AI Sciences

Deep Learning Fundamentals

An Introduction for Beginners

AI SCIENCES

Chao Pan

DEEP LEARNING FUNDAMENTALS

An Introduction for Beginners

Chao Pan

AI SCIENCES

How to contact us

Please address comments and questions concerning this book to our customer service by email at:

contact@aisciences.net

Our goal is to provide high-quality books for your technical learning in computer science subjects.

Thank you so much for buying this book.

If you noticed any problem, please let us know by sending us an email at review@aisciences.net before writing any review online. It will be very helpful for us to improve the quality of our books.

AI SCIENCES

Table of Contents

First Printing, 2018

Edited by Davies Company
Ebook Converted and Cover by Pixels Studio
Publised by AI Sciences LLC

ISBN-13: 978-1721230884
ISBN-10: 1721230882

Legal Notice:

Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. No warranties of any kind are expressed or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, errors, omissions, or inaccuracies.

AI SCIENCES

- Do you want to discover, learn and understand the methods and techniques of artificial intelligence, data science, computer science, machine learning, deep learning or statistics?
- Would you like to have books that you can read very fast and understand very easily?
- Would you like to practice AI techniques?

If the answers are yes, you are in the right place. The AI Sciences book series is perfectly suited to your expectations!

Our books are the best on the market for beginners, newcomers, students and anyone who wants to learn more about these subjects without going into too much theoretical and mathematical detail. Our books are among the best sellers on Amazon in the field.

About Us

We are a group of experts, PhD students and young practitioners of Artificial Intelligence, Computer Science, Machine Learning and Statistics. Some of us work in big companies like Google, Facebook, Microsoft, KPMG, BCG and Mazars.

We decided to produce a series of books mainly dedicated to beginners and newcomers on the techniques and methods of Machine Learning, Statistics, Artificial Intelligence and Data Science. Initially, our objective was to help only those who wish to understand these techniques more easily and to be able

to start without too much theory and without a long reading. Today we also publish more complete books on some topics for a wider audience.

About our Books

Our books have had phenomenal success and they are today among the best sellers on Amazon. Our books have helped many people to progress and especially to understand these techniques, which are sometimes considered to be complicated rightly or wrongly.

The books we produce are short, very pleasant to read. These books focus on the essentials so that beginners can quickly understand and practice effectively. You will never regret having chosen one of our books.

We also offer you completely free books on our website: Visit our site and subscribe in our Email-List: www.aisciences.net

By subscribing to our mailing list, we also offer you all our new books for free and continuously.

To Contact Us:

- Website: www.aisciences.net
- Email: contact@aisciences.net

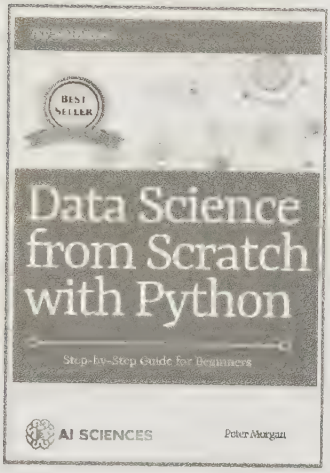
Follow us on social media and share our publications

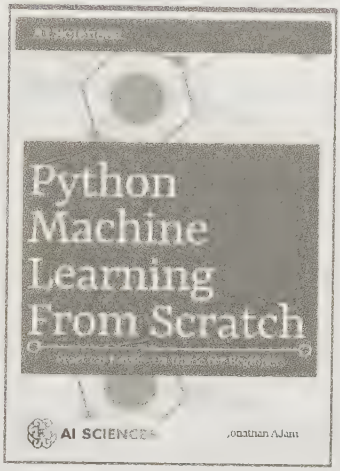
- Facebook: **@aisciencesllc**
- LinkedIn: **AI Sciences**

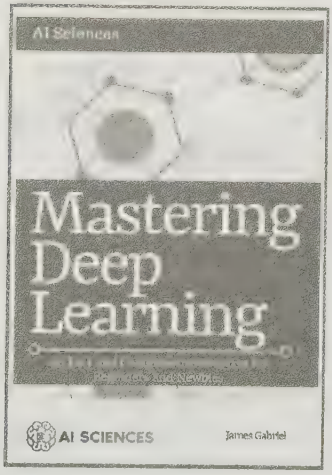
Author Biography

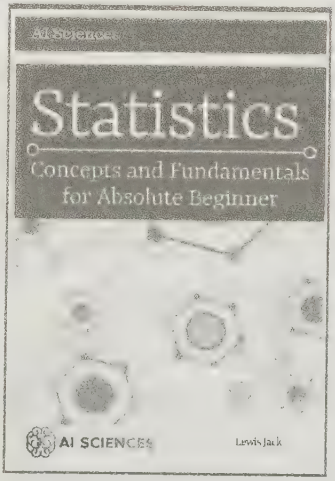
Chao Pan is a computer scientist and entrepreneur. After finishing his undergraduate degree in mathematics at MIT, he went on to do quantitative analysis work at P&G in Ohio and Facebook in Palo Alto. He now works for his own consulting company and helping big companies in their marketing and statistical analysis using machine learning and deep learning techniques.

From AI Sciences Publishing









WWW.AISCIENCES.NET

EBooks, free offers of eBooks and online learning courses.

Did you know that AI Sciences offers free eBooks versions of every books published? Please subscribe to our email list to be aware about our free eBook promotion. Get in touch with us at contact@aisciences.net for more details.

AI SCIENCES

PUBLISHING

AI SCIENCES LLC

At www.aisciences.net , you can also read a collection of free books and receive exclusive free ebooks.

WWW.AISCIENCES.NET

Did you know that AI Sciences offers also online courses?

We want to help you in your career and take control of your future with powerful and easy to follow courses in Data Science, Machine Learning, Deep learning, Statistics and all Artificial Intelligence subjects.

Most courses in Data science and Artificial Intelligence simply bombard you with dense theory. Our course don't throw complex maths at you, but focus on building up your intuition for infinitely better results down the line.

AI SCIENCES
ACADEMY
AI SCIENCES LLC

Please visit our website and subscribe to our email list to be aware about our free courses and promotions. Get in touch with us at academy@aisciences.net for more details.

Preface

"Humanity is on the verge of digital slavery at the hands of AI and biometric technologies. One way to prevent that is to develop inbuilt modules of deep feelings of love and compassion in the learning algorithms."

— Amit Ray, Compassionate Artificial Superintelligence AI 5.0 - AI with Blockchain, BMI, Drone, IOT, and Biometric Technologies

The main purpose of this book is to provide the reader with the most fundamental knowledge of deep learning so that they can understand what these are all about.

Why the AI Sciences Books are different?

The AI Sciences Books explore every aspect of Artificial Intelligence and Data Science using computer Science programming language such as Python and R. Our books may be the best one for beginners; it's a step-by-step guide for any person who wants to start learning Artificial Intelligence and Data Science from scratch. It will help you in preparing a solid foundation and learn any other high-level courses will be easy to you.

Step-by-Step Guide and Visual Illustrations and Examples

This book and the accompanying examples, you would be well suited to tackle problems, which pique your interests using machine learning and deep learning models.

Instead of tough math formulas, this book contains several graphs and images.

Book Objectives

This book will help you:

- Have an appreciation for deep learning and an understanding of their fundamental principles.
- Have an elementary grasp of deep learning concepts and algorithms.
- Have achieved a technical background in deep learning and neural networks.

Who Should Read This?

Deep learning is quickly becoming the dominant branch of machine learning as it continues to produce spectacular results across a range of tasks and benchmarks. Almost all commercial applications of machine learning today incorporates deep learning to drive higher returns as it has proven to be a game changer in the world of artificial intelligence by making possible tasks that were deemed impossible for machines to perform as recently as five years ago. This book is written to give insights into this relatively nascent field of computer science. The following categories of people would find this book and the techniques in it particularly useful:

- The absolute beginner who has heard or read news items about the awesome powers of deep learning and now seeks to understand the algorithms that underpin the field. This book would take such a reader through a journey that transforms him/her from interested

spectator to a practitioner with a set of tools to solve challenging problems.

- The experienced software developer or programmer who has tons of experience in various programming stacks but desires to prepare adequately for the upcoming industrial revolution that would be lead by Artificial Intelligence. This reader can expect to draw on experience from their years of programming but needs to keep an open mind as deep learning techniques offer a completely different approach to making machines do what the users intend as opposed to traditional programming.

- The experts or veterans who have been in the field of artificial intelligence and machine learning but require a detailed overview of cutting edge techniques in deep learning. The concepts from other fields of machine learning are easily transferable and such a reader would typically pick up concepts faster and can use this book as a reference guide or implementation manual.

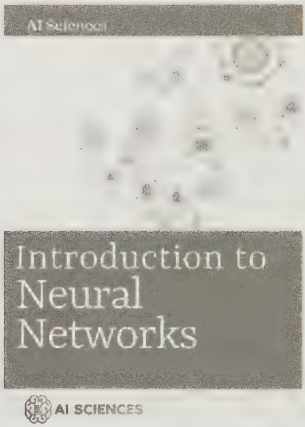
Why this book?

This book is written to help you learn deep learning. If you are an absolute beginner in this field, you'll find that this book explains complex concepts in an easy to understand manner without math or complicated theorical elements. If you are an experienced data scientist, this book gives you a good base from which to explore deep learning application.

Topics are carefully selected to give you a broad exposure to deep learning application. While not overwhelming you with information overload.

Your Free Gift

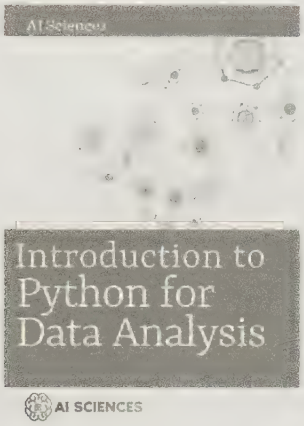
As a way of saying thanks you for your purchase, AI Sciences Publishing Company offering you a free eBook in Neural Networks written by the data scientist François Duval.



It is a full book that contains useful introduction to Neural Networks. It is 120 pages book. AI Sciences encourage you to print, save and share. You can download it by going to the link below or by clicking in the book cover above.

http://aisciences.net/free-book-ann/

AI Sciences Publishing Company offers you also a free eBook in Introduction to Python for Data Analysis written by Robert Danboard.



It is a full introduction book in Python for Data Analysis. It is a 100 pages book. AI Sciences encourage you to print, save and share. You can download it by going to the link below or by clicking in the book cover above.

http://aisciences.net/free-book-3/

If you want to help us produce more material like this, then please leave an honest review on amazon. It really does make a difference.

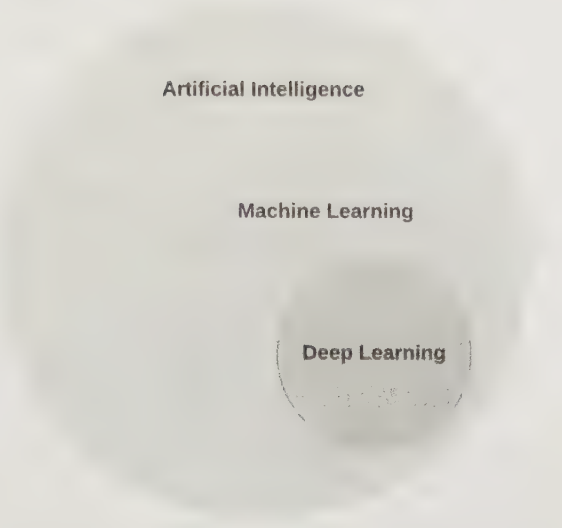
Introduction

Teaching Approach

The teaching approach of this book involves a low level explanation of core concepts, then a composition of these core concepts into more complex representations such that the reader is always exposed to a level of difficulty that is in tune with what they are already comfortable with. While this book employs a hands on approach, it does not trade best practices for simplicity. The mathematical foundations required to develop a great mental picture are not glossed over but are explained in an intuitive way using examples and analogies. To get the most out of this book, users are encouraged to follow the steps outlined to run the algorithms presented in this book. It is therefore advised that this book be read alongside an open editor, notebook or terminal so as to enable experimentation because it has been observed that learning a new skill is more about practicing than it is about passively gaining knowledge.

What is Artificial Intelligence, Machine Learning and Deep Learning?

The fields of artificial intelligence, machine learning and deep learning are often used interchangeably in press articles because they all seem to relate to the same thing. However, this is not strictly true as they are concerned with solving slightly different problems using different techniques. A good way to think about these fields is to know that deep learning is a subfield of machine learning and machine learning in turn is a subfield of artificial intelligence.



Artificial intelligence is a branch of computer science that aims to create machines, which are capable of demonstrating decision-making processes or problem solving of the kind usually associated with humans through the display of natural intelligence. Artificial intelligence therefore tries to turn computer programs into intelligence agents that are capable of understanding their environment and taking actions in such a way that increases their ability to achieve success at a predefined task. The study of such programs usually focus on agents that display a level of cognitive ability such as hearing, seeing, making decisions, solving puzzles etc. It is an embodiment of various fields including machine learning, representation learning and deep learning. Artificial Intelligence has been an active area of academic research since the 1950s when researchers sought to build machines that could perform tasks with human level intelligence. The term "artificial intelligence" was coined by the American computer

scientist John McCarthy and he with several others such as Marvin Minsky, Allen Newell and Herbert A. Simon are regarded as the founding fathers of AI.

Machine learning on the other hand is a subfield of artificial intelligence that deals with making computer programs improve their performance at a set of specific tasks through experience. This experience comes in the form of data that is made available to the machine learning algorithm. The main difference between machine learning and other approaches is that in machine learning the programs are not explicitly programmed, rather the algorithms are taught to improve their performance over time through exposure to more training examples (data). Machine learning algorithms usually employ statistical techniques to learn representations of the underlying data structure thereby giving the algorithms the ability to make predictions in future. The term "machine learning" was first used in 1959 by Arthur Lee Samuel who is also famous for his Samuel Checkers-playing program which involved an algorithm that learnt how to play the game of checkers on its own without being explicitly programmed to obey any hand crafted rules.

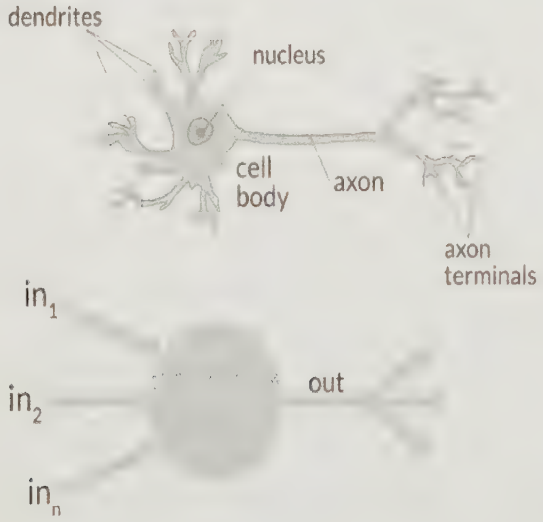
Deep learning is the new kid on the block although it has been known under different names such as cybernetics and connectionism. Deep learning is a class of machine learning algorithms that at their core learn data representations across multiple layers. With higher layers learning more meaningful representations usually composed of simpler representations discovered by earlier layers. As a result of this hierarchy in the

way features are learnt from data, deep learning is sometimes referred to as hierarchical learning or deep structured learning. It must be noted that the "deep" in deep learning is a pointer to the fact that the algorithm is composed of many layers so it is several layers "deep". Deep learning algorithms find a mapping function from a set of inputs to a set of outputs. The main algorithm that makes this mapping possible is known as a deep neural network.

Neural networks and Deep Learning

An Artificial Neural Network (ANN) is a machine learning algorithm that is roughly modelled around what is currently known about how the human brain functions. The brain of humans consist of billions of neurons interconnected to form neural networks which are somehow responsible for carrying out cognitive tasks such as identifying familiar faces, grouping entities into categories, solving puzzles, making decisions etc. These neurons communicate with each other through electrical signals that are fired to interconnected neurons. When a set of neurons are triggered by an event which they find interesting, the connections between them are strengthened such that it becomes easier for those neurons to easily identify that event when it occurs in future. Learning usually takes place in this way through the strengthening or weakening of connections. While this is an over-simplified way to think about how the brain functions, for our purposes it provides us with a clear intuition on why deep neural networks are structured the way they are.

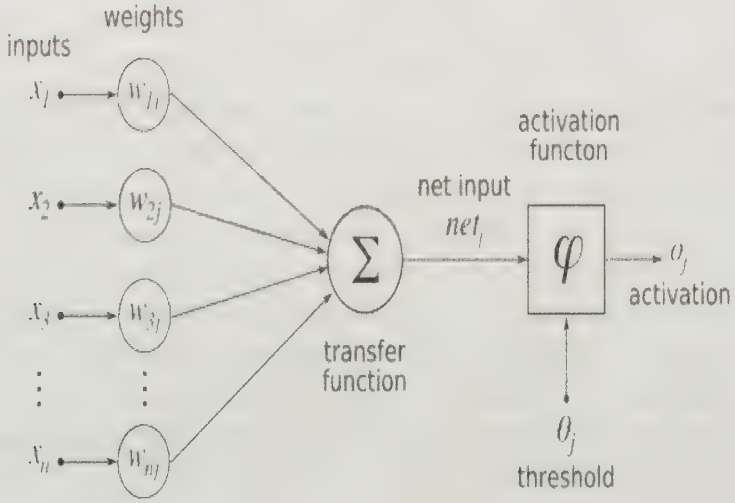
Let us briefly look at a single artificial neuron to understand its operations and develop our intuitions further. After that, we would translate the concepts learned to design an artificial neuron model.



The image above shows a single biological neuron with its component parts. A neuron is structured in such a way that it receives several inputs from neighboring neurons. These inputs provide information that has been sent in the form of electrical signals in from other parts of the brain. Dendrites are the gateway through which information is fed into a neuron. The neuron then has to determine whether the information received is sufficient for it to be excited in order to send an electrical signal down the line to indicate the presence of a phenomenon which it is interested in detecting. The chemical composition of the cell body which includes the nucleus

decides whether to fire the signal or not based on the feedback it received from neurons earlier in the chain. The axon is the part of the neuron that is responsible for relaying the information processed in the cell body to other neurons. Neurons act in conjunction with each other to provide cognition such as a new born baby recognising the face of its mother. The bottom part of the image shows a simplified representation of a neuron using a schematic diagram of arrows and nodes.

Now let us formulate the same principles using mathematical notation to describe an artificial neuron.



In the diagram above x_1 to x_n can be seen as information being carried into the neuron in the same way dendrites feed in information to a biological neuron in the form of electrical

signals. Here the information is in the form of features (x) which describe the attributes of a data point. The weights are the parameters that modulate the strength or importance of certain features feed into the neuron. The biological equivalent are synapses (connections between neurons) which are strengthened or weakened depending on the current cognition task. The transfer function can be as the cell body into which all signals flow. In the case, the transfer function is a summer that adds together all features in a weighted fashion. After this linear addition, the computation is passed to an activation function whose role is similar to a biological neuron deciding whether to fire the electrical signal or not. In the artificial case, if the computation of the activation function satisfies some threshold value, the computation is sent as the output else it is suppressed.

What a deep neural network does is to arrange multiple neurons into several layers with the outputs of one layer serving as inputs to the next layer. This makes the deep neural network capable of learning very complex functions especially when it incorporates non-linear activation functions. An artificial neural network can approximate any continuous function by adding more nodes (neurons) to a layer and by assumptions encoded in the activation function. This phenomenon is expressed as the universal approximation theorem.

Why is Deep Learning Popular

Deep learning has become popular mainly because of the impressive results that it has attained in several learning

problems. Unlike traditional machine learning algorithms whose performance seem to plateau beyond a certain point regardless of the availability of data, deep learning models generally get better with more data. The factors for the meteoric rise of deep learning are summarized below:

- Data: The availability of data which has been ushered in large part by the internet has lead to more data being collected in the past few years than at any point in human history. The amounts of data being stored is gigantic and this has lead to a wave of deep learning applications. A good example is in the task of image detection. With billions of natural images now available on the internet, deep learning models have been trained on millions of well labelled examples to the extent that they now outperform humans at the task of image classification. This is true across other learning problems where big data is now available to train large models.

- Hardware: The processing speed of computers has grown exponentially over several years in line with Moore's law. This has enabled larger computations to run on more powerful processors in a shorter period of time. The creation of Graphics Processing Units (GPUs) which are generally 50 times faster than Central Processing Units (CPUs) at deep learning tasks made it possible to train larger models on bigger datasets. GPUs which were originally designed to render graphics in the gaming industry have proven to

be specially suited to parallel computation of the kind which is predominant in deep learning as feature learning is usually separated into small chunks and computed via a distributed infrastructure.

- Algorithms: Recent advances in deep learning algorithms is also responsible for the uptick in funding and research activities. New algorithms with multiple real world use cases have been invented such as Generative Adversarial Networks (GANs) which can be used to generate artificial images that are indistinguishable from professionally taken photographs. Attention mechanism for Recurrent Neural Networks (RNNs) is helping machine translation and speech recognition systems achieve results that are mind blowing. The value created by these discoveries has lead to companies investing heavily in AI talents as almost every technological product would be impacted in some way by this movement.

A Brief History of Deep Learning

The field of deep learning has a long and rich history dating back to the 1940s. In 1943, Walter Pitts and Warren McCulloch launched the first wave of biological inspired learning models through the creation of the McCulloch-Pitts neuron which was an artificial linear computation model of a biological neuron. It was capable of representing a simple decision process and used what they called "threshold logic". In 1958, American

psychologist Frank Rosenblatt extended the original techniques shown by the McCulloch-Pitts neuron to learn the weights that helped determine the category of inputs fed into the model. He called it the Perceptron and it used a training algorithm known as the Adaptive Linear Element (ADALINE). The perceptron algorithm generated great interests across academia and funding poured in for these biological learning methods.

However, it was soon proven by Marvin Minsky that the perceptron could not learn some simple functions like XOR (Exclusive OR) and interest in the field waned. The field was known as cybernetics throughout the 1940s to 1960s. This lack of interest resulted in an extended period of little or no research in the field.

The next wave of deep learning was known as connectionism and occured in the 1980s and early 1990s when the backpropagation algorithm was invented by David Rumelhart, Ronald Williams, and Geoffrey Hinton. Backpropagation showed that a neural network could be trained by back propagating the errors so as to provide credit assessment as to which nodes contributed to a final prediction or output. Backpropagation uses the chain rule and together with gradient descent reduces the error contained in the prediction by adjusting the weights slightly at each iteration. In 1989, Yann Lecun demonstrated the first practical use of backpropagation algorithm when at Bell Labs he trained Convolutional Neural Networks (CNNs) to recognize handwritten digits. However, the promises offered by artificial neural networks to solve

various learning tasks were not fulfilled because training datasets were very small and computers too slow. This lead to another winter during which research in these techniques were stifled.

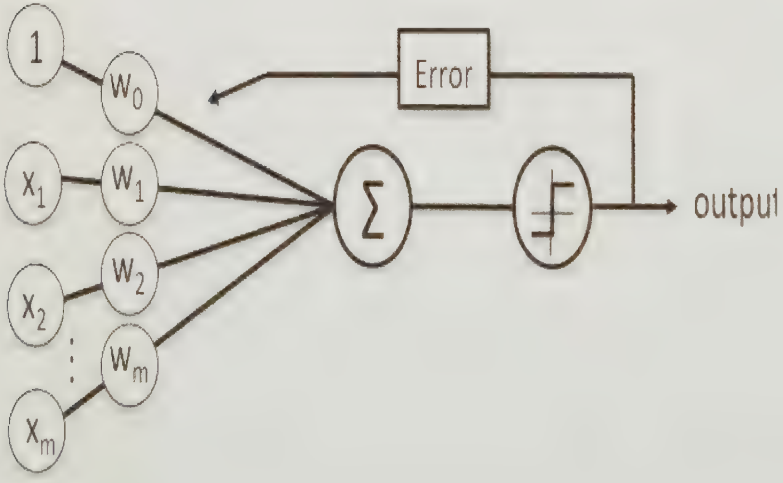
The next major turning point came in 2006 when Geoffrey Hinton invented Deep Belief Networks which showed that deep neural networks that were several layers deep could be trained efficiently using greedy layer-wise pretraining. This breakthrough sparked renewed research in neural networks. Then in 2012, Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with Alexnet, a Convolutional Neural Network that reduced the previous error from 26% to 15.3%. The error gap between the winning model and the second placed model which was not a Convolutional Neural Network was 10.9%. This heralded the current rise of deep learning as researchers and companies tried deep learning models on various benchmarks with state of the art performance being recorded across domains.

Mathematical Foundations of Deep Learning

Introduction to Perceptron

The perceptron learning algorithm in many ways can be seen as the precursor of deep neural networks. The perceptron was invented by Frank Rosenblatt who showed that it was capable of learning simple classification tasks provided there existed a set of weights in the solution space that could correctly separate samples into classes via a straight line.

Let us now use the diagram below to walk through the components that made the perceptron learning algorithm novel for its time.



In the perceptron classifier as shown above, the features X_1 to X_m are the inputs to the algorithm. The node containing the number 1 represents the bias fed into the learning algorithm. The bias can be seen as the intercept from the equation of a straight line. The weights which are the trainable parameters of the model are multiplied with the inputs. They therefore directly determine the importance of features as relates to getting a correct prediction or output.

The equation of a straight line is given by:

$$y = Wx + b$$

Where y represents the output or prediction of the model. W represents the weights of the model, often referred to as the slope when talking about the equation of a straight line but in our case they are the trainable parameters that are learnt by the algorithm. The intercept which determines by how much we shift the plane is represented by the bias term b.

The reason we talked about the equation of a straight line is because the perceptron learning algorithm is a linear classifier, that is it uses a straight line to separate data points into classes. The perceptron is particularly efficient as a binary classifier where both classes are linearly separable. In such a case, it is guaranteed to find a plane that correctly predicts all samples.

When the inputs are first fed into the algorithm, the weights are randomly initialized to very small numbers. The weights are multiplied with the features (X) and the result of this computation for each input is summed linearly. The resulting net input (the summation) is then fed to an activation function. The activation function used in the perceptron learning algorithm is the unit step function. The unit step function separates samples into classes based on a threshold value. All data points whose computation through the model are greater than zero belongs to one class while the remaining points whose computation values are less than zero (negative) belongs to the second class. For each predicted classification, it then looks at the ground truth, that is the correct label to see if it correctly classified the sample. If it did get the correct class, it makes no updates to the weights as relates to that sample. However, in the case where it misclassified the sample, it updates the weights in such a way that it pushes the computation towards the correct class such that on subsequent passes during training, it stands a high chance of correctly predicting the class label for that particular sample.

The perceptron learning algorithm is easy to understand because it makes use of an error signal which is computed as the difference between the true class label and the predicted class label. The error signal then determines by how much the weights should be updated and adjusts this set of weights in such a way that it finds an optimal set of weights that correctly predicts all samples in the dataset. The simplicity of the perceptron learning algorithm also has several drawbacks as we are assuming that there exist a set of weights in the hypothesis space that adequately separates classes. Second, we are also

assuming that the dataset is linearly separable into distinct classes. This is not always the case as was famously demonstrated that the perceptron cannot learn the XOR gate because the nature of the data distribution of a XOR gate is such that its samples are not separable by one straight line.

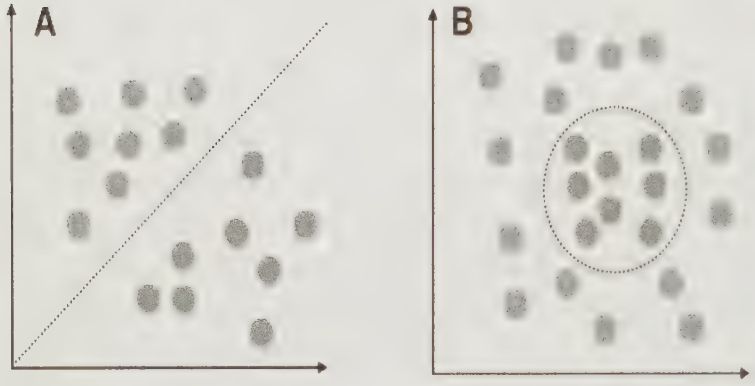
The intuition to take out of this is that to model more complex functions, we need to make use of non-linear activation functions because they are inherently capable of more complex representations. In simple terms, they help us extend our hypothesis space such that we are more likely to come across a set of parameters that solve the learning task as defined by the data.

The limitations of the perceptron learning algorithm notwithstanding, it inspired later models which were modifications that included more neurons, non-linear activation functions and a different kind of learning rule known as gradient descent. These extensions are how artificial neural networks were born.

Activation Functions

As we saw with the perceptron, it is important that we have the ability to model complex representations such that we can separate samples into classes using hyperplanes that are not linear. What non-linear activation functions do is to perform geometric transformations from one coordinate to another. After several of such transformations, the data points are now

distributed in a way that they are easily separable into various classes.



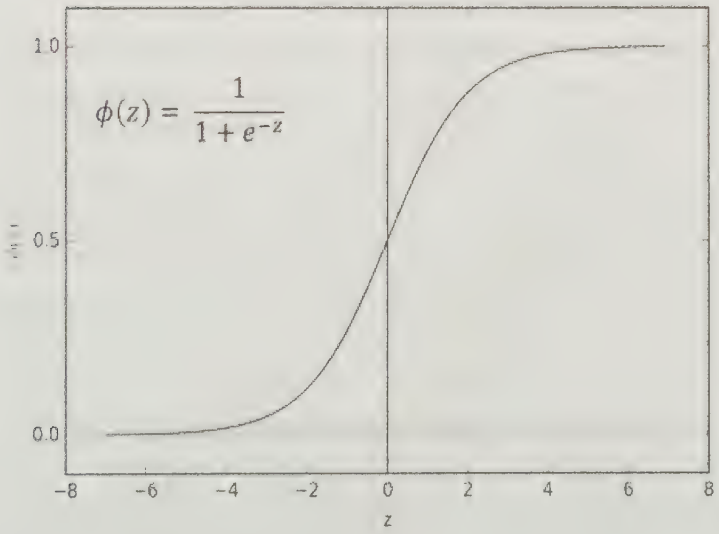
In the diagram above, graph A presents a binary classification problem which is linearly separable. What that means is that we can find a straight line that correctly separates both classes. Linear classifiers like the perceptron learning algorithm which we have discussed can handle such a case because the nature of the data is such that trainable parameters (weights) can be learnt since this a simple linear classification problem. However for graph B, we need a more powerful representation that would be able to categorize data points since this cannot be achieved with the use of a straight line. This is what motivates the use of non-linear activation functions, so that these more complex data distributions can also be learnt.

There are several non-linear activation functions used in deep neural networks to enable the network break linearity such as sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit

(ReLU) etc. Each of these functions are useful for a range of learning tasks but they are more suited for some tasks than others. Let us have a look at each of them.

Sigmoid Activation Function

The sigmoid activation function is mainly used for binary classification tasks where we have two mutually exclusive classes. The main intuition is that it squashes values from a large range of possibilities into a smaller range that is continuous and bounded. Since sigmoid is a continuous function, it has the advantage of being differentiable, that is we can take the derivative and use it in backpropagation during the training of a deep neural network.



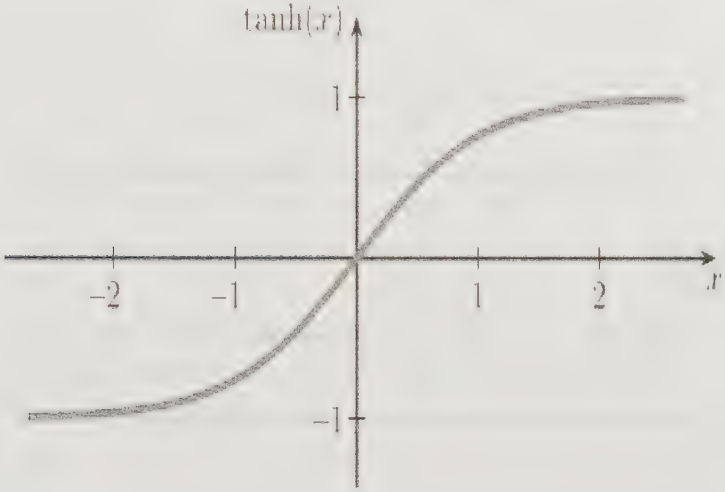
As we can see in the plot above, a sigmoid function takes as input z which is the summation or net input from the linear computation step we saw when we talked about the equation of a straight line. It then transforms the net input (z) into a point along its curve as represented by the y-axis. The range is between 0 and 1 which tells us that for any value fed into the sigmoid activation function, the output is a number between 0 and 1. We can interpret this number as a probability of how confident we are that the sample belongs to one class or the other. In the same vein, we can see 0.5 as the threshold that determines which sample a class is a member of, with values greater than or equal to 0.5 belonging to the first class and values less than 0.5 the second. This simple interpretation is why the sigmoid activation function is almost always universally used as the output activation function for binary classification problems in deep neural networks.

A closer look at the sigmoid curve also known as the S-curve shows that if we concentrate on the middle segment, then we basically have a straight line classifier, if we consider the earlier part of the curve then we can model an exponential function and finally, the later part of the curve can be used to model a decay function or saturation. The sigmoid function is one of the most important activation functions in all of deep learning

Hyperbolic Tangent Activation Function

The hyperbolic tangent activation function also known as tanh is a transfer function that is widely used in deep neural networks, particularly as the activation function in Recurrent

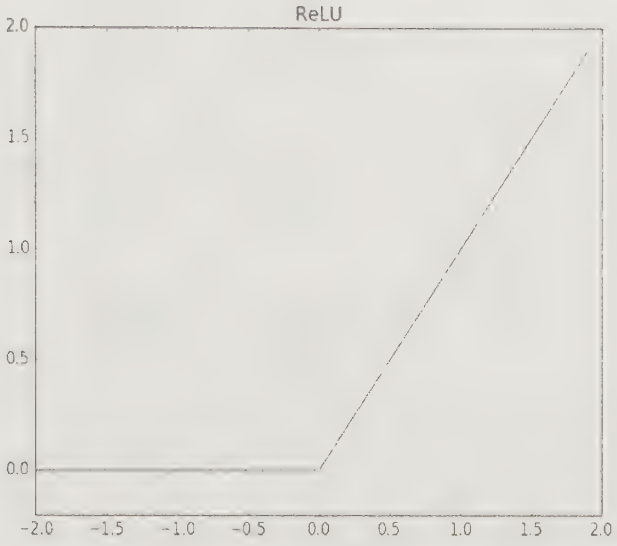
Neural Networks. It can be seen as a shifted version of sigmoid activation and it outputs a wider range of values.



Similar to sigmoid activation function seen thus far, hyperbolic tangent activation function takes as input the linear computation results and transforms it into a value within the range -1 and 1. For a binary classification problem, the threshold can be set at 0, with output values greater than 0 belonging to the positive class and outputs less than 0 in the negative class. The output from the hyperbolic tangent function is spread across a wider numerical range and it also has the benefit of being fully differentiable. In practice, tanh activation function typically converges faster than the logistic function and it is less susceptible to overfitting in later layers of the neural network.

Rectified Linear Unit Activation (ReLU) Function

In deep neural networks, Rectified Linear Units (ReLU) have become the defacto standard for activation functions in hidden layers. This is because they have been found to perform well in practice when compared to other activation functions such as sigmoid, tanh etc. One of the main benefits of using a Rectified Linear Unit as the activation function in hidden layers is that there is less of a tendency for gradients to vanish during training. The phenomenon of vanishing gradients is when gradients that are being propagated downstream become so small that earlier layers in a neural network do not learn what later layers already know. This can be seen as forgetfulness and is a huge problem when training networks that are several layers deep. Rectified Linear Units due to their nature of zeroing out negative values combat this problem.



The input to a Rectified Linear Unit (ReLU) is the result of the linear computation step. If the net input (z) is negative, ReLU makes the output for that point zero but if it is positive, it retains the value as the output of the activation function. The Rectified Linear Unit activation function is therefore the maximum between zero and the net input. It is given as:

$$f(z) = \max(0, z)$$

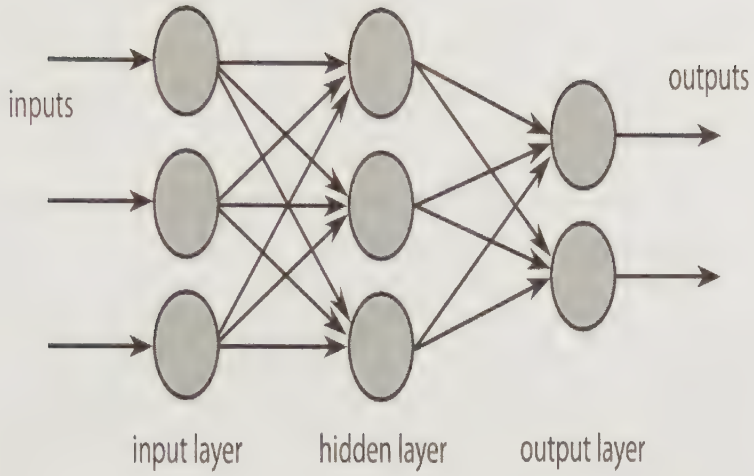
ReLU does not suffer from saturation of values as in the case of sigmoid, the values output by the Rectified Linear Unit activation function can be as large as possible provided they are positive.

There are other modifications of ReLU used in practice such as Leaky ReLU which allows a small range of negative values slip through example max(0.1z, z), Randomized Rectified Linear Unit (RReLUs), Parametric Rectified Linear Unit (PReLUs) etc.

A Feed Forward Neural Network

Now that we have a good understanding of artificial neurons, perceptron learning algorithm and activation functions, it is now time to assemble what we have learnt into a new model structure called a feed forward neural network.

A feed forward neural network is an artificial neural network that is made up several layers containing neurons in a lattice structure that is unidirectional. What that means is that the arrangement of the network is set up in such a way that it does not form a cycle hence the name feed forward. A feed forward neural network is sometimes referred to as a Multi-Layer Perceptron (MLP). The main difference between a Multi-Layer Perceptron (MLP) and the perceptron learning algorithm is that the former is made up of multiple layers, multiple neurons and uses the backpropagation algorithm via gradient descent to train the network. A feed forward neural network is made up of 3 types of layers, the input layer, the hidden layer(s) and the output layer.



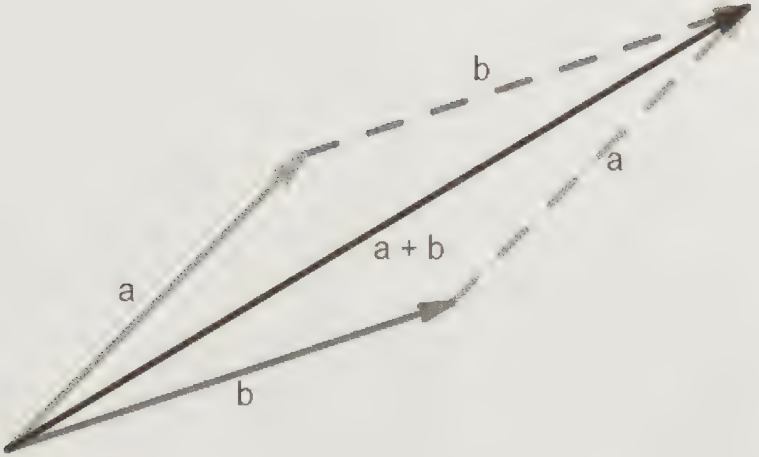
The above network is called a 2-layer feed forward neural network. The input layer is traditionally not counted when describing the number of layers in an artificial neural network. The input layer contains the data points which are fed into the network in the form of informative features. The arrows represents the weights of the network. It would be observed that neurons in one layer are connected to every neuron in the next layer. These kind of layers are referred to as fully connected layers or dense layers because of the nature of connections. The hidden layers of a feed forward neural network is where the real learning takes place as input features are transformed from one representation to another in order that their underlying pattern may be learnt. Hidden layers are so called because we cannot directly observe what goes on inside of them. The output layer computes the final prediction of the network using features that were learnt by earlier layers. The type of activation function used in the output layer is

dependent on the kind of learning problem. Sigmoid and tanh activation are popular output activation functions for binary classification problems while softmax activation function is mainly used for multi-class classification.

The Concept of Geometric Transformations and Mappings

A good way to think about what deep neural networks do is to see output predictions as the end product of a series of geometric transformations of input features. This mental representation would help give us a deeper understanding of how learning occurs in these networks. Given a set of data points represented as input features, what the network does is to use neurons in successive layers to carry out non-linear transformations of the data that unfolds it from complex manifolds into more meaningful representations that can then be separated efficiently by a hyperplane.

Since inputs to deep neural networks are tensors which are generalization of matrices, all mathematical operations carried out on data points can be thought of as geometric transformations involving addition, multiplication, nonlinear calculations etc. A simple example would be when the bias term is added to the product of the weights and the features. That addition can be represented as a geometric transformation between both vectors.



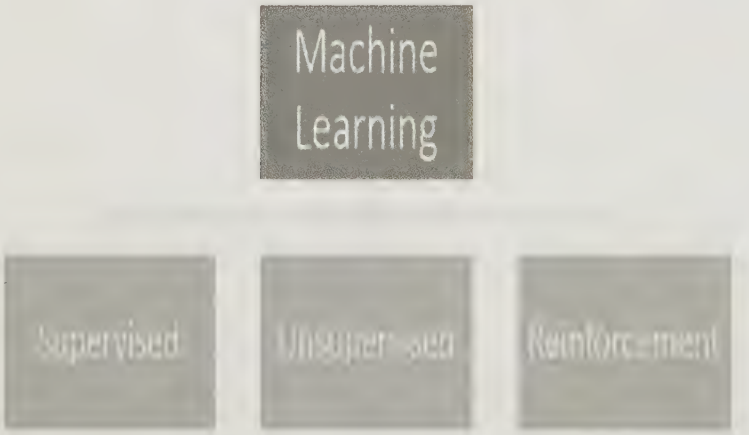
In the diagram above, the sum of vectors a and b is represented by the diagonal of the parallelogram. The combined vector a + b is a transformation of the two vectors a and b using the parallelogram law of vectors. This is what happens for every computation step in a deep neural network. The plane and hence the associated tensors representing different parameters such as the features, weights, bias etc are changed slightly, such that the neural network learns something new about the data distribution in order to enable it predict desired outputs.

A deep neural network is therefore a huge mapping function that goes from inputs to outputs, with the trainable parameters being the guide that enables it perform the transformation. In a supervised learning task, given a set of inputs and the corresponding targets, a neural network is capable of defining a complex geometric transformation that enables it predict labels for data points which it was not trained on.

Machine Learning Fundamentals

Understanding Machine Learning Models

Machine learning as we have earlier mentioned involves getting programs to execute tasks without being explicitly programmed but by having them look at lots of examples. To understand how machine learning models work, it is paramount that we have a broad intuition of the field as a whole and the kind of learning problems that it can solve.



There are three main branches of machine learning namely supervised learning, unsupervised learning and reinforcement learning. Each of these branches is named after the type of learning problem they try to solve. The most popular branch and one which currently has the most number of commercial applications is supervised learning. Supervised learning as the name implies involves training a machine learning model on

examples that are clearly annotated to show what the model is supposed the learn. These annotations or labels act as a supervisor providing guidance to the machine learning model as it steadily improves its performance at the task which it is to learn. The examples are usually labelled by humans with a domain expertise in the task posed as a learning problem. Common examples of supervised learning tasks includes classification and regression. In classification, a trained machine learning model is expected to take as input an example and predict the category the example belongs to. During training, the machine learning classifier is shown many examples and corresponding targets (labels). It then slowly learns to predict the correct labels. At first its prediction is poor and random but with each training pass, the predicted class is compared to the ground truth and the error signal is sent to the model. The model uses this error signal as a supervisor to learn features that enable it to correctly predict the class on the next iteration. Classification is used in many day to day products such as email classification, where an email is detected to be spam or not spam and is correctly sent to the relevant folder. Fraud detection is also an example of classification which is commercially viable for financial institutions as they can automatically segregate credit card transactions into fraudulent or normal transactions. Regression on the other hand is a popular instance of supervised learning where the task is to predict a single real valued number. It may be used to determine a number that tells a story about data given its features. As a fun example, let's imagine we had a house we want to put up for sale, it would be in our interest to set an asking price that is not too low as we desire to maximize profit. However, we do not want to set the price too high such that it scares potential suitors. Wouldn't it be nice to be able to

predict the optimum price the house would go for given information about the house such as number of rooms, location, crime index of neighbourhood, quality of schools in neighbourhood etc. This is a classic example of a regression problem. If we could amass a dataset containing houses and their features alongside the worth of those houses (prices), we could build a regression machine learning model that learns to map the features of a house to its price. Supervised learning is a powerful tool in the machine learning engineer's toolbox which can create real value given well labelled examples.

Unsupervised learning is the branch of machine learning that deals with raw data input. The data is provided as is and the model is expected to discover patterns and correlations that describe the data. The main difference between unsupervised learning and supervised learning is that in unsupervised learning, there is no supervisor in the training loop in form of labels. The algorithm is its own teacher, relying on the underlying distribution to make sense of data. Unsupervised learning is often a first step for supervised learning as it enables us develop a deeper understanding of patterns in our data. Two popular use cases of unsupervised learning are clustering and dimensionality reduction. In clustering, unsupervised machine learning algorithms try to group data points into similar groups. Data points in a group are similar to members of that group but dissimilar to other groups. Groups are also referred to as clusters. Clustering is particularly useful to the ecommerce industry as they may want to understand the kind of customers they have based on their purchases. Such grouping of customers according to similarity could lead to better user

experiences as certain clusters of customers are targeted with services that are most beneficial to them.

Dimensionality reduction is used as a feature selection step to choose only features that offer high discriminability. In other words, those features that drive the prediction of a model. It is an important step in supervised learning as bogus uninformative features often lead to redundancy and reduction in performance of machine learning models. Through unsupervised learning, dimensionality reduction discovers those features that are most important thereby giving us an opportunity to discard irrelevant features and train our model on a smaller feature set which generally leads to better performance.

Reinforcement learning deals mainly with creating an intelligent agent that receives information about its environment and is somehow supposed to process that information in order to make a decision and carry out actions in line with the decision that maximizes the possibility of occurrence of some end goal. Simply put, it involves training an agent to take intelligent decisions that help it achieve a reward. Reinforcement learning is an active area of research and has achieved some spectacular results in the field of game play such as Google DeepMind training an algorithm that used deep reinforcement techniques to defeat a World champion in the ancient Chinese game of Go, Lee Sedol in 2016. While reinforcement learning currently does not have practical use cases outside of game play, it is hoped that the techniques developed would be used for drug discovery, robotics, self driving cars etc.

Machine Learning Workflow

To tackle any machine learning problem, it is desirable for us to develop a detailed workflow that can be applied to a variety of tasks. The first thing we need to do is clearly define our problem. This is the most important step as all other decisions flow from it. This step involves asking the right questions such as what is the nature of the problem that we want to solve, is machine learning the best way to solve the problem, can machine learning be used to solve the problem. After we have come up with answers, if we do decide to go ahead, we are making an explicit assumption that machine learning techniques can help solve this problem given appropriate data. This leads us directly to the next step. We need to assemble a dataset that contains enough information within which a solution can be found. Depending on the type of problem that was earlier defined, the dataset acquired would reflect these assumptions. If we had earlier determined that what we faced was a classification problem, then our dataset would contain examples and labels corresponding to the classes we are interesting in detecting.

The data so obtained has to be cleaned up and preprocessed to be in a form where it can be fed to a machine learning model. The next step involves choosing a metric to determine success. What would we accept as a successful model. Would we use accuracy as a measure of success or some other metric like the mean square error. All these decisions are dependent on the type of problem and what we seek to achieve.

Next we develop an evaluation protocol. This commonly involves splitting our data into a train, validation and test split. The training set would be used to train the model while the validation set is used to tune hyperparameters of the best performing models. Finally, the trained model is run on the test set which the model has never seen and the performance of the model on the test set is reported as the final performance. Using this approach would give us a good approximation on how well our model would perform in the real world.

Data Preprocessing

Even Though we touched briefly on data preprocessing in the machine learning workflow section, it is an integral part of developing a machine learning solution. It is generally estimated that a large percentage of time is spent in data preprocessing when trying to create a performant model. This is because it is unreasonable for us to believe a machine learning model is capable of discovering correlations from arbitrary data. Encoded in the data has to be assumptions designed through domain expertise that presents the data in a way that makes it easier for machine learning models to learn mappings from inputs to outputs. These may be in the form of feature engineering whereby data is transformed into simpler representations before they are fed into the model. Although deep learning has reduced the importance of domain expertise in some fields due to automatic feature learning, data must still be presented in such a way that it is easy for deep learning algorithms to learn.

Inputs to machine learning models are usually numeric, that is data features are transformed into numbers. Images for instance are transformed into a grid of numbers with each pixel a number between 0 and 255, detailing its brightness. Text characters are constructed into a dictionary with a set of numbers representative of individual words in the dictionary and output categories are one-hot encoded to convert classes to a sparse matrix containing 0s in all positions and 1 in the position of the relevant category.

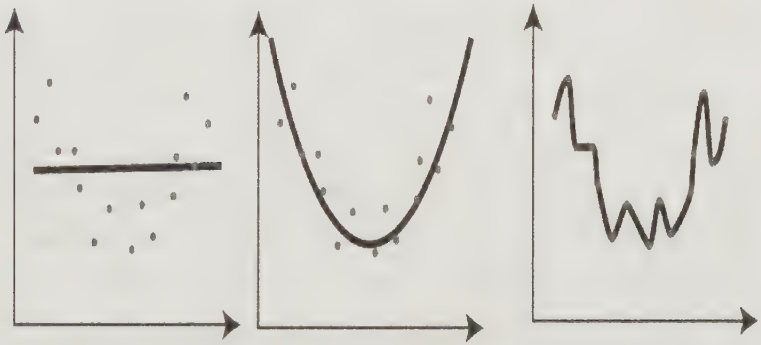
This preprocessing step is important as machine learning algorithms would complain when they are fed data that is not in a suitable form. Missing values are also handled using various techniques like mean or mode imputation and features are normalized to make sure they are all on the same scale.

Evaluation of Machine Learning Models: Overfitting, Underfitting, Bias Variance Tradeoff

Machine learning models are evaluated according to a metric chosen to determine whether a model has learn the desired representations which enable correct predictions. During training, the performance of a model improves rapidly as it is exposed to more data. It learns correlations inherent in the training data. This process of getting the model to perform better on the training set is known as optimization. However, our main aim is not to train a model that performs exceptionally only on the training set, our aim is to achieve generalization. Generalization means training a model that discovers important patterns in the training set which it can use to make predictions on new data. This ability to generalize well

to unseen data is paramount in machine learning and is usually balanced against the desire to optimize a model. An overly optimized model risks overfitting to the training set, that is merely memorizing patterns in the training set instead of learning important patterns that would enable it generalize (perform well) to new test data.

Let us use a visual representation to enable us understand these concepts better.



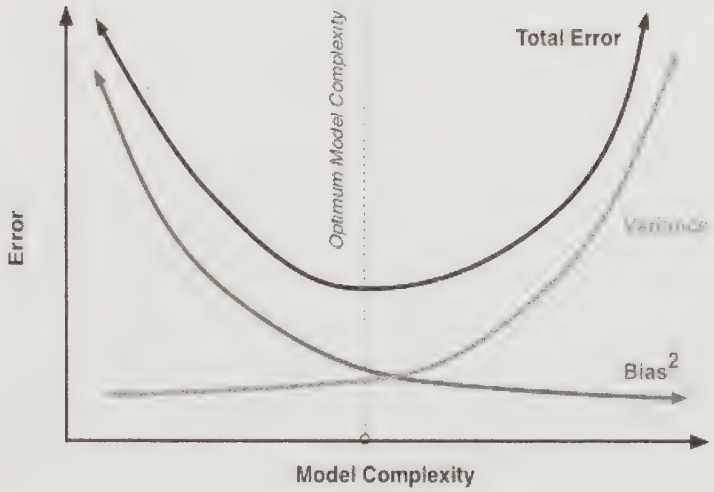
In the figure above, the first plot shows a straight line, which does a poor job of fitting to the data points. The straight line has failed to learn the inherent data distribution even though it offers a simpler model. This is a clear case of underfitting as the straight line does not use enough of the information provided to it to solve the problem. In machine learning, underfitting can usually be solved by creating a more powerful model or training a model for longer periods.

The second plot shows a curve that passes through a number of points but misses some others. The important thing to note here is that unlike the straight line, the curve captures the distribution of data points which is U-shaped. This representation can be said to have learnt the most important features from the training set and is likely to generalize well to new data points.

The last plot is a clear case of overfitting. The model goes out of its way to take into account every data point in the training set. By so doing, it uses a complex polynomial to describe a convoluted curve. The model would achieve near perfect accuracy on the training set but would have terrible accuracy on the validation or test set because it basically memorized every data point and failed to learn the underlying dynamics of the dataset. It modelled itself after every data point including noisy points and by so doing could not discover the correct data distribution.

Overfitting is the central problem in machine learning because it can make our models to hallucinate about patterns that are not present in the dataset, leading to a disaster when they are deployed to handle real world data. Overfitting can be fought by increasing the amount of training data and by regularization which basically punishes more complicated models.

Models that suffer from underfitting have high bias because they make simplistic assumptions about the data whereas models that overfit are said to have high variance as they modelled noise contained in the training set hence their performance on new data varies widely.



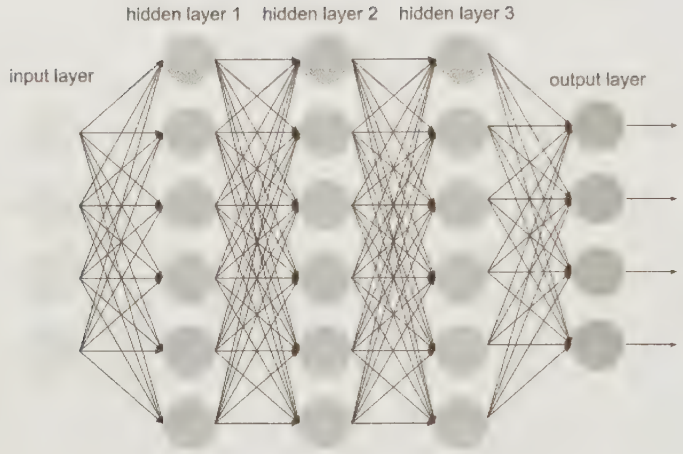
What we want when training machine learning models is to get a model that has low bias and low variance. From the image above, when we begin training, the error rate drops rapidly, then it stagnates and starts increasing. The moment the error begins to increase, the model has started overfitting to the training data. The variance also begins to increase at this point. The optimal approach is to train the model long enough for the bias to reduce (low bias) without increasing the variance and error rate. This point is attained roughly at the middle of the plot. That point indicates a model that is sufficiently complex to capture important patterns but is not too complex to overfit to noise present in the data. In machine learning, this principle is known as Occam's razor. Occam's razor basically tells us to choose the simplest model that achieves the most accuracy, that is the model that explains the data well enough while making few assumptions.

Presentation of Deep Learning Techniques

Fully Connected Neural Networks

Fully connected neural networks are a class of artificial neural networks in which nodes in one layer are connected completely to nodes in the next layer. A fully connected neural network forms a dense layer of neurons since neurons from previous layers are connected to all neurons in the next layer. A layer featuring this type of connection is sometimes referred to as an affine or dense layer. Several dense layers arranged from input to output form a fully connected neural network.

A fully connected neural network usually has a lot of trainable parameters as each connection represents a weight and a bias. It is therefore not used for learning problems where the input data is high dimensional such as images. A fully connected neural network is commonly used for classification and regression tasks with the output layer having a sigmoid or softmax activation function in the case of classification and no activation function in the case of regression.



In the fully connected neural network above, it can be seen that the network of arrows representing the weights, join every neuron to every other neuron in the next layer. It is this concentration of connections that gave rise to the term dense layer. The neurons in various layers can have different activation functions as usual, however the input layer is not adjusted during training because we do not modify our inputs. The fully connected neural network above is a 4-layer deep neural network with 3 hidden layers and depending on the type of activation functions in the output layer can be used to solve classification or regression problems.

Convolutional Neural Networks

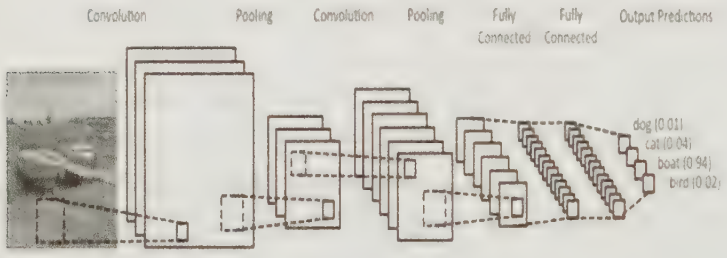
In the field of Computer Vision, Convolutional Neural Networks (CNNs) cast an imposing figure because they have been used to achieve state of the arts results in a myriad of vision tasks. The performance of Convolutional Neural Networks in benchmarks have dwarfed other models and algorithms.

Convolutional Neural Networks are a type of deep neural networks that are inspired by biological processes associated with human vision. The model builds into its architecture an implicit bias that it would be dealing with 2-dimensional data such as images. This shows in the way neurons are arranged whereby a neuron can only see part of the input which is in its receptive field, similar to how the visual cortex operates in animals. As we move deeper into the neural network, the combination of what individual neurons analyze sum to give the network a bird's eye view of the entire input. convolutional neural networks automatically learn a set of filters which is synonymous to hand crafted features in traditional approaches. The convolutional neural network is free to learn features which it feels is important for the succeeding at the current task example determining if the is a human face in a picture or not. This is one of the main advantages of a convolutional approach as relevant features are automatically discovered and there is no need for and expensive and slow manual feature engineering step.

To understand a convolutional neural network, it is important to get a grasp of its core operation, the convolution. The convolution operation involves finding local patterns in the input unlike in the case of a fully connected layer that finds global patterns. The benefit of this is that the patterns so learnt can be recognized in any other part of the image, so it leads to convolutional neural networks exhibiting translation invariant capabilities. For example, an image of a dog viewed from the side should still produce a correct classification of a dog as if

the dog was viewed frontally. The same thing is also true in cases of occlusion when an object to be detected is partially blocked from view. The architecture of a convolutional neural network makes it robust to these kind of scenarios.

Let us now develop our intuitions further by having a look at the architecture of a simple Convolutional Neural Network.



In the network above, an input image is fed to CNN. The input is interpreted as a tensor of 4 dimensions. The first axis represents the batch, in this case 1, the second axis represents the height of the image, the third axis represents the width of the image and the fourth axis represents the number of color channels. Note that this way of representing an image uses the channels last convention, that is the number of channels appear in the fourth dimension ($N \times H \times W \times C$). There is also an alternative first channel convention which places the number of channels immediately after the batch axis ($N \times C \times H \times W$). For an input image, the number of channels is 3 if it

is a color photo representing the red, green, blue (RGB) channels respectively and it is 1 for a monochrome image.

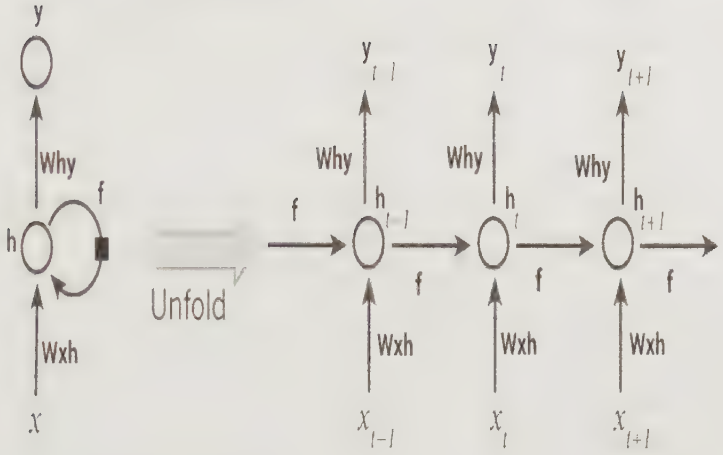
A convolution operation involving a number of filters is then carried out on the image to produce corresponding feature maps. The feature maps represents the features automatically learnt by the CNN. The next operation involves pooling which means downsizing feature maps to a lower size. Max pooling or average pooling may be employed. The former involves choosing the maximum features in a slide window while the later involves averaging features. The downsampled feature maps are then convolved further and passed deeper into the network. The main intuition to get is that earlier layers detect simple features such as edges while later layers combine these features earlier detected into complex features such as patterns, object parts etc. Also, as we move deeper in the network, the size of the image reduces but the depth (number of channels) increases.

Finally, the advanced feature maps are fed into a regular fully connected neural network made up of dense layers and activation functions. The final prediction is then made using the normal process we are familiar with in a dense neural network. Convolutional neural networks have a wide range of applications in object detection, image segmentation, image classification etc and are the go to solution for image related tasks.

Recurrent Neural Networks

Recurrent neural networks are a class of deep neural networks that are well suited for handling text data or sequential data. They are particularly powerful when modelling data that has a time axis such as in weather forecasting or audio processing. The important of the sequential nature of such data cannot be overemphasized as a change in the sequence of inputs can completely skew the meaning. Natural language is a prominent example, where the placement of words in a sentence can give it a different meaning. Recurrent neural networks receive different inputs at various timesteps and have the ability to model dependencies of sequences earlier encountered.

A recurrent neural network can therefore be seen as a neural network with an internal loop. It processes inputs according to the time axis and maintains a hidden state for each timestep which is passed to subsequent timesteps in the sequence. Let us look at an example recurrent neural network below.



The first part of the image shows a compact representation of a recurrent neural network. The input is represented as x, h represented the internal state, y represents the output while f represents the recurrent loop which contains the time transformations. It is often beneficial to unroll the compact representation into one that shows time varied inputs. At each timestep, an input is fed into the RNN, a state is computed and an output is given. At the next timestep, a new input is sent into the network and the previous computed state is taken into cognizance when the current state is computed. An output for that state is given and the new state flows forward into the next timestep. These continue until all timesteps contained in the sample have been computed. In the case, the number of input timesteps is equal to the number of output timesteps. This is not always true as there are applications that involve variations such as when translating from one language to another, the

number of words required for a correct translation may not be the same number of words in the original translation.

Recurrent neural networks are heavily used in the fields of Natural Language Processing (NLP) and Natural Language Understanding (NLU) because an RNN has the ability to remember sequence of words which it has come across in the past. Other areas where RNNs produce impressive results are speech recognition, sentiment analysis, time series forecasting etc. A common thread would be noticed across these applications, the nature of the input is such that remembering what happened in the recent past improves the quality of predictions about the future.

However, training recurrent neural networks are difficult because they suffer from vanishing and exploding gradients. Vanishing gradient problem is encountered when gradients being backpropagated through time from later layers to earlier layers become exponentially close to zero. This makes learning impossible as weights cannot be properly updated to learn relevant features. Luckily, a variant of recurrent neural networks known as Long Short Term Memory (LSTM) solves this problem by giving RNNs a memory component that decides what memories of seen sequences it should discard or keep. Another popular modification of Long Short Term Memory (LSTM) which is not as powerful but simpler to use is the Gated Recurrent Unit (GRU). The problem of exploding gradients whereby gradients become too large is relatively easier to solve as gradients can be clipped by a predetermined threshold. The main idea to take from this explanation is that

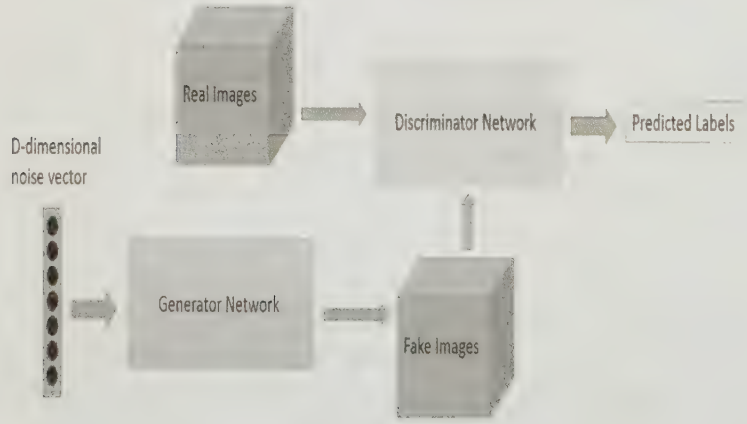
the same set of weights are used across the timesteps in a recurrent neural network and as such it is important for the gradients to be neither too small or too large, so that training can occur.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) belong to a family of generative models used in unsupervised learning. The first thing to note about generative adversarial networks is that there is no external source of labels used in training. Training proceeds in an unsupervised manner. To understand GANs it is vital to understand the broader family of models which it belongs to - generative models. Generative models are models which learn to predict features given labels, that is they learn to model the underlying data distribution from labels. The opposite of generative models are discriminative algorithms which we are familiar with like in the case of classification. Discriminative models aim to predict a class label given its input features. We can think about this in terms of probability, discriminative algorithms try to predict the probability of y given x, $p(y|x)$, that is the output label given the data whereas generative algorithms model the probability of x given y, $p(x|y)$. In generative models we ask the question, given this particular label y, what is the probability that it came from x distribution of features.

Generative Adversarial Networks try to answer the question of $p(x|y)$ by setting up two competing neural networks in a minimax game with opposing objectives. The first network known as the generator network or to use gaming analogy -

player 1, generates fake samples (images) from a prior distribution. It then feds these fake samples into the second neural network known as the discriminator or player 2. The discriminator is usually trained on images from the training set which are all real, so it knows how to discriminate between real images, contained in the training set and fake images produced by the generator. As the game proceeds, the generator becomes better at producing fake images which look real in order to fool the discriminator and the discriminator gets adept at spotting fake images from real ones. The purpose of this setup is to attain balance such that the generator eventually produce realistic images that are almost indistinguishable from those in the training set while the discriminator becomes better at recognizing fakes. It should be noted that the generator never sees examples of images in the training set but only uses feedback from the discriminator to improve its artistic prowess. Let us look at a simple schematic diagram of a GAN below.



As can be seen the generator starts sampling from a random noise image conditioned on a simple prior distribution example a Gaussian while the discriminator is initiated trained on real images. Then the training of the discriminator is freezed (paused) and the the fake images are mixed with the real images. The generator uses feedback from the discriminator - whether it identified an image as real or fake to update its weights to improve its generation techniques.

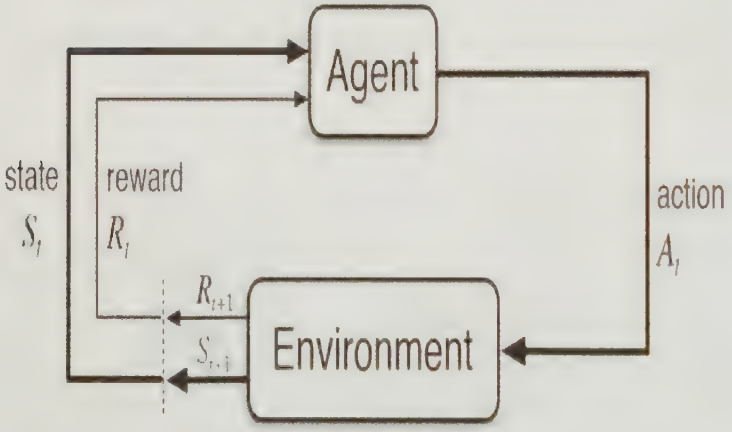
The generator network in GANs is usually a deconvolutional neural network. What that means is that the generator performs upsampling of random noise into an image. The discriminator on the other hand is a convolutional neural network that reduces an image into a prediction label, real or face.

Generative adversarial networks despite being tricky to train are considered one of the trendiest areas of deep learning and active research continues into this exciting technique. GANS are currently the state of the art in image generation as images produced are highly realistic, continuous and they have also been applied to other domains such as text generation.

Deep Reinforcement Learning

Deep reinforcement learning is an area of machine learning that combines approaches from reinforcement learning and deep learning to achieve the best of both worlds. Reinforcement learning is concerned with creating an

intelligent agent that steadily accumulates its rewards in an environment through a trial-and-error mechanism by performing those actions that maximizes its total rewards. The training signal in reinforcement learning comes takes the form of rewards or punishments issued by the environment in response to actions taken. Deep reinforcement learning avoids hand engineered features but rather encourages agents to learn from raw inputs in their environment. A deep neural network is used to represent the Q-network which predicts the total rewards an agent can expect to it by taking certain actions. The key insight is to allow the agent to learn such that it prioritizes taking good actions over bad ones.



The diagram above provides a schematic description of the key tenets of reinforcement learning. An agent in an environment performs certain actions which alter the state of the environment and the environment rewards or punishes the

agents in line with the actions taken. The aim of the agent is to maximize the amount of total reward accumulated at the end of the game, so it has to balance actions that promise short term rewards with those that offer long term rewards. This is usually posed as the exploration-exploitation dilemma, where the desire to exploit currently profitable actions has to be balanced against the need to explore for even better actions.

Deep reinforcement learning has garnered a number of impressive feats in game play such as attaining human level performance in Atari games, 3-D navigation and puzzles, Go etc. Reinforcement learning techniques would offer a high degree of practicality across domains in future as the questions sought to be answered by this field are some of the most fundamental questions about intelligence.

Introduction to Deep Neural Networks with Keras

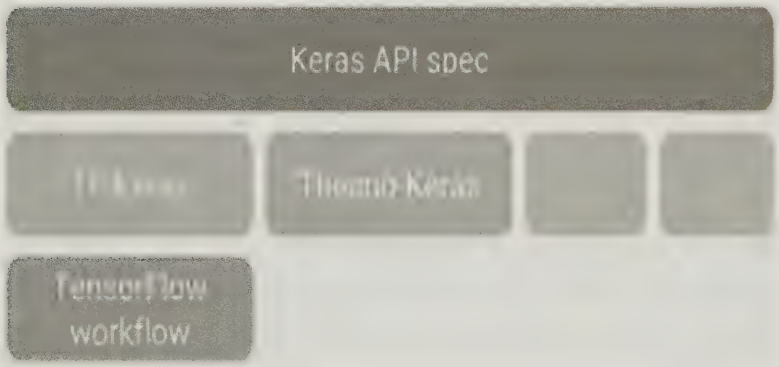
The Keras Philosophy

In the world of deep learning, there are many deep learning frameworks and libraries targeted at various classes of users. Some are based on different programming languages like Java, Lua, Python etc. Others provide low-level tools that enable optimizations on Central Processing Units (CPUs) and Graphics Processing Units (GPUs). In the deep learning landscape, Python has grown to become the preferred programming language of choice, not just because it is a very expressive language but because of the entire ecosystem which is mature and made of of powerful libraries that enable scientific computation.

The most popular deep learning framework that offers a Python Application Programming Interface (API) is TensorFlow. TensorFlow was open sourced by Google in 2015 and it contains a bunch of awesome tools that makes machine learning and deep learning projects easier to execute. It is also worth mentioning that there are other frameworks such as Theano, Pytorch, MXNet, Chainer etc which are widely used. In fact we would build an artificial neural network using one of those deep learning frameworks in the next chapter - Pytorch.

We took the detour of explaining the tools available today to a deep learning researcher because we want to understand where Keras fits in the stack. Keras is a high level deep learning

framework that abstracts away many of the low level tasks integral to building a deep neural network. Keras allows for building prototypes quickly and for experimenting research ideas as it exposes a friendly API which is built on what is called a Keras backend engine. Keras does not by itself perform any numerical computation but passes it off to lower level APIs which are available in its backend. Keras can be currently be run using three frameworks as it backend engine namely, TensorFlow, Theano and Microsoft's Cognitive Toolkit.



The advantage of using Keras is that it provides an easy API for building convolutional and recurrent neural networks. It also provides these building blocks in such a form that different kinds of models can be constructed such as multi-input models, multi-output models etc. It also allows for advanced features such as weight sharing and model sharing. One important design philosophy of Keras is that the same code can be run on CPUs and GPUs without any modification whatsoever. It is because of these perks that Keras is popular

among deep learning practitioners as it focuses on modularity and scalability.

In this book, we would use Keras with TensorFlow as the backend. Keras is compatible with TensorFlow and models developed in Keras can easily be exported to TensorFlow.

Install Keras

To install Keras, we first need to have a compatible backend such as TensorFlow installed. So let's quickly go through the installation of TensorFlow. We would go through installing TensorFlow on Windows, Linux and macOS and we would use the default Python package manager pip for all installations.

Install TensorFlow on Windows

For TensorFlow to be installed on Windows, you need to first install Python which comes bundled with pip as the package manager. Python can be downloaded as an executable at https://www.python.org/downloads/windows

Once you have it installed, start a terminal and run the following command:

```
C:\> pip3 install --upgrade tensorflow
```

Install TensorFlow on Linux

For Linux distributions like Ubuntu and its variants, pip is usually already installed. To check which version of pip is installed, from the terminal run

```
$ pip -V
```

or

```
$ pip3 -V
```

This depends on the version of Python you have, pip for version 2.7 and pip3 for version 3.x

If you do not have pip installed, run the appropriate command for your Python version below:

```
$ sudo apt-get install python-pip python-dev   # for Python 2.7
$ sudo apt-get install python3-pip python3-dev # for Python 3.n
```

It is recommended that your version of pip or pip3 is 8.1 or greater. Now you can install TensorFlow with the following command

```
$ pip install tensorflow      # Python 2.7
$ pip3 install tensorflow     # Python 3.x
```

Install TensorFlow on macOS

To install TensorFlow on macOS, you need to have Python installed as it is a prerequisite. To check if you have pip installed run

$ **pip -V**

or

$ **pip3 -V**

This depends on the version of Python you have, pip for version 2.7 and pip3 for version 3.x

If you do not have pip installed, or you have a version lower than 8.1, run the commands to install or upgrade:

$ **sudo easy_install --upgrade pip**
$ **sudo easy_install --upgrade six**

Now you can install TensorFlow with the following command

$ **pip install tensorflow** # Python 2.7
$ **pip3 install tensorflow** # Python 3.x

We can now validate we have TensorFlow correctly installed by running the following command from the terminal.

$ **python -c 'import tensorflow as tf; print(tf.__version__)'** # for Python 2.7

$ **python3 -c 'import tensorflow as tf; print(tf.__version__)'** # for Python 3.x

If the above prints a version number you have successfully installed TensorFlow and we can go ahead to install Keras.

The command to install Keras is common to all operating systems we have covered so far. In your open terminal, run the following command.

$ pip install keras

To check if you have successfully installed keras, run the following command.

$ python -c 'import keras; print(keras.__version__)' # for Python 2.7

$ python3 -c 'import keras; print(keras.__version__)' # for Python 3.x

You should have the version number of Keras printed to your screen. Now that we have set up Keras, let's now build a neural network

A First Look at Neural Networks in Keras

For our hands on example, we would tackle a multi-class classification problem. What that means is that the classification task would involve multiple classes and each sample can only belong to one class. We would use a document classification dataset from the Reuters news agency. The dataset contains short newswires that are annotated by their topics. The topics serve as the label of documents in the collection. We are therefore faced with a natural language understanding task as each newswire represents a story.

The Reuters dataset is bundled with other datasets in Keras. This enables easy experimentation as we can access various datasets from the **datasets** module.

Let us import the dataset, Keras and Numpy which is an efficient numerical computation library that would help us perform certain tasks.

Next we load the dataset into memory separating it into data and labels for the train and test sets respectively.

```
rain data, train labels), (test_data, test labels)
                          ords=10000)
```

Note that the first time this is run, it would download the dataset to your machine. This should take a few seconds. We then run the following print statements to get an idea of the size of our dataset.

```
print(len(train data), 'train sequences')
print(len(test_data), 'test sequences')
```

```
8982 train sequences
2246 test sequences
```

We have 8,982 samples in the training set and 2,246 samples in the test set. Let us know look at a random data point from the training set.

```
print(train_data[10])
```

```
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979, 3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

The sample is a list of integer numbers which represents word indices. We can decode the word indices above using the following suite of code.

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# note that the indices are offset
and 2 are reserved by datas
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
print(decoded_newswire)
```

```
? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to three dlrs reuter 3
```

Let us check the number of classes we are to predict. They are currently stored as integer values starting from index 0.

There are 46 distinct categories for newswires.

The next step is data preparation as it is important that we convert the data into a desirable form. We first convert our features which are currently stored as word indices into one-hot representations using the tokenizer class from the preprocessing module.

```
from keras.preprocessing.text import Tokenizer

train_data =
tokenizer.sequences_to_matrix(train_data,
mode='binary')

okenizer.sequences_to_matrix(test_data,
```

We also use convert our labels to one hot representations. These are sometimes called categorical encodings and we use `to_categorical` utility from Keras.

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels,

one_hot_test_labels = to_categorical(test_labels,
num_classes)
```

We start describing the model as a sequence of layers using the sequential class in Keras.

```
model.add(layer
el.add(layers.Dense(46, activation='softmax'))
```

The model is a 3-layer feedforward neural network with 64 units in the hidden layers and 46 units in the output layer which corresponds to the number of classes. Next we compile the described model by providing a learning strategy and a success metric.

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy'
              metrics=['accuracy'])
```

We use categorical_crossentropy as the loss function because this is a multi-class classification problem.

Next, we train the model on the training set, specifying that 10% of training data should be used for validation.

```
history = model.fit(train_data, train_labels,
                    batch_size=512,
                    epochs=20,
                    verbose=1,
```

We achieve a validation accuracy of 77%. Let us now plot the training and validation losses using the code below.

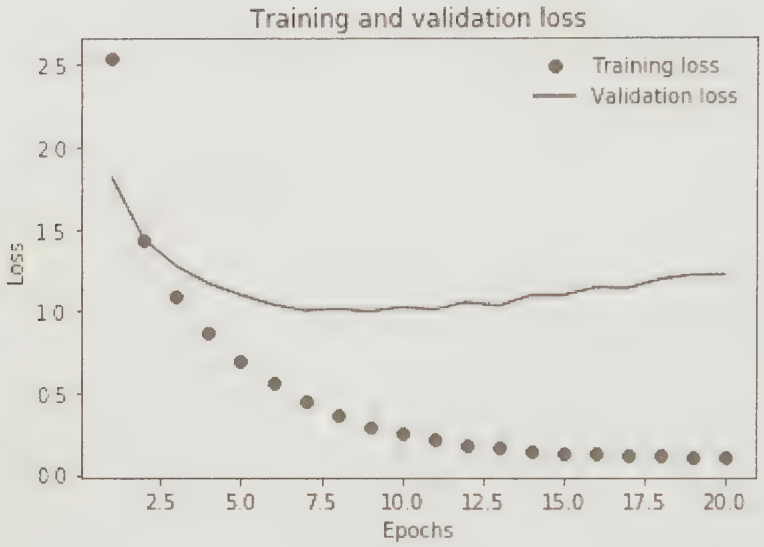
```
# uncomment the next line if running in Jupyter
notebook to allow matplotlib inline plots
# %matplotlib inline
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation
loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.legend()

plt.show()
```



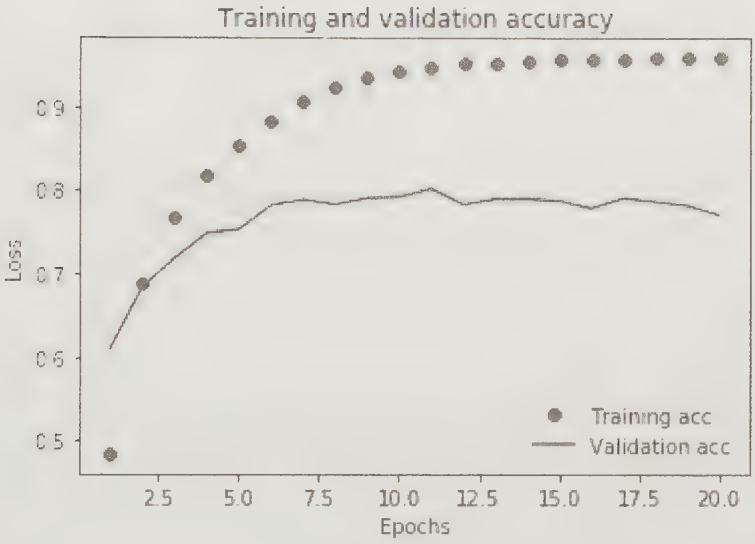
We can observe that the loss initially drops rapidly both in the train and validation split, before it starts rising in the validation set due to overfitting.

Next we plot the accuracy of the train and validation sets.

```
plt.clf()   # clear figure

acc = history.history['acc']
val_acc = history.history['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation
acc')
plt.title('Training and validation accuracy')
```



From the plot above, the training accuracy continues increasing before plateauing, overfitting to the training set in the process whereas the validation accuracy stagnates faster. This indicates that we should train the model for less epochs to avoid overfitting or try employing some regularization techniques.

Finally, we can evaluate our model on the test set by calling **`evaluate`** method on the trained model instance.

```
Test accuracy: 0.78
```

We achieve an accuracy of 0.78 on the test set. This means that for 78% of samples contained in the test set, our trained model was able to correctly predict the topic of those newswires.

Conclusion

There are a lot more real world applications of deep learning in consumer products today than at any point in history. It is generally said that if you can get large amounts of data and enormous computation power to process that data, then deep learning models could help you provide business value especially in tasks where humans are experts and the training data is properly annotated.

Thank you !

Thank you for buying this book! It is intended to help you understanding deep learning fundamentals. If you enjoyed this book and felt that it added value to your life, we ask that you please take the time to review it.

Your honest feedback would be greatly appreciated. It really does make a difference.

If you noticed any problem, please let us know by sending us an email at review@aisciences.net before writing any review online. It will be very helpful for us to improve the quality of our books.

AI SCIENCES

We are a very small publishing company and our survival depends on your reviews.
Please, take a minute to write us an honest review.

If you want to help us produce more material like this, then please leave an honest review on amazon. It really does make a difference.

https://www.amazon.com/dp/B07FTPKJMM

Sources & References

Software, libraries, & programming language

- Python (https://www.python.org/)
- Anaconda (https://anaconda.org/)
- Virtualenv (https://virtualenv.pypa.io/en/stable/)
- Numpy (http://www.numpy.org/)
- Pandas (https://pandas.pydata.org/)
- Matplotlib (https://matplotlib.org/)
- Keras (https://keras.io/)
- Pytorch (https://pytorch.org/)
- Open Neural Network Exchange (https://onnx.ai/)
- TensorFlow (https://www.tensorflow.org/)

Datasets

- Kaggle (https://www.kaggle.com/datasets)
- Keras Datasets (https://keras.io/datasets/)
- Pytorch Vision Datasets (https://pytorch.org/docs/stable/torchvision/datasets.html)
- MNIST Database Wikipedia (https://en.wikipedia.org/wiki/MNIST_database)
- MNIST (http://yann.lecun.com/exdb/mnist/)
- CIFAR-10 (https://www.cs.toronto.edu/~kriz/cifar.html)

- Reuters dataset (https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection)
- IMDB Sentiment Analysis (http://ai.stanford.edu/~amaas/data/sentiment/)

Online books, tutorials, & other references

- Coursera Deep Learning Specialization (https://www.coursera.org/specializations/deep-learning)
- fast.ai - Deep Learning for Coders (http://course.fast.ai/)
- Keras Examples (https://github.com/keras-team/keras/tree/master/examples)
- Pytorch Examples (https://github.com/pytorch/examples)
- Pytorch MNIST example (https://gist.github.com/xmtbir/b27cdbff6887011 8bdb8cefa86a2d558)
- Overfitting (https://en.wikipedia.org/wiki/Overfitting)
- A Neural Network Program (https://playground.tensorflow.org/)
- TensorFlow Examples (https://github.com/aymericdamien/TensorFlow-Examples)
- Machine Learning Crash Course by Google (https://playground.tensorflow.org/)

Thank you !

Thank you for buying this book! It is intended to help you understanding deep learning fundamentals. If you enjoyed this book and felt that it added value to your life, we ask that you please take the time to review it.

Your honest feedback would be greatly appreciated. It really does make a difference.

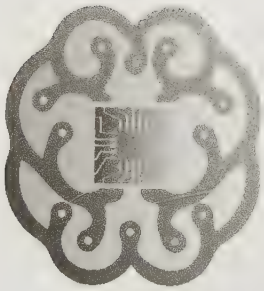
If you noticed any problem, please let us know by sending us an email at review@aisciences.net before writing any review online. It will be very helpful for us to improve the quality of our books.

AI SCIENCES

We are a very small publishing company and our survival depends on your reviews.
Please, take a minute to write us an honest review.

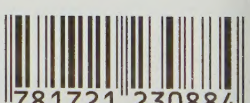
If you want to help us produce more material like this, then please leave an honest review on amazon. It really does make a difference.

https://www.amazon.com/dp/B07FTPKJMM



AI SCIENCES

CPSIA information can be obtained
at www.ICGtesting.com
Printed in the USA
BVHW081510270519
549348BV00021B/2382/P



lease address comments and questions concerning this book to our customer service by email at:
contact@aiscience.net

Our goal is to provide high-quality books for your technical learning in computer science subjects.

Thank you so much for buying this book.

ISBN 9781721230884

9 781721 230884