is an introduction to the fundamental mathematical methods and ideas, as well as the terminology used in machine learning. In Part II, we will talk about the deep learning algorithms that are the most well-known and commonly utilised. These algorithms are virtually solved technologies.

In the third and last half of this paper, we will talk about a few ideas that are more theoretical in nature but are nonetheless considered essential to the development of deep learning research in general. The reader shouldn't be afraid to skim over any portions that don't apply to him because of the things he already knows or the things that interest him. For instance, readers who are already acquainted with linear algebra, probability, and other fundamental notions related to machine learning may skip Part I, while readers who just wish to create a system that is functional do not have to do so.

A flowchart that depicts the general structure of the book and that may be used as a reference to decide which chapters to read is provided in Figure 1.6. You can access this guide by clicking on the figure. We make the assumption that all of our readers are proficient with computers. We are going to proceed on the assumption that you are proficient in programming, that you have a fundamental knowledge of computational problems, complexity theory, basic mathematics, and a vocabulary of graph theory.

1.4 HISTORICAL TRENDS IN DEEP LEARNING

When placed in its historical context, deep learning is much simpler to grasp. Instead of offering a comprehensive history of deep learning, let's focus on identifying a few important developments.

- Deep learning has a long and eventful history, yet during its existence, it has been known by a variety of titles that each express a unique philosophical viewpoint, and its popularity has waxed and waned.
- Deep learning has grown increasingly beneficial as the quantity of data for training purposes that is now accessible has risen.
- As the computational infrastructure (including hardware and software) for deep learning has matured over time, deep learning models have acquired traction, which has led to their increased popularity.
- Over the course of time, deep learning has been able to handle problems that are more difficult while also improving their accuracy.

1.5 MANY NAMES AND CHANGING HISTORIES OF NEURON NETWORKS

Deep learning is an exciting new technology, and we expect that a considerable proportion of people who read this book will have that experience. They will also realise that the "story" is included in a book that is devoted to an area of study that is still developing. Deep learning was initially invented in the 1940s, but since it wasn't extremely popular for many years before it became popular today, deep learning gives off the idea that it is "new." This is because deep learning wasn't very popular for many years before it became popular today. In addition, deep learning has been known by a variety of names throughout history, and it wasn't until very recently that it was given the moniker "deep learning."

Multiple revisions have been made to the domain name in order to accurately represent the contributions of a large number of scholars and the diversity of their findings. A comprehensive account of the development of deep learning cannot be provided within the confines of this guide due to its expansive nature. Having some prior knowledge, on the other hand, is helpful when attempting to comprehend what deep learning is. The history of deep learning may be broken down into three distinct periods: the time from the 1940s to the 1960s, which was known as cybernetics; the period from the 1980s to the 1990s, which was known as connectionism; and the current renaissance, which began in 2006 and is known as deep learning.

The graphic presents this information in numerical form. Some of the early learning algorithms that are still in use today were first created as computer models of biological learning. This indicates that the games are models of how learning occurs or has the capacity to occur inside the brain. The term "artificial neural networks" refers to one of the titles that have been given as a direct outcome of deep learning. (ANN). The opinion is that these models are capable of deep learning, which is consistent with the concept that deep learning models are constructed systems inspired by biological brains. (regardless of whether the brain in question is that of a human or any other species).

Although the sort of neural networks that are used for machine learning have on occasion been used in an effort to get a better understanding of how the brain operates (Hinton & Shallice, 1991), these networks are not typically meant to be exact representations of the way in which biological systems operate. The neuronal perspective of deep learning draws its motivation mostly from two different underlying principles. It has been hypothesised that the human brain provides instances to explain how intelligent behaviour is both possible and feasible. In addition, it has been hypothesised that deciphering the fundamental computational principles of the brain and modelling its operation might be a straightforward and theoretically easy method for the development of intelligence.

Even if they are unable to address practical applications, machine learning models that provide light on these basic scientific challenges are still relevant. One point of view is that it would be very interesting to have an understanding of the human brain as well as the fundamentals that underlie human intellect. Today, the meaning of the phrase "deep learning" extends more than the neuroscientific comprehension of the most recent generation of machine learning models. To do this, it makes use of the more generic concept of learning many composition layers, which can be implemented in machine learning frameworks that don't need the use of nervously inspired algorithms. The study of neuroscience provides us with grounds for optimism about the potential for a single deep learning algorithm to handle a variety of issues. Researchers in the field of neuroscience have shown that a section of a ferret's brain responsible for visual processing may be altered to transport signals responsible for auditory processing; this enables the animal to utilise its processing system to learn to "see." (from Melchner et al., 2000).

This leads to the belief that a considerable chunk of the mammalian brain is capable of using a single algorithm to handle the majority of the various issues that the brain encounters. Before the discovery of this notion, the field of study into machine learning was much more splintered, with numerous groups of researchers focused on different elements of the topic: B. Speech recognition, visual processing, natural language processing, and motion planning. To this day, these app communities are distinct from one another. On the other hand, it is very uncommon for deep learning research organisations to investigate the majority of these application fields at the same time, if not all of them. From the study of neurobiology, we are able to glean several useful generalisations. The structure of the human brain served as the inspiration for the fundamental idea of having a vast number of processing units that can only become intelligent via their interaction with one another.

Fukushima's Neocognitron, which was first made available to the public in 1980, included an extremely sophisticated model architecture for image processing. The structure of the mammalian visual system served as an inspiration for this design, which in turn served as the foundation for the present convolutional network, which was subsequently suggested by LeCun et al. As was stated in episode 9.10, it has now been released. The rectified linear unit is the model neuron that is used to build the vast majority of contemporary neural networks. The original Cognitron, which was created by Fukushima in 1975, was a more complicated model that was greatly inspired by studies on brain activity. While Jarrett et al. (2009) highlight inputs that are more engineering-oriented, other researchers, such as Nair and Hinton (2010) and Glorot et al. (2011a), cite neurology as an impact.

The present version's superior aerodynamics were accomplished by fusing together ideas gleaned from a wide variety of sources. Even while neuroscience is a valuable source of ideas, no one should feel like they have to stick to it to the letter just because it's popular. There has not been a correlation established between increased neuronal realism and improved performance in machine learning. This is the case despite the fact that we are aware that actual neurons compute functions in a manner that is extremely distinct from that of real linear units that have been rectified. Even while neuroscience has been effective in inspiring a wide variety of neural network topologies, there are still many aspects of biological learning that we do not understand well enough for neuroscience to influence the learning algorithms that we use to train these constructs to emulate. that many neural network topologies are inspired by neuroscientific research.

The media often places an emphasis on the parallels that may be seen between deep learning and the brain. Researchers who work in the field of deep learning are more likely to believe that the human brain has some kind of influence on machine learning than researchers who specialise in other areas of machine learning, such as core machines or Bayesian statistics. Deep learning, on the other hand, should not be interpreted as an effort to recreate the human brain. Deep learning in its current form necessitates the incorporation of concepts derived from a diverse range of academic fields, in particular fundamental subfields of applied mathematics, such as linear algebra, probability theory, information theory, and numerical optimization, amongst others. Some researchers in deep learning look to neuroscience as the primary source of inspiration for their work, whereas other researchers in deep learning are not interested in neuroscience at all.

It is essential to emphasise the fact that ongoing research is being conducted to learn more about how algorithmic processes operate in the brain. These initiatives are showing a lot of life even now. Deep learning is its own distinct field of study that should not be confused with the work being done in this area, which is more popularly known as computational neuroscience. It is not unusual for scholars to shift their focus back and forth between the two fields during the course of their careers. Deep learning is mainly focused on the creation of computer systems that are capable of carrying out intelligent tasks in an effective manner, while computational neuroscience is primarily focused on the creation of models that are more accurate representations of how the brain really functions.

These two subfields are included under the umbrella of the field of artificial intelligence. Connectionism, also known as distributed parallel processing, is a school of thinking that was primarily responsible for the creation of the second wave of neural network research in the 1980s. This study sought to understand how the brain forms connections between neurons. (Rumelhart et al., 1986c; McClelland et al., 1995). The connectivist idea has been gaining traction as a viable option in the study of cognitive science. An example of an interdisciplinary approach that draws from a wide range of fields of study is the study of the mind from the perspective of cognitive science. This line of inquiry draws from a variety of fields of inquiry.

In the early 1980s, the majority of cognitive scientists concentrated the majority of their study on symbolic thinking processes. In spite of their extensive applicability, symbolic models are notoriously difficult to explain in terms of how neurons in the brain actually carry out the application of models. Connectivists have recently started looking at other theories of cognition, some of which may truly be founded on neurological applications. In doing so, they appropriated several ideas that can be traced back to the research conducted by the psychologist Donald Hebb in the 1940s.

Connectivity is a method of computing that postulates intelligent behaviour may be produced by interconnecting a large number of simple processing units in a system. This theory is known as the "connectedness hypothesis." This discovery is accurate for hidden units in computer models as well as neurons in genuine nervous systems. The connectivist movement of the 1980s gave birth to a number of significant ideas that are still at the core of deep learning today.

These ideas include the following: One such concept that may be taken into consideration is that of distributed representation. (Hinton et al., 1986). According to this theory, every conceivable input to a system is represented by a big number of features, and each feature plays a role in representing a large number of potential inputs. Additionally, this theory claims that each feature contributes to the representation of a large number of possible inputs. Imagine the following scenario: we have built a vision system that is able to differentiate between several types of vehicles, such as automobiles, trucks, and birds; these objects may all be coloured red, green, or blue. B. "red truck, red car, red bird, green truck, etc." is one technique of displaying these entries.

This serves as an illustration of how something like this may operate. In order to do this, nine distinct neurons are required, and each neuron has to comprehend the idea of colour as well as the identification of an element on its own. Utilizing a distributed representation, in which colour is defined by three neurons and object identification is defined by three neurons, is one method for making this issue easier to solve. One further technique to utilise a distributed representation is as described above. It just requires six neurons as opposed to the usual nine, and the neuron that determines redness may learn about it through pictures of other things, such as vehicles, trucks, and birds, rather than only pictures of one particular thing. This book focuses primarily on the concept of distributed representation, which is investigated in more depth throughout the text of the book.

Two more key accomplishments that may be credited to the connectionist movement are the successful use of backpropagation to train deep neural networks with internal representations and the widespread acceptance of the backpropagation method. (Rumelhart et al., 1986a; LeCun, 1987). Although its notoriety has waxed and waned over the course of the years, the aforementioned algorithm is, as of the time of this writing, the deep learning training approach that enjoys the widest level of adoption. During the 1990s, researchers made significant progress towards their ultimate aim of precisely recreating sequences by using neural networks. Bengio et al. (1994) outlined some of the most significant mathematical obstacles that arise when modelling big arrays. The long-short term memory network, sometimes referred to as the LSTM network, was designed by Hochreiter and Schmidhuber (1997) in order to overcome some of the issues that were presented. LSTM is being used for a wide range of sequence modelling applications, including a significant number of natural language processing jobs at Google.

It is also used extensively in a variety of other sectors. The second wave of research into neural networks continued until the middle of the 1990s: Companies based on neural networks and other kinds of artificial intelligence technologies started showing an excessive amount of optimism while they were in the process of looking for investors. When artificial intelligence research could not live up to the excessively optimistic expectations of investors, they felt let down. During the same time span, advancements were achieved in machine learning's practise in other fields. Both graphic models (Jordan, 1998) and core machines were successful in accomplishing a number of significant goals. As a result of these two issues, the use of neural networks went through a period of falling popularity that lasted until the year 2007.

During this period, neural networks continued to demonstrate exceptional performance across a broad spectrum of activities. Through its work in Neural Computing and Adaptive Perception, the Canadian Institute for Advanced Research (CIFAR) has been an essential contributor to the advancement of the field of neural network research, which has been a significant area of focus for the organisation. (NCAP). Through the efforts of this program, researchers in the fields of machine learning Yoshua Bengio of the University of Montreal, Geoffrey Hinton of the University of Toronto, and Yann LeCun of New York University were able to collaborate. As a result of the programme's emphasis on interdisciplinary collaboration, participants in the CIFAR NCAP research programme have included not just neuroscientists but also specialists in human and machine vision, as well as academics from a variety of other areas.

At the time, most people had the impression that it was exceedingly difficult to train deep neural networks. Although we now know that algorithms that have been there since the 1980s perform quite well, this fact did not become clear until about the year 2006. In 2006, a significant advance marked the beginning of the third wave of neural network research. There is a method known as greedy layered pre-training, which may be used to effectively train the belief web. More information on this method can be found here. Other research groups affiliated with CIFAR swiftly proved that the same method could be used to produce a large number of different species. He has been of tremendous assistance in the testing of generalisation and the building of many instances of deep neural networks (Bengio et al., 2007; Ranzato et al., 2007a). The phrase "deep learning" has been widely used as a result of the recent surge of interest in neural network research.

This is done to highlight the fact that researchers are now able to train neural networks that are deeper than in the past, and to bring attention to the theoretical significance of depth. During that time period, deep neural networks beat alternative artificial intelligence (AI) systems that were based on other machine learning methods and functions that were hand-designed. At the time that this article was written, the third wave of popularity for neural networks was still going strong, although throughout this wave, the primary emphasis of research on deep learning had shifted significantly. The third wave began with an emphasis on newly developed unsupervised learning methods and the capacity of deep models to generalise effectively from small datasets. Today, however, there is a greater interest in more sophisticated algorithms, much more established forms of supervised learning, and the capacity of deep models to make use of big datasets. labelled datasets.

1.6 INCREASE LOG SIZE

Despite the fact that the earliest studies with artificial neural networks were conducted in the 1950s, it wasn't until relatively recently that deep learning was acknowledged as a significant step forward in technological development. However, up until recently, many people considered that deep learning was more of an art than a technology, something that could only be applied by a seasoned expert. Deep learning has been employed effectively in commercial applications since the 1990s. It is a well-known truth that a certain degree of knowledge is required in order to acquire the outcomes that are wanted from a system that utilises deep learning. The good news is that the degree of skill needed will rise in tandem with the quantity of training data as it grows.

While the models that we train with these algorithms have experienced advances that make it simpler to train incredibly deep architectures, the learning algorithms that are now approaching human performance on difficult tasks are nearly identical to the learning algorithms that were used to solve toy problems in the 1980s. This is despite the fact that training these models has become easier as a result of these changes. Learning algorithms of today are capable of matching human performance on difficult tasks. The most important development that has taken place in recent times is that we are now able to provide these algorithms with the necessary hardware and support for them to be successful.

The total amount of the underlying datasets has become noticeably larger throughout the course of time. The most important factor in the progression of this phenomenon is the widespread use of digital technology in contemporary culture. Since more and more of what we do in modern times takes place on computers, a greater proportion of this activity is being tracked. Because our computers are becoming more and more networked with one another, it is becoming simpler to centralise this information and organise it into a dataset that is appropriate for applications that use machine learning. The advent of "Big Data" has made machine learning much simpler as a result of the substantially decreased cost of statistical prediction. Statistical prediction requires looking at relatively small quantities of data and then effectively generalising those findings to fresh data.

Because of this, machine learning becomes far more precise. As of the year 2016, a supervised deep learning algorithm will normally perform pretty well with about 5,000 category-labeled instances, and when trained on a dataset with at least 10 million labelled examples, it will meet or surpass human performance. Since 2016, this has been a general rule of thumb that may be seen spreading on the internet. An important field of research that focuses on how we can make the most of big unlabeled sample sets that may be acquired either unsupervised or unsupervised, working successfully with smaller datasets is an area of study that is becoming more essential. semi-supervised learning.

1.7 INCREASED MODEL DIMENSIONS

It is one of the main reasons neural networks have exploded in popularity today, despite their modest success since the 1980s: the idea that animals can be intelligent and that their neurons interact with each other in large numbers is one of the main ideas that connectivity offers. Today, we have the computational capacity to run much larger models than we did in the 1980s. A solitary neuron or a very small collection of neurons does not make a significant contribution to the operation of the brain as a whole. Biological neurons are often not connected to one another very intimately. Our machine learning models have, for decades, even had the amount of connections per neuron in the order of mammalian brains.

And this is in spite of the fact that the models we use are just a few years old at most. As can be seen in Figure 1.2, the scale of neural networks, as evaluated by the total number of neurons, was fairly low up until quite recently. Since the development of hidden drivers, the size of neural networks has generally doubled every 2.4 years. This trend began when hidden drivers were first introduced. The availability of faster computers with higher storage capabilities as well as larger data sets is helping to enable this increase. When doing activities that demand a higher level of complexity, vast networks have the potential to improve their accuracy.

It would seem that this trend will go on for at least a few more decades. Artificial neural networks will not have the same number of neurons as the human brain until at least the year 2050. This presupposes that advances in technology will enable quicker scaling in the future. The scale of biological neural networks may be substantially bigger than what is seen in this graphic. This is due to the fact that actual neurons are capable of reflecting more complicated tasks than the artificial neurons that we have available today. When seen in hindsight, it should not come as a surprise that neural networks with fewer neurons than a leech are unable to handle difficult AI challenges.

Even today's networks, which we consider to be pretty huge in terms of computer systems, are even smaller than the neural systems of relatively basic vertebrates like frogs. This is the case even if we perceive today's networks to be rather large. This is true in spite of the fact that today's networks are becoming more complicated. The availability of increasingly powerful processors throughout time may be ascribed to the gradual increase in the size of models.

1.8 IMPROVE ACCURACY, COMPLEXITY, AND REAL-WORLD IMPACT

Since the 1980s, there has been a steady development in the capacity of deep learning to deliver accurate identification or prediction. Additionally, practical applications of deep learning are being made in an increasing number of domains, which speaks well for the field's future. The difficulty of recognising individual items in images that were very cropped and tiny was one of the first applications of early kinds of deep learning. (Rumelhart et al., 1986a). Since then, the total amount of pictures that neural networks are capable of analysing has been gradually but consistently growing. The modern object identification networks are able to analyse photos with a high level of detail and a high resolution, and they do not need the photograph to be cropped in the vicinity of the item that is to be recognised.

In a similar manner, early networks were only able to discriminate between two distinct categories of items (or, under some circumstances, the absence or presence of just one category of object), but modern networks are often capable of differentiating between at least a thousand such categories of things. The ImageNet Large-Scale Visual identification Challenge, often known as the ILSVRC, is the most prestigious object identification competition. It takes place eleven times each year. When a convolutional network triumphed against this opponent for the very first time, it was a significant step forward in the meteoric growth of deep learning. This victory marked an important milestone in the field.

The mistake rate of the prior art top 5 was decreased from 26.1% to 15.3% thanks to this performance, which involves the neural network providing a ranking of alternative categories for each picture and the proper category being generated omitting It means. 15.3% of the assessments were ranked within the top five spots on this ranking. Since then, deep convolutional networks have emerged victorious in each of these events. At the time that this article was written, developments in deep learning had supposedly brought the error rate of the top five participants in this tournament down to 3.6%. Another domain that has been considerably altered as a result of the use of deep learning is speech recognition.

After making steady progress towards reducing mistake rates in voice recognition during the 1990s, it halted that improvement around the year 2000. 2011; Hinton et al., 2012a) resulted in a significant drop in mistake rates, with certain error rates falling by as much as fifty percent. This narrative will be discussed in further depth in the following paragraphs. Deep webs also displayed superhuman performance in classifying traffic signs and pedestrian photos, achieving remarkable results in both areas. Deep webs successfully detected and segmented photographs of pedestrians. (Ciresan). Deep neural networks have been growing in size and accuracy over the last several years, which has coincided with a rise in the level of complexity of the issues that they are able to answer.

Goodfellow et al. (2014d) demonstrated that neural networks can be taught to construct a whole string of letters copied from an image, as opposed to only seeing a single item.

According to Gulcehre and Bengio (2013), prior to this discovery, it was thought that this particular kind of learning required the labelling of the sequence's component parts. However, new study has shown that this viewpoint is incorrect. Iterative neural networks are increasingly being utilised to explain interactions between sequences and other sequences, rather than just modelling fixed inputs as they have been in the past. The LSTM array model that was presented before is an example of this kind of model. It would seem that this sequential learning would revolutionise yet another application, which is machine translation. (Sutskever et al., 2014; Bahdanau et al., 2015).

This trend of rising complexity has been slowly interrupted with the creation of Turing neural machines (Graves et al., 2014a), which learn to read any information from memory cells and write any material to them. These machines are able to learn to read any information from memory cells and write any material to them. These neural networks are able to learn fundamental programming concepts by seeing instances of desirable behaviours and then attempting to imitate those behaviours. For instance, teaching students to sort lists of numbers by giving them examples of jumbled and sorted lists of numbers is an effective way to teach this skill.

This technique of self-programming is currently in its early stages, but in the not-too-distant future, it may be applicable to a wide variety of settings and applications. The field of artificial intelligence has scored another win with the use of deep learning in the area of reinforcement learning. In the context of reinforcement learning, it is necessary for an autonomous agent to learn to do a task via a process of trial and error without any direction from the human operator in order to fulfil the requirements of this kind of learning. DeepMind has shown that a system that uses deep learning and reinforcement learning can learn to play Atari video games and attain human-level performance across a wide variety of challenges. Learning at a deeper level has helped make this achievement feasible. (Mnih et al., 2015).

Deep learning has also helped improve the effectiveness of reinforcement learning in the area of robotics, which has led to a major rise in the field's overall performance. (Finn et al., 2015). Many of these uses of deep learning lead to large financial rewards. Many of the most successful technological businesses in the world today, like Google, Microsoft, Facebook, IBM, Baidu, Apple, Adobe, Netflix, NVIDIA, and NEC, make use of deep learning in their operations. The advancement of deep learning is also heavily dependent on technological improvements made in the software architecture that supports it. Both Theano and TensorFlow are used. (Abadi et al. Additionally, deep learning has been essential in the advancement of other scientific domains.

The creation of convolutional networks for object recognition has provided neuroscientists with a visual processing model that may be investigated. (DiCarlo, 2013 Additionally, Deep Learning offers helpful tools for the processing of enormous volumes of data as well as the development of reliable forecasts in a variety of scientific domains. It has been used effectively in the hunt for subatomic particles (Dahl et al., 2014), to anticipate how chemicals would interact (Dahl et al., 2014), to assist pharmaceutical businesses in automatically interpreting new pharmaceutical compounds made (Dahl et al., 2014), and to find subatomic particles (Dahl et al., 2014). (Dahl et al., 2014). Dahl et al., 2014) as well as photos obtained using a microscope to produce a three-dimensional map of the human brain (Baldi et al., 2014).

(KnowlesBarley et al., 2014) We postulate that in the not too distant future, the area of deep learning will become ever more prominent in the scientific community. In conclusion, "deep learning" has grown into a "machine learning" method over the course of the last several decades. This technique was established and is primarily based on our knowledge of the human brain, statistics, and applied mathematics. In recent years, it has seen significant growth, not just in terms of popularity, but also in terms of its utility. The development of more powerful computers, datasets of a greater size, and techniques of network training that are more in-depth may be credited for the majority of this advancement. In the years to come, deep learning will be confronted with a number of obstacles and presented with a number of chances that will enable it to push new limits and continue to advance.

CHAPTER 2

LINEAR ALGEBRA

In recent years, the quantity of information that has been made accessible on picture analysis via the use of deep learning has substantially increased, and there is no evidence that this expansion will cease any time in the near future. Because of this, there is no one book that can be read to get an understanding of all that is required; rather, there are a great many academic works to study in order to locate ones that are relevant. Because the process is so difficult and time-consuming, a substantial percentage of them investigate the most cutting-edge deep learning approaches that are now on the market. The researchers examined a wide variety of topics; nevertheless, it became clear that many of these questions would not contribute to the larger goals of the study. In the next paragraph, we will discuss the pertinent concerns, and then we will assess those issues to determine the pertinent pros and negatives.

2.1 INTERNATIONAL AERIAL ROBOTICS COMPETITION (IARC)

Ascend NTNU is a student organisation that is not for business and was established in order to provide NTNU students with the opportunity to participate in international air robotics competitions [7]. The year 2015 marked the beginning of operations for the company, and the following year, he took part in the competition for the very first time. The International Airborne Robotics tournament, often known as the IARC, is an ongoing international tournament that was first held in 1994 with the intention of promoting the technology of air robots. This gives the participants the ability to solve issues that were thought to be tough or impossible at the time, and when someone eventually solves one of the questions, a new one that is far more challenging appears. The seventh assignment of the competition is now underway, and contestants are being asked to develop completely autonomous drones that are capable of guiding a group of ground robots in a certain direction.

As can be seen, the competition takes place indoors on a white grid of 20 metres by 20 meters, with one boundary line of the grid coloured green and the other border line coloured red. A white line that is more substantial than the other grid lines can be seen along the two remaining sides of the border. There are 10 ground robots, each with a diameter of 34 centimeters, that were deployed from a single starting point and are now traversing the grid either autonomously or semi-randomly. Robots can move ahead in a straight line, but they rotate their bodies 180 degrees every 20 seconds. In addition, their orbits have up to 20 degrees of angular noise pumped into them at regular intervals of five seconds at a rate of five degrees per second.

When you land on one of the robots, a switch is triggered, and the robot will rotate itself so that it faces in the opposite direction of where you landed. You also have the choice to position yourself in front of it and obstruct its route, which will force it to go in circles. When travelling through the airspace, it is essential to keep a safe distance from the four extra ground robots that have vertical tubes linked to them measuring two metres in length. Within the next 10 minutes, or until all of the ground bots have left the area, it will be necessary to accomplish the aim of the current assignment, which is to guide as many ground bots as possible to the green boundary of the arena. When a ground robot crosses the boundary of the arena, it is automatically disqualified from further participation in the tournament.

2.2 WHY IS ASCEND NTNU RELATED

What kinds of deep learning techniques may be used to the production of time-efficient and adaptable solutions for autonomous flight? In order to do any kind of hands-on testing in this area, an actual test drone is necessary. Even those who have never flown a drone before may easily learn and navigate through one of the many amazing alternatives that are now accessible on the consumer market. The fact that these functions are already pre-installed on the drones makes them very simple to use. The difficulty is that nearly none of them come with open source software, and even while the great majority of them contain a camera that faces forward, there is often no straightforward method to connect additional cameras, sensors, or computing power to them. This is the primary reason why the problem exists. Keeping this in mind, we are going to examine obtaining a fully operational test aircraft via one of the following three methods:

To begin, let's take into consideration the option of putting together a drone using prefabricated parts or a prefabricated kit. The most significant drawback of pursuing this course of action is that it is very challenging to construct the drone without having any previous understanding of the drone. It is quite simple to make errors that are relatively little but expensive (both in terms of time and money), which will need the purchase of new components and will interrupt the research process. The same is true for a solution that involves adding more computer power, sensors, and cameras to an already existing drone. This arrangement ought to be less complicated, however errors committed during this procedure might result in significantly increased expenses.

This solution calls for the drone to additionally have a flight controller that has an open source application programming interface (API), which is something that only a select few ready-made drones have. It is by far the most demanding choice to join the Ascend NTNU team and make this article the emphasis of improving current autonomous systems utilising computer vision and deep learning. However, it is also the option that stands out as being the most desired overall for a multitude of reasons. In the first place, the ways of autonomous flying that depend entirely on visual and sensory information are quite pertinent to the discussion that we are having. Second, the degree of resources, expertise, and support that this Master will acquire from working with Ascend is extremely desired and will significantly minimise the amount of time that is necessary to achieve and maintain a functional test unit.

Perform a wide variety of tests on the system up to the month of May, when Ascend will finally get access to it. Up until this point, we have conducted all of our testing and evaluations utilising the more outdated Jetson TX1. Ascend NTNU has already started working on this topic, and as a consequence, they have a drone that is partially autonomous and entirely functioning. They even have a devoted staff of individuals whose job it is to keep it functioning and always enhance it in some way. The drone has been painstakingly handcrafted from the ground up and boasts a fuselage made of carbon fibre as well as four boom arms that can be detached independently.

Every component of the drone has been assembled in such a manner that even an experienced drone builder would have a difficult time replicating the final product. Because some of the more visible sections surrounding each motor are 3D printed and custom designed, these parts are readily replaceable in the event that they get damaged. The onboard processing capability is provided by a Pixhawk Flying Controller as well as four Nvidia Jetson TX2 cards. These two aspects are at the core of the problem. It is vital to have a specialised carrier board in order to contain the requisite connection interfaces for the separate components on a board that is both small and lightweight.

The flight controller comes complete with its own own Inertial Measurement Unit, often known as an IMU.

This component is in charge of determining the direction in which the drone is travelling in all possible directions. In addition to the four cameras that are pointing forward, backward, left, and right, there is also one fisheye camera that is facing below. There are input devices in use. A Light Detection and Range Variation (LIDAR) system that covers 360 degrees will do a horizontal check of the arena for any hazards. There is also a computer available for heavy processing, but since there is a latency when data is transferred back and forth, onboard processing is much quicker responsive and is often chosen.

There is also a computer accessible for heavy processing. On the other hand, it continuously transfers photographs and data collected by sensors to the touch computer so that the overall performance of the system may be enhanced after the competition. It illustrates the links between the primary components of the computer as well as between the various cameras and sensors. It is essential to keep in mind that Nvidia introduced the Jetson TX2 onboard in March of 2017, which rendered this option inaccessible.

The subject's definition as presented in Ascend NTNU. The partnership with Ascend NTNU has been formally established. Examining the inner workings of Ascension is necessary if we are to have an understanding of how something of this kind operates. The team has been broken up into a number of smaller sections, each of which is tasked with a distinct endeavour.

- Hardware: The primary emphasis of this team is the construction of the drone and the verification that all essential and optional parts have been installed. They guarantee that it is both modular and durable so that it can more readily adapt to various hardware configurations and that it can crash several times without harming any essential components.
- Perception: Make an effort to comprehend all of the sensory information coming from the many drone sources. In particular, this group concentrates on the data collected by five onboard cameras that are attempting to take pictures of the environment around them. In the IARC competition, the area of the grid that constitutes the world to be observed is strictly confined. The planning team

is given the ability to make better judgements as a result of Perception's creation of the most accurate picture of the world's situation.

- Planning - The planning group takes the scenario model that was developed by the perception team and applies it to the problem of determining the most effective route for the drone to travel. Within a time frame of ten minutes, moving as many ground robots as possible to the green side of the grid should be considered the optimal approach.
- Control: Control is responsible for obtaining the planning team's recommended route and ensuring that the drone is able to reach the next destination in a secure manner, avoiding any collisions with tower robots that might block the path. This mount has a direct line of communication with the hardware of the drone.
- Admin: Admins are responsible for ensuring that communication with other groups runs well and that everyone is aware of the overall aim of the endeavour. They are also responsible for handling all administrative responsibilities and managing the budget.

This focuses on the perception aspect of Ascension. In particular, it will be aimed to investigate whether it is possible to use deep neural networks to detect and position terrestrial robots in real time in photographs taken on the drone. This characterization of the subject raises a number of difficult questions, such as how localization should be done, how accurately it identifies robots of different sizes with different image resolutions, and how all this can be done in real time. There seems to be no question of whether tracking and locating is possible; Rather, the question seems to be whether they can be completed fast enough to board a drone with limited computing resources. It is important that this solution performs well and fast if it is to be useful for the Ascend project. The following sections contain a substantial amount of research, chosen primarily because of their desire to be computationally inexpensive compared to other approaches.

2.3 AVAILABLE IMAGE DATA

As a result of Ascend's involvement in the 2016 IARC, the business now possesses video footage of many arena races, with each one lasting around one minute. A video feed from the downward-facing fisheye camera is also included in the data along with a video feed from each of the four side-facing cameras that are located on board. This demonstrates that we have over 50,000 shots of our competitors that we wish to optimize, and these images may be analysed and used to test or train various methodologies.

The drone is outfitted with a LIDAR (Light Detection and Range Modification), which continually monitors flying height, as well as an IMU (Inertial Measurement Unit), which records yaw, pitch, and yaw at all times. Both of these sensors are continuously recording data. These two sensors monitored the drone's movement in a continual manner. A comprehensive journal of all sensor data and picture recordings, complete with accurate timestamps, was also maintained by the team. This provides assistance in determining the height and angle at which each photograph was shot.

2.4 DEEP LEARNING MODELS

Deep learning is a very large range of machine learning approaches that are all based on deep graphics and are capable of learning complicated concepts without the assistance of a programmer. The phrase "deep learning" refers to this category of machine learning techniques. Simply by seeing new things and adjusting his behaviour appropriately, he should be able to acquire new knowledge. The chart lays up an orderly hierarchy of concepts, each of which may be disassembled into its component pieces and investigated on its own.

When you bring all of these separate thoughts together, you can finally understand the overarching concept, which is more difficult. The most common kind of deep learning model is what's known as a feedforward deep network, which is also often referred to as a multilayer perceptron. A multilayer perceptron is nothing more than a mathematical function that correlates one set of input parameters, such as an image, with another set of output values. In other words, it takes in one set of parameters and generates another set of output values.

MLPs will also be the primary emphasis as we try to construct a network that is able to recognise and locate ground robots inside a picture. Our goal is to do this via the use of machine learning techniques. In order to accomplish this goal, we want to construct a network that is able to identify and locate MLPs. In the next part of this article, we will talk about a few deep learning models that may either be helpful on their own or serve as the foundation for the creation of something else.

2.5 EVOLUTIONARY NEURON NETWORKS (CNNS)

Since its discovery, over two decades have elapsed since convolutional neural networks were first put into use. In 1990, Yann is credited with becoming the first person to successfully use them. He was the brains of the machine that eventually became known as LaNet, which is capable of correctly categorising various kinds of numbers. Although it has been around for a while, it has only very recently become mainstream as a result of recent advancements in graphics cards and interfaces that make it possible for several networks to operate in parallel on the same dataset.

The creation of AlexNet in 2012 was the spark that sparked interest in CNNs, and various enhancements have been presented in the following years illustrating the usefulness of CNNs. AlexNet was the spark that kindled interest in CNNs. In the not too distant future, we plan to take a more in-depth look at AlexNet as well as other of its offshoots to determine the extent of the benefits that may be gained by include them. CNNs get their strength from the convolutional layers, which each have their own set of learnable filters and are the key to their effectiveness.

Through the process of training, each filter acquires the knowledge necessary to become operational for a certain category of characteristics. When processing these features, there are times when a maximum pooling level is used in order to downsample the pictures. This is done in order to make it easier to establish assumptions about the various scales on which different components of the picture are represented at varying levels of detail.

Fold levels make it easier to recognise the abstract form of an element and offer a concept of what its representation looks like when they are used in conjunction with maximum grouping levels. The output that they create is known as a feature map, and it provides information to us on the features that are located in various areas of the picture. Then, by using unique permutations of these characteristics, it is feasible to categorise either the whole picture or just a portion of it according to a variety of criteria. This last step is often reached by using a limited number of layers, all of which are completely integrated by the time the network reaches its conclusion. The extraordinary computer science class that Andrej Karpathy teaches at the university The concepts of DNN and CNN are broken out in CS231n in a great deal more depth.

2.6 ALEXNET

Offered as a submission for the 2012 competition, which, in the words of the people in charge of organising the tournament, "evaluates object detection and image classification algorithms at scale." Soon, we will see that ImageNet is and will continue to be an excellent location for attracting the attention of forward-thinking CNNs. In 2012, AlexNet had a significant performance edge over competing computer vision algorithms and was thus an immediate success because of this advantage. It ended the competition with a mistake rate of just 16%, which was much lower than the error rate of the latter, which was 26%. It was meant to sort 1.2 million images and organise them in 1000 distinct categories. As a result of this, there has been a renewed interest in CNNs, which has resulted in a number of additional design improvements that aim to further increase accuracy. In order to construct AlexNet, the initial step was to link 5 convolutional layers, which was then followed by the addition of 3 fully connected layers.

This is a lot of parameters and neurons for such a flat mesh, as we will see in a moment. In all, there are sixty million parameters and five hundred thousand neurons. The topology of the network is quite dense. This may lead to a method that is not entirely convincing but is reliable for categorising photographs in order to establish whether or not they include ground robots. This makes it possible for slower software to determine with a high level of precision whether sections of a picture include robots and which do not contain robots. This segmentation may be utilised further to offer reliable data for a network that can genuinely recognise the various robots that are working. This network will be able to distinguish between each of the robots.

2.7 GOOGLE LAUNCH MODEL

Additionally, a team from Google took part in ImageNet 2014 and emerged victorious with a network conclusion. Since its inception, GoogleNet has been subjected to a variety of modifications and enhancements. The most recent version of the system is referred to as the Inception v4 model [43]. Architecturally, there is no way that the standard model that AlexNet uses can be compared to it in any manner. This is accomplished by the use of a layered setup that is referred to as the starting module, which drastically decreases the necessary parameter set.

In spite of the complexity of its structure, it employs a factor of twelve less parameters than AlexNet does while yet achieving a far greater degree of accuracy. The fact that this tactic might be tough to completely comprehend and far more challenging to modify is undoubtedly the most significant drawback associated with using it. Instead of using only one loss function, three distinct loss functions should be employed so that the appropriate amount of fine-tuning may be accomplished. displays the whole of the building. The beginning model is appropriate for two distinct applications, both of which are described here.

In the beginning, we intended to make use of AlexNet, which has the ability to provide segmentation results that are more accurate, comparable to what we have accomplished in the past. However, since this model is intended to differentiate between a thousand distinct classes, but only needs to differentiate between two, it is possible that it may not deliver a substantial performance increase in comparison to the AlexNet model or similar approaches. Second, we can utilise a version of this that has previously been taught to students as a foundation for constructing a new sort of network.

It is not inconceivable to scrap the whole of the first model except for its output layer at the very top and then construct an entirely new network architecture in its stead. This sort of learning is often referred to using the phrase "transfer of learning" as its name. Every subsequent convolutional layer that the initial model employs in its training process offers a description of the characteristics that are present in the input picture, and this description progressively becomes more exhaustive as the layer becomes deeper. As a direct consequence of this, the ultimate convolutional layer is responsible for the generation of activations for a considerable number of generic characteristics.

They are stored in the very first iteration of the template, which places them in the very last layer. This layer is completely connected, and its purpose is to describe the kind of picture that is being shown. We may also utilise feature activations as a starting point for a segmentation network or other sorts of useful networks that would benefit from having features as input. For example, a network that would benefit from having features as input is a network that would benefit from having features. A network that is able to recognise patterns is an example of this kind of network. Such a network would benefit from having features as input and would also be able to recognise patterns.

2.8 CARTOON

It competed and placed as a finalist in ImageNet in 2014. It was initially designed as an expansion and augmentation of the AlexNet 2012 model; nevertheless, it is far more comprehensive and efficient than its predecessor. The primary contribution he made was to show that there is a correlation between network depth and performance accuracy. As a result, the 16 levels were seen as being very deep and challenging to train due to the fact that it was difficult to manage the convergence of the deeper layers. The issue was remedied by the developers of VGGNet, who did so by segmenting the network into more manageable parts and then training those parts as a kind of pre-training.

Although it took some time, the end product was rather remarkable. There is now a startup that can be used that does away with the need for pre-training and makes it much simpler to train VGGNet from the ground up. VGGNet is a network that has over 140 million parameters, making it a pretty huge network. The first completely linked layer, seen in blue in Figure 2.5, has perhaps in the neighbourhood of one hundred million of these characteristics. There are strategies that, when implemented, may drastically lower this amount without negatively impacting the performance of the network as a whole. In addition, VGGNet is often used as a feature extraction network. This use of the network solely makes use of convolutional layers in order to produce feature maps for other kinds of networks.

This indicates that we still have around 15 million parameters available. Nevertheless, the combination of its ease of use and its precision is the primary reason why it is used. It does this by piling up convolutional layers of a basic 3x3 grid one at a time, sometimes mixing maximum levels in order to scale functionality. This is done directly on top of the AlexNet model. Because of this, it is simple to comprehend, alter, and put to use. Because of its modularity, it is available in a variety of sizes, although the depths of 16 and 19 are by far the most frequent.

2.9 NETWORK

The outcomes of the Res Net competition that took place in 2015 provided evidence that the deeper the model, the better it performed. Does this imply expanding the available processing power, adding more layers, and enhancing the performance of the network? However, this does not seem to be the case. This is due to the fact that there is a large fall in the amplitude of the gradient flux after about 20 slices while the signal is being backpropagated. Because of this, the network is unable to pick up any new information. A "network within a network" design is what ResNet proposes as a solution to this challenge. This architecture may be found in ResNet.

The most important distinction is that ResNet is made up of a vast array of flat networks, some of which overlap and very rarely pool together. Consequently, the issue with the loss gradient may be sidestepped during the training phase. Adjustments after a workout are considerably simpler to do when utilising this system, which is another advantage of using it. When even just one layer of a trained VGGNet is removed, the network's performance suffers substantially, and it eventually becomes almost completely worthless. Using ResNet, you can eliminate one of the deeper layers with an accuracy loss of no more than 5% at most. The initial few layers will have a greater influence, but the overall effect will be considerably lower than it would be with a more typical mesh. A research that Veit and his colleagues developed tries to provide a more elucidating explanation of how and why ResNet operates. Read this post if you want further information that is more in-depth, since that is what we propose you do.

2.10 COMPARISON OF CLASSIFICATION NETWORKS

The trend that has been seen up to this point is confirmed by ResNet[17], another winner of the ImageNet competition, which took place in 2015 this time around. 2015 was the year that the tournament took place for this year. Does this imply that all we need to do to increase the performance of the network is add more layers on top of the ones that are already there? Do we also need additional processing power? Sorry, but that won't work right now. Because of this, the network is unable to learn new information beyond around 20 layers, since the quantity of gradient flux that takes place during backpropagation substantially diminishes after reaching this point.

This issue is resolved by ResNet via the development of an architecture referred to as "network-in-network," which is comprised of two independent networks. The primary distinction between the two is that ResNet is made up of a multitude of flat networks that are layered one on top of the other, and it also often includes a pooling layer. As a result, the issue of the disappearance of the gradient may be solved using the training approach. After the first training, it is much simpler to make adjustments using this layout, which is another advantage of the design.

If you remove only one layer from a trained VGGNet, you will see a significant drop in performance, and beyond that point, the network will effectively no longer be functional in any meaningful way. By using ResNet, you are able to get rid of each of the deeper layers with an accuracy loss that ranges from 5% to 0%. Even while the influence of the initial layers will be higher, it will still be far less than it would be with a mesh that is more conventional. If you want more in-depth information, you can read the essay that Veit and his colleagues have published that tries to explain how and why ResNet works. It is absolutely something you should read if you are interested in the topic.

2.11 SENSING NETWORKS

Although their performance on the tasks for which they were intended is astounding, neural networks are, however, restricted to fairly rigid framework topologies. They have a set number of input nodes and an equally fixed number of output nodes, and none of these nodes can be readily replaced without first undergoing a considerable amount of retraining. One of the most prevalent uses of neural networks is image classification since it only requires a single input picture of a particular size and provides a single output with a probability distribution across a given set of categories. This makes image classification one of the most common applications of neural networks.

Altering the top layer of a classification network so that it learns to also output the bounding box coordinates in addition to the classification output makes it simple to convert the output of a classification network into a single-element recognition network. This allows the output of the classification network to be easily converted into a single-element recognition network. The implementation of such a system is doable given that adequate training data of acceptable quality is available, which may be challenging to get. When we seek to recognise a variety of items that are shown in a picture, we run into the true difficulty of the situation.

When evaluating alternative categorization networks in terms of the number of layers, speed, and accuracy, it has become clear that using a constant number of bounding boxes for output is no longer the optimal solution. When operating on a Titan X Pascal GPU with cuDNN 5.1 loaded, the speed of forward and backward scans is measured in milliseconds for both directions. The Top 1 Error metric is a measurement of how often the best output estimate is labelled as inaccurate, while the Top 5 Error metric is a measurement of the proportion of times that the right label is not even included in the top 5 output estimates.

These two mistakes are together referred to as prediction errors. The photographs included in the 2012 ImageNet validation kit include the most frequent instances of faults 1 and 5. These photographs have the middle portion of the picture cut off before they are sent to the network for viewing. Only data consisting of multiple slices taken from a variety of picture locations were available for VGG-16 and VGG-19. These statistics were the only ones that were at our disposal. The performance of all models improved when they were tested with numerous harvests at the same time, with the VGG models holding a modest edge.

A variable amount of things are available to be seen. Creating a combination of meshes that, first, finds viable places for the object, and then, second, gives a categorization mesh for each of the proposed regions, is one approach that might be used to solve the problem. The R-CNN networks are particularly prominent examples of this phenomena. The second possibility is to produce a mesh that produces a certain number of detections for each and every part of the picture that might possibly exist, and then to apply a threshold to the findings produced by that mesh. A one-shot detection network is the second method, and YOLO, SSD, and YOLOv2 are some of the most well-known and widely used examples of this kind of network.

When we talk about a "one-time detection network," this is exactly what we mean. Both approaches are shown throughout the presentation as well as the ensuing debate that follows. There is a continuous production of a significant number of research papers and articles on the subject of object identification techniques for numerous objects using deep learning. The goal of these works is to establish which method is both the most accurate and the most efficient. They are always improving, and the degree of precision that some of them have already exhibited will be more than sufficient to produce an outstanding sensing and tracking system for ground robots. However, the speed of computing is not necessarily the most important factor, and many of the recommended approaches have a performance that is too sluggish to be performed in real time on a system powered by Nvidia Jetson. Therefore, the primary emphasis of the sections that follow will be on efficient alternatives to the conventional approaches that are used to identify a large number of objects.

2.12 R-CNN FASTER

The Faster-R-CNN approach[37] accepts that the Faster-R-CNN method considerably increases CNN performance, but it disregards the fact that the creation of regional proposals is still moving at a very sluggish pace. Even when utilising CNNs that are accelerated by the GPU, regions are still formed using methods that are programmed into the CPU. The issue is resolved by Faster R-CNN by the use of Region Proposal Networks (RPN), which share convolutional layers with conventional CNN. As a result, the amount of time necessary to create proposals is drastically cut down. As a direct consequence of this, the total performance has been much enhanced, and the new frame rate is about 7 per second. It is possible to do this in real time, but given the restricted hardware characteristics that our project will employ in comparison to the powerful GPUs used in the articles, we want a solution that is even quicker.

2.13 YOLO - YOU SEE ONLY ONCE

The University of Washington is responsible for the development of the method known as YOLO[34]. It entails the implementation of a one-of-a-kind neural network that, given a full-size picture, can anticipate bounding boxes together with their associated classifications. This is an effort to do away with techniques that are focused on regions, such as the one where you try to categorise the same area thousands of times. As its name indicates, the objective of this technique is to concentrate the viewer's attention on a single facet of the picture at a time.

In order to do this, they began by dividing each picture into a 7x7 grid, with the expectation that each cell would provide a unique prediction. To begin, they start with the presumption that there is a finite number of bounding boxes and that each one contains the core cell. A minor constant that is always set to 2 throughout the run is the number of predictions that are made for each individual cell. In the second step, it must determine an estimate for the confidence values associated with each of the bounding boxes. This demonstrates how firmly YOLO Network feels that the bounding box encompasses a number of different things.

Even if there are no objects in the cell, the same bounding boxes will be shown; nevertheless, the confidence values should be set to a low number. When we aggregate the findings of all 7x7 cells, we obtain something that has a strong similarity to the bounding box estimates mentioned in the R-CNN techniques, but with considerably less overlap than the original estimates. We now have $7 \times 7 \text{ x } 2 = 98$ boxes with accompanying confidence values, as contrast to R-CNN, which has over 2000 possible bounding boxes that need to be individually categorised. Each cell on the grid is represented by two individual boxes. Now, the network works to give each bounding box a category that corresponds to it.

This is accomplished by carrying out the action for each of the table cells rather than for each box on its own. We may make the assumption that the object that is included inside the bounding box is the same object that was declared for the cell since the centre of each bounding box is located within one of the cells. All of the bounding boxes, in addition to the class probabilities and confidence levels that correspond to them, are included in the output in its final form. We just establish a cutoff point for the confidence values, and after that, we look for the appropriate boxes and their classes, rejecting those that have a low possibility of being correct. The process that YOLO went through may be seen in this section.

2.14 SSD - MULTIBOX ONE SHOT DETECTION

In a manner similar to that of YOLO, SSD[25] is an acknowledgement network that generates a fixed-size collection of receipts. SSD[25], on the other hand, chooses the detection outputs with the greatest values and supposing that these are the real object detections. This is in contrast to YOLO, which chooses the detection outputs with the lowest values. The standard classification networks are used to produce the initial SSD level, which is then used in Section 2.2. Nevertheless, in order to produce a collection of feature maps, the fully linked final layers of the underlying network had to be eliminated. In the investigation, they came to the conclusion that it would be best to make use of the VGG16[6] mesh that is shown in Figure 2.1.

However, they did make some adjustments to it in order to make it shorter and quicker to run without severely compromising its accuracy at the end of the backbone mesh. Because the size of these feature layers continuously decreases, it is possible to extract features at a wide variety of sizes, for example. Each map has the dimensions mn times p, where mn represents the number of feature cells it includes and p represents the number of depth levels that the feature has. - Every single one of these cells may make use of the vector. After that, the feature maps of each layer are put to use for two distinct purposes: first, as input for the layer that comes after it, and second, directly to forecast detections.

We are able to construct a fixed number of recognition estimates for each cell, which we have set to four on paper, depending on the location of that cell inside the picture. This allows us to accept bounding boxes that have slightly varied sizes and outlines. Because the values of the initial layers are lower, the image is broken into smaller fragments, which leads to the finding of smaller details in the picture. Cells that make up deeper layers occupy a bigger section of the picture, which enables you to describe objects that have a larger surface area. These cells are also more densely packed. The fact that SSD is completely expandable and adaptable to a wide variety of sizes is the primary benefit of using SSD in comparison to YOLO.

None of these skills are possessed by YOLO. Because they need a significant number of connections, fully bonded layers come at an exceedingly high cost. In point of fact, increasing the number of connections results in a significant slowdown of the calculation. Due to the fact that SSD only employs convolutional layers, it is able to channel all of that additional processing power to concentrate far more detection for each point in the picture. It has been shown that YOLO only produces 98 predictions for each class, but SSD is capable of providing 8732 predictions for each class while maintaining the same frame rate. Because of this, SSD has a huge edge when it comes to identifying the size of a variety of objects, despite the fact that it is far more likely to make placement mistakes.

At a satisfactory level despite the limited number of training samples. The detector has been trained to locate things of a certain size, but it can be simply applied to a wide range of picture scales, which enables it to identify objects of a variety of sizes. Additionally, Davis E. King is the creator of dlib, which is a free and open-source machine learning toolbox. Recent developments at dlib have resulted in the implementation of the MMOD methodology. This method replaces the HOG detector with a convolutional network. It was necessary to train the face detector on a collection of four images, each containing between four and six faces, in order to get a good result.

Even more unexpected is the fact that it was able to match the performance of the cutting-edge R-CNN Face ID system trained on 160,000 photographs, despite the fact that just 4,600 images were utilised in the training process. It further asserts that frames with a resolution of 640 by 480 may be displayed in 45 milliseconds, which, when carried out in a sequential manner on a GPU, results in a frame rate of 22 frames per second. When pictures are processed in batches, it only takes 18 milliseconds for each image, which results in 55 frames being produced per second. (FPS). The main drawback is that it seems to measure on a single scale, which means that it only searches for items that are the same size throughout. This methodology is referred to as MMOD.

2.15 FAST IMAGE SCANNING WITH DEEP MAX POOLING CONVERSIONAL NETWORKS

A broader image will be applied to each patch using the sliding window approach that is explained in a classification network. Feature maps will be created for each of these patches individually. Due to the fact that we need to recalculate the identical entity mappings in a slightly different place for each overlapping component in these patches, the process is somewhat more time-consuming. A alternative method known as "extended maps" is detailed in the article. A collection of all of the feature maps of each patch in an image is what constitutes an extended map.

In contrast to the floating window approach, which often results in the production of multiple feature maps owing to overlapping windows, this methodology never makes two copies of the same feature map when it computes it, which proves its use. If there is not a maximum number of people that may be grouped together, then this simple adjustment can be accomplished with ease. However, the power of current neural networks may be primarily attributable to the combination of maximal pooling levels and convolution levels.

This is one of the main reasons why these networks are so effective. When maximum clustering is applied to feature maps, this causes the maps to scale, and as a consequence, some of the information about individual areas in the picture is lost. For instance, if we utilise a process called 2x2 max pooling, then we will lose information for every other patch that we apply. When maximum pooling is performed at several levels, the quantity of information that is lost rises with each subsequent level. As a preventative precaution, we are able to alter the maximum pooling levels, which will result in the generation of a number of fragments that, when combined, will fill the full picture.

This may be shown by presenting a 2x2 filter, which is represented by four little boards. These boards, when merged, will achieve maximum pooling of all components that make up the input piece. The completed collection of components may then be pieced together in a variety of different configurations to represent the feature map that will be generated in a specific region of the input picture. The authors of the research took use of the chance to build a per-pixel segmentation method that was capable of segmenting a picture in 15 seconds, even when executed in a Matlab CPU programme that had not been optimised.

This was accomplished by following the instructions in his picture. On the other hand, while executing highly optimised Cuda code on a graphics card, the amount of time it took to execute the same thing with the typical floating window approach was 492 seconds, which is 32 times slower. The removal of features from the input picture only has to be done once in order to realise significant time savings as a direct consequence of this fact. The implementation of a method that makes use of augmented maps that Cuda provides may be used to construct a real-time segmentation algorithm..

2.16 NETWORKS OF RETURN NERONS

In recurrent neural networks, also known as RNNs, the topology of the network is constructed in such a manner that the output of the network feeds back into the network itself, either with or without the involvement of new inputs. This may take place with or without the participation of additional inputs. Because of this, the network has highly powerful properties. RNNs are very modular and have the capacity to carry out an infinite number of iterations, accept an infinite number of inputs, and generate an infinite number of outputs. Every single one of the several neural network processing methods that we have looked at up to this point always has a fixed-size input, a fixed-size number of operations, and a fixed-size output.

Detection networks, like other networks, have a certain amount of detections every frame. We only separate and throw out detections that have a low level of confidence. The second characteristic of RNNs is that they have the capacity to retain information. The network is able to "remember" what it has done in the past and build on it because it has an internal state that is continually updated as new inputs are received. This allows the network to "remember" what it has done in the past and build on it. Although iterative neural networks are most typically employed to tackle issues that require a series of sequential stages in nature, such as voice and video processing, they may also be used to circumstances that do not intrinsically contain steps.

This is because iterative neural networks are recursive in nature. First, let's look at a few more unusual circumstances. Research that use deep tracking makes an effort to illustrate that one application of RNNs that might stand to gain from being loosely linked is object tracking. The next step is to make an informed prediction on the subsequent appearance of the sensory output. Because the input data is a straightforward one-dimensional vector of lidar measurements, the model that they utilise is far simpler to comprehend when compared to the model that we make use of.

In order to do this, the LIDAR sensor first conducts a search for obstacles in the area in front of it and then turns the information it gathers into a two-dimensional picture. It then draws using the information it has acquired about the motion of objects in the line of sight, making a prediction about what the 2D picture will look like after the next phase has been completed. Due to the fact that we get our data in the form of an extremely noisy two-dimensional picture, doing this task is practically impossible for us.

In addition to this, we employ a moving camera, which messes with the natural flow of the dynamics inside the photos. It is quite difficult to create an exact estimate of how the full picture will seem in the subsequent image due to the many different aspects that have the potential to alter the final outcome. This will only be successful if the noise that can be created is mitigated in some way, such as by physically separating the robots. Having said that, this will call for a much increased amount of computational power. The last and most damaging thing that can be said about this technology is that it needs a constant amount of time in between each frame in order to provide estimations.

Because our operation is carried out on a computer system, it calls for a number of programmes to be active all at once. The central processing unit (CPU) of the Jetson TX2 board will be used for other activities, but we still need to be able to accept even very short delays between frames. Multiple tracking algorithms, including DeepTracking, will have difficulty making accurate predictions if there isn't a regular amount of time between each frame. In this regard, other procedures are more trustworthy than this one.

Even more impressively, it eliminates the challenges associated with data mapping and trajectory prediction, which is a significant advance in the right direction. Even the most promising monitoring strategies are still lacking in accuracy when compared to more contemporary monitoring methods. The underlying difficulty is that the area of RNN monitoring is still in its infancy and that this is the case.

The fields of science and engineering make substantial use of linear algebra, which is a subfield of linear algebra, which is a mathematical study. Because linear algebra belongs to the continuous rather than the discrete branch of mathematics, the majority of computer scientists have relatively little knowledge of this topic. A strong grasp of linear algebra is required for working with many machine learning algorithms, notably deep learning algorithms. This expertise is necessary in order to comprehend and operate with these algorithms. In conclusion, we will start our talk on deep learning with a shortened explanation of the most important aspects of linear algebra.

You are free to skip this chapter if you have prior experience with linear algebra and do not need to learn the material presented here. We suggest that you check out The Matrix Cookbook if you are already aware with the concepts discussed above and are searching for an exhaustive reference sheet to brush up on fundamental equations. (Petersen and Pedersen, 2006). You'll learn enough about linear algebra in this chapter to be able to read the remainder of this book; nevertheless, we highly suggest that you consult another resource that is devoted completely to teaching linear algebra, such as B.Shilov. (1977). This chapter totally glosses over a number of significant linear algebra topics since having that knowledge is not required in order to comprehend the material covered in deep learning.

The link that exists between each individual frame in the film. When attempting to forecast the appearance of feature maps at time t+1 using optical flow, feature maps at time t should be used. This is done as an alternative to starting from scratch every time a new frame is added to the movie and constructing feature maps. Instead of a conventional CNN such as VGG-16 or Inception v3, the new feature maps are generated with the help of something that is referred to as the feature flow mesh. Considering that feature extraction is the step that takes the most time in the majority of recognition and partitioning networks, this is a huge improvement in efficiency.

When using a solution that is frame-by-frame, the computation is performed on the first frame at the point in time when the time value is equal to zero. The feature maps for the frame at time t=1 are calculated by first calculating the optical flux between frames 0 and 1, and then progressing via a particular propagation function using the feature maps obtained at time t=0. This allows for the computation of the feature maps for the frame at time t=1. This then results in the generation of feature maps for the frame at time t = 1, which can be shown below. Calculating optical flow may be accomplished via the use of a variety of approaches; however, the approach known as FlowNet, which is based on neural networks, has been established as the industry standard.

The efficiency of a detection method that is founded on ResNet-101 is therefore elevated as a result. decreases from 4.05 FPS to 20.25 FPS in terms of frame rate. Although the tested network projects include a significant amount of documentation, it does not seem that there is any code that can be accessed at this time. It would seem that this approach is capable of significantly increasing its speed, however doing so would reduce its accuracy. using a number of different approaches, each of which is aimed to speed up a particular step of the process of deep learning. Performance may be increased by using these several approaches. Some people concentrate on the process of training in order to make it more effective.

A good example of this kind of system is one that is intended to hasten the process of generating segmentation networks by making use of the maximum pooling levels. In light of the fact that this is a much more interesting issue in light of the conditions, we will investigate some of the tactics that concentrate on enhancing the performance of the network during runtime.

CHAPTER 3

PROBABILITY AND INFORMATION THEORY

3.1 NETWORK DETECTION METHODS

As the objectives of this study can be approached from different angles, our first step is to evaluate all available options and choose approaches that require further research. In the following sections, we will discuss approaches and how they can be combined to achieve the above goals. In this section, we will look at the deep learning approaches discussed in it to see how they can contribute, both as stand-alone solutions and in combination with other learning strategies. As some of these approaches were not found in the initial phase of the research, the conclusion of this section has been modified in the process.

We will distinguish between the techniques chosen to be followed at the beginning, and the methods that are gradually added and developed on the chosen ones later on. These distinctions are made on the basis of the chosen methods to be followed in the initial phase. The goal of this part of the project is to find a system that can measure a single Jetson TX2 and receive input from four cameras simultaneously. In a perfect world, we'd like the Jetson system to be able to process at least four frames per second so we get updated directions for each camera every second.

3.2 R-CNN BASED SENSING NETWORKS

Despite many improvements to the original implementation to enable faster development, the algorithm is still quite slow, yielding 7 frames per second at a high GPU range, probably only about 1-2, as the data shown in Running on a Jetson TX2 shows. frames per second, which is an unacceptable frame rate. However, there are two different ways to improve the approach to make it more comfortable. One strategy would be to choose the lesser field idea without discarding any, which will result in bots being included in the endgame. An alternative approach would be to implement a faster way to classify fields. It's unlikely we'll make an improvement that speeds up the technology enough to hit the required FPS limit. Despite R-CNN's great research interest, it is unlikely that we will make this improvement, as no one has developed the tools to do so .

3.3 JOLO

YOLO undoubtedly meets the speed criteria for real-time operation, and its developers say it has very good accuracy on larger detection datasets such as VOC[9] and COCO[24]. The accuracy of YOLO's detection of very small objects is the main concern of this technology. The research work acknowledges that it lacks precise position information and divides the image into a 7x7 grid with two detections in each cell. The final mesh may need to be able to identify many overlapping robots that might be problematic. To combat this, the number of cells can be increased to just over 7x7; However, it has the potential to greatly increase the speed of the forward pass. Decisions like this will need to be re-evaluated as we learn more about how accurately YOLO does the job of identifying the robot. But this is one of the approaches we want to use for further testing.

3.4 FAST SEGMENTATION NETWORK

The ability of segmentation networks to discover regions of an input image containing features of interest within a single run of a network is the primary reason they are considered in this context. As you can see, the mesh isn't as fast as we'd hoped, but DeepLab shows the biggest promise, reaching around 9fps on a high-end GPU. DeepLab is an extremely interesting option for me because we can skip the CRF step in the process and make it a bit faster. We were also able to reduce the size of the input image to make it even faster. If we make these changes, we can create a segmentation process that runs fast enough on a Jetson system to meet our speed requirements.

However, although it has proven to be the most promising approach so far in terms of accuracy, there are still problems with the position accuracy of the YOLO strategy. Therefore, in addition to YOLO, we investigated whether a solution combining segmentation and location extraction is a viable option as a final solution. However, this requires a fast method to extract robot positions from the segmentation image. As we look at strategies for creating a training dataset below, we'll cover the many approaches that can be used to achieve this.

With these two main possibilities, we started the process of creating the training dataset described in it. Other techniques, such as SSD and YOLOv2, had previously received little attention in the field of object identification work, so they were identified only much later in the process. As a direct result, the focus was on developing a survey dataset for YOLO and a segmentation dataset for DeepLab. The implications of this will be discussed in more detail later.

3.5 SSDs

Therefore, the technique of partitioning networks was mostly a topic of discussion at the time of writing. It turns out that we are not the only ones who consider the usefulness of such a strategy leading to the implementation of SSD networking. This is a simpler technique that creates recognition directly from feature maps, rather than splitting the process into segmentation and location extraction. It captures exactly what we want to achieve, but in a simpler way. It shows that SSD outperforms both YOLO and any segmentation-based solutions that can be derived from DeepLab in terms of speed and accuracy. The system can be trained and tested very easily as the entire source code can be viewed online. SSD is a strong competitor for robot identification and the ultimate goal will be to allow the SSD to be trained and tested with various input sizes to determine the configuration that provides the best speed and accuracy in robot identification. .

According to the developer of YOLOv2, it is much faster and more accurate than SSD. The only problem with this strategy in terms of connectivity is that it's released after the YOLO investigation is complete. Therefore, v2 was only found at the end of the file write operation. However, because it is easy to implement, the source code of the project is tested and evaluated to determine whether it is a viable option for the final solution.

3.6 FLOW EXPANSION OF DEEP FEATURE

The in-depth feature flow approach presented in Item 46 is not a stand-alone solution, but can improve the efficiency of a network such as an SSD that uses a backbone network to generate feature maps. It has been shown that the solution can locate the feature map of a frame by combining the optical flow in the image sequence with the feature map of the previous frame. This can speed up the object detection process in video footage, but only if the underlying mesh is up and running after a few iterations. This is necessary to ensure that the feature map contains all the necessary features, including those that may have been added to newer frameworks.

They use a step value of 10; this indicates that the slow but sure spine is responsible for rendering an image every 10 frames in the article. Since the video used for benchmarking was recorded at 30 frames per second (FPS), this indicates that the backbone had to be divided by 30 by 10 to get 3 times per second to match the feature map and sustain throughout the day. Assuming the finished product can sustain a frame rate of 12 FPS on a Jetson TX2, each of the four cameras has a frame rate of 3 FPS. After that, the backbone network has to run every frame to keep the feature map up to date, leaving no time for FlowNet to run. However, if the required step size is different, a very slow-moving drone is detected at the entrance, or the final solution reaches a frame rate of over 12 FPS, it may be worth implementing.

These two scenarios will increase the likelihood that the required step size will be different. Assuming that FlowNet can poll twice as fast as the VGGNet model used on SSD, we only need an SSD framerate of 18 FPS to reach 24 FPS with a combined solution. In fact, there is enough time to run FlowNet after each VGGNet discovery. The execution time of a VGGNet is 1/18th of a second, but the execution time of a FlowNet is 1/36th of a second. One second is equal to 12 times 1/18 plus 12 times 1/36. As this is more of an add-on than a complete solution, it will be reviewed as the detection solution is developed to potentially increase speed and capture higher frames per second (FPS).

3.7 MMOD AND DLIB

There are several aspects of the MMOD algorithm and implementation of CNN in the dlib that have a lot of potential, but there are others that are cause for concern. Its most obvious feature is that it searches for items within a predetermined bounding box. A robot detection algorithm should be able to detect robots of different sizes; This means that the MMOD-based mesh must be run multiple times at different scales for each image to ensure the correct size and location. To recognize robots of various sizes, a robot recognition algorithm must be able to recognize robots of various shapes. This can significantly slow down performance. As we have more information about the overall performance of other techniques such as SSDs, we will implement this approach and make any necessary adjustments.

The first conclusion that can be drawn from reading this section is that you should test the YOLO network, as well as a mix of segmentation and localization, to see which works best. At the time, YOLO was generally the fastest end-to-end solution we could find and therefore one of the top contenders for always-on response. The only thing stopping some was the possibility that a segmentation network could provide faster and/or more reliable results when used in conjunction with rapid location extraction.

Because the research industry is changing so fast, we had to change direction mid-way through when solid state drives (SSDs) were found and chosen as the most promising alternative. SSD's effectiveness in identifying real-time bots with limited access to computing resources will be evaluated when it undergoes rigorous testing as a standalone solution. This will be the main focus of real-time sensing studies from now on. Due to the inclusion of YOLOv2 so late in the process, tests will be secondary so that we can evaluate the comparison to SSD. Third on the priority list is whether dlib and Deep Feature Flow are worth using.

3.8 METHODS OF RECOGNIZING RECORDS

Now that we have a general idea of what we want our final solution to look like, we can begin to meet the requirements for it to perform at its best. The first thing you need to do before developing a sensor network is to collect a substantial amount of training data that simulates a variety of plausible real-world situations. We understand that one of our goals is to design a method that accepts camera images as input and processes them to produce a segmentation image or collection of detections. The vast majority of neural networks are created and evaluated using one of several standard datasets provided by various image processing competitions . (network photo 2015). However, segmentation and detection of different colored ground robots with poles added from time to time is not a standard task and requires a dataset of robots and their positions at different angles and directions.

Segmenting and detecting different colored ground robots with the occasional pole attached to them is no ordinary task. We noticed that Ascend's participation in last year's IARC competition resulted in a significant number of photos that, in many ways, reflect the types of tests we want to train the sensing network. The image extracted from the dataset is shown in Figure 3.1. However, none of the currently available photos contain information about the robot's location. Therefore, the focus will be on ground-based robotic sensor technology, where recognition accuracy is far more important than computational speed to reliably generate a training dataset based on photos from the previous year's competition. This information is then used in training in the procedures covered. Given the importance of accuracy at this stage of the process and the lack of pre-developed strategies that can speed the work, a significant part of this is devoted to developing a reliable dataset for robot detection.

The least complex and time-consuming technique would be to manually define each image. However, as this would be time consuming and tedious, the focus will be on developing an automated system that uses multiple computer vision algorithms to reliably tag photos. To see if we can quickly create a recognition set, we will first look at a set of basic object recognition strategies that have been used in computer vision for many years. These often depend on some form of pattern recognition, such as edge detection, color matching, or feature matching.

Pattern recognition can also be in the form of matching features. If these approaches fail, our team will focus on developing a two-stage detection method that first separates the robots from the image and then determines the exact position of each robot based on the segmentation image. This allows us to hit two birds with one stone. This gives us a collection of segmentation photos and a dataset of bot detection images that we can use to evaluate the effectiveness of both techniques.

3.9 COMPARISON OF MODELS

The simplest solution would be to use a feature detection method such as Harris-Keil or scale-invariant feature transform to extract local feature descriptors from the input image, and then use an algorithm to match them to identifiers we already know represent the terrain. Robot. This will be the most effective way to fix the problem. This approach has a high probability of failure for two different reasons. First, there should be a very large collection of photos to find suitable descriptions. These should be general enough to allow any type of robot to relate from every conceivable angle, but not so general as to create false perceptions in the background around the arena.

Second, the items we are looking for are so small compared to the full image that we may not get a bot identifier set large enough to accurately identify them without adding a very large set of unwanted identifiers. This is because we include many identifiers unrelated to bots. This has the potential to significantly slow down the process and negatively affect perception. The Ascend NTNU team has been experimenting with feature detection techniques over the past year and the results have been rather disappointing. This should serve as a final argument against using this strategy.

This has been written about Ascension for various reasons, one of which is this particular aspect. Extracting images taken at different times to determine the positions of moving objects is the basis for another practical object tracking method that has applications in a variety of environments. We are aware that ground robots will spend most of their time in motion and it is possible for them to be removed from the permanent land grid beneath them. This strategy is often used only when the camera position is fixed at one point, as it simplifies the process of determining which components of an image are moving and which are still.

We are aware that the cameras mounted on the drone will move; Therefore, for this solution to be effective, it is necessary to understand the movement of the spherical image from the movement of the spherical image in order to monitor the movement of the camera and isolate the movement of the robot. This strategy leads to the creation of difficulties. At first, the robots seem to be static relative to the ground while actually spinning. Second, this method will at best provide a segmentation image that distinguishes ground and background robots . Other methods, discussed in the next section, can produce segmentation images with a higher degree of confidence.

3.10 SECTION OF SLIDING WINDOWS

Instead of training a network to recognize multiple objects, a much simpler classification network can be trained to understand the difference between ground and background robots. This is a simpler alternative to using a multi-object detection network. Again, this can be used in a Sliding Window action shown in . This solution is probably not fast enough to be used as a final solution, but we're keeping our fingers crossed. Its main use would be to create a collection of photos where the robots and the background contrast with each other. Differentiating between background and robot should be a reasonable problem for most traditional deep classification networks, as these networks have been shown to perform well in more general classification tasks such as the ImageNet test [40]. This technique still requires a large amount of training data, but it is much easier to generate than tracking data. All we have to do is collect still images of the robots from different angles and then use these images directly.

This strategy includes the following stages: first, the development of a CNN bot classification network; second, using the mesh developed in the first step as a floating window to generate a segmentation image; and third, deriving the robot's positions from the segmentation image.

3.11 REMOVING LOCATION FROM SEGMENTATION IMAGES

Images obtained through segmentation are advantageous because they contain only parts of the original image that show things of interest. So we don't have to worry about the possibility of unwanted color combinations or patterns appearing in the background, which could cause false positives. Therefore, it will be easier to extract location data from a segmentation image where the probability of background noise is high and there is confusion about the full view of the elements from all angles. However, segment images are also often very simplified versions of the original images from which they were created. The only information left is whether a collection of pixels (called "drops" by the way) is part of an object.

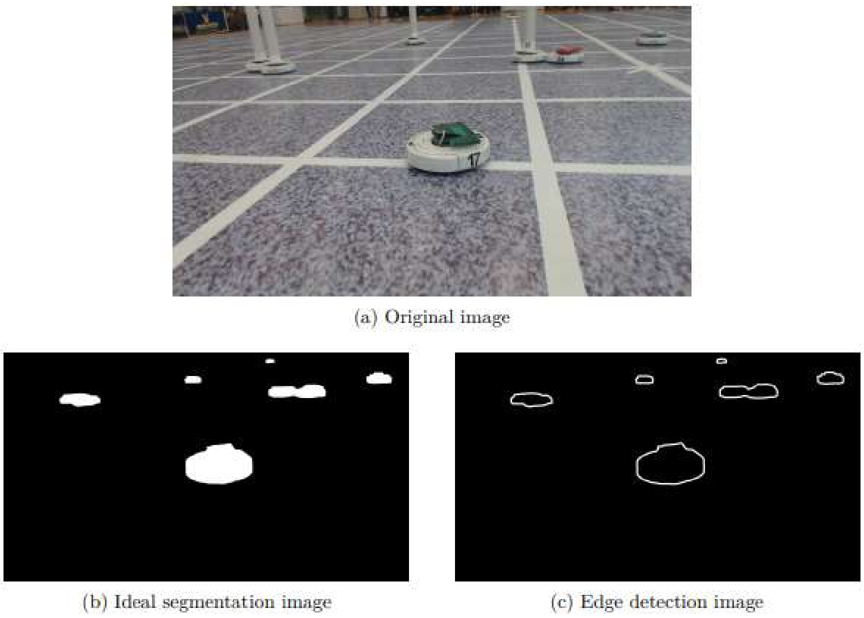


Figure 3.1: The original image in (A) and the corresponding manually generated ideal segmentation in (B). Image (C) shows the detected edges of the segmentation image.

The methods we're interested in, that's really trying to take advantage of it one way or another. They scan the image for specific shapes and make a living from the fact that the only information they can get is the size and shape of the different droplets. It is rather arbitrary to determine the position of drops containing only one element. The real test comes when we need to determine whether a single droplet contains more than one element, which in itself can be a difficult task. To get a better idea of how subtracting positions from segmentation photos can actually be done, let's take a quick look at the many possibilities.

3.12 GENERAL HOUGH CONVERSION:

The first strategy we can consider is to apply a generalized Hough transform to the image to find robot shapes in the image. An edge detection approach would be needed for the Hough transform to focus only on the edges of the image . The segmentation image shown in Figure 3.1 is shown again in Figure 3.2(c), but this time with a Sobel contour filter applied. Next, we need to provide a form for the generalized Hough transform so that it can look at the image.

This could become a problem. The generalized Hough transform can be said to be rotation and scale invariant according to user demand, but the shape of the item it is looking for must still be specific in order for it to accurately describe objects. In Figure 3.1, there is no single figure that completely encompasses robots. They are all oval in shape, but the rear boot is longer on the sides than the front boot. We can also observe that the two entangled robots on the left can easily be confused with a single larger robot. Therefore, it is difficult to create a generic form that can be used to search for the image.

3.13 INTERMEDIATE GEAR AND CAM:

The purpose of the process, known as average shifting, is to find a window with the highest possible pixel density in the images used to represent probability distributions. The individual probability for each pixel in an image can be determined using histogram back projection, which is often used with this technique. This is exactly the same as the segmentation image created by the floating window here; however, histogram backprojection works similarly to per-pixel pattern matching and is therefore much less reliable. The shift averaging technique is initiated in an image area containing pixel data that contains more than solid black. It uses a fixed-size search window, and at the start of each iteration the center of the window is moved to the area of the image where the pixel density is highest. provides an example of this technique and demonstrates how it can work on a sample robot segmentation image. One of Meanshift's limitations is that it uses a fixed window size; this means it will have trouble matching boots correctly as it doesn't know what size they will be.

3.14 MMOD AND DLIB

It is possible that the fast learning capabilities of the MMOD loss function are an interesting feature that deserves further investigation. By manually creating a small survey dataset and using it to train the CNN, we were able to train the CNN dlib to at least a certain level of accuracy. It will then be tested on many new records, the results will be carefully evaluated, minor errors will be quickly corrected and the data will be used for further training. Even at this point there are no performance limits, so multiple resizes can be done per image to create bounding boxes that fit the image perfectly. Unfortunately, this technique was found shortly after the dataset was created; Had it been discovered early, it would have been much easier to describe a robot that does not require a large sensor dataset for training.

Given the assumption that traditional object recognition approaches, such as pattern matching, cannot provide sufficient reliability when used to identify small ground robots, it is necessary to construct the dataset in a different way. The method can be simplified by first separating the problem into its components, called the segmentation part and the location extraction part, respectively. This strategy has the advantage of generating data for segmentation and discovery, which can be considered a segmentation solution with the potential to be as or better than an end-to-end discovery network.

Cam shift and average shift are interesting approaches that should be evaluated to determine whether they provide a direct answer to the location mining problem. If the results do not provide useful information, we will examine the possibility of using large filters to allow for more controlled removal of pixel centers of mass. Since the dlib was just discovered, it was not included in the compilation of the datasets. However, it has the potential to be much more efficient than the approach adopted here and is therefore seen as a potential upgrade for the overall process.

3.15 CREATING A CNN ROBOT CLASSIFICATION

The approach we want to focus on is based on a simple classifier that can label a 64 x 64 image whether it contains a robot or not. The classifier will be used to detect where the robots are in a larger image by drawing a 64x64 window on it and matching the classification result between 0 and 1 as the probability that the window contains a robot. This is accomplished by dragging the window over the larger image. We want CNN to be able to more accurately classify whether the center pixel is part of the ground robot and will train it accordingly. This is because its intended use is in a technique known as casement windows. Probably this approach isn't fast enough to be used as a permanent solution, but we're confident it's reliable. Its main use would be to create a collection of segmented photos where the robot and the landscape are separated.

The AlexNet network was chosen as the best option for the design of this CNN classifier. It was chosen for a number of different reasons, including the following: First, it demonstrated the ability to accurately distinguish a significant number of classes, although several CNNs claim it surpasses Alex Net in terms of accuracy, it should demonstrate it. It's accurate enough for the binary sorting job that ground robots have to do. Second, unlike the other networks discussed, it does not have a very deep structure. Therefore, it is an advantage that the images we want to classify are small in size. When we ran the image at all maximum pooling levels, it shrank to zero as we could only feed 64x64 pixels as input.

To avoid this we had to make significant changes to all other networks. Third, Alex Net speed has been improved compared to Google Net, VGGNet and processing time. This mesh is done very often for each image. In a nutshell, Alex Net was an innovative CNN broadcast in 2012; As a result, it is already included in most deep learning frameworks in use today. Setup is child's play and starting training won't take as long as you might think. Now that the structure of the network has been determined, we can move on to collecting training data. It is known that deep learning requires large amounts of training data to successfully arrive at a precise but general answer to avoid overfitting. The purpose of the network is to classify robot images and blank photos into their respective categories.

To ensure that every element of the arena, including the background areas outside the 20m x 20m grid, is correctly identified, the dataset containing the blank photos must accurately represent the various states. The image shown in Figure 3.1 represents a real situation and shows how messy a normal image can be. The image is taken from the dataset in question, which also includes many other photos that are not robots in sight. These images can be easily parsed into more manageable images with 64 x 64 pixel resolution and then used as training data. We will also design it so that there is some overlap between the side-by-side windows so that we can identify the different features we want to learn about in different parts of the extracted image.

It is important to generate this data from photographs that capture a wide variety of possible outcomes, as these results can occur during a live study. As a result, we will select for each possible window the photos that should present the greatest difficulty in classifying them correctly. The blank image portion of the dataset was created from this. The next step is to compile the database of robot photos we have collected. This time we are not taking advantage of a huge database of pre-existing photos that we can sort into hundreds of individual models. Since there is no easy way to systematically remove bot photos from the IARC registry without going through the lengthy manual deletion process, we need to find a more efficient option. One strategy to teach the network the basics of a ground robot would be to take a series of photos of different ground robots and change the camera angles and backgrounds for each photo.

The Ascend NTNU team has access to a collection of ground robots that are great for photography in any setting and in any lighting. Since the network needs to be able to distinguish robots from blank photos, this strategy is a viable method for generating a dataset no matter what surface the robot is on. It also creates high-resolution photos that can be easily adapted to almost any network by resizing, editing, or both. It is estimated that more than 400 photos were taken to capture the different ground robots from every imaginable angle. In technical preparation, we tried to make the dataset larger and more general by blurring, scaling and rotating the photos. This was done in order to create the dataset. The goal was to improve overall system performance without having to buy additional images.

Blurring was found to be the only preprocessing strategy that had such a large impact that it would require further investigation and use. Other methods included various growth and rotation techniques. As a result, the dataset we finally agreed on contained a total of 68,000 blank photos and 1,600 robot images. The relationship between the two may not seem equal; However, we must take into account that blank photos will need to cover a much larger number of potential outcomes than robot images, and will, on average, cover more than 95% of images.

Next, we need to train AlexNet on the dataset we just created so we can assess how well it understands the general visual ideas needed to build ground robots. Prior to joining AlexNet, the best-performing dataset in the expertise study was randomly split into a training set and a validation set. Figure 3.1 shows the result of the experiment. After a single period, it shows a very high test score of 99.97% on the independent validation set. This can be taken as proof that the network is very efficient and extremely fast, but it can also be a cause for concern. Of course, we do not over-edit the photos selected for the validation set, which is generally considered good. The difficulty is that approaching a near-perfect answer so quickly can mean that the data used for training and validation are very similar without necessarily replicating the real-world situation. This is worrisome as it may indicate that the data used for training and validation are very similar.

If so, we have a problem that even if the network doesn't overfit the training data, the dataset as a whole can fit on it. Even if the network doesn't overfit the training data, it can still outrun the dataset. Another topic covered in the tech article was this, and the answer included both the good and the bad. One of its drawbacks was that the robot image was generic enough to include any robot placed right in the middle of the image, but it also had a number of extreme cases where it struggled. AlexNet has demonstrated the ability to learn the visual model of a ground robot extremely quickly, which is a positive aspect. Additionally, AlexNet is expected to be able to improve the categorization of edge cases as long as there are a few edge examples to practice with.

3.16 BOAT BELTS

In terms of object detection, the "bootstrap" approach is a technique that can be used to find and extract edge instances. Several papers use it to describe a system that can detect network-presented errors, accurately label them, and then feed them back to the training set. If we had such a system, it could detect extreme cases that we found difficult to describe, and then use these examples to directly train the network. Once we conclude that the ultimate use of our neural network is an image segmentation algorithm, we can devise a strategy to quickly label large numbers of windows without having to examine them one by one . First, we use a manual process to create what we call a binary ideal segmentation image.

This image ideally represents the results we would like to observe. shows an example of such things. This serves as the ground truth and is used as a benchmark from which network performance is evaluated and compared. We can now associate each classified window with a specific position in the full-size image at exactly the same position in optimum segmentation. The window is then retained as false positives or false negatives and added to the dataset for the next training iteration if the classification performed by the network deviates from the baseline facts. However, to test this bootstrap approach, we first need to develop an appropriate strategy to achieve a floating window. So we'll come back to Bootstrap when we're done developing the Sliding Window.

3.17 CREATING SLIDING WINDOWS

Now that we have a CNN rating bot, we can start applying the floating window technique to the images provided. While AlexNet is the fastest network to achieve the required level of accuracy, it is also not fast enough to work for every pixel of the 1920 by 1080 input image. few pixels. This results in lower resolution of the output image, which is acceptable for most classifications, but can be a problem for some small robots appearing in the image. Choosing an appropriate window size is another challenge using this methodology. There will be various things within the image, each with its own unique size and therefore requiring its own unique window size. More specifically, robots far from the camera are smaller than robots closest to the camera. We consider two different approaches to solving this problem. As part of the specialist work, we tried running the procedure multiple times with different window widths to identify different sized robots.

For the n different window sizes we used, we got n different segmentation images as output. Each of these segmentation images showed where robots could be placed, roughly the same size as the window in the image. The final multiscale segmentation image was created by merging and normalizing n individual photos of the segmentation process. The result of using this strategy is shown in Figure 3.5 below. We can see that this option is able to find the smallest robot in the original image, which is almost completely hidden in the background. On the other hand, it detects a significant number of false positives in the foreground, especially when bots are much larger than the window size. Both identify medium-sized bots, but still give false positives when the window size is too large or too small. usually true, but much less accurate when describing small robots than smaller window widths.

Finally, the merged segmentation is displayed at (g) on the original image, showing which parts of the image the algorithm considers to be robots. Admittedly it manages to mark where the robot is, but it costs a lot of noise. More importantly, a large number of false positives are discovered in the final output, as the different window sizes are greatly disproportionate to what is sought in parts of the image. This is because the different window sizes are disproportionate to what is sought. This strategy gives medium bots the advantage of being discoverable in a variety of window sizes, while smaller and larger bots can only be hosted in one size. As a result, we realized that scale independence would be a very useful feature.

In fact, the classification mesh works significantly better when the windows are roughly the same size as the robot it's trying to describe. It is also important to note that this is an extremely time consuming process as it has to carefully analyze the image for each of the n different window sizes used. A technique using fixed window widths was not reliable enough, as the combination of noise, uneven weight distribution, robot negligence, and time usage issues showed. Using a window size that properly wraps the bots we're looking for anywhere in the image can solve all the problems we've had so far with the solution against the scrollable window. So far we have shown that Alex Net is pretty good at learning to distinguish between ground and background robots, and using a bootstrap method in categorization can be very close to error-free categorization as long as the size of the robots remains relatively constant.

This can be achieved in several ways; one is to use data from the discussed IMU and LIDAR sensors to make assumptions about the diameter of any land robot, given the position of its pixel in the image. . The IMU provides information about the drone's rotation and tilt, which can be used to determine the angle of one of the side cameras. LIDAR provides information about the altitude the drone is currently at. All cameras have the same focal length and pinhole structure and are positioned in a fixed position at an angle of 66 degrees when measured from the ground up. The roll and pitch angles fluctuate very little, as the drone doesn't perform a lot of jerky movements.

This leads to the assumption that photos are taken more or less horizontally to the ground, suggesting that the size of any robot depends solely on its position along the y-axis on Earth. However, we will use rotation and tilt to fine-tune the camera angle, as even minor adjustments in this area will have a significant impact on the final product. Using a combination of these different measurements, we can determine the width of a single pixel in meters for all data and values in the image. You can see this because the value refers to the width of a pixel, measured in meters, in the physical world.

3.18 CLASSIFIER IMPROVEMENT

The fact that the classifier can work with a floating window and generate segmentation images opens the door to a whole new way of measuring success. We can now use a bootstrapping strategy that can discover network-generated errors much faster and more reliably than one could do by manually combining classifications. The main reason for doing this is to filter out classifier-induced errors, as shown in Figure 4.1(d). Being biased. Another reason to do this is to make sure your segmentation limits are correct. The first thing you need to do to complete this procedure is to create a collection of excellent segmentation images like the ones below. . Some won't take much time or effort to make because they're easy to make.

We also produce an image set that is robot-free and contains only all-black images with perfect segmentation. The main purpose of empty frames is to improve the overall performance of CNN against the dynamic background outside the arena. They are not intended to target a specific extreme situation. As you continue reading, it becomes clear that this is necessary to avoid false detections in areas outside the arena. The next step is to perform window scrolling on the original photos while using the CNN classifier, which currently works better to classify each window.

If we compare each window with the corresponding window in the ideal segmentation image, we can now determine whether the classification is correct for each window. This allows us to remove misclassified windows, rename them with appropriate information, and then add them to the dataset used for classification. We hope that adding the extra dataset will help the classifier make fewer background errors and distinguish between bots and backgrounds in Edge instances. If the results of a single iteration of the boot process are not satisfactory, the process can be repeated multiple times using the same batch of photos.

3.19 ROBOT POSITION

Now that the auto-aligning sliding window approach can be used to reliably generate segmentation images, it is now time to position the robots in the environment. This step is required to implement a segmentation-based solution and an end-to-end bot detection network. To avoid disrupting the recognition process, a segmentation solution should complete this step as quickly as possible. This will be an important part of the answer. However, given the benefits of using an end-to-end sensor network as described in it, we will place more emphasis on developing a reliable system rather than fast.

The fact that robot segmentations are more or less simple patches of white pixels that the classifier is very confident detecting a robot adds to the difficulty associated with the robot location task. If these spots represent only one robot, the situation is simple; However, it often happens that most robots stand close to each other and partially block each other's view. The main problem is to remove two robots from these blob types without generating false positives on blobs containing only one robot. Completing this procedure is not just about locating the robot hubs; instead, bounding boxes should be created around each of the bots. This is the typical method of constructing a dataset for one of the most important contemporary sensor networks.

Therefore, it would be crucial to convert midpoints into bounding boxes. However, the only information Ascend needs is the position of each robot's spot on the square at any given time. This indicates that the bounding boxes do not need to match exactly as long as they are accurate enough for the recognition network to determine where they should be placed. As long as the sensing mesh can reliably place the boxes in roughly the same position relative to the center of each robot it detects, we can easily extract and use the exact center point once determined.

3.20 INTERMEDIATE GEAR AND CAM

First, we only want to test the camshaft algorithm to see if it can detect the positions and dimensions of the robot based on the second possible answer we want to explore, i.e. a personalized variation of the offset, meaning a protected window uses a fixed height. width ratio. However, the width is determined for sliding windows the same way as for windows with auto-adjustment. The window is initially placed in a random position, provided it is in an area containing at least one white pixel. When the mean displacement converges, we consider whether the window has robotic sensing and set all pixels in the window to zero. For a window to be considered good, it must contain a certain threshold percentage of white pixels that can only be determined by trial and error. The results of cam shift and custom mean shift approaches and their respective implications are discussed.

3.21 ROBOT SIZE FILTER

Just looking at the shape of the blob, it was difficult even for humans to detect an unknown number of robots hidden in "robot spots". That's our experience so far. One strategy to determine if a blob is too large to hold a single robot is to use our knowledge of its size when the robots are in different positions. The next step is to use large blur filters to achieve significant image blur to highlight local maxima in segmentation images. Blurring an image is often used to reduce noise in the image. however, it can be reused to find the center of the earth robots.

The basic concept for creating spikes in the middle of the blobs is to blur segmentation image areas containing the robots using kernel sizes proportional to the expected size of the robots. Then, these peaks can be extracted as local maximums and considered as potential focal points for ground robots. During initial tests, a simple box filter is used to calculate an average for the near field. The size of the filter is dynamic and changes according to the y coordinate to ensure it always matches the expected robot size. Box filters perform their functions by averaging and averaging evenly over each pixel contained within the frame boundaries of the filter.

Since the size of the filter is proportional to the size of the bots to be used, we expect to get a higher value in the center of the blobs than in the blobs near the edges. In the simple scenario of drops containing only one robot each, this gives us the center of mass, which is most likely the robot's center of gravity. If the blobs are larger than the average robot size, our first plan is to modify the recognition mesh so that it can classify the robots more accurately. We ran into an issue where the classifier was not picking up enough edge samples to be considered during training.

Therefore, we were able to improve the overall segmentation output to resemble the ideal segmentation we wanted to examine more closely. On the other hand, it turned out to be not a good place to start the localization process in the end. The problem is, this results in rather large blobs that leave little or no clue as to where one bot starts and where the next one starts. This is worrisome as it makes it clear that this is the case regardless of the quality of the boundary partitioning. We will retrain the CNN classifier to identify only perfectly centered windows as bots and classify everything else as background. This will help us combat the problem.

The goal is to get smaller robot blobs in the segmentation image, making them easier to distinguish with a filter. Figure 3.6 shows an example of the desired segmentation. To develop a CNN that can sort in this way, we need to collect many more modern examples. These cases represent the dividing line between businesses that qualify as robotic hubs and those that do not. One way to do this is to personally take and attach photographs that we believe adequately represent these conditions. This will happen to some extent; However, most images are created using the built-in staging process that was used in the past to improve classifier accuracy. This is done to improve the accuracy of the classifier.

The only difference between this iteration of Bootstrap and the previous one is that we now only accept a smaller and more centralized set of robot classifications. Therefore, all false positives found in the previous iteration of our boot procedure are still valid, but false negatives should be discarded as they may represent windows that are not in the middle of the robot. Therefore, the first dataset serves as the first dataset, but also contains false positives encountered during the classifier seeding process. Now that the dataset is complete, we can proceed to generate the images for the central segmentation. As we have observed in the past that some segmentation photos can cause significant changes to the classifier, we will generate 10 to 20 images in the hope that this will be enough without requiring excessive manual rendering time. This approach will identify many extreme situations; however, some windows extremely close to the median boundary may be classified differently under the same conditions due to human error in establishing the ground truth.

This is because people are not perfect. This has the potential to shake up the classification network more than it is useful, so we want a statistic that compares the accuracy of a classification to a relatively small area around the central pixel. A threshold can then be applied to the result to only select windows that are obviously incorrect and exclude only those that are incorrect. After introducing Gaussian filters with dynamic kernel size, he explains how to measure the "fix". We hope that by retraining AlexNet on the new dataset, we can achieve a more datacenter-centric classifier and be implemented in a sliding window to more efficiently split large robotic blobs into smaller blobs.

The second change we made has to do with the filter itself: by focusing the classifier more on the center of the robot, we can provide better discrimination between robots, even if their centers are close together. This indicates that larger blobs containing two robots are more likely to occur as midpoints of both robots with a bridge between them than smaller blobs containing a single robot. Figure 3.7(a) shows how this concept is applied in real life, showing that the result no longer appears as a single solid drop, but as two small overlapping drops. This feature can be fully exploited by replacing the box filter with the Gaussian filter seen in . In these special cases, a circular weight distribution is desirable as it tends to hit spherical objects. This indicates that even if two blobs overlap, we can extract each blob separately.

If the Gaussian filter is calibrated to the size of each blob as shown, it will give higher density values towards the centers of the blobs than between them. By changing the sigma value, we can make all the surrounding pixels contribute, not just the pixels in the center of the blob. Likewise, we want the Center to make the most important contribution. The goal is to create a bell-shaped structure in the form of a Gaussian curve in each robot, using local maximums as reference points for the center of the bell. Once this is done, the highs can be recovered. Now that we understand Gaussian functions, we can step back to determine the "correctness" of a classification and see how this idea can be applied at bootstrapping.

The first thing you need to do is take the ideal segmentation image, apply the dynamic Gaussian filter to it, and then normalize the output to be between 0 and 1. When you are done, the value 1 is used to indicate the neighborhood. where each pixel is assigned the role of robot, with a value of zero indicating a completely empty space. This blurs the edges and they take values from zero to one instead of the strict binary format of ideal segmentation. In extreme cases, we can now determine whether a particular extreme scenario is true by comparing it with fuzzy classification.

States near the edge of the segmentation have a value of about 0.5, but states further away have a value close to one or zero, depending on whether the surrounding pixels are robot centers. You can then create a policy that states that there must be a difference of at least 0.9 between their correct classification and their fuzzy classification for samples to be considered misclassified and included in the new dataset. As the barrier is lowered, more extreme states are included. We can also try trying separate criteria for false positives and false negatives if we want to expand or reduce the center margin a bit.

If we use this approach, the result is an output that is also a list of the intensities of each window. This density list can be converted to an image or used in its current form. While all results are displayed as photos, the robot location algorithm uses the density list to avoid further unnecessary processing. Subtraction of local maximums is possible in a variety of different ways. After applying the folded Gaussian filter to the segmentation image, we now turn our attention to a more global level. The candidate robot with the highest intensity value in the filtered image is considered a strong competitor. If the score is found acceptable after verification, it is retained and used as a focal point; otherwise it is discarded. In both cases, the point within a radius equal to the robot's intended radius and all windows around it are devalued.

The next global maximum is then graded using the same methodology, and this process continues until it reaches an intensity threshold that is considered too low to represent any value. The result is a list of midpoints where we can design appropriate bounding boxes since we already know the size of the robot that should be there. The whole procedure is explained in detail while the scoring function and minimization environment windows are discussed in the next sentence. The first thing to look at is how to devalue windows placed around a potential research point. We do this primarily to remove the busiest window from the view, but also to remove surrounding windows that may be nearly as dense as the window we're removing.

If they haven't been removed, it's entirely possible that the first bot provided lots of overlapping detections before any of the others were investigated. Since we're looking for local Maxmas, if we find one, we'll have to narrow the search field for the next one before we can continue our search. One technique to achieve this is to set all windows in range of the intended robot to zero and then exit. If our rating only looked at high intensity levels and accepted anything above a certain threshold, it would pass.

However, this leads to a large number of false positives due to inherent errors in the classifier. These defects allow the classifier to leave unique windows of relatively high value just beyond the beam radius, which are then treated as detections. Another problem when using this strategy is that robots whose centers are close to that of other robots will inadvertently have their radii just beyond the other robot's radius, as the central density of the other robot will be completed, the central density of the other robot will be disabled. The Gaussian drop, which represents the robot whose performance we have just evaluated, is what we have to get rid of here. Therefore, it is a possible approach to reduce all surrounding windows by their respective Gaussian weights. We can see that we can solve the problem by subtracting G(d) from the intensity value, where d is the distance from the candidate point. This should completely delete a single bot's blob without significantly affecting overlapping blobs, which should resolve the issue of deleting centers that are close together.

To overcome the problem of outliers, the use of the scoring function should be made. There is no guarantee that a window will be considered bot detection just because it has a high density rating. This is because smaller groups of false detections may need to be purged without the classifier having difficulty correctly classifying the robots or partially disguising bots. Often this will be an issue when analyzing outliers and clogged robots where the overlapping robot has already been identified and its neighbor windows have been downloaded. The approach checked that the Gaussian distribution for each candidate window was still correct, despite the fact that some of the neighboring windows might have collapsed.

If most of the homes in the neighborhood have already been significantly reduced, it's safe to assume that the home in question is an exception. While this is happening, partially obscured robots can pass the test because they get higher Gaussian scores at their radius than any higher isolated window value, even if their intensity level is reduced. In fact, the Gauss score takes the environment into account. Provides a frame-by-frame representation of an example of the process, showing how blobs are evaluated, saved as detections, and removed from the image. The purpose of this step is to develop a more realistic segmentation to better distinguish and identify partially hidden robots. In the boot procedure, we use the center-oriented segmentation photos as a reference as we think what the final result should be like, but the only result that really matters is the final position of the robot from the segmentation images. Center oriented segmentation images were used. As a result, we will evaluate how well the center-driven strategy works in conjunction with Gaussian filters and use it to derive these positions.

As mentioned in the introduction to this section on positioning the robot, the bounding box need not be exactly at the robot boundary in any direction as long as its center is constantly positioned at the very center of the robot. We know how wide the robots will be , but we don't know how high they will reach. Height is determined by the angle and height of the camera and its relative distance from the robot. It is possible to calculate the height of each robot, but this is not necessary as long as the side bounding boxes are close enough to each other. Therefore, we assume that every bounded box is square and that the height and width are always the same.

This will be enough data that the recognition network can use to generalize and locate the observable features of the robots. Modern sensor networks can distinguish between a large number of different objects, but in this scenario they only need to identify one. Therefore, we can use training data that is not 100% accurate in terms of the bounding box location without affecting the final output. There should be some indication that this assumption leads to a loss of accuracy, but adjusting the altitude later is not a problem and can be easily done.

3.22 NETWORK TRAINING

Now that we have some sensor data, we can begin the long-awaited networking process. SSD Network is a highly complete system that integrates all the necessary components into the source code. To test the network, all you have to do is translate the dataset into an understandable format, adjust the chunk size and learning speed according to the available memory of the GPU you are using, and then start training. The monitoring dataset is divided into a training set and a validation set. The training set will contain data from five runs performed during IARC 2016 and the validation set will include the sixth and final run.

To the untrained eye, each of the marks looks quite similar; However, each has a unique bot placement and flight pattern setup. If the validation set is generated by randomly selecting frames from the entire data set, we get two sets that are essentially the same in terms of the information they contain, with only minor differences between each frame. Since the validation set only contains images of a run that the network has never seen before, we can get a much better idea of whether the training process is robust by dividing the two into separate runs. This allows us to better assess whether the training process is over-adapted.

The generalization of the training procedure is controlled by a large number of preprocessing parameters included in the SSD code. We won't be changing any of them from the first iteration of the tutorial, because the SSD document explains pretty well why they were set up that way, so you don't need to change them. After the first test cycle is complete, the remaining variables are examined to see if they can be adjusted for a better result. A number of changes can be made to the overall topology of the network to change not only the training efficiency but also the speed/accuracy ratio. We can make the network faster by reducing the input size, which results in less accurate detection, or we can increase the input size, resulting in slower but more accurate detection.

To get things going, you'll need to run some tests on the Jetson TX2 platform first to evaluate how well the two storage capacities suggested in the SSD article perform. Therefore, the first two networks to be formed will be 300x300 and 512x512, respectively. We designed the entry-level 420x420 version to find a solution between SSD300 and SSD512. This would certainly have accurately represented the results.

3.23 ADD COLOR

Now that we have a well-functioning network, we can focus on developing their capabilities. More specifically, we want you to understand the differences between the three categories of robots participating in the tournament. While there is currently no discernible difference between green and red robots, this distinction will be necessary for future operations. Locating the robots to which the rods are attached is always vital to ensure that the drone does not collide with the robots during flight. The only major change that needs to be made to the dataset is to replace the simple expression with a robot with the exact color that matches each detection. As a result, we now have a good understanding of categorization networks and the level of certainty with which they can operate.

Therefore, the recommended strategy is to train a classifier to distinguish different colors and then run it for each predefined robot to label it with colors. The classifier can be built by taking the sizable dataset of robot clippings currently available, sorting them by color, and then teaching an AlexNet classifier how to identify different robots. This time, the classifier does not need to recognize the background, because it already knows that it will only take pictures of the robots as input. After the dataset is recolored with the new colors, all that remains is to re-adapt and recycle the number of possible output detections to the detection network. That's all it takes.

3.24 NETWORK TEST

Once the meshes are trained, the meshes can be placed on Jetson boards for evaluation. There is already pretty good data on what to expect in this particular area, as the accuracy of the network has already been verified in the training phase using the validation set. With this in mind, it is difficult to accurately determine network performance based on mAP score alone. Thus, a more visual representation of the data is created by highlighting the network while validation is in progress.

This chapter is about probability theory and information theory. The mathematical basis for expressing propositions with uncertainty is known as probability theory. It provides a method for measuring uncertainty and axioms for deriving new statements that are subject to uncertainty. The probability theory that we use in the field of artificial intelligence has two main applications. First, probability principles tell us how AI systems should think, and then we create our own algorithms to calculate or approximate various expressions constructed using probability theory. These expressions are then used in our systems. Second, using probabilities and statistics, we can make a theoretical analysis of the performance of proposed AI systems. The study of probability is an integral part of various scientific and engineering disciplines.

We have included this section so that readers with a primary background in software engineering and little or no exposure to probability theory can still understand the information presented in this book. Information theory gives us the ability to measure uncertainty in a probability distribution, unlike probability theory, which gives us the ability to make explanations about uncertainty and its causes. If you already have a good working knowledge of probability and information theory, you can skip the rest of this section and go directly to this section if you already have this basics. This section aims to enable you to carry out deep learning research projects even if you have no prior knowledge of the above topics; However, we recommend that you find more information in an additional reference book such as Jaynes' book. (2003).

Many areas of computer science focus primarily on what can be described as strictly deterministic and explicit. A programmer can often assume that a central processing unit (CPU) will correctly execute every instruction of the machine. Hardware can have bugs, but these are so rare that the vast majority of software doesn't need to be developed to fix them. Considering the environment in which many computer scientists and software developers work is fairly organized and consistent, some people may be surprised that machine learning relies so heavily on probability theory. Because machine learning is constantly dealing with unknown numbers, sometimes with stochastic (non-deterministic) values.

A situation has many potential sources of uncertainty and stochasticism. Since at least the 1980s, researchers have made compelling arguments for measuring uncertainty using probabilities. Pearl is the source of many of the arguments put forward, or the inspiration for many. (1988). Arguing in the face of uncertainty is a necessary skill in almost any job imaginable. In fact, it is impossible to imagine a statement that is absolutely true or an event that is absolutely certain that it will happen. This is because mathematical expressions are true by definition; yet it is difficult to come up with a completely correct idea.

Even if the actual rule is predictable and our modeling system is sensitive to processing a complex rule, it is often more practical to use a simple rule with an uncertain outcome rather than a complex rule with uncertain outcome. . This is the case even if the underlying rule is simple. A flightless bird such as the cassowary, ostrich, and kiwi cannot fly, but this is not the case for very young birds that have not yet learned to fly, or sick or injured birds that have lost flight. fly. Flightless bird species such as flies and those mentioned above. "Birds fly, except very young birds that have not yet learned to fly, sick or injured, flightless birds that have lost their ability to fly." While it is obvious that we need a way to express ourselves when we think about uncertainty, it is clear that we need a way, although it is not immediately clear that probability theory can provide all the tools we need for AI applications. to represent and justify uncertainty. Probability theory was originally developed as a tool to identify patterns in how events occur.

It is easy to see how probability theory can be applied to the study of events, for example B. drawing a particular hand in a poker game. Such events will yield the desired result p percent of those times, even if we want to run an experiment (for example, drawing a hand card) an infinite number of times. This is what we mean when we say that a particular outcome occurs with probability p. This reasoning does not seem to apply directly to non-repeatable statements. When a doctor examines a patient and finds that they have a 40% chance of having the flu, it means something else entirely.

This means that we cannot generate an infinite number of patient replicates, and there is no reason to assume that different patient replicates exhibit the same symptoms while having different underlying conditions. Also, you cannot create an infinite number of patient copies. We use the concept of probability to describe the level of confidence we have in the context of the doctor's diagnosis of the patient. A probability of 1 indicates that the patient does not suspect the flu, and a probability of 0 indicates that the patient does not suspect that he has the flu.

The first type of probability, known as frequentist probability, is directly related to how often events occur, while the second type of probability, known as Bayesian probability, is related to the level of certainty we have in a proposition. In listing the various properties that we believe uncertainty thinking should have, the only way to provide these properties is to consider Bayesian probabilities to behave exactly like Frequentist probabilities.

For example, if we want to calculate the probability that a player will win a game of poker because he owns a certain deck of cards, we use exactly the same formulas that we use when we want to calculate the probability that a patient will become ill because he has a disease. some symptoms. In both cases, we want to determine the probability that the patient has the disease in question. For more explanation on why a simple collection of common sense assumptions implies that the same axioms should govern both forms of probability, see Ramsey's work. (1926).

The concept of probability can be seen as an extension of the logic used to deal with uncertain situations. Logic provides a set of formal rules for deciding which statements are true or false, assuming another set of statements is true or false. These rules can be used to determine which statements are assumed to be true or false. Probability theory provides a formal set of rules for judging the probability that a statement is true given the probability of other statements.

3.25 RANDOM VARIABLES

A variable that can take any new value randomly is called a random variable. In most cases, we use lowercase letters in regular font to represent the random variable itself, and lowercase letters to denote the many values it can have. For example, the numbers x1 and x2 are examples of possible outcomes that the random variable x can have. For vector-valued variables, write the random variable x and one of its values as x. A random variable is nothing more than a description of the states it can reach; To be useful, it must be combined with a probability distribution that gives the probability of each of these situations occurring. Discrete and continuous random variable formats are possible. A discrete random variable is a variable that can have only one of a fixed number of possible states at a time. It is important to note that these states are not always integers; Instead, they may be called cases where no numerical meaning is added. A continuous random variable always has a true value associated with it.

If the covariance has large absolute values, this indicates that the values vary widely and also differ significantly from the individual means of both. A positive sign of the covariance then indicates that the two variables tend to have relatively high values at the same time. A negative sign of covariance indicates that one variable tends to have a relatively high value and the other variable tends to have a relatively low value, and vice versa. Other sizes, eg. B. correlation normalizes the contribution of each variable to simply measure how correlated the variables are, rather than being more influenced by the scale of each variable.

This allows correlation to measure how closely the variables are related. While somewhat related, the concepts of covariance and dependency can be viewed as separate mental constructs. They are related because the covariance between the two independent variables is zero, but the covariance between the two dependent variables is not zero. However, independence can be compared as a different quality from covariance. For two variables to have zero covariance, there can be no linear dependence between the variables. Independent constraints are more stringent than zero covariance constraints because independent constraints exclude the possibility of nonlinear interaction. It is conceivable that the two variables depend on each other without covariance.

For example, consider the scenario where we first plot the real number x from a fixed distribution in the interval [1, 1]. Then we sample a random variable. Both the Bernoulli distribution and the multinomial distribution are suitable for describing any distribution in their domain. They cannot describe every distribution in their field because their fields are simple as much as they are particularly powerful; represent discrete variables for which it is appropriate to list all states. This way, they can identify any distribution in their domain. The possible situations are endless when working with continuous variables; Therefore, any distribution that can be defined with a limited number of parameters must impose strict constraints on the distribution it defines.

First of all, most of the distributions we are interested in modeling are quite similar to the normal distribution. The central limit theorem shows that the sum of a large number of independent random variables is nearly normally distributed with all probabilities. This indicates that many complex systems can actually be effectively described as normally distributed noise, but the system can be decomposed into parts with more organized behavior. This is true even if the system itself can be treated as normally dispersed noise. Second, the normal distribution encodes the largest degree of uncertainty about real numbers among all potential probability distributions with the same variance. In fact, the normal distribution is the most common. The normal distribution is the distribution that includes the least amount of prior information in a model, so we can consider it the distribution that does the best. The development and justification of this concept should be deferred to the next section, as more mathematical tools are needed.

For the purposes of our research, measurement theory is very useful for expressing theorems in Rn that are valid for most points but invalid for some vertex examples. Measurement theory offers a methodical approach to expressing that a collection of points is unimportant. It would seem that the measure of such a collection is zero. We do not give an official definition of this term in this particular guide. For the purpose of our research, it is necessary to understand the concept that an empty set of empty measurements does not occupy any volume in the measured area. For example, the measurement of a line through R2 is always zero, while the measurement of a complete polygon is always positive.

Similarly, the measure of a single point is always 0. Any union of a countable number of sets, all of which have a measure of zero, also has a measure of zero for each measure. (For example, the measure of the entire set of rational numbers is zero). Another practical concept derived from measure theory is practically ubiquitous. A property that applies almost everywhere is valid for all space except for a set with a measure of zero. Because exceptions take up relatively little space, it's perfectly acceptable to ignore them for many applications. Some important discoveries in the field of probability theory can be applied to all discrete values, while "almost everywhere" can only be applied to continuous values.

3.26 INFORMATION THEORY

Quantifying the amount of information contained in a signal is central to the computer science subfield of applied mathematics, which can be divided into several subfields. When it was designed, its original purpose was to examine the problem of transmitting messages of different alphabets over a noisy channel. Messages from certain probability distributions are calculated. It also explains how information is decoded from random sequences. We can also apply information theory to continuous variables in machine learning; However, some of these message length comments do not apply in this context. This field is essential for studying various subfields of electrical engineering and computer science.

When we try to describe probability distributions or measure similarities between different probability distributions, we rely mainly on a small number of key terms from information theory. For more discussion on information theory, see Cover and Thomas (2006) or MacKay. (2003). The idea that knowing that an unusual event has occurred is more informative than knowing that a probable event has occurred is the basic idea behind information theory. A message saying "The sun has risen this morning" is so informative that it would be pointless to post it, while a message saying "There was a solar eclipse this morning" is quite informative.

These factorizations have the potential to greatly reduce the set of parameters required to adequately represent the distribution. The total number of parameters used by each factor is proportional to the total number of variables used by that factor. This indicates that we can significantly reduce the cost of expressing a distribution if we can find factorization in distributions with fewer variables. Using graphs, we can explain many types of factorization. In this context, the term "graph" means a set of vertices that can be related to each other via edges.

This definition comes from the field of graph theory. When we use a graph to show the factorization of a probability distribution, we often refer to a structured probability model called a graphical model. Directed and undirected structured probabilistic models are the two main types of this model type. Each node of the G graph corresponds to a random variable, and an arc connecting two random variables means that the probability distribution can represent direct interactions between these two random variables. Both types of chart models use a G chart. Neither type of chart model is exclusive to the other.

CHAPTER 4

NUMERICAL COMPUTATION

This section presents the results of many detailed processes in creating a robot identification and segmentation dataset from scratch using baseline photographs and information about the height and angle at which they were taken. These results are presented in this section. Before analyzing the results of different robot localization strategies, it is necessary to go through the process of generating segmentation images using the sliding window method.

Next, we will discuss the results of training and testing performed on the various network configurations described. This largely depends on the evaluation of the visual output and the visual output itself. This section will therefore include photos and illustrations to convey the result, as well as links to videos that can provide a more complete overview of the behavior of different sensing networks. These items will be included as they will be included in this section. Time is given for further discussion of the results.

4.1 ROBOTIC SEGMENTATION

Figure 4.1(c) and Figure 4.1(d), respectively, show two different methods of creating scrollable windows: one uses fixed-size windows and the other uses variable-size windows. Charts have been created to allow a visual comparison of different strategies, taking into account the possible advantages and disadvantages of each. If we evaluate the outputs using perfect segmentation, we get a percentage of 96.98% for fixed size and 98.17% for auto-adjustment. Despite our performance improvements, we still tend to err in identifying white bars and other irrelevant objects located outside the arena. B.

The technique used to develop the classifier discussed in Figure 1 was initially focused on eliminating underlying classification errors. The preparation process was done in stages and each stage resulted in further progress. The final result can be seen here and it took a total of seven laps, the last two using some extra photos suitable for segmentation. The final result was a 99.13% correlation between effective segmentation and ideal segmentation.

4.2 DETECTING ROBOTS

The results of the different localization strategies previously proposed in Section 3.3.2 are discussed in this section of the report. In contrast to the typical grayscale representation of segmentation photos used up to this point in this section, we will use a heatmap instead.

4.3 LAST FINDING

Dataset The dataset was created using the sliding window segmentation technique on 7100 photos, and the results, when applied to the dataset, correctly recognized all robots and positioned them 83% of the time. The detector often had issues with very blurry photos or errors in the height and angle metadata, resulting in a miscalculation of the robot's dimensions. Therefore, the final dataset had to be manually reviewed to remove the erroneous photos. This was undesirable, but all it took to complete the process was to answer yes or no to each photo whether or not it was taken. Even some photos with very minor errors were edited and added to the collection.

The completed project resulted in a recognition dataset of 6100 photos complete with bounding boxes for each robot. The next step was to apply the paint. After training, the AlexNet classifier achieved a validation score of 99.4%, which proved accurate enough to be implemented without requiring human supervision. The completed project resulted in a revised version of the dataset containing the tower, green and red robot labels as separate data point categories.

4.4 SENSING NETWORK

After spending a lot of time and effort building a robot identification dataset, we can finally examine how an end-to-end recognition network can be generalized to the dataset. The network was built using a rugged computer equipped with the latest and most advanced Nvidia Titan X graphics. Compared to most other desktop computers, the 12 gigabytes of storage allowed for a relatively large chunk size and 11 teraflops of processing power allowed for much more. better training times . The first network we wanted to work on was a 300x300 SSD, and the smaller VGGNet would serve as a model for feature mining. The number of iterations in the process was 10,000 and the batch size was set to 32.

The learning rate started at 0.001 and dropped to 0.0001 after 10,000 iterations. None of the other values in the SSD code have been changed, as the combination that seems to have worked well for the same developers as the SSD model was not changed. This was done to prevent tampering with the combination. Figure 4.8 shows the results of running the model for a total of 16,000 iterations, a process that took 43 hours and 14 minutes. As the validation set accuracy increases until it stabilizes with a mAP of 83.78 over the last few iterations, the graph shows that the network can learn without much aliasing during training. Indeed, the accuracy of the validation set increases.

The same was done on a network with an input size of 512x512. Since the graphics card only had enough memory, the heap size had to be reduced to only 8 and the initial learning rate was 104. Figure 4.9 shows the results of a total of 32,000 model runs to get the results shown in this figure. The total time required for the training is 24 hours and 12 minutes. However, as can be seen from the training graph, the network manages to learn without being overloaded throughout the process, resulting in an overall average performance of 90.55. This is in contrast to the 300 x 300 network, whose validation results increase more sporadically.

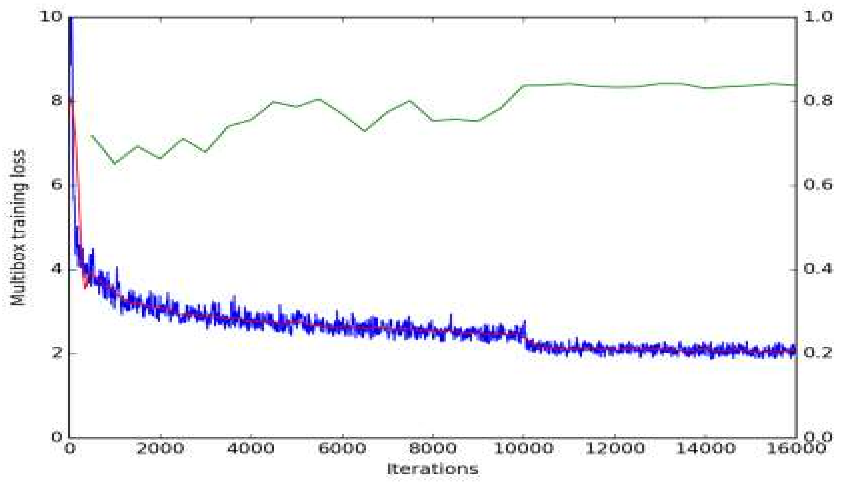


Figure 4.1: Training plot of an SSD array with input image size set to 300x300.

Often times, the algorithms that make up machine learning require a significant amount of numerical computation. Rather than analytically generating a formula that provides a symbolic expression for the appropriate response, it often refers to algorithms that solve mathematical problems through techniques that update solution estimates through an iterative process. This is in contrast to traditional algorithms that solve mathematical problems analytically by deriving a formula.

Examples of common operations are optimization, where the value of an argument is determined to minimize or maximize the effect of a function, and operations such as solving systems of linear equations. Even the simplest task, such as evaluating a mathematical function on a digital computer, can be difficult when that function contains real numbers. This is because real numbers cannot be accurately represented by available memory.

4.5 TOO MUCH AND LITTLE FILLED

The main difficulty in performing continuous mathematical operations on a digital computer is that we have to represent an infinite number of real values using only a finite number of bit patterns. When we represent a number on the computer, we almost always make mistakes when estimating the number. This means that almost all real numbers are subject to this problem. In most cases it's just a rounding error. Rounding errors are especially problematic when they accumulate over a large number of transactions. This can cause algorithms to actually fail, even if they work well in theory, when the algorithms are not designed to limit the accumulation of rounding errors.

Overflow is a type of rounding error that can have particularly fatal consequences. An underflow can occur when numbers very close to zero are rounded to zero. When a function's argument is zero instead of a small positive integer, the function sometimes behaves radically differently. For example, we should avoid operations such as division by zero as much as possible and take the logarithm of zero. When the first happens, some software environments may throw exceptions, while the second produces a result with a non-numeric value. (This often becomes non-number when used for many other arithmetic operations).

The overflow error is another type of very harmful numerical error. An overflow phenomenon occurs when adding very large numbers. Subsequent arithmetic usually converts these infinite values to non-numeric values. The Softmax function is an example of a function that must be balanced to protect against underflow and overflow.

The softmax function is often used when trying to estimate the probabilities associated with a multiple zero distribution. The definition of the softmax function is as follows:

There is still a small problem to be solved. Even if the denominator has an overflow, an expression can still be treated as zero if it is below the numerator. This shows that if we apply log softmax(x) by first executing the softmax subroutine and then passing the result to the log function, we may make an error and get an incorrect result. Instead, we need to create a standalone function that can calculate the log softmax in a way that is not subject to numerical instability. The approach we used to stabilize the softmax function can also be used to stabilize the log softmax function.

Often, we do not go into detail about all the numerical considerations required to successfully implement the various algorithms described in this book. Low-level library developers should keep numerical challenges in mind when it comes to implementing deep learning algorithms. The vast majority of readers of this book do not hesitate to rely on low-level libraries that provide reliable implementations. It is possible to apply a new algorithm and then automatically execute the new application under certain conditions.

4.6 BAD PACKAGING

The degree to which a function quickly adapts to even small changes in the values of its inputs is called conditioning. Problematic functions for scientific computing include functions that change rapidly when their inputs change very little. This is because even the smallest rounding errors in the inputs can cause large changes in the output of the function. It is the ratio of the size of the largest eigenvalue to the smallest eigenvalue.

When this set is large, the sensitivity to input inaccuracies increases significantly during matrix inversion. This precision is not the product of a rounding error during matrix inversion; When we multiply by the inverse of the correct matrix, ill-conditioned matrices magnify already existing errors. In reality, the inaccuracy is further compounded by numerical inaccuracies that occur during the investment process.

By examining the Hessian eigenvalues, we can determine whether the turning point is a local maximum, local minimum, or saddle point. If the Hessian is positive definite, the point is considered a local minimum, that is, all its eigenvalues are positive. This can be demonstrated by directly stating that the second derivative must have a positive value in all directions and referring to the univariate test of the second derivative.

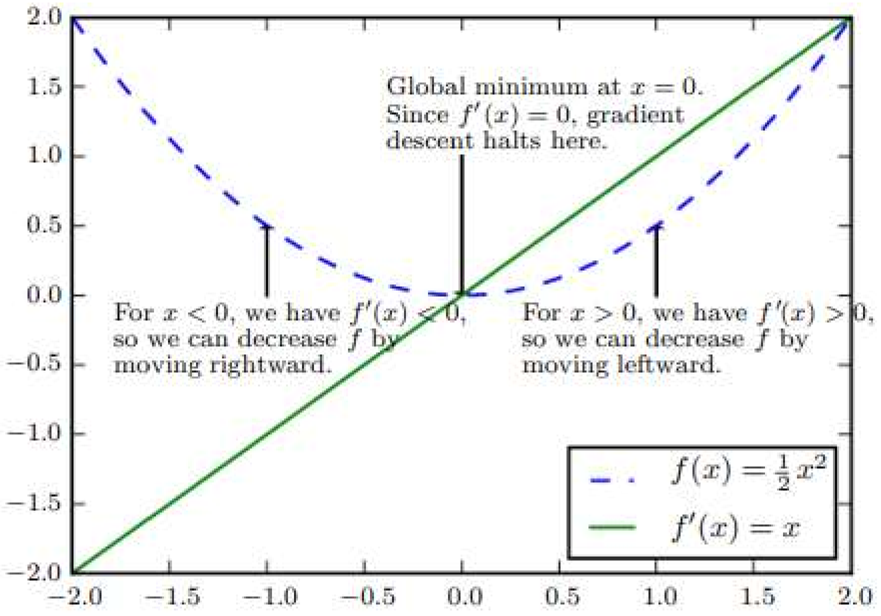


Figure 4.2 An illustration of how the gradient descent algorithm uses derivatives of a function can be used to plot the descent of the function to a minimum.

Similarly, a point is considered a local maximum if its Hessian is negative definite, that is, all its eigenvalues are negative. In multidimensional cases, finding conclusive evidence of saddle points may be considered, but is not always guaranteed. If at least one of the eigenvalues is positive and at least one of the eigenvalues is negative, we know that x is a local maximum in one section of f and a local minimum in another section. For a picture, see Figure 4.2. Consequently, the multivariate second derivative test, like its univariate counterpart, has the potential to produce inaccurate results. If all non-zero eigenvalues have the same sign, but there is still at least one eigenvalue that is zero, the test results are inconclusive.

This is because the results of the univariate second derivative test are uncertain in the cross-section corresponding to zero eigenvalue. In multidimensional spaces, a single position can have only one second derivative in each direction. At this point, the Hessian condition number gives a measure of how different the second derivatives are from each other. Gradient descent does not work well if the Hessian condition number is low. This is because the derivative increases rapidly in one direction and slowly in the other.

Since the gradient descent is not aware of this shift in the derivative, it does not understand that the derivative must be positive in the direction it remains negative for a long time. Also, choosing an appropriate step size is difficult. The step size should be small enough to avoid climbing in directions of high positive curvature and exceeding the minimum required step size. In most cases, this indicates that the step size is insufficient to make significant progress in other directions with less curvature. For a picture, see Figure 4.2. A solution to this problem can be found using the information in the Hessian matrix.

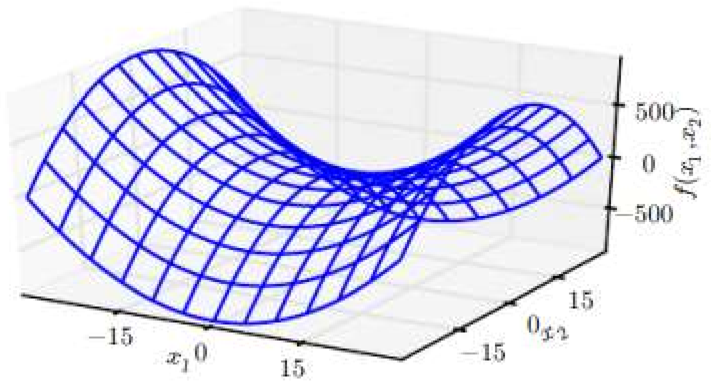


Figure 4.3: A saddle point with both positive and negative curvature.

The gradient descent algorithm does not correctly use the curvature information contained in the Hessian matrix. In this case, we minimize a quadratic function f(x) using gradient descent where condition 5 is present in the Hessian matrix. This indicates that the direction with the greatest curvature contains five times the curvature in the direction with the least curvature. In this particular case, the direction has the greatest curvature while the direction has the least curvature. The path of the steep descent is marked with red lines.

This very extended quadratic function looks like a recess extending into the thousands. Since the canyon walls have the steepest slope, going down all the time is time consuming. Because the step size is a bit too large, it tends to go beyond the bottom of the element, so the next iteration should include the opposite canyon wall. Since the large positive eigenvalue corresponding to the Hessian eigenvector pointing in that direction indicates that this directional derivative is increasing rapidly, an optimization algorithm based on Hessian can predict that the steeper direction is not a promising search direction when applied to it. custom Scenario direction.

If f is a positively definite quadratic function, Newton's approach is to do it only once to jump directly to the minimum value of the function. If f is not really quadratic but can be estimated locally as positively definite quadratic, then Newton's approach is to repeatedly apply the critical point; this can be achieved much faster by updating the approach iteratively and jumping to the minimum of the approach instead of using the dropdown list. would be an alternative. This function is useful near a local minimum, but can be detrimental near a saddle point. While Newton's approximation is applicable only when the nearby critical point is minimal (all Hessian eigenvalues are positive), as discussed, the gradient descent method does not unsaddle the point unless the gradient is going towards it.

First-order optimization algorithms are those that only use gradients in their processes, eg B. gradient descent algorithm. Second-order optimization methods are those that also use the Hessian matrix. Some examples of quadratic optimization techniques are Newton's method. (Nocedal and Wright, 2006). The optimization techniques used in most of the scenarios presented in this book can be applied to a wide variety of functions, but offer almost no guarantees. Understanding the family of functions used in deep learning is notoriously difficult, so deep learning algorithms often make no guarantees. Creating optimization algorithms for a selected set of functional families is the most common type of optimization applied in many different research areas.

This feature is valuable as it allows us to quantify our hypothesis that a small change in input produced by an algorithm such as gradient descent will have a small change in output. This feature is useful as it allows us to quantify our hypothesis that there will be little change in the output of the gradient descent. The Lipschitz continuum is also a very weak constraint, and many optimization problems in deep learning can be made Lipschitz continuum with relatively little tuning. In fact, the Lipschitz continuum is a constraint that can be met with quite a number of solutions. Convex optimization is generally considered one of the most productive subfields of optimization science as a whole. Convex optimization algorithms can give their users much more certainty by applying stricter constraints.

Methods that can only be applied to convex functions, i.e. functions where Hessian is positive semi-definite everywhere, can be considered convex optimization algorithms. Such functions perform well because there are no saddle points and all their local minimums are necessarily global minimums. However, it can be difficult to articulate the deepest learning challenges in terms of convex optimization. Several deep learning algorithms use convex optimization as a subroutine. To determine the convergence of deep learning algorithms, concepts from the study of convex optimization algorithms may be relevant. However, in the context of deep learning, the importance of convex optimization in general is greatly diminished. Boyd and Vandenberghe (2004) or Rockafellar are two sources you can refer to for more details on convex optimization. (1997).

4.7 LIMITED OPTIMIZATION

We often have a desire to find a solution that is a little more compact. When faced with such problems, a common strategy is to apply a norm constraint such as "||x|| 1". To perform the constrained optimization with a simple method, it is sufficient to change the gradient descent algorithm to fit the constraint. We are able to build uphill steps with a small fixed step size. After that we can project the result onto S. If we use a line search, we have the option of finding other step sizes that generate new allowed x points, or projecting each point of the line into the constraint region.

Both of these options are available to us when we use a line by line search. This procedure can be made more efficient by projecting the gradient onto the tangential area of the vital region before performing the transition or initiating the line search, before performing the transition or initiating the line search. (Rosen, 1960). Constructing a new unconstrained optimization problem, whose answer can initially be transformed into a solution to the constrained optimization problem, is considered a complex technique. Take, for example, the scenario where we want to reduce it.

CHAPTER 5

MACHINE LEARNING BASICS

A subcategory of machine learning is called "deep learning". To understand deep learning well, you need to have a good understanding of the fundamentals of machine learning. This chapter provides a crash course in key concepts used in later chapters and sections of the book. Readers new to the subject or looking to gain a broader perspective are strongly encouraged to seriously consider machine learning textbooks that offer an in-depth look at the principles of the subject, for example: B. Murphy (2012) or Bishop. (2006). If you already know the basics of machine learning, feel free to skip it.

This section discusses various perspectives on classical machine learning approaches that have had a significant impact on the development of deep learning algorithms. We'll start by explaining what a learning algorithm is and then move on to an example with a linear regression method. Next, we will show in more detail how the difficulty of fitting training data differs from the difficulty of discovering patterns that can be generalized to new data. Most machine learning algorithms use parameters called hyperparameters. These hyperparameters should be chosen independently of the learning algorithm itself. Here we look at how to set these parameters using additional data.

Machine learning is basically a form of applied statistics that puts more emphasis on using computers to statistically predict complex functions and less on showing confidence intervals around those functions. Accordingly, we will present two main approaches to statistics known as Frequentist estimators and Bayesian inference. The vast majority of machine learning algorithms fall into one of two categories: supervised learning and unsupervised learning. In this article, we will discuss these two categories and give some examples of the key learning algorithms that go into each. The stochastic gradient descent optimization technique forms the basis of the vast majority of deep learning algorithms.

In this section, we describe the process of creating a machine learning algorithm by combining many different types of algorithms, including an optimization algorithm, a cost function, a model, and a dataset. In the final section, we discuss some of the variables that contribute to the limited generalizability of the traditional machine learning framework. These issues have accelerated the development of deep learning algorithms that can overcome these challenges.

5.1 LEARNING ALGORITHMS

An algorithm that can learn from experience is called a machine learning algorithm. But what are we actually talking about when we talk about learning? "A computer program is said to learn from experience E of a class of tasks (T) and a measure of performance (P) if its performance on tasks at T measured with respect to P improves with E experience." This definition comes from Mitchell (1997). A wide range of experiments E, tasks T, and performance indicators P can be designed, and in this book we do not seek to provide a formal definition of what can be used for each of these entities. Instead, we focus on providing a framework for analyzing the relationships between them. Instead, the following sections provide easy-to-understand explanations and examples of the many types of tasks, performance measures, and experiments that can be used to design machine learning algorithms.

These explanations and examples are presented in the order in which they are presented in the following sections. Machine learning allows us to solve unsolvable problems using traditional, man-made, pre-designed, ready-to-use computer programs. Machine learning is fascinating both scientifically and philosophically because mastering machine learning requires understanding the fundamental principles that underpin intelligence. This makes machine learning an interesting topic. The act of learning itself is not seen as a task in this more technical sense of the term "task". Learning is the tool that enables us to acquire the skill of doing business.

For example, if we want a robot to be able to walk, we have to walk. We can teach the robot to walk by programming it to walk on its own, or we can try to develop software that directly teaches it to walk by hand. Machine learning tasks are usually specified by how a sample should be processed by the machine learning system. An example of this would be an item to be analyzed by the machine learning system, or a set of features that are objectively measured and extracted from an event. In most cases we will express an example using a vector of the form xRn, where each xi element of the vector represents a different feature. For example, the pixel values that make up an image are often considered one of its properties.

Classification becomes more difficult if not every measurement is guaranteed to always output in the input vector of the computer program. For the learning algorithm to successfully complete the classification task, it only needs to create a single function that maps a vector input to a categorical output. Rather than specifying a single classification function, the learning algorithm must learn several functions to handle situations where some input may be missing. Each function conforms to the categorization of x, ignoring a different subset of its inputs. Situations like these are very common in the medical diagnostic field, as there are many different types of medical tests, from expensive to intrusive.

Learn a probability distribution over all relevant variables and then solve the classification problem by marginalizing the missing variables. This is a technique used to efficiently construct such a collection of functions. We only need to learn a function that describes the joint probability distribution, but since we have n input variables, we can now learn the 2n alternative classification functions required for any conceivable collection of missing inputs. Goodfellow et al. (2013b) for an example of a complex probabilistic model used in this way to overcome this challenge. Classification with missing inputs is only one of the things machine learning can do; Many of the other tasks described in this section can also be modified to work with missing entries.

In this type of job, the machine learning system needs to look at a highly unstructured representation of some kind of data and convert it into a discrete text format. In other words, the machine learning system must learn to read. For example, in optical character recognition, a picture with an image of text is shown to computer software and is then responsible for returning the text as a string of characters to be added to the picture. (eg in ASCII or Unicode format). To analyze address numbers in this way, Google Street View uses deep learning. (Goodfellow et al., 2014d). Another example would be speech recognition, where computer software takes an audio waveform and then, after the waveform sound wave is received, generates a set of characters or word identification codes that identify words spoken in an audio recording. Deep learning is an integral part of today's advanced speech recognition systems used by large companies such as Microsoft, IBM and Google.

Any job that has an output that is a vector (or other data structure with many values) that has meaningful connections between various components is called structured output. This is a huge area, involving not only the transcription and translation tasks discussed above, but also a large amount of additional work. Analysis is an example; You can map a sentence written in natural language to a tree that defines the grammatical structure of the sentence, and the nodes in the tree can be used as verbs, nouns, adverbs, etc. includes labeling to indicate whether For a demonstration of how deep learning can be applied to the analytics process, see Collobert (2011).

Another example of this concept is the pixel-by-pixel partitioning of photographs, where computer software divides an image into separate categories for each pixel. For example, one application of deep learning is to annotate aerial photographs to indicate the location of roads. (Minh and Hinton, 2010). For jobs that use an annotation-style approach, the shape of the output need not exactly match the structure of the input. For example, when used for captioning, computer software looks at an image and creates a natural language sentence describing what it sees in the image. In this type of association, computer software examines a group of events or objects and marks a subset of them as extraordinary or unusual. Identifying fraudulent credit card charges is an example of anomaly detection activity.

A credit card issuer can detect fraudulent use of your cards by analyzing your typical shopping behavior. When a thief gets your credit card or credit card information, the criminal's actions often come from a different probability distribution among purchase types than yours. If a credit card is used for a transaction that does not match that card, the credit card company may take anti-fraud measures by blocking the account. For an overview of the many anomaly detection techniques, see Chandola et al. (2009).

Given this job, the machine learning algorithm is responsible for generating new samples similar to those found in the training data. In multimedia applications where it can be time-consuming and expensive for an artist to manually develop large amounts of information, machine learning-enabled synchronization and sampling can be useful tools. For example, video games can automatically create textures for solid objects or landscapes, so an artist doesn't have to manually mark every pixel in the process. (Luo et al., 2013). Given the input parameters, it is desirable that the sampling or synthesis process produces a certain output format under certain conditions.

For example, for text-to-speech purposes, we first give the software a written sentence and then ask it to generate an audio waveform containing a spoken version of the text. This is a structured output job format, but with the added caveat that there is no single suitable output for every input and we specifically want a wide variety of output variations to make the output more natural and authentic. Therefore, it is a kind of structured exit task with additional attributes.

5.2 POWER MEASUREMENT, P.

To ensure an accurate evaluation of a machine learning algorithm, a quantitative method must be developed to measure program effectiveness. In most cases, this particular performance metric P is specific to the task T performed by the system. When performing tasks such as transcription, classification, and classification with missing inputs, it is common to evaluate model performance for accuracy. When we talk about accuracy, we mean the percentage of distinct instances where the model produces correct output. Yes. In most cases, we are concerned with how the machine learning algorithm behaves with previously unseen data.

This really shows how well it works when used in the real world. Therefore, we evaluate these counters using a separate set of test data from the data used to train the machine learning system. Choosing a performance metric may seem easy and objective, but in reality it is sometimes difficult to choose a performance metric that matches the desired system behavior. In some cases this is the case because it can be difficult to determine what to evaluate.

For example, when doing a job involving transcription, should we evaluate the system's accuracy when it comes to transcribing entire sequences, or should we use a more detailed performance metric that gives partial credit for getting selected portions of the correct sequence offers? If the system regularly makes medium-sized errors or rarely makes large-scale errors, should we be more strict with the system when doing the regression study? The application determines suitable options for this type of design. In other cases, we ideally know the amount we want to measure, but we cannot measure it because it is not possible.

This is often the case, for example, in density estimates. Many of the most accurate probability models implicitly reflect probability distributions. In many of these models it is impossible to calculate the assumed exact probability value at a particular location in space. In such cases, you need to find another benchmark that still meets your design goals, or find one that is a good approximation to the required criteria.

5.3 E EXPERIENCE

Machine learning algorithms can be divided into two main categories, unsupervised and supervised, based on the type of experience they can get during training. Many of the learning algorithms described in this book can be understood as having access to a complete dataset. By definition, a data set is a collection of many different samples. In some cases we also refer to examples of data points. The Iris dataset is one of the first to be studied by statisticians and academics interested in machine learning. This is a collection of measurements made on various parts of one hundred and fifty iris plants. An entity is represented by each of the various activities.

The length and width of the leaves, as well as the length and width of the sepals and the length of the petiole, are features included in each of the specimens. In addition, the dataset tracks which species each plant belongs to. This dataset contains a total of three different types. Unsupervised learning algorithms are exposed to a dataset containing many features and then learn useful aspects of the structure of the dataset through this experience. Deep learning is usually about learning about the exact probability distribution responsible for generating a dataset. This can be done explicitly, as in the case of density estimation, or indirectly, as in the case of matching or noise removal. Different unsupervised learning algorithms perform different functions such as: B.

Clustering, which divides the dataset into groups of similar samples. Supervised learning algorithms are exposed to a dataset containing features, but each sample is also associated with a label or target. For example, the iris dataset contains descriptions describing the species of each iris plant. By analyzing the iris dataset, a supervised learning system can obtain the information needed to classify iris plants into three different types based on their measurable characteristics.

While unsupervised learning and supervised learning are not entirely formal or distinct ideas, they serve to roughly classify some of the things we do with machine learning algorithms. In fact, unsupervised learning happens when the student is not directly supervised by an instructor. Supervised learning is a term traditionally used to describe problems such as regression, classification, and structured production. Unsupervised learning is often understood as the process of estimating intensities to support other activities. Various permutations of the learning paradigm can occur. For example, in semi-supervised learning, some situations include a supervisory goal while others do not. Other examples do not include a tracking target.

In multi-instance learning, the entire collection of instances is marked according to whether it contains an instance of a class, even if the individual members of the collection are not marked. This is in contrast to traditional learning, where each instance in a collection is individually labeled. See the work of Kotzias et al. for a contemporary example of multi-instance learning with deep models. (2015). Some machine learning algorithms do not learn from the experience of seeing a fixed dataset. For example, reinforcement learning algorithms are designed to interact with their environment; this creates a feedback loop that connects the learning system with the experiences made. Such algorithms are beyond the scope of what can be covered in this book.

If you want to learn more about reinforcement learning, read Sutton and Barto (1998) or Bertsekas and Tsitsiklis (1996). Please also Mnih et al. (2013) for more details on the deep learning method for reinforcement learning. The vast majority of machine learning algorithms only look at data in a dataset. There are several methods to characterize a data set. In any case, a dataset consists of a collection of examples, which in turn are collections of features. It goes without saying that to define a dataset as a design matrix, each sample can be characterized as a vector and the size of each of these vectors must be the same. It won't always be like this.

For example, if you have a collection of images that differ in width and height, the resulting photos will also differ in the number of pixels they contain. So probably not all photos can be represented at the same vector length. Both explain how to deal with different situations. Of course, the label is not always a unique number; sometimes it will be a series of numbers. For example, if we want to train a speech recognition system to transcribe entire sentences, the label for each example sentence is a sentence. There is no precise definition of supervised and unsupervised learning, nor a strict taxonomy of datasets or experiments. The frameworks presented in this article cover the vast majority of use cases; however it is still possible to build new ones for different purposes.

5.4 LINEAR REGRESSION

Our definition of a machine learning algorithm as an algorithm that can improve the performance of a computer program on a task through experience is somewhat vague. To illustrate and make this point clearer, let's look at an example of a simple machine learning algorithm: linear regression. We will return to this example over and over as we continue to discuss ideas about machine learning that will help us better understand its behavior. In this scenario, the weight corresponding to the additional input 1 assumes the function of the offset parameter. In this book, the word "linear" is used very often to denote affine functions. The affine transformation has a parameter often called the bias parameter.

This parameter is the interrupt term b. This terminology assumes that the transform output tends to assume the value of b even when there is no input. This concept should not be confused with the concept of statistical bias, which refers to the fact that the estimate of a number estimated by a prediction algorithm does not match the actual quantity. The linear regression model is a good example of how learning algorithms can work, even though it is an incredibly simple and restricted form of a learning algorithm. In the following sections, we will discuss some of the key ideas behind the design of learning algorithms and explain how these principles can be used to create increasingly complex learning algorithms.

5.5 CAPACITIES, COMPLETE AND EXTRA EQUIPMENT

We need to succeed not only with the inputs on which our model is trained, but also with new inputs that we have never seen before. This is the main problem in machine learning. The ability to perform well on previously unrecognized inputs is known as generalization. Normally, when we train a machine learning model, we have access to a training set, we can calculate an error measure for the training set, which we call training error, and we want to reduce this learning error. What we've discussed so far is basically just an optimization problem.

The difference between machine learning and optimization is that in machine learning we want to minimize the generalization error, which is often called test error. The amount of error that should occur with a new entry is understood from the term "generalization error". In this case, the expectation is applied to various potential inputs selected from the set of inputs that we expect the system to encounter in the real world. The generalization error of a machine learning model is typically calculated by comparing the model's performance to a set of test samples collected elsewhere and separate from the training set.

The statistical learning theory discipline offers some explanation for this question. If the data for the training set and the test set were randomly obtained, there's really nothing we can do about it. We can make progress if we are allowed to make certain assumptions about the training and the process used to obtain the test kit. The data generation process is a probability distribution over datasets responsible for generating both the train data and the test data. In most cases, we create a set of hypotheses that are collectively referred to as the iid hypotheses.

These assumptions include that the samples of each dataset are independent of each other and that the train set and test set are distributed in the same way, with one chosen as the other from the same probability distribution. We also assume that the samples in each dataset are randomly selected from the same probability distribution. Based on this hypothesis, we can explain the data generation process using a probability distribution that includes a single sample. Then, every sample move and every sample test is created with the same distribution. This common base distribution is called the data generating distribution and its symbol will be pdata.

Thanks to this probabilistic framework and the assumption of independent and uniformly distributed data, we can quantitatively analyze the relationship between learning error and test error. Of course, when we apply a machine learning technique, we do not predetermine the parameter values; Instead, we take samples from both datasets. First we take a sample from the training set, then we use it to select the parameters in the training set that minimize the error, and finally we get a sample from the test set. According to this method, the expected test error value is greater than or equal to the expected learning error value . An algorithm's capabilities are the most important determinants of its ability to apply machine learning.

Under- and over-fitting are two of the biggest problems in machine learning, and these two metrics are relevant to those challenges. Insufficient fit occurs when the model does not get a low enough error value for the desired target in the training set. Overfitting occurs when training error and test error differ too much. By changing the ability of a model, we can control whether it produces more or less results. Generally, the ability of a model refers to its ability to adapt to various processes. Models with limited skills may find it difficult to adapt to the training set. Large-capacity models have a higher risk of overfitting, as they are more likely to remember features from the training set that are not useful for the test set.

Choosing the hypospace of a learning algorithm that expresses the set of functions that the learning algorithm can identify as a solution is a method of checking the ability of a learning algorithm. For example, the hypospace of the linear regression method is the set of linear functions derived from the input. We can make linear regression more general by extending the hypospace of linear regression to include polynomials as well as linear functions. Following these steps will improve model performance. A linear regression model is one we already know, and a first-order polynomial gives us that model so we can make predictions.

The optimal conditions for machine learning algorithms to work are those where their processing power is proportional to the actual difficulty of the job they are responsible for and the amount of learning data they feed. Models incapable of doing this are incapable of solving difficult problems. High-performance models are capable of solving complex problems; However, if this capacity is greater than needed to solve the current problem, the models may become too large.

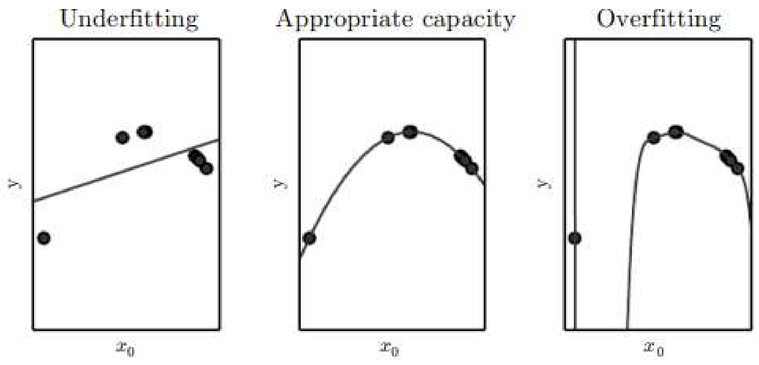


Figure 5.1 We fit three models into this sample training set.

The training data were prepared by taking a random sample from the x values and calculating a quadratic function and choosing the y values in a predetermined way. This process was done on a computer. A linear function fitted to the data suffers from underfitting, which means it cannot capture the curvature found in the data. (Left) (Middle) A quadratic function fitting the data provides a good generalization to unobserved points. It is not affected in any way by excessive or under-fitting. (Right) Overfitting occurs when a ninth-order polynomial is applied to the data. In this case, we tried to solve the underdetermined normal equations using the Moore-Penrose pseudo-inverse.

The approach iterates through each of the training points exactly, but so far we haven't been successful enough to determine if it will extract the correct structure. There is now a significant valley in the middle of the two training points, but this valley is absent in the real feature below. Also, although the actual function falls on that area of the graph, it increases significantly on the left side of the data. So far, we have only discussed one method of increasing or decreasing the capacity of a model; this is to change the set of input properties of the model by adding new parameters related to these properties.

The capacity of a model can actually be changed in several ways. Model selection is not the only factor that determines capacity. The model defines families of functions that the learning algorithm can select when adjusting parameters to achieve the desired level of proficiency in a training goal. This aspect of the model is called "representation". Solving the optimization challenge of choosing the most appropriate member of functions in this family can be difficult in many cases. In real use, the learning algorithm doesn't really recognize the optimal function; instead find the function that greatly reduces the number of training errors. B.

Due to these additional constraints, such as the imperfection of the optimization algorithm, the effective capacity of the learning method may be less than the representative capacity of the model family. This is because the optimization algorithm is flawed. Improvements to the concept, which date back at least to Ptolemy, can be found in current considerations of improving the generalizability of machine learning models. Occam's razor was the term most used by many early thinkers for the concept of frugality. According to this concept, if more than one hypothesis provides an equally plausible explanation for a known fact, it is advisable to choose the "simplest" competing hypotheses.

In the 20th century, the pioneers of statistical learning theory were responsible for formalizing and specifying this concept. (Vapnik and Chervonenkis, 1971; Vapnik, 1982; Blumer et al., 1989; Vapnik, 1995). The statistical learning idea offers a number of different methods for assessing model ability. The Vapnik-Chervonenkis size, sometimes known as the VC size, is the most notable of these. The capacity of a binary classifier can be determined using the VC dimension. The VC dimension is defined as the maximum possible value of m where there is a training set of m distinct x points that the classifier can label in any way it sees fit. This is my greatest value imaginable.

The ability of statistical learning theory to provide quantitative predictions depends on the model's ability to quantify. The most important results in the field of statistical learning theory show that the gap between learning error and generalization error is limited from above by a size that increases proportionally with the size of the model capacity but shrinks with the number of training examples. increases. . . This is one of the most important discoveries in this field. (Vapnik and Chervonenkis, 1971; Vapnik, 1982; Blumer et al., 1989; Vapnik, 1995). While these limitations intellectually justify the operation of machine learning algorithms, in reality their use in relation to deep learning algorithms is extremely rare.

This is because the constraints are often very loose and it can be quite difficult to evaluate the performance of deep learning algorithms. The problem of determining the performance of a deep learning model is particularly difficult because the actual performance is limited by the performance of the optimization algorithm, and we only have a limited theoretical understanding of the very general non-convex optimization models involved in deep learning . This makes the problem particularly difficult.

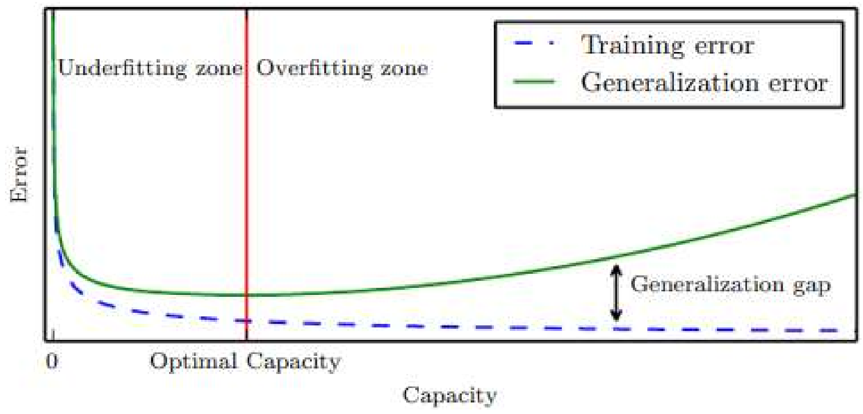


Figure 5.2: Typical relationship between skill and error.

It is important to keep in mind that although simpler functions tend to generalize more (resulting in a smaller gap between learning and testing errors), we still need to choose a hypothesis that is complex enough to support in order to arrive at a low conclusion. training error level. In general, as the model capability increases, the training error continues to decrease until it reaches the lowest possible error value asymptotically. (assuming the error measure has a minimum value). In most cases, the generalization error can be represented as a U-shaped curve when plotted against model capability. Figure 5.1 illustrates this idea visually. To get there, we'll start with the most extreme scenario of arbitrarily large capacity.

You can distinguish error behavior during training and testing. Training errors and generalization errors are higher on the left side of the graph. Improper diet is what you see here. As we increase capacity, we see a decrease in training errors; However, we see a growing gap between learning errors and generalization errors. Eventually, the magnitude of this deviation becomes greater than the training error reduction, and we enter the regime of overfitting, which is characterized by a very large capacity relative to optimal capacity. idea of templates that don't require parameters. Until now, only parametric models such as linear regression have been presented. Before any data analysis, parametric models acquire a function defined by a vector of parameters. The size of this vector is both finite and predetermined.

There is no such restriction for non-parametric models . The size of the training set can affect both the training error and the generalization error. The amount of training samples does not under any circumstances lead to an increase in the expected generalization error. More data leads to more generalization when nonparametric models are used, until the best potential error is reached. Any fixed parametric model whose capacity is less than its optimal value will always be asymptotic to an error value greater than the Bayesian error. For an example, see Figure 5.2. It is important to note that even at the maximum capacity of the model, there can still be a large discrepancy between the training error and the generalization error. In this case, it is possible to fill this gap if we collect more training cases.

5.6 FREE NOUN THEOREM

You can distinguish error behavior during training and testing. Training errors and generalization errors are more on the left side of the graph. Improper diet is what you see here. As we increase capacity, we see a decrease in training errors; However, we see a growing gap between learning errors and generalization errors. Eventually, the magnitude of this deviation becomes greater than the training error reduction, and we will enter the regime of overfitting, which is characterized by a very large capacity compared to the optimal capacity. idea of templates that don't require parameters. Until now, only parametric models such as linear regression have been presented. Before any data analysis, parametric models acquire a function defined by a vector of parameters. The size of this vector is both finite and predetermined.

There is no such restriction for non-parametric models. The size of the training set can affect both the training error and the generalization error. The amount of training samples does not under any circumstances lead to an increase in the expected generalization error. More data leads to more generalization when nonparametric models are used, until the best potential error is reached. Any fixed parametric model whose capacity is less than its optimal value will always be asymptotic to an error value greater than the Bayesian error. Go to Figure 5.2 to see an example. It is important to remember that even when the model is at its maximum capacity, there can be a large mismatch between the training error and the generalization error. In this case, it is possible to fill this gap if we collect more training cases.

5.7 REGULATION

The idea that there is no such thing as a "free lunch" suggests that we should design our machine learning algorithms with the goal of excelling at a particular activity. To do this, we have added a number of preferences to the learning process. The performance of the algorithm improves when these preferences are matched with the learning problems it is designed to address. So far, the only technique we have specifically considered for modifying a learning algorithm is to improve or reduce the representativeness of the model by adding or subtracting features from the hypospace of possible solutions that the learning algorithm can choose. To show how to solve a regression problem, we give the example of changing the degree of a polynomial. The point of view we have described so far is very simple.

Not only does the size of the set of functions allowed in our algorithm's hypospace has a significant impact on the behavior of our algorithm, but the identities of the functions also play an important role in this sense. The learning method known as linear regression that we have discussed so far includes a hypospace consisting of linear functions of its input. These linear functions can be very useful in solving problems where the coupling between input and output is highly linear or nearly linear. They are of little use when dealing with problems with very nonlinear behavior patterns.

For example, if we use linear regression to estimate sin(x) from x, this doesn't work very well because sin(x) is not linearly related to x. Thus, we can control the performance of our algorithms by determining the types of functions from which solutions can be generated and the number of functions from which solutions can be derived. We also have the ability to ask a learning algorithm to prefer a particular answer over others in the hypospace it uses. This indicates that both activities are acceptable, but one of them is preferred. The least preferred solution is selected only if it fits the test data significantly better than the currently used solution.

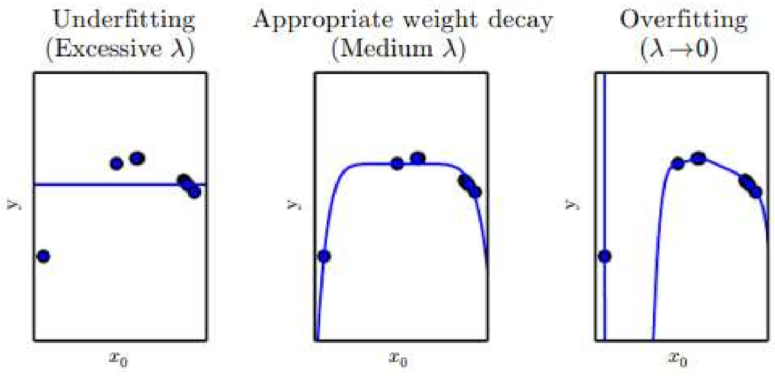


Figure 5.3 We apply a high-order polynomial regression model to our sample training set

One method for adjusting the capacity of a model more extensively than simply adding or excluding members of the hypospace is to express preferences for one feature over another. The exclusion of a function from a hypospace can be viewed as an expression of an infinitely strong preference for that function. In the example of decreasing weights, we have made our preference for linear functions created with lower weights very clear by adding one more term to the criteria we have reduced. This allowed us to find the optimal solution. There are a variety of other methods, both implicit and explicit, for communicating perspective on various possible solutions. Regularization is a common term for these many methods.

The term "regularization" refers to any change made in a learning algorithm with the aim of reducing the generalization error of that method, leaving only the training error. Regulation is an important topic in machine learning work and is second only to optimization in the importance of the role it plays. Because of the no free lunch theorem, it is now perfectly clear that there is no optimal machine learning algorithm, and more specifically, no optimal way of organizing. Instead, we need to choose a regularization method that tries to solve the specific problem we're trying to solve. The idea behind deep learning in general, and this book in particular, is that a wide variety of problems (for example, any intellectual problem an individual can solve) can be tackled effectively using versatile types of editing. That's the concept behind these two studies.

5.8 HYPERPARAMETERS AND VERIFICATION SETS

Most machine learning algorithms give us a number of different options that we can tweak to adjust how the algorithm learns. These are called hyperparameters in the industry. The learning algorithm itself does not make any changes to the hyperparameter values. (we can still design a nested learning scheme where one learning algorithm learns the best hyperparameters for another learning algorithm). In the polynomial regression example we see in Figure 5.3, there is only one hyperparameter. This hyperparameter is the degree of the polynomial that acts as the ability hyperparameter. An example of such a hyperparameter is the parameter value used to regulate the amount of weight loss.

Sometimes a parameter is chosen as a hyperparameter even though the learning algorithm does not learn it because it is difficult to optimize. This happens when adjustment is difficult. Most of the time optimization should be a hyperparameter, as it is unacceptable to train this hyperparameter on the training set. This applies to any hyperparameter that controls model capability. If such hyperparameters were trained on the training set, they would always choose the largest model capacity available, leading to overfitting. (see Figure 5.3). For example, we can always better fit the training set with a higher order polynomial and weight loss parameter = 0 than with a lower order polynomial and a positive weight loss parameter.

This is because a positive weight reduction parameter causes the order of the polynomial to increase as the order of the polynomial decreases. To find a solution to this problem, we need a set of validation states that are not seen by the training method. Earlier, we explained how a suspended test set of samples with the same distribution as the training set can be used to predict a student's generalization error after the process is complete. It is important to avoid basing decisions about the model, especially its hyperparameters, on the results of the test samples in any way. Because of this factor, the validation set cannot use samples from the test set. That's why we never build the validation set without first using the training data. Specifically, we divided the training data into two different subsets and used both. Parameters are learned with one of these subgroups.

The second subset is called our validation set and is used to estimate the generalization error during or after training. This gives you the ability to modify the hyperparameters accordingly. While this term can be confused with the larger data pool used for the rest of the training process, the subset of data used to learn the parameters is still commonly referred to as the training set. The subset of data that serves as the basis for determining which hyperparameters to use is called the validation set. In most cases, about 80% of the training data is used for training, while the remaining 20% is used for validation. Because the validation set is used to "train" the hyperparameters, the validation set error will underestimate the generalization error, but will usually be by a smaller amount than the training error.

In fact, the validation set is used to "train" hyperparameters. The generalization error can be calculated using the test set after all hyperparameter tuning is complete. In practice, when the same tests have been used repeatedly over many years to evaluate the performance of different algorithms, and especially given all the attempts by the scientific community to exceed the mastery level of this test set, we finally conclude that there are optimistic ratings at the bottom, even with the test set. As a result, references may become outdated and no longer accurately represent the true performance of a field-trained system. We are fortunate that the community is turning to new benchmark datasets, which are often more ambitious and larger.

5.9 CROSS VERIFICATION

This can be a problematic strategy if the test set size gets smaller because the dataset is split into a fixed training set and a fixed test set. Due to the limited size of the test set, it is difficult to say that algorithm A performs better than algorithm B at the given job, as the estimated mean error of the test is surrounded by statistical uncertainty. This is no longer a major issue when the dataset contains at least tens of thousands or even hundreds of thousands of samples.

If the dataset is too small, there are alternative approaches that allow all samples to be included in the process of estimating the mean test error; However, this comes at the expense of an increase in the computational effort required. These approaches are based on the concept of performing training and testing calculations multiple times on a few randomly selected subsets or sections of the initial dataset. The k-fold cross validation method shown in Figure 1 is the most commonly used method.

This method splits the dataset into k separate, non-overlapping subsets. The test error can then be determined by averaging the test error over all k trials. The ith subset of data is used as the test set during trial i, while the remaining data is used as the training set for that trial. Since there are no unbiased estimators for the variance of these mean error estimates, which is one of the problems (Bengio & Grandvalet, 2004), approximations are mostly used.

The study of statistics offers several techniques that can be used to achieve the goal of machine learning, which is not only to solve a problem in the training set, but also to generalize the results. It is helpful to have a basic understanding of concepts such as parameter estimation, bias, and variance to clearly define ideas such as generalization, under-fitting, and over-fitting. The variance of an estimator, also known as the standard error, is a statistic that tells us how well we can predict our estimate based on the data that will change as we use the data set, regardless of the process that generates the data. resample the first location.

If possible, we prefer an estimator to have relatively low variance, just as we would like it to have low bias when used. When we calculate a statistic with a small number of samples, our estimate of the actual underlying metric is uncertain. This may be because we got other champions from the same distribution and these champions have different stats. We have to find a way to measure a source of error, that is, the expected degree of variance each estimator will have.

The effective variance of the samples is denoted by xi. An estimate is often used instead of a standard error estimate. Unfortunately, it is not possible to obtain an unbiased estimate of the standard deviation using the square root of the sample variance or the square root of the unbiased variance estimator. Both methods tend to underestimate the true standard deviation, but both are commonly used in true statistical analysis. The square root of the unbiased estimator of variance is an estimate that is least likely to be wrong. For large m the approximation is correct up to a point. Machine learning experiments can benefit greatly from using the standard error of the mean.

When trying to estimate the generalization error, it is common for us to calculate the sample mean of the error in the test set. The accuracy of this estimation is directly proportional to the number of samples included in the test set. We can use the standard error to determine the probability that the actual expectation will fall into any range we choose using the central limit theorem, which says that the mean will be approximately normally distributed. This theorem tells us that the mean will be approximately normally distributed.

5.10 TREND AND VARIANCE CHANGE TO MINIMIZE MEAN SQUARE ERROR

Bias and variance are used to measure the different types of errors that can be included in an estimator. Bias tries to measure the expected deviation from the true value of the function or parameter. On the other hand, variance provides a measure of the deviation from the predicted predictor value that a particular data sample is likely to produce. This inconsistency may result from any data sampling. What are the implications of offering a choice between two estimators, one with higher bias and the other with higher variance? How do we choose between them? For example, let's take a scenario where we want to approximate the function shown in Figure 5.3, but the only option we have is to choose between a highly biased model and a model with significant variance. How do we choose between them? So far, we have discussed the properties of various estimators using a training set of fixed dimensions.

In most cases, we are also interested in the behavior of an estimator in response to an increase in the amount of training data. In particular, we hope that our point estimates will eventually approach the true value as the number of data points in our dataset continues to increase. In the last section, we looked at several definitions of general estimators and discussed the properties of these estimators. But where do all these forecasting methods come from? We want a principle from which to derive some functions that are very good estimators for different models. This allows us to avoid the process of making educated guesses about which functions will be efficient estimators and then examining the bias and variance of those functions.

The cross-entropy between distributions can be reduced by reducing KL to this exactly same divergence. Many authors make the mistake of using the term "cross-entropy" when they want to refer to the negative log probability of a Bernoulli or softmax distribution. This is not an accurate definition of the concept. For any loss of negative log probability, there is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by the model. For example, the cross-entropy between an empirical distribution and a Gaussian model is called the "mean squared error.

Maximum probability can be viewed as an attempt to reconcile the empirical distribution of the p-data with the model distribution. Ideally we could match the actual data produced by the pdata distribution, but unfortunately we do not have direct access to that distribution. Consistent estimators can have different statistical power levels; This means that a consistent estimator may have a lower generalization error for a given number of samples (m), or in other words, require fewer samples to achieve the same level of generalization error.

Statistical power work is usually performed in the parametric scenario (like linear regression) where the goal is to estimate the value of a parameter rather than the value of a function (and that's a real parameter). The expected mean square error is a method that can be used to determine how close we are to the true value of the parameter. This method consists of squaring the difference between the estimated and actual values of the parameters on which the expectation is based, according to the m training samples drawn from the distribution used to generate the data.

It shows that the parametric mean squared error decreases as m increases and for large values of m the Cramér-Rao lower bound (Rao, 1945; Cramer, 1946) shows that the error of no consistent estimator is smaller than the mean squared error. probability estimator When it comes to machine learning, maximum probability is generally considered the best estimator due to its consistency and efficiency. Regularization algorithms such as Weight Reduction can be used to generate a biased maximum-likelihood version with less variance when the training data is constrained. This can be useful in cases where the sample count is too low to avoid overfitting behavior.

The contribution of the Bayesian predistribution is the reason for the second significant difference observed between the maximum likelihood estimation approach and the Bayesian estimation method. The first acts by shifting the probability mass density to preselected parts of the parameter space. This is how the former uses its power. In practice, predecessors often prefer models that are simpler or more fluid in their behavior. Prior is cited by opponents of Bayesian methodology as a source of subjective human judgment that can influence estimates. The Bayesian approach generally generalizes much better when only a small amount of training data is available, but often suffers from high computational costs when there are many training examples.

Just as Full Bayesian Inference has the advantage of using the information outlined above but cannot be found in the training data, Bayesian MAP Inference has the advantage of using this information. This additional information is used to reduce the standard deviation of the MAP point estimate. (compared to ML estimation).

However, this goal is achieved at the cost of increased injury. Many regular estimation methods, such as regularized maximum probability learning with decreasing weights, can be understood as performing the MAP approach of Bayesian inference. This interpretation is possible because of how regular learning of maximum probability works with weight loss. This point of view is applicable where regularization involves incorporating an additional term into the objective function whose value corresponds to the logarithm of p(). There is no one-to-one overlap between regularization sanctions and Bayesian MAP inference.

For example, the logarithm of a probability distribution may not be equal to the logarithm of some regularization terms. Other smoothing factors depend on the data, which is an unavoidable obstacle to a previous probability distribution. Bayesian MAP inference is a simple method that allows the generation of complex but interpretable regularization terms. For example, a more difficult penalty term can be generated by using a combination of Gaussian distributions as the preceding distribution instead of a single Gaussian distribution. In fact, a combination of Gaussian distributions is more complex than a single Gaussian distribution.

The decision tree (Breiman et al., 1984) and its many variants are examples of another category of learning algorithms. This class of algorithms similarly divides the input space into regions and uses different parameters for each region. As seen in Figure 5.7, each node in the decision tree corresponds to a specific region in the input space. The internal nodes of the tree then divide this region into a subregion for each of the node's descendants. (usually with axis cut). As a result, the space is divided into non-overlapping regions and there is a one-to-one correlation between leaf nodes and input regions.

In most cases, every point in the input range served by a leaf node is assigned the same output. Training decision trees often involves the use of special methods, which is beyond the scope of this book. Although decision trees are usually ordered with size limits that make them effectively parametric models, the learning process can be considered non-parametric if you allow learning a tree of unlimited size. However, non-parametric models are not as common as parametric models. Decision trees, when used in the traditional way they are used, struggle to solve many simple problems, even for logistic regression, with axis-aligned splits and constant output at each node. For example, if we have a two-class problem and the positive class appears where x2 is greater than x1, then the decision limit will not align with the axis.

The decision tree must then implement a step function that continuously moves back and forth through the actual decision function in axis-aligned steps to approach the decision limit. This requires the use of a large number of nodes. As we have seen, both decision trees and nearest neighbor estimators have a number of disadvantages. However, they are efficient learning algorithms when the amount of computing resources is limited. We can also understand more advanced learning algorithms by considering the parallels and contrasts between more advanced algorithms and baselines such as k-NNs or decision trees.

5.11 UNSUPERVISED LEARNING ALGORITHMS

You may recall from the previous section that unsupervised algorithms are algorithms that only encounter "features", not in a heartbeat. The boundary between supervised and unsupervised algorithms cannot be clearly defined, as there is no objective criterion for distinguishing whether a value is a property or a goal set by a supervisor. Indeed, the distinction between a property and an end cannot be made. Informally, "unsupervised learning" refers to most efforts to extract information from a distribution that does not require human labor to annotate examples.

In this type of learning, there is no supervisor who oversees the process. It is commonly used to find a manifold with data close to it, group data into clusters of similar samples, estimate the density of a population, learn how to sample from a distribution, learn how to de-noise data from a distribution. . Finding the "best" representation of the data is a common task in traditional unsupervised learning. The term "best" can mean a number of different things, but in general we aim for a representation that contains as much information about x as possible, and we meet some penalty or limitation to make the representation easier to understand or more accessible than x.

There are many different approaches to defining a simpler representation. Low-dimensional representations, sparse representations, and independent representations are some of the most common types of representations. In low-dimensional representations, an attempt is made to condense as much information about x as possible into a more compact representation. Sparse representations place the dataset in a representation that is mostly zero for most of the inputs (Barlow, 1989; Olshausen & Field, 1996; Hinton & Ghahramani, 1997). These representations were developed by Barlow et al. When using sparse representations, it is often necessary to increase the size of the representation.

This is done to ensure that converting the representation mostly to zeros does not cause too much information loss. The end result is a general representation structure that tends to distribute the data along the axes of the represented space. Independent representations are representations that attempt to resolve the different sources of variation underlying the data distribution to ensure that the representation sizes are statistically independent. These three requirements are far from compatible with each other. Often, low-dimensional representations produce elements that have fewer or weaker dependencies on the high-dimensional data originally used. This is because finding and removing redundancy is a technique for reducing the size of a representation.

By finding and eliminating more instances of iteration, the dimensionality reduction technique can achieve more compression while discarding less information. The concept of representation is one of the main pillars that supports deep learning and is therefore one of the main pillars that this book supports. In this part of the article, we will create some simple examples of representative learning algorithms. These three example algorithms combined show how the above three requirements are implemented. Most of the sections below describe new representative learning algorithms, many of which develop these criteria differently or introduce new criteria.

One of the difficulties with clustering is that the clustering issue itself is inherently ill-positioned. This means that there is no single criterion that can measure how well a set of data corresponds to the reality against which it was collected. We can measure clustering parameters such as the mean Euclidean distance between the centroid of a cluster and its members. This gives us the opportunity to evaluate how well we can reconstruct training data from cluster assignments. We don't know how well the real-world features match the clustering assignments. It's also possible that there are several alternative groupings, each corresponding to a different real-world feature. It's possible that we could get in there hoping to find a cluster that matches an attribute, but end up with another cluster that is just as legitimate but unrelated to our business.

Consider the following scenario: We have a dataset containing images of red trucks, red cars, gray trucks, and gray cars. We decided to perform two different clustering methods on this dataset. If we ask each clustering algorithm to identify two clusters, one method may find a cluster of cars and a cluster of trucks, while another algorithm may find a cluster of red vehicles and a cluster of gray vehicles. If we ask each algorithm to find two clusters, we find which clustering algorithms give the most accurate results. Suppose, in addition to the first two clustering algorithms, we run a third algorithm that can count the number of clusters. This can classify champions into one of four categories: Red Cars, Red Trucks, Gray Cars or Gray Trucks.

This new grouping method takes into account information about at least both attributes, but not information about similarities. Just as red cars are grouped separately from gray trucks, red cars are grouped separately from gray cars and gray trucks. The results of the clustering method do not indicate that red cars look more like gray cars than gray trucks. Instead, the results show that red cars and gray cars look more alike. You left with these two things, and that's all the information we have right now. These concerns highlight some of the possible reasons why we prefer distributed representation to warm representation.

A distributed representation may contain two attributes for each vehicle, one that indicates the color of the vehicle and the other that reflects whether the vehicle is a car or a truck. The best distributed representation isn't quite clear yet (how does the learning algorithm know if the two features we care about are color and car versus truck rather than manufacturer and age?). However, the large number of features makes it easy for the algorithm to identify the most important feature for us. It also allows us to measure the degree of similarity between two objects in more detail by comparing multiple properties rather than checking whether a single value is the same.

5.12 DOWN OF THE STOCHASTIC GRADIAN

Stochastic gradient descent, sometimes known as SGD, is a very important method used in almost all deep learning applications. An extension of stochastic gradient descent is stochastic gradient descent. In general, gradient descent has a reputation for being a slow and unpredictable process. The use of gradient descent was previously seen as irresponsible and lacking in concept when applied to non-convex optimization problems. II of this series. We now know that training machine learning models with graded descent provide excellent performance. While the optimization process is not guaranteed to reach even a local minimum in an acceptable time, it usually finds a very low value of the cost function early enough to be useful.

The use of stochastic gradient descent is not limited to the field of deep learning and has many other related applications. It is the primary method used to train giant linear models on extremely large datasets. When the model size is fixed, there is no relationship between the size of the training set and the cost of an SGD update. In fact, as the size of the training set grows, we usually move to a more complete model, although this is not necessary at all. In most cases, a larger training set will require more updates before converging. However, as m approaches infinity, the model eventually converges to its best potential test error before sampling each sample from the SGD training set.

Although designed by hand, it can often be interpreted as using a special case optimizer. Some models, such as B. decision trees and k-means, require special-case optimizers because the cost functions of these models contain flat parts that prevent them from being minimized by gradient-based optimizers. These straight sections do not make models suitable for shrinking. Understanding that most machine learning algorithms can be characterized by applying this recipe makes it easy to see different machine learning algorithms as components of a taxonomy of ways to perform related tasks that work for similar reasons. It's best to think of various machine learning algorithms as a long list of algorithms, each with its own individual explanations.

5.13 DEEP LEARNING MOTIVATION CHALLENGES

The simple machine learning techniques detailed and discussed in this section work well for a variety of key challenges. However, they failed because of the AI's fundamental difficulties in recognizing language or objects in their environment. Traditional AI algorithms did not generalize very well when applied to deep learning problems, which was the main motivation behind the invention of deep learning. This section shows how it becomes exponentially difficult to generalize to new examples when working with high-dimensional data, and how the mechanisms used to arrive at generalization in traditional machine learning are not adequate for learning functions that are complex in large domains. Specifically, this section focuses on how the difficulty of generalizing to new samples becomes exponentially more difficult when working with high-dimensional data. Moreover, the computational requirements of such spaces are often quite high. Deep learning is specifically designed to address challenges like this and more.

The "curse of dimensionality" presents a series of challenges, one of which is a statistical problem. A statistical problem arises as the number of alternative configurations of x is significantly larger than the number of training examples, as shown in Figure 5.9. To understand the available material, let's first consider that the entrance room is arranged as a grid as shown in the figure. We can characterize low-dimensional space using a small number of cells mostly filled with data. Usually, when we want to generalize to a new data point, we can see what to do by analyzing training samples in the same cell as the new input. Then we can know what to do when generalizing to a new data point. For example, if we wanted to estimate the probability density at position x, we could return the number of training samples contained in the same unit volume cell as x and then divide this result by the total number of training samples.

If we want to classify a sample, we can take training samples from the same cell with the most common category. When we run a regression, we can take the target values observed in all states of that cell and then average those values. What about cells with no known pattern? A typical grid cell has no associated training examples, since the number of possible configurations in high-dimensional spaces is very large and far greater than the number of examples we have. How should we begin to express something meaningful for these new configurations? Traditional machine learning algorithms generally assume that the output at a new point should be approximately equal to the output at the nearest learning point.

5.14 LOCAL CONSISTENCE AND FLUIDITY CONTROL

For machine learning algorithms to generalize well, they must be guided by previous ideas of what kind of function they need to learn. In the past, we have seen these priorities embedded in the model as explicit beliefs in the form of probability distributions on model parameters. Informally, we can also define assumptions as having a direct effect on the function itself and only indirectly affecting the parameters through their effect on the function.

This way of thinking allows us to speak more generally of past beliefs. While these biases cannot be expressed (or even impossible to express) in terms of the probability distribution that represents our level of belief in various features, we informally discuss prior beliefs implicitly expressed in choosing algorithms based on the selection of a feature with which they align. the class of functions is above the other. We also discuss previous beliefs that were implicit in choosing algorithms to favor one class of functions over another. The softening precedent, or local tissue precedent, is one of the most frequently used implicit "previous" expressions by people.

That is, a priori, the function we learned should not change much over a very narrow range. The ability of many simpler algorithms to generalize successfully depends solely on these precedents, and so many algorithms cannot adapt to the statistical problems inherent in solving AI-level tasks. In this book, we will discuss how deep learning combines additional explicit and implicit priorities to reduce the amount of generalization errors the model produces when applied to complex tasks.

In this section, we will discuss why the previous leveling was not sufficient to perform these tasks. There are many different ways to implicitly or explicitly convey a previous view that the learned function must be smooth or locally constant. There are also many different beliefs about what should happen. All these different ways aim to advance the learning process to learn a function f that satisfies the requirement. This is the purpose of all these strategies.

Is there a way to express a complex function with the number of regions to differentiate much more than the number of samples used for training? It's pretty obvious that assuming the underlying function is smooth does not allow a student to do so. For the purposes of this illustration, imagine that the objective function looks like a chessboard. There are many different configurations of a chessboard, but each follows a simple plan. Imagine a situation where the number of training examples is much less than the number of black and white squares on a chessboard. What would it be? If we had previously only relied on local generalization and local regularity or consistency, we would certainly accurately predict the color of a new spot if it were within the same checkerboard square of a training pattern.

There is no guarantee that the student will be able to successfully extend the checkerboard pattern to points in squares that do not contain any training examples. From this precedent alone, the only information that can be obtained from a sample is the color of the square that composes it, and the only method of obtaining the colors of the smoothing hypothesis and learning algorithms that do not accompany parametric ones work exceptionally. Enough examples for the learning algorithm to see maximums at most peaks and minimums at most valleys of the True Base Value subfunction to be learned. In other words, assumption smoothing and non-parametric learning algorithms work very well provided there are enough examples.

This is true in most cases where the function to be learned is smooth enough and changes in size enough. Even a very smooth function can exhibit discontinuous behavior in high dimensions, changing smoothly but only in all dimensions. When the feature works differently elsewhere, it can be very difficult to characterize it from a collection of training examples. If the function is challenging (we want to distinguish a large number of fields in terms of number of samples), does the generalization have any chance of succeeding?

Other machine learning methods often involve generating more robust and task-specific hypotheses. For example, if we assume that the objective function is periodic, it will be much easier to solve the checkerboard problem. So that neural networks can be generalized to a much wider range of structures, we often ignore such robust and task-specific assumptions when designing them. Because the structure of AI tasks is too complex to be constrained by simple, manually determined features like periodicity, we need learning algorithms with more general assumptions.

The basic principle of deep learning is based on the assumption that data is created by combining a large number of different features or elements that can be located at different levels in a hierarchy. There are a number of additional assumptions that fall into the same category as deep learning and can be improved upon. These seemingly innocuous assumptions allow for an exponential improvement in the ratio between the number of samples and the number of differentiable regions.

5.15 MULTIPLE LEARNING

The existence of transformations on the manifold that allow movement from one place to another neighbor is implied in the concept of adjacency, which encompasses every position of the manifold adjacent to that point. In a diversity representation of the Earth's surface, one can go in each of the four cardinal directions: north, south, east or west. In machine learning, the term "diversity" is often used more generally to refer to a set of related points that can be well approximated by considering only a small number of degrees of freedom or dimensions that cover the most space. are high built-in dimensions. While the term "multiplicity" has a formal mathematical meaning, it's pretty comfortable to use in machine learning in general.

Each dimension is associated with a different direction of local variation. Figure 5.11 shows an example of training data near a one-dimensional manifold embedded in a two-dimensional space. When we delve deeper into the topic of machine learning, we allow the dimensionality of diversity to circulate from one point to the next. This usually happens when a manifold intersects. For example, a manifold is said to be eight-shaped if it has two dimensions at the central junction, but only one dimension at many other points.

Many of the machine learning challenges seem impossible to solve if we wait for the machine learning algorithm to learn to work with interesting variations across Rn. To circumvent this problem, multiplicity learning algorithms assume that most of Rn consists of invalid inputs and that interesting inputs occur through a manifold collection that contains only a small subset of the points. They also assume that interesting changes in the output of the learned function occur only in directions found on the manifold, or that interesting changes occur only when moving from one manifold to another. These assumptions allow them to solve the problem. While the concept of probability concentration was originally developed for use with continuously scored data and an unsupervised learning framework, it can also be generalized to work with discrete data and a supervised learning framework.

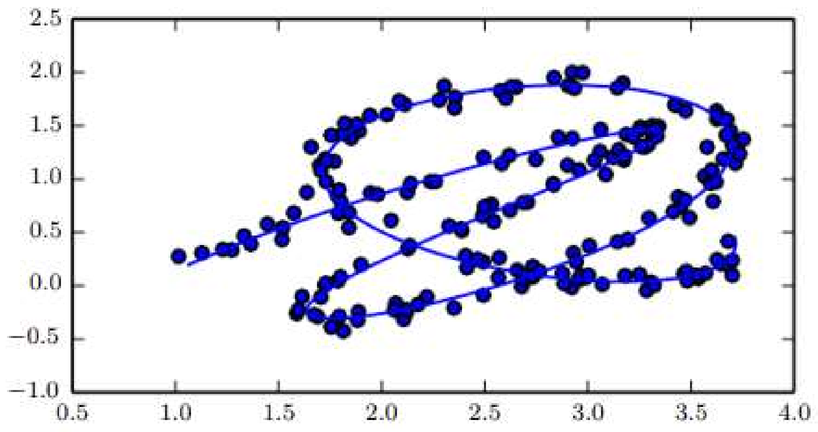


Figure 5.4 Data from a distribution in truly clustered two-dimensional space alongside a one-dimensional manifold such as a twisted string. The solid line shows the basic variety that the student should extract.

The basic assumption here is that the mass of probabilities is quite dense. Multiple learning was originally developed for this scenario. There is no guarantee that the assumption that the data lies along a low-dimensional manifold will always be correct or useful. In the context of AI tasks such as image, sound or text processing, we show that the multiple hypothesis is at least approximately valid. An example of such work is photo editing. There are two different groups of observations that constitute the proof of this concept. The first piece of evidence to support the multiple theory is the fact that the probability distribution of images, text strings and sounds that actually occur in the world is quite dense.

Inputs in these ranges almost never look structured, and continuous noise almost never looks structured. Figure 5.4 shows how evenly sampled points resemble the noise patterns seen on analog TVs when there is no signal. If you create a document by choosing letters with the same random frequency, how likely are you to end up with meaningful text in English? Again, almost zero, as the vast majority of long strings do not match any natural language strings: the distribution of natural language strings occupies a relatively small volume in the total area occupied by strings of letters.

Of course, using aggregated probability distributions as evidence is not enough to show that the data refer to a manageable number of manifolds. We also need to show that the instances we encounter are linked to other instances, and that each instance is surrounded by other instances that are extremely similar to it, and can be obtained by applying transformations as we go. Picker. That people can envision such localities and transitions in general terms is the second proof of plural theory. In the case of images, many different transformations can undoubtedly be envisioned, which make it possible to grasp a diversity in pictorial space. For example, we can gradually dim or brighten the lights, gradually move or rotate the objects in the image, gradually change the colors of the object surfaces, etc. It is still very likely that many adders will be used in most implementation processes. For example, the variety of human face images may not be correlated with the variety of cat face images.

5.16 SECTION OF THE ROBOT

Much work has been done to develop a segmentation solution that is as reliable as possible compared to ideal robotic segmentation. The original plan was to use this to generate a dataset for a segmentation-based neural network as a definitive answer. Due to the relatively recent release dates of these detection networks, we were not aware of existing SSDs or other networks with the required efficiency level at the time. As a result, more emphasis was placed on segmentation in the early stages. The goal has always been to extract location data from segmentation images so that the solution can be used regardless of using a tracking or segmentation technique.

This was done to make the solution useful. For this it was crucial to be able to isolate the areas of the image containing the robots; However, the program did not have to work as hard as it did to achieve error-free segmentation. Results FIG. Considering that the background constitutes 96.20% of the image, these techniques should be able to reach very high values. Although there is only 1.19% difference between the two methods in terms of percentage similarity, this indicates that the self-adaptive window approach works much better than the fixed size approach.

A single scan of an image takes approximately the same time either way; However, the self-adaptive solution only requires one scan versus the four or more scans required by the fixed size solution, depending on the number of scales present in the image. This provides a significant improvement in execution times. As a result, the final resolution of the image as a whole will also be inconsistent. Resolution is determined by a size called stride size, which is currently set to one quarter of the robot's expected size. This indicates that each line has its own resolution and gradually decreases as you move through the image.

This effect is evident in Figure 4.1(d), where the image becomes more fragmented as you move down the page. One option to improve resolution would be to simply reduce the step size; however, this will increase the time required for computation. Since the segmentation image will not be used directly by any network, we can assume that the 4 step size is accurate enough. It would be better if we focused on single solution for segmentation. Hypo has been confirmed by other research. In Figure 4.2(b) we see how the floating window works on an image with a lot of background but no real robot to identify.

This is because there are no real robots to search for in the image. The segmentation image reveals several locations where the classifier incorrectly detects robots of roughly the same size, shape, or color as a robot on the ground in parts of the image. These areas can be found in different places. When a collection of false classifiers is large enough, it is more likely to be misidentified as a robot, giving a false positive when parsing the final result. This violates the assumption made in the research question of developing a reliable detection method. It was clear that the classifier needed to be extended, which led to the improvement of the boot algorithm.

The planting process took much longer than on paper. Indeed, more iterations than expected were required to properly characterize the possibilities of the extreme case. While each iteration of the boot method mostly served to improve the model in the look that needed it most, it also left room for a few minor issues to grow into. However, the problems encountered were almost never as great as the problems encountered, and after sufficient iteration on a reasonable number of perfect segmentations, the system was able to settle on a very stable solution. Even though we didn't need full segmentation, we still managed to get surprisingly close to our target. This was achieved despite the fact that the output resolution of self-adjusting sliding windows was rather unstable. The results are presented in Figure 5.4 showing 99.13% accuracy against ideal segmentation, and Figure 5.4 shows that almost all background errors have been eliminated (c).

Resolution can be further improved by setting the step size of algorithm 1 to a smaller value. Also, the accuracy of the classifier could be further improved by planting even more photo samples or increasing the input image size to just over 64x64. This further demonstrates that the technique is designed to be modular enough to be used in a variety of different segmentation tasks, which has proven to be very beneficial in the long run. The bootstrap approach was the main factor contributing to its modularity. Although this method took a few iterations to reach its full potential, it helped streamline and speed up the process. Overall, the segmentation of the robot was successful and proved very useful as a starting point for a complete and reliable algorithm for its identification.

5.17 ROBOT LOCATION

The segmentation images generated by the combination of the self-adjusting sliding window and the fine-tuned CNN classifier served as the basis for the localization procedure. This has led to some issues that weren't considered before, eg. B. Difficulty visually distinguishing between different perceptions in the case of larger stains. This was a necessary step that needed to be considered extensively to make the survey dataset sufficiently complete; otherwise, networks probably wouldn't be able to effectively distinguish between conflicting bots. Both the custom cam shift and average shift methods failed because they both tried to find the highest density of the entire blob.

However, neither technique considered the possibility that the blob could contain more than one robot at a time. These strategies were unforgivably greedy and made no effort to optimize multiple hubs at once, resulting in massive errors and therefore inadequate answers for our study topic. The box filter technique was the first method used to design a filter with a kernel size proportional to the size of the robots. If we look at the figure, we can see that the blur technique only does a good job on blobs containing a robot. Even two robots, side by side but not overlapping, were able to output an output that determined the positions of the two centers as it should. The leftmost blob, on the other hand, has two robots that partially obscure each other and are therefore larger than the filter size.

This design decision was made to allow for the emergence of two different centers of mass, which would make it easier to separate the different components of the drop from different robots. We can see that this is not the case because the intensity values are centered over the entire block, rather than splitting in half as needed in the average and cam shift algorithms. Due to the square shape of the filter and the equal weight of everything in it, we obtained a cluster of high density values in the central area of the entire droplet, rather than in each of the centers individually. This is because the filter is square in shape. To illustrate this concept, we took the region containing two overlapping robots in Figure 5.4 and illustrated it in Figure 5.1. This allows us to show what the region looks like in the original image, the segmented image, and the appropriately filtered segmentation in (a), (b) and (c).

Another problem that arose was the time required for computation. While the method could be developed and made slightly faster than the simple and easy version used for testing, it would have taken a lot of effort to get it to work in real time had the procedure been implemented. for each pixel of the full size image. The main motivation for using this technique was to see if smoothing the segmentation with a filter adapted to the size of the robots would make it easier to find the robot centers. In that sense, it was a successful venture. It not only showed that the idea should be pursued, but also provided a clear understanding of the components of the concept that needed improvement to achieve the desired results. Overall, we were faced with two main issues that needed to be addressed.

5.18 GAUS FILTER APPROACH

A number of fixes and improvements were released simultaneously in a crucial step to address the issues caused by the box filter. Two separate factors contributed to its not being investigated as an independent component. First, we felt that once we combined these separate processes, the probability of this process resulting in a viable product was greatly increased. Therefore, testing each component is unlikely to tell us much about the overall process. Secondly, we didn't want to make the mistake of wasting too much time and energy before integrating a single module with other components. Much attention has already been paid to the development of the ideal segmentation algorithm, while a more center-oriented method is actually much more efficient in the later stages.

Therefore, the modifications were introduced separately, but tested and evaluated together as a larger unit. This seemed to have a positive effect on the result, as the tests had a direct impact on the final product and were more focused. The fact that the preparation process can now be managed much more broadly than in the past by changing the accuracy threshold for extreme cases is a significant change that is now being incorporated into the review process.

The final strategy examined the ground truth of the center pixel of each window to decide whether the classification was correct. Once the entire process has been reviewed, we can determine if the accuracy of false positives and false negatives in the new dataset needs to be increased or decreased. Ultimately, this proved to be a useful advantage. Alongside the complete discovery process, all necessary mechanisms have been put in place to ensure that datasets for multiple cores are automatically created, tested and improved. This made it possible to try different thresholds simply out of curiosity, rather than thoroughly evaluating which particular value was most likely to result in the greatest improvement before testing. This made it possible to try other sleepers out of curiosity.

We tried to cleverly change the thresholds, but the tests were cheap and the final classifier dataset is a collection of small wins from trial and error. Figure 4.5 shows the results of the procedure performed. If we want to fully understand why the different steps are followed, we also need to look at Figure 4.6. In the first iterations, denoted by the letter c in both images, our main goal was to develop a classifier that could produce a correctly centered segmentation image. To do this, set a threshold of 0.5 for both false positives and false negatives.

This meant that any classification in its immediate environment that even remotely conflicted with its own had to be reclassified and then fed back into the dataset. After the first few cycles, we began to see a pattern that did more harm than good to help us get closer to our goal. We can see that the classifier is too busy separating the overlapping bots into two separate blobs. The rear has been reduced to almost zero, making it extremely difficult to distinguish from background noise . In Figure 4.6(c), the blobs are grouped into a single entity and their attention is drawn only to the most important robot. The difficulty is that the fixed separation between adjacent robots is too rigidly enforced.

The small distance that exists between two relatively close centers is classified as false positive and then fed back into the dataset. To fix this, simply adjust the accuracy level of false positives to make it tighter. By setting it to 0.9 we make sure it doesn't add false positives unless the space it is in is mostly empty space. The result of this harmony and its Gaussian notation . While this is a good start, the two spots are now a little too close together, making it difficult to tell the two robots apart. relative to each other. This was a problem even before the city center strategy was implemented; Nothing has changed. In it), we can observe that each of them has the same high-density bridge connecting the centers of the overlapping robots in their respective regions. In the next test, the false-positive threshold is set to 0.8 while the false-negative accuracy level remains at 0.5. Hopefully this results in a threshold in the middle of the other thresholds evaluated.

The results are presented in and in respectively. (And). We've now reached the sweet spot where the drops have a size that allows them to be judged accurately, but still too large to be considered a single giant robot. The completed procedure has taken the appearance of our intended purpose as shown in Figure 3.9 and the final output confirmations are shown in . The accuracy criterion for false positives was set at 0.8, which was investigated with many other possible thresholds. But this figure gave the best results. It is also very important to note that the image used in this case is not part of the photos used for training and therefore represents an unbiased result.

CHAPTER 6

DEEP FEEDFORWARD NETWORKS

Collectively, these networks are called neural because they are influenced by neurology in general. In most cases, values in each embedded layer of the network are represented as vectors. The width of the model is determined by the size of these invisible base layers. It is possible to understand that each component of the vector performs a function similar to that of a neuron. Instead of thinking of the layer as representing a single vector-vector function, we can think of the layer as consisting of multiple units operating in parallel, each representing a scalar function from the vector. This allows us to look at the level in a different way.

Each unit is similar to a neuron in that it receives information from many other units and then calculates its own firing rate. The field of neurobiology has inspired the use of many layers of vector representation. The choice of functions used to calculate these representations is less informed by neuroscientific knowledge of functions that compute real neurons. However, contemporary neural network research is conducted by various fields of mathematics and engineering, and the purpose of neural networks is not to create an exact copy of the brain. Rather than seeing predictive networks as models of brain function, it makes more sense to think of them as function approximation machines.

These machines are made for statistical generalization purposes and sometimes take some insight from what we know about the brain. One approach to understanding feed-forward networks is to start with linear models and consider how to overcome the disadvantages of these models. The appeal of linear models is that they can be fitted easily and consistently, both in closed form and by applying convex optimization. Examples of linear models are logistic regression and linear regression. Also, linear models suffer from the obvious flaw that the model's capacity is limited to linear functions; Therefore, the model cannot understand the interaction between two input variables.

This global concept of improving models by acquiring new features is not limited to the feedforward networks discussed in this chapter. This is a fundamental principle of deep learning that can be applied to any of the many types of models discussed in this book. Applying this approach to the learning task of deterministic x to y mappings without using feedback links leads to feedforward networks. These basic ideas are applied in subsequent models as stochastic map learning, feedback function learning, and single vector probability distribution learning. This chapter begins with a basic demonstration of what a feedforward network looks like.

Next, we will discuss all the design considerations to consider for implementing a feedforward network. First of all, training a feedforward network requires consideration of many of the same design considerations as in a linear model. These include the choice of optimizer, the cost function, and the shape of the output units. We start by reviewing the fundamentals of gradient-based learning, then we discuss some design considerations specific to feedforward networks. Since feedforward networks are responsible for introducing the idea of the hidden level, we must choose which activation functions to use when calculating the values of the hidden level.

We also need to design the architecture of the network, which includes determining how many layers the network should have, how these layers should be related to each other, and how many entities each layer should contain. Gradient computation of complex functions is required for learning in deep neural networks. In this article, we will discuss the backpropagation method and its newer extensions, both of which can be used to calculate these gradients efficiently. Finally, we'll end with some historical context.

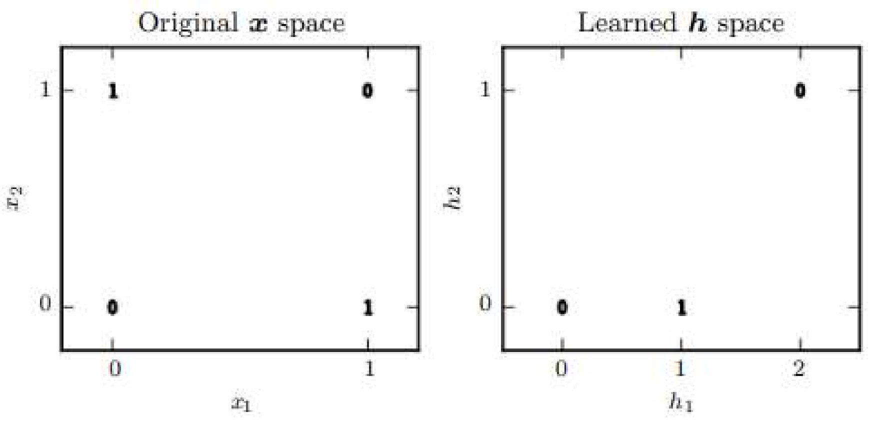


Figure 6.1: Solving the XOR problem by learning a representation.

The neural network was able to identify the appropriate response for each of the samples in the batch. In this proof, all we had to do was explain the solution and then prove it error free. Because there may be billions of model parameters and billions of training examples in a real-world scenario, it is impossible to simply predict the response as we did in this scenario. An alternative would be a gradient-based optimization method that looks for parameters that almost cause a few bugs. Since our solution to the XOR problem is at the global minimum of the loss function, the gradient descent can eventually go that far. There are other good alternative options for solving the XOR problem, all of which can be found via gradient descent. The initial values of the parameters affect where the gradient descent converges. In reality, gradient descent does not usually lead to neat, easy-to-understand solutions with integer results like the one given here.

6.1 DEGREE LEARNING

The process of designing and training a neural network is not significantly different from the process of training other progressive descent machine learning models. In the previous section, we explained how to build a machine learning algorithm by defining an optimization process, a cost function, and a family of models. This was done in Section 5.10. The nonlinear nature of neural networks means that most of the associated loss functions are not convex; this is the main difference between linear models and neural networks we have encountered so far.

This indicates that neural networks are mostly trained with gradient-based iterative optimizers that reduce the cost function to a very low value. This is in contrast to linear equation solvers used to train linear regression models, or convex optimization algorithms used to train logistic regression or SVM, both of which use global convergence guarantees. In principle, convex optimization converges from any initial parameter; It's actually quite durable but can run into digital issues. When used for non-convex loss functions, the stochastic gradient descent algorithm offers no guarantee of convergence and is sensitive to the values provided for the seed parameters. When working with feedforward neural networks it is important to break all weights into small random numbers.

It is possible to initialize biases to zero or very modest positive values. We will discuss in detail the iterative gradient-based optimization methods used to train feedforward networks and almost any other deep model. For now, it's enough to realize that the training algorithm almost always relies on the use of the gradient to somehow reduce the cost function. Individual methods are built on the concepts presented in it and are extensions and refinements of those ideas. In particular, individual algorithms are almost always extensions and refinements of the stochastic gradient descent technique offered within.

It goes without saying that we can also train models such as linear regression and support vector machines using gradient descent; In fact, this is typical practice when the training set is very large. In this respect, the process of training a neural network is not significantly different from training other models. Calculating the gradient in a neural network involves some additional complexity, but it is still possible to do it accurately and quickly. The back propagation technique and some recent changes to the back propagation algorithm are discussed in this section. This section describes how the gradient is derived. To implement gradient-based learning, just like any other machine learning model, we need to choose a cost function and also choose how to express the model's output. We will now return to these design considerations, paying special attention to the neural network case.

The gradient of the cost function should be large and predictable enough to serve as an effective guide for the learning algorithm, which is an iterative problem throughout the neural network design process. This goal is thwarted by functions that reach saturation or become excessively flat due to the effect of reducing the gradient to too small a value. This is, in many cases, the result of saturating the activation functions used to generate the output of the hidden units, or saturating the output units themselves. A number of different models could benefit from using a negative log probability to solve this problem. Used in many output units, the exp function can potentially be truncated if the input it receives is very negative.

When used with the negative daily probability cost function, the log function reverses the effect of experience value for certain output units. In the following, we will talk about the interaction that occurs between the cost function and the selected output unit. When applied to models commonly used in practice, the cross-entropy costs used to perform a maximum likelihood estimation typically have no minimum value. This is an unusual feature of the cross-entropy cost used to perform the maximum probability estimation. Most of the model parameters are structured in such a way that they cannot accurately represent the probability of zero or one for discrete output variables, but can approach arbitrarily. One model that falls into this category is known as logistic regression.

If the real-valued output variable model is capable of controlling the density of the output distribution (for example, by learning the variance parameter of a Gaussian output distribution), then with the right training extremely high densities can be assigned. Adjust the outputs to cause the cross-entropy to approach negative infinity. This can be achieved by learning the variance parameter of a Gaussian distribution of the output. The regularization methods discussed offer several approaches to modifying the learning problem in a way that prevents the model from gaining infinite rewards in this way.

6.2 OUTPUT UNITS

The choice of the production unit is inextricably linked with the choice of the appropriate cost function. We often rely only on cross-entropy, which is calculated by comparing the distribution of the data with that of the model. The shape of the cross-entropy function is then determined by the method chosen to represent the result. Any driver that can work at the output of a neural network can also work as a hidden driver in the network. In this section, we will focus on using these units as model output; in theory they can also be used in the model itself. We'll look at these units again in the Later section, with more information on how they're used this time as stealth units. Achieving the highest possible log probability consists of reducing the mean squared error to the lowest possible value. The maximum likelihood framework makes it easy to know Gauss's covariance and make Gauss's covariance a function of the input.

On the other hand, the covariance should be constrained to be a positive definite matrix for each input. Because these constraints are difficult to meet with a linear output level, alternative output units are often used to adjust the covariance. Modeling approaches for covariance are briefly discussed in the following sections. Since linear units are not saturated, gradient-based optimization methods are relatively easy to implement and are also compatible with a variety of other optimization methods. The Softmax function can be used whenever it is necessary to define a probability distribution for a discrete variable consisting of n different potential values.

One way to look at it is to generalize the sigmoid function, which was once the only way to describe the probability distribution over a binary variable. The output of a classifier is usually a softmax function representing the probability distribution over n different classes. This is the most common use of Softmax functions. Less commonly, softmax functions can be used in the model itself when the model needs to choose one of n possible alternatives for an internal variable.

Such use of Softmax functions is extremely rare. Since the maximum probability is a consistent estimator, it is bound to happen if the model family can accurately model the distribution on which it is trained. In reality, the model can only predict these rates as the amount of data it can store is limited and the optimization process is not error-free. The softmax function does not work well with most objective functions except the log-probability function. In particular, objective functions that don't use a protocol to cancel softmax Exp cannot learn when the Exp input is too negative, causing the gradient to disappear. Protocols are used to override the effects of exponential functions.

In particular, the square of the error is a weak loss function for softmax units and may not train the model to adjust its output, even if the model gives fairly reliable erroneous predictions. This can happen even if the model has a high degree of confidence in the accuracy of its predictions. (Bridge, 1990). Examining the softmax function itself is necessary if we want to understand why these other loss functions might fail. Softmax activation, like the sigmoid, has the potential to reach saturation. When the input of the sigmoid function is extremely negative or extremely positive, the function's single output reaches its maximum value. There are many different values that can be generated when using Softmax. These output values can become saturated if the deviations between the input values are too large. When softmax reaches saturation, many cost functions based on softmax will also be saturated, unless these cost functions can reverse the saturation trigger function.

6.3 HIDDEN UNITS

We focused on neural network design decisions typical of most parametric machine learning models trained using gradient-based optimization. We will now discuss a unique problem of feedforward neural networks, namely how to decide which type of hidden unit to apply in the hidden layers of the model. This is a problem specific to feedforward neural networks. Hidden unit design is a very active area of study, but there are not yet many established theoretical ideas in this area that can serve as a guide. Linear rectified units are a great option to use as your choice of standard concealed units. A wide variety of additional hidden volume types are available. It is not always easy to decide which one to use in a given situation. (although corrected linear units are usually an acceptable choice).

In this section, we explain some of the basic ideas behind the different types of hidden volumes. This instinctive feeling can help determine when to challenge each of these components. In most cases, it is not possible to predetermine which strategy will be more effective. The process of designing anything involves a lot of trial and error, having an intuitive idea that some kind of hidden driver might work well, building a network with that particular type of hidden driver, and then evaluating its effectiveness against a validation set.

Using this method, a linear convex function can be learned from one Maxout unit to k pieces. Therefore, studying Maxout units can be seen as learning the activation function itself, rather than learning the link between units. A Maxout unit can learn to approximate any convex function with arbitrary precision if k is large enough and the learning threshold is high enough. Specifically, a two-part Maxout layer can learn to implement a completely different function, or learn to perform the same input function x, from a traditional layer using a linearly rectified activation function, an absolute value smoothed function, a leaky or parametric ReLU. as conventional film using the corrected linear activation function.

Alternatively, he can learn to perform the function of a normal diaper. Even in cases where maxout learns to implement the same x function as one of the other layer types, the learning dynamics are different because the maxout layer is parameterized differently from each of these other layer types. This is because the Maxout level has more hidden units. Since each maximum output unit is now parameterized by k weight vectors instead of just one, maximum output units often require more editing than smoothed linear units. If the transmission group is large and the number of parts making up each unit is kept to a minimum, they can work well without being organized. (Cai et al., 2013).

There are some additional benefits offered by Max Units. If certain conditions are met, certain statistical and computational advantages can be achieved by using fewer parameters. In particular, if the features recorded by n different linear filters can be added without loss of information by taking the maximum value of each set of k features, the next layer can use k times less weight. In fact, you can use the knowledge obtained at the previous level. Because each unit is controlled by multiple filters, Maxout units have a degree of redundancy that allows them to avoid a phenomenon known as catastrophic forgetfulness. This is a situation where neural networks forget how to perform the tasks they were trained to do in the past. The redundancy helps Maxout units resist this phenomenon. (Goodfellow et al., 2014a).

Flattened linear units and all its generalizations are based on the idea that models are easier to optimize when their behavior is more linear. This principle underlies all these generalizations of corrected linear units. This general idea that utilizing linear behavior to achieve simpler optimization applies not only to deep linear networks, but also to a variety of other scenarios. Learn sequences and generate a set of states and outcomes through recurrent networks that can learn sequences. During their learning, information needs to be transmitted in many time steps, which is fairly easy for some linear computations (some directional derivatives of magnitude close to 1). The Long Short Term Memory (LSTM) architecture is one of the most successful iterative network designs. It transmits information over time, especially by adding a basic type of linear activation.

We are familiar with using sigmoid units as output units that allow us to estimate the probability of a binary variable taking the value 1. Unlike piecewise linear units, sigmoid units are saturated for most ranges of values. In particular, they have high saturation when z is very positive and low saturation when z is very negative. However, they are very sensitive to their input only when z is close to 0. Gradient-based learning can be very difficult for students when there is widespread saturation of sigmoidal units. Precisely for this reason, their use as covert units in predictive networks is not welcome today. If a suitable cost function can reverse the saturation of the sigmoid in the output layer, using them as output units is consistent with using gradient-based learning. This is the case where the output layer has many sigmoid layers.

In contexts other than feedforward networks, sigmoid activation functions are much more commonly used. Recurrent networks, many probabilistic models, and some autoencoders have additional criteria that preclude the use of piecewise linear activation functions and make sigmoidal units more attractive despite saturation problems. In fact, some models and autoencoders have additional requirements that preclude the use of piecewise linear activation functions.

6.4 ARCHITECTURAL DESIGN

Determining the architecture of artificial neural networks is another important design concern for these systems. The overall structure of the network, including the number of components that the network must have and how these components should be related to each other, is called the architecture of the network. Most neural networks are structured as a layered array of unit groups. Most neural network topologies organize these layers in a chain-like structure, and each layer is linked to the previous layer in the chain. That is, the first layer of this structure consists of: When designing a chain architecture, the most important architectural decisions to make are those regarding the depth of the network and the width of each layer. As we will see, a network only needs to have one hidden layer to match the training data. Deeper networks are generally capable of using far fewer units per level, far fewer parameters, and are often generalized to the test set. However, deeper networks sometimes have difficulty optimizing. Experiments with validation set error monitoring are required to discover the best network design for a given task.

6.5 FEATURES AND ADDITIONAL INFORMATION OF THE UNIVERSAL APPROACH

Only linear functions can be represented by a linear model that maps inputs to outputs by multiplying matrices. When applied to linear models, many loss functions present convex optimization problems, which are easier to train than other methods. The advantage of this method is that it simplifies training. Unfortunately, we often need to know more about nonlinear functions. At first glance, it may seem that specializing in a nonlinear function requires constructing a single family of models that correspond to the particular type of nonlinearity we want to master. A picture of a universal approach could be provided by hidden layer feedforward networks, which is an encouraging development.

In particular, the universal approximation theorem (Hornik et al., 1989; Cybenko, 1989) states that there is a predictive network with at least one hidden layer with a linear output layer and an arbitrary d'crush activation function (such as sigmoid). logistic activation function) can be any approximate Borel function measurable with any desired non-zero error amount from one finite-dimensional space to another, provided the network keeps enough hidden units to fit. Derivatives of the forward-looking network can also approximate derivatives of the higher-order function (Hornik et al., 1990. Borel's idea of scalability is beyond the scope of this book, but it is sufficient to note for our discussion that every continuous function is in a closed and bounded subset of Rn).

While the original theorems were originally formulated in units with satisfactory activation functions for both very negative and very positive arguments, universal approximation theorems are also commonly used. has been proven for a wider range of activation functions, including the rectified linear unit used today. While original. Recent data were initially given in units with saturating activation functions for both very negative and very positive arguments, universal approximation theorems (Leshno et al., 1993) The theory of universal approach Because of its importance, we can be confident that a large MLP can accurately represent any function we want to learn, regardless of the unique nature of the function.

However, there is no guarantee that the training algorithm can learn the function in question. Even if the MLP represents the function correctly, learning can fail for two different reasons. First, the learning optimization process may not be able to determine the parameter value corresponding to the expected function. Second, it is possible for the training algorithm to choose the wrong function due to overfitting. You'll remember that the phrase "no free lunch" indicates that there is no machine learning method that is inherently better than any other. Feedforward networks provide a universal system for expressing functions in the sense that, given a function, there is a feedforward network that can approximate the function.

This makes predictive networks a universal function representation system. There is no single procedure to follow for analyzing a training set of several samples and selecting a function to apply to points not in the training set. The universal approximation theorem states that there is a mesh large enough to achieve the desired degree of precision; However, the theory does not specify how large this network would be to achieve this level of accuracy. Barron (1993) provides some limitations on the size of a single-layer mesh required to simulate various functional classes. Unfortunately, in the worst case, you may need an exponential number of hidden drivers (perhaps with a corresponding hidden driver for each input configuration that needs to be distinguished). This is easier to understand in terms of the binary case: in general, choosing one of these binary functions requires O(2n) degrees of freedom because the number of potential binary functions on vectors v ranging from 0 to 1n is 22. one of these functions requires 2n bits.

In summary, a single-layer feedforward network is sufficient to represent any function; however, the level may be too large, in which case the network may not learn enough and may not generalize. When used appropriately, using deeper models has the potential to reduce the amount of generalization errors and the number of units required to accurately describe the function of interest. Some function families can be effectively predicted by an architecture with a depth greater than a given value of d, but the same function families require a much larger model when the depth must be less than or equal to d. In most cases, the flat model requires an exponential number of hidden units relative to the total number of units.

This type of reasoning was introduced for models unlike the continuous and differentiable neural networks originally used for machine learning; However, it was later expanded to these models. Early discoveries concerned the construction of logic gate circuits (Hstad, 1986). Extending similar results to linear thresholding units with non-negative weights, subsequent research showed that flat meshes exhibit properties of universal approximations with a large family of non-polynomial activation functions, including smoothed linear units. However, these results do not answer questions of depth or effectiveness; Instead, they point out that a large enough rectifier network can represent any function. Montufar et al. (2014) functions that can be represented by a deep correction network may require an exponential number of hidden units when implemented in a flat network with a single hidden layer.

More specifically, they showed that functions with a set of exponential regions at lattice depth can be represented by piecewise linear lattices that can be derived from rectifier nonlinearities or Maxout units. Figure 6.5 is a diagram showing how a network using the absolute value setting produces mirror representations of the function computed on a hidden unit based on the input used by that unit. Each stealth driver has its own peculiarity of where to join the input field to produce mirrored responses. (on either side of non-linearity in absolute value). By combining different folding operations, we can create multiple segmented linear regions. different regions may show various regular patterns such as B. Repeating patterns.

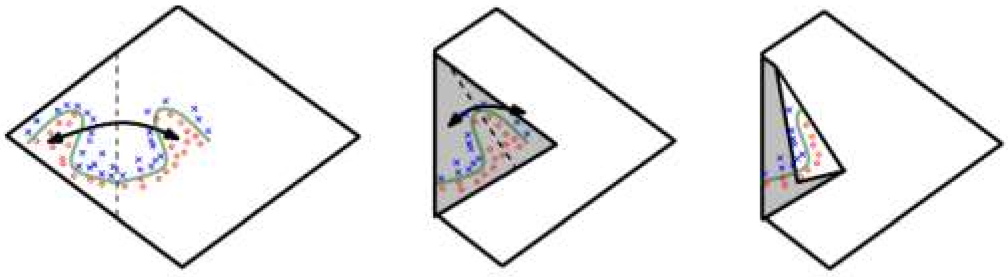


Figure 6.2: An intuitive geometric explanation of the exponential advantage of deeper rectifier networks

Of course, there is no guarantee that the type of functionality we want to learn in machine learning applications (and especially AI) will have such a feature. Indeed, there is no guarantee that machine learning will be used for artificial intelligence. We may also want to use a deep model for statistical reasons. When we choose a particular machine learning algorithm, we implicitly indicate a set of previous ideas we have about the type of function that the algorithm needs to learn. By choosing a deep model, we encode the extremely broad view that the function we want to learn must require the synthesis of many other fundamental functions.

This can be interpreted from a representative learning perspective to mean that we believe the problem with learning is to find a set of underlying variational factors, which can be explained by simpler underlying variational factors. This can be said because we believe the learning problem is to discover a set of underlying drivers of variation that can be identified by other, simpler drivers underlying variation. Alternatively, we can interpret the use of Deep Architecture as an expression of the belief that the function we want to learn is a computer program consisting of several steps, where each step uses the output of the previous step.

This interpretation allows us to view the use of Deep Architecture as an expression of the belief that the function we are trying to learn is a computer program. These intermediate results do not always represent factors of variation; Rather, they can be like counters or beacons that the network uses to regulate the transactions taking place within the network. Indeed, based on the available evidence, deeper understanding seems to lead to better generalization across a wide range of tasks.

6.6 OTHER ARCHITECTURAL MATTERS

So far we have talked about neural networks as simple layer chains, where the depth of the network and the width of each layer are the most important factors to consider. In reality, neural networks exhibit a much wider range of behavior. There are many neural network topologies designed for different tasks. Convolutional networks are specialized architectural frameworks for use in computer vision and . Also, feed-forward networks can be extended to recurrent neural networks to process sequences, Recurrent neural networks have their own architectural problems and feed-forward networks can take advantage of them. Another important aspect to consider when designing architecture is how the different levels are linked. Each input unit is associated with each output unit at the default level of a neural network characterized by a linear transformation via a W matrix. This level is called the default level.

The next few sections cover various private networks, each with fewer connections than the previous. Therefore, each input layer unit represents only a limited number of output layer units. The number of parameters and calculations required to evaluate the network can be reduced by using various tactics to reduce the number of connections. however, these strategies are often highly problem dependent. For example, the convolutional networks discussed here use certain infrequent models that are very useful for solving computer vision problems. It would be impossible to provide much more specific guidance on building a general neural network in this section. The following sections describe specific architectural methods that have proven themselves in various fields of application.

6.7 BACKpropagation AND OTHER DIFFERENCE ALGORITHMS

If we want to generate an output from an input, we can use a feedforward neural network that allows information to flow forward through the network. The inputs are where the first information is received, which is then transmitted to the hidden entities at each level and finally to the outermost level. The term used for this process is forward propagation. In the training process, forward propagation can continue as long as no scalar costs are incurred. To calculate the gradient, the backpropagation method (Rumelhart et al., 1986a), more commonly referred to in its abbreviated version, allows cost information to flow back over the network.

Calculating an analytical equation for the gradient is not difficult; However, the numerical evaluation of such an expression can be computationally intensive. The backpropagation algorithm achieves this goal easily and inexpensively. It is common for people to mistakenly believe that the term "backpropagation" refers to the entire learning process of multilayer neural networks. In fact, the term "backpropagation" simply refers to the technique used to calculate the gradient. To perform learning using this gradient, another algorithm is needed, such as stochastic gradient descent. Also, many people mistakenly think that backpropagation only applies to multilayer neural networks, but in fact it can theoretically be used to calculate derivatives of any function. (For some functions, the correct answer is to state that the derivative of the function is undefined).

6.8 CALCULATION CHARTS

So far we have approached neural networks with a somewhat looser graphical language. It would be helpful to have a more accurate graph computation language to provide a more accurate description of the backpropagation technique. It is possible to formalize the calculation using a variety of different graphs. In this case, we refer to each node of the graph as a different variable. It is possible that the variable in question is a scalar, vector, matrix, tensor, or perhaps some other type of variable. To make our charts more formal, we also need to introduce the concept of operations. An operation is a simple function that can use any number of variables. Our chart language is complemented by a list of different operations that can be performed on charts. It can be used to combine many operations, expressing more complex functions than the operations in this collection. To maintain universal applicability of the operation, we define it to always return a single output variable.

This does not cause any loss of generality, because the output variable can hold many inputs, similar to a vector. Backpropagation is often implemented in software to enable actions with many outputs. However, we will not cover this scenario in our description because it contains a lot of additional information that is not necessary to understand the idea. When you determine the value of a variable y by operating on the value of an x variable, you draw an arc in the x direction and into y. We note the name of the action that is occasionally applied to the output node, and sometimes we ignore this tag when the context makes it clear what action was taken.

The Calculus chain rule (not to be confused with the probability chain rule) is used to calculate derivatives of functions produced by combining other functions whose derivatives are already known. This rule is used to calculate derivatives of functions formed by combining other functions whose derivatives are known. The chain rule can be calculated using an algorithm called backpropagation, which follows a certain sequence of operations that is known to be very efficient.

6.9 REPEAT APPLY THE CHAIN RULE TO GET BACKPROP

Using the chain rule, it is easy to construct an algebraic equation for a scalar gradient with respect to each node in the computational network that creates a scalar. This expression is easy to write. However, when this set is actually evaluated on a computer, there are a few new factors to consider. More specifically, many different subexpressions can potentially appear multiple times within the larger expression that defines the gradient. Each operation that calculates the gradient must decide whether to store these subexpressions temporarily or to calculate them repeatedly from scratch. Figure 6.2 shows how these repeated subexpressions are structured.

Under certain circumstances, evaluating the same subexpression twice would be an unnecessary and inefficient use of time. A naive application of the chain rule is not always possible because of the potential for exponentially large numbers of unnecessary computations when dealing with complex graphs. Under certain circumstances, processing the same subexpression twice may be an acceptable strategy to reduce memory usage, but this strategy may be at the expense of longer execution times. We begin by developing a version of the backpropagation algorithm that explicitly defines the actual gradient computation for the corresponding direct computation, in the order in which it is actually performed, and according to the iterative application of the chain rule.

You can perform these calculations directly or view the method description as a symbolic description of the computational mesh to calculate backpropagation. In any case, one of these two approaches will apply. However, the manipulation and creation of the symbolic graph that performs the gradient computation is not explained by this description. This formulation is recommended in conjunction with Method 6.5 later in the Text. In this section, we also generalize to nodes containing arbitrary tensors. Therefore, backpropagation inhibits the exponential growth that occurs with repeated subexpressions. However, some algorithms can save memory by recalculating instead of storing certain subexpressions, or avoid more subexpressions by simplifying the computation graph. Also, other algorithms can avoid multiple subexpressions by making simplifications to the diagram.

Now that we've discussed the backpropagation method, let's get back to these concepts. as part of the implementation of a computational network for derivatives. Each subset of the graph can then be evaluated by applying certain numerical values to the evaluation results. This saves us from having to define exactly when which action should be calculated. If it is a generic graph evaluation engine, it can evaluate each node as soon as its parent's values are reached. The explanation of the symbol-number technique is included in the definition of the symbol-symbol approach. One way to understand the symbol-to-number technique is to imagine doing exactly the same calculations performed on the graph created by the symbol-to-symbol approach. The main difference is that the strategy that converts symbols to numbers does not display the chart.

6.10 BACKSPREAD GENERAL

The backpropagation process is relatively easy to understand. To calculate the gradient of a scalar z with respect to x from its predecessors in the graph, we start by saying that the gradient with respect to z is dz dz = 1, then we continue to calculate the gradient of a scalar z. x according to one of his ancestors. The gradient with respect to each parent of z in the graph can be calculated by multiplying the current gradient by the Jacobian of the operation that produced z. This allows us to determine the gradient relative to each parent of z in the graph. To get to X, we'll continue to multiply by the Jacobian numbers moving backwards on the graph. Simply put, if there is a node returning from z two or more paths, we need to sum all the gradients coming from these separate paths to that node. Technically speaking, each node of the G graph represents a different variable. Let's call this variable tensor V to cover as many cases as possible.

In general, a tensor can have any number of dimensions. This includes scalars, vectors, and matrices. Even if it doesn't , the op.bprop method should behave as if each of its inputs is unique from the others, even if it isn't. For example, if the mul operator is given two copies of x to calculate x2, the op.bprop function should always return x as the derivative with respect to both inputs. This is because mul works on multiple x copies. After some time, the backpropagation algorithm adds these two parameters to get the result 2x, which is the exact sum derivative based on x.

Users of the deep learning software library can learn matrix multiplication, exponentiation and logarithms etc. This allows users to propagate backwards between charts. Developers creating a new backpropagation application, or power users who need to add their operations to an existing library, usually have to manually derive the op.bprop method for each new operation. This is the case when they create a new backpropagation version or add their own process to an existing library.

Our explanation of the backpropagation method here is much simpler than in real world applications. As we said earlier, the concept of operation is reduced to a function that never returns a single tensor. The vast majority of software applications require support for operations that can return multiple tensors. It is preferable to compute both in a single buffer, so it is more efficient to implement this method in a single operation with two outputs. For example, if we want to calculate both the maximum value of a tensor and the index of that value, it is best to calculate both in a single memory pass. Back propagation memory usage can be handled more easily, but we haven't talked about that yet.

Backpropagation usually requires combining a large number of tensors. With the naive method, each of these tensors is first calculated separately and then combined in a second pass. Keeping a single buffer during computation and adding each value to it is a way to get around the memory bottleneck caused by the naive method with an excessively large capacity. For backpropagation to be useful in real-world applications, it must be able to handle a variety of data types. These data types include 32-bit floating-point, 64-bit floating-point, and integer values. Developing a strategy to address each of these categories requires extra attention and care. It is important to keep these situations in mind and to decide whether the slope requested by the user is defined.

Some actions have undefined gradients and it is important to pay attention to this. The differentiation of real-world concepts is complicated by a series of additional challenges. These technical details are not insurmountable and this chapter has covered the basic conceptual tools needed to calculate derivatives; However, it is important to know that there are many more nuances than those mentioned.

6.11 MAKING DIFFERENCE OUTSIDE THE DEEP LEARNING COMMUNITY

As the deep learning community has somewhat separated itself from the broader computing community, it has often formed its own cultural views on how to differentiate. These attitudes are mainly influenced by the history of the deep learning community. More broadly, the argument from automatic derivative concerns the question of how derivatives can be calculated computationally. There are several ways to perform automatic differentiation; The backpropagation method is just one of them. It is a subcategory of a more general category of methods collectively called inverse mode accumulation. Various methodologies parse the constituent statements of the chain rule in several different sequences.

In general, determining the order of evaluation that requires the least computation required is a difficult task. Finding the best sequence of operations for calculating the gradient is NP-complete (Naumann, 2008), which means that algebraic expressions may need to be reduced to their simplest and cheapest form. If this is the case, the naive approach may require k times as many operations. This is because the naive method calculates k gradients instead of a single gradient for each internal scalar node in the original forward graph. When the number of graphics outputs is greater than the number of inputs, it is often advantageous to use another type of automatic derivation known as direct mode accumulation.

This is when the number of outputs in the graph is greater than the number of inputs. Real-time calculation of gradients in recurrent networks can be obtained, for example, using the proposed direct mode calculation. (Williams and Zipser, 1989). This eliminates the need to preserve values and gradients for the entire graph, resulting in a reduction in computational efficiency gained at the expense of memory. For example, the relationship between forward mode and reverse mode can be compared to the relationship between left multiplication and right multiplication of a set of matrices.

It is increasingly common in many non-machine learning communities to create differentiator software that runs directly on the code of traditional programming languages such as Python or C, and automatically creates programs that differentiate functions written in those languages. An example of such a community is artificial neural networks used in machine learning. In the field of deep learning, computer graphics are often represented by open data structures created by special libraries. The custom approach has the problem that the developer of the library has to specify the bprop methods for each action, so the user of the library is limited to being able to perform the operations described earlier.

However, the proprietary technique also has the advantage of allowing unique backpropagation rules to be created for each transaction. This allows the developer to increase speed or stability in non-obvious ways that an automated process is unlikely to emulate. Another benefit of the custom approach is that it allows the development of a more robust algorithm. It is clear that backpropagation is neither the only nor the best way to calculate the gradient; However, it is a very practical approach that continues to serve the deep learning community very well. Backpropagation has been around for a long time. It is possible that deep network differentiation technology will evolve in the not-too-distant future as deep learning professionals become more familiar with developments in the wider field of automated differentiation.

In the 17th century, Leibniz and L'Hôpital independently proposed the chain rule, the basic concept behind the back propagation algorithm. Analysis and algebra have been around for a long time, but gradient descent was not developed as a method for repeatedly estimating the solution of optimization problems until the 19th century. Analysis and algebra have long been used to answer closed-form optimization problems. (Cauchy, 1847). With these function approximation methods, the development of machine learning models such as the 1940's sensor was advanced, while linear models formed the basis of the first models. Pointing to many problems with the family of linear models, critics such as Marvin Minsky sparked a backlash against engineering global neural networks.

One such error was that the linear model family could not learn the XOR function. This led to the setback. Learning the nonlinear functions required the creation of a method for estimating the gradient using a model with such a device as well as a multilayer perceptron. In the 1960s and 1970s , efficient applications of the chain rule based on dynamic programming began to emerge, particularly for control applications (Kelley, 1960; Bryson and Denham, 1961; Dreyfus, 1962; Bryson and Ho 1969; Dreyfus, 1973). also for sensitivity analysis (Kelley, 1960; Bryson and Denham, 1961; Dreyfus, 1973). (Linnainmaa, 1976). Werbos (1981) suggested that similar methods could be used to train neural networks. After several reinventions, the concept was finally implemented and further developed. (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986a).

The results of some of the first successful backpropagation experiments are presented in a chapter of Parallel Distributed Processing (Rumelhart et al., 1986b). This chapter was instrumental in popularizing backpropagation and ushering in a very active period of research in multilayer neural networks. Parallel Distributed Processing was published in 1986. However, the concepts presented in this book, and especially those of Rumelhart and Hinton, go far beyond the concept of backpropagation. They contain important insights into the possible computational application of various central aspects of cognition and learning known as "connectivity" because this school of thought emphasizes the connections between neurons as areas of learning and memory.

In addition, the name "connectivity" originated from the fact that the connections between neurons were called "connectivity". One such concept is specifically the idea of distributed representation. (Hinton et al., 1986). Neural network research became more popular with the use of the back propagation algorithm and reached its peak in the early 1990s. After that, various machine learning approaches gradually gained popularity until the current deep learning renaissance began in 2006. Since the 1980s, there has been no major change in the basic concepts behind today's predictive networks. It still uses the same back propagation technique and gradient descent methods. Most of the progress in neural network performance between 1986 and 2015 can be attributed to two different sources.

First, the availability of larger datasets has reduced the extent to which neural networks struggle with the problem of statistical generalization. Second, the size of neural networks has increased significantly due to the development of more powerful processors and better software infrastructure. However, few algorithmic changes have led to significant improvements in neural network performance. One such algorithmic improvement was to replace the previously used mean squared error metric with a family of cross-entropy loss functions. The mean square error was an important concept in the 1980s and 1990s, but has since been replaced by cross-entropy leaks and the maximum likelihood principle as ideas have moved from the statistician to the statistics community to machine learning.

When using Root Mean Squared Error Loss, the sigmoid and softmax output models have traditionally struggled with saturation and slow learning; However, the application of cross-entropy losses has led to a significant improvement in the performance of such models. Replacing sigmoidal hidden units with piecewise linear hidden units such as B. rectified linear units was the second major algorithmic change that contributed significantly to the significant performance improvement brought by feedforward networks. Early neural network models such as Cognitron and Neocognitron were the first to implement correction using the max0z function.

This practice dates back at least to that time. (Fukushima, 1975, 1980). These early models did not use corrected linear units; Instead, they applied the correction to nonlinear functions. It is possible for sigmoids to work best when neural networks are relatively small, so although the fix was initially quite popular, the fix was largely replaced by sigmoids in the 1980s. Corrected linear drivers fell out of favor in the early 2000s and were eventually phased out entirely, due to the somewhat superstitious assumption that activation functions with non-differentiable points should be avoided at all costs.

This started to change around 2009. Among the many aspects of building neural network architecture, Jarrett and colleagues (2009) found that "the use of corrective nonlinearity is the single most important factor in improving the performance of a discovery." Factors Jarrett et al. (2009) concluded that using nonlinear correction for analysis of small datasets is even more critical than learning the hidden layer weights. classification layer on top of the network to learn how different feature vectors match their class IDs. As more data is available, the learning begins to extract enough usable information to handle the randomly set parameters. n is much simpler in smoothed deep linear networks than in deep networks that exhibit bilateral curvature or saturation in activation functions, as reported by Glorot et al. in the 2011a study.

Corrected linear units are also of historical importance as they show that neuroscience continues to influence the development of deep learning algorithms. This is one of the reasons why these units are interesting. Biological causes have been suggested by Glorot et al. (2011a) to justify the use of corrected linear units. Semi-corrected nonlinearity was developed to simulate the following properties of biological neurons: 1) Biological neurons can be completely inactive in response to certain stimuli. 2) The output of a biological neuron is proportional to some of its inputs under certain conditions. 3) Biological neurons spend most of their time working in a resting regime. (i.e. they need to have infrequent activations). 2006 saw a resurgence of contemporary deep learning, but feedforward networks still had a bad image.

Between 2006 and 2012 there was a common misconception that feedforward networks could work without the help of other models such as B. Probability models were not getting satisfactory results. It is common knowledge today that feedforward networks work quite well, given the right resources and technical approaches. Gradient-based learning in feedforward networks is a method used today to create probabilistic models such as Variable Autoencoder and Generative Anti-Networks described in This tool is used to build these models. Gradient-based learning in feedforward networks has been recognized since 2012 as a powerful technology that can be used for a wide variety of machine learning tasks. Before this year, this type of learning was viewed as an unstable technology that needed to be supplemented by other methods.

Ironically, while in 2006 the community used supervised learning to support unsupervised learning, it has now become more common to use unsupervised learning to support supervised learning. In 2006 the community used unsupervised learning to support supervised learning. To date, the potential of feedforward networks has not yet been exhausted. Going forward, we hope that they will become accustomed to a much wider range of problems and that improvements in optimization methods and model design will lead to even more significant improvements in their overall performance. The family of neural network models is discussed in detail in this section. In the next sections, we'll see how to use these models, specifically how to tune and train them.

6.12 LAST REGISTRATION

There is no need to explain the steps of the Gaussian filter technique in detail, as they show how the modularity of the bootstrap method can improve a CNN classifier, as they show how the combination of the bootstrap algorithm and the Gaussian filter approach produces the result. an answer to our research object. Given the level of blurriness and inaccuracy in the elevation data in some of the 2016 IARC photographs, we find the final result positive if it can accurately identify all the robots in the images. . 83% of the time. After a supervised review and dataset review, we now have a large and reliable dataset. Indeed, the dataset consists of many instances of bot detection, multiple degrees of congestion, and multiple dimensions.

In this section, we will discuss the methodologies that lead to answering the second research question: how do we reliably perform real-time object detection in an embedded system with potentially limited processing?

6.13 NETWORK TRAINING

Colorless SSD arrays required surprisingly simple training procedures. We can see that the loss decreases as the verification accuracy increases until it reaches a point where it is stable. Both SSD300 and SSD512 managed to approach solutions with higher final mAP values than those described in the article. This was true for both algorithms. This is because the difficulty of recognizing robots in a defined context is much easier than the problem of object recognition in general. We also found that the effect of input size was much more pronounced in robot recognition than in general recognition. In robot detection, the mAP difference between SSD300 and SSD512 is 6.8 points, but the mAP difference between each of the generic detection versions is only 2.5 points.

This is because some bots are smaller and much more likely with a larger input size. Because items in normalized tracking datasets are usually not very small, the 2.5-point improvement is due to much more accurate feature maps than the ability to identify more specific items. The formation of the colored versions of the mesh showed behavior completely comparable to those of the colorless formation; However, the network needed a few more iterations to set up properly. Considering that the network converges easily to solutions with high mAP values, the inclusion of the two additional classes does not seem to have made the localization process much more difficult. However, a decrease in mAP is observed when comparing the unpainted versions with the stained versions; This will be explored in more detail when we look at the network test results.

It took many cycles to train the SSD420 array, but the end result was 85.16 mAP points, just 2.7 points lower than the SSD512 array. Depending on how fast it goes, this has the potential to be a strong contender for the final answer. It seems that YOLOv2 is unable to perform a validation procedure while it is being trained, making it very difficult to assess if it is starting to overfit. Due to the late discovery and implementation of YOLOv2, we were unable to develop our own validation technique due to time constraints. It also did not have a simple tool for calculating the map. Therefore, we have no choice but to believe that the detection dataset for YOLOv2 is large enough, like SSD, and hope this prevents overfitting.

6.14 NETWORK TEST

All evaluated networks show remarkable recognition results on the validation set and are apparently able to generalize their performance on the training set. Everyone has trouble recognizing robots far from the drone due to the poor resolution and sometimes blurry appearance of the smaller robots, but other than that, the networks work fine. When locating the robot, the detections in the colored and colorless variants are almost indistinguishable from each other. The color version, on the other hand, cannot detect robots in the floor model, but other than that, there isn't much difference between the two. The detections of the colored versions appear to be mislabeled rather than misdiagnosed, which explains the slightly lower mAP the colored versions have compared to their non-colored equivalents.

Due to the depressing lack of support for SSD optimization in the TX2, we use frame-per-second performance on the Jetson TX1 when comparing different runtimes. This decision was made because of the TX1's superior performance. With work ahead of the 2017 IARC competition, this optimization challenge will be a top priority. The SSD300's output is mostly accurate, but its detections tend to drop a frame or two seemingly randomly, causing the detections to flicker in the output video. Most of the time the output is correct. The biggest problem with this is that bad results can occur when the network is running at a slow refresh rate such as: B. the same loss is taken for multiple consecutive frames on the same robot. The low refresh rate then renders the robot invisible to the rest of the Ascend system for a time, which can lead to crashes if the undetected robot is stuck on a pole at that time.

Even if collisions are unlikely due to SSD300 errors, this would conflict with the research goal of providing a reliable solution in a timely manner, as detections appear to be more reliable the closer they are to the drone. . SSD300's speed is one of its advantages; However, the amount of error it produces, especially for robots with limited range of motion, makes it unreliable as a definitive solution. SSD512 verification video shows much more consistent inter-frame detection than SSD300. It's pretty rare for it to lose tracking more than a frame or two, and it will usually detect farther robots with a significantly higher degree of accuracy. The source of the problem is the low FPS produced by the SSD512. You can shoot at 3.5 frames per second on the TX1; this is not a bad speed for such an accurate recognition result.

However, faster detection always means more up-to-date environmental status, and we're happy to sacrifice some accuracy for a much faster solution. The 420x420 array is the final SSD size covered in this discussion. The verification video and mAP score show that the network is almost as accurate as SSD512 but could run slightly faster. Most errors result from not detecting the robots at great distances and mislabeling the robots for one or two frames at a time. Of the three different size solid state drives (SSDs) evaluated for this, we can conclude that the SSD420 is the one that best answers our research question because it has the most desirable combination of accuracy and speed. Finally, we turn our attention to YOLOv2.

Watching the verification video, it immediately becomes clear that the bots are not as accurate as SSD in determining whether it is blurry. Can identify most types of robots; However, beds prefer certain spots in the frame and seem to stay there even as the element slowly moves away from them. When the robot has traveled a sufficient distance, the bounding box will perform a jumping motion to hold the robot again. Overall the detection is satisfactory and the robot can determine their positions more accurately than the SSD300, but has more difficulty distinguishing detections that are too close together, as shown in Figure 4.15.(e).

Compared to the SSD420, this is a little further from our research target, because while it is faster, it is less reliable than we would like. If the TX2 optimization issues cannot be resolved before the 2017 IARC competition, we will have to do one of the following: Run YOLOv2 on TX2s or replace the two Jetson TX2 boards used to detect Ascend onboard -Drone uses TX1 boards and runs at 4fps' each de SSD420 array. The other option is to just run YOLOv2. It will be very useful to run the SSD420 network, which we think is much more reliable than the YOLOv2 network, and switch to TX1. It would be more advantageous to use TX1s as the drone has two Jetsons dedicated to the detection network. While we hope this isn't really the case, this is the best option as of the last business day.

The dlib library discussed in . While this certainly wasn't better than the approach currently used, it could have produced excellent results and required far less preparation. The dlib technique does not require height or camera angle to find robots of different sizes. Therefore, the dlib method can give more accurate results than the current solution. A supervised adjustment was made, although not necessary, as the data collected was not accurate enough for the drone's altitude and camera angle.

6.15 NOT MAXIMUM CANCELLATION

Non-maximum suppression, commonly known as NMS, is a technique that produces the single detection bounding box that is most likely to be correct after finding the local maximums of multiple overlapping detection bounding boxes. There are several strategies that can be used to solve NMS, some of which are simple greedy local optimization methods while others are more thorough and comprehensive. NMS approach capable of determining the best local maximum by evaluating every possible subsampling present in a region. This method was developed in 1980. The SSD selects the most likely detection from a large number of overlapping bounding boxes using a method known as non-maximal suppression at many scales. This could have been tried to use similarly to extract position data from the floating window technique. Another option would be to use it differently. Using this technique is likely to be an easier way to solve the localization problem.

CHAPTER 7

REGULARIZATION FOR DEEP LEARNING

Developing an algorithm that works efficiently not only with data used for training but also with new data is one of the biggest challenges in machine learning. There are many different tactics in the field of machine learning that explicitly aim to reduce test errors, sometimes at the expense of higher training errors. Regularization is the umbrella term for all these different techniques. As we will see, there are many types of regularization that the deep learning practitioner can choose from. Indeed, one of the main directions of work in this area has been the development of more efficient regularization methods.

Chapter 5 introduced the reader to the basic ideas of generalization, underfitting, overfitting, bias, variance, and regularization. Before moving on to this section, you should read the previous one and familiarize yourself with these concepts if you are new. In this section, we will provide a more detailed description of regularization by focusing on deep pattern regularization techniques or patterns that can be used as building blocks for creating deep patterns. In this section, we'll discuss a few key ideas that are crucial to machine learning. If you've used these ideas before, you can skip the relevant sections.

On the other hand, much of this chapter is devoted to discussing how these basic concepts can be applied to the specific context of neural networks. In section 5.2.2, we defined regularization as "any change made to a learning algorithm that aims to reduce generalization error, not learning error". In other words, regularization is any change made to a learning algorithm to reduce the generalization error. There are many different approaches to regularization. Some people impose additional constraints on a machine learning model; B. Restrictions on the values that can be used for parameters. Some people add additional terms to the objective function that can be loosely interpreted as equivalent to the limiting parameter values.

If chosen carefully, these additional restrictions and penalties have the potential to help improve test set performance. Sometimes the purpose of these restrictions and penalties is to encode some kind of prior knowledge. Sometimes these restrictions and penalties are to instill a general preference for a simpler class of models to encourage generalization. It does this by modeling it to reflect a preference for a simpler model class. Sometimes sanctions and restraints need to be used to find a solution to an unidentified problem. The term "batch methods" refers to a different type of arrangement that combines many assumptions into a single model to explain training data. The vast majority of regularization algorithms used in deep learning are based on the concept of regularization predictors.

The process of regularizing an estimator involves trading more bias for less variance to keep it working. An effective regulator is one that yields by significantly reducing variance and slightly increasing the amount of biased gain. When we talked about generalization and overfitting, we focused on three scenarios: the family of models being trained did not match the actual data generation process, which corresponded to underfitting and causing bias; the model family exactly matched the actual data generation process; The family of models or models included the production process, as well as many other possible production processes, corresponding to the overfit regime, where the estimation error was dominated by variance rather than bias.

In all three scenarios, the estimation error was: The purpose of the normalization process is to transfer a model from the third regime to the second regime. In reality, an extremely complex family of models may not contain the desired function or the actual data generation process, or even a close approximation of both. Since we have little access to the actual process that generated the data, we cannot be sure whether the predicted model family includes the manufacturing process. However, the vast majority of applications of deep learning algorithms take place in areas where the actual data generation process is unlikely to be part of the model family.

Deep learning techniques are typically used in extremely difficult areas such as images, sound sequences and writing, where the real creative process is to effectively model the entire universe. These fields include: images, sound clips, and text. In a sense, we are constantly working towards the goal of pressing a square needle (the data creation process) into a round hole. (our model family). This indicates that constructing an appropriately sized model with the appropriate number of parameters is not the only way in which model complexity can be controlled. Instead, we might find that the model that best fits the data (in terms of generalization error reduction) is a properly regularized giant model - and indeed, when we use deep learning in the real world, we almost always find it. This is because large models are easier to use for deep learning. We will now explore the different approaches that can be used to construct such a broad, deep and regulated model.

7.1 PENALTY AGAINST PARAMETER STANDARDS

Editing was already common a few decades before deep learning was developed. Linear models such as linear regression and logistic regression allow for simple, straightforward and highly efficient editing procedures. Before looking at how different norms perform regularization, it's important to remember that a parameter norm penalty is commonly used when dealing with neural networks. This standard only penalizes the affine conversion weights in each layer and does not affect deviations in any way. Compared to weights, setting offset usually requires less data. Each weight describes how the two variables relate to each other. To get good weight regulation, both variables need to be monitored under a variety of different conditions. Every bias has a single variable that can affect it.

This indicates that we did not create excessive variation as the perturbations were not regulated. A high degree of underfitting can also be achieved by smoothing the bias parameters. Therefore, we use the vector w to indicate all the weights that should be affected by a standard penalty, while the vector specifies all parameters, including w and the uncontrolled parameters. w is used to indicate all weights that should be affected by a standard penalty. In the context of neural networks, it is often advantageous to use a separate penalty with different coefficients for each layer of the network. In fact, different layers of the network have different levels of complexity. While it is costly to find the appropriate value for many hyperparameters, it is always fair to assign the same weight loss to all levels. This is done to reduce the amount of search space to be searched.

Solid ellipses show unadjusted target contours with the same value. The dashed circles are intended to indicate contours of the L2 editor with the same value. These exchanges reach a state of equilibrium when they meet in the middle at the w point. The Hessian eigenvalue of J is not very large and the first dimension is where it comes in. The objective function does not increase significantly as you move away from w horizontally. The modifier has a significant influence on this axis because the objective function does not show a strong preference in this direction.

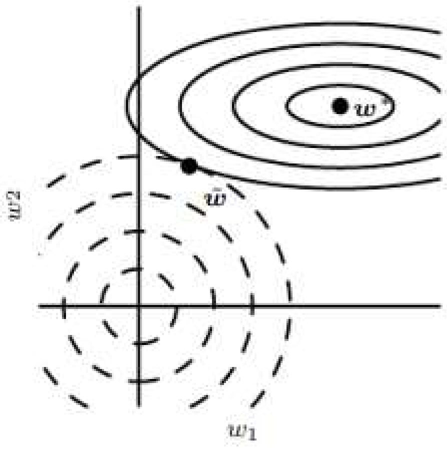


Figure 7.1: An example of the effect of Ofl2 regulation (or weight loss) on optimum W value

The modifier drives w1 very close to zero. The objective function has high sensitivity to movements going in the opposite direction to w in the second dimension. The corresponding eigenvalue is high, indicating that the curve is very sloping. As a direct result, the effect of weight loss on the position of w2 is negligible. Only those aspects where the parameters contribute significantly to reducing the objective function are kept reasonably intact. This limits the scope of storage. Moving in a direction with a small Hessian eigenvalue tells us that moving in that direction will not contribute significantly to increasing the gradient.

This is the case of aspects that do not allow reduction of the objective function. The use of the regularization technique in the learning process causes the components of the weight vector corresponding to these irrelevant directions to lose their meaning gradually. So far, we've talked about weight loss in terms of its effect on optimizing a large abstract quadratic cost function. What exactly is the connection between these effects and machine learning in particular? Examining linear regression, a model in which the efficient cost function is quadratic and therefore suitable for the same type of analysis we have done so far, will enable us to find the answer to this question.

If we run the analysis again, we might get a specific sample of the same results, but this time the answer is expressed in terms of the data used for training. The cost function is defined as in linear regression. For another reason, it is recommended to use explicit constraints and reprojections rather than the imposition of sanctions. Penalties have the potential to hinder non-convex optimization techniques at local minimums corresponding to small values of the constraint. This usually occurs when neural networks are trained while neural networks are trained with a set of "dead units". These are entities that do not significantly contribute to the operation of the learned network function. This is because the weights going in or out of these units are all extremely low. While it is possible to drastically reduce the J by making the weights much heavier, these setups can still be locally optimal when training at the weight norm with a penalty.

This is because the weight standard is penalized. Explicit constraints imposed by reprojection have the potential to work much better in such cases, as they do not encourage weights to approach the origin. Only then can the explicit constraints imposed by the reprojection take effect if the weights get big enough and try to get out of the constraint region. Finally, reprojective explicit constraints can be useful as they add some stability to the optimization approach.

This provides greater precision in the final result. When using high learning rates it is possible to enter a positive feedback loop where high weights cause high gradients followed by a large update to the weights. This continues until the loop is stopped. If these updates cause the amount of weight to increase again, it quickly moves away from the origin until it reaches a point where the numbers overflow. This feedback loop is prevented from further increasing the size of the unbound weights by using explicit constraints in conjunction with reprojection. Rather than limiting the Frobenius norm to an integer weight matrix, Hinton et al. (2012c) advocate the use of constraints with a fast learning rate to allow rapid exploration of the parameter space while maintaining some stability. a neural network layer.

By limiting the norm of each column separately, we eliminate the possibility of a hidden unit having very large weights. If we convert this constraint to a penalty in a Lagrangian, the result would be very similar to reducing the L2 weights, but a separate KKT multiplier is applied to the weight of each hidden unit. Each of these KKT multipliers undergoes a separate dynamic update to ensure all hidden units are within the limit. In practical applications, the column norm constraint is enforced as an explicit constraint, almost often using reprojection.

7.2 CONTROL AND SUB-RESTRICTION PROBLEMS

Some linear problems have solutions expressed in closed form if the corresponding matrix is invertible. Even if there is no closed solution to the problem, it is also conceivable that a problem has not been adequately identified. An example would be to use logistic regression in a situation where classes can be linearly separated. If a weight vector w achieves a perfect classification, the probability of 2w getting a perfect classification increases. An iterative optimization process such as stochastic gradient descent will continually increase the magnitude of w and in principle never stop completely. In practice, a numerical implementation of gradient descent will eventually reach weights large enough to cause a numerical overflow. At this point, the behavior of the algorithm is determined by how the programmer handles unreal values.

In other words, the behavior depends on the programmer's decision. Many of the different types of regularization can converge iterative techniques when applied to problems with ill-defined solutions. For example, reducing the weight causes the gradient to stop decreasing by increasing the magnitude of the weights when the slope of the probability is equal to the weight reduction coefficient. This happens when the slope of the probability is equal to the coefficient of distortion of the weight. The concept of streamlining apps to find answers to unanswered questions isn't limited to machine learning. The same basic technique can be used to solve some other basic linear algebra problems.

7.3 RECORD INCREASE

Training a machine learning model on more data is the most effective technique for improving its ability to generalize results. Of course, the amount of data we can access in practice is limited. Generating misinformation and incorporating it into the training package is one strategy to overcome this challenge. Generating false new data for some machine learning related tasks can be done very easily. This classification method is the simplest. For a classifier to be successful, a complex, high-dimensional input denoted by x must be reduced to a singular category ID denoted y. This indicates that the biggest challenge facing a classifier is maintaining its integrity after going through a number of changes. By changing the x entries in our training set, we can quickly and easily create new (x,y) mappings. This strategy is not easily applicable in a variety of different contexts.

For example, generating new bad data for a job involving density estimation is impossible unless you find a solution to the density estimation problem. The process of adding new information to an existing dataset has proven to be a very useful strategy for solving the object detection classification problem. Images are large in size and contain many different features that can be used to create variations; Many of these factors are easy to fake. Operations such as flipping the training images by a few pixels in each direction can significantly improve generalization, even if the model is built partially as translational invariant using the convolution and clustering methods already described in C. This is because of the immutability of the Translation. It refers to the degree to which the output of the model is unaffected by changes in the training data.

Numerous additional methods such as rotating or resizing the image have also proven to be very effective. It is important to avoid applying transformations that could cause the corresponding class to change. Optical character recognition tasks include, for example, distinguishing between the letters "b" and "d" and distinguishing between the numbers "6" and "9"; Therefore, horizontal flips and 180-degree rotations are not suitable techniques for enlarging datasets for such tasks.

There are also modifications where we want our classifiers to be immutable but difficult to achieve. For example, you cannot directly perform a geometry operation on the input pixels to rotate them out of the plane. Besides being useful for speech recognition tasks, register expansion has a number of other uses. (Jaitly and Hinton, 2013). Adding noise to data fed into a neural network (Sietsma and Dow, 1991) is another example of data augmentation that can be used. Even if a small amount of random noise goes into the input, it should still be workable for most classification problems and some regression problems. However, neural networks have proven to be quite fragile in the face of noise. (Tang and Eliasmith, 2010). Using random noise to input when training neural networks is a technique that can be used to make neural networks more resilient. B.

Some unsupervised learning algorithms, such as auto-encoder noise removal, use noise input as a component of the learning process. (Vincent et al., 2008). Noise injection also succeeds when noise is applied to hidden entities, which can be seen as an example of improving the dataset at multiple abstraction levels. Recent research by Poole et al. (2014) have shown that this strategy has the potential to be very effective, provided noise levels are properly regulated. The powerful regularization approach known as abandonment can be seen as a method of generating new inputs through the noise amplification process. When comparing machine learning benchmark results, it's important to consider the impact of growing datasets. In many cases, the generalization error of a machine learning approach can be significantly reduced by using hand-designed dataset expansion strategies. Conducting controlled experiments is imperative when evaluating the effectiveness of one machine learning algorithm over another. When comparing machine learning algorithm A to machine learning algorithm B, it is important to ensure that both algorithms are evaluated using the same strategies to increase the amount of manually generated data. In fact, the comparison between the two algorithms aims to determine which algorithm is superior.

Suppose algorithm A performs poorly when the dataset is not expanded in any way, but algorithm B performs well when it contains a large number of artificial changes in the input. Instead of using machine learning algorithm B, synthetic changes were more likely to be responsible for better performance in this particular scenario. Sometimes a subjective assessment is needed to determine whether an experiment has been adequately controlled. For example, a kind of dataset expansion is performed by machine learning algorithms when they add noise to the data they receive as input. In most cases, general operations (such as adding Gaussian noise to the input) are considered part of the machine learning method, while application domain-specific actions (such as randomly cropping an image). be independent preprocessing steps.

7.4 NOISE RESISTANCE

provided the impetus to apply noise to the inputs to achieve the goal of enlarging the dataset. In some models, adding noise to the model input with infinitesimal variation is functionally similar to penalizing standard weights. (Bishop, 1995a,b). It is important to keep in mind that the noise injection process can generally be much more efficient than simply reducing parameter values, especially when noise is introduced in invisible units. Applying noise to hidden drives is an important enough topic to deserve a separate explanation; The deletion algorithm described in section 7.12 is the main evolution of this strategy. Adding noise to model weights is another strategy for regularizing them. The application of this method has been most frequently observed in the field of recurrent neural networks. (Jim et al., 1996; Graves, 2011). This can be seen as a stochastic application of Bayesian inference on weights.

The Bayesian learning approach will take into account that model weights are subject to uncertainties and can be represented using a probability distribution that takes this into account. Including noise in the weights is a pragmatic and stochastic way of expressing this uncertainty. Because examples belonging to the same class have very similar representations. Unsupervised learning can provide useful insights into the organization of data samples in a representation space. Instances that are tightly grouped in the input field should be mapped to fairly similar representations. In most cases, a linear classifier can achieve superior generalization by applying it to the new space. (Belkin and Niyogi, 2002; Chapelle et al., 2003). The use of principal component analysis (PCA) as a preprocessing step before applying a classifier is a long-standing strategy and is a variation of this method. (in predicted data).

7.5 MULTI-TASTING LEARNING

Multitasking learning is a generalization development method developed by Caruana in 1993. This method involves grouping instances of multitasking that can be visualized as soft constraints applied to parameters. When a part of a model is used for more than one task, that part of the model is more constrained to good values (provided the sharing is justified), often leading to better generalization. Additional training examples place more pressure on model parameters to move towards well generalizable values, while additional training examples increase pressure on model parameters towards well generalizable values.

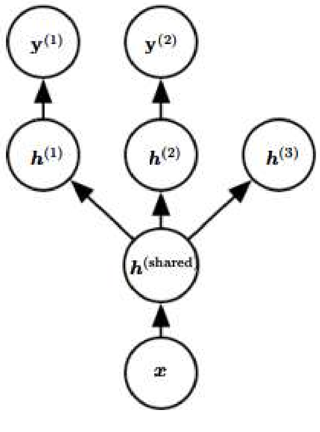


Figure 7.2 Multitasking learning can be rejected in various ways in deep learning

7.6 EARLY CLOSING

When we train large models capable of impersonation to overcrowd the task, we often find that the training error gradually decreases over time, but the validation set error begins to increase again. This is because large models have enough capacity to handle the job. See Figure 7.2 for an example of this behavior. This behavior has major consequences. This indicates that we can construct a model with a better validation set error (and therefore ideally a lower test set error) by restoring the parameter configuration to when it had weaker validation set error. When the amount of error in the validation set decreases, we save a new copy of the model parameters. These settings, not the latest ones, are returned when the learning algorithm finishes working. Curve representing performance.

As shown in Figure 7.2, validation group performance curves take an inverted U shape for most of the hyperparameters that affect model capacity. In the case of early termination, we manage the effective capacity of the model by calculating the number of passes required to adapt to the training set. This allows us to determine how fast the model can do its job. Identifying a hyperparameter early in training and then performing multi-step training to examine how it affects the model is necessary for selecting the vast majority of hyperparameters, which is a time-consuming and time-consuming process. The Training Time hyperparameter, according to its description, is unique in that a single training run can experience many different hyperparameter values. The only significant overhead associated with the automatic selection of this hyperparameter via early termination is the need to perform validation set evaluation at regular intervals during the training process.

In a perfect world this would run parallel to the computer, CPU or GPU training process, whichever is used for the main training. If such resources are not available, the cost of these periodic assessments can be reduced by using a relatively small validation set compared to the training set; Alternatively, it may be possible to evaluate the error in the validation set less frequently and generate a lower resolution estimate for the optimum training time. One of the extra costs associated with premature shutdown is the need to keep a copy of the optimal settings. This overhead is usually negligible, as it is legal to store these features in a slower, larger storage format. (e.g. training in GPU memory but storing optimal settings in host memory or on a disk drive).

Because the best metrics are written very rarely and never read at any point in the process, these occasional lazy writes have little effect on the overall training time required. Early stopping is a relatively inconspicuous type of regularization because it does not require any fundamental changes in the basic training technique, objective function, or set of allowed parameter values. This is why early termination is one of the most effective forms of regulation. This shows that it is easy to implement early termination without adversely affecting the dynamics of the learning process. This is the opposite of weight drop, where one has to be careful not to use too much weight drop and keep the mesh at an insufficient local minimum corresponding to a solution with pathologically small weights.

If you are using too much weight gainer, care should be taken not to use too much weight gainer. Early termination of use is an option, but can also be used in conjunction with other regulatory measures. Even when using regularization methods that change the objective function in favor of better generalization, the best generalization rarely occurs at the local minimum of the training target. In fact, the objective function is modified to support better generalization. Because early termination requires a validation set, some model training data must be retained to meet the requirement. After initial early termination training has been completed, additional training may be undertaken to make the most of the additional data. In the second and subsequent training stages, each training data is used.

With the second training method, you can choose between two basic training techniques. One approach is to restart the training process with all the data and then reset the model. In this second training session, we train for the same number of steps as in the first training session, suggesting that the early stop strategy is more effective. This process includes some nuances that must be taken into account. For example, there is no reliable way to determine whether it will be reused for the same number of parameter changes or the same number of dataset executions. Due to the larger training set size, the second training round requires more frequent parameter changes after each dataset run.

Maintaining the parameters obtained in the first cycle of education and training using all the data is another way in which all available knowledge can be used. At this point, we no longer have a guide telling us when to stop in terms of the number of steps.

Instead, we can keep an eye on the average loss function in the validation set and continue training until the validation set falls below the target value at which the early stop method ends. We can do this by following the function. This technique avoids the significant financial burden of recycling the model from scratch; still not that well educated. For example, there is no guarantee that the validation target set will approach the target value; Therefore, the result of this method is also not an inevitable result. A more formal representation of this approach is provided. An additional benefit of completing the training phase early is the reduced IT workload required. In addition to the advantage of providing editing without having to add penalty terms to the cost function or calculate the gradients of these additional terms, there is a clear cost reduction from reducing the number of learning iterations. This reduces the cost of the algorithm.

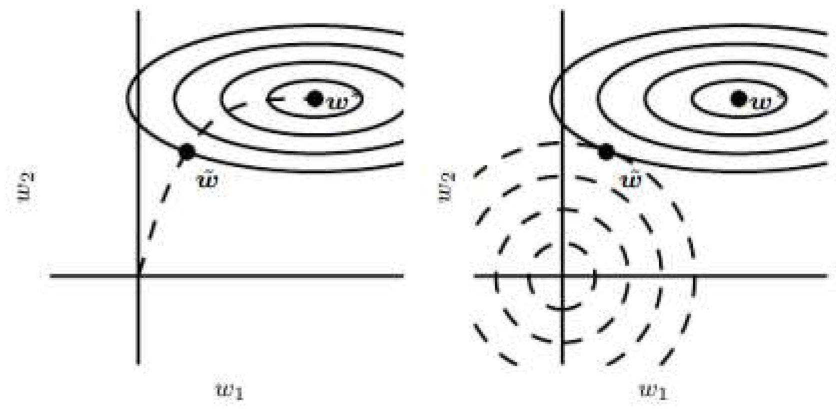


Figure 7.3: An example of the impact of early leaving .

Less regularization is applied to parameter values corresponding to directions with significant curvature (objective function) than directions with less curvature. Of course, in the context of early closure, this indicates that parameters corresponding to high curvature directions tend to learn earlier than parameters corresponding to lower curvature directions. The evidence presented in this section has shown that a length trajectory ends at a position at least equal to the L2-directed target. Early stopping does not only limit the length of the trajectory; Conversely, an early stop usually requires monitoring the validation set error to stop the orbiter at a particularly important location in space. In fact, stopping early isn't just about limiting the length of the trajectory. Stopping early has the advantage over weight loss in that it automatically calculates the appropriate amount of regularization, while weight loss requires many training trials with different values of its hyperparameter. Therefore, quitting early has this advantage over weight loss.

7.7 CONNECTION AND SHARING PARAMETERS

So far in this chapter, when we've talked about adding limits or penalties to parameters, we've always done it in relation to a specific range or point. Because we are trying to solve a problem. For example, the L2 regulation (also known as weight loss) imposes a penalty on the model parameters when they deviate from the zero-centered value. On the other hand, there are situations where we may need other methods to represent our prior knowledge of appropriate values for model parameters. Based on our understanding of model space and design , we are aware that certain relationships must exist between model parameters. However, there may be cases where we are not sure what specific values the parameters should take.

This method was developed by Lasserre et al. (2006) normalized the parameters of a model trained as a classifier in a supervised paradigm to approximate the parameters of another model trained in an unsupervised paradigm. This made the two models look more similar. (to capture the distribution of the observed input data). The architectures are developed in such a way that multiple parameters in the classifier model can be mapped to comparable parameters in the unsupervised model. One approach to smoothing the parameters so that they are close to each other is to use a parameter norm penalty. However, the most common method is to use constraints that involve applying equal parameter sets.

This regularization approach is often referred to as parameter sharing, as we expect many models or model components to share a single parameter set. This is because: 1. When comparing parameter sharing to parameter regularization as closely as possible (with a norm penalty), one of the most notable benefits of parameter sharing is that you only refer to a subset of parameters as a single sentence. , should be kept in mind. This may apply to some models, eg. B. convolutional neural network leads to a large reduction in the memory requirements of the model. This can be achieved by switching to a new model form that uses a different algorithm or objective function. Bagging is a technique that refers to the practice of reusing the exact same model, training procedure, and objective function in multiple situations.

More specifically, bagging involves creating k different databases. The number of samples in each dataset is equal to the number of samples in the original dataset; However, each dataset is created by taking samples from the original dataset and then modifying some of those samples. This indicates that some of the samples in the original dataset are likely to be missing in each dataset, and also that there are several duplicate samples in each dataset (on average, about two-thirds of the samples in the original dataset are in the dataset). included in the resulting training dataset, provided it is the same size as the original dataset). The ith model is then trained using the ith dataset. Inconsistencies in the trained models are the result of inconsistencies in the datasets, especially regarding the samples included in each dataset.

For a picture, see Figure 7.3. Even if all models are trained on the same dataset, neural networks can still benefit from model averaging, as the solutions they provide cover a wide variety of possibilities. Variations in random initialization, random mini-batch selection, changes in hyperparameters, or variable results of non-deterministic neural network applications are often sufficient to cause different members of the community to make at least partially independent errors. The strategy known as the "mean model" is one of the most effective and reliable ways to reduce generalization errors. Because any machine learning algorithm can benefit significantly from averaging models, it is generally not recommended for use in evaluating methods for academic papers. This is because using model averaging comes at the expense of increased computing power and storage capacity.

This is why benchmark comparisons are almost always made using a single model. Methods that use model averaging over hundreds of different models are often the winners of machine learning competitions. A recent and notable example is the first prize awarded by Netflix. (Coren, 2009). There are several methods of team building, and not all of them are meant to set up more teams than individual models. For example, a method known as boosting (Freund and Schapire, 1996b,a) can be used to create an ensemble with greater capacity than individual models. Reinforcing is a technique used to create ensembles of neural networks by gradually adding more neural networks to the ensemble (Schwenk & Bengio, 1998). Another use of amplification has interpreted each neural network as part of a larger whole (Bengio et al., 2006a). This was achieved by gradually adding hidden units to the neural network.

7.8 WITHDRAWAL

Regularization of a large set of models can be done using dropout (Srivastava et al., 2014), which provides a computationally efficient and effective strategy. Dropout can be seen as a strategy that makes bagging suitable for communities with a large number of extremely large neural networks. However, this is only a preliminary assessment. Training many models and running evaluations on many models using each test case is what bagging entails. When each model is a large neural network, this seems like a practical approach, as training and evaluating these networks requires a significant amount of execution time and memory. It is common to use sentences consisting of five to ten neural networks; Szegedy et al. (2014a) won ILSVRC with a system of six neural networks; However, faster usage becomes unmanageable.

When training and evaluating exponentially large numbers of neural networks as part of a bagged community, abandonment offers a very cost-effective approach. Specifically, the formation of the droplet prepares a cluster containing all the subnets that can be created by eliminating non-aggressive entities from an underlying network, as shown in Figure 7.6. In most modern neural networks built on a set of similar transformations and nonlinearity, we can essentially remove a unit from a network by multiplying that unit's output value by zero. This method is used in most existing neural networks. Models such as radial basis function meshes that use the difference between the state of the unit and a reference value require some minor adjustments for this to work properly.

For simplicity, in this section we explain the termination method using the zero multiplication method; However, it is simple enough to modify this algorithm to work with various processes that remove a device from the network. It's important to remember that to learn how to use bagging, we first need to design k separate models, then change samples, randomly select k new datasets from the training set, and finally train model i on dataset i. Dropout's goal is to approach this process with an increasing number of neural networks. To be more specific, we use a step-by-step mini-batch based learning algorithm like B. Stochastic Gradient Descent to train with passion. When we load a sample in a mini-batch, we randomly choose a new bitmask to apply to all input and hidden units in the network. This is done to ensure the proper functioning of the network.

The original of each unit is scanned separately from that of the other units. One of the hyperparameters defined before the training starts is the sampling probability of a mask value, which causes a unit to be trapped. It does not depend on the current values of the model parameters or the sample entered into the system. In most cases, the probability of including an input unit is eighty-five percent, while the probability of including a hidden unit is fifty-five percent. Next, we propagate the training forward as usual, propagate it backwards, and update the training. The forward propagation process by shear is shown in Figure 7.3.

The term "bagging training" should not be confused with "drop training". Each of the models can be considered independent when bagging. Once completed, the models change parameters, with each model inheriting a specific subset of parameters from the main neural network. By sharing parameters , it is now possible to view an infinite number of models in a manageable memory space. Using wrapping, each model learns to assemble in its own individual training set until ready to assemble. When it comes to stopping, the vast majority of models do not go through explicit training. In fact, in most cases, the model is so large that it would be impossible to sample every conceivable sublattice in the entire universe.

Instead, only a small fraction of all potential subnets are trained for a single pass at a time, and metric sharing ensures that the remaining subnets achieve optimal values for all parameters. These are just two differentiating factors. Afterwards, the activity continues in the bagging process. For example, the training set to which each subnet is subject is actually a subset of the original replacement-sampled training set. I. It is impossible to evaluate this sum, as it contains an exponentially increasing number of terms, except where the structure of the model allows for some simplification. It is not yet clear whether deep neural networks will enable a simplification. Instead, we can approach inference by sampling it and then averaging the results of many masks.

Usually 10-20 masks are sufficient for satisfactory performance. On the other hand, there is a better method than this that provides a good estimate of the estimates made by the entire community at the expense of a single forward propagation. For this, we now use the geometric mean of the expected distributions of the cluster members instead of the arithmetic mean. In this particular context, WardeFarley et al. (2014) gives reasons and empirical data that geometric mean works similarly to arithmetic mean. There is no guarantee that the geometric mean of many different probability distributions is itself a probability distribution. We apply the constraint that none of the submodels assign a probability of 0 to any event, then renormalize the distribution resulting from this requirement. This allows us to make sure that the end result is a probability distribution. It is provided and is the unnormalized probability distribution defined directly by the geometric mean. Understanding that we can approximate p(y | x) by evaluating it in a model - a model with all units but with weights from i increasing the probability of including i - One of the most important lessons we can draw is latency (Hinton et al., 2012c).

This is "Stall: The purpose of this change is to ensure that the appropriate value is captured for the expected output of this unit. We call this strategy the scaling inference rule, although there is no theoretical explanation about the accuracy yet. To support the inference method in deep nonlinear networks, it works quite well in practice. Usually , the method of scaling weights from training is to divide the weights by 2 and then continue using the model as usual to include probability. normally we use is equal to 1 2. There is another approach that achieves the same goal during training, that is to multiply the unit states by 2. In both cases, the goal is to ensure that all inputs approximately belong to one unit, equal to the total input, provided during the test period, although on average half of the units will not be present during the rotation, I am in favor of this unit during the rotation.

The weight scaling rule is also true in other cases, eg. B. Regression networks with conditional normal outputs and deep networks with non-linearity hidden levels. Both types of networks have conditionally normal outputs. However, the weight scaling rule is only an approximation for deep models with nonlinear features in their structure. In practice, the approach is generally successful, although it is not well defined in theory. Experiments by Goodfellow and colleagues (2013a) have shown that the weight scaling approach has the potential to work together more efficiently (in terms of classification accuracy) than the d'predictor Monte Carlo approaches. This was the case even when the Monte Carlo approach was able to sample up to a thousand different subnets.

According to research by Gal and Ghahramani (2015), some models achieve better classification accuracy when twenty samples are used with the Monte Carlo approach. It seems that choosing the most appropriate inference approach depends on the particular problem. It was developed by Srivastava et al. (2014), B. that abandonment is a more efficient regulator than other typical regulators that are computationally inexpensive, such as weight loss, filter norm constraints, and sparse activity regulation. Release can be used with many other types of edits to achieve even more improvements.

Quitting has numerous benefits, and one of the most important is that it does not place any significant limitations on the training models or methods that can be used. It is effective in almost any model that uses a distributed representation and can be trained with stochastic gradient descent. These include recurrent neural networks, feed-forward neural networks, and probabilistic models such as Bounded Boltzmann machines (Srivastava et al., 2014). (Bayer and Osendorfer, 2014; Pascanu et al., 2014a). There are many other regularization methods that perform just as well but impose tighter constraints on model design. While the cost of demolition to a particular model has minimal impact on the cost per floor, the cost of using demolition can have a significant impact on the entire system. Dropout is a regularization approach and thus results in a reduction of the effective capacity of a model. To reduce the impact of this effect, we need to make the model much larger.

When using undo, the error of the ideal validation set is usually much smaller than with any other method. However, this comes at the expense of a significantly more complex model and numerous additional iterations of the training procedure. Regularization does not reduce the generalization error much when applied to very large datasets. In such cases, the benefit of editing may not be worth the extra computational cost of abandoning it or using larger models. Stall training is less effective when only a small number of cases need to be reported. In the alternative coupling dataset (Xiong et al., 2011), Bayesian neural networks (Neal, 1996) outperform dropout when less than 5,000 samples are available. (Srivastava et al., 2014). Unsupervised feature learning has the potential to have an advantage over abandonment where additional unlabeled data is available.

each input feature The magnitude of the weight loss coefficient for each feature is determined by the variance of that feature. Other linear models provide comparable results. For deep-fitting models, losing weight is not the same as losing weight. The stochastic used in training with the recognition process is not necessary for the success of the method. Simply put, it is a method for estimating the total value of all submodels. In response to this marginalization, analytical approaches have been developed by Wang and Manning (2013). Their approach, which they call rapid fallout, resulted in a reduction in the stochastics found in the gradient computation, resulting in a faster convergence time. This technique can also be applied at test time as a more principled averaging approach than the weight scaling approach. However, using this method at test time incurs higher computational costs.

The performance of fast pauses is used to be almost equal to normal pauses on small neural network difficulties. However, the fast stall has yet to show any significant improvement or just got used to a general problem. The presence of stochastics is not necessary, nor is their presence sufficient to create the moderating effect of the pause. For evidence, Warde-Farley et al. (2014) designed a series of control experiments using a technique known as stall stimulation. This technique should use the same masking noise as a conventional stall, but lacks the modulating effect of a traditional stall. Using the dropout gain, the entire ensemble is trained to work together to get the highest possible log probability on the training set. This strategy is similar to reinforcement, just as the more traditional method of quitting is similar to relaxation.

Experiments using Churn stimulation show essentially little editing effect compared to training the entire network as a single model. This is exactly what the researchers were hoping for. This shows that it is helpful to understand a stall as immunity to noise, as well as to interpret a stall as a failure. The regulatory effect of the bagged ensemble can only be achieved when members of the stochastically sampled ensemble are instructed to play well independently of each other and the bagged ensemble. Other stochastic methods for training exponentially large ensembles of weight sharing models have been inspired by Dropout. Any product between a single scalar weight and a single hidden unit state is considered a disposable unit under the terms of the DropConnect variant of the abandonment algorithm. (Van et al., 2013).

The construction of convolutional network clusters, where each convolutional network responds to different spatial locations of each feature map, can be accomplished with stochastic clustering, a type of random clustering. Dead end is still the default approach used by most people. One of the most important lessons learned from deduction is that one can implement a kind of parameter sharing bag by training a network with stochastic behavior and generating estimates by averaging a set of different stochastic decisions. We have defined dropout before, so far we have only discussed it in terms of how it can be used as an efficient and rough packaging method. On the other hand, there is an alternative perspective on destruction that goes much further. Abandoned Trains is not just a collection of pocket models; Instead, it's a collection of models that share hidden drives. This indicates that each hidden volume in the model must be able to operate effectively independently of other hidden volumes in the model.

It requires hidden units that can be traded and exchanged between different models. A theory of biology guided the research of Hinton and others. (2012c). This idea states that sexual reproduction, which involves the exchange of genes between two different creatures, creates an evolutionary push for genes to become not only desirable but easily interchangeable between different organisms. These genes and traits are particularly resistant to changes in their environment, as they cannot adapt inappropriately to the distinctive features of a particular organism or model. This makes them particularly resistant. As such, Release makes every hidden drive not only a great feature but also a useful one in many different contexts. WardeFarley et al. (2014) compared independent model training with distraction to large group training.

They concluded that omission confers significant benefits to the generalization error beyond those derived from independent model ensembles. It is important to have a solid understanding that applying noise masking to hidden drives is the source of a significant percentage of performance loss. This can be seen as a rather clever and adaptive destruction of the informational content of the input, rather than destroying the raw values of the input. For example, if the model learns of a hidden entity called hi, which determines whether an image has a face by locating the nose, removing the hi is equivalent to removing awareness that the image contains a nose. The model must learn a different greeting, which unnecessarily encodes the presence of a nose or recognizes a face from a feature other than the nose, such as a smile. B. by mouth.

Conventional noise injection approaches that add unstructured noise to the input cannot inadvertently remove information about a nose from an image unless the noise level is high enough to remove almost all image information. If the target of the segmentation is not the original values but the extracted features, it is possible that the segmentation process uses all the input distribution information that the model has collected so far.

Stall is notable for a number of important reasons, one of which is the proliferation of noise.

If the noise is collected on a fixed scale, it will be a corrected latent linear unit. Hello, with additional noise, the new noise can be easily allowed to overgrow so that it appears low in comparison. The pathological solution to the noise robustness problem cannot be achieved with multiplicative noise. Batch normalization is another deep learning approach that, when applied to a model, can reparameterize the model to add additional and multiplicative noise to the hidden units during the training process. The main purpose of heap normalization is to improve optimization; However, noise can have a modulating effect and it is often not necessary to stop it. Batch standardization

7.9 FORMATION OF COMPETITORS

In many cases, neural networks are starting to approach the level of performance that humans achieve, as measured by a series of iid tests. It is therefore reasonable to ask whether these models achieve a real understanding of these activities at the person level. To examine how well a network understands the required work, we can look for cases where the model is misclassified. Szegedy et al. (2014b) found that even neural networks that work as accurately as humans have an error rate of about 100% in samples that were intentionally created using an optimization technique. The scope of this section does not include many conclusions that can be drawn from conflicting scenarios, for example in the field of information security.

However, they are of interest in the context of regularization because with adversarial perturbed samples from the training set, the error rate in the distributed, independent first test set can be reduced in the same way as with adversarial training training. . (Szegedy et al., 2014b; Goodfellow et al., 2014b). Goodfellow et al. (2014b) showed that extreme linearity is one of the main causes of these contentious situations. Linear building blocks are widely used in the creation of neural networks. After some testing, it turned out that the overall function implemented in the end is very simple. These simple linear functions require very little effort.

The power of using a large family of functions in conjunction with aggressive regularization is best demonstrated using contentious training. Logical regression is an example of a purely linear model, and such models cannot withstand contradictory situations because they are limited by linearity. Neural networks have the ability to represent functions that can range from nearly linear to nearly locally constant. As a result, these networks have the flexibility to capture linear trends in the training data while learning to resist local perturbations.

There is a strong link between tangential spanning and dataset expansion. The user of the algorithm encodes his prior knowledge of working in both scenarios by defining a series of transformations that should not change the output of the network. This is done to keep network output consistent. The difference is that in the event of an increase in dataset size, the network is deliberately trained to correctly categorize discrete entries created by applying these changes more than an infinitesimal fraction of the time. It is not necessary to visit a new entry point for tangential propagation to occur. Instead, it analytically tunes the model to make it resistant to distortions in directions consistent with the given transformation.

While this analytical technique is beautiful in terms of intellectual sophistication, it suffers from two major flaws. First, the model is simply tuned to tolerate a very small perturbation. Open enrollment growth provides resilience to larger disruptions. Second, the infinitesimal method creates complexities for models based on true linear units. These models can reduce the size of their variants by simply disabling units or reducing the weights of those units. They cannot shrink the size of their derivatives the way Sigmoid or Tanh units do, and saturate themselves with a significant amount of weight at a high stat. The use of corrected linear units is useful for data set augmentation. This is because different subsets of set entities can become active depending on the converted version of each original entry.

A connection can also be made between tangential diffusion and the double back propeller (Drucker and LeCun, 1992) and the opposing formation. (Szegedy et al., 2014b; Goodfellow et al., 2014b). Double backprop makes Jacobi more compact, while contentious training finds inputs that are geographically close to the original inputs and then trains the model to produce the same output for those inputs as the original inputs. To allow for tangential propagation and dataset expansion using manually specified transformations, the model must be invariant with respect to certain directions of change defined in the input. Double backprop training and competitor training require the model to be invariant in all aspects of input change, provided that the change in question is negligible. Just as enlarging the dataset is the non-infinite equivalent of tangential spread, enemy training can be thought of as the non-infinite equivalent of double rear propeller. The multi-tangent classifier developed by Rifai et al., 2011c eliminates the need to know the tangent vectors beforehand. Autoencoders can do a variety of things, as we'll see below.

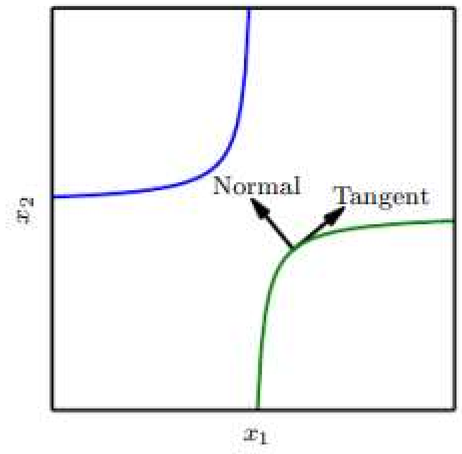


Figure 7.4: Representation of the basic idea of the tangential helix algorithm

7.10 IMPROVED WAY TO TRAIN THE RECOGNITION SECTION OF SSD

To ensure that the SSD performs as well as possible, a significant portion has been devoted to the process of developing location data for multiple objects from the ground up. A much better method would be to first train the final detection layers of the SSD network to distinguish between the many types of ground robots and their respective locations. It will make the system much more efficient. This requires training convolution prediction layers using 3x3xP feature maps created from blank or object-containing images. P is the number of channels the estimator must accept.

Such a method will save a lot of time as it is much easier to collect information about just one thing. The hardest part of implementing this strategy is figuring out how to localize. This can be achieved with a technique similar to that suggested in Section 6.2.4, or it can be done in the very easy-to-follow recommended way. Note that this is only a theoretical option and there may be limitations limiting the use of this strategy that are not explored here.

CHAPTER 8

OPTIMIZATION FOR TRAINING DEEP MODELS

Deep learning algorithms need to be optimized in different contexts. Solving an optimization problem is necessary to draw conclusions in models such as principal component analysis (PCA). When writing proofs or designing algorithms, we often apply analytical optimization. Training a neural network is by far the most challenging of the various deep learning optimization problems. It is extremely common to work on hundreds of computers for a few days to several months to solve even a single instance of a neural network training problem. A unique toolbox of optimization strategies has been developed to solve this vital and financially grueling problem. These different optimization strategies for learning neural networks are presented in this section.

If you're not familiar with the basics of gradient-based optimization, we recommend the following. This chapter provides a brief introduction to numerical optimization in general. This section focuses on a specific optimization application that determines the parameters of a neural network that significantly reduces a cost function. A cost function typically consists of a performance measure evaluated over the entire training set and additional smoothing constraints. First, we'll discuss how pure optimization differs from optimization used as a training technique for a machine learning job. Next, we'll discuss some of the more tangible obstacles that add to the difficulty of optimizing neural networks.

Next, we will continue to develop a number of useful methods that include not only optimization algorithms but also parameter initialization techniques. More complex algorithms adjust learning rates during training or use information contained in the second derivative of the cost function. Finally, we conclude this discussion by looking at a number of different optimization strategies. These strategies are created by combining a number of different basic optimization algorithms in higher-level processes.

Empirical risk minimization describes the training procedure aimed at reducing the number of errors that occur on average during training. In this context, machine learning is still quite comparable to optimization in its simplest form. Rather than explicitly optimizing for risk , we're optimizing for empirical risk and keeping our fingers crossed for this to translate into significant risk reduction. Various theoretical results describe situations where different amounts of real risk reduction can reasonably be expected. On the other hand, empirical risk reduction can easily lead to overfitting. The training set can be easily stored thanks to the large-capacity models. Empirical risk reduction is not really possible in many cases.

The most successful current optimization techniques are based on gradient descent. However, many useful loss functions, such as the 0-1 loss, have no corresponding derivative. (the derivative is everywhere zero or indefinite). Because of these two problems, empirical risk mitigation is rarely used in deep learning. Instead, we need to use a slightly different strategy where the number we're actually optimizing is even further away from the amount we actually want to maximize.

8.1 LOSS OF REPLACEMENT AND EARLY CLOSING FUNCTIONS

There are cases where the loss function we are really interested in (e.g. misclassification) cannot be effectively improved. Even with a linear classifier it is often impossible to perfectly minimize the expected loss of 0-1 because the problem is exponentially related to the input size. (Marcotte and Saverd, 1992). In such cases, it is common to optimize a proxy loss function instead, which serves the same purpose as the proxy but has additional benefits. For example, the common substitution of the 0-1 loss is the negative log probability of the corresponding class. This is because it is easier to calculate. Because of the negative log probability, the model can predict the conditional probability of classes given input. If the model can accurately predict the conditional probability, it can choose the classes that produce the smallest expectation classification error.

Substitution loss functions can occasionally add to the amount of information that can be learned. For example, when training using the log-probability proxy, the 0-1 test set loss usually continues to decrease for a significant amount of time after the 0-1 training set loss reaches the point where it is zero. Therefore, it is possible to increase the robustness of the classifier by further separating the classes, resulting in a safer and more reliable classifier, and thus extracting more information from the training data than is possible, minimizing 0-1. means loss over the training set. Even if the expected

0-1 loss is zero , the robustness of the classifier can actually be improved by moving the classes further apart.

The fact that training algorithms generally do not stop when they reach a local minimum is one of the most important distinctions that can be made between optimization in general and optimization based on training algorithms. Instead, a machine learning algorithm attempts to minimize a substitution loss function before stopping when the early stop convergence condition is met (Section 7.8). In most cases, the early stopping criterion is determined by the actual underlying loss function, such as 0-1 defeat B scored on a validation set. Its purpose is to stop the algorithm when it detects signs of overfitting. When learning is stopped, the loss function by substitution usually still has significant derivatives. This differs significantly from the pure optimization context, which assumes that an optimization method converges when the gradient is extremely small.

Redundancy in the training set is another factor that contributes to the statistical gradient estimation using a small number of samples as a starting point. In the worst case, each of the m samples in the training set may be an exact copy of the other. The naive method requires m times more operations than the sample-based gradient estimation, which can calculate the correct gradient with a single sample. In real life, it is quite rare that we encounter a worst-case scenario, but it is possible that we will encounter a large number of situations, each contributing very similarly to the slope.

Deterministic gradient or batch techniques are a type of optimization algorithm that uses aggregate learning. These methods got their name due to the fact that they process all training samples simultaneously in one large chunk. This terminology can be somewhat misleading, as the term "stack" is often used to describe the mini-cluster used by the mini-stack stochastic gradient descent. Using the full training set is often implied by the phrase "bulk gradient descent", but using the word "group" to refer to a collection of examples usually does not imply this. For example, it's pretty common to use the word lot size when talking about the size of a mini-batch.

Stochastic techniques or online methods are two names sometimes used to refer to optimization algorithms that only consider one situation at a time. The word "online" is generally reserved for situations where samples are drawn from a continuously generated stream of samples, rather than learning from a fixed-size training set with many runs. Most algorithms for deep learning fall somewhere in between, using more than one, but less than all training examples. It is now common to refer to them simply as stochastic methods, but in the past they were more commonly referred to as minibatch or minibatch stochastic methods.

The size of small batches is usually determined by the following factors:

- Larger batches provide a more accurate gradient estimate, but less than linear yield.
- Multi-core architectures are typically underutilized by extremely small groups. This motivates the use of an absolute minimum lot size, below which there is no reduction in the processing time of a mini-batch.
- If all instances in the stack are to be processed in parallel (which is usually the case), the amount of memory scales with the size of the stack. For many hardware configurations this is the limiting factor for stack size.
- Certain hardware types work better with certain array sizes. It is a common use of the performance of 2 batch sizes to provide better execution, especially when using GPUs. Typical horsepower for 2-size batches ranges from 32 to 256, and sometimes 16 is attempted for large models.
- Small parties can have a regularizing effect, perhaps because of the noise they add to the learning process (Wilson & Martinez, 2003). Generalization error is usually better when heap size is 1. Training with such a small group size may require a low learning rate to maintain stability due to the high variance in the gradient estimation. Total execution time can be very high, both due to the low learning rate and the need to perform multiple steps as more steps are required to observe the entire process.

It is also important that the mini-batches are chosen randomly. These samples must be independent of each other to provide an accurate estimate of the expected gradient based on a given data set. We also want the two consecutive estimates of the gradient to be independent of each other; Therefore, the next two sample mini-batches should also be independent of each other. Many datasets have a natural order that places consecutive samples highly correlated with each other. For example, we may have a medical record consisting of a long list of test results from blood samples. This list can be structured so that the first five blood samples will be drawn at different times from the first patient, then three blood samples will be drawn from the second patient, then blood samples will be drawn from the third patient, and so on. C.

Alternatively, the list could be written starting with the blood samples from the third patient and continuing from there. If we were to select cases in the order of presentation from this list, each of our mini-lots would be highly biased, as each would represent a single patient from among many patients largely included in the dataset. Before choosing minisets, it is important to mix samples where the order in which the dataset is presented has some effect on the results. When dealing with extremely large datasets such as B. datasets with billions of samples in a data center, it may not be possible to select truly random samples every time we want to create a mini-batch.

This is due to the size of the dataset. The good news is that it is usually sufficient to change the order of the dataset once and then save it as shuffled. This creates a fixed set of possible sequential sample mini-batches that will then be used by all trained models, and each model must repeat this sequence each time it switches between training data. This has the effect of imposing a fixed set of possible minimum amounts of sequential copies. However, this deviation from truly random selection does not appear to have any major adverse effects. It is possible that the effectiveness of the algorithm will suffer significantly if the samples are not reordered in some way at some point.

When you're done, the dropdown will shuffle the posting and then post it multiple times. During the first iteration, each mini-batch is subjected to a calculation that provides an independent estimate of the true generalization error. The result is one that is skewed in a skewed direction, as the second estimate is derived from resampling the values already used, rather than taking new fair samples from the distribution from which the data was produced. The e-Learning scenario, in which samples or mini-batches are selected from a data stream, is the simplest way to show how stochastic gradient descent can reduce the amount of generalization error that occurs. In other words, the student is like a living person in that, instead of receiving a learning set of predetermined size, he is exposed to new samples such as each sample (x,y) from the distribution that constantly generates data pdata (x). , y).

In this case, the examples are never repeated; Each encounter is a representative sample of pdata. It is increasingly common for machine learning systems to use each training sample only once or even perform an incomplete analysis of the training set. This is because the size of some datasets is increasing faster than the capacity of computers can keep up with the growth. Overfitting is not a problem when working with a very large set of drives; Therefore, poor fit and computational efficiency become major problems. For a discussion of the increasing number of training examples and the impact of computational bottlenecks on generalization errors, see also. Bottou and Bousquet (2008).

8.2 CHALLENGES IN OPTIMIZING NEURON NETWORKS

In general, the optimization process is quite difficult. In the past, machine learning overcame the overall optimization challenge by carefully constructing the objective function and constraints to make the optimization problem convex. This allowed machine learning to get rid of the complexity of universal optimization. When training neural networks, we have to deal with the general non-convex scenario. Convex optimization also has its share of difficult aspects. In this section, we will discuss many of the main challenges when trying to optimize deep model training.

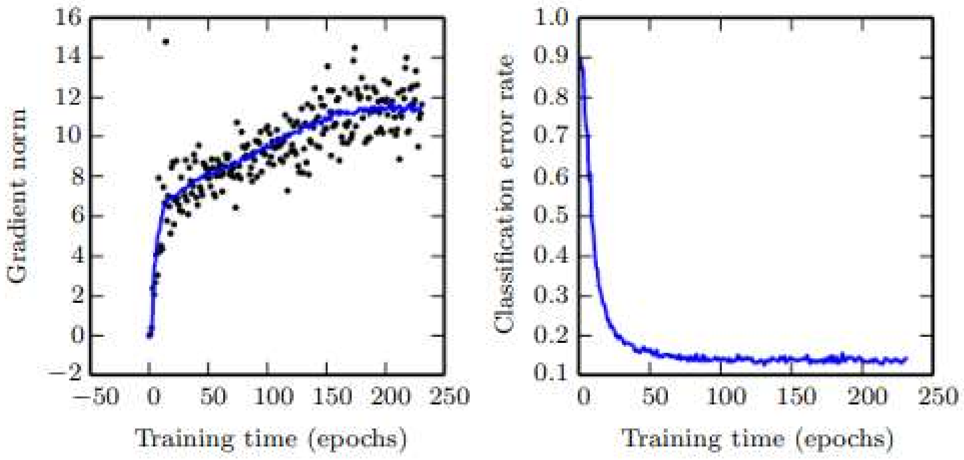


Figure 8.1: Gradient descent usually does not reach a critical point.

While bad conditioning exists in contexts other than training neural networks, some techniques used to combat it in other contexts are less applicable to neural networks. For example, Newton's method is an excellent tool for minimizing convex functions with ill-conditioned Hessian matrices, but in later sections we will discuss that Newton's method requires significant modifications before it can be applied to neural networks.

8.3 MINIMUM LOCATION

It is possible to simplify a convex optimization problem according to the difficulty of finding a local minimum, which is one of the most striking features of a convex optimization problem. For a local minimum to be valid, a global minimum must be established. Any point within such a flat region is a valid solution, although some convex functions have a flat region underneath instead of a single spherical minimum point below. While trying to find the best solution for optimizing a convex function, we will know we did it when we find some kind of critical point. When working with non-convex functions such as neural networks, it is possible to have several different local minima. In fact, it is almost certain that there will be an exceptionally large number of local minimums in almost every deep model.

However, as we shall see, this should not be a major problem at all. Because of the difficulty associated with model identifiability, they all have many local minimums, like neural networks and all other models containing hidden variables with many equivalent parameters. A model is said to be identifiable if a sufficiently sufficient training set can rule out all other possible configurations of a model's parameters except one. Patterns with hidden variables often become unrecognizable because we can catch identical patterns by changing the hidden variable values in other patterns. For example, we could take a neural network and replace layer 1 by replacing the input weight vector for entity i with the input weight vector for entity j, and then repeat the process for the output weight vectors.

This will result in a modified neural network. If there are M layers and each layer contains n units, then n! Different ways to organize hidden units. Weight-space symmetry is the term used to describe this type of indefinability. In addition to weight-domain symmetry, many different forms of neural networks have other factors that contribute to their undefinability. For example, in any linear or maxout set network, if we scale all the outgoing weights of that unit by 1, we can scale all the inbound weights and deviations by one unit. This allows us to scale weights and incoming distortions independently.

If the cost function does not contain words based directly on weights and not on model outputs, such as weight loss, this indicates that each local minimum or maximum output of a linearly corrected network falls into a hyperbola of the size (min) of the minimum. local equivalent This is when the cost function does not include terms such as weight reduction that are directly dependent on the weights and not on the model outputs. Because of these model identifiability issues, the neural network cost function may have an infinitely uncountable number of local minimums in addition to an extremely large number. On the other hand, the value of the cost function for each of these local minimums is the same, due to the impossibility of defining them.

As a result, these locally minimized areas do not pose a problematic non-convex condition. Compared to the global minimum, the effort to reach the local minimum can be prohibitive, leading to discomfort. Even without hidden drivers, it is possible to build small neural networks that have a local minimum and cost higher than the global minimum. (Sontag and Sussman, 1989; Brady et al., 1989; Gori and Tesi, 1992). Gradient-based optimization algorithms can encounter significant difficulties when a significant cost local minimum applies. The question remains whether there are many expensive local minimums of practical importance for networks and whether optimization algorithms fit them.

For a long time most practitioners were under the impression that local minimums were a typical problem in neural network optimization. That doesn't seem to be the case at the moment. The problem is still under investigation, but experts now believe that for neural networks large enough, most local minimums have cheap functional value. They also argue that finding a true global minimum is not as important as finding a low but not minimum cost point in the parameter space. In fact, finding a true global minimum is more difficult than finding a low-cost point in the parameter space. (Saxe et al., 2013; Dauphin et al., 2014; Goodfellow et al., 2015; Choromanska et al., 2014). According to some practitioners, local minimums are the source of almost all the difficulties involved in neural network optimization.

We strongly recommend that professionals perform rigorous testing for a variety of issues. Plotting the gradient norm against time is a technique that can be used to rule out the possibility that the problem is a local minimum. The problem is not a local minimum or some other form of critical point unless the gradient norm drops to an insignificant magnitude. Such a negative test can rule out the possibility of a local minimum. When dealing with high dimensional spaces it can be quite difficult to show conclusively that the problem is caused by local minima. In addition to local minimums, there are a large number of structures with modest slopes.

8.4 CHAINS, SADDLE AND OTHER FLAT SURFACES

Local minimums (and maximums) are actually quite rare compared to another zero gradient point called the saddle point. This is true for many high-dimensional non-convex functions. There are spots around a saddle point, some of which cost less than the saddle point, while others are more expensive. When at a saddle point, the Hessian matrix has both positive and negative eigenvalues. Compared to the saddle point, points along eigenvectors associated with positive eigenvalues have a greater value, while points along eigenvectors associated with negative eigenvalues have a smaller value. The local minimum along one cross section of the cost function and the local maximum along another cross section are one way of thinking about a saddle point. Another way to think about it is where these two parts meet. See Figure 8.1 for a specific example.

Moving into low-cost regions, one of the more intriguing features many random functions share is that Hessian eigenvalues are more likely to be positive. If we are at a crucial stage with a low cost, this shows that we are more likely to flip our coins n times using the coin flip example. This indicates that lower costs are more likely to be associated with local minimums than higher costs. Saddle spots are much more likely to occur in high cost critical areas. The probability that a critical point is a local maximum increases when the costs involved are excessive. This applies to various different types of random functions.

Does this also apply to neural networks? Theoretically, Baldi and Hornik (1989) showed that flat autoencoders, which are feedforward networks trained to multiply their inputs to their outputs, have global minimum and saddle points, but no minimum predecessors with costs above minimum global. They observed, but provided no evidence that the same results apply to deeper networks without nonlinearity. Although the output of such networks is a linear function of their input, it is advantageous to examine them as a model for nonlinear neural networks because their loss function is a non-convex function of their parameters. This makes the results of these networks valuable. Such networks are actually a collection of strings put together.

Sachsen et al. (2013) established precise solutions to the general dynamics of learning in such networks and showed that learning in these models retains many of the qualitative aspects observed when constructing deep models with nonlinear activation functions. These results were published in the journal Nature Communications. Yunus et al. (2014) showed that real neural networks, such as artificial networks, have loss functions that involve a large number of expensive saddle points. Choromanska et al. (2014) provided further theoretical consideration and showed that another class of high-dimensional random functions associated with neural networks does the same. What is the effect of increasing number of saddle points on training algorithms? The problem is not obvious for first-order optimization algorithms, as the only data they need is gradient information.

Near a saddle point, the slope often flattens to an almost negligible level. On the other hand, hill descent seems to be able to avoid saddle points in some cases through empirical testing. Goodfellow et al. (2015) presented many representations of contemporary neural network learning trajectories, with Figure 8.2 exemplifying such a trajectory. These visualizations show the cost function flattening around a major saddle point where the weights are completely zero, but also show the gradient descent path rapidly exiting this region. This is because the weights at the saddle point are all zero. Goodfellow et al. (2015) further suggest that it is possible to show analytically that the time-continuous gradient descent is repelled rather than pulled towards a nearby saddle point; however, the situation may be different in more realistic downhill applications. Not surprisingly, saddle points defy Newton's technique.

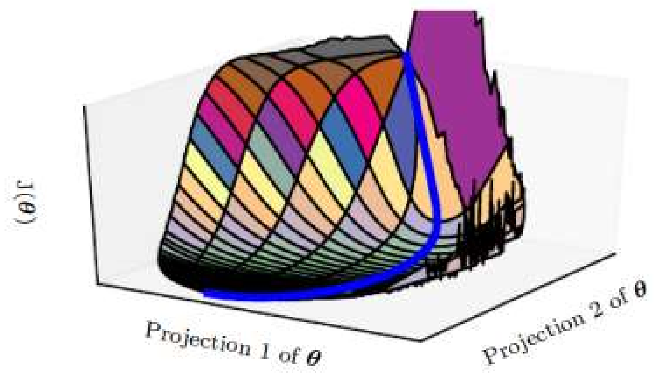


Figure 8.2: Visualization of the cost function of a neural network.

Image modified with permission of the author and courtesy of Goodfellow et al. (2015). When used for real-world object identification and natural language processing applications, feed-forward neural networks, convolutional networks, and recurrent networks all appear to have comparable pictures. Surprisingly, these depictions usually do not have many easily visible barriers. Before the success of stochastic gradient descent to train extremely large models around 2012, it was believed that the surfaces of the neural network cost function contained much more non-convex structure than these projections suggest.

This belief persisted even after the success of the stochastic gradient descent. The primary challenge highlighted in this projection is an important cost saddle point close to the source of the settings; But as shown on the blue track, the SGD training track can easily bypass this saddle point. Most of the training time is spent traversing the relatively flat valley of the cost function. This may be due to intense noise in the gradient, insufficient conditioning of the Hessian matrix in that region, or the need to skip the large "mountain" shown in the figure via an indirect arc path. All these factors can play a role.

The purpose of gradient descent is not to detect a critical point; Rather, its design is geared towards "downhill" travel. Newton's approach, on the other hand, aims to find a solution where the slope does not change. It can jump to a saddle point if the necessary changes are not made. Due to the prevalence of saddle points in high-dimensional spaces, it is likely that second-order approximations have not successfully replaced gradient descent in the training process of neural network models. In their 2014 study, Dauphin and colleagues presented a bareback Newtonian technique for quadratic optimization, showing that it is a huge improvement over the more traditional approach.

While it is still difficult to scale quadratic approaches in large neural networks, this bareback approach is proving to be a potentially useful alternative if it can be made scalable. In addition to low and saddle points, there are many other types of positions with zero slope. There are also maximum values that are quite similar to saddle points in terms of optimization; However, while most algorithms don't push their limits, Newton's unmodified technique does. The maxima of many random functions become exponentially rarer in higher dimensional spaces, just as minimas become increasingly rare in higher dimensional spaces.

There may also be large fixed-valued areas that are perfectly flat. At these points in time and space both the gradient and the Hessian are zero. These corrupt places pose great challenges to the entire community of numerical optimization methods. For a convex problem, a large flat region should consist only of global minimums. Now, in a general optimization problem, this range may correspond to a high value of the objective function.

The reef is potentially dangerous whether you approach it from above or below; However, the gradient clipping heuristic discussed in allows us to avoid the more serious effects of the abyss. It is important to note that the gradient does not show the best step size, but only the ideal direction over an infinitesimal range. This is the basic principle of this concept. When the typical gradient descent method suggests taking a step that is too large, the gradient clipping heuristic steps in to reduce the step size so that it is small enough to reduce the likelihood of the gradient shifting outside the region where the steepest descent is pointing in the approximate direction of the descent.

This is accomplished by reducing the step size to a value small enough to make it less likely to go out of range. Reef structures are most commonly observed in cost functions of recurrent neural networks. This is because such models involve multiplying multiple factors by a component representing each time step. Therefore, an extremely large amount of multiplication is required for long-term work. Rather than focusing on developing algorithms that take advantage of nonlocal motion, many current research areas focus on finding suitable starting points for problems with complex global structures. Taking small localized steps is the foundation of gradient descent, along with almost all other learning techniques useful for the neural network training process.

It has been noted in previous chapters that choosing the appropriate path for these local movements can be difficult. We can specify some properties of the objective function, such as g. gradient can only be approximated, meaning that our estimate of optimal direction may be subject to bias or fluctuation. Under these circumstances, the local descent may or may not define a short enough path for a valid solution, but even if it does, we cannot follow the local descent path to completion. There may be problems with the objective function, eg B. underconditioning or discontinuous gradients, so the region where the gradient gives a smooth model of the objective function is extremely small. These problems can cause problems with the target function.

In these circumstances, a local descent in large steps may provide a relatively quick path to resolution; However, we can calculate the local absorption direction using only orders of magnitude. In these cases, local ancestry may or may not identify a response pathway; However, since the path consists of many steps, it significantly increases the calculations to be followed. If the function contains a large flat surface or we can reach a critical point exactly (the latter case usually only applies to techniques that explicitly solve the critical points, for example Newton's method), sometimes local knowledge does not provide us with any information. guidance.

In these special cases, the local landing process is in no way a solution. In other cases, local movements can be too greedy and put us on a downward path that takes us away from any solution as shown in Figure 8.4, or on a very long trajectory towards solution as shown in Figure 8.2. Both of these scenarios can be avoided by avoiding overly greedy local moves. We do not currently understand which of these issues is more important in complicating neural network optimization, and this is an issue that is actively being researched. If space has a domain and is directly tied to a solution in a way that can trace local origin, and we can learn from within that higher region, then all these problems are possible to study. may be around. This applies regardless of the more serious problem. The second perspective suggests conducting a study on the selection of suitable starting points for the use of traditional optimization techniques.

8.5 THEORETICAL LIMITS OF OPTIMIZATION

The performance of any optimization method we can create for neural networks is limited by a number of theoretical studies. (Blum and Rivest, 1992; Judd, 1989; Wolpert and MacReady, 1997). In most cases, these results have little impact on the use of neural networks in actual scientific research. Some theoretical insights apply only to cases where units of a neural network produce discrete values. On the other hand, most neural network units generate higher and higher values that allow local search optimization. There are classes of problems that cannot be solved according to some theoretical research, but it is not always easy to determine whether a particular problem belongs to one of these classes.

Further research has shown that it is impossible to find a solution for a network of a given size; In reality, however, a solution can be obtained without much difficulty by using a larger mesh where a much larger variety of parameter values can correspond to a satisfactory answer. Even when training neural networks, we usually don't care about finding the exact minimum of a function; instead, our goal is to minimize the value of the function as much as possible while maintaining a satisfactory level of generalization error. It is quite difficult to conduct a theoretical study to determine whether an optimization technique can achieve this goal. Therefore, machine learning research should focus on achieving its main goal, which is to provide more realistic limits to the performance of optimization algorithms.

8.6 PARAMETER INITIATION STRATEGIES

Some optimization algorithms are not iterative by design and instead only solve for one solution point. Other optimization methods are iterative in nature; However, when applied to the appropriate category of optimization problems, they converge to acceptable solutions in a reasonable time regardless of the initialization used. Commonly used training algorithms for deep learning lack these conveniences. Training methods for deep learning models are often iterative in nature. As a result, the user must specify a starting point from which to start the iterations for the training to continue. Also, training deep models is such a difficult process that initialization selection has a significant impact on the performance of most methods. It is possible for the starting point to decide whether the algorithm converges or not.

Some starting points are so unstable that the algorithm encounters computational problems and eventually fails. When learning converges, it can affect where it starts, how fast it takes place, and whether you end up in a high- or low-cost position. Also, points of equivalent cost can have very different degrees of uncertainty in generalization, and the starting point can also affect generalization. Initialization methods today are often simple and heuristic. Due to the lack of in-depth knowledge of neural network optimization, the challenge of designing more efficient initialization methods is daunting. The vast majority of startup procedures focus on finding some desired attributes after network setup. However, after the learning process begins, it is not clear to us which of these features are preserved and under what conditions.

Another hurdle is the possibility that certain starting points are advantageous for optimization but harmful for generalization. This is a double-edged sword. Our knowledge of how the starting point affects generalization is very rudimentary and therefore we give little or no guidance on how to choose the first point. It is perhaps the only property that can be said to be known with absolute certainty that core parameters must "break the symmetry" between different entities. If two hidden units with the same activation function are combined with the same inputs, the initial parameters of these units must be different from each other. If the initial parameters are the same, a deterministic learning algorithm applied to deterministic costs and models will always update these two entities in the same way, provided they have the same initial parameters.

Even if the model or training method can use stochastics to calculate different updates for different units (for example, when training using abandon), it is often desirable to initialize each unit so that it has a separate calculation function from all other units. This is true even if the training model or algorithm uses stochastics to generate different updates for different units. It is possible that this helps to ensure that no input patterns are lost in forward-spreading empty space, and that no gradient patterns are lost in backward-spreading empty space. A unique function calculation goal of each unit provides the impetus to randomize the order in which parameters are first set. We can do an explicit search for a large collection of core functions, all of which differ from one another, but this often results in a large increase in the required processing overhead.

For example, if we don't have more outputs than the same number of inputs, we can apply Gram-Schmidt orthogonalization to an initial weight matrix and make sure that each entity computes a significantly different function of the other entity. This would be the case if we had as many outputs as the maximum number of inputs. When using a random initialization derived from the high entropy distribution in a high-dimensional space, the two units are more likely to be unaffected by the computation of the same function. This method also reduces the amount of computation required. In most cases, we use heuristics to select constant values for each unit to use as a bias, and just assign random initial values for the weights. Additional parameters such as B. The parameters encoding the conditional variance of an estimate are usually set to heuristically chosen constants, similar to the deviation.

This is done for the same reason as choosing bias voltages. When we first run a model, we almost always specify all weight values randomly generated from a uniform or Gaussian distribution. Although not extensively studied, using a Gaussian or uniform distribution doesn't seem to make much of a difference. However, the size of the initial distribution plays an important role not only in determining the success of the optimization technique, but also in determining the extent to which the network can generalize its results. Higher initial weights will produce a greater symmetry breaking effect, which will help prevent the use of duplicate units. They also serve to prevent signal loss during forward or backward propagation along the linear component of each layer.

Larger values in the matrix yield larger outputs when the matrix is multiplied. On the other hand, very high initial weights can lead to explosive values in the forward or backward propagation process. Large weights can potentially wreak havoc on recurring networks if not properly distributed. (Hence the extreme sensitivity to small perturbations in the input, where the behavior of the deterministic forward propagation method seems random). Gradient clipping is a technique that can help alleviate the expanding gradient problem to some extent. (The threshold of gradient values before performing a gradient descent step). Large weights can also lead to extreme values leading to saturation of the activation function and lead to complete gradient loss in saturated units.

This can happen if the weights are too high. The interaction of these contradictory forces results in the proper initial equilibrium. When it comes to launching a network, regularization and optimization perspectives can provide very different insights into how we should go about it. However, there are some regularization issues with keeping the weights low, which goes against the optimization point of view, meaning that the weights have to be large enough to propagate information properly. Using an optimization algorithm such as B. Stochastic Gradient Descent that makes small incremental changes in weights and tends to stop at regions closest to the initial parameters (after getting stuck in a low gradient region or triggering an early stop). Over-fitting criteria), a premise states that the final parameters should be close to the initial parameters. This preliminary distribution is called the posterior distribution. Recall from section 7.8 that the equivalent weight drop for some models is an incline descent with an early stop.

The goal of the second heuristic is to strike a balance between the goals of starting all slices with the same trigger variance and starting all slices with the same gradient variance. The formula is built on the assumption that the mesh is non-linear and consists only of a set of matrices multiplied by each other. For real neural networks, this assumption is clearly wrong; However , many tactics developed for the linear model work satisfactorily when applied to their nonlinear equivalents. Initialization should be done with random orthogonal arrays as suggested. A scale or gain factor g must be chosen carefully and take into account the non-linearity applied to each layer. They arrive at unique values for the scaling factor for each of the many nonlinear activation functions they are considering.

This initialization approach is also inspired by a deep mesh model that defines it as a series of non-linear matrix multiplications. This initialization strategy ensures that the total number of training iterations required to achieve convergence is not affected by the depth of the dataset under investigation. Increasing the scaling factor g brings the network closer to the region where the activation norm increases as it propagates forward along the network, but the gradient norm increases as it spreads backwards across the network. Sussillo (2014) has shown that a correct gain factor configuration is sufficient to train neural networks up to a depth of 1000 without using vertical initializations. One of the key lessons of this method is the realization that in feed-forward networks, activations and gradients can expand or contract at each stage of forward or backward propagation, similar to the behavior of a random walk.

This is because feedforward networks implement a unique weight matrix for each layer. In general, if this random walk is defined in accordance with the standards, feedforward networks can avoid the extinction and burst gradient problem that arises when the same weight matrix is used at each stage. This issue is detailed in Section 8.2.5. Unfortunately, when these optimal criteria are used to determine take-off weight, ideal performance is often not achieved. There are three possible explanations for this phenomenon. For starters, we may be using the wrong criteria. It's possible that maintaining a network-wide signal standard doesn't really make sense. Second, the originally created attributes may not be preserved after the learning process has begun. Third, it is possible that the criterion successfully accelerates the optimization rate while inadvertently increasing the generalization error. In fact, we often need to think of equilibrium as a hyperparameter whose ideal value is fairly close to the theoretical estimates, but not exactly equal. In fact, the optimal scale value of the weights lies somewhere between these two extremes.

If available computational resources allow, it is generally a good idea to consider the initial scale of the weights at each level as hyperparameters and to randomly select these scales using a hyperparameter search technique mentioned in B. In other words, the initial weight scale for each layer should be considered as a hyperparameter. Another option for a hyperparameter is whether the initialization will be intensive or infrequent. It is also possible to manually search for scales that give the best initial results. When choosing initial scales, it makes sense to look at the firing range or the standard deviation of the gradients in a single mini dataset. This can serve as a solid rule of thumb. If the weights are not large enough, the deeper the activations go in the network, the narrower the activation range that occurs during the mini-stack.

Finally, it is possible to construct a network with acceptable initial activations by continuously identifying the first layer with unacceptably small activations and then increasing the weights of that layer. If learning is still too slow at this point, it might make sense to consider the range or standard deviation of the gradients in addition to the activations. This is possible if the slopes still change too much. Because it relies on feedback of the behavior of the first model to a single dataset, not feedback from a trained model to the validation set, this technique can in principle be automated and is generally less computationally intensive than a hyperparameter. Error-based optimization of the validation set. In fact, it is based on the behavior feedback of the original model. This longstanding heuristic protocol has only recently been made clearer. So far we have focused on the weighting process. Fortunately, initializing additional parameters is usually easier than initializing the first one.

In addition to these simple techniques for initializing model parameters, it is also possible to initialize model parameters by applying machine learning, for example, using B. constants or random values. In this book, we discuss a typical method called "Initialize a supervised model with parameters learned from an unsupervised model trained with the same inputs". This strategy involves populating the supervised model with parameters from the unsupervised model. Another option is to attend supervised training for a specific task. Even supervised training on an unrelated task can sometimes lead to an initialization that converges faster than a random initialization. This can happen even when tasks are completely independent of each other. Because they store distribution-related information in the initial parameters of the model, some of these initialization methods may result in faster convergence and more generalization than others. Others seem to work fine, mostly due to setting parameters to the correct scale or setting separate units to calculate functions separately from each other.

8.7 ADAPTABLE LEARNING FAST ALGORITHMS

Researchers studying neural networks have long known that learning speed is one of the most difficult hyperparameters to configure, as it has such a huge impact on model performance. Depending on what is covered, costs are usually quite sensitive to some aspects of the parameter space and relatively insensitive to others. The pulse method has the potential to alleviate some of these issues, but at the expense of one more hyperparameter. With that in mind, it's reasonable to ask if there is another way to do things. Assuming that the sensitivity directions are relatively aligned along an axis, it might make sense to use a different learning rate for each parameter and then automatically change those learning rates as the learning process progresses.

The delta bar delta method as the first heuristic solution to the problem of changing individual learning rates for model parameters during model training. This strategy is based on a simple concept: let the learning rate accelerate if the partial derivative of the loss with respect to a given model parameter has the same sign as before. If the partial derivative changes sign after this parameter, the learning rate should be slowed down. Obviously, the only type of optimization that can use such a rule is full batch optimization. More recently, various incremental (or mini-batch-based) approaches have been created that modify the learning rates of model parameters. These methods have proven themselves well. In this section, we'll take a closer look at some of these algorithms.

8.8 SETTING

The AdaGrad method, shown in , adjusts the learning rates of each model parameter by scaling it inversely to the square root of the sum of all previous square values. In contrast to parameters with relatively large partial derivatives, parameters with relatively small partial derivatives have a relatively moderate decrease in learning rates. Parameters with very large partial derivatives experience a relatively large decrease in learning rates. The end result is that we have made more progress in parameter space directions with smoother gradients. In the field of convex optimization, the AdaGrad method has a number of highly sought-after theoretical qualities. On the other hand, empirical research has shown that when training deep neural network models, the accumulation of square gradients at the beginning of training can lead to an unnecessarily and excessively slowing of the effective learning rate. This is the case with deep neural network model learning. AdaGrad works effectively for a variety of deep learning models, but not all.

The name Adam comes from the term "adaptive moments" from which it derives. In the context of the algorithms above, it might be easier to think of it as a variation of the combination of RMSProp and Momentum, but there are some key differences between the two. First, in Adam, momentum is included directly as an estimate of the momentum of the first-order gradient (exponentially weighted). Here's how it worked in the original algorithm. Applying momentum to scale gradients is the easiest and simplest way to incorporate momentum into RMSProp. There is no clear theoretical justification for using momentum in conjunction with a debt restructuring.

Second, Adam applies bias adjustments to the estimates of first-order moments (moment term) and second-order moments (off-center) to account for their initialization at the origin. This is done to account for the fact that first order moments are not centered. RMSProp also includes a quadratic (off-center) moment estimate; however, it does not include the correction factor in its calculations. Therefore, unlike Adam, the RMSProp quadratic moment estimation can show a large deviation early in the training process. While the learning rate should almost always be changed from the recommended default, Adam is generally considered to be very resistant to hyperparameter choices.

8.9 SELECTING THE RIGHT OPTIMIZATION ALGORITHM

In this part of the article, we looked at a group of similar algorithms, each of which attempts to overcome the challenge of developing deep models by adjusting the learning rate for each model parameter. At this point, the question arises which algorithm to choose. The sad truth is that there is currently no broad consensus on this. An insightful analysis of various optimization methods applied to various learning activities. While the results show that the family of adjustable learning rate algorithms (represented by RMSProp and AdaDelta) work quite well, no superior method has yet been identified. Currently, SGD, SGD with Momentum, RMSProp, RMSProp with

Momentum, AdaDelta and Adam are the best known and widely used optimization algorithms today.

At this point, the decision on which algorithm to use seems to depend on whether Newton's approximation can be used iteratively when dealing with non-square surfaces, provided the Hessian remains positive and unique. This suggests an iterative process consisting of two steps. To get started, refresh or recalculate the reverse Hessian. Second, make the necessary changes according to the parameters. We have explained that Newton's approximation is valid only when Hessian is positive definite. The surface of the objective function is usually not convex and has various properties like B. Saddle points that challenge Newton's deep learning approach. When not all Hessian eigenvalues are positive, for example near a saddle point, Newton's approximation can actually force updates to move in the wrong direction.

This can happen if not all Hesser eigenvalues are positive. It will be possible to prevent this situation by standardizing the canvas. Adding a symbolized constant along the Hessian diagonal is a common regularization strategy. The updated version is edited. This regularization approach is used in approximations to the Newtonian method such as the Levenberg-Marquardt algorithm and works quite well as long as negative Hessian eigenvalues are still very close to zero. Under conditions with more extreme directions of curvature, the value should be large enough to compensate for negative eigenvalues. Hesse, on the other hand, begins to dominate the diagonal I as growth increases, and as growth increases, the direction determined by Newton's technique begins to converge as the standard gradient divided by growth increases.

When there is a significant amount of negative curvature, it may need to be so large that using Newton's technique will result in smaller steps than using gradient descent at a reasonably defined learning rate. Conjugate gradients are a way to effectively avoid calculating the inverse Hessian by successively decreasing in conjugate directions. This approach adds to the barriers created by some aspects of the objective function. The impetus for the development of this strategy was an in-depth analysis of the shortcomings of the steepest descent technique for iterative line searches in a slope-related direction. Figure 8.6 shows how the steepest descent technique when used in a square pit continues with a rather inefficient back and forth zigzag motion. This pattern can be seen moving from the top of the shell to the bottom. This is because any line search direction, provided by the gradient, is guaranteed to be orthogonal to the direction in which the line search was previously performed.

Nonlinear Conjugate Gradients: So far we have talked about the conjugate gradient approach and how it is used to solve problems with quadratic objective functions. Naturally, the main goal we aim to achieve at the end of this chapter is to explore different optimization strategies that can be used to train neural networks and other similar deep learning models where the objective function is not square. The conjugate gradient approach can still be used in this environment, although some modifications are required. This may come as a surprise to some. Since there is no guarantee that the target is square, it is no longer possible to guarantee that the conjugate directions will retain the minimum target value set for the previous directions. Therefore, the nonlinear conjugate gradient algorithm includes periodic resets where the conjugate gradient technique is iterated with a line search along the invariant gradient.

Practitioners of the nonlinear conjugate gradient method report good results in neural network training applications; However, it is often advantageous to perform the optimization with a few cycles of stochastic gradient descent before starting nonlinear conjugate gradients. In fact, starting the optimization with stochastic gradient descent can often lead to better results. Also, although the (non-linear) conjugate gradient technique is often referred to as the batch approach, mini-batch variants of the algorithm have been well used to train neural networks. (Le et al., 2011). Previous adaptations of conjugate gradients designed specifically for neural networks have been developed, such as the scaling conjugate gradient method.

8.10 PARTY STANDARDIZATION

Batch normalization is one of the most exciting new breakthroughs in deep neural network optimization. However, it is by no means an optimization technique. Stack normalization is one of the most common. Instead, it is an adaptive reparameterization approach inspired by the challenge of training extremely deep models. Building very deep models requires combining multiple functions or layers. The gradient contains instructions on how to change each setting, assuming the information in other layers does not change. In fact, we will update each of the levels at the same time. When we apply the update, unexpected results may occur when many combined features are changed at the same time. These updates are calculated with the assumption that other features will remain the same.

8.11 SUPERVISED PRE-EDUCATION

In some cases, simply training a model to solve a particular problem may be overly ambitious. This is especially true when the model is complex and difficult to solve, or when the problem to be solved is very difficult. When trying to solve a problem, it is more helpful to train a simpler model than to make it more complex. It may be more effective to train the model by first solving a smaller problem and then moving on to the more difficult problem. The common name for these methods of training simple models for simple tasks before overcoming the challenge of training the model required to perform the intended task is called pre-training. Greedy algorithms break a problem into its many components, then work independently to look for the best possible solution for each component. Unfortunately, there is no guarantee that combining the best parts will generally result in an optimal solution.

However, greedy algorithms are much cheaper to run on a computer than algorithms that seek the most common answer, and the quality of a greedy solution is often satisfactory, if not the best possible. It is also possible that greedy algorithms are followed by a fine-tuning phase involving a common optimization algorithm that seeks the best possible solution to the entire problem. If you start the co-optimization method with a greedy solution, you can greatly improve both the speed of finding solutions and the quality of those solutions. Deep learning relies heavily on pre-training algorithms, especially greedy pre-training algorithms. Pre-workout techniques are everywhere. In this section, we will clearly explain the pre-training techniques used to decompose supervised learning problems into other minor supervised learning problems.

The method in question is called enthusiastic supervised preliminary training. In the first implementation of supervised greedy pre-training proposed by Bengio et al. In 2007 each phase consisted of a supervised learning training activity that included only a portion of the layers that would eventually form the final neural network. Figure 8.7 is an illustration of an example of supervised greedy pre-learning, where each new hidden layer is pre-trained as part of a supervised MLP plan, accepting the output of the pre-trained hidden layer as input. This is an example of willingly supervised pre-training. A method in which they create a deep web of folds with eleven layers of weights. They then used the first four and last three levels of this network to initiate even deeper networks.

This process was superior to the traditional one layer at a time preforming process. (up to nineteen weights). The randomization process starts in the middle levels of the brand new very deep web. Then the new network goes through co-creation. Yu et al. (2010) is to use the outputs of pre-trained MLPs plus raw input as input for each additional step added. Why can guided gourmet pre-training be useful? The idea that a deep hierarchy helps to better orient their middle levels was initially put forward and discussed. In general, pre-workout can be useful both for optimizing performance and for generalizing these improvements. The concept has been extended to the transfer learning framework using a method associated with supervised pre-learning: a deep convolutional network pre-learning with 8 weight levels for a set of activities (a subset of 1000 ImageNet object categories) followed by initialization, they form a network. same size using the first k layers of the first mesh.

This allowed both networks to have equal accuracy. All layers of the second network are then trained together to perform a new task path (another subset of thousands of ImageNet object categories) using fewer training samples than was used for the previous task path. This is done by starting the upper layers of the second network in random order. Other methods of using neural networks for learning transfer are analyzed and explored. The FitNets technique (Romero et al., 2015) is another area of study in this area. This strategy starts with creating a mesh that is shallow enough in depth and large enough in width (number of units per layer) to be easily constructed.

This network then takes on the role of the teacher for a second network called the student. Since the student network is much more complex and has fewer levels (from eleven to nineteen), it will be difficult to train it with SGD under typical conditions. Teaching the student network not only to predict the output of the original work, but also to predict the intermediate value of the teacher network makes it easier to train the student network. In fact, by teaching the student network to predict the output of the original assignment, it is also trained to estimate the value of the middle tier of the teacher network. This additional activity provides a set of tips on how to use hidden layers and has the potential to help solve the optimization problem. Some additional factors come into play to align the middle layer of the 5-level teacher network with the middle layer of the deeper student network: the goal is not to predict the ultimate goal of classification; instead, it aims to provide the hidden middle layer of the teacher network.

Thus, the lower layers of the student network serve two purposes: first, to help the outputs of the student network do their job, and second, to predict the performance of the middle layer of the teacher network. While training a deep sparse network may seem more difficult than training a shallow and wide network, a deep sparse network is better generalizable and certainly has a lower computational cost if it is sparse enough to have far fewer parameters. This is only true if the network is deep enough to have fewer parameters. The student network performs quite poorly in experiments where hidden level signals are not included, and this applies to both the training set and the test set. Intermediate hints may be one of the tools that can help train neural networks, which at first glance may seem difficult to train. But other optimization approaches or design changes can also fix the problem.

8.12 DESIGN OF MODELS SUPPORTING OPTIMIZATION

This is not always an ideal approach to improve the optimization process for better optimization results. Instead, many advances in deep model optimization come from making models easier to optimize. This has led to remarkable developments in this field. Theoretically, we can use activation functions with irregular non-monotonic growth and decay patterns in the values they produce. But that would make optimization an incredibly difficult task. In reality, it is more necessary to choose a family of simple models for optimization than to resort to an efficient optimization procedure. Much of the progress in learning neural networks over the past three decades has been the result of model family modification rather than process optimization.

The Momentum stochastic gradient descent algorithm, which was used in neural network training in the 1980s, is still used in newer neural network applications, considered cutting-edge. In fact, the choice to use linear transformations between layers and activation functions that can be differentiated practically everywhere and to have a significant slope in large portions of their domains is reflected in the architecture of existing neural networks. This option was chosen as part of the design process. In particular, advances in models such as LSTM, rectified linear units, and Maxout units have all evolved to adopt more linear functions than previous models, such as deep meshes based on sigmoid units. These templates offer great features that make optimization much easier.

If the Jacobian of the linear transform has suitable singular value parameters, the gradient will flow over multiple layers. In addition, linear functions always increase in the same direction, so even if the model output is far from it should be, only the gradient can be calculated to determine which direction the model output should move. .] to minimize loss function. In other words, modern neural networks are built in such a way that information about their local gradient correlates quite well with the process of moving to a more distant solution. The optimization process can be simplified using other model design methods. For example, linear paths or jump connections between levels can reduce the length of the shortest path between the lower level settings and the output, which can help reduce the vanishing gradient problem.

Regarding the concept of skipping links, there is the practice of adding extra copies of the outputs that refer to the hidden middle layers of the network, such as B. in Google Net and deeply monitored networks. This makes the network ignore connections. These "secondary headers" are designed to function the same as the main outlet at the top of the mesh to provide a distinct slope to the lower layers. Extra heads can be removed after formation is successfully completed. This is an alternative approach to the pre-workout tactics discussed in the previous section. With this method, it is possible to create all layers simultaneously in one phase and also to change the architecture, so that the middle layers, especially the lower layers, can receive signals about what to do.

A significant additional contribution has been made to curriculum learning research in the context of training recurrent neural networks to capture long-term dependencies: Zaremba and Sutskever (2014) found that a stochastic curriculum provides significantly better outcomes than a traditional course. In a stochastic program, the student is always presented with a random mix of easy and difficult examples; however, the average proportion of more difficult samples, ie samples with long-term dependencies, gradually increases throughout the training. No improvement has been shown on the standard training set using a deterministic program.

The basis was considered a "joint formation". Now that we've covered the basics, including optimization and regularization of neural network models, let's look at some examples. In the next few chapters, we'll discuss the specializations of the neural network family. These specializations allow neural networks to scale to extremely large sizes and process input data with a specific structure. The optimization techniques discussed in this section can often be applied directly to these specific projects with little or no additional modifications.

Editors Details

ISBN: 978-93-94707-79-5

Mr. Amol Dattatray Dhaygude is renowned professional in field of Machine Learning, Artificial Intelligence, Data Science and Computer Science. He is alumni of University of Washington, Seattle, USA with Master of Science in Data Science and specialization in Machine Learning. Amol has 16 years of software industry experience in top tier organizations including IBM, Cognizant and Microsoft Corporation. He is currently employed at Microsoft Corporation for last 10 years in role of Senior Data & Applied Lead at Redmond, Washington. He is inspired to make use of cutting edge technological advancements in field of Machine Learning and Artificial Intelligence to solve real world practical problems making a difference to world. He has has strong techno business acumen to formulate and solve business problems with applications of Data Science, Machine Learning and Artificial Intelligence. He is well versed in Deep Learning, Natural Language Processing, and Computer Vision fields of Artificial Intelligence. Amol celebrates growth mindset with continuous learning, embracing challenges, experimentation, fail fast, feedback and continuous improvement principles. He believes in learning from community and at the same time giving back to community by sharing his knowledge through various avenues such as research publications, blogs, patent publications, book publishing etc. Amol has continuously served as editor for books and journals in his areas of expertise.

Dr. Pushpendra Kmar Verma, is an Associate Professor, School of Computer Science and Applications, IIMT University, Meerut, UP, India. Hehasdone MCA, MTech-CSE, MPhil (CS) and Doctorate in Computer Science. His area of research is in Cryptography and Network and Security/Cyber Security, and his other areas of specialization include Artificial Intelligence and Machine Learning. He has Convener, Keynote Speakers and participated in several high-profile conferences,seminars and workshops. The author has many research papers in international journals, reviewarticles in various Scopus Indexed Journals, international journals and interested in academics and research. He has !7+years extensive teaching and research experience in various Universities as well as colleges across India. He also reviewed the Reputed International Journal Papers.He is member of Vidwan, Springer, International Association of Engineers (I-AENG),Society of Digital Information and Wireless Communications (SDIWC) etc.

Dr. Sheshang D. Degadwala is presently working as Associate Professor and Head of Computer Engineering Department, Sigma University , Vadodara. He has published 235 research papers in reputed international journals and conferences including IEEE, Elsevier and Springer. His main research work focuses on Image Processing, Computer Vision, Information Security, Theory of Computation and Data Mining. He is also Microsoft Certified in Python Programming and Excel. He has published 18 books and he got grant for 3 patent. He has published 125 Indian Patent. He has received 50 awards for academic and research achievement.

Renato Racelis Maaliw III is an Associate Professor and currently the Dean of the College of Engineering in Southern Luzon State University, Lucban, Quezon, Philippines. He has a doctorate degree in Information Technology with specialization in Machine Learning, a Master's degree in Information Technology with specialization in Web Technologies, and a Bachelor's degree in Computer Engineering. His area of interest is in artificial intelligence, computer engineering, web technologies, software engineering, data mining, machine learning, and analytics. He has published original researches, a multiple time best paper awardee for various IEEE sanctioned conferences; served as technical program committee for world-class conferences, author, editor and peer reviewer for reputable high-impact research journals.

Xoffencer International Publication
838- Laxmi Colony. Dabra,
Gwalior, Madhya Pradesh, 475110
www.xoffencerpublication.in

MRP: ₹ 450/-
ISBN
9 789394 707795