MACHINE LEARNING MASTERY

Calculus for Machine Learning

Understanding the Language of Mathematics

Authors

Stefania Cristina

Mehreen Saeed

Founder

Jason Brownlee

Disclaimer

The information contained within this eBook is strictly for educational purposes. If you wish to apply ideas contained in this eBook, you are taking full responsibility for your actions.

The author has made every effort to ensure the accuracy of the information within this book was correct at time of publication. The author does not assume and hereby disclaims any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from accident, negligence, or any other cause.

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic or mechanical, recording or by any information storage and retrieval system, without written permission from the author.

Credits

Founder: Jason Brownlee
Authors: Stefania Cristina and Mehreen Saeed
Lead Editor: Adrian Tam
Technical reviewers: Andrei Cheremskoy, Darci Heikkinen, and Arun Koshy

Copyright

Calculus for Machine Learning

Edition: v1.05

Contents

Preface

When we try to understand machine learning algorithms, it is quite difficult to avoid calculus. This book is to help you refresh the calculus you learned or give you a quick start on just enough calculus to move forward.

When calculus is brought up, the first thing that comes to mind for many is difficult math. However, evaluating a calculus problem is just following some rules to manipulate it. However, the most important thing in studying calculus is to remember the physical nature it represents. At times, calculus can be abstract but is often not hard to visualize.

So why must we learn about calculus when studying machine learning? In many machine learning algorithms, we have a goal of what the machine should do, and we expect it will behave in a certain way. Calculus is the tool for us to model the algorithm's behavior. It allows us to see how the behavior of the algorithm will change if a parameter is changed. It also gives us insight into which direction we should fine-tune the algorithm or whether it is achieving the best it can do, even if it doesn't perfectly fit the training data.

As a practitioner, you need to know calculus is a tool for modeling. After reading this book, you should know why we cannot use an accuracy measure as a loss function in training a neural network because accuracy is not a differentiable function. You will also be able to explain why a larger-sized neural network will be trained disproportionately slower by counting the number of differentiations we need to compute in backpropagation. Furthermore, if you understand calculus, you can convert an idea of the machine learning algorithm into code.

This is a book on the theoretical side of machine learning, but it does not aim to be comprehensive. The objective of this book is to provide you with the background to understand the API documentation or other people's work on machine learning. This book provides you with an overview so you can go deeper with more advanced calculus books if you would like to.

The earlier chapters of this book focus on the foundation. They introduce the notation and terminologies, as well as the concepts of calculus, but are deliberately kept to the minimum. The later chapters of this book introduce some examples where calculus is applied. The examples are in Python, so you may try to run them on your computer. We will see how we can build a neural network and support vector machine from scratch in the last few chapters, in which some calculus evaluation has to be done in order to program it correctly. We hope this will give you some insight and pave the way to better understand machine learning literature.

Introduction

Welcome to *Calculus for Machine Learning*.

The modern study of calculus traces its roots back to the time of Issac Newton and Gottfried Leibniz. Physics has found calculus a very useful tool immediately after it was invented. Economics also sees calculus as a handy tool to model behaviors. Similarly, in machine learning, we can use calculus to find how a model's output will change if we skew some parameters a bit. We can also use calculus to find the optimal model parameters. This is especially pronounced in the case of traditional machine learning models. Linear regression, for example, can be considered as a task to find the coefficients to an equation that minimized the sum of squared error.

This book will provide you with the basic knowledge of calculus with the connection to several use cases in machine learning. Some of the examples are provided as executable programs in Python. We cover just enough to let you feel empowered to work on your machine learning projects.

Who is this book for?

Calculus is not the indispensable knowledge in applied machine learning. A lot of people can do well without it. But from time to time, you may see literature mentioning some calculus ideas or equations to explain a machine learning algorithm. You need to fully understand them to implement the algorithm. This book is for those developers or practitioners who want to go one step further to get a deeper understanding on the machine learning research. Before you begin, this book assumes:

- You know your way around basic Python for programming. (It's best if you also know NumPy.)
- You know the basic machine learning techniques, for example, how to training a model, what is a regression, and so on.
- You learned a bit of linear algebra and trigonometry, hence you feel comfortable reading math symbols, especially matrix equations.

This book begins with the fundamental concepts and leads you to the application of calculus. Most of this guide was written in the top-down and results-first machine learning style that you're used to from MachineLearningMastery.com.

What to expect?

This book will teach you the basics concepts of calculus. You will learn not only the univariate calculus that you will see in elementary textbooks, but also the multivariate one that we will often encounter in machine learning literature. Our focus is inclined toward differential calculus as we will find it more useful in machine learning. After reading and working through the book, you will know:

- Calculus arose from studying how to add up a lot of infinitesimal amounts
- What is limit and that differentiation is a result of taking limits
- Integration is the reverse of differentiation
- The physical meaning of differentiation is the rate of change, or the slope if put into a geometric perspective
- The many rules to evaluate the differentiation of a function
- What is the differentiation of multivariate functions and vector-valued functions, and how to evaluate them
- Differentiation is a tool for optimization, and the method of Lagrange multipliers can let us perform function optimization with constraints
- Differentiation is a tool to find an approximation of a function
- How we apply calculus in coding a neural network from scratch
- How we apply calculus to implement a support vector machine

We do not want this book to be a substitute for a formal calculus course. In fact, the textbooks on calculus should give you more detail, more exercises, and more examples. They are beneficial if you need a deeper understanding. However, this book can complement the textbooks to help you make connections to machine learning applications.

How to read this book?

This book was written to be read linearly, from start to finish. However, if you are already familiar with a topic, you should be able to skip a chapter without losing track. If you want to learn a particular topic, you can also flip straight to a particular section. The content of this book is created in a guidebook format. There are a substantial amount of example codes in this book. Therefore, you are expected to have this book opened on your workstation with an editor side-by-side so you can try out the examples while you read them. You can get the most from the content by extending and modifying the examples.

This book will not cover all topics in calculus. In fact, no book can do that. Instead, you will be provided with intuitions for the bits and pieces you need to know and how to get things done with calculus. This book is divided into seven parts:

- ▷ **Part I: Foundations**. A gentle introduction to what calculus is about and why it is useful.
- ▷ **Part II: Limits and Differential Calculus**. Define what a limit of a function is, and from there, we see how infinitesimal quantities are measured. Then, we learn how to find the derivative of a function or the differentiation. While the differentiation is defined from the limits, we will discover the rules that allow us to find the differentiation of many functions easier.
- ▷ **Part III: Multivariate Calculus**. Extended from the simple differential calculus, we will see the differentiation of more complex functions, namely, those with multiple variables or those with vectors as their values. This is where we learn the terms that we often encounter in machine learning literature, such as partial derivatives, gradient vectors, Jacobian, Hessian, and Laplacian.
- ▷ **Part IV: Mathematical Programming**. We will see one important use of calculus: optimization. The gradient descent algorithm could be useful for functional optimization, but there we focus on the case of optimization with constraints. We will see how to convert an optimization problem with constraint into one without. Then, in turn, we need to use differentiation to solve it.
- ▷ **Part V: Approximation**. Another use of differentiation is to find an approximate function. This can be useful if we prefer the function to be written as a polynomial. In fact, with the technique of the Taylor series, we can approximate any function with a polynomial as long as the function is differentiable.
- ▷ **Part VI: Calculus in Machine Learning**. In the final part of this book, we study the case of training a neural network and a support vector machine classifier. Both will need to evaluate some calculus to get it done correctly. The backpropagation in neural network training involves a Jacobian matrix, while a support vector machine classifier is indeed a constrained optimization problem. The chapters in this part will lay out the mathematical derivation and then convert them into Python code.

These are not designed to tell you everything but to give you an understanding of how they work and how to use them. This is to help you learn, and by doing so, you can get the result the fastest.

How to run the examples?

All examples in this book are in Python. The examples in each chapter are complete and standalone. You should be able to run it successfully as-is without modification, given you have installed the required packages. No special IDE or notebooks are required. A command line execution environment is all it needs in most cases. A complete working example is always given at the end of the chapter. To avoid mistakes with copy-and-paste, all source codes are also provided with this book. Please use them whenever possible for a better learning experience.

All code examples were tested on a POSIX-compatible machine with Python 3.7.

About Further Reading

Each lesson includes a list of further reading resources. This may include:

▷ Books and book chapters

▷ API documentation

▷ Articles and Webpages

Wherever possible, links to the relevant API documentation are provided in each lesson. Books referenced are provided with links to Amazon so you can learn more about them. If you find some good references, feel free to let us know so we can update this book.

I

Foundations

1 What is Calculus?

Calculus is the mathematical study of change. The effectiveness of calculus to solve a complicated but continuous problem lies in its ability to slice the problem into infinitely simpler parts, solve them separately, and subsequently rebuild them into the original whole. This strategy can be applied to study all continuous elements that can be sliced in this manner, be it the curvatures of geometric shapes, as well as the trajectory of an object in flight, or a time interval.

In this tutorial, you will discover the origins of calculus and its applications. After completing this tutorial, you will know:

▷ What is calculus?

▷ How can calculus be applied to the real-world?

Let's get started.

Overview

This tutorial is divided into two parts; they are:

▷ Calculus

▷ Applications of Calculus

1.1 Calculus

Calculus is a Latin word for stone, or pebble.

The use of this word has seeped into mathematics from the ancient practice of using little stones to perform calculations, such as addition and multiplication. While the use of this word has, with time, disappeared from the title of many methods of calculation, one important branch of mathematics retained it so much that we now refer to it as *The* Calculus.

> “Calculus, like other forms of mathematics, is much more than a language; it's also an incredibly powerful system of reasoning.”
>
> — Page xii, *Infinite Powers*, 2020.

Calculus matured from geometry.

At the start, geometry was concerned with straight lines, planes and angles, reflecting its utilitarian origins in the construction of ramps and pyramids, among other uses. Nonetheless, geometers found themselves tool-less for the study of circles, spheres, cylinders and cones. The surface areas and volumes of these curved shapes was found to be much more difficult to analyze than rectilinear shapes made of straight lines and flat planes. Despite its reputation for being complicated, the method of calculus grew out of a quest for simplicity, by breaking down complicated problems into simpler parts.

> “Back around 250 BCE in ancient Greece, it was a hot little mathematical startup devoted to the mystery of curves.”
>
> — Page 3, *Infinite Powers*, 2020.

In order to do so, calculus revolved around the controlled use of infinity as the bridge between the curved and the straight.

> “The Infinity Principle. To shed light on any continuous shape, object, motion, process, or phenomenon — no matter how wild and complicated it may appear — reimagine it as an infinite series of simpler parts, analyze those, and then add the results back together to make sense of the original whole.”
>
> — Page xvi, *Infinite Powers*, 2020.

To grasp this concept a little better, imagine yourself traveling on a spaceship to the moon. As you glance outwards to the moon from earth, its outline looks undoubtedly curved. But as you approach closer and smaller parts of the outline start filling up the viewing port, the curvature eases and becomes less defined. Eventually, the amount of curvature becomes so small that the infinitesimally small parts of the outline appear as a straight line. If we had to slice the circular shape of the moon along these infinitesimally small parts of its outline, and then arrange the infinitely small slices into a rectangle, then we would be able to calculate its area: by multiplying its width to its height.

This is the essence of calculus: the breakthrough that if one looks at a curved shape through a microscope, the portion of its curvature being zoomed upon will appear straight and flat. Hence, analyzing a curved shape is, in principle, made possible by putting together its many straight pieces.

Calculus can, therefore, be considered to comprise of two phases: cutting and rebuilding.

> “In mathematical terms, the cutting process always involves infinitely fine subtraction, which is used to quantify the differences between the parts. Accordingly, this half of the subject is called differential calculus. The reassembly process always involves infinite addition, which integrates the parts back into the original whole. This half of the subject is called integral calculus.”
>
> — Page xv, *Infinite Powers*, 2020.

With this in mind, let us revisit our simple example. Suppose that we have sliced the circular shape of the moon into smaller pieces, and rearranged the pieces alongside one another.

The shape that we have formed is similar to a rectangle having a width equal to half the circle circumference, $C/2$, and a height equal to the circle radius, r.

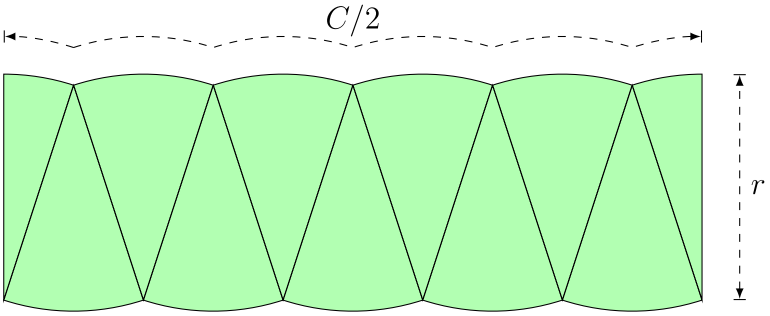


Figure 1.1: Rearranging slices of a circle into a rectangle.

To flatten out the curvature further, we can slice the circle into thinner pieces.

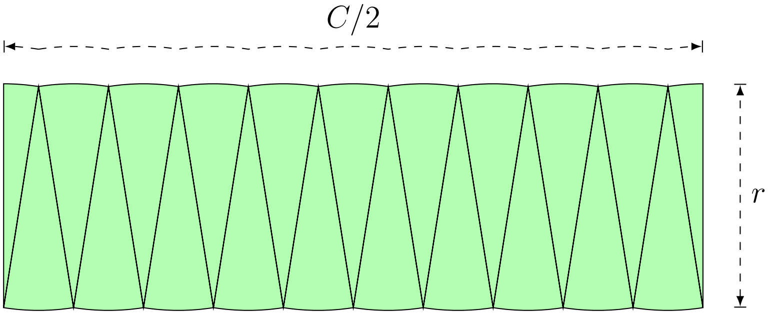


Figure 1.2: Rearranging thinner slices of a circle into a rectangle.

The thinner the slices, the more the curvature flattens out until we reach the limit of *infinitely* many slices, where the shape is now perfectly rectangular.

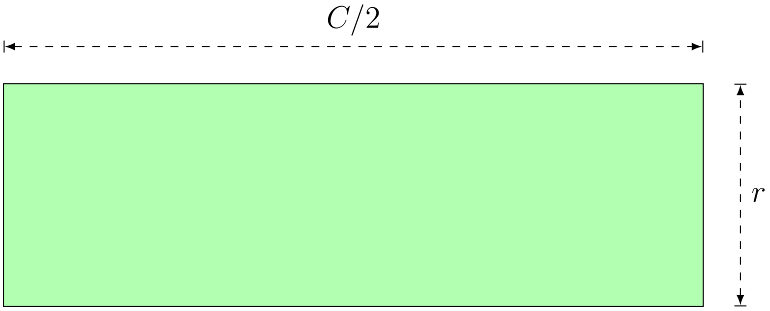


Figure 1.3: Rearranging infinitely thin slices of a circle into a rectangle.

We have cut out the slices from the circular shape, and rearranging them into a rectangle does not change their area. Hence, calculating the area of the circle is equivalent to calculating the area of the resulting rectangle: $A = rC/2$.

Curves are not only a characteristic of geometric shapes, but also appear in nature in the form of parabolic arcs traced by projectiles, or the elliptical orbits of planets around the sun.

> “ And so began the second great obsession: a fascination with the mysteries of motion on Earth and in the solar system. ”
>
> — Page xix, *Infinite Powers*, 2020.

And with curves and motion, the next natural question concerns their rate of change.

> With the mysteries of curves and motion now settled, calculus moved on to its third lifelong obsession: the mystery of change.
>
> — Page xxii, *Infinite Powers*, 2020.

It is through the application of the Infinity Principle that calculus allows us to study motion and change too, by approximating these into many infinitesimal steps.

It is for this reason that calculus has come to be considered the language of the universe.

1.2 Applications of calculus

Calculus has been applied in many domains, from Newton's application in solving problems of mathematical physics, to the more recent application of Newton's ideas in the work done at NASA by mathematician, Katherine Johnson, and her colleagues.

In the 1860s, James Clerk Maxwell used calculus to recast the experimental laws of electricity and magnetism, eventually predicting not only the existence of electromagnetic waves, but also revealing the nature of light as an electromagnetic wave. Based on his work, Nikola Tesla created the first radio communication system, Guglielmo Marconi transmitted the first wireless messages, and eventually many modern-day devices, such as the television and the smartphone, came into existence.

Albert Einstein, in 1917, also applied calculus to a model of atomic transitions, in order to predict the effect of stimulated emission. His work later led to the first working lasers in the 1960s, which have since then been used in many different devices, such as compact-disc players and bar code scanners.

> Without calculus, we wouldn't have cell phones, computers, or microwave ovens. We wouldn't have radio. Or television. Or ultrasound for expectant mothers, or GPS for lost travelers. We wouldn't have split the atom, unraveled the human genome, or put astronauts on the moon. We might not even have the Declaration of Independence.
>
> — Page vii, *Infinite Powers*, 2020.

More interestingly is the integral role of calculus in machine learning. It underlies important algorithms, such as gradient descent, which requires the computation of the gradient of a function and is often essential to train machine learning models. This makes calculus one of the fundamental mathematical tools in machine learning.

1.3 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Steven Strogatz. *Infinite Powers. How Calculus Reveals the Secrets of the Universe*. Mariner Books, 2020.
https://www.amazon.com/dp/0358299284/

Mark Ryan. *Calculus Essentials For Dummies*. Wiley, 2019.
https://www.amazon.com/dp/1119591201/

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Michael Spivak. *The Hitchhiker's Guide to Calculus*. American Mathematical Society, 2019.
https://www.amazon.com/dp/1470449625/

Marc Peter Deisenroth. *Mathematics for Machine Learning*. Cambridge University Press, 2020.
https://www.amazon.com/dp/110845514X

Articles

Calculus. Wikipedia.
https://en.wikipedia.org/wiki/Calculus

1.4 Summary

In this tutorial, you discovered the origins of calculus and its applications. Specifically, you learned:

- ▷ That calculus is the mathematical study of change that is based on a cutting and rebuilding strategy.
- ▷ That calculus has permitted many discoveries and the creation of many modern-day devices as we known them, and is also a fundamental mathematical tool in machine learning.

In the next chapter, we are going to see the algebraic meaning of calculus, namely, the rate of change.

Rate of Change

2

The measurement of the rate of change is an integral concept in differential calculus, which concerns the mathematics of change and infinitesimals. It allows us to find the relationship between two changing variables and how these affect one another. The measurement of the rate of change is also essential for machine learning, such as in applying gradient descent as the optimization algorithm to train a neural network model.

In this tutorial, you will discover the rate of change as one of the key concepts in calculus, and the importance of measuring it. After completing this tutorial, you will know:

- How the rate of change of linear and nonlinear functions is measured.
- Why the measurement of the rate of change is an important concept in different fields.

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- Rate of Change
- The Importance of Measuring the Rate of Change

2.1 Rate of change

The rate of change defines the relationship of one changing variable with respect to another.

Consider a moving object that is displacing twice as much in the vertical direction, denoted by y, as it is in the horizontal direction, denoted by x. In mathematical terms, this may be expressed as:

$$\delta y = 2\delta x$$

The Greek letter *delta*, δ, is often used to denote *difference* or *change*. Hence, the equation above defines the relationship between the change in the x-position with respect to the change in the y-position of the moving object.

This change in the x and y-directions can be graphed by a straight line on an x-y coordinate system.

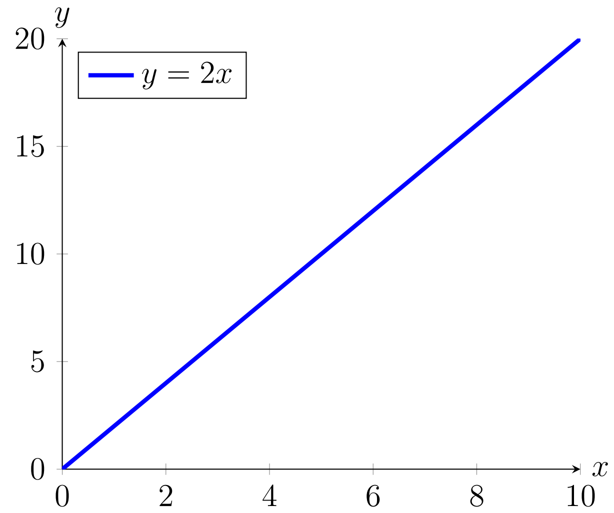


Figure 2.1: Line plot of a linear function

In this graphical representation of the object's movement, the rate of change is represented by the *slope* of the line, or its gradient. Since the line can be seen to *rise* 2 units for each single unit that it *runs* to the right, then its rate of change, or its slope, is equal to 2.

> Rates and slopes have a simple connection. The previous rate examples can be graphed on an x-y coordinate system, where each rate appears as a slope.
>
> — Page 38, *Calculus Essentials For Dummies*, 2019.

Tying everything together, we see that:

$$\text{rate of change} = \frac{\delta y}{\delta x} = \frac{\text{rise}}{\text{run}} = \text{slope}$$

If we had to consider two particular points, $P_1 = (2, 4)$ and $P_2 = (8, 16)$, on this straight line, we may confirm the slope to be equal to:

$$\text{slope} = \frac{\delta y}{\delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{16 - 4}{8 - 2} = 2$$

For this particular example, the rate of change, represented by the slope, is positive since the direction of the line is increasing rightwards. However, the rate of change can also be negative if the direction of the line decreases, which means that the value of y would be decreasing as the value of x increases. Furthermore, when the value of y remains constant as x increases, we would say that we have *zero* rate of change. If, otherwise, the value of x remains constant as y increases, we would consider the range of change to be *infinite*, because the slope of a vertical line is considered undefined.

So far, we have considered the simplest example of having a straight line, and hence a linear function, with an unchanging slope. Nonetheless, not all functions are this simple, and if they were, there would be no need for calculus.

> Calculus is the mathematics of change, so now is a good time to move on to parabolas, curves with changing slopes.
>
> — Page 39, *Calculus Essentials For Dummies*, 2019.

Let us consider a simple nonlinear function, a parabola:

$$y = \frac{1}{4}x^2$$

In contrast to the constant slope that characterizes a straight line, we may notice how this parabola becomes steeper and steeper as we move rightwards.

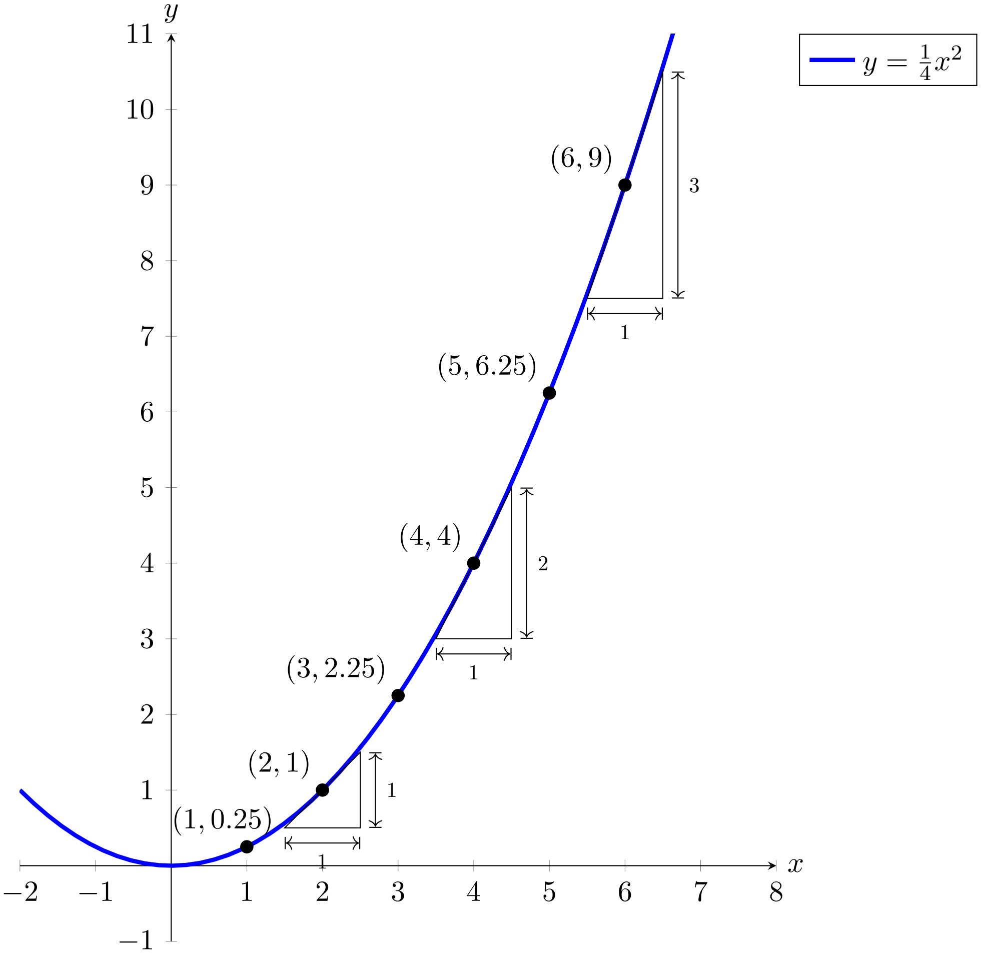


Figure 2.2: Line plot of a parabola

Recall that the method of calculus allows us to analyze a curved shape by cutting it into many infinitesimal straight pieces arranged alongside one another. If we had to consider one of such pieces at some particular point, P, on the curved shape of the parabola, we see that we find ourselves calculating again the rate of change as the slope of a straight line. It is important to keep in mind that the rate of change on a parabola depends on the particular point, P, that we happened to consider in the first place.

For example, if we had to consider the straight line that passes through point, $P = (2, 1)$, we find that the rate of change at this point on the parabola is:

$$\text{rate of change} = \frac{\delta y}{\delta x} = \frac{1}{1} = 1$$

If we had to consider a different point on the same parabola, at $P = (6, 9)$, we find that the rate of change at this point is equal to:

$$\text{rate of change} = \frac{\delta y}{\delta x} = \frac{3}{1} = 3$$

The straight line that *touches* the curve as some particular point, P, is known as the *tangent* line, whereas the process of calculating the rate of change of a function is also known as finding its *derivative*.

> A derivative is simply a measure of how much one thing changes compared to another — and that's a rate.
>
> — Page 37, *Calculus Essentials For Dummies*, 2019.

While we have considered a simple parabola for this example, we may similarly use calculus to analyze more complicated nonlinear functions. The concept of computing the instantaneous rate of change at different tangential points on the curve remains the same.

We meet one such example when we come to train a neural network using the gradient descent algorithm. As the optimization algorithm, gradient descent iteratively descends an error function towards its global minimum, each time updating the neural network weights to model better the training data. The error function is, typically, nonlinear and can contain many local minima and saddle points. In order to find its way downhill, the gradient descent algorithm computes the instantaneous slope at different points on the error function, until it reaches a point at which the error is lowest and the rate of change is zero.

2.2 The importance of measuring the rate of change

We have, thus far, considered the rate of change per unit on the x-y coordinate system.

> But a rate can be anything per anything.
>
> — Page 38, *Calculus Essentials For Dummies*, 2019.

Within the context of training a neural network, for instance, we have seen that the error gradient is computed as the change in error with respect to a specific weight in the neural network.

There are many different fields in which the measurement of the rate of change is an important concept too. A few examples are:

- In physics, *speed* is computed as the change in position per unit time.
- In signal digitization, *sampling rate* is computed as the number of signal samples per second.
- In computing, *bit rate* is the number of bits the computer processes per unit time.
- In finance, *exchange rate* refers to the value of one currency with respect to another.

> In either case, every rate is a derivative, and every derivative is a rate.
>
> — Page 38, *Calculus Essentials For Dummies*, 2019.

2.3 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Mark Ryan. *Calculus Essentials For Dummies.* Wiley, 2019.
https://www.amazon.com/dp/1119591201/

Steven Strogatz. *Infinite Powers. How Calculus Reveals the Secrets of the Universe.* Mariner Books, 2020.
https://www.amazon.com/dp/0358299284/

Mark Ryan. *Calculus For Dummies.* 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Michael Spivak. *The Hitchhiker's Guide to Calculus.* American Mathematical Society, 2019.
https://www.amazon.com/dp/1470449625/

Marc Peter Deisenroth. *Mathematics for Machine Learning.* Cambridge University Press, 2020.
https://www.amazon.com/dp/110845514X

2.4 Summary

In this tutorial, you discovered the rate of change as one of the key concepts in calculus, and the importance of measuring it. Specifically, you learned:

- The measurement of the rate of change is an integral concept in differential calculus that allows us to find the relationship of one changing variable with respect to another.
- This is an important concept that can be applied to many fields, one of which is machine learning.

In the next chapter, we will drill deeper on why calculus is helpful in building machine learning algorithms.

3 Why it Works?

Calculus is one of the core mathematical concepts in machine learning that permits us to understand the internal workings of different machine learning algorithms. One of the important applications of calculus in machine learning is the gradient descent algorithm, which, in tandem with backpropagation, allows us to train a neural network model.

In this tutorial, you will discover the integral role of calculus in machine learning. After completing this tutorial, you will know:

▷ Calculus plays an integral role in understanding the internal workings of machine learning algorithms, such as the gradient descent algorithm for minimizing an error function.

▷ Calculus provides us with the necessary tools to optimize complex objective functions as well as functions with multidimensional inputs, which are representative of different machine learning applications.

Let's get started.

Overview

This tutorial is divided into two parts; they are:

▷ Calculus in Machine Learning

▷ Why Calculus in Machine Learning Works

3.1 Calculus in machine learning

A neural network model, whether shallow or deep, implements a function that maps a set of inputs to expected outputs.

The function implemented by the neural network is learned through a training process, which iteratively searches for a set of weights that best enable the neural network to model the variations in the training data.

> A very simple type of function is a linear mapping from a single input to a single output.
>
> — Page 187, *Deep Learning*, 2019.

Such a linear function can be represented by the equation of a line having a slope, m, and a y-intercept, c:

$$y = mx + c$$

Varying each of parameters, m and c, produces different linear models that define different input-output mappings.

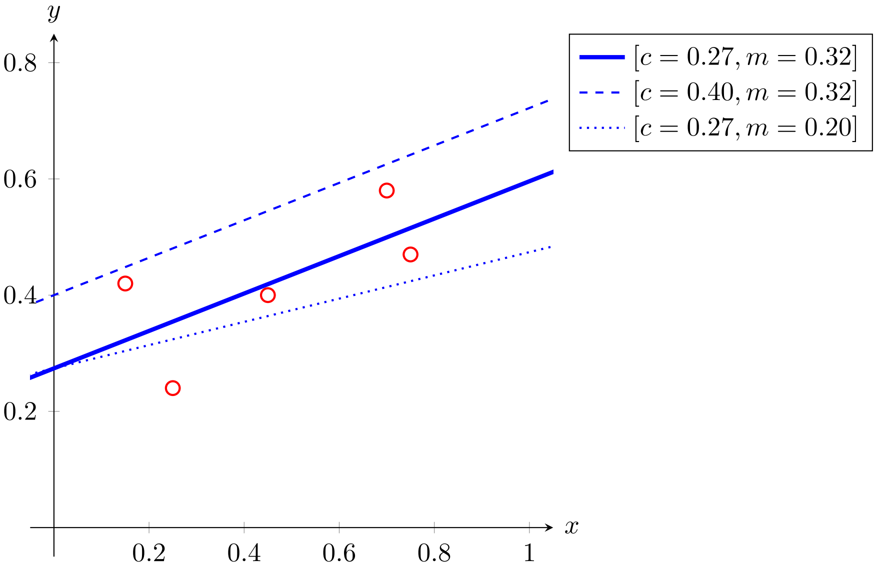


Figure 3.1: Line plot of different line models produced by varying the slope and intercept

The process of learning the mapping function, therefore, involves the approximation of these model parameters, or *weights*, that result in the minimum error between the predicted and target outputs. This error is calculated by means of a loss function, cost function, or error function, as often used interchangeably, and the process of minimizing the loss is referred to as *function optimization*.

We can apply differential calculus to the process of function optimization. In order to understand better how differential calculus can be applied to function optimization, let us return to our specific example of having a linear mapping function.

Say that we have some dataset of single input features, x, and their corresponding target outputs, y. In order to measure the error on the dataset, we shall be taking the sum of squared errors (SSE), computed between the predicted and target outputs, as our loss function.

Carrying out a parameter sweep across different values for the model weights, $w_0 = m$ and $w_1 = c$, generates individual error profiles that are convex in shape (i.e., like letter U, as in Figure 3.2).

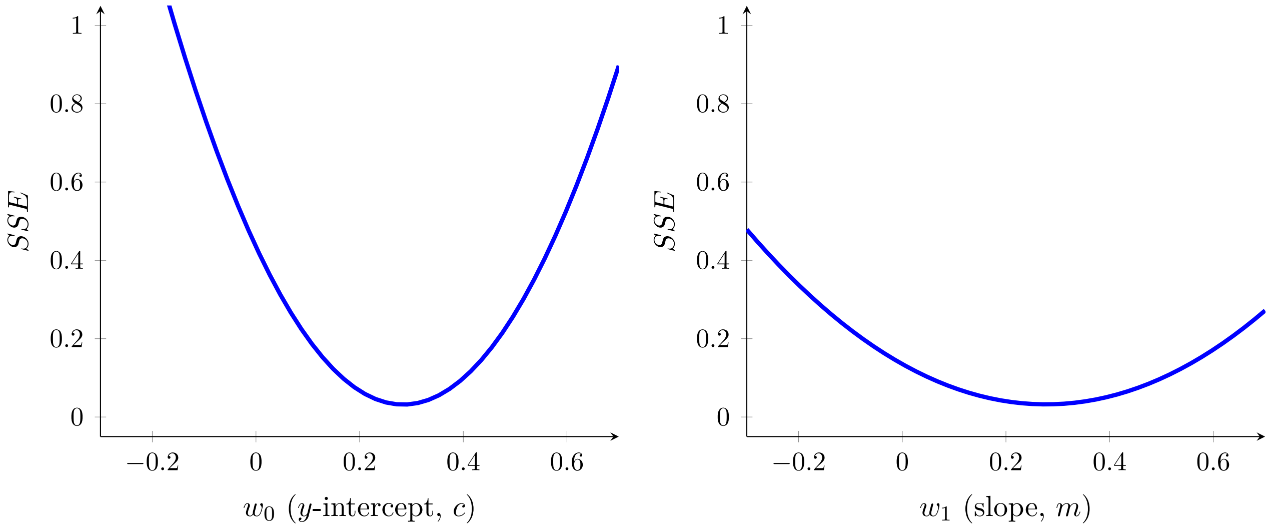


Figure 3.2: Line plots of error (SSE) profiles generated when sweeping across a range of values for the slope and intercept

Combining the individual error profiles generates a three-dimensional error surface that is also convex in shape. This error surface is contained within a weight space, which is defined by the swept ranges of values for the model weights, w_0 and w_1.

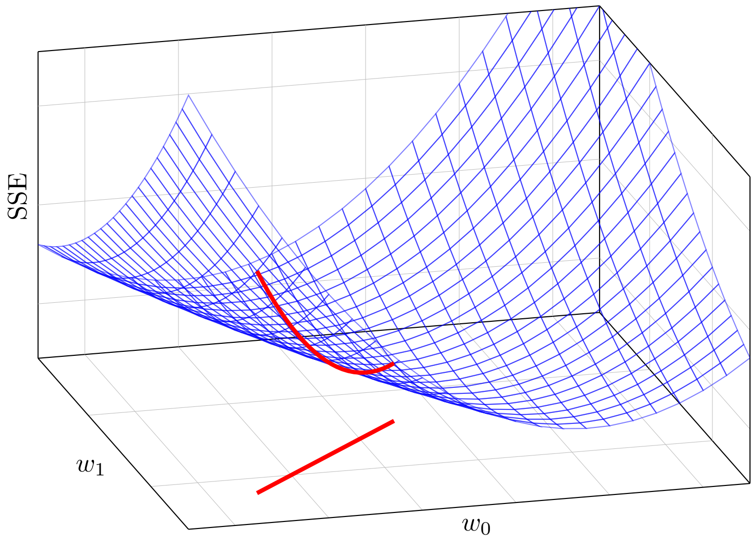


Figure 3.3: Three-dimensional plot of the error (SSE) surface generated when both slope and intercept are varied

Moving across this weight space is equivalent to moving between different linear models. Our objective is to identify the model that best fits the data among all possible alternatives. The best model is characterized by the lowest error on the dataset, which corresponds with the lowest point on the error surface.

> A convex or bowl-shaped error surface is incredibly useful for learning a linear function to model a dataset because it means that the learning process can be framed as a search for the lowest point on the error surface. The standard algorithm used to find this lowest point is known as gradient descent.
>
> — Page 194, *Deep Learning*, 2019.

The gradient descent algorithm, as the optimization algorithm, will seek to reach the lowest point on the error surface by following its gradient downhill. This descent is based upon the computation of the gradient, or slope, of the error surface.

This is where differential calculus comes into the picture.

> Calculus, and in particular differentiation, is the field of mathematics that deals with rates of change.
>
> — Page 198, *Deep Learning*, 2019.

More formally, let us denote the function that we would like to optimize by:

$$\text{error} = f(\text{weights})$$

By computing the rate of change, or the slope, of the error with respect to the weights, the gradient descent algorithm can decide on how to change the weights in order to keep reducing the error.

3.2 Why calculus in machine learning works

The error function that we have considered to optimize is relatively simple, because it is convex and characterized by a single global minimum. Nonetheless, in the context of machine learning, we often need to optimize more complex functions that can make the optimization task very challenging. Optimization can become even more challenging if the input to the function is also multidimensional.

Calculus provides us with the necessary tools to address both challenges. Suppose that we have a more generic function that we wish to minimize, and which takes a real input, x, to produce a real output, y:

$$y = f(x)$$

Computing the rate of change at different values of x is useful because it gives us an indication of the changes that we need to apply to x, in order to obtain the corresponding changes in y.

Since we are minimizing the function, our goal is to reach a point that obtains as low a value of $f(x)$ as possible that is also characterized by zero rate of change; hence, a global minimum. Depending on the complexity of the function, this may not necessarily be possible since there may be many local minima (i.e., minimum point among its neighbor but not necessary minimum over all region) or saddle points that the optimization algorithm may remain caught into.

> In the context of deep learning, we optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions
>
> — Page 84, *Deep Learning*, 2016.

Hence, within the context of deep learning, we often accept a suboptimal solution that may not necessarily correspond to a global minimum, so long as it corresponds to a very low value of $f(x)$.

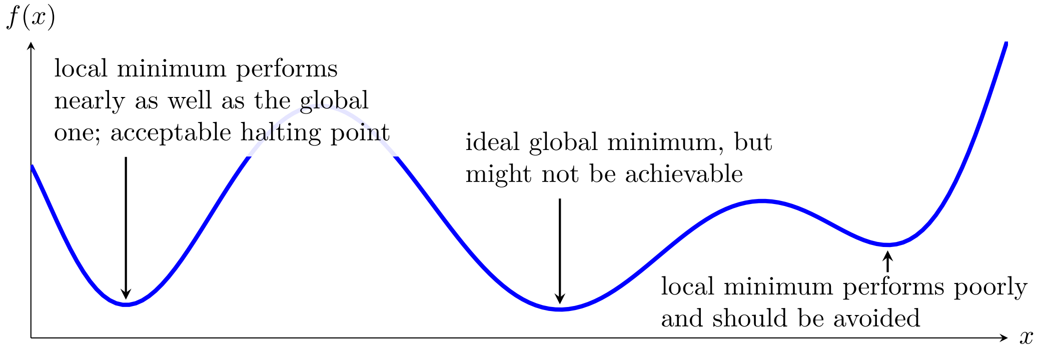


Figure 3.4: Line plot of cost function to minimize displaying local and global minima.

If the function we are working with takes multiple inputs, calculus also provides us with the concept of *partial derivatives*; or in simpler terms, a method to calculate the rate of change of y with respect to changes in each one of the inputs, x, while holding the remaining inputs constant.

> This is why each of the weights is updated independently in the gradient descent algorithm: the weight update rule is dependent on the partial derivative of the SSE for each weight, and because there is a different partial derivative for each weight, there is a separate weight update rule for each weight.
>
> — Page 200, *Deep Learning*, 2019.

Hence, if we consider again the minimization of an error function, calculating the partial derivative for the error with respect to each specific weight permits that each weight is updated independently of the others.

This also means that the gradient descent algorithm may not follow a straight path down the error surface. Rather, each weight will be updated in proportion to the local gradient of the error curve. Hence, one weight may be updated by a larger amount than another, as much as needed for the gradient descent algorithm to reach the function minimum.

3.3 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

John D. Kelleher. *Deep Learning*. Illustrated edition. The MIT Press Essential Knowledge series. MIT Press, 2019.
https://www.amazon.com/dp/0262537559/

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

3.4 Summary

In this tutorial, you discovered the integral role of calculus in machine learning. Specifically, you learned:

- ▷ Calculus plays an integral role in understanding the internal workings of machine learning algorithms, such as the gradient descent algorithm that minimizes an error function based on the computation of the rate of change.
- ▷ The concept of the rate of change in calculus can also be exploited to minimize more complex objective functions that are not necessarily convex in shape.
- ▷ The calculation of the partial derivative, another important concept in calculus, permits us to work with functions that take multiple inputs.

In the next chapter, we are going to review what we should know before we move on to learn about how to manipulate the math.

A Brief Tour of Calculus Prerequisites

4

We have previously seen that calculus is one of the core mathematical concepts in machine learning that permits us to understand the internal workings of different machine learning algorithms.

Calculus, in turn, builds on several fundamental concepts that derive from algebra and geometry. The importance of having these fundamentals at hand will become even more important as we work our way through more advanced topics of calculus, such as the evaluation of limits and the computation of derivatives, to name a few.

In this tutorial, you will discover several prerequisites that will help you work with calculus. After completing this tutorial, you will know:

- Linear and nonlinear functions are central to calculus and machine learning, and many calculus problems involve their use.
- Fundamental concepts from algebra and trigonometry provide the foundations for calculus, and will become especially important as we tackle more advanced calculus topics.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- The Concept of a Function
- Fundamentals of Pre-Algebra and Algebra
- Fundamentals of Trigonometry

4.1 The concept of a function

A function is a rule that defines the relationship between a dependent variable and an independent variable.

> Examples are all around us: The average daily temperature for your city depends on, and is a function of, the time of year; the distance an object has fallen is a function of how much time has elapsed since you dropped it; the area of a circle is a function of its radius; and the pressure of an enclosed gas is a function of its temperature.
>
> — Page 43, *Calculus For Dummies*, 2016.

In machine learning, a neural network learns a function by which it can represent the relationship between features in the input, the independent variable, and the expected output, the dependent variable. In such a scenario, therefore, the learned function defines a deterministic mapping between the input values and one or more output values. We can represent this mapping as follows:

$$\text{Output}(s) = \text{function}(\text{Input})$$

More formally, however, a function is often represented by $y = f(x)$, which translates to y *is a function of* x. This notation specifies x as the independent input variable that we already know, whereas y is the dependent output variable that we wish to find. For example, if we consider the squaring function, $f(x) = x^2$, then inputting a value of 3 would produce an output of 9:

$$y = f(3) = 9$$

A function can also be represented pictorially by a *graph* on an x-y coordinate plane.

> By the graph of the function f we mean the collection of all points $(x, f(x))$.
>
> — Page 13, *The Hitchhiker's Guide to Calculus*, 2019.

When graphing a function, the independent input variable is placed on the x-axis, while the dependent output variable goes on the y-axis. A graph helps to illustrate the relationship between the independent and dependent variables better: is the graph (and, hence, the relationship) rising or falling, and by which rate?

A straight line is one of the simplest functions that can be graphed on the coordinate plane. Take, for example, the graph of the line $y = \frac{1}{3}x + \frac{5}{3}$:

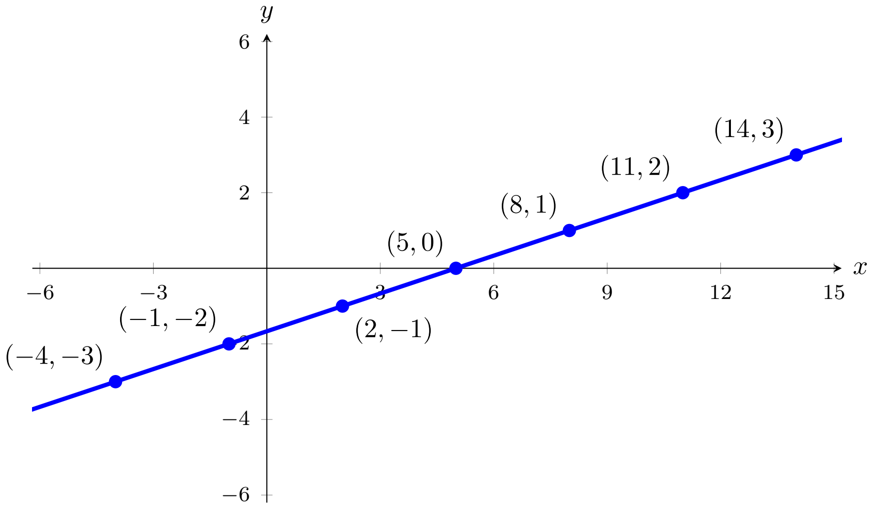


Figure 4.1: Line plot of a linear function

This straight line can be described by a *linear* function, so called because the output changes proportionally to any change in the input. The linear function that describes this straight line can be represented in slope-intercept form, where the slope is denoted by m, and the y-intercept by c:

$$f(x) = mx + c = \frac{1}{3}x - \frac{5}{3}$$

We had seen how to calculate the slope when we addressing the topic of rate of change in the last chapter. If we had to consider the special case of setting the slope to zero, the resulting horizontal line would be described by a $f(x) = c = -\frac{5}{3}$

Within the context of machine learning, the calculation defined by such a linear function is implemented by every neuron in a neural network. Specifically, each neuron receives a set of n inputs, x_i, from the previous layer of neurons or from the training data, and calculates a weighted sum of these inputs (where the *weight*, w_i, is more common term for the slope, m, in machine learning) to produce an output, z:

$$z = \sum_{i=1}^{n} x_i \times w_i$$

The process of training a neural network involves learning the weights that best represent the patterns in the input dataset, which process is carried out by the gradient descent algorithm.

In addition to the linear function, there exists another family of *nonlinear* functions.

The simplest of all nonlinear functions can be considered to be the parabola, that may be described by:

$$y = f(x) = x^2$$

When graphed, we find that this is an even function, because it is symmetric about the y-axis, and never falls below the x-axis.

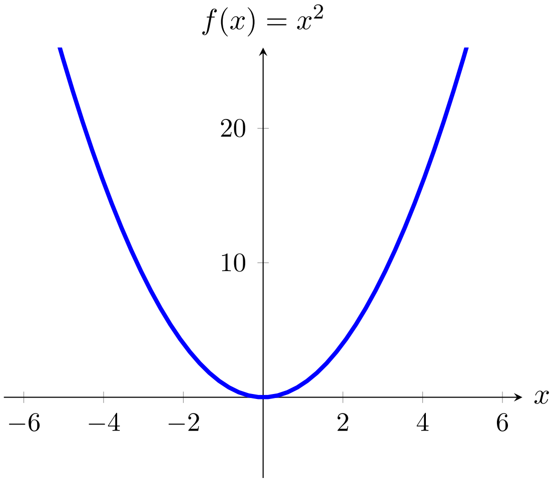


Figure 4.2: Line plot of a parabola

Nonetheless, nonlinear functions can take many different shapes. Consider, for instance, the exponential function of the form $f(x) = b^x$, which grows or decays indefinitely, or *monotonically*, depending on the value of x:

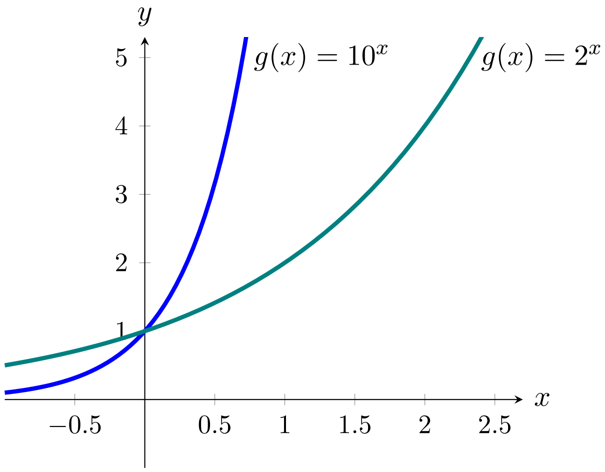


Figure 4.3: Line plot of an exponential function

Or the logarithmic function of the form $f(x) = \log_2 x$, which is similar to the exponential function but with the x- and y-axes switched:

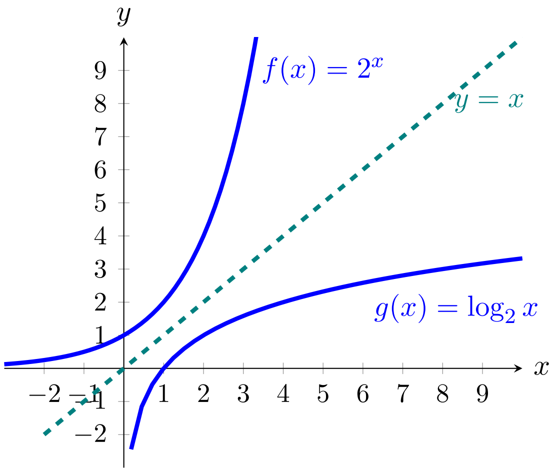


Figure 4.4: Line plot of a logarithmic function

Of particular interest for deep learning are the logistic, tanh, and the rectified linear units (ReLU) nonlinear functions, which serve as *activation functions*:

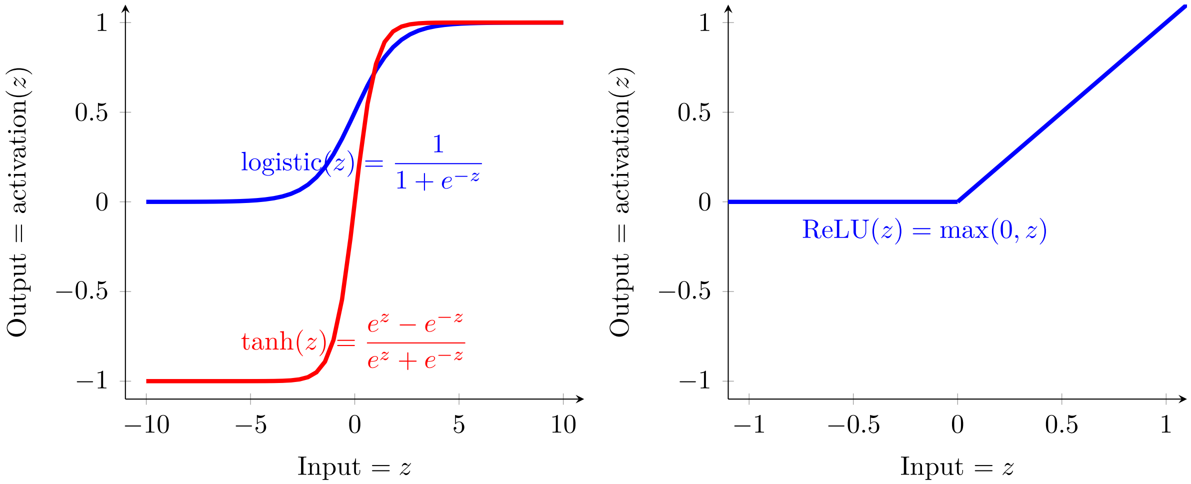


Figure 4.5: Line plots of the logistic, tanh and ReLU functions

The importance of these activation functions lies in the introduction of a nonlinear mapping into the processing of a neuron. If we had to rely solely on the linear regression performed by each neuron in calculating a weighted sum of the inputs, then we would be restricted to learning only a linear mapping from the inputs to the outputs. However, many real-world relationships are more complex than this, and a linear mapping would not accurately model them. Introducing a nonlinearity to the output, z, of the neuron, allows the neural network to model such nonlinear relationships:

$$\text{Output} = \text{activation}(z)$$

> “ ...a neuron, the fundamental building block of neural networks and deep learning, is defined by a simple two-step sequence of operations: calculating a weighted sum and then passing the result through an activation function. ”
>
> — Page 76, *Deep Learning*, 2019.

Nonlinear functions appear elsewhere in the process of training a neural network too, in the form of error functions. A nonlinear error function can be generated by calculating the error between the predicted and the target output values as the weights of the model change. Its shape can be as simple as a parabola, but most often it is characterized by many local minima and saddle points. The gradient descent algorithm descends this nonlinear error function by calculating the slope of the *tangent* line that touches the curve at some particular instance: another important concept in calculus that permits us to analyze complex curved functions by cutting them into many infinitesimal straight pieces arranged alongside one another.

4.2 Fundamentals of pre-algebra and algebra

Algebra is one of the important foundations of calculus.

> “ Algebra is the language of calculus. You can’t do calculus without knowing algebra any more than you can write Chinese poetry without knowing Chinese. ”
>
> — Page 29, *Calculus For Dummies*, 2016.

There are several fundamental concepts of algebra that turn out to be useful for calculus, such as those concerning fractions, powers, square roots, and logarithms.

Let's first start by revising the basics for working with fractions.

- ▷ **Division by Zero**: The denominator of a fraction can never be equal to zero. For example, the result of a fraction such as 5/0 is undefined. The intuition behind this is that if it is a number, say, x, you will end up making $5 = 0 \times x = 0$ and hence all numbers will be equal to zero.
- ▷ **Reciprocal**: The reciprocal of a fraction is its multiplicative inverse. In simpler terms, to find the reciprocal of a fraction, flip it upside down. Hence, the reciprocal of 3/4, for instance, becomes 4/3.
- ▷ **Multiplication of Fractions**: Multiplication between fractions is as straightforward as multiplying across the numerators, and multiplying across the denominators:
$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}$$
- ▷ **Division of Fractions**: The division of fractions is very similar to multiplication, but with an additional step; the reciprocal of the second fraction is first found before multiplying. Hence, considering again two generic fractions:
$$\frac{a}{b} \div \frac{c}{d} = \frac{a}{b} \times \frac{d}{c} = \frac{ad}{bc}$$
- ▷ **Addition of Fractions**: An important first step is to find a common denominator between all fractions to be added. Any common denominator will do, but we usually find the *least* common denominator. Finding the least common denominator is, at times, as simple as multiplying the denominators of all individual fractions:
$$\frac{a}{b} + \frac{c}{d} = \frac{ad + cb}{bd}$$
- ▷ **Subtraction of Fractions**: The subtraction of fractions follows a similar procedure as for the addition of fractions:
$$\frac{a}{b} - \frac{c}{d} = \frac{ad - cb}{bd}$$
- ▷ **Canceling in Fractions**: Fractions with an unbroken chain of multiplications across the entire numerator, as well as across the entire denominator, can be simplified by canceling out any common terms that appear in both the numerator and the denominator:
$$\frac{a^3b^2}{ac} = \frac{a^2b^2}{c}$$

The next important prerequisite for calculus revolves around exponents, or powers as they are also commonly referred to. There are several rules to keep in mind when working with powers too.

- ▷ **The Power of Zero**: The result of any number (whether rational or irrational, negative or positive, *except* for zero itself) raised to the power of zero, is equal to one:
$$x^0 = 1$$

▷ **Negative Powers**: A base number raised to a negative power turns into a fraction, but does not change sign:

$$x^{-a} = \frac{1}{x^a}$$

▷ **Fractional Powers**: A base number raised to a fractional power can be converted into a root problem:

$$x^{\frac{a}{b}} = \sqrt[b]{x}^a = \sqrt[b]{x^a}$$

▷ **Addition of Powers**: If two (or more) *equivalent* base terms are being multiplied to one another, then their powers may be added:

$$x^a \times x^b = x^{(a+b)}$$

▷ **Subtraction of Powers**: Similarly, if two equivalent base terms are being divided, then their power may be subtracted:

$$\frac{x^a}{x^b} = x^{a-b}$$

▷ **Power of Powers**: If a power is also raised to a power, then the two powers may be multiplied by one another:

$$(x^a)^b = x^{ab}$$

▷ **Distribution of Powers**: Whether the base numbers are being multiplied or divided, the power may be distributed to each variable. However, it *cannot* be distributed if the base numbers are, otherwise, being added or subtracted:

$$(xyz)^a = x^a y^a z^a$$

$$\left(\frac{x}{y}\right)^a = \frac{x^a}{y^a}$$

Similarly, we have rules for working with roots:

▷ **Identities**: $\sqrt{0} = 0$ and $\sqrt{1} = 1$

▷ **Products**: $\sqrt{a} \cdot \sqrt{b} = \sqrt{a \cdot b}$, $\sqrt[3]{a} \cdot \sqrt[3]{b} = \sqrt[3]{a \cdot b}$, and $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{a \cdot b}$

▷ **Quotients**: $\frac{\sqrt{a}}{\sqrt{b}} = \sqrt{\frac{a}{b}}$, $\frac{\sqrt[3]{a}}{\sqrt[3]{b}} = \sqrt[3]{\frac{a}{b}}$, and $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$

▷ **Multiplication of indices**: $\sqrt[3]{\sqrt[4]{a}} = \sqrt[12]{a}$ and $\sqrt[m]{\sqrt[n]{a}} = \sqrt[mn]{a}$

▷ **Even number root**: $\sqrt{a^2} = |a|$, $\sqrt[4]{a^4} = |a|$, $\sqrt[6]{a^6} = |a|$, and so on

▷ **Odd number root**: $\sqrt[3]{a^3} = a$, $\sqrt[5]{a^5} = a$, and so on

▷ **Common mistake to avoid**: $\sqrt{a^2 + b^2} \neq a + b$

and rules for working with logarithms:

▷ **Identities**: $\log_c 1 = 0$ and $\log_c c = 1$

▷ **Products**: $\log_c(ab) = \log_c a + \log_c b$

▷ **Quotients**: $\log_c \frac{a}{b} = \log_c a - \log_c b$

▷ **Exponents**: $\log_c a^b = b \log_c a$

▷ **Change of base**: $\log_a b = \dfrac{\log_c b}{\log_c a}$

▷ **Anti-logarithm**: $\log_a a^b = b$ and $a^{\log_a b} = b$

Finally, knowing how to solve quadratic equations can also come in handy in calculus. If the quadratic equation is factorizable, then the easiest method to solve it is to express the sum of terms in product form. For example, the following quadratic equation can be factored as follows:

$$x^2 - 9 = (x+3)(x-3) = 0$$

Setting each factor to zero permits us to find a solution to this equation, which in this case is $x = \pm 3$ Alternatively, the following quadratic formula may be used:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If we had to consider the same quadratic equation as above, then we would set the coefficient values to, $a = 1$, $b = 0$, and $c = 9$, which would again result in $x = \pm 3$ as our solution.

4.3 Fundamentals of trigonometry

Trigonometry revolves around three main trigonometric functions, which are the sine, the cosine and the tangent, and their reciprocals, which are the cosecant, the secant and the cotangent, respectively.

When applied to a right angled triangle, these three main functions allow us to calculate the lengths of the sides, or any of the other two acute angles of the triangle, depending on the information that we have available to start off with. Specifically, for some angle, x, in the following 3-4-5 triangle:

$$\sin x = \frac{O}{H} = \frac{3}{5}$$

$$\cos x = \frac{A}{H} = \frac{4}{5}$$

$$\tan x = \frac{O}{A} = \frac{3}{4}$$

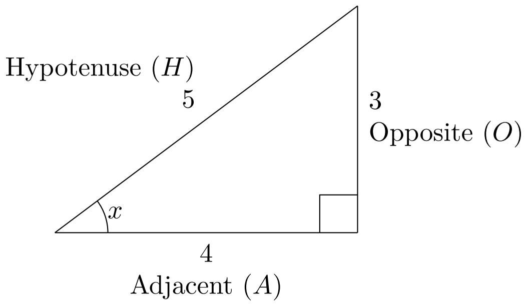


Figure 4.6: The 3-4-5 triangle

The sine, cosine and tangent functions only work with right-angled triangles, and hence can only be used in the calculation of acute angles that are smaller than 90°. Nonetheless, if we had to work within the *unit circle* on the x-y coordinate plane, then we would be able to apply trigonometry to all angles between 0°and 360°:

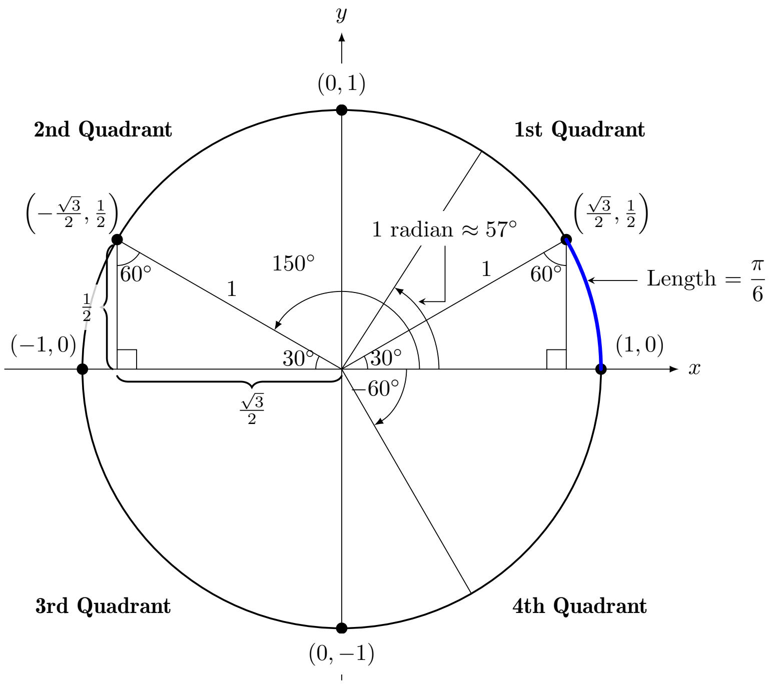


Figure 4.7: The unit circle

The unit circle has its center at the origin of the x-y coordinate plane, and a radius of one unit. Rotations around the unit circle are performed in a counterclockwise manner, starting from the positive x-axis. The cosine of the rotated angle would then be given by the x-coordinate of the point that hits the unit circle, whereas the y-coordinate specifies the sine of the rotated angle. It is also worth noting that the quadrants are symmetrical, and hence a point in one quadrant has symmetrical counterparts in the other three.

The graphed sine, cosine and tangent functions appear as follows:

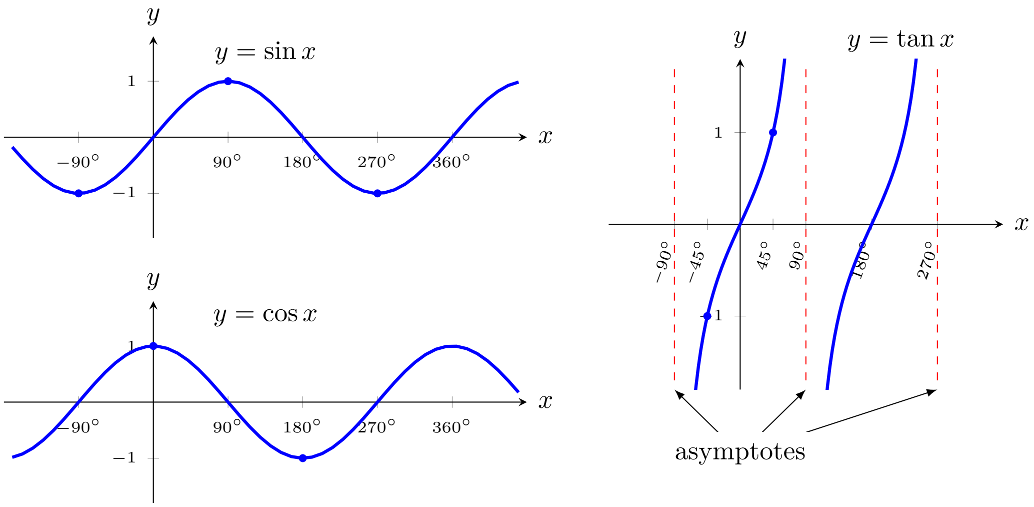


Figure 4.8: Line plots of the sine, cosine and tangent functions

All functions are periodic, with the sine and cosine functions featuring the same shape albeit being displaced by 90° between one another. The sine and cosine functions may, indeed, be easily sketched from the calculated x- and y-coordinates as one rotates around the unit circle. The tangent function may also be sketched similarly, since for any angle θ this function may be defined by:

$$\tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{y}{x}$$

The tangent function is undefined at ±90°, since the cosine in the denominator returns a value of zero at this angle. Hence, we draw vertical *asymptotes* at these angles, which are imaginary lines that the curve approaches but never touches.

One final note concerns the inverse of these trigonometric functions. Taking the sine function as an example, its inverse is denoted by $\sin^{-1}$. This is not to be mistaken for the cosecant function, which is rather the *reciprocal* of sine, and hence not the same as its inverse.

4.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Michael Spivak. *The Hitchhiker's Guide to Calculus*. American Mathematical Society, 2019.
https://www.amazon.com/dp/1470449625/

John D. Kelleher. *Deep Learning*. Illustrated edition. The MIT Press Essential Knowledge series. MIT Press, 2019.
https://www.amazon.com/dp/0262537559/

4.5 Summary

In this tutorial, you discovered several prerequisites for working with calculus.

Specifically, you learned:

- Linear and nonlinear functions are central to calculus and machine learning, and many calculus problems involve their use.
- Fundamental concepts from algebra and trigonometry provide the foundations for calculus, and will become especially important as we tackle more advanced calculus topics.

Starting from next chapter, we will see how we can manipulate the math in calculus. We will start with the process on how to evaluate a limit.

II

Limits and Differential Calculus

5

Limits and Continuity

There is no denying that calculus is a difficult subject. However, if you learn the fundamentals, you will not only be able to grasp the more complex concepts but also find them fascinating. To understand machine learning algorithms, you need to understand concepts such as gradient of a function, Hessians of a matrix, and optimization, etc. The concept of limits and continuity serves as a foundation for all these topics.

In this tutorial, you will discover how to evaluate the limit of a function, and how to determine if a function is continuous or not. After reading this tutorial, you will be able to:

- Determine if a function $f(x)$ has a limit as x approaches a certain value
- Evaluate the limit of a function $f(x)$ as x approaches a
- Determine if a function is continuous at a point or in an interval

Let's get started.

Overview

This tutorial is divided into two parts

- Limits
  - Determine if the limit of a function exists for a certain point
  - Compute the limit of a function for a certain point
  - Formal definition of a limit
  - Examples of limits
  - Left and right hand limits
- Continuity
  - Definition of continuity
  - Determine if a function is continuous at a point or within an interval
  - Examples of continuous functions

5.1 A simple example

Let's start by looking at a simple function $f(x)$ given by:

$$f(x) = 1 + x$$

What happens to $f(x)$ near -1?

We can see that $f(x)$ gets closer and closer to 0 as x gets closer and closer -1, from either side of $x = -1$. At $x = -1$, the function is exactly zero. We say that $f(x)$ has a limit equal to 0, when x approaches -1.

x	$f(x) = 1 + x$
-1.003	-0.003
-1.002	-0.002
-1.001	-0.001
-1	0
-0.999	0.001
-0.998	0.002
-0.997	0.003

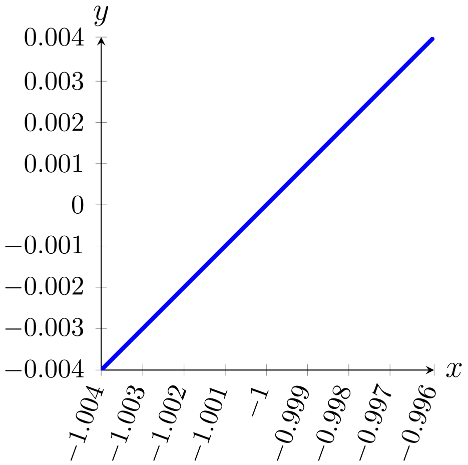


Figure 5.1: Plotting $f(x) = 1 + x$

Extend the example

Extending the problem. Let's define $g(x)$:

$$g(x) = \frac{1 - x^2}{1 + x}$$

We can simplify the expression for $g(x)$ as:

$$g(x) = \frac{(1 - x)(1 + x)}{1 + x}$$

If the denominator is not zero then $g(x)$ can be simplified as:

$$g(x) = 1 - x, \quad \text{if } x \neq -1$$

However, at $x = -1$, the denominator is zero and we cannot divide by zero. So it looks like there is a hole in the function at $x = -1$. Despite the presence of this hole, $g(x)$ gets closer and closer to 2 as x gets closer and closer -1, as shown in the figure:

x	$g(x) = \dfrac{1-x^2}{1+x}$
-1.003	2.003
-1.002	2.002
-1.001	2.001
-1	?
-0.999	1.999
-0.998	1.998
-0.997	1.997

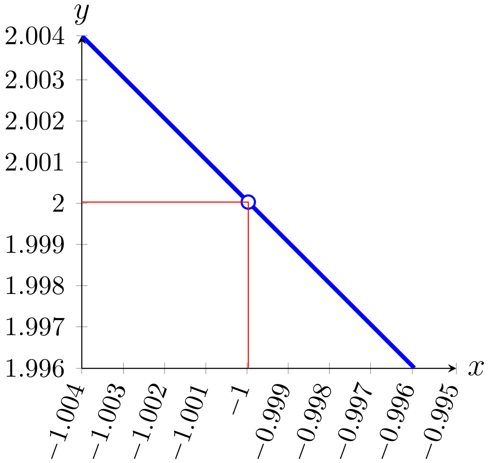


Figure 5.2: Plotting $g(x) = 1 - x$

This is the basic idea of a limit. If $g(x)$ is defined in an open interval that does not include -1, and $g(x)$ gets closer and closer to 2, as x approaches -1, we write this as:

$$\lim_{x \to -1} g(x) = 2$$

In general, for any function $f(x)$, if $f(x)$ gets closer and closer to a value L, as x gets closer and closer to k, we define the limit of $f(x)$ as x approaches k, as L. This is written as:

$$\lim_{x \to k} f(x) = L$$

Left and right hand limits

For the function $g(x)$, it doesn't matter whether we increase x to get closer to -1 (approach -1 from left) or decrease x to get closer to -1 (approach -1 from right), $g(x)$ still gets closer and closer to 2. This is shown in the figure below:

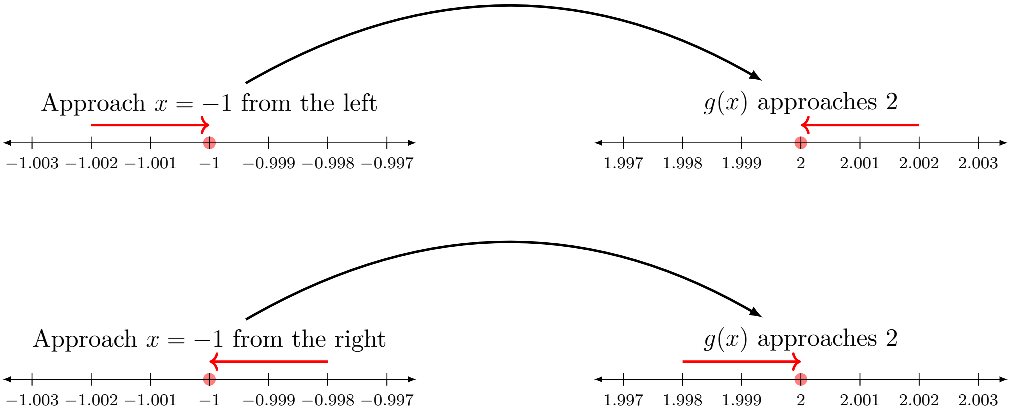


Figure 5.3: Left and right hand limit

This gives rise to the notion of one-sided limits. The left hand limit is defined on an interval to the left of -1, which does not include -1, e.g., $(-1.003, -1)$. As we approach -1 from the left, $g(x)$ gets closer to 2.

Similarly, the right hand limit is defined on an open interval to the right of -1 and does not include -1, e.g., $(-1, 0.997)$. As we approach -1 from the right, the right hand limit of $g(x)$ is 2. Both the left and right hand limits are written as follows:

(Left hand limit)

$$\lim_{x \to -1^-} g(x) = 2$$

(Right hand limit)

$$\lim_{x \to -1^+} g(x) = 2$$

We say that $f(x)$ has a limit L as x approaches k, if both its left and right hand limits are equal. Therefore, this is another way of testing whether a function has a limit at a specific point, i.e.,

$$\lim_{x \to k^-} f(x) = \lim_{x \to k^+} f(x) = L$$

If we want to compute left and right hand limits using a computer, it is not difficult to achieve. But first, we have to understand that computers cannot evaluate at *any* number since computers use floating point arithmetic, namely, not all numbers can be represented. The smallest floating point number that a computer can use is called the *machine epsilon*, and it can be found using a simple trick to trigger a floating point rounding error:

```
epsilon = 7/3.0 - 4/3.0 - 1.0
```

or we can also use a numpy function in Python to get the same result. We can make use of this concept to approximate the left and right hand limits. In the example above, $g(x) = 1 - x$, we can compute them as follows:

```python
import numpy as np

def g(x):
    return 1-x

x = -1
epsilon = np.finfo(float).eps

print(”Left limit is”, g(x-epsilon))
print(”Right limit is”, g(x+epsilon))
```

Program 5.1: Evaluating left and right limits for $g(x) = 1 - x$

The `np.finfo(float).eps` is to get the machine epsilon for the `float` type. Most likely it is a 64-bit floating point in your computer. We may also use a 32-bit floating point by saying `np.finfo(np.float32).eps` instead. The above code will compute the limit as follows:

```
Left limit is 2.0
Right limit is 1.9999999999999998
```

Output 5.1: Left and right limits as evaluated with numerically

5.2 Formal definition of a limit

In mathematics, we need to have an exact definition of everything. To define a limit formally, we'll use the notion of the Greek letter epsilon, ϵ. The mathematics community agrees to use ϵ for arbitrarily small positive numbers, which means we can make ϵ as small as we like and it can be as close to zero as we like, provided $\epsilon > 0$ (so ϵ cannot be zero).

The limit of $f(x)$ is L as x approaches k, if for every $\epsilon > 0$, there is a positive number $\delta > 0$, such that:

$$\text{if} \quad 0 < |x - k| < \delta \quad \text{then} \quad |f(x) - L| < \epsilon$$

The definition is quite straightforward. $x - k$ is the difference of x from k and $|x - k|$ is the distance of x from k that ignores the sign of the difference. Similarly, $|f(x) - L|$ is the distance of $f(x)$ from L. Hence, the definition says that when the distance of x from k approaches an arbitrary small value, the distance of $f(x)$ from L also approaches a very small value. The figure below is a good illustration of the above definition:

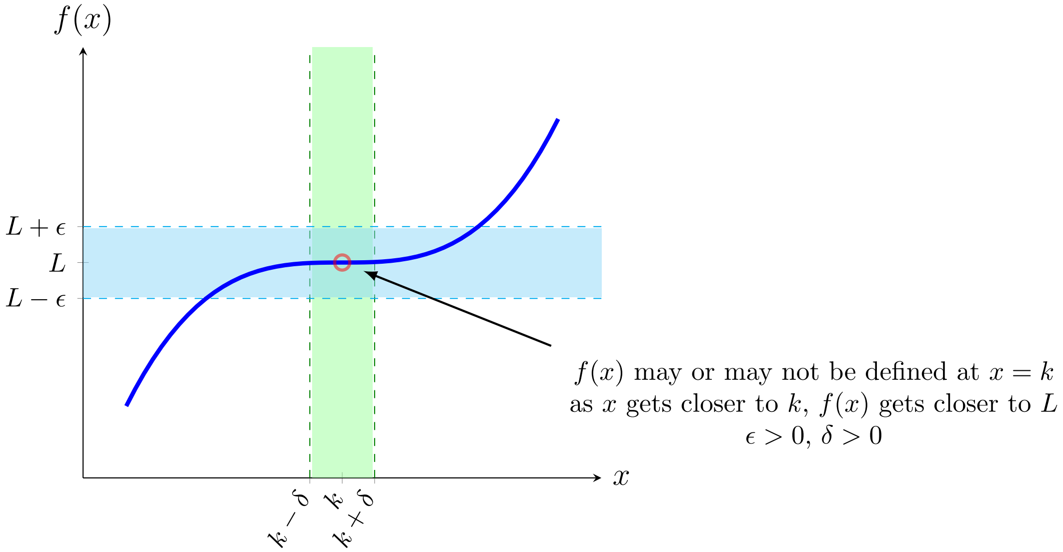
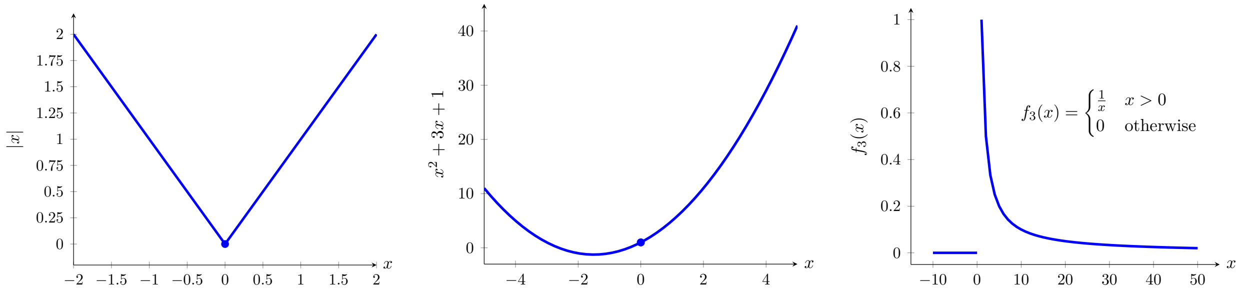

$f(x)$
$L + \epsilon$
L
$L - \epsilon$
x
$k - \delta$
k
$k + \delta$
$f(x)$ may or may not be defined at $x = k$
as x gets closer to k, $f(x)$ gets closer to L
$\epsilon > 0$, $\delta > 0$

Figure 5.4: Definition of a limit

5.3 Examples of limits

The figure below illustrates a few examples, which are also explained below:



$\lim_{x \to 0} f_1(x) = \lim_{x \to 0} |x| = 0$ $\quad$ $\lim_{x \to 0} f_2(x) = \lim_{x \to 0} (x^2 + 3x + 1) = 1$ $\quad$ $\lim_{x \to \infty} f_3(x) = 0$

Figure 5.5: Three examples of limits

Example with absolute value

$$f_1(x) = |x|$$

The limit of $f_1(x)$ exists at all values of x, e.g., $\lim_{x \to 0} f_1(x) = 0$.

Example with a polynomial

A polynomial is a sum of multiple terms of power of x, e.g.,

$$f_2(x) = x^2 + 3x + 1$$

The limit of $f_2(x)$ exists for all values of x, e.g., $\lim_{x \to 1} f_2(x) = 1 + 3 + 1 = 5$.

Example with infinity

$$f_3(x) = \begin{cases} 1/x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

For the above as x becomes larger and larger, the value of $f_3(x)$ gets smaller and smaller, approaching zero. Hence, $\lim_{x \to \infty} f_3(x) = 0$.

5.4 Example of functions that don't have a limit

From the definition of the limit, we can see that the following functions do not have a limit:

The unit step function

The unit step function $H(x)$ is given by:

$$H(x) = \begin{cases} 0, \text{if } x < 0 \\ 1, \text{otherwise} \end{cases}$$

As we get closer and closer to 0 from the left, the function remains a zero. However, as soon as we reach $x = 0$, $H(x)$ jumps to 1, and hence $H(x)$ does not have a limit as x approaches zero. This function has a left hand limit equal to zero and a right hand limit equal to 1.

The left and right hand limits do not agree, as $x \to 0$, hence $H(x)$ does not have a limit as x approaches 0. Here, we used the equality of left and right hand limits as a test to check if a function has a limit at a particular point.

The reciprocal function

Consider $h_1(x)$:

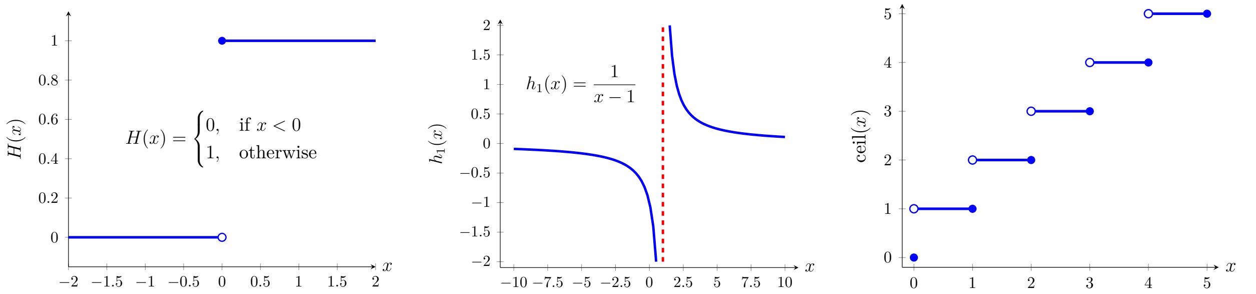
$$h_1(x) = \frac{1}{x-1}$$

As we approach $x = 1$ from the left side, the function tends to have large negative values. As we approach $x = 1$, from the right, $h_1(x)$ increases to large positive values. So when x is close to 1, the values of $h_1(x)$ do not stay close to a fixed real value. Hence, the limit does not exist for $x \to 1$.

The ceil function

Consider the ceiling function that rounds a real number with a non-zero fractional part to the next integer value. Hence, $\lim_{x \to 1} \text{ceil}(x)$ does not exist. In fact $\text{ceil}(x)$ does not have a limit at any integer value.

All the above examples are shown in the figure below:



No limit at $x = 0$ *No limit at $x = 1$* *ceil(x) has no limit at integer points*

Figure 5.6: Three examples of no limits

5.5 Continuity

If you have understood the notion of a limit, then it is easy to understand continuity. A function f(x) is continuous at a point a, if the following three conditions are met:

1. $f(a)$ should exist
2. $f(x)$ has a limit as x approaches a
3. The limit of $f(x)$ as $x \to a$ is equal to $f(a)$

If all of the above hold true, then the function is continuous at the point a. Some examples follow:

Examples of continuity

The concept of continuity is closely related to limits. If the function is defined at a point, has no jumps at that point, and has a limit at that point, then it is continuous at that point. The figure below shows some examples, which are explained below:

The square function

The following function $f_4(x)$ is continuous for all values of x:

$$f_4(x) = x^2$$

The rational function

Our previously used function $g(x)$:

$$g(x) = \frac{1-x^2}{1+x}$$

$g(x)$ is continuous everywhere except at $x = -1$ because that will be a division by zero.

We can modify $g(x)$ as $g^*(x)$:

$$g^*(x) = \begin{cases} \dfrac{1-x^2}{1+x}, & \text{if } x \neq -1 \\ 2, & \text{otherwise.} \end{cases}$$

Now we have a function that is continuous for all values of x. We come up with this by knowing that $1 - x^2 = (1-x)(1+x)$ and hence if $x \neq -1$, $g(x)$ would be same as $1 - x$.

The reciprocal function

Going back to our previous example of $f_3(x)$:

$$f_3(x) = \begin{cases} 1/x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

$f_3(x)$ is continuous everywhere, except at $x = 0$ as the value of $f_3(x)$ has a big jump at $x = 0$. Hence, there is a discontinuity at $x = 0$.

5.6 Further reading

This section provides more resources on the topic if you are looking to go deeper. Math is all about practice, and below is a list of resources that will provide more exercises and examples on this topic.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

5.7 Summary

In this tutorial, you discovered calculus concepts on limits and continuity. Specifically, you learned:

- Whether a function has a limit when approaching a point
- Whether a function is continuous at a point or within an interval

As we understand what a limit of a mathematical function is about, we will see how we can evaluate it in the next chapter.

Evaluating Limits

The concept of the limit of a function dates back to Greek scholars such as Eudoxus and Archimedes. While they never formally defined limits, many of their calculations were based upon this concept. Isaac Newton formally defined the notion of a limit and Cauchy refined this idea. Limits form the basis of calculus, which in turn defines the foundation of many machine learning algorithms. Hence, it is important to understand how limits of different types of functions are evaluated.

In this tutorial, you will discover how to evaluate the limits of different types of functions. After completing this tutorial, you will know:

- The different rules for evaluating limits
- How to evaluate the limit of polynomials and rational functions
- How to evaluate the limit of a function with discontinuities
- The Sandwich Theorem

Let's get started.

Overview

This tutorial is divided into 3 parts; they are:

- Rules for limits
  - Examples of evaluating limits using the rules for limits
  - Limits for polynomials
  - Limits for rational expressions
- Limits for functions with a discontinuity
- The Sandwich Theorem

6.1 Rules for limits

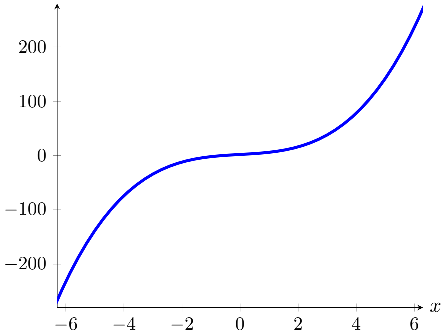
Limits are easy to evaluate if we know a few simple principles, which are listed below. All these rules are based on the known limits of two functions $f(x)$ and $g(x)$, when x approaches a point k:

$\lim\limits_{x\to k} f(x) = L$	$\lim\limits_{x\to k} g(x) = M$		
Constant multiple rule	$\lim\limits_{x\to k}(af(x))$	$= aL$	
Sum rule	$\lim\limits_{x\to k}(f(x)+g(x))$	$= L + M$	
Difference rule	$\lim\limits_{x\to k}(f(x)-g(x))$	$= L - M$	
Product rule	$\lim\limits_{x\to k}(f(x)\cdot g(x))$	$= L \cdot M$	
Quotient rule	$\lim\limits_{x\to k}(\frac{f(x)}{g(x)})$	$= \frac{L}{M},$	$M \neq 0$
Root rule	$\lim\limits_{x\to k}(\sqrt[n]{f(x)})$	$= \sqrt[n]{L},$	$n > 0,$ and if n is even then $L > 0$
Power rule	$\lim\limits_{x\to k}((f(x))^n)$	$= L^n,$	$n > 0$

Figure 6.1: Rules for evaluating limits

Examples of using rules to evaluate limits

Example 1

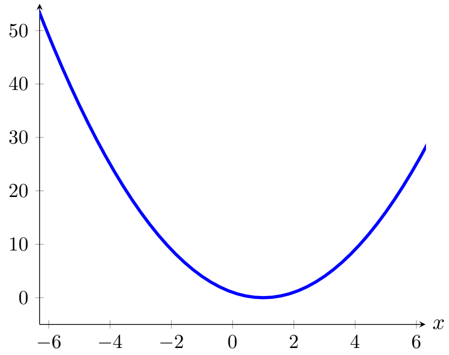


Evaluate $\lim\limits_{x\to 2} x^3 + 3x + 2$

We can use $\lim\limits_{x\to 2} x = 2$

$$\begin{aligned}\lim_{x\to 2} x^3 + 3x + 2 &= \lim_{x\to 2} x^3 + \lim_{x\to 2} 3x + 2 \quad \text{(Sum rule)}\\ &= 2^3 + 3 \times 2 + 2 \quad \text{(Power+const multiple)}\\ &= 8 + 6 + 2\\ &= 16\end{aligned}$$

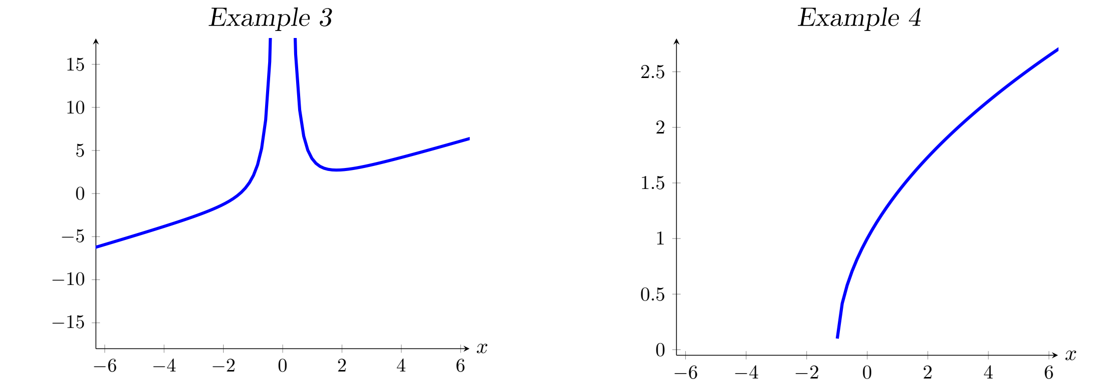
Example 2



Evaluate $\lim\limits_{x\to 3} x^2 - 2x + 1$

We can use $\lim\limits_{x\to 3} x = 2$

$$\begin{aligned}\lim_{x\to 3} x^2 - 2x + 1 &= \lim_{x\to 3} x^2 - \lim_{x\to 3} 2x + 1 \quad \text{(Difference rule)}\\ &= 3^2 - 2 \times 3 + 2 \quad \text{(Power+const multiple)}\\ &= 9 - 6 + 1\\ &= 4\end{aligned}$$



Evaluate $\lim\limits_{x\to 1} \dfrac{x^3+3}{x^2}$

We can use $\lim\limits_{x\to 1} x = 1$

$$\begin{aligned}\lim_{x\to 1} \frac{x^3+3}{x^2} &= \frac{\lim_{x\to 1} x^3+3}{\lim_{x\to 1} x^2} && \text{(Quotient rule)}\\ &= \frac{1^3+3}{1^2} && \text{(Power rule)}\\ &= 4\end{aligned}$$

Evaluate $\lim\limits_{x\to 0} \sqrt{x+1}$

We can use $\lim\limits_{x\to 0} x = 0$

$$\begin{aligned}\lim_{x\to 0} \sqrt{x+1} &= \sqrt{\lim_{x\to 0}(x+1)} && \text{(Root rule)}\\ &= \sqrt{\lim_{x\to 0} x+1} && \text{(Sum rule)}\\ &= 1\end{aligned}$$

Figure 6.3: Examples of evaluating limits using simple rules

Here are a few examples that use the basic rules to evaluate a limit. Note that these rules apply to functions which are defined at a point as x approaches that point.

6.2 Limits for polynomials

Examples 1 and 2 are that of polynomials. From the rules for limits, we can see that for any polynomial, the limit of the polynomial when x approaches a point k is equal to the value of the polynomial at k. It can be written as:

$$\begin{aligned}\lim_{x\to k} P(x) &= \lim_{x\to k} a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0\\ &= a_n k^n + a_{n-1}k^{n-1} + \cdots + a_1 k + a_0\\ &= P(k)\end{aligned}$$

Hence, we can evaluate the limit of a polynomial via direct substitution, e.g.

$$\lim_{x\to 1} x^4 + 3x^3 + 2 = 1^4 + 3(1)^3 + 2 = 6$$

6.3 Limits for rational functions

For rational functions that involve fractions, there are two cases. One case is evaluating the limit when x approaches a point and the function is defined at that point. The other case involves computing the limit when x approaches a point and the function is undefined at that point.

Case 1: Function is defined

Similar to the case of polynomials, whenever we have a function, which is a rational expression of the form $f(x)/g(x)$ and the denominator is non-zero at a point then:

$$\lim_{x\to k} \frac{f(x)}{g(x)} = \frac{f(k)}{g(k)} \quad \text{if } g(k) \neq 0$$

We can therefore evaluate this limit via direct substitution. For example:

$$\lim_{x\to 0} \frac{x^2+1}{x-1} = -1$$

Here, we can apply the quotient rule or easier still, substitute $x = 0$ to evaluate the limit. However, this function has no limit when x approaches 1. See the first graph in the figure below.

Case 2: Function is undefined

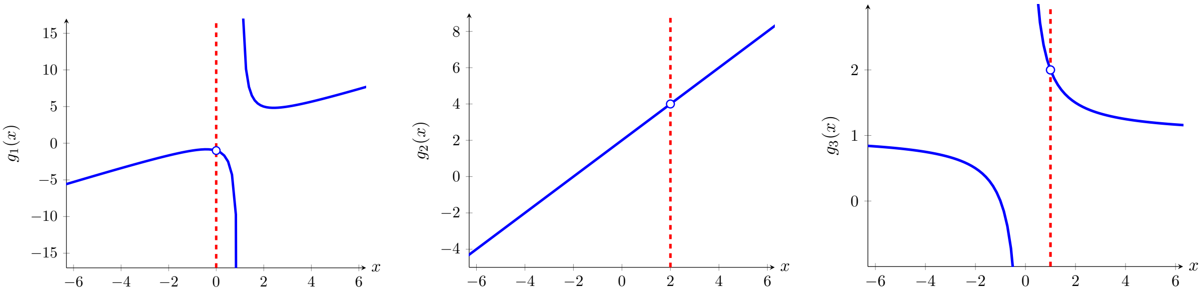
Let's look at another example:

$$\lim_{x\to 2} \frac{x^2-4}{x-2}$$

At $x = 2$ we are faced with a problem. The denominator is zero, and hence the function is undefined at $x = 2$. We can see from the figure that the graph of this function and $(x+2)$ is the same, except at the point $x = 2$, where there is a hole. In this case, we can cancel out the common factors and still evaluate the limit for $(x \to 2)$ as:

$$\lim_{x\to 2} \frac{x^2-4}{x-2} = \lim_{x\to 2} \frac{(x-2)(x+2)}{x-2} = \lim_{x\to 2}(x+2) = 4$$

Following image shows the above two examples as well as a third similar example of $g_3(x)$:



$$g_1(x) = \frac{x^2+1}{x-1}$$

$$\lim_{x\to 0} \frac{x^2+1}{x-1} = -1$$

(no limit when $x \to 1$)

$$g_2(x) = \frac{x^2-4}{x-2}$$

$$\begin{aligned}\lim_{x\to 2} \frac{x^2-4}{x-2} &= \lim_{x\to 2} \frac{(x+2)(x-2)}{x-2}\\ &= \lim_{x\to 2}(x+2)\\ &= 4\end{aligned}$$

$$g_3(x) = \frac{x^2-1}{x^2-x}$$

$$\begin{aligned}\lim_{x\to 1} \frac{x^2-1}{x^2-x} &= \lim_{x\to 1} \frac{(x-1)(x+1)}{x(x-1)}\\ &= \lim_{x\to 1} \frac{x+1}{x}\\ &= 2\end{aligned}$$

(no limit when $x \to 0$)

Figure 6.4: Examples of computing limits for rational functions

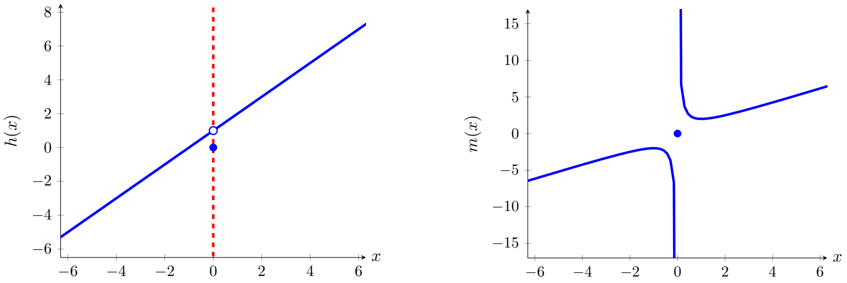
6.4 Case for functions with a discontinuity

Suppose we have a function h(x), which is defined for all real numbers:

$$\begin{aligned} h(x) &= \frac{x^2+x}{x}, && \text{if } x \neq 0\\ h(x) &= 0, && \text{if } x = 0 \end{aligned}$$

The function $g(x)$, has a discontinuity at $x = 0$, as shown in the figure below. When evaluating $\lim\limits_{x\to 0} h(x)$, we have to see what happens to $h(x)$ when x is close to 0 (and not when $x = 0$). As we approach $x = 0$ from either side, $h(x)$ approaches 1, and hence $\lim\limits_{x\to 0} h(x) = 1$.

The function $m(x)$ shown in the figure below is another interesting case. This function is also defined for all real numbers but the limit does not exist when $x \to 0$.



$$h(x) = \begin{cases} \frac{x^2+x}{x}, & \text{if } x \neq 0 \\ 0, & \text{if } x = 0 \end{cases}$$

$$m(x) = \begin{cases} \frac{x^2+1}{x}, & \text{if } x \neq 0 \\ 0, & \text{if } x = 0 \end{cases}$$

$$\lim_{x\to 0} h(x) = 1$$

(no limit when $x \to 0$)

Figure 6.5: Evaluating limits when there is a discontinuity

6.5 The sandwich theorem

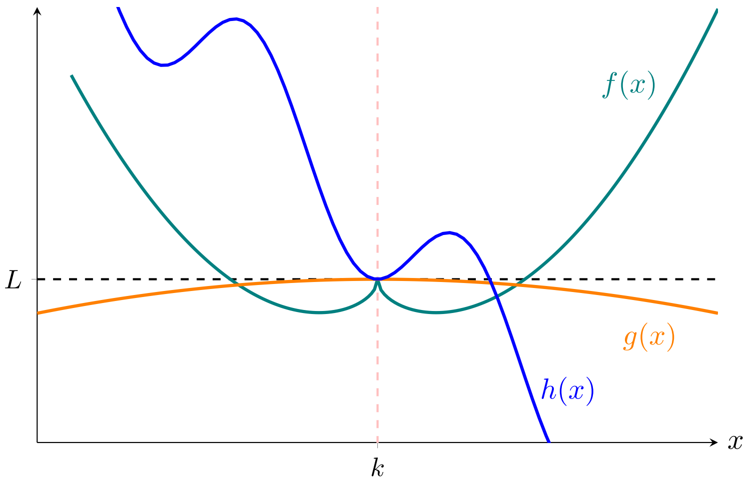
This theorem is also called the *squeeze theorem* or the *pinching theorem*. It states that when the following are true:

1. x is close to k
2. $f(x) \leq g(x) \leq h(x)$
3. $\lim_{x\to k} f(x) = \lim_{x\to k} h(x) = L$

then the limit of $g(x)$ as x approaches k is given by:

$$\lim_{x\to k} g(x) = L$$

This theorem is illustrated in the figure below:



As $x \to k$ and

$$f(x) \le g(x) \le h(x) \qquad \text{then} \qquad \lim_{x\to k} g(x) = L$$

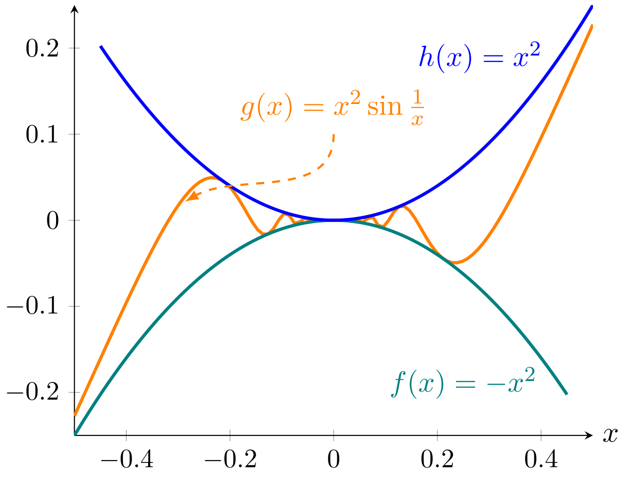
$$\lim_{x\to k} f(x) = \lim_{x\to k} h(x) = L$$

Figure 6.6: The sandwich theorem

Using this theorem we can evaluate the limits of many complex functions. A well known example involves the sine function:

$$\lim_{x\to 0} x^2 \sin\frac{1}{x}$$

We know that the $\sin(x)$ always alternates between -1 and $+1$. Using this fact, we can solve this limit as shown below:



As the following is true:

$$-1 \le \sin\frac{1}{x} \le +1$$

$$-x^2 \le x^2 \sin\frac{1}{x} \le +x^2$$

$$\lim_{x\to 0}(-x^2) = \lim_{x\to 0}(+x^2) = 0$$

From sandwich theorem:

$$\lim_{x\to 0} x^2 \sin\frac{1}{x} = 0$$

Figure 6.7: Computing limits using sandwich theorem

6.6 Evaluating limits with Python

In case you want to explore how to find the limit on some functions, you may try out some computer algebraic system (CAS). In Python, the SymPy library may help. For example, if you want to evaluate the limit of Example 4 above:

```python
from sympy import limit, sqrt, pprint
from sympy.abc import x

expression = sqrt(x+1)
result = limit(expression, x, -1)
print(”Limit of”)
pprint(expression)
print(”at x=-1 is”, result)
```

Program 6.1: Evaluating limit as in Example 4

its output would match that we evaluated earlier:

```
Limit of
  _______
╲╱ x + 1
at x=-1 is 0
```

Output 6.1: Result in Example 4

Note that in the code above, the variable `x` must be defined as a symbol in SymPy. Hence, we use the one imported from `sympy.abc` for this purpose. The limit is evaluated algebraically rather than solely numerical, hence its result should be accurate. We can see that it can evaluate for the point that the function is undefined, too, such as the case we see in the previous section:

$$\lim_{x \to 0} x^2 \sin \frac{1}{x}$$

```python
from sympy import limit, sin, pprint
from sympy.abc import x

expression = x**2 * sin(1/x)
result = limit(expression, x, 0)
print(”Limit of”)
pprint(expression)
print(”at x=0 is”, result)
```

Program 6.2: Evaluate the limit from the example of sandwich theorem

```
Limit of
 2    ⎛1⎞
x ⋅sin⎜─⎟
      ⎝x⎠
at x=0 is 0
```

Output 6.2: The limit from the example of sandwich theorem

6.7 Further reading

The best way to learn and understand mathematics is via practice, and solving more problems. This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

6.8 Summary

In this tutorial, you discovered how limits for different types of functions can be evaluated. Specifically, you learned:

- Rules for evaluating limits for different functions.
- Evaluating limits of polynomials and rational functions
- Evaluating limits when discontinuities are present in a function

In the next chapter, we will connect the limit to *differential calculus*.

Function Derivatives

The concept of the derivative is the building block of many topics of calculus. It is important for understanding integrals, gradients, Hessians, and much more.

In this tutorial, you will discover the definition of a derivative, its notation and how you can compute the derivative based upon this definition. You will also discover why the derivative of a function is also a function.

After completing this tutorial, you will know:

- ▷ The definition of the derivative of a function
- ▷ How to compute the derivative of a function based upon the definition
- ▷ Why some functions do not have a derivative at a point

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ The definition and notation used for derivatives of functions
- ▷ How to compute the derivative of a function using the definition
- ▷ Why some functions do not have a derivative at a point

7.1 What is the derivative of a function

In very simple words, the derivative of a function $f(x)$ represents its rate of change and is denoted by either $f'(x)$ or df/dx. Let's first look at its definition and a pictorial illustration of the derivative.

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}$$

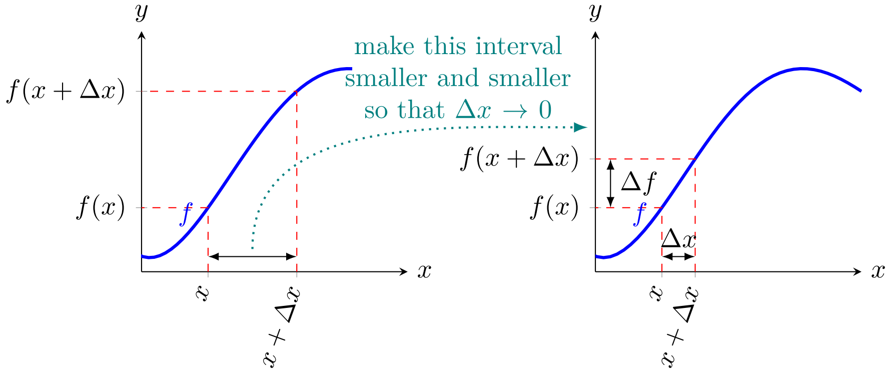


Figure 7.1: Derivative of f is the rate of change of f

In the figure, Δx represents a change in the value of x. We keep making the interval between x and $(x + \Delta x)$ smaller and smaller until it is infinitesimal. Hence, we have the limit $\Delta \to 0$. The numerator $f(x + \Delta x) - f(x)$ represents the corresponding change in the value of the function f over the interval Δx. This makes the derivative of a function f at a point x, the rate of change of f at that point.

An important point to note is that Δx, the change in x can be negative or positive. Hence:

$$0 < |\Delta x| < \epsilon,$$

where ϵ is an infinitesimally small value.

About the notation

The derivative of a function can be denoted by both $f'(x)$ and df/dx. The mathematical giant Newton used $f'(x)$ to denote the derivative of a function. Leibniz, another mathematical hero, used df/dx. So df/dx is a single term, not to be confused with a fraction. It is read as the derivative of a function f with respect to x, and also indicates that x is the independent variable.

Connection with velocity

One of the most commonly cited examples of derivatives is that of velocity. Velocity is the rate of change of distance with respect to time. Hence if $f(t)$ represents the distance traveled at time t, then $f'(t)$ is the velocity at time t. The following sections show various examples of computing the derivative.

7.2 Differentiation examples

The method of finding the derivative of a function is called differentiation. In this section, we'll see how the definition of the derivative can be used to find the derivative of different functions. Later on, once you are more comfortable with the definition, you can use the defined rules to differentiate a function.

Example 1: $m(x) = 2x + 5$

Let's start with a simple example of a linear function $m(x) = 2x + 5$. We can see that $m(x)$ changes at a constant rate. We can differentiate this function as follows.

$$\begin{aligned} m(x) &= 2x + 5 \\ m(x + \Delta x) &= 2(x + \Delta x) + 5 \\ m'(x) &= \lim_{\Delta x \to 0} \frac{m(x + \Delta x) - m(x)}{\Delta x} \\ &= \lim_{\Delta x \to 0} \frac{2x + 2\Delta x + 5 - 2x - 5}{\Delta x} \\ &= \lim_{\Delta x \to 0} \frac{2\Delta x}{\Delta x} \\ &= 2 \end{aligned}$$

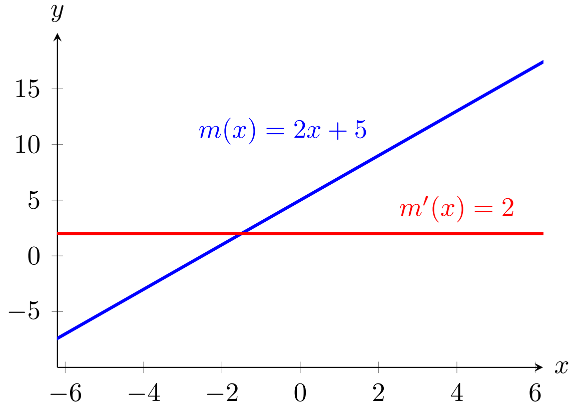


Figure 7.2: Derivative of $m(x) = 2x + 5$. The rate of change of a linear function is a constant

The above figure shows how the function $m(x)$ is changing and it also shows that no matter which value of x we choose the rate of change of $m(x)$ always remains a 2. We can verify the above result with SymPy:

```
from sympy import diff, sqrt, pprint
from sympy.abc import x

expression = 2*x + 5
result = diff(expression, x)
print("Derivative of")
pprint(expression)
```

```
print("with respect to x is")
pprint(result)
```

Program 7.1: Find derivative of $m(x) = 2x + 5$

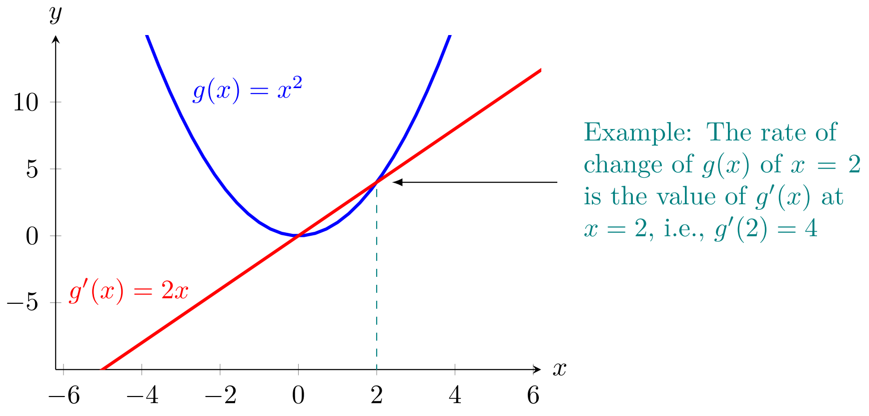
```
Derivative of
2⋅x + 5
with respect to x is
2
```

Output 7.1: Derivative of $m(x) = 2x + 5$

Example 2: $g(x) = x^2$

Suppose we have the function $g(x)$ given by: $g(x) = x^2$. The figure below shows how the derivative of $g(x)$ with respect to x is calculated. There is also a plot of the function and its derivative in the figure.

$$
\begin{aligned}
g(x) &= x^2 \\
g(x + \Delta x) &= (x + \Delta x)^2 \\
&= x^2 + 2x\Delta x + (\Delta x)^2 \\
g'(x) &= \lim_{\Delta x \to 0} \frac{g(x + \Delta x) - g(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + (\Delta x)^2 - x^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{2x\Delta x + \Delta x^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} (2x + \Delta x) \\
&= 2x
\end{aligned}
$$



Example: The rate of change of $g(x)$ of $x = 2$ is the value of $g'(x)$ at $x = 2$, i.e., $g'(2) = 4$

Figure 7.3: Derivative of $g(x) = x^2$: The rate of change is positive when $x > 0$, negative when $x < 0$ and 0 when $x = 0$

As $g'(x) = 2x$, hence $g'(0) = 0$, $g'(1) = 2$, $g'(2) = 4$ and $g'(-1) = -2$, $g'(-2) = -4$

From the figure, we can see that the value of $g(x)$ is very large for large negative values of x. When $x < 0$, increasing x decreases $g(x)$ and hence $g'(x) < 0$ for $x < 0$. The graph flattens out for $x = 0$, where the derivative or rate of change of $g(x)$ becomes zero. When $x > 0$, $g(x)$ increases quadratically with the increase in x, and hence, the derivative is also positive. We can verify the above result with SymPy:

```
from sympy import diff, sqrt, pprint
from sympy.abc import x

expression = x**2
result = diff(expression, x)
print(”Derivative of”)
pprint(expression)
print(”with respect to x is”)
pprint(result)
```

Program 7.2: Find derivative of $g(x) = x^2$

```
Derivative of
 2
x
with respect to x is
2⋅x
```

Output 7.2: Derivative of $g(x) = x^2$

Example 3: $h(x) = 1/x$

Suppose we have the function $h(x) = 1/x$. Shown below is the differentiation of $h(x)$ with respect to x (for $x \neq 0$) and the figure illustrating the derivative. The blue curve denotes h(x) and the red curve its corresponding derivative.

$$\begin{aligned}
h(x) &= \frac{1}{x} \\
h(x+\Delta x) &= \frac{1}{x+\Delta x} \\
h'(x) &= \lim_{\Delta x \to 0} \frac{h(x+\Delta x) - h(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{\frac{1}{x+\Delta x} - \frac{1}{x}}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{-1}{x(x+\Delta x)} \\
&= \frac{-1}{x^2}
\end{aligned}$$

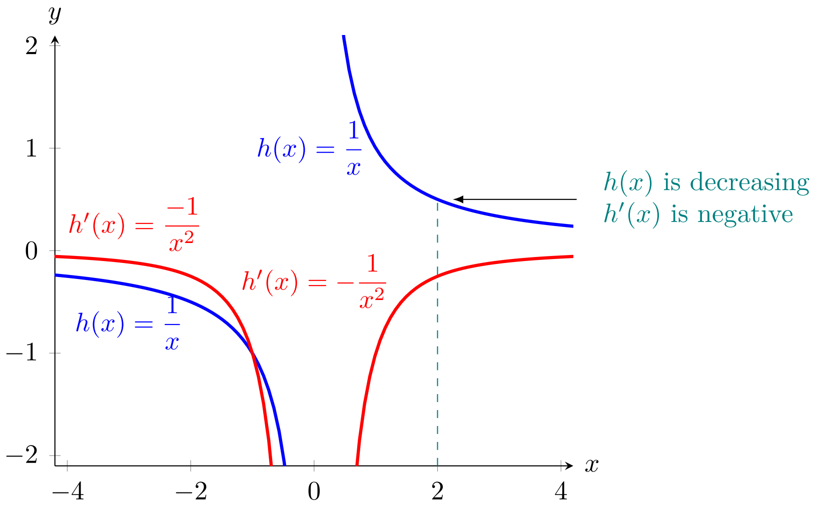


Figure 7.4: Derivative of $h(x) = 1/x$

We can verify the above result with SymPy:

```python
from sympy import diff, sqrt, pprint
from sympy.abc import x

expression = 1/x
result = diff(expression, x)
print(”Derivative of”)
pprint(expression)
print(”with respect to x is”)
pprint(result)
```

Program 7.3: Find derivative of $h(x) = 1/x$

```
Derivative of
1
─
x
with respect to x is
-1
───
  2
 x
```

Output 7.3: Derivative of $h(x) = 1/x$

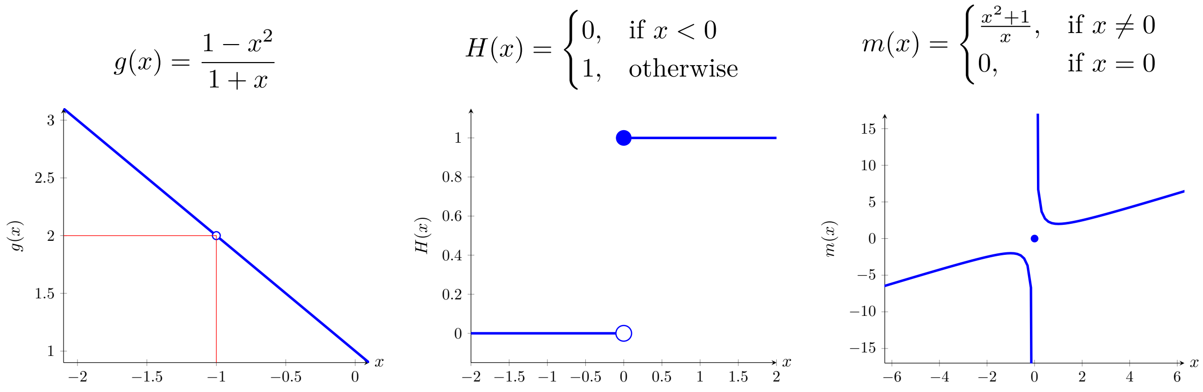
7.3 Differentiability and continuity

In example 3, the function $h(x) = 1/x$ is undefined at the point $x = 0$. Hence, its derivative $-1/x^2$ is also not defined at $x = 0$. If a function is not continuous at a point, then it does not have a derivative at that point. Below are a few scenarios, where a function is not differentiable:

1. The function is not defined at a point

2. The function does not have a limit at that point
3. The function is not continuous (e.g., has a sudden jump) at a point

Following are a few examples:



$g(x)$ has no derivative at $x = -1$ *$H(x)$ has no derivative at $x = 0$* *$m(x)$ has no derivative at $x = 0$*

Figure 7.5: Examples of points at which there is no derivative

7.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

7.5 Summary

In this tutorial, you discovered the function derivatives and the fundamentals of function differentiation. Specifically, you learned:

- ▷ The definition and notation of a function derivative
- ▷ How to differentiate a function using the definition
- ▷ When a function is not differentiable

In the next chapter, we will explore more properties related to the continuity of a function.

8 Continuous Functions

Many areas of calculus require an understanding of continuous functions. The characteristics of continuous functions, and the study of points of discontinuity are of great interest to the mathematical community. Because of their important properties, continuous functions have practical applications in machine learning algorithms and optimization methods.

In this tutorial, you will discover what continuous functions are , their properties, and two important theorems in the study of optimization algorithms, i.e., intermediate value theorem and extreme value theorem.

After completing this tutorial, you will know:

- Definition of continuous functions
- Intermediate value theorem
- Extreme value theorem

Let's get started.

Overview

This tutorial is divided into 2 parts; they are:

- Definition of continuous functions
  - Informal definition
  - Formal definition
- Theorems
  - Intermediate value theorem
  - Extreme value theorem

8.1 Prerequisites

This tutorial requires an understanding of the concept of limits. To refresh your memory, you can take a look at limits and continuity (Chapter 5), where continuous functions are also briefly defined. In this tutorial we'll go into more details.

We'll also make use of intervals. So square brackets mean closed intervals (include the boundary points) and parenthesis mean open intervals (do not include boundary points), for example,

▷ $[a, b]$ means $a \leq x \leq b$

▷ (a, b) means $a < x < b$

▷ $[a, b)$ means $a \leq x < b$

From the above, you can note that an interval can be open on one side and closed on the other.

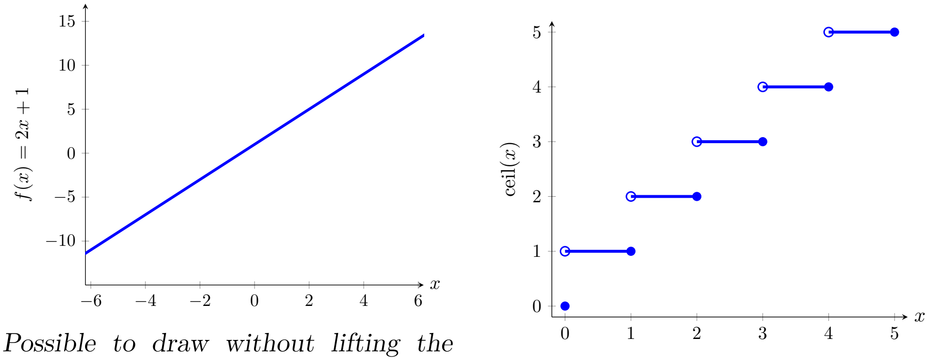
As a last point, we'll only be discussing real functions defined over real numbers. We won't be discussing complex numbers or functions defined on the complex plane.

8.2 An informal definition of continuous functions

Suppose we have a function $f(x)$. We can easily check if it is continuous between two points a and b, if we can plot the graph of $f(x)$ without lifting our hand. As an example, consider a straight line defined as:

$$f(x) = 2x + 1$$

We can draw the straight line between $[0, 1]$ without lifting our hand. In fact, we can draw this line between any two values of x and we won't have to lift our hand (see figure below). Hence, this function is continuous over the entire domain of real numbers. Now let's see what happens when we plot the ceil function:



Possible to draw without lifting the hand

Cannot draw without lifting the hand

Figure 8.1: Continuous function (left), and not a continuous function (right)

The ceil function has a value of 1 on the interval $(0, 1]$, for example, ceil$(0.5) = 1$, ceil$(0.7) = 1$, and so on. As a result, the function is continuous over the domain $(0, 1]$. If we adjust the interval to $(0, 2]$, ceil(x) jumps to 2 as soon as $x > 1$. To plot ceil(x) for the domain $(0, 2]$, we must now lift our hand and start plotting again at $x = 2$. As a result, the ceil function isn't a continuous function.

If the function is continuous over the entire domain of real numbers, then it is a continuous function as a whole, otherwise, it is not continuous as whole. For the later type of functions, we can check over which interval they are continuous.

8.3 A formal definition

A function $f(x)$ is continuous at a point a, if the function's value approaches $f(a)$ when x approaches a. Hence to test the continuity of a function at a point $x = a$, check the following:

1. $f(a)$ should exist
2. $f(x)$ has a limit as x approaches a
3. The limit of $f(x)$ as $x \to a$ is equal to $f(a)$

If all of the above hold true, then the function is continuous at the point a.

8.4 Examples

Some examples are listed below and also shown in the figure:

▷ $f(x) = 1/x$ is not continuous as it is not defined at $x = 0$. However, the function is continuous for the domain $x > 0$.

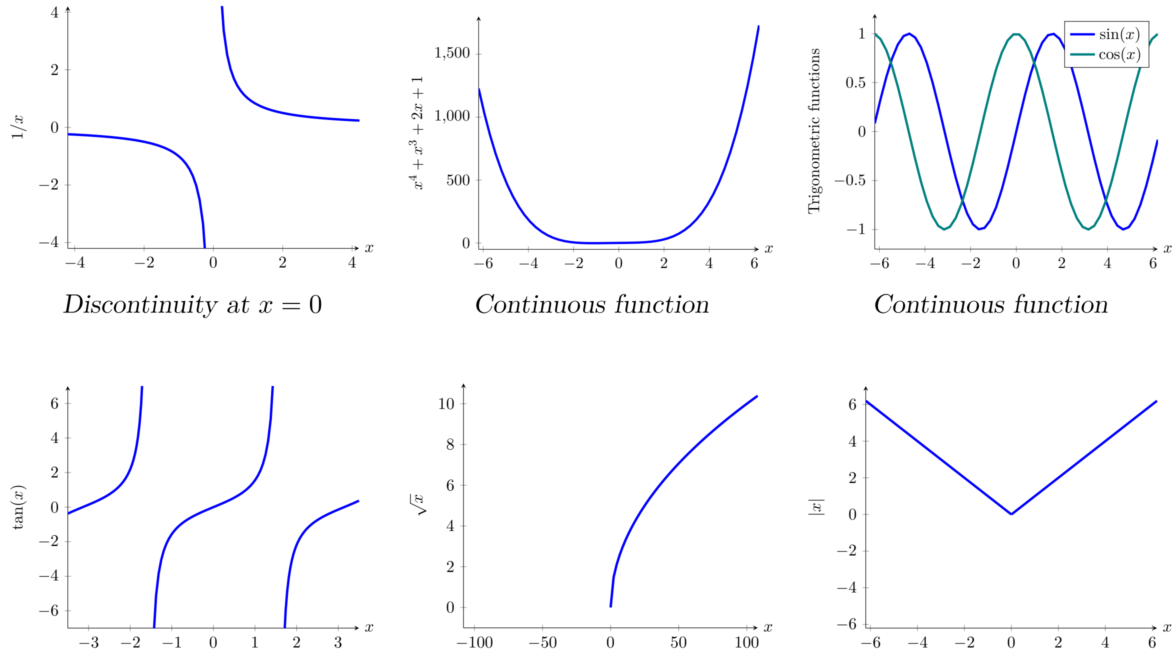
▷ All polynomial functions are continuous functions.

▷ The trigonometric functions $\sin(x)$ and $\cos(x)$ are continuous and oscillate between the values -1 and 1.

▷ The trigonometric function $\tan(x)$ is not continuous as it is undefined at $x = \pi/2$, $x = -\pi/2$, etc.

▷ $\sqrt{x}$ is not continuous as it is not defined for $x < 0$.

▷ $|x|$ is continuous everywhere.



Discontinuity at $x = 0$

Continuous function

Continuous function

Discontinuity at $x = \pi/2$ and $x = -\pi/2$

Not defined for $x < 0$

Continuous function

Figure 8.2: Examples of continuous functions and functions with discontinuities

8.5 Connection of continuity with function derivatives

From the definition of continuity in terms of limits, we have an alternative definition. $f(x)$ is continuous at x, if:

$$f(x+h) - f(x) \to 0 \quad \text{when} \quad h \to 0$$

Let's look at the definition of a derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Hence, if $f'(x)$ exists at a point a, then the function is continuous at a. The converse is not always true. A function may be continuous at a point a, but $f'(a)$ may not exist. For example, in the above graph $|x|$ is continuous everywhere. We can draw it without lifting our hand, however, at $x = 0$ its derivative does not exist because of the sharp turn in the curve.

8.6 Intermediate value theorem

The intermediate value theorem states that, if:

- ▷ function $f(x)$ is continuous on $[a, b]$
- ▷ and $f(a) \leq K \leq f(b)$

then:

- ▷ There is a point c between a and b, i.e., $a \leq c \leq b$ such that $f(c) = K$

In very easy words, this theorem says that if a function is continuous over $[a, b]$, then all values of the function between $f(a)$ and $f(b)$ will exist within this interval as shown in the figure below.

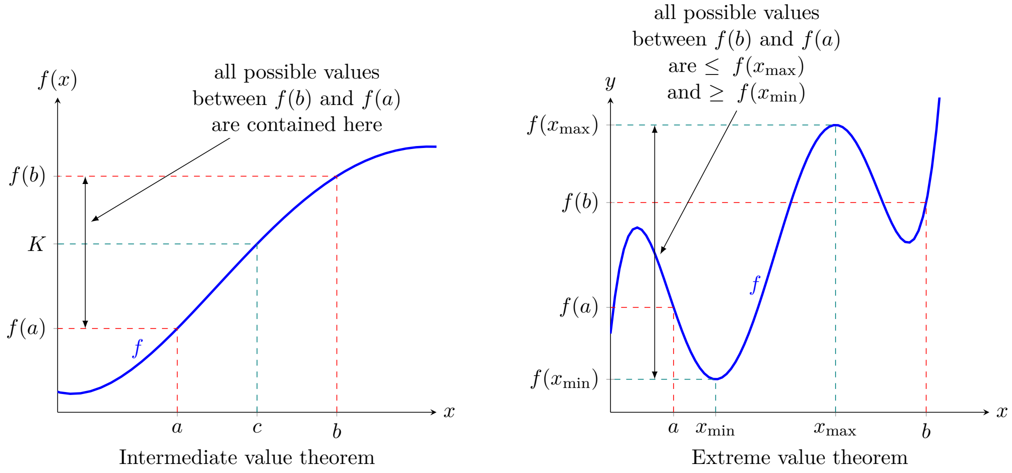


Figure 8.3: Illustration of intermediate value theorem (left) and extreme value theorem (right)

8.7 Extreme value theorem

This theorem states that, if:

- ▷ function $f(x)$ is continuous on $[a, b]$

then:

- ▷ There are points $x_{\min}$ and $x_{\max}$ inside the interval $[a, b]$, i.e.,
  - ○ $a \leq x_{\min} \leq b$
  - ○ $a \leq x_{\max} \leq b$
- ▷ and the function $f(x)$ has a minimum value $f(x_{\min})$, and a maximum value $f(x_{\max})$, i.e.,
  - ○ $f(x_{\min}) \leq f(x) \leq f(x_{\max})$ when $a \leq x \leq b$

In simple words a continuous function always has a minimum and maximum value within an interval as shown in the above figure.

8.8 Continuous functions and optimization

Continuous functions are very important in the study of optimization problems. We can see that the extreme value theorem guarantees that within an interval, there will always be a point where the function has a maximum value. The same can be said for a minimum value.

Many optimization algorithms are derived from this fundamental property and can perform amazing tasks.

8.9 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

8.10 Summary

In this tutorial, you discovered the concept of continuous functions. Specifically, you learned:

- What are continuous functions
- The formal and informal definitions of continuous functions
- Points of discontinuity
- Intermediate value theorem
- Extreme value theorem
- Why continuous functions are important

While differentiation is ultimately about evaluating the limits, we will see in next chapter that by we can find the derivative of a function easier and faster by remembering a few rules.

Derivatives of Powers and Polynomials 9

One of the most frequently used functions in machine learning and data science algorithms are polynomials or functions involving powers of x. It is therefore, important to understand how the derivatives of such functions are calculated.

In this tutorial, you will discover how to compute the derivative of powers of x and polynomials. After completing this tutorial, you will know:

- General rule for computing the derivative of polynomials
- General rule for finding the derivative of a function that involves any non-zero real powers of x

Let's get started.

Overview

This tutorial is divided into two parts; they are:

1. The derivative of a function that involve integer powers of x
2. Differentiation of a function that has any real non-zero power of x

9.1 Derivative of the sum of two functions

Let's start by finding a simple rule that governs the sum of two functions. Suppose we have two functions $f(x)$ and $g(x)$, then the derivative of their sum can be found as follows:

$$\begin{aligned}
m(x) &= f(x) + g(x) \\
m(x+h) &= f(x+h) + g(x+h) \\
m'(x) &= \lim_{h \to 0} \frac{m(x+h) - m(x)}{h} \\
&= \lim_{h \to 0} \frac{f(x+h) + g(x+h) - f(x) - g(x)}{h} \\
&= \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} + \lim_{h \to 0} \frac{g(x+h) - g(x)}{h} \\
&= f'(x) + g'(x)
\end{aligned}$$

Here we have a general rule that says that the derivative of the sum of two functions is the sum of the derivatives of the individual functions.

9.2 Derivative of integer powers of x

Before we talk about derivatives of integer powers of x, let's review the binomial theorem, which tells us how to expand the following expression (here $\binom{n}{k}$ is the choose function):

$$(a+b)^n = a^n + \binom{n}{1} a^{n-1} b + \binom{n}{2} a^{n-2} b^2 + \ldots + \binom{n}{n-1} a b^{n-1} + b^n$$

We'll derive a simple rule for finding the derivative of a function that involves x^n, where n is an integer and $n > 0$. Let's go back to the definition of a derivative and apply it to kx^n, where k is a constant.

$$\begin{aligned}
f(x) &= kx^n \\
f(x+h) &= k(x+h)^n \\
&= k(x^n + \binom{n}{1} x^{n-1} h + \binom{n}{2} x^{n-2} h^2 + \cdots + \binom{n}{n-1} x h^{n-1} + h^n)
\end{aligned}$$

Then the derivative is

$$\begin{aligned}
f'(x) &= \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \\
&= \lim_{h \to 0} \frac{k(x^n + \binom{n}{1} x^{n-1} h + \binom{n}{2} x^{n-2} h^2 + \cdots + \binom{n}{n-1} x h^{n-1} + h^n) - kx^n}{h} \\
&= \lim_{h \to 0} k \left(\binom{n}{1} x^{n-1} + \binom{n}{2} x^{n-2} h + \cdots + \binom{n}{n-1} x h^{n-2} + h^{n-1} \right) \\
&= knx^{n-1}
\end{aligned}$$

Following are some examples of applying this rule, and we can verify them with SymPy:

- ▷ Derivative of x^2 is $2x$
- ▷ Derivative of $3x^5$ is $15x^4$
- ▷ Derivative of $4x^9$ is $36x^8$

```python
from sympy import diff, pprint
from sympy.abc import x

expressions = [x**2, 3*x**5, 4*x**9]
for expression in expressions:
    result = diff(expression, x)
    print(”Derivative of”)
    pprint(expression)
    print(”with respect to x is”)
    pprint(result)
    print()
```

Program 9.1: Verifying derivative of some examples

```
Derivative of
 2
x
with respect to x is
2·x

Derivative of
   5
3·x
with respect to x is
    4
15·x

Derivative of
   9
4·x
with respect to x is
    8
36·x
```

Output 9.1: Derivative of some examples

9.3 How to differentiate a polynomial?

The two rules, i.e., the rule for the derivative of the sum of two functions, and the rule for the derivative of an integer power of x, enable us to differentiating a polynomial. If we have a polynomial of degree n, we can consider it as a sum of individual functions that involve different powers of x. Suppose we have a polynomial $P(x)$ of degree n, then its derivative is given by $P'(x)$ as:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
$$P'(x) = a_n n x^{n-1} + a_{n-1}(n-1) x^{n-2} + \cdots + a_1$$

This shows that the derivative of the polynomial of degree n, is in fact a polynomial of degree $(n-1)$.

9.4 Examples

Some examples are shown below, where the polynomial function and its derivatives are all plotted together. The blue curve shows the function itself, while the red curve is the derivative of that function.

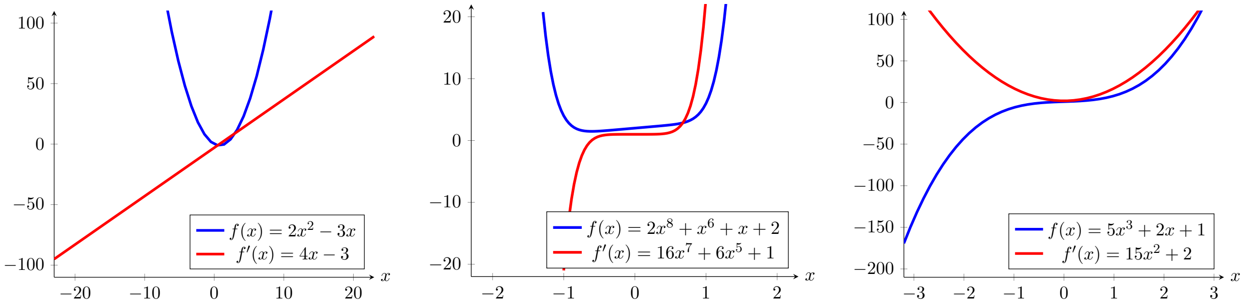


Figure 9.1: Examples of polynomial functions and their derivatives

9.5 What about non-integer powers of x?

The rules derived above extend to non-integer real powers of x, which can be fractions, negative numbers or irrational numbers. The general rule is given below, where a and k can be any real numbers not equal to zero.

$$f(x) = kx^a$$
$$f'(x) = kax^{a-1}$$

A few examples are:

- Derivative of $x^{0.2}$ is $(0.2)x^{-0.8}$
- Derivative of x^{π} is $\pi x^{\pi-1}$
- Derivative of $x^{-\frac{3}{4}}$ is $-\frac{3}{4}x^{-\frac{7}{4}}$

Here are a few examples, which are plotted along with their derivatives. Again, the blue curve denotes the function itself, and the red curve denotes the corresponding derivative:

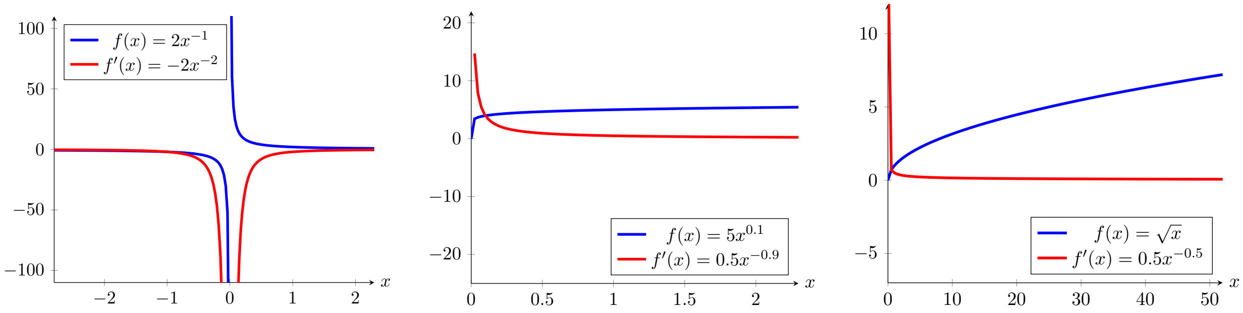


Figure 9.2: Examples of derivatives of expressions involving real powers of x

We can verify these results with SymPy. But to prevent the fraction $-\frac{3}{4}$ from being converted to floating point numbers so we can match our expressions above, we provided these functions in string and ask SymPy to parse it:

```
from sympy import diff, pprint, powsimp, simplify
from sympy.abc import x

expressions = [”k*x**a”, ”x**0.2”, ”x**pi”, ”x**(-3/4)”]
for expression in expressions:
    expression = simplify(expression)
    result = diff(expression, x)
    print(”Derivative of”)
    pprint(expression)
    print(”with respect to x is”)
    pprint(powsimp(result))
    print()
```

Program 9.2: Finding derivatives of real powers of x

```
Derivative of
   a
k⋅x
with respect to x is
     a - 1
a⋅k⋅x

Derivative of
 0.2
x
with respect to x is
     -0.8
0.2⋅x

Derivative of
 π
x
with respect to x is
   -1 + π
π⋅x

Derivative of
```

```
 1
────
 3/4
x
with respect to x is
 -3
──────
   7/4
4⋅x
```

Output 9.2: Derivatives of real powers of x

9.6 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

9.7 Summary

In this tutorial, you discovered how to differentiate a polynomial function and functions involving a sum of non-integer powers of x. Specifically, you learned:

- Derivative of the sum of two functions
- Derivative of a constant multiplied by an integer power of x
- Derivative of a polynomial function
- Derivative of a sum of expressions involving non-integers powers of x

In the next chapter, we will see how we can find derivatives involving not powers of x, but $\sin(x)$ and $\cos(x)$.

10 Derivative of the Sine and Cosine

Many machine learning algorithms involve an optimization process for different purposes. Optimization refers to the problem of minimizing or maximizing an objective function by altering the value of its inputs.

Optimization algorithms rely on the use of derivatives in order to understand how to alter (increase or decrease) the input values to the objective function, in order to minimize or maximize it. It is, therefore, important that the objective function under consideration is *differentiable*.

The two fundamental trigonometric functions, the sine and cosine, offer a good opportunity to understand the maneuvers that might be required in finding the derivatives of differentiable functions. These two functions become especially important if we think of them as the fundamental building blocks of more complex functions.

In this tutorial, you will discover how to find the derivative of the sine and cosine functions.

After completing this tutorial, you will know:

▷ How to find the derivative of the sine and cosine functions by applying several rules from algebra, trigonometry and limits.

▷ How to find the derivative of the sine and cosine functions in Python.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

▷ The Derivative of the Sine Function

▷ The Derivative of the Cosine Function

▷ Finding Derivatives in Python

10.1 The derivative of the sine function

The derivative $f'(x)$ of some function, f, at a particular point, x, may be specified as:

$$f'(x) = \frac{df(x)}{dx} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

We shall start by considering the sine function. Hence, let's first substitute for $f(x) = \sin x$:

$$\sin'(x) = \lim_{h\to 0} \frac{\sin(x+h) - \sin(x)}{h}$$

If we have a look at the trigonometric identities, we find that we may apply the *addition formula* to expand the $\sin(x+h)$ term:

$$\sin(x+y) = \sin x \cos y + \cos x \sin y$$

Indeed, by substituting y with h we can define the derivative of $\sin x$ as:

$$\sin'(x) = \lim_{h\to 0} \frac{\sin x \cos h + \cos x \sin h - \sin x}{h}$$

We may simplify the expression further by applying one of the limit laws, which states that the limit of a sum of functions is equal to the sum of their limits:

$$\sin'(x) = \lim_{h\to 0} \left(\frac{\cos x \sin h}{h}\right) + \lim_{h\to 0} \left(\frac{\sin x \cos h}{h} - \frac{\sin x}{h}\right)$$

We may simplify even further by bringing out any common factor that is a function of x. In this manner, we can factorize the expression to obtain the sum of two separate limits that do not depend on x:

$$\sin'(x) = \lim_{h\to 0} \left(\frac{\sin h}{h}\right) \cos x + \lim_{h\to 0} \left(\frac{\cos h - 1}{h}\right) \sin x$$

Solving each of these two limits will give us the derivative of $\sin x$.

Let's start by tackling the first limit. Recall that we may represent angle, h in radians, on the unit circle. The sine of h would then be given by the perpendicular to the x-axis (BC), at the point that meets the unit circle:

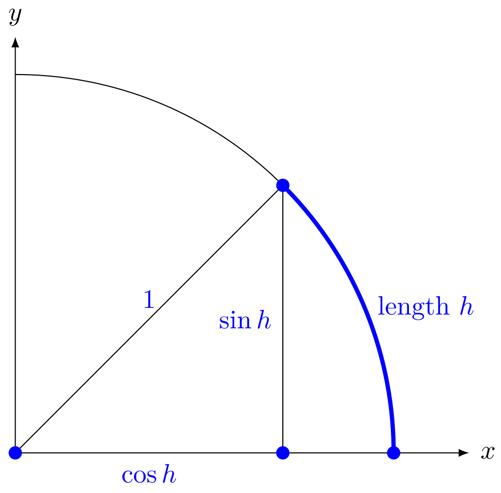


Figure 10.1: Representing angle, h, on the unit circle

We will be comparing the area of different sectors and triangles, with sides subtending the angle h, in an attempt to infer how $\dfrac{\sin h}{h}$ behaves as the value of h approaches zero. For this purpose, consider first the area of sector OAB:

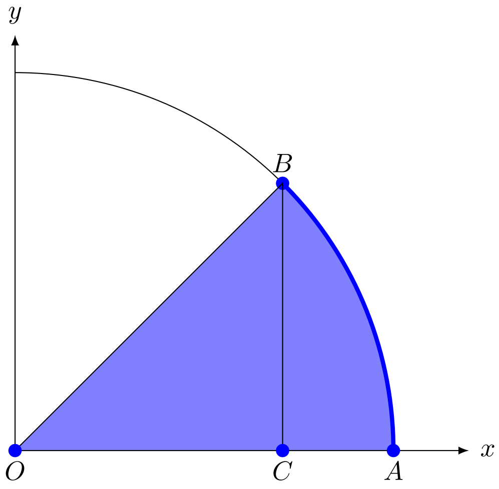


Figure 10.2: Finding the area of sector, OAB

The area of a sector can be defined in terms of the circle radius, r, and the length of the arc AB, h. Since the circle under consideration is the *unit* circle, then $r = 1$:

$$\text{area of sector } OAB = \frac{rh}{2} = \frac{h}{2}$$

We can compare the area of the sector OAB that we have just found, to the area of the *triangle* OAB within the same sector.

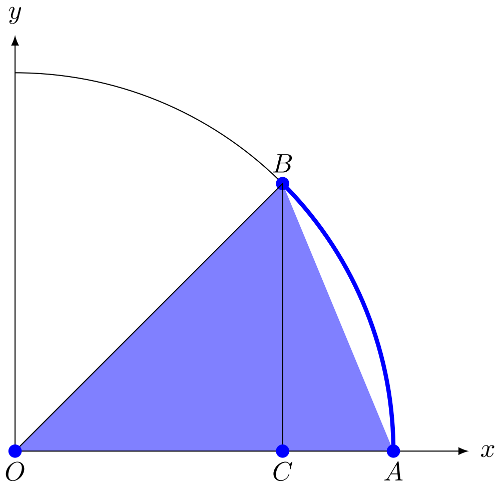


Figure 10.3: Finding the area of triangle, OAB

The area of this triangle is defined in terms of its height, $BC = \sin h$, and the length of its base, $OA = 1$:

$$\text{area of triangle } OAB = \frac{(BC)(OA)}{2} = \frac{\sin h}{2}$$

Since we can clearly see that the area of the triangle, OAB, that we have just considered is smaller that the area of the sector that it is contained within, then we may say that:

$$\frac{\sin h}{2} < \frac{h}{2}$$

$$\frac{\sin h}{h} < 1$$

This is the first piece of information that we have obtained regarding the behavior of $\dfrac{\sin h}{h}$, which tells us that its upper limit value will not exceed 1.

Let us now proceed to consider a second triangle, OAB', that is characterized by a larger area than that of sector, OAB. We can use this triangle to provide us with the second piece of information about the behavior of $\dfrac{\sin h}{h}$, which is its lower limit value:

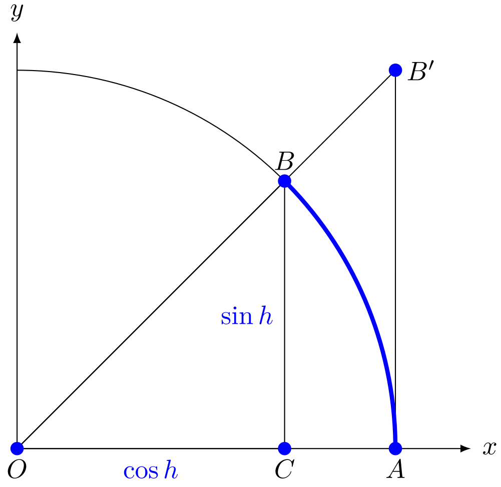


Figure 10.4: Comparing similar triangles, OAB and OAB'

Applying the properties of similar triangles to relate OAB' to OCB, gives us information regarding the length, $B'A$, that we need to compute the area of the triangle:

$$\frac{B'A}{OA} = \frac{BC}{OC} = \frac{\sin h}{\cos h}$$

Hence, the area of triangle OAB' may be computed as:

$$\text{area of triangle } OAB' = \frac{(B'A)(OA)}{2} = \frac{\sin h}{2\cos h}$$

Comparing the area of triangle OAB' to that of sector OAB, we can see that the former is now larger:

$$\frac{h}{2} < \frac{\sin h}{2\cos h}$$

$$\cos h < \frac{\sin h}{h}$$

This is the second piece of information that we needed, which tells us that the lower limit value of $\frac{\sin h}{h}$ does not drop below $\cos h$. We also know that as h approaches 0, the value of $\cos h$ approaches 1.

Hence, putting the two pieces of information together, we find that as h becomes smaller and smaller, the value of $\frac{\sin h}{h}$ itself is *squeezed* to 1 by its lower and upper limits. This is, indeed, referred to as the *squeeze* or *sandwich* theorem.

Let's now proceed to tackle the second limit. By applying standard algebraic rules:

$$\lim_{h\to 0}\frac{\cos h - 1}{h} = \lim_{h\to 0}\frac{\cos h - 1}{h}\cdot\frac{\cos h + 1}{\cos h + 1}$$

We can manipulate the second limit as follows:

$$\lim_{h\to 0}\frac{\cos h - 1}{h} = \lim_{h\to 0}\frac{\cos^2 h - 1}{h(\cos h + 1)}$$

We can then express this limit in terms of sine, by applying the Pythagorean identity from trigonometry, $\sin^2 h = 1 - \cos^2 h$:

$$\lim_{h\to 0}\frac{\cos h - 1}{h} = \lim_{h\to 0}\frac{-\sin^2 h}{h(\cos h + 1)}$$

Followed by the application of another limit law, which states that the limit of a product is equal to the product of the separate limits:

$$\lim_{h\to 0}\frac{\cos h - 1}{h} = \lim_{h\to 0}\frac{\sin h}{h}\cdot\lim_{h\to 0}\frac{-\sin h}{\cos h + 1}$$

We have already tackled the first limit of this product, and we have found that this has a value of 1.

The second limit of this product is characterized by a $\cos h$ in the denominator, which approaches a value of 1 as h becomes smaller. Hence, the denominator of the second limit approaches a value of 2 as h approaches 0. The sine term in the numerator, on the other hand, attains a value of 0 as h approaches 0. This drives not only the second limit, but also the entire product limit to 0:

$$\lim_{h\to 0}\frac{\cos h - 1}{h} = 0$$

Putting everything together, we may finally arrive to the following conclusion:

$$\begin{aligned}\sin'(x) &= \left(\lim_{h\to 0}\frac{\sin h}{h}\right)\cos x + \left(\lim_{h\to 0}\frac{\cos h - 1}{h}\right)\sin x\\ \sin'(x) &= (1)(\cos x) + (0)(\sin x)\\ &= \cos x\end{aligned}$$

This, finally, tells us that the derivative of $\sin x$ is simply $\cos x$.

10.2 The derivative of the cosine function

Similarly, we can calculate the derivative of the cosine function by re-using the knowledge that we have gained in finding the derivative of the sine function. Substituting for $f(x) = \cos x$:

$$\cos'(x) = \lim_{h \to 0} \frac{\cos(x+h) - \cos x}{h}$$

The *addition formula* is now applied to expand the $\cos(x+h)$ term as follows:

$$\cos(x+y) = \cos x \cos y + \sin x \sin y$$

Which again leads to the summation of two limits:

$$\cos'(x) = \left(\lim_{h \to 0} \frac{\sin h}{h} \right) \cdot (-\sin x) + \left(\lim_{h \to 0} \frac{\cos h - 1}{h} \right) \cdot \cos x$$

We can quickly realize that we have already evaluated these two limits in the process of finding the derivative of sine; the first limit approaches 1, whereas the second limit approaches 0, as the value of h become smaller:

$$\begin{aligned} \cos'(x) &= (1)(-\sin x) + (0)(\cos x) \\ &= -\sin x \end{aligned}$$

Which, ultimately, tells us that the derivative of $\cos x$ is conversely $-\sin x$.

The importance of the derivatives that we have just found lies in their definition of the *rate of change* of the function under consideration, at some particular angle, h. For instance, if we had to recall the graph of the periodic sine function, we can observe that its first positive peak coincides with an angle of $\pi/2$ radians.

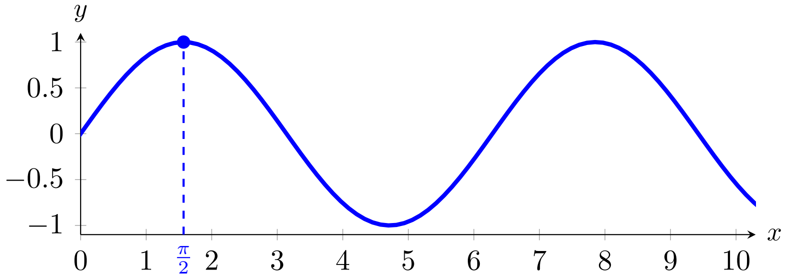


Figure 10.5: Line plot of the periodic sine function

We can use the derivative of the sine function in order to compute directly the rate of change, or slope, of the tangent line at this peak on the graph:

$$\sin'(\pi/2) = \cos(\pi/2) = 0$$

We find that this result corresponds well with the fact that the peak of the sine function is, indeed, a stationary point with zero rate of change.

A similar exercise can be easily carried out to compute the rate of change of the tangent line at different angles, for both the sine and cosine functions.

10.3 Finding derivatives in Python

In this section, we shall be finding the derivatives of the sine and cosine functions in Python.

For this purpose, we will be making use of the SymPy library, which will let us deal with the computation of mathematical objects symbolically. This means that the SymPy library will let us define and manipulate the sine and cosine functions, with unevaluated variables, in symbolic form. We will be able to define a variable as symbol by making use of `symbols` in Python, whereas to take the derivatives we shall be using the `diff` function.

Before proceeding further, let us first load the required libraries.

```python
from sympy import diff
from sympy import sin
from sympy import cos
from sympy import symbols
```

Program 10.1: Libraries required

We can now proceed to define a variable x in symbolic form, which means that we can work with x without having to assign it a value.

```python
# define variable as symbol
x = symbols('x')
```

Program 10.2: Define a symbol in SymPy

Next, we can find the derivative of the sine and cosine function with respect to x, using the `diff` function.

```python
# find the first derivative of sine and cosine with respect to x
print('The first derivative of sine is:', diff(sin(x), x))
print('The first derivative of cosine is:', diff(cos(x), x))
```

Program 10.3: Computing derivatives

We find that the `diff` function correctly returns `cos(x)` as the derivative of sine, and `-sin(x)` as the derivative of cosine.

```
The first derivative of sine is: cos(x)
The first derivative of cosine is: -sin(x)
```

Output 10.1: Result of computing derivatives

The `diff` function can take multiple derivatives too. For example, we can find the second derivative for both sine and cosine by passing `x` twice.

```python
# find the second derivative of sine and cosine with respect to x
print('The second derivative of sine is:', diff(sin(x), x, x))
print('The second derivative of cosine is:', diff(cos(x), x, x))
```

Program 10.4: Computing second derivatives

This means that, in finding the second derivative, we are taking the derivative of the derivative of each function. For example, to find the second derivative of the sine function, we take the derivative of `cos(x)`, its first derivative. We can find the second derivative for the cosine function by similarly taking the derivative of `-sin(x)`, its first derivative.

```
The second derivative of sine is: -sin(x)
The second derivative of cosine is: -cos(x)
```

Output 10.2: Result of second derivatives

We can, alternatively, pass the number 2 to the diff function to indicate that we are interested in finding the second derivative.

```python
# find the second derivative of sine and cosine with respect to x
print('The second derivative of sine is:', diff(sin(x), x, 2))
print('The second derivative of cosine is:', diff(cos(x), x, 2))
```

Program 10.5: Alternative way of finding second derivatives

Tying all of this together, the complete example of finding the derivative of the sine and cosine functions is listed below.

```python
# finding the derivative of the sine and cosine functions
from sympy import diff
from sympy import sin
from sympy import cos
from sympy import symbols

# define variable as symbol
x = symbols('x')

# find the first derivative of sine and cosine with respect to x
print('The first derivative of sine is:', diff(sin(x), x))
print('The first derivative of cosine is:', diff(cos(x), x))

# find the second derivative of sine and cosine with respect to x
print('\nThe second derivative of sine is:', diff(sin(x), x, x))
print('The second derivative of cosine is:', diff(cos(x), x, x))

# find the second derivative of sine and cosine with respect to x
print('\nThe second derivative of sine is:', diff(sin(x), x, 2))
print('The second derivative of cosine is:', diff(cos(x), x, 2))
```

Program 10.6: Complete code for computing derivatives using SymPy

10.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Michael Spivak. *The Hitchhiker's Guide to Calculus*. American Mathematical Society, 2019.
https://www.amazon.com/dp/1470449625/

Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.
https://amzn.to/3je8O1J

10.5 Summary

In this tutorial, you discovered how to find the derivative of the sine and cosine functions. Specifically, you learned:

- ▷ How to find the derivative of the sine and cosine functions by applying several rules from algebra, trigonometry and limits.
- ▷ How to find the derivative of the sine and cosine functions in Python.

This completes the most common building blocks for finding a derivative of a function. In the next chapter, we will see how the derivative of a function involving power, multiplication, and division can be broken down by using these building blocks.

The Power, Product, and Quotient Rules

11

Optimization, as one of the core processes in many machine learning algorithms, relies on the use of derivatives in order to decide in which manner to update a model's parameter values, to maximize or minimize an objective function.

This tutorial will continue exploring the different techniques by which we can find the derivatives of functions. In particular, we will be exploring the power, product and quotient rules, which we can use to arrive to the derivatives of functions faster than if we had to find every derivative from first principles. Hence, for functions that are especially challenging, keeping such rules at hand to find their derivatives will become increasingly important.

In this tutorial, you will discover the power, product and quotient rules to find the derivative of functions.

After completing this tutorial, you will know:

- The power rule to follow when finding the derivative of a variable base, raised to a fixed power.
- How the product rule allows us to find the derivative of a function that is defined as the product of another two (or more) functions.
- How the quotient rule allows us to find the derivative of a function that is the ratio of two differentiable functions.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- The Power Rule
- The Product Rule
- The Quotient Rule

11.1 The power rule

If we have a variable base raised to a fixed power, the rule to follow in order to find its derivative is to bring down the power in front of the variable base, and then subtract the power by 1.

For example, if we have the function, $f(x) = x^2$, of which we would like to find the derivative, we first bring down 2 in front of x and then reduce the power by 1:

$$f(x) = x^2$$

$$f'(x) = 2x$$

For the purpose of understanding better where this rule comes from, let's take the longer route and find the derivative of $f(x)$ by starting from the definition of a derivative:

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Here, we substitute for $f(x) = x^2$ and then proceed to simplify the expression:

$$\begin{aligned} f'(x) &= \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h} \\ &= \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} \\ &= \lim_{h \to 0} \frac{2xh + h^2}{h} \\ &= \lim_{h \to 0} (2x + h) \end{aligned}$$

As h approaches a value of 0, then this limit approaches $2x$, which tallies with the result that we have obtained earlier using the power rule.

If applied to $f(x) = x$, the power rule give us a value of 1. That is because, when we bring a value of 1 in front of x, and then subtract the power by 1, what we are left with is a value of 0 in the exponent. Since, $x^0 = 1$, then $f'(x) = (1)(x^0) = 1$.

> “The best way to understand this derivative is to realize that $f(x) = x$ is a line that fits the form $y = mx + b$ because $f(x) = x$ is the same as $f(x) = 1x + 0$ (or $y = 1x + 0$). The slope (m) of this line is 1, so the derivative equals 1. Or you can just memorize that the derivative of x is 1. But if you forget both of these ideas, you can always use the power rule.”
>
> — Page 131, *Calculus For Dummies*, 2016.

The power rule can be applied to any power, be it positive, negative, or a fraction. We can also apply it to radical functions by first expressing their exponent (or power) as a fraction:

$$f(x) = \sqrt{x} = x^{1/2}$$

$$f'(x) = \frac{1}{2} x^{-1/2} = \frac{1}{2\sqrt{x}}$$

These examples can be verified with SymPy:

```
from sympy import diff, sqrt, pprint
from sympy.abc import x

expressions = [x**2, sqrt(x)]
for expression in expressions:
    result = diff(expression, x)
    print(”Derivative of”)
    pprint(expression)
    print(”with respect to x is”)
    pprint(result)
    print()
```

Program 11.1: Verifying the examples of finding derivatives using power rule

```
Derivative of
 2
x
with respect to x is
2⋅x

Derivative of
√x
with respect to x is
 1
────
2⋅√x
```

Output 11.1: Derivatives using power rule

11.2 The product rule

Suppose that we now have a function, $f(x)$, of which we would like to find the derivative, which is the product of another two functions, $u(x) = 2x^2$ and $v(x) = x^3$:

$$f(x) = u(x)v(x) = (2x^2)(x^3)$$

In order to investigate how to go about finding the derivative of $f(x)$, let's first start with finding the derivative of the product of $u(x)$ and $v(x)$ directly:

$$(u(x)v(x))' = ((2x^2)(x^3))' = (2x^5)' = 10x^4$$

Now let's investigate what happens if we, otherwise, had to compute the derivatives of the functions separately first and then multiply them afterwards:

$$u'(x)v'(x) = (2x^2)'(x^3)' = (4x)(3x^2) = 12x^3$$

It is clear that the second result does not tally with the first one, and that is because we have not applied the *product rule*.

The product rule tells us that the derivative of the product of two functions can be found as:

$$f'(x) = u'(x)v(x) + u(x)v'(x)$$

We can arrive at the product rule if we our work our way through by applying the properties of limits, starting again with the definition of a derivative:

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

We know that $f(x) = u(x)v(x)$ and, hence, we can substitute for $f(x)$ and $f(x+h)$:

$$f'(x) = \lim_{h \to 0} \frac{u(x+h)v(x+h) - u(x)v(x)}{h}$$

At this stage, our aim is to factorize the numerator into several limits that can, then, be evaluated separately. For this purpose, the subtraction of terms, $u(x)v(x+h) - u(x)v(x+h)$, shall be introduced into the numerator. Its introduction does not change the definition of $f'(x)$ that we have just obtained, but it will help us factorize the numerator:

$$f'(x) = \lim_{h \to 0} \frac{u(x+h)v(x+h) + u(x)v(x+h) - u(x)v(x+h) - u(x)v(x)}{h}$$

The resulting expression appears complicated, however, if we take a closer look we realize that we have common terms that can be factored out:

$$f'(x) = \lim_{h \to 0} \frac{[u(x+h) - u(x)] \cdot v(x+h) + u(x) \cdot [v(x+h) - v(x)]}{h}$$

The expression can be simplified further by applying the limit laws that let us separate the sums and products into separate limits:

$$f'(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h} \cdot \lim_{h \to 0} v(x+h) + \lim_{h \to 0} u(x) \cdot \lim_{h \to 0} \frac{v(x+h) - v(x)}{h}$$

The solution to our problem has now become clearer. We can see that the first and last terms in the simplified expression correspond to the definition of the derivative of $u(x)$ and $v(x)$, which we can denote by $u(x)'$ and $v(x)'$, respectively. The second term approaches the continuous and differentiable function, $v(x)$, as h approaches 0, whereas the third term is $u(x)$.

Hence, we arrive again at the product rule:

$$f'(x) = u'(x)v(x) + u(x)v'(x)$$

With this new tool in hand, let's reconsider finding $f'(x)$ when $u(x) = 2x$ and $v(x) = x^3$:

$$\begin{aligned} f'(x) &= u'(x)v(x) + u(x)v'(x) \\ &= (4x)(x^3) + (2x^2)(3x^2) \\ &= 4x^4 + 6x^4 \\ &= 10x^4 \end{aligned}$$

The resulting derivative now correctly matches the derivative of the product, $(u(x)v(x))'$, that we have obtained earlier.

This was a fairly simple example that we could have computed directly in the first place. However, we might have more complex problems involving functions that cannot be multiplied directly, to which we can easily apply the product rule. For example:

$$\begin{aligned} f(x) &= x^2 \sin x \\ f'(x) &= (x^2)'(\sin x) + (x^2)(\sin x)' \\ &= 2x \sin x + x^2 \cos x \end{aligned}$$

This can be verified using SymPy:

```python
from sympy import diff, sin, pprint
from sympy.abc import x

u = x**2
v = sin(x)
f = u * v
result = diff(f, x)
print(”Derivative of”)
pprint(f)
print(”with respect to x is”)
pprint(result)
```

Program 11.2: Finding the derivative of $f(x) = x^2 \sin x$

```
Derivative of
 2
x ⋅sin(x)
with respect to x is
 2
x ⋅cos(x) + 2⋅x⋅sin(x)
```

Output 11.2: The derivative of $f(x) = x^2 \sin x$

We can even extend the product rule to more than two functions. For example, say $f(x)$ is now defined as the product of three functions, $u(x)$, $v(x)$ and $w(x)$:

$$f(x) = u(x)v(x)w(x)$$

We can apply the product rule as follows:

$$f'(x) = u'(x)v(x)w(x) + u(x)v'(x)w(x) + u(x)v(x)w'(x)$$

11.3 The quotient rule

Similarly, the quotient rule tells us how to find the derivative of a function, $f(x)$, that is the ratio of two differentiable functions, $u(x)$ and $v(x)$:

$$\begin{aligned} f(x) &= \frac{u(x)}{v(x)} \\ &= \frac{u'(x)v(x) - u(x)v'(x)}{v(x)^2} \end{aligned}$$

We can derive the quotient rule from first principles as we have done for the product rule, that is by starting off with the definition of a derivative and applying the properties of limits. Or we can take a shortcut and derive the quotient rule using the product rule itself. Let's take this route this time around:

$$f(x) = \frac{u(x)}{v(x)} \quad \longrightarrow \quad u(x) = f(x)v(x)$$

We can apply the product rule on $u(x)$ to obtain:

$$u'(x) = f'(x)v(x) + f(x)v'(x)$$

Solving back for $f'(x)$ gives us:

$$f'(x) = \frac{u'(x) - f(x)v'(x)}{v(x)}$$

One final step substitutes for $f(x)$ to arrive to the quotient rule:

$$f'(x) = \frac{u'(x) - \frac{u(x)}{v(x)}v'(x)}{v(x)} = \frac{u'(x)v(x) - u(x)v'(x)}{v^2(x)}$$

We had seen how to find the derivative of the sine and cosine functions in the previous chapter. Using the quotient rule, we can now find the derivative of the tangent function too:

$$f(x) = \tan x = \frac{\sin x}{\cos x}$$

Applying the quotient rule and simplifying the resulting expression:

$$\begin{aligned} f'(x) &= \frac{(\sin x)' \cos x - \sin x (\cos x)'}{\cos^2 x} \\ &= \frac{\cos x \cos x - \sin x(-\sin x)}{\cos^2 x} \\ &= \frac{\cos^2 x + \sin^2 x}{\cos^2 x} \end{aligned}$$

From the Pythagorean identity in trigonometry, we know that $\cos^2 x + \sin^2 x = 1$, hence:

$$f'(x) = \frac{1}{\cos^2 x} = \sec^2 x$$

Therefore, using the quotient rule, we have easily found that the derivative of tangent is the squared secant function. This is the same as what we can verify with SymPy:

```python
from sympy import diff, sin, cos, simplify, pprint
from sympy.abc import x

f = sin(x) / cos(x)
result = diff(f, x)
```

```
print("Derivative of")
pprint(f)
print("with respect to x is")
pprint(simplify(result))
```

Program 11.3: Finding the derivative of $\tan(x)$

```
Derivative of
sin(x)
──────
cos(x)
with respect to x is
   1
───────
   2
cos (x)
```

Output 11.3: The derivative of $\tan(x)$

11.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Articles

Power rule. Wikipedia.
https://en.wikipedia.org/wiki/Power_rule
Product rule. Wikipedia.
https://en.wikipedia.org/wiki/Product_rule
Quotient rule. Wikipedia.
https://en.wikipedia.org/wiki/Quotient_rule

11.5 Summary

In this tutorial, you discovered how to apply the power, product and quotient rules to find the derivative of functions. Specifically, you learned:

- ▷ The power rule for finding the derivative of a variable base, raised to a fixed power.
- ▷ How the product rule and quotient allows us to find the derivative of a function that is defined as the product and ratio of another two (or more) functions, respectively.

In the next chapter, we will see what can be done to find limits of a function that cannot be evaluated directly.

Indeterminate Forms and l’Hôpital’s Rule

12

Indeterminate forms are often encountered when evaluating limits of functions, and limits in turn play an important role in mathematics and calculus. They are essential for learning about derivatives, gradients, Hessians, and a lot more.

In this tutorial, you will discover how to evaluate the limits of indeterminate forms and the l’Hôpital’s rule for solving them. After completing this tutorial, you will know:

- How to evaluate the limits of functions having indeterminate types of the form $0/0$ and ∞/∞
- l’Hôpital’s rule for evaluating indeterminate types
- How to convert more complex indeterminate types and apply l’Hôpital’s rule to them

Let’s get started.

Overview

This tutorial is divided into 2 parts; they are:

- The indeterminate forms of type $0/0$ and ∞/∞
    - How to apply l’Hôpital’s rule to these types
    - Solved examples of these two indeterminate types
- More complex indeterminate types
    - How to convert the more complex indeterminate types to $0/0$ and ∞/∞ forms
    - Solved examples of such types

12.1 Prerequisites

This tutorial requires a basic understanding of the following two topics:

- Limits and Continuity (Chapter 5)
- Evaluating limits (Chapter 6)

12.2 What are indeterminate forms?

When evaluating limits, we come across situations where the basic rules for evaluating limits might fail. For example, we can apply the quotient rule in case of rational functions:

$$\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)} \qquad \text{if} \qquad \lim_{x\to a} g(x) \neq 0$$

The above rule can only be applied if the expression in the denominator does not approach zero as x approaches a. A more complicated situation arises if both the numerator and denominator both approach zero as x approaches a. This is called an indeterminate form of type 0/0. Similarly, there are indeterminate forms of the type ∞/∞, given by:

$$\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)} \qquad \text{when} \qquad \lim_{x\to a} f(x) = \infty \text{ and } \lim_{x\to a} g(x) = \infty$$

12.3 What is l'Hôpital's rule?

The l'Hôpital rule states the following:

> If we have an indeterminate type of the form 0/0 or ∞/∞, i.e.,
>
> $$\lim_{x\to a} f(x) = 0 \text{ and } \lim_{x\to a} g(x) = 0 \quad \text{or} \quad \lim_{x\to a} f(x) = \pm\infty \text{ and } \lim_{x\to a} g(x) = \pm\infty$$
>
> then
>
> $$\lim_{x\to a}\frac{f(x)}{g(x)} = \lim_{x\to a}\frac{f'(x)}{g'(x)}$$

When to apply l'Hôpital's rule

An important point to note is that l'Hôpital's rule is only applicable when the conditions for $f(x)$ and $g(x)$ are met. For example:

- ▷ $\lim_{x\to 0}\frac{\sin x}{x+1}$ cannot apply l'Hôpital's rule as it's not 0/0 form
- ▷ $\lim_{x\to 0}\frac{\sin x}{x}$ can apply the rule as it's 0/0 form
- ▷ $\lim_{x\to\infty}\frac{e^x}{1/(x+1)}$ cannot apply l'Hôpital's rule as it's not ∞/∞ form
- ▷ $\lim_{x\to\infty}\frac{e^x}{x}$ can apply l'Hôpital's rule as it is ∞/∞ form

12.4 Examples of 0/0 and ∞/∞

Some examples of these two types, and how to solve them are shown below. You can also refer to the figure below to refer to these functions.

Example 1: 0/0

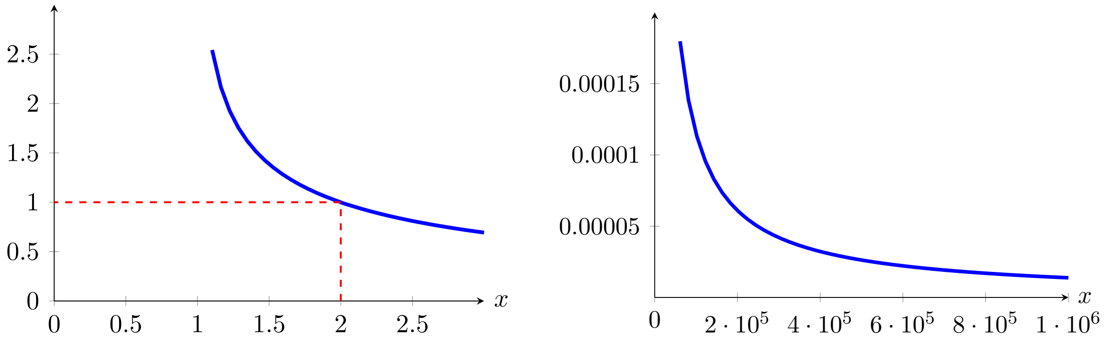
Evaluate $\lim\limits_{x \to 2} \dfrac{\ln(x-1)}{x-2}$ (See the left graph in the figure)

$$\begin{aligned}
\lim_{x \to 2} \frac{\ln(x-1)}{x-2} &= \lim_{x \to 2} \frac{\frac{d}{dx}\ln(x-1)}{\frac{d}{dx}(x-2)} && \text{apply l'Hôpital's rule to type 0/0} \\
&= \lim_{x \to 2} \frac{1/(x-1)}{1} \\
&= 1
\end{aligned}$$

Example 2: ∞/∞

Evaluate $\lim\limits_{x \to \infty} \dfrac{\ln x}{x}$ (See the right graph in the figure)

$$\begin{aligned}
\lim_{x \to \infty} \frac{\ln x}{x} &= \lim_{x \to \infty} \frac{\frac{d}{dx}\ln x}{\frac{d}{dx}x} && \text{apply l'Hôpital's rule to type } \infty/\infty \\
&= \lim_{x \to \infty} \frac{1/x}{1} \\
&= 0
\end{aligned}$$



$$\lim_{x \to 2} \frac{\ln(x-1)}{x-2} = 1 \qquad\qquad \lim_{x \to \infty} \frac{\ln x}{x} = 0$$

Figure 12.1: Graphs of Examples 1 and 2

We can verify these limit with SymPy:

```python
from sympy import limit, oo, ln, simplify, pprint
from sympy.abc import x

expression = ln(x-1)/(x-2)
result = limit(expression, x, 2)
print(”Limit of”)
pprint(expression)
print(”at x = 2 is”, result)
print()

expression = ln(x)/x
result = limit(expression, x, oo)
print("Limit of")
pprint(expression)
print("at x = infinity is", result)
```

Program 12.1: Verifying Examples 1 and 2

```
Limit of
log(x - 1)
──────────
  x - 2
at x = 2 is 1

Limit of
log(x)
──────
  x
at x = infinity is 0
```

Output 12.1: Verifying Examples 1 and 2

12.5 More indeterminate forms

The l'Hôpital rule only tells us how to deal with 0/0 or ∞/∞ forms. However, there are more indeterminate forms that involve products, differences, and powers. So how do we deal with the rest? We can use some clever tricks in mathematics to convert products, differences and powers into quotients. This can enable us to easily apply l'Hôpital rule to almost all indeterminate forms. The table below shows various indeterminate forms and how to deal with them.

Form	Limit	Conditions	Conversion rule
$\frac{0}{0}$	$\lim\limits_{x\to a}\frac{f(x)}{g(x)}$	$\lim\limits_{x\to a} f(x) = 0$ and $\lim\limits_{x\to a} g(x) = 0$	No conversion
$\frac{\infty}{\infty}$	$\lim\limits_{x\to a}\frac{f(x)}{g(x)}$	$\lim\limits_{x\to a} f(x) = \infty$ and $\lim\limits_{x\to a} g(x) = \infty$	No conversion
$0\cdot\infty$	$\lim\limits_{x\to a} f(x)g(x)$	$\lim\limits_{x\to a} f(x) = 0$ and $\lim\limits_{x\to a} g(x) = \infty$	Convert to $\lim\limits_{x\to a}\frac{f(x)}{g(x)}$
$\infty-\infty$	$\lim\limits_{x\to a}(f(x)-g(x))$	$\lim\limits_{x\to a} f(x) = \infty$ and $\lim\limits_{x\to a} g(x) = \infty$	Take out a common factor, rationalize
0^0	$\lim\limits_{x\to a} f(x)^{g(x)}$	$\lim\limits_{x\to a} f(x) = 0$ and $\lim\limits_{x\to a} g(x) = 0$	$f(x)^{g(x)} = e^{g(x)\ln f(x)}$ or let $y = f(x)^{g(x)}$ then $\ln y = g(x)\ln f(x)$
∞^0	$\lim\limits_{x\to a} f(x)^{g(x)}$	$\lim\limits_{x\to a} f(x) = \infty$ and $\lim\limits_{x\to a} g(x) = 0$	
1^∞	$\lim\limits_{x\to a} f(x)^{g(x)}$	$\lim\limits_{x\to a} f(x) = 1$ and $\lim\limits_{x\to a} g(x) = \infty$	

Table 12.1: How to solve more complex indeterminate forms

12.6 Examples

The following examples show how you can convert one indeterminate form to either 0/0 or ∞/∞ form and apply l'Hôpital's rule to solve the limit. After the worked out examples you can also look at the graphs of all the functions whose limits are calculated.

Example 3: $0 \cdot \infty$

Evaluate $\lim_{x\to\infty} x \cdot \sin\frac{1}{x}$ (See the first graph in Figure 12.2)

$$\begin{aligned}
\lim_{x\to\infty} x\cdot\sin\frac{1}{x} &= \lim_{x\to\infty}\frac{\sin(1/x)}{1/x} && \text{Convert type } 0\cdot\infty \text{ to } 0/0\\
&= \lim_{x\to\infty}\frac{\frac{d}{dx}\sin(1/x)}{\frac{d}{dx}(1/x)} && \text{Apply l'Hôpital's rule}\\
&= \lim_{x\to\infty}\frac{(-1/x^2)\cos(1/x)}{-1/x^2}\\
&= \lim_{x\to\infty}\cos\frac{1}{x}\\
&= 1
\end{aligned}$$

Example 4: $\infty - \infty$

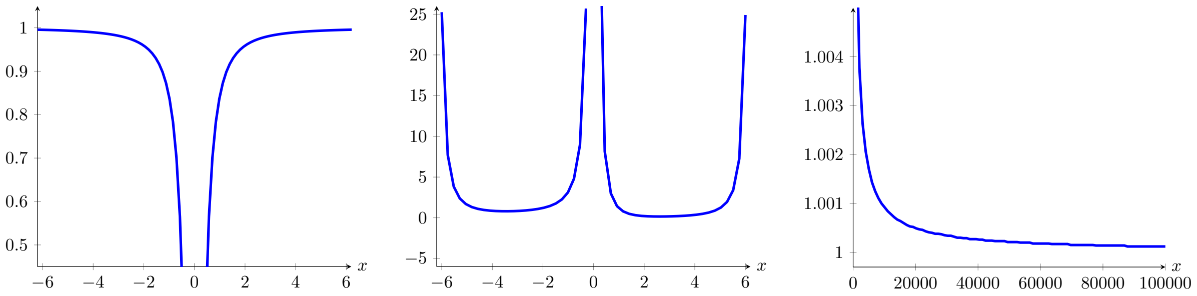
Evaluate $\lim_{x\to 0}\frac{1}{1-\cos x} - \frac{1}{x}$ (See the second graph in Figure 12.2)

$$\begin{aligned}
\lim_{x\to 0}\frac{1}{1-\cos x} &= \lim_{x\to 0}\frac{x-1+\cos x}{x(1-\cos x)} && \text{Convert type } \infty-\infty \text{ to type } 0/0\\
&= \lim_{x\to 0}\frac{\frac{d}{dx}(x-1+\cos x)}{\frac{d}{dx}(x(1-\cos x))} && \text{Apply l'Hôpital's rule}\\
&= \lim_{x\to 0}\frac{1-\sin x}{x\sin x+(1-\cos x)}\\
&= \infty
\end{aligned}$$

Example 5: Power form

Evaluate $\lim_{x\to\infty}(1+x)^{1/x}$ (See the third graph in Figure 12.2)

$$
\begin{aligned}
\lim_{x\to\infty}(1+x)^{1/x} &= \lim_{x\to\infty} e^{1/x\ln(1+x)} && \text{L.H.S. is type } \infty^0 \\
&= \lim_{x\to\infty} e^{y} && \text{let } y = \frac{\ln(1+x)}{x} \\
&= e^{\lim_{x\to\infty} y} && (*) \\
\lim_{x\to\infty} y &= \lim_{x\to\infty}\frac{\ln(1+x)}{x} && \text{Type } \infty/\infty \\
&= \lim_{x\to\infty}\frac{\frac{d}{dx}\ln(1+x)}{\frac{d}{dx}x} && \text{Apply l'Hôpital's rule} \\
&= \lim_{x\to\infty}\frac{\frac{1}{1+x}}{1} \\
&= 0 \\
\lim_{x\to\infty}(1+x)^{1/x} &= e^{0} && \text{Substitute } (*) \\
&= 1
\end{aligned}
$$



$\lim_{x\to\infty} x\cdot\sin\frac{1}{x} = 1$ $\lim_{x\to 0}\frac{1}{1-\cos x} - \frac{1}{x} = \infty$ $\lim_{x\to\infty}(1+x)^{1/x} = 1$

Figure 12.2: Graphs of examples 3, 4, and 5

We can verify these examples using SymPy as follows:

```
from sympy import limit, oo, sin, cos, simplify, pprint
from sympy.abc import x

expression = x * sin(1/x)
result = limit(expression, x, oo)
print("Limit of")
pprint(expression)
print("at x = infinity is", result)
print()
```

```
expression = 1/(1-cos(x)) - 1/x
result = limit(expression, x, 0)
print("Limit of")
pprint(expression)
print("at x = 0 is", result)
print()

expression = (1+x)**(1/x)
result = limit(expression, x, oo)
print("Limit of")
pprint(expression)
print("at x = infinity is", result)
```

Program 12.2: Verifying the limits in Examples 3, 4, and 5

```
Limit of
     ⎛1⎞
x⋅sin⎜─⎟
     ⎝x⎠
at x = infinity is 1

Limit of
    1        1
────────── - ─
1 - cos(x)   x
at x = 0 is oo

Limit of
x _______
╲╱ x + 1
at x = infinity is 1
```

Output 12.2: The limits in Examples 3, 4, and 5

12.7 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

12.8 Summary

In this tutorial, you discovered the concept of indeterminate forms and how to evaluate them. Specifically, you learned:

- ▷ Indeterminate forms of type $0/0$ and ∞/∞
- ▷ l'Hôpital rule for evaluating types $0/0$ and ∞/∞
- ▷ Indeterminate forms of type $0 \cdot \infty$, $\infty - \infty$, and power forms, and how to evaluate them.

In the next chapter, we are going to see how knowing the derivative of a function can help us.

13 Applications of Derivatives

The derivative defines the rate at which one variable changes with respect to another.

It is an important concept that comes in extremely useful in many applications: in everyday life, the derivative can tell you at which speed you are driving, or help you predict fluctuations on the stock market; in machine learning, derivatives are important for function optimization.

This tutorial will explore different applications of derivatives, starting with the more familiar ones before moving to machine learning. We will be taking a closer look at what the derivatives tell us about the different functions we are studying. In this tutorial, you will discover different applications of derivatives. After completing this tutorial, you will know:

▷ The use of derivatives can be applied to real-life problems that we find around us.

▷ The use of derivatives is essential in machine learning, for function optimization.

Let's get started.

Overview

This tutorial is divided into two parts; they are:

▷ Applications of Derivatives in Real-Life

▷ Applications of Derivatives in Optimization Algorithms

13.1 Applications of derivatives in real-life

We have seen that derivatives model rates of change.

> Derivatives answer questions like "How fast?" "How steep?" and "How sensitive?" These are all questions about rates of change in one form or another.
>
> — Page 141, *Infinite Powers*, 2020.

This rate of change is denoted by, $\frac{\delta y}{\delta x}$, hence defining a change in the dependent variable, δy, with respect to a change in the independent variable, δx.

Let's start off with one of the most familiar applications of derivatives that we can find around us.

> Every time you get in your car, you witness differentiation.
>
> — Page 178, *Calculus For Dummies*, 2016.

When we say that a car is moving at 100 kilometers an hour, we would have just stated its rate of change. The common term that we often use is *speed* or *velocity*, although it would be best that we first distinguish between the two.

In everyday life, we often use *speed* and *velocity* interchangeably if we are describing the rate of change of a moving object. However, this in not mathematically correct because speed is always positive, whereas velocity introduces a notion of direction and, hence, can exhibit both positive and negative values. Hence, in the ensuing explanation, we shall consider velocity as the more technical concept, defined as:

$$\text{velocity} = \frac{\delta y}{\delta t}$$

This means that velocity gives the change in the car's position, δy, within an interval of time, δt. In other words, velocity is the *first derivative* of position with respect to time.

The car's velocity can remain constant, such as if the car keeps on traveling at 100 kilometers an hour consistently, or it can also change as a function of time. In case of the latter, this means that the velocity function itself is changing as a function of time, or in simpler terms, the car can be said to be *accelerating*. Acceleration is defined as the first derivative of velocity, v, and the second derivative of position, y, with respect to time:

$$\text{acceleration} = \frac{\delta v}{\delta t} = \frac{\delta^2 y}{\delta t^2}$$

We can graph the position, velocity and acceleration curves to visualize them better. Suppose that the car's position, as a function of time, is given by

$$y(t) = t^3 - 8t^2 + 40t$$

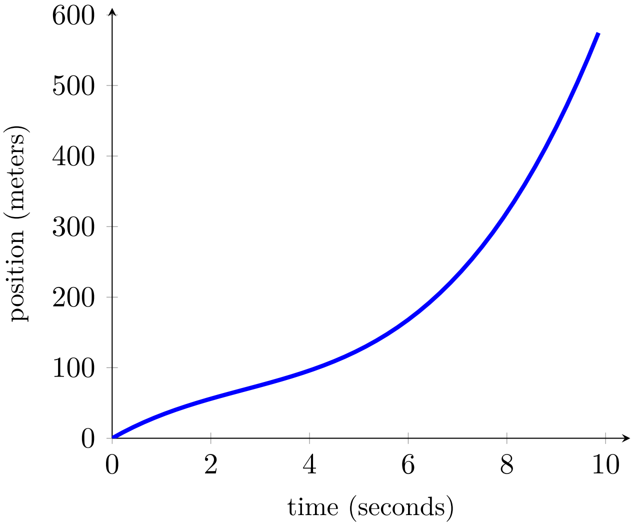


Figure 13.1: Line plot of the car's position against time

The graph indicates that the car's position changes slowly at the beginning of the journey, slowing down slightly until around t = 2.7s, at which point its rate of change picks up and continues increasing until the end of the journey. This is depicted by the graph of the car's velocity:

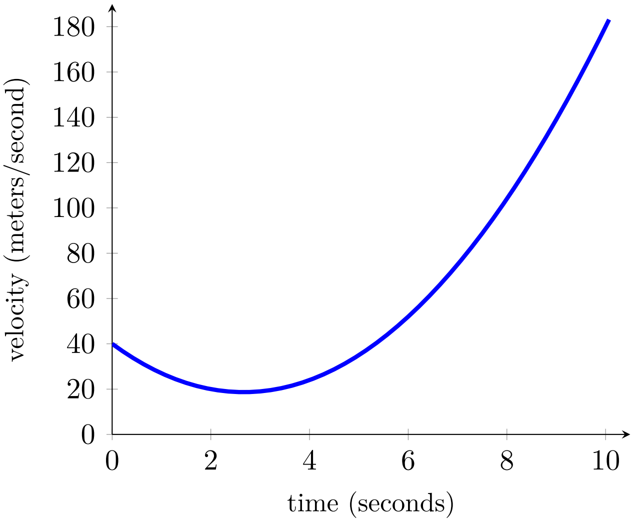


Figure 13.2: Line plot of the car's velocity against time

Notice that the car retains a positive velocity throughout the journey, and this is because it never changes direction. Hence, if we had to imagine ourselves sitting in this moving car, the speedometer would be showing us the values that we have just plotted on the velocity graph (since the velocity remains positive throughout, otherwise we would have to find the absolute value of the velocity to work out the speed). If we had to apply the power rule to $y(t)$ to find its derivative, then we would find that the velocity is defined by the following function:

$$v(t) = y'(t) = 3t^2 - 16t + 40$$

We can also plot the acceleration graph:

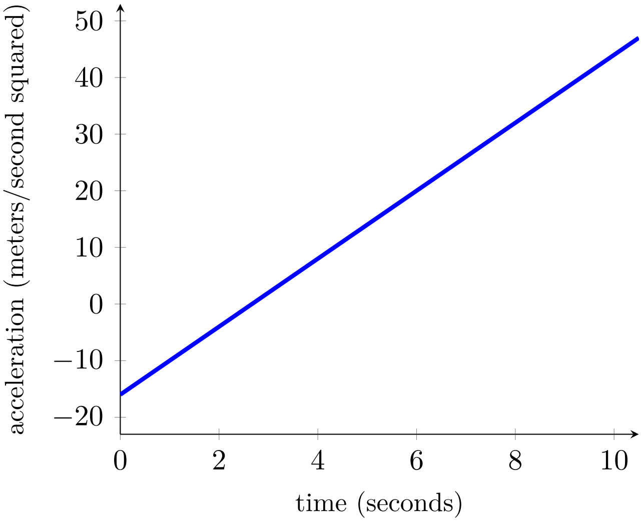


Figure 13.3: Line plot of the car's acceleration against time

We find that the graph is now characterized by negative acceleration in the time interval, $t = [0, 2.4]$ seconds. This is because acceleration is the derivative of velocity, and within this time interval the car's velocity is decreasing. If we had to, again, apply the power rule to $v(t)$ to find its derivative, then we would find that the acceleration is defined by the following function:

$$a(t) = v'(t) = 6t - 16$$

Putting all functions together, we have the following:

$$\begin{aligned} y(t) &= t^3 - 8t^2 + 40t \\ v(t) &= 3t^2 - 16t + 40 \\ a(t) &= 6t - 16 \end{aligned}$$

If we substitute for $t = 10$ seconds, we can use these three functions to find that by the end of the journey, the car has traveled 600m, its velocity is 180 m/s, and it is accelerating at 44 m/s^2. We can verify that all of these values tally with the graphs that we have just plotted.

We have framed this particular example within the context of finding a car's velocity and acceleration. But there is a plethora of real-life phenomena that change with time (or variables other than time), which can be studied by applying the concept of derivatives as we have just done for this particular example. To name a few:

- Growth rate of a population (be it a collection of humans, or a colony of bacteria) over time, which can be used to predict changes in population size in the near future.
- Changes in temperature as a function of location, which can be used for weather forecasting.
- Fluctuations of the stock market over time, which can be used to predict future stock market behavior.

Derivatives also provide salient information in solving optimization problems, as we shall be seeing next.

13.2 Applications of derivatives in optimization algorithms

We had already seen in Chapter 3 that an optimization algorithm, such as gradient descent, seeks to reach the global minimum of an error (or cost) function by applying the use of derivatives.

Let's take a closer look at what the derivatives tell us about the error function, by going through the same exercise as we have done for the car example.

For this purpose, let's consider the following one-dimensional test function for function optimization:

$$f(x) = -x \sin(x)$$

We can apply the product rule to $f(x)$ to find its first derivative, denoted by $f'(x)$, and then again apply the product rule to $f'(x)$ to find the second derivative, denoted by $f''(x)$:

$$f'(x) = -\sin(x) - x\cos(x)$$
$$f''(x) = x\sin(x) - 2\cos(x)$$

We can confirm the answer with SymPy:

```python
from sympy import diff, sin, pprint
from sympy.abc import x

f = -x * sin(x)
d1 = diff(f, x)
d2 = diff(f, x, x)
print(”Function”)
pprint(f)
print(”has first derivative”)
pprint(d1)
print(”and second derivative”)
pprint(d2)
```

Program 13.1: Finding the first and second derivatives of $f(x)$

```
Function
-x⋅sin(x)
has first derivative
-x⋅cos(x) - sin(x)
and second derivative
x⋅sin(x) - 2⋅cos(x)
```

Output 13.1: First and second derivatives of $f(x)$

We can plot these three functions for different values of x to visualize them:

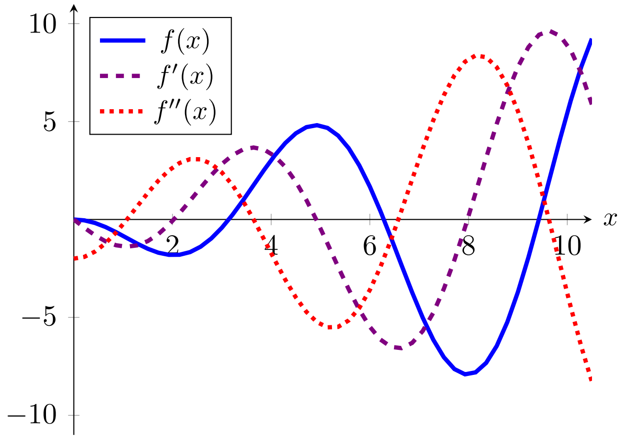


Figure 13.4: Line plot of $f(x)$, its first derivative $f'(x)$, and its second derivative $f''(x)$

Similar to what we have observed earlier for the car example, the graph of the first derivative indicates how $f(x)$ is changing and by how much. For example, a positive derivative indicates that $f(x)$ is an increasing function, whereas a negative derivative tells us that $f(x)$ is now decreasing. Hence, if in its search for a function minimum, the optimization algorithm performs small changes to the input based on its learning rate, ϵ:

$$x_{\text{new}} = x - \epsilon f'(x)$$

Then the algorithm can reduce $f(x)$ by moving to the opposite direction (by inverting the sign) of the derivative.

We might also be interested in finding the second derivative of a function.

> We can think of the second derivative as measuring curvature.
>
> — Page 86, *Deep Learning*, 2016.

For example, if the algorithm arrives at a critical point at which the first derivative is zero, it cannot distinguish between this point being a local maximum, a local minimum, a saddle point or a flat region based on $f'(x)$ alone. However, when the second derivative intervenes, the algorithm can tell that the critical point in question is a local minimum if the second derivative is greater than zero. For a local maximum, the second derivative is smaller than zero. Hence, the second derivative can inform the optimization algorithm on which direction to move. Unfortunately, this test remains inconclusive for saddle points and flat regions, for which the second derivative is zero in both cases.

Optimization algorithms based on gradient descent do not make use of second order derivatives and are, therefore, known as *first-order optimization algorithms*. Optimization algorithms, such as Newton's method, that exploit the use of second derivatives, are otherwise called *second-order optimization algorithms*.

13.3 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Steven Strogatz. *Infinite Powers. How Calculus Reveals the Secrets of the Universe*. Mariner Books, 2020.
https://www.amazon.com/dp/0358299284/

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.
https://amzn.to/3je8O1J

13.4 Summary

In this tutorial, you discovered different applications of derivatives. Specifically, you learned:

- The use of derivatives can be applied to real-life problems that we find around us.
- The use of derivatives is essential in machine learning, for function optimization.

In the next chapter, we will take a closer look at one application of differentiation, namely, the slope of a curve.

Slopes and Tangents

14

The slope of a line and its relationship to the tangent line of a curve is a fundamental concept in calculus. It is important for a general understanding of function derivatives.

In this tutorial, you will discover what is the slope of a line and what is a tangent to a curve. After completing this tutorial, you will know:

- ▷ The slope of a line
- ▷ The average rate of change of $f(x)$ on an interval with respect to x
- ▷ The slope of a curve
- ▷ The tangent line to a curve at a point

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- ▷ The slope of a line and a curve
- ▷ The tangent line to a curve

14.1 The slope of a line

Let's start by reviewing the slope of a line. In calculus, the slope of a line defines its steepness as a number. This number is calculated by dividing the change in the vertical direction to the change in the horizontal direction when moving from one point on the line to another. The figure shows how the slope can be calculated from two distinct points, A and B, on a line.

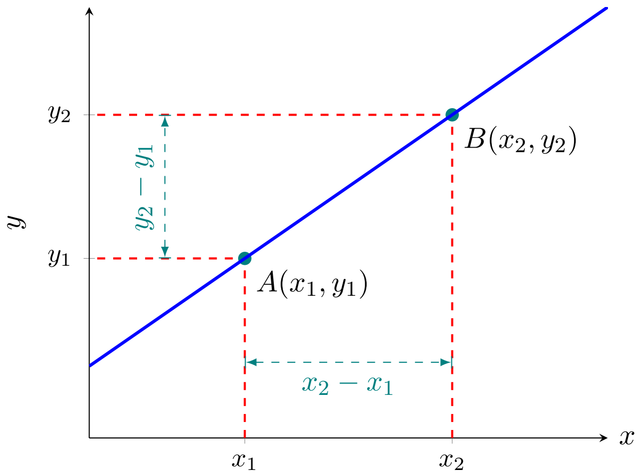


Figure 14.1: Slope of a line calculated from two points on the line

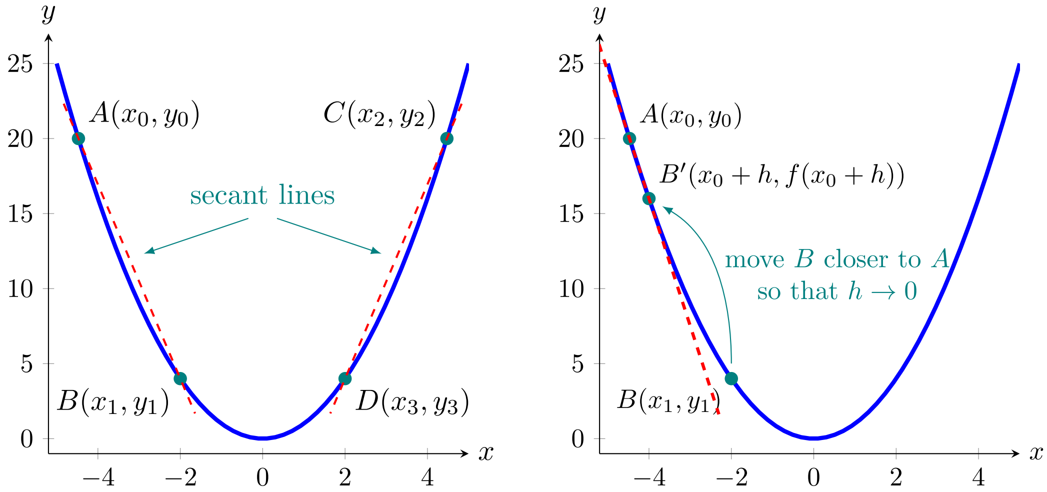
A straight line can be uniquely defined by two points on the line. The slope of a line is the same everywhere on the line; hence, any line can also be uniquely defined by the slope and one point on the line. From the known point, we can move to any other point on the line according to the ratio defined by the slope of the line.

14.2 The average rate of change of a curve

We can extend the idea of the slope of a line to the slope of a curve. Consider the left graph of the figure below. If we want to measure the "steepness" of this curve, it is going to vary at different points on the curve. The average rate of change when moving from point A to point B is negative as the value of the function is decreasing when x is increasing. It is the same when moving from point B to point A. Hence, we can define it over the interval $[x_0, x_1]$ as:

$$\frac{y_1 - y_0}{x_1 - x_0}$$

We can see that the above is also an expression for the slope of the secant line that includes the points A and B. To refresh your memory, a secant line intersects the curve at two points. Similarly, the average rate of change between point C and point D is positive, and it's given by the slope of the secant line that includes these two points.



Average rate of change over an interval

Average rate of change at a point

Figure 14.2: Rate of change of a curve over an interval vs. at a point

14.3 Defining the slope of the curve

Let's now look at the right graph of the above figure. What happens when we move point B toward point A? Let's call the new point B'. When the point B' is infinitesimally close to A, the secant line will turn into a line that touches the curve only once. Here the x coordinate of B' is $(x_0 + h)$, with h an infinitesimally small value. The corresponding value of the y-coordinate of the point B' is the value of this function at $(x_0 + h)$, i.e., $f(x_0 + h)$.

The average rate of change over the interval $[x_0, x_0 + h]$ represents the rate of change over a very small interval of length h, where h approaches zero. This is called the slope of the curve at the point x_0. Hence, at any point $A(x_0, f(x_0))$, the slope of the curve is defined as:

$$m = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

The expression of the slope of the curve at a point, A, is equivalent to the derivative of $f(x)$ at the value x_0. Hence, we can use the derivative to find the slope of the curve.

Examples of slope of the curve

Here are a few examples of the slope of the curve.

- ▷ The slope of $f(x) = \frac{1}{x}$ at any point k ($k \neq 0$) is given by $-\frac{1}{k^2}$. As an example:
  - ◦ Slope of $f(x) = \frac{1}{x}$ at $x = 2$ is $-\frac{1}{4}$
  - ◦ Slope of $f(x) = \frac{1}{x}$ at $x = -1$ is -1
- ▷ The slope of $f(x) = x^2$ at any point k is given by $2k$. For example:
  - ◦ Slope of $f(x) = x^2$ at $x = 0$ is 0

  - ○ Slope of $f(x) = x^2$ at $x = 1$ is 2

- ▷ The slope of $f(x) = 2x+1$ is a constant value equal to 2. We can see that $f(x)$ defines a straight line.
- ▷ The slope of $f(x) = k$, (where k is a constant) is zero as the function does not change anywhere. Hence, its average rate of change at any point is zero.

14.4 The tangent line

It was mentioned earlier that any straight line can be uniquely defined by its slope and a point that passes through it. We also just defined the slope of a curve at a point, A. Using these two facts, we'll define the tangent to a curve $f(x)$ at a point $A(x_0, f(x_0))$ as a line that satisfies two of the following:

1. The line passes through A
2. The slope of the line is equal to the slope of the curve at the point A

Using the above two facts, we can easily determine the equation of the tangent line at a point $(x_0, f(x_0))$. A few examples are shown next.

14.5 Examples of tangent lines

$f(x) = \frac{1}{x}$

The graph of $f(x)$ along with the tangent line at $x = 1$ and $x = -1$ are shown in the figure. Below are the steps to determine the tangent line at $x = 1$.

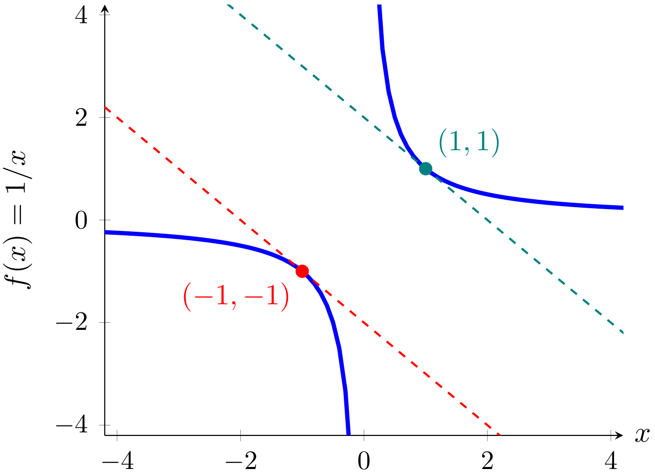


Figure 14.3: $f(x) = \frac{1}{x}$

- ▷ Equation of a line with slope m and y-intercept c is given by: $y = mx + c$
- ▷ Slope of the line at any point is given by the function $f'(x) = -\frac{1}{x^2}$

- ▷ Slope of the tangent line to the curve at $x = 1$ is -1, we get $y = -x + c$
- ▷ The tangent line passes through the point $(1, 1)$, and hence substituting in the above equation, we get:

$$1 = -(1) + c \quad \implies \quad c = 2$$

- ▷ The final equation of the tangent line is $y = -x + 2$

The similar procedure can be applied for $x = 2$ to find the equation of another tangent line to be $y = -x - 2$. We can verify the above result numerically in Python:

```python
import numpy as np

def f(x):
    return 1/x

epsilon = np.finfo(np.float32).eps
for x in [1, -1]:
    slope = (f(x+epsilon) - f(x))/epsilon
    y = f(x)
    c = y - slope * x
    print(”Slope at x={} is {}”.format(x, slope))
    print(”Tangent line is y={:f}x{:+f}”.format(slope,c))
```

Program 14.1: Find tangents for function $f(x) = 1/x$

```
Slope at x=1 is -0.9999998807907104
Tangent line is y=-1.000000x+2.000000
Slope at x=-1 is -1.0000001192092896
Tangent line is y=-1.000000x-2.000000
```

Output 14.1: Tangents found for $f(x) = 1/x$

$f(x) = x^2$

Shown below is the curve and the tangent lines at the points $x = 2$, $x = -2$, and $x = 0$. At $x = 0$, the tangent line is parallel to the x-axis as the slope of $f(x)$ at $x = 0$ is zero. This is how we compute the equation of the tangent line at $x = 2$:

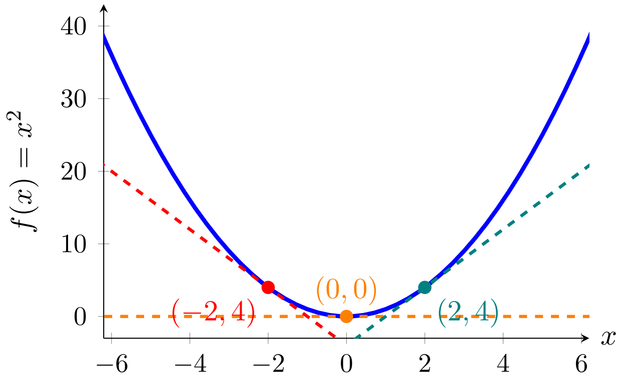


Figure 14.4: $f(x) = x^2$

- ▷ Equation of a line with slope m and y-intercept c is given by: $y = mx + c$
- ▷ Slope of the line at any point is given by the function $f'(x) = 2x$
- ▷ Slope of the tangent line to the curve at $x = 2$ is 4, we get $y = 4x + c$
- ▷ The tangent line passes through the point $(2, 4)$, and hence substituting in the above equation, we get:
$$4 = 4 \times (2) + c \quad \Longrightarrow \quad c = -4$$
- ▷ The final equation of the tangent line is $y = 4x - 4$

The similar procedure can be applied for $x = 2$ to find the equation of another tangent line to be $y = -4x - 4$. We can verify the above result numerically in Python:

```python
import numpy as np

def f(x):
    return x**2

epsilon = np.finfo(np.float32).eps
for x in [2, -2]:
    slope = (f(x+epsilon) - f(x))/epsilon
    y = f(x)
    c = y - slope * x
    print(”Slope at x={} is {}”.format(x, slope))
    print(”Tangent line is y={:f}x{:+f}”.format(slope,c))
```

Program 14.2: Find tangents for function $f(x) = x^2$

```
Slope at x=2 is 4.0000001192092896
Tangent line is y=4.000000x-4.000000
Slope at x=-2 is -3.9999998807907104
Tangent line is y=-4.000000x-4.000000
```

Output 14.2: Tangents found for $f(x) = x^2$

$f(x) = x^3 + 2x + 1$

This function is shown below, along with its tangent lines at $x = 0$, $x = 2$, and $x = -2$. Below are the steps to derive an equation of the tangent line at $x = 0$.

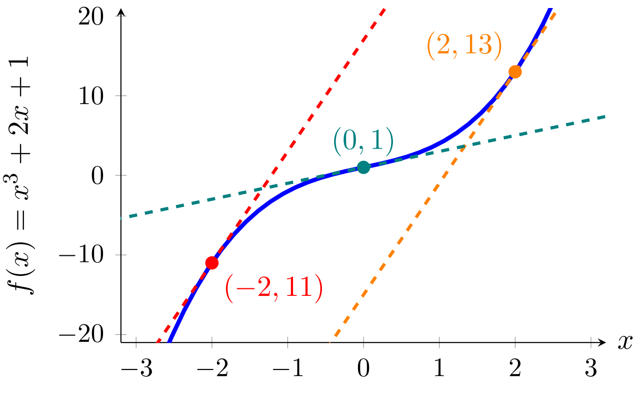


Figure 14.5: $f(x) = x^3 + 2x + 1$

- ▷ Equation of a line with slope m and y-intercept c is given by: $y = mx + c$
- ▷ Slope of the line at any point is given by the function $f'(x) = 3x^2 + 2$
- ▷ Slope of the tangent line to the curve at $x = 0$ is 2, we get $y = 2x + c$
- ▷ The tangent line passes through the point $(0, 1)$, and hence substituting in the above equation, we get:

$$1 = 2 \times (0) + c \implies c = 1$$

- ▷ The final equation of the tangent line is $y = 2x + 1$

The similar procedure can be applied for $x = 2$ to find the equation $y = 14x - 15$, and for $x = -2$ to find the equation $y = 14x + 17$. Note that the curve has the same slope at both $x = 2$ and $x = -2$, and hence the two tangent lines at $x = 2$ and $x = -2$ are parallel. The same would be true for any $x = k$ and $x = -k$ as $f'(x) = f'(-x) = 3x^2 + 2$. We can verify the above result numerically in Python:

```
import numpy as np

def f(x):
    return x**3 + 2*x + 1

epsilon = np.finfo(np.float32).eps
for x in [2, 0, -2]:
    slope = (f(x+epsilon) - f(x))/epsilon
    y = f(x)
    c = y - slope * x
    print(”Slope at x={} is {}”.format(x, slope))
    print(”Tangent line is y={:f}x{:+f}”.format(slope,c))
```

Program 14.3: Find tangents for function $f(x) = x^3 + 2x + 1$

```
Slope at x=2 is 14.000000715255737
Tangent line is y=14.000001x-15.000001
Slope at x=0 is 2.0
Tangent line is y=2.000000x+1.000000
```

```
Slope at x=-2 is 13.999999284744263
Tangent line is y=13.999999x+16.999999
```

Output 14.3: Tangents found for $f(x) = x^3 + 2x + 1$

14.6 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

14.7 Summary

In this tutorial, you discovered the concept of the slope of a curve at a point and the tangent line to a curve at a point. Specifically, you learned:

- What is the slope of a line
- What is the average rate of change of a curve over an interval with respect to x
- Slope of a curve at a point
- Tangent to a curve at a point

In the next chapter, we will introduce integral calculus as the reverse of differential calculus.

15 Differential and Integral Calculus

Integral calculus was one of the greatest discoveries of Newton and Leibniz. Their work independently led to the proof, and recognition of the importance of the fundamental theorem of calculus, which linked integrals to derivatives. With the discovery of integrals, areas and volumes could thereafter be studied.

Integral calculus is the second half of the calculus journey that we will be exploring. In this tutorial, you will discover the relationship between differential and integral calculus. After completing this tutorial, you will know:

- The concepts of differential and integral calculus are linked together by the fundamental theorem of calculus.
- By applying the fundamental theorem of calculus, we can compute the integral to find the area under a curve.
- In machine learning, the application of integral calculus can provide us with a metric to assess the performance of a classifier.

Let's get started.

Overview

This tutorial is divided into four parts; they are:

- The link between differential and integral calculus
- The fundamental theorem of calculus
- Integration Example
- Application of Integration in Machine Learning

15.1 The link between differential and integral calculus

In our journey through calculus so far, we have learned that differential calculus is concerned with the measurement of the rate of change. We have also discovered differentiation, and applied it to different functions from first principles. We have even understood how to apply rules to arrive to the derivative faster. But we are only half way through the journey.

> From A twenty-first-century vantage point, calculus is often seen as the mathematics of change. It quantifies change using two big concepts: derivatives and integrals. Derivatives model rates of change ...Integrals model the accumulation of change ...
>
> — Page 141, *Infinite Powers*, 2020.

Recall having said that calculus comprises two phases: cutting and rebuilding.

The cutting phase breaks down a curved shape into infinitesimally small and straight pieces that can be studied separately, such as by applying derivatives to model their rate of change, or *slope*. This half of the calculus journey is called *differential* calculus, and we have already looked into it in some detail.

The rebuilding phase gathers the infinitesimally small and straight pieces, and sums them back together in an attempt to study the original whole. In this manner, we can determine the area or volume of regular and irregular shapes after having cut them into infinitely thin slices. This second half of the calculus journey is what we shall be exploring next. It is called *integral* calculus.

The important theorem that links the two concepts together is called the *fundamental theorem of calculus.*

15.2 The fundamental theorem of calculus

In order to work our way towards understanding the fundamental theorem of calculus, let's revisit the car's position and velocity example:

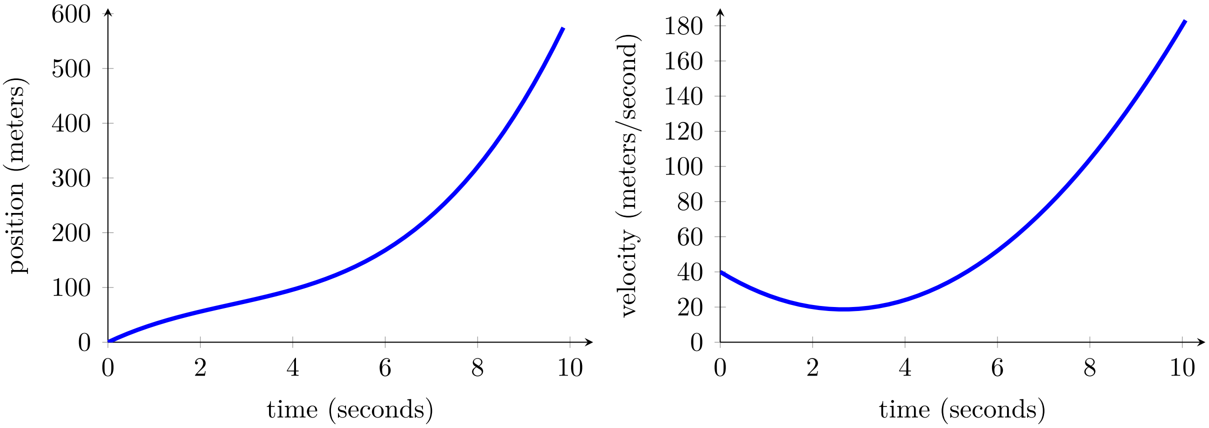


Figure 15.1: Line plot of the car's position and velocity against time

In computing the derivative we had solved the *forward* problem, where we found the velocity from the slope of the position graph at any time, t. But what if we would like to solve the *backward* problem, where we are given the velocity graph, $v(t)$, and wish to find the distance traveled? The solution to this problem is to calculate the *area under the curve*(the shaded region) up to time, t:

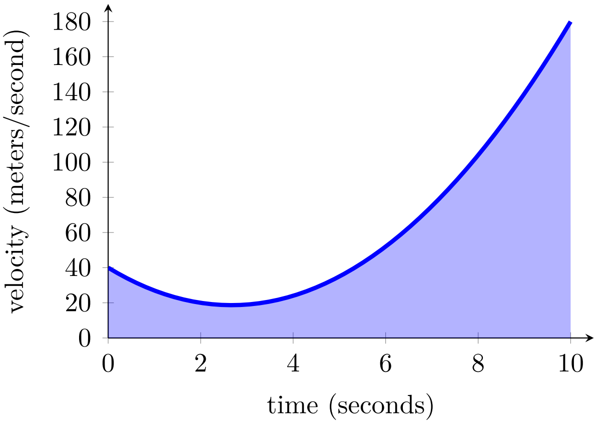


Figure 15.2: The shaded region is the area under the curve

We do not have a specific formula to define the area of the shaded region directly. But we can apply the mathematics of calculus to cut the shaded region under the curve into many infinitely thin rectangles, for which we have a formula:

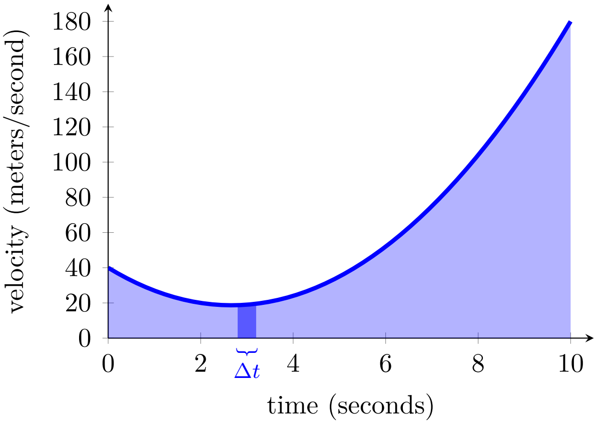


Figure 15.3: Cutting the shaded region into many rectangles of width, Δt

If we consider the i-th rectangle, chosen arbitrarily to span the time interval Δt, we can define its area as its length times its width:

$$\text{area of rectangle} = v(t_i)\Delta t_i$$

We can have as many rectangles as necessary in order to span the interval of interest, which in this case is the shaded region under the curve. For simplicity, let's denote this closed interval by $[a, b]$. Finding the area of this shaded region (and, hence, the distance traveled), then reduces to finding the sum of the n number of rectangles:

$$\text{total area} = v(t_0)\Delta t_0 + v(t_1)\Delta t_1 + \cdots v(t_n)\Delta t_n$$

We can express this sum even more compactly by applying the Riemann sum with sigma notation:

$$\sum_{i=1}^{n} v(t_i)\Delta t_i = v(t_0)\Delta t_0 + v(t_1)\Delta t_1 + \cdots + v(t_n)\Delta t_n$$

If we cut (or divide) the region under the curve by a finite number of rectangles, then we find that the Riemann sum gives us an *approximation* of the area, since the rectangles will not fit the area under the curve exactly. If we had to position the rectangles so that their upper left or upper right corners touch the curve, the Riemann sum gives us either an underestimate or an overestimate of the true area, respectively. If the midpoint of each rectangle had to touch the curve, then the part of the rectangle protruding above the curve *roughly* compensates for the gap between the curve and neighboring rectangles:

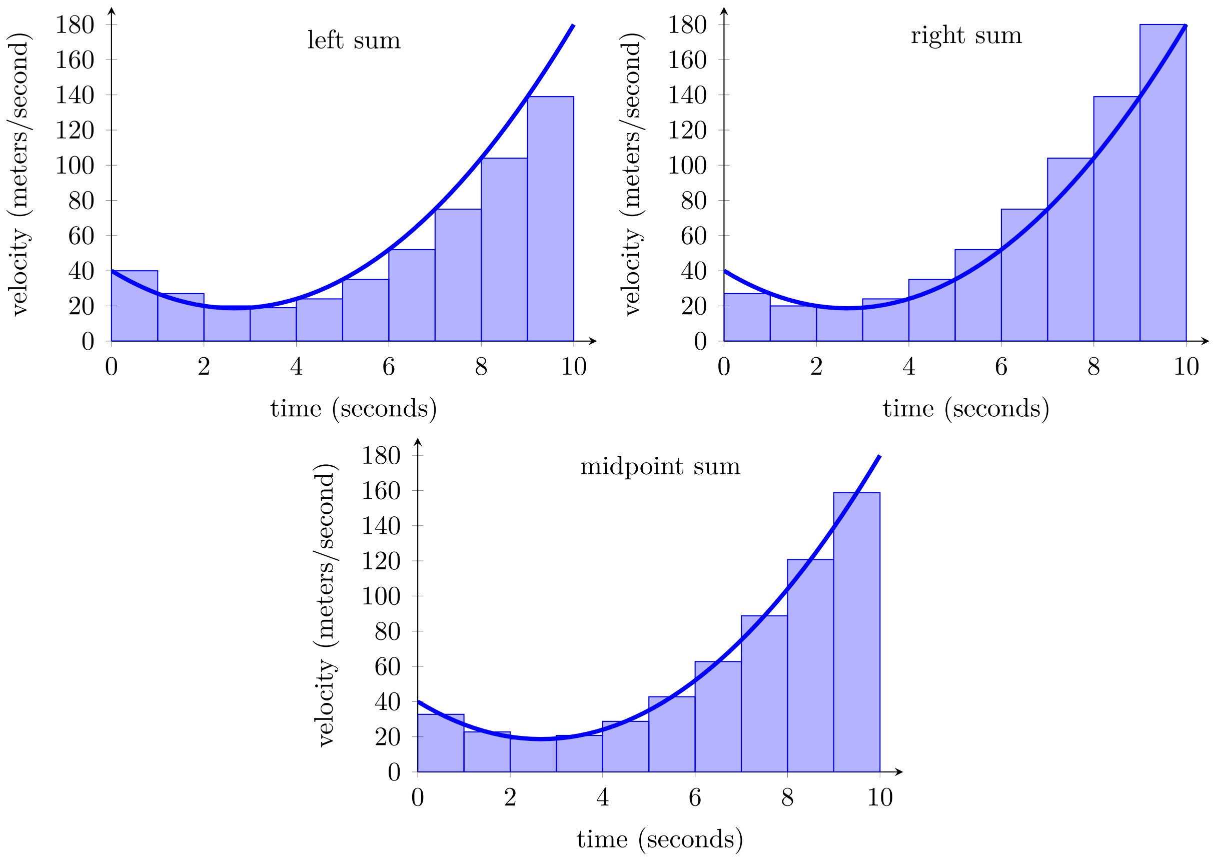


Figure 15.4: Approximating the area under the curve with left sum, right sum, and midpoint sums

The solution to finding the *exact* area under the curve, is to reduce the rectangles' width so much that they become *infinitely* thin (recall the infinity principle in calculus). In this manner, the rectangles would be covering the entire region, and in summing their areas we would be finding the *definite integral*.

> The definite integral ("simple" definition): The exact area under a curve between $t = a$ and $t = b$ is given by the definite integral, which is defined as the limit of a Riemann sum ...
>
> — Page 227, *Calculus For Dummies*, 2016.

The definite integral can, then, be defined by the Riemann sum as the number of rectangles, n, approaches infinity. Let's also denote the area under the curve by $A(t)$. Then:

$$A(t) = \int_a^b v(t)dt = \lim_{n \to \infty} \sum_{i=1}^{n} v(t_i)\Delta t_i$$

Note that the notation now changes into the integral symbol, $\int$, replacing sigma, $\sum$. The reason behind this change is, merely, to indicate that we are summing over a huge number of thinly sliced rectangles. The expression on the left hand side reads as, the integral of $v(t)$ from a to b, and the process of finding the integral is called *integration*.

The sweeping area analogy

Perhaps a simpler analogy to help us relate integration to differentiation, is to imagine holding one of the thinly cut slices and dragging it rightwards under the curve in infinitesimally small steps. As it moves rightwards, the thinly cut slice will sweep a larger area under the curve, while its height will change according to the shape of the curve. The question that we would like to answer is, at which *rate* does the area accumulate as the thin slice sweeps rightwards?

Let dt denote each infinitesimal step traversed by the sweeping slice, and $v(t)$ its height at any time, t. Then the infinitesimal area, $dA(t)$, of this thin slice can be found by multiplying its height, $v(t)$, to its infinitesimal width, dt:

$$dA(t) = v(t)dt$$

Dividing the equation by dt gives us the derivative of $A(t)$, and tells us that the rate at which the area accumulates is equal to the height of the curve, $v(t)$, at time, t:

$$\frac{dA(t)}{dt} = v(t)$$

We can finally define the fundamental theorem of calculus.

The fundamental theorem of calculus — Part 1

We found that an area, $A(t)$, swept under a function, $v(t)$, can be defined by:

$$A(t) = \int_a^b v(t)dt$$

We have also found that the rate at which the area is being swept is equal to the original function, $v(t)$:

$$\frac{dA(t)}{dt} = v(t)$$

This brings us to the first part of the fundamental theorem of calculus, which tells us that if $v(t)$ is continuous on an interval, $[a, b]$, and if it is also the derivative of $A(t)$, then $A(t)$ is the *antiderivative* of $v(t)$:

$$A'(t) = v(t)$$

Or in simpler terms, integration is the reverse operation of differentiation. Hence, if we first had to integrate $v(t)$ and then differentiate the result, we would get back the original function, $v(t)$:

$$\frac{d}{dt} \int_a^b v(t)dt = v(t)$$

The fundamental theorem of calculus — Part 2

The second part of the theorem gives us a shortcut for computing the integral, without having to take the longer route of computing the limit of a Riemann sum.

It states that if the function, $v(t)$, is continuous on an interval, $[a, b]$, then:

$$\int_a^b v(t)dt = F(b) - F(a)$$

Here, $F(t)$ is any antiderivative of $v(t)$, and the integral is defined as the subtraction of the antiderivative evaluated at a and b.

Hence, the second part of the theorem computes the integral by subtracting the area under the curve between some starting point, C, and the lower limit, a, from the area between the same starting point, C, and the upper limit, b. This, effectively, calculates the area of interest between a and b.

Since the constant, C, defines the point on the x-axis at which the sweep starts, the simplest antiderivative to consider is the one with $C = 0$. Nonetheless, any antiderivative with any value of C can be used, which simply sets the starting point to a different position on the x-axis.

15.3 Integration example

Consider the function, $v(t) = x^3$. By applying the power rule, we can easily find its derivative, $v'(t) = 3x^2$. The antiderivative of $3x^2$ is again x^3: we perform the reverse operation to obtain the original function.

Now suppose that we have a different function, $g(t) = x^3 + 2$. Its derivative is also $3x^2$, and so is the derivative of yet another function, $h(t) = x^3 - 5$. Both of these functions (and other similar ones) have x^3 as their antiderivative. Hence, we specify the family of all antiderivatives of $3x^2$ by the *indefinite* integral:

$$\int 3x^2 dt = x^3 + C$$

The indefinite integral does not define the limits between which the area under the curve is being calculated. The constant, C, is included to compensate for the lack of information about the limits, or the starting point of the sweep.

If we do have knowledge of the limits, then we can simply apply the second fundamental theorem of calculus to compute the *definite* integral:

$$\int_2^3 3x^2 dt = 3^3 - 2^3 = 19$$

We can simply set C to zero, because it will not change the result in this case.

We can also find the integration using SymPy for the exact solution or apply numerical integration using NumPy for an approximation:

```
from sympy import integrate, pprint
from sympy.abc import x
import numpy as np

f = 3 * x**2
result = integrate(f, x)
print(”Antiderivative of”)
pprint(f)
print(”is”)
pprint(result)
print()

result = integrate(f, (x, 2, 3))
print(”Integration of”)
pprint(f)
print(”for x=2 to x=3 is”)
pprint(result)
print()

dx = 0.001
x = np.arange(2, 3, dx)
y = 3 * x**2
result = (y * dx).sum()
print(”Numerically using left sum:”, result)
x = np.arange(2, 3, dx) + dx
y = 3 * x**2
result = (y * dx).sum()
print(”Numerically using right sum:”, result)
x = np.arange(2, 3, dx) + dx/2
y = 3 * x**2
result = (y * dx).sum()
print(”Numerically using midpoint sum:”, result)
```

Program 15.1: Finding the integration of $3x^2$

```
Antiderivative of
   2
3·x
is
 3
x

Integration of
   2
3·x
for x=2 to x=3 is
19

Numerically using left sum: 18.99250049999912
Numerically using right sum: 19.007500499999118
Numerically using midpoint sum: 18.99999974999912
```

Output 15.1: The integration of $3x^2$

15.4 Application of integration in machine learning

We have considered the car's velocity curve, $v(t)$, as a familiar example to understand the relationship between integration and differentiation.

> But you can use this adding-up-areas-of-rectangles scheme to add up tiny bits of anything — distance, volume, or energy, for example. In other words, the area under the curve doesn't have to stand for an actual area.
>
> — Page 214, *Calculus For Dummies*, 2016.

One of the important steps of successfully applying machine learning techniques includes the choice of appropriate performance metrics. In deep learning, for instance, it is common practice to measure *precision* and *recall*.

> Precision is the fraction of detections reported by the model that were correct, while recall is the fraction of true events that were detected.
>
> — Page 423, *Deep Learning*, 2016.

It is also common practice to, then, plot the precision and recall on a Precision-Recall (PR) curve, placing the recall on the x-axis and the precision on the y-axis. It would be desirable that a classifier is characterized by both high recall and high precision, meaning that the classifier can detect many of the true events correctly. Such a good classification performance would be characterized by a higher area under the PR curve.

You can probably already tell where this is going. The area under the PR curve can, indeed, be calculated by applying integral calculus, permitting us to characterize the performance of the classifier.

15.5 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Steven Strogatz. *Infinite Powers. How Calculus Reveals the Secrets of the Universe*. Mariner Books, 2020.
https://www.amazon.com/dp/0358299284/

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Michael Spivak. *The Hitchhiker's Guide to Calculus*. American Mathematical Society, 2019.
https://www.amazon.com/dp/1470449625/

15.6 Summary

In this tutorial, you discovered the relationship between differential and integral calculus. Specifically, you learned:

- The concepts of differential and integral calculus are linked together by the fundamental theorem of calculus.
- By applying the fundamental theorem of calculus, we can compute the integral to find the area under a curve.
- In machine learning, the application of integral calculus can provide us with a metric to assess the performance of a classifier.

In the previous chapters, we learned how to do differentiation on a function with single variable. Starting from the next chapter, we will see how to do the same in a function with multiple variables.

III
Multivariate Calculus

Introduction to Multivariate Calculus

16

It is often desirable to study functions that depend on many variables.

Multivariate calculus provides us with the tools to do so by extending the concepts that we find in calculus, such as the computation of the rate of change, to multiple variables. It plays an essential role in the process of training a neural network, where the gradient is used extensively to update the model parameters.

In this tutorial, you will discover a gentle introduction to multivariate calculus. After completing this tutorial, you will know:

- ▷ A multivariate function depends on several input variables to produce an output.
- ▷ The gradient of a multivariate function is computed by finding the derivative of the function in different directions.
- ▷ Multivariate calculus is used extensively in neural networks to update the model parameters.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ Revisiting the Concept of a Function
- ▷ Derivatives of Multivariate Functions
- ▷ Application of Multivariate Calculus in Machine Learning

16.1 Revisiting the concept of a function

We have already familiarized ourselves with the concept of a function, as a rule that defines the relationship between a dependent variable and an independent variable. We have seen that a function is often represented by $y = f(x)$, where both the input (or the independent variable), x, and the output (or the dependent variable), y, are single real numbers.

Such a function that takes a single, independent variable and defines a one-to-one mapping between the input and output, is called a *univariate* function.

For example, let's say that we are attempting to forecast the weather based on the temperature alone. In this case, the weather is the dependent variable that we are trying to forecast, which is a function of the temperature as the input variable. Such a problem can, therefore, be easily framed into a univariate function.

However, let's say that we now want to base our weather forecast on the humidity level and the wind speed too, in addition to the temperature. We cannot do so by means of a univariate function, where the output depends solely on a single input.

Hence, we turn our attention to *multivariate* functions, so called because these functions can take several variables as input.

Formally, we can express a multivariate function as a mapping between several real input variables, n, to a real output:

$$f : \mathbb{R}^n \mapsto \mathbb{R}$$

For example, consider the following parabolic surface:

$$f(x, y) = x^2 + y^2$$

This is a multivariate function that takes two variables, x and y, as input, hence $n = 2$, to produce an output. We can visualize it by graphing its values for x and y between -1 and 1.

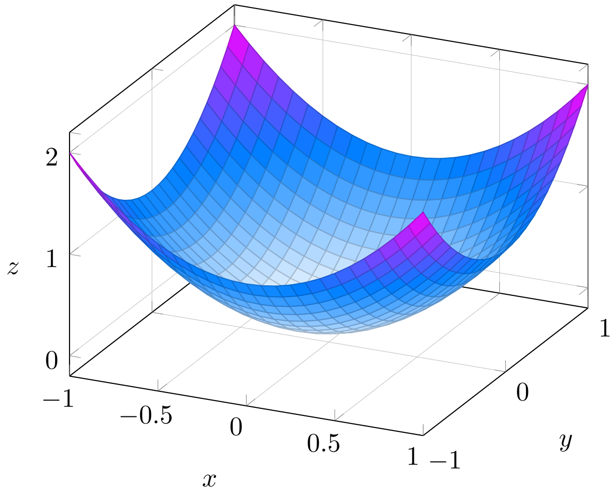


Figure 16.1: Three-dimensional plot of a parabolic surface

Similarly, we can have multivariate functions that take more variables as input. Visualizing them, however, may be difficult due to the number of dimensions involved.

We can even generalize the concept of a function further by considering functions that map multiple inputs, n, to multiple outputs, m:

$$f : \mathbb{R}^n \mapsto \mathbb{R}^m$$

These functions are more often referred to as *vector-valued* functions.

16.2 Derivatives of multivariate functions

Recall that calculus is concerned with the study of the rate of change. For some univariate function, $g(x)$, this can be achieved by computing its derivative:

$$g'(x) = \frac{dg(x)}{dx} = \lim_{h \to 0} \frac{g(x+h) - g(x)}{h}$$

> The generalization of the derivative to functions of several variables is the gradient.
>
> — Page 146, *Mathematics for Machine Learning*, 2020.

The technique to finding the gradient of a function of several variables involves varying each one of the variables at a time, while keeping the others constant. In this manner, we would be taking the *partial derivative* of our multivariate function with respect to each variable, each time.

> The gradient is then the collection of these partial derivatives.
>
> — Page 146, *Mathematics for Machine Learning*, 2020.

In order to visualize this technique better, let's start off by considering a simple univariate quadratic function of the form:

$$g(x) = x^2$$

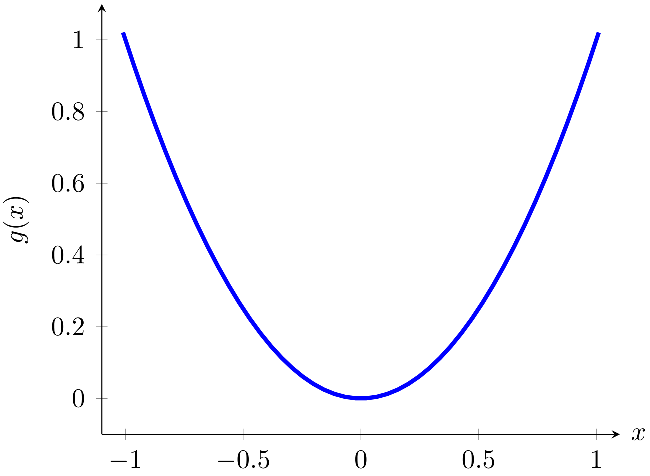


Figure 16.2: Line plot of a univariate quadratic function

Finding the derivative of this function at some point, x, requires the application of the equation for $g'(x)$ that we have defined earlier. We can, alternatively, take a shortcut by using the power rule to find that:

$$g'(x) = 2x$$

Furthermore, if we had to imagine slicing open the parabolic surface considered earlier, with a plane passing through $y = 0$, we realize that the resulting cross-section of $f(x, y)$ is the quadratic curve, $g(x) = x^2$. Hence, we can calculate the derivative (or the steepness, or *slope*) of the parabolic surface in the direction of x, by taking the derivative of $f(x, y)$ but keeping y constant. We refer to this as the *partial* derivative of $f(x, y)$ with respect to x, and denote it by ∂ to signify that there are more variables in addition to x but these are not being considered for the time being. Therefore, the partial derivative with respect to x of $f(x,y)$ is:

$$\frac{\partial}{\partial x}(x^2 + 2y^2) = g'(x) = 2x$$

We can similarly hold x constant (or, in other words, find the cross-section of the parabolic surface by slicing it with a plane passing through a constant value of x) to find the partial derivative of $f(x,y)$ with respect to y, as follows:

$$\frac{\partial}{\partial y}(x^2 + 2y^2) = 4y$$

What we have essentially done is that we have found the univariate derivative of $f(x,y)$ in each of the x and y directions. Combining the two univariate derivatives as the final step, gives us the multivariate derivative (or the gradient):

$$\frac{df}{d(x,y)} = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right] = [2x, 4y]$$

The same technique remains valid for functions of higher dimensions.

We can also find the derivatives of multivariate functions in SymPy:

```python
from sympy.abc import x, y
from sympy import diff, pprint

f = x**2 + 2 * y**2
dx = diff(f, x)
dy = diff(f, y)
print(”Derivative of”)
pprint(f)
print(”with respect to x is”)
pprint(dx)
print(”and with respect to y is”)
pprint(dy)
```

Program 16.1: Finding derivatives of $f(x) = x^2 + 2y^2$

```
Derivative of
 2      2
x  + 2⋅y
with respect to x is
2⋅x
and with respect to y is
4⋅y
```

Output 16.1: Derivatives of $f(x) = x^2 + 2y^2$

16.3 Application of multivariate calculus in machine learning

Partial derivatives are used extensively in neural networks to update the model parameters (or weights). We had seen that, in minimizing some error function, an optimization algorithm will seek to follow its gradient downhill. If this error function was univariate, and hence a function of a single independent weight, then optimizing it would simply involve computing its univariate derivative.

However, a neural network comprises many weights (each attributed to a different neuron) of which the error is a function. Hence, updating the weight values requires that the gradient of the error curve is calculated with respect to all of these weights.

This is where the application of multivariate calculus comes into play. The gradient of the error curve is calculated by finding the partial derivative of the error with respect to each weight; or in other terms, finding the derivative of the error function by keeping all weights constant except the one under consideration. This allows each weight to be updated independently of the others, to reach the goal of finding an optimal set of weights.

16.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge, 2020.
https://www.amazon.com/dp/110845514X

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.
https://amzn.to/3je8O1J

John D. Kelleher. *Deep Learning*. Illustrated edition. The MIT Press Essential Knowledge series. MIT Press, 2019.
https://www.amazon.com/dp/0262537559/

16.5 Summary

In this tutorial, you discovered a gentle introduction to multivariate calculus. Specifically, you learned:

- A multivariate function depends on several input variables to produce an output.
- The gradient of a multivariate function is computed by finding the derivative of the function in different directions.
- Multivariate calculus is used extensively in neural networks to update the model parameters.

In the nexdt chapter, we will learn more about vector-valued functions

17 Vector-Valued Functions

Vector-valued functions are often encountered in machine learning, computer graphics and computer vision algorithms. They are particularly useful for defining the parametric equations of space curves. It is important to gain a basic understanding of vector-valued functions to grasp more complex concepts.

In this tutorial, you will discover what vector-valued functions are, how to define them and some examples. After completing this tutorial, you will know:

- ▷ Definition of vector-valued functions
- ▷ Derivatives of vector-valued functions

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- ▷ Definition and examples of vector-valued functions
- ▷ Differentiating vector-valued functions

17.1 Definition of a vector-valued function

A vector-valued function is also called a vector function. It is a function with the following two properties:

1. The domain is a set of real numbers
2. The range is a set of vectors

Vector functions are, therefore, simply an extension of scalar functions, where both the domain and the range are the set of real numbers.

In this tutorial we'll consider vector functions whose range is the set of two or three dimensional vectors. Hence, such functions can be used to define a set of points in space.

Given the unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ parallel to the x, y, z-axis respectively, we can write a three dimensional vector-valued function as:

$$r(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$$

It can also be written as:

$$r(t) = \langle x(t), y(t), z(t) \rangle$$

Both the above notations are equivalent and often used in various textbooks.

Space curves and parametric equations

We defined a vector function $r(t)$ in the preceding section. For different values of t we get the corresponding (x, y, z) coordinates, defined by the functions $x(t)$, $y(t)$ and $z(t)$. The set of generated points (x, y, z), therefore, define a curve called the space curve C. The equations for $x(t)$, $y(t)$ and $z(t)$ are also called the parametric equations of the curve C.

17.2 Examples of vector functions

This section shows some examples of vector-valued functions that define space curves. All the examples are also plotted in the figure shown after the examples.

A circle

Let's start with a simple example of a vector function in 2D space:

$$r_1(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j}$$

Here the parametric equations are:

$$x(t) = \cos(t)$$

$$y(t) = \sin(t)$$

The space curve defined by the parametric equations is a circle in 2D space as shown in the figure. If we vary t from $-\pi$ to π, we'll generate all the points that lie on the circle.

A helix

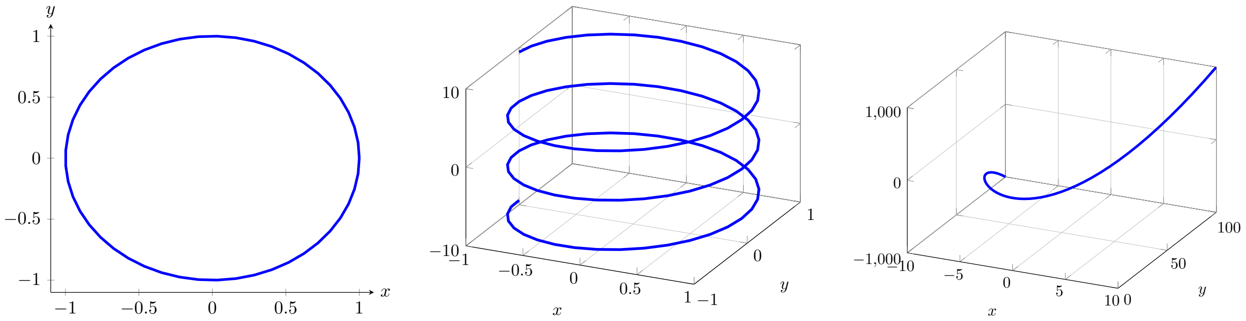
We can extend the $r_1(t)$ function of example 1.1, to easily generate a helix in 3D space. We just need to add the value along the z axis that changes with t. Hence, we have the following function:

$$r_2(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j} + t\mathbf{k}$$

A twisted cubic

We can also define a curve called the twisted cubic with an interesting shape as:

$$r_3(t) = t\mathbf{i} + t^2\mathbf{j} + t^3\mathbf{k}$$



$r_1(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j}$ $r_2(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j} + t\mathbf{k}$ $r_3(t) = t\mathbf{i} + t^2\mathbf{j} + t^3\mathbf{k}$

Figure 17.1: Parametric curves

17.3 Derivatives of vector functions

We can easily extend the idea of the derivative of a scalar function to the derivative of a vector function. As the range of a vector function is a set of vectors, its derivative is also a vector. If

$$r(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$$

then the derivative of $r(t)$ is given by $r'(t)$ computed as:

$$r'(t) = x'(t)\mathbf{i} + y'(t)\mathbf{j} + z'(t)\mathbf{k}$$

17.4 Examples of derivatives of vector functions

We can find the derivatives of the functions defined in the previous example as:

A circle

The parametric equation of a circle in 2D is given by:

$$r_1(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j}$$

Its derivative is therefore computed by computing the corresponding derivatives of $x(t)$ and $y(t)$ as shown below:

$$x'(t) = -\sin(t)$$

$$y'(t) = \cos(t)$$

This gives us:

$$r_1'(t) = x'(t)\mathbf{i} + y'(t)\mathbf{j}$$
$$r_1'(t) = -\sin(t)\mathbf{i} + \cos(t)\mathbf{j}$$

The space curve defined by the parametric equations is a circle in 2D space as shown in the figure. If we vary t from $-\pi$ to π, we'll generate all the points that lie on the circle.

A helix

Similar to the previous example, we can compute the derivative of $r_2(t)$ as:

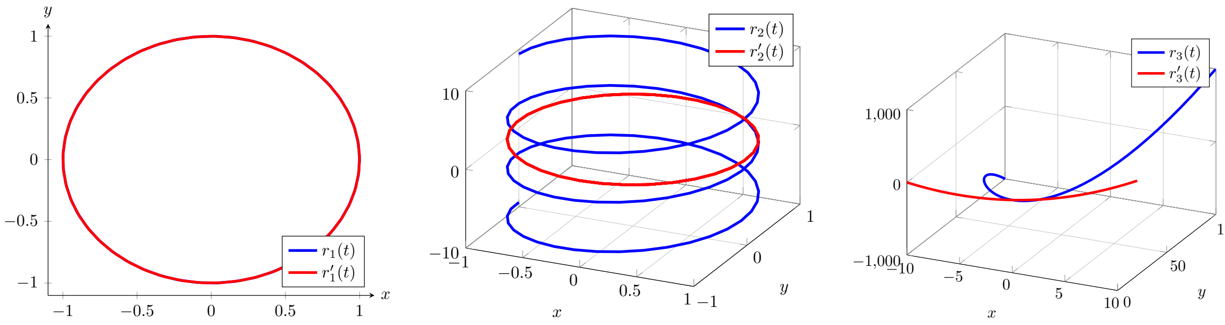
$$r_2(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j} + t\mathbf{k}$$
$$r_2'(t) = -\sin(t)\mathbf{i} + \cos(t)\mathbf{j} + \mathbf{k}$$

A twisted cubic

The derivative of $r_3(t)$ is given by:

$$r_3(t) = t\mathbf{i} + t^2\mathbf{j} + t^3\mathbf{k}$$
$$r_3'(t) = \mathbf{i} + 2t\mathbf{j} + 3t^2\mathbf{k}$$

All the above examples are shown in the figure, where the derivatives are plotted in red. Note the circle's derivative also defines a circle in space.



$r_1(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j}$

$r_1'(t) = -\sin(t)\mathbf{i} + \cos(t)\mathbf{j}$

$r_2(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j} + t\mathbf{k}$

$r_2'(t) = -\sin(t)\mathbf{i} + \cos(t)\mathbf{j} + \mathbf{k}$

$r_3(t) = t\mathbf{i} + t^2\mathbf{j} + t^3\mathbf{k}$

$r_3'(t) = \mathbf{i} + 2t\mathbf{j} + 3t^2\mathbf{k}$

Figure 17.2: Parametric functions and their derivatives

17.5 More complex examples

Once you gain a basic understanding of these functions, you can have a lot of fun defining various shapes and curves in space. Other popular examples used by the mathematical community are defined below and illustrated in the figure.

The toroidal spiral:

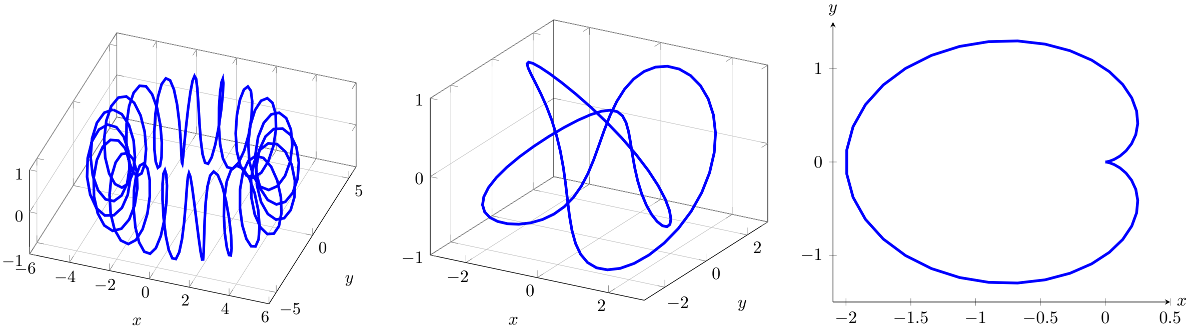
$$r_4(t) = (4 + \sin(20t))\cos(t)\mathbf{i} + (4 + \sin(20t))\sin(t)\mathbf{j} + \cos(20t)\mathbf{k}$$

The trefoil knot:

$$r_5(t) = (2 + \cos(1.5t))\cos(t)\mathbf{i} + (2 + \cos(1.5t))\sin(t)\mathbf{j} + \sin(1.5t)\mathbf{k}$$

The cardioid:

$$r_6(t) = \cos(t)(1 - \cos(t))\mathbf{i} + \sin(t)(1 - \cos(t))\mathbf{j}$$



Toroidal spiral *Trefoil knot* *Cardioid*

Figure 17.3: More complex curves

17.6 Vector-valued functions in machine learning

Vector-valued functions play an important role in machine learning algorithms. Being an extension of scalar valued functions, you would encounter them in tasks such as multi-class classification and multi-label problems. Kernel methods, an important area of machine learning, can involve computing vector-valued functions, which can be later used in multi-task learning or transfer learning.

17.7 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

17.8 Summary

In this tutorial, you discovered what vector functions are and how to differentiate them. Specifically, you learned:

- ▷ Definition of vector functions
- ▷ Parametric curves
- ▷ Differentiating vector functions

In the next chapter, we will introduce the concept of partial derivatives that is closely related to what we learned in this chapter.

18 Partial Derivatives and Gradient Vectors

Partial derivatives and gradient vectors are used very often in machine learning algorithms for finding the minimum or maximum of a function. Gradient vectors are used in the training of neural networks, logistic regression, and many other classification and regression problems.

In this tutorial, you will discover partial derivatives and the gradient vector. After completing this tutorial, you will know:

- Function of several variables
- Level sets, contours, and graphs of a function of two variables
- Partial derivatives of a function of several variables
- Gradient vector and its meaning

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- Function of several variables
- Definition of partial derivatives
- Gradient vector

18.1 A function of several variables

You can review the concept of a function and a function of several variables in Chapter 16. We'll provide more details about the functions of several variables here.

A function of several variables has the following properties:

- Its domain is a set of n-tuples given by $(x_1, x_2, x_3, \ldots, x_n)$
- Its range is a set of real numbers

For example, the following is a function of two variables ($n = 2$):

$$f_1(x, y) = x + y$$

In the above function, x and y are the independent variables. Their sum determines the value of the function. The domain of this function is the set of all points on the xy Cartesian plane. The plot of this function would require plotting in the 3D space, with two axes for input points (x, y) and the third representing the values of f. Here is another example of a function of two variables. $f_2(x, y) = x \times x + y \times y$

To keep things simple, we'll do examples of functions of two variables. Of course, in machine learning, you'll encounter functions of hundreds of variables. The concepts related to functions of two variables can be extended to those cases.

Level sets and graph of a function of two variables

The set of points in a plane, where a function $f(x, y)$ has a constant value, i.e., $f(x, y) = c$, is the level set or level curve of f.

As an example, for function f_1, all (x, y) points that satisfy the equation below define a level set for f_1:

$$x + y = 1$$

We can see that this level set has an infinite set of points, e.g., $(0, 2)$, $(1, 1)$, $(2, 0)$, etc. This level set defines a straight line in the xy plane.

In general, all level sets of f_1 define straight lines of the form (c is any real constant):

$$x + y = c$$

Similarly, for function f_2, an example of a level set is:

$$x \times x + y \times y = 1$$

We can see that any point that lies on a circle of radius 1 with center at $(0, 0)$ satisfies the above expression. Hence, this level set consists of all points that lie on this circle. Similarly, any level set of f_2 satisfies the following expression (c is any real constant ≥ 0):

$$x \times x + y \times y = c$$

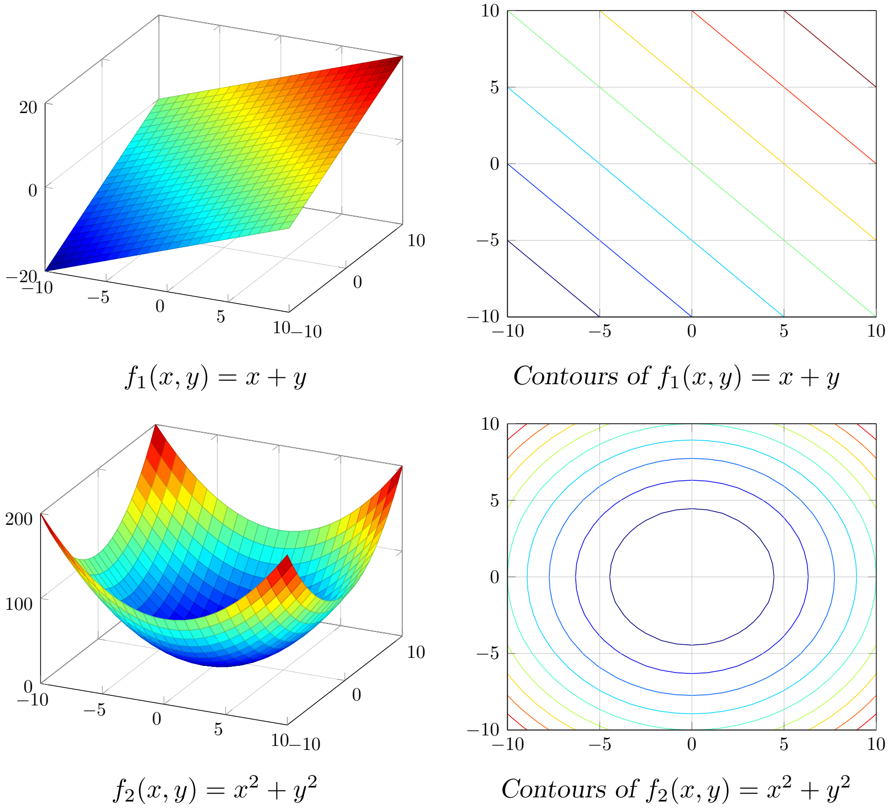
Hence, all level sets of f_2 are circles with center at $(0, 0)$, each level set having its own radius.

The graph of the function $f(x, y)$ is the set of all points $(x, y, f(x, y))$. It is also called a surface $z = f(x, y)$. The graphs of f_1 and f_2 are shown in the left side of Figure 18.1.

Contours of a function of two variables

Suppose we have a function $f(x, y)$ of two variables. If we cut the surface $z = f(x, y)$ using a plane $z = c$, then we get the set of all points that satisfy $f(x, y) = c$. The contour curve is the set of points that satisfy $f(x, y) = c$ in the plane $z = c$. This is slightly different from the level set, where the level curve is directly defined in the xy plane. However, many books treat contours and level curves as the same.

The contours of both f_1 and f_2 are shown in the right side of Figure 18.1.



$f_1(x, y) = x + y$ *Contours of* $f_1(x, y) = x + y$

$f_2(x, y) = x^2 + y^2$ *Contours of* $f_2(x, y) = x^2 + y^2$

Figure 18.1: The functions f_1 *and* f_2 *and their corresponding contours*

18.2 Partial derivatives and gradients

The partial derivative of a function f with respect to the variable x is denoted by $\partial f/\partial x$. Its expression can be determined by differentiating f with respect to x. For example, for the functions f_1 and f_2, we have:

$$\frac{\partial f_1}{\partial x} = 1$$
$$\frac{\partial f_2}{\partial x} = 2x$$

$\partial f_1/\partial x$ represents the rate of change of f_1 with respect to x. For any function $f(x, y)$, $\partial f/\partial x$ represents the rate of change of f with respect to variable x.

Similar is the case for $\partial f/\partial y$. It represents the rate of change of f with respect to y. You can look at the formal definition of partial derivatives in Chapter 16.

When we find the partial derivatives with respect to all independent variables, we end up with a vector. This vector is called the gradient vector of f denoted by $\nabla f(x, y)$. A general expression for the gradients of f_1 and f_2 are given by (here $\mathbf{i}, \mathbf{j}$ are unit vectors parallel to the coordinate axes):

$$\nabla f_1(x,y) = \frac{\partial f_1}{\partial x}\mathbf{i} + \frac{\partial f_1}{\partial y}\mathbf{j} = \mathbf{i} + \mathbf{j}$$

$$\nabla f_2(x,y) = \frac{\partial f_2}{\partial x}\mathbf{i} + \frac{\partial f_2}{\partial y}\mathbf{j} = 2x\mathbf{i} + 2y\mathbf{j}$$

From the general expression of the gradient, we can evaluate the gradient at different points in space. In case of f_1, the gradient vector is a constant, i.e.,

$$\mathbf{i} + \mathbf{j}$$

No matter where we are in the three dimensional space, the direction and magnitude of the gradient vector remains unchanged.

For the function f_2, $\nabla f_2(x,y)$ changes with values of (x,y). For example, at $(1,1)$ and $(2,1)$, the gradient of f_2 is given by the following vectors:

$$\nabla f_2(1,1) = 2\mathbf{i} + 2\mathbf{j}$$

$$\nabla f_2(2,1) = 4\mathbf{i} + 2\mathbf{j}$$

We can reproduce the same result by first finding the partial derivatives in SymPy and then evaluating for its numerical value at the point (x,y):

```
from sympy.abc import x, y
from sympy import diff, pprint

f2 = x**2 + y**2
df2dx = diff(f2, x)
df2dy = diff(f2, y)
print(”Partial derivative of”)
pprint(f2)
print(”with respect to x is”)
pprint(df2dx)
print(”and with respect to y is”)
pprint(df2dy)
print(”gradient at (1,1) is ({},{})”.format(df2dx.subs([(x,1),(y,1)]),
                                        df2dy.subs([(x,1),(y,1)])))
print(”gradient at (2,1) is ({},{})”.format(df2dx.subs([(x,2),(y,1)]),
                                        df2dy.subs([(x,2),(y,1)])))
```

Program 18.1: Finding $\nabla f_2(1,1)$ and $\nabla f_2(2,1)$

```
Partial derivative of
 2    2
x  + y
with respect to x is
2⋅x
and with respect to y is
2⋅y
gradient at (1,1) is (2,2)
gradient at (2,1) is (4,2)
```

Output 18.1: Finding $\nabla f_2(1,1)$ and $\nabla f_2(2,1)$

18.3 What does the gradient vector at a point indicate?

The gradient vector of a function of several variables at any point denotes the direction of the maximum rate of change.

We can relate the gradient vector to the tangent line. Suppose we are standing at a point in space and we come up with a rule that tells us to walk along the tangent to the contour at that point. Then it means wherever we are, we find the tangent line to the contour at that point and walk along it. If we walk following this rule, we'll end up walking along the contour of f. The function's value will never change as the function's value is constant on the contour of f.

The gradient vector, on the other hand, is normal to the tangent line and points to the direction of maximum rate of increase. If we walk along the direction of the gradient, we'll start encountering the next point where the function's value would be greater than the previous one.

The positive direction of the gradient indicates the direction of the maximum rate of increase, whereas the negative direction indicates the direction of the maximum rate of decrease. The following figure shows the positive direction of the gradient vector at different points of the contours of function f_2. The direction of the positive gradient is indicated by the red arrow. The tangent line to a contour is shown in green.

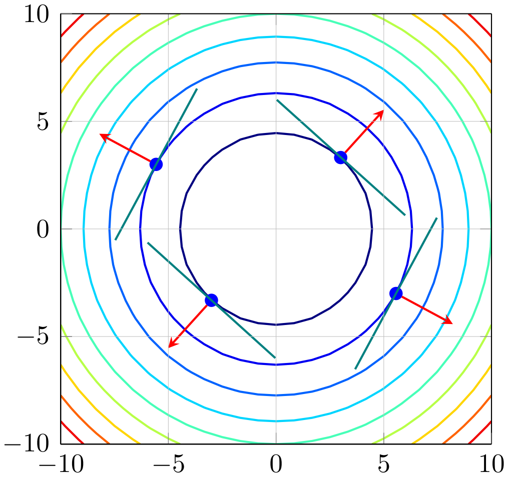


Figure 18.2: The contours and the direction of gradient vectors: Gradient vectors at various points shown with red arrows, tangent to the contour is in green

We can verify that the gradient vector we found before is indeed in the direction of the maximum rate of change by exhaustive search. In the following code, we search from 0° (i.e., positive direction along the x-axis) to 360° at 5° increments. The unit vector at the direction of angle θ is $\sin\theta\mathbf{i} + \cos\theta\mathbf{j}$, so for a small step, the corresponding size along the x- and y-axes can be found as follows:

```
import numpy as np

angle = 45     # angle in degree
```

```
step = 0.001  # size of a small step

rad = angle * np.pi / 180    # convert degree angle into radians
dx = np.sin(rad) * step      # size of small step along x-axis
dy = np.cos(rad) * step      # size of small step along y-axis
```

Program 18.2: Size of small steps along x- and y-axes at an angle

Then, we can find the derivative of $f(x, y)$ on that direction using the first principle:

```
import numpy as np

def f(x, y):
    return x**2 + y**2

x, y = 1, 1
step = 0.001
angles = np.arange(0, 360, 5) # 0 to 360 degrees at 5-degree steps
maxdf, maxangle = -np.inf, 0
for angle in angles:
    rad = angle * np.pi / 180 # convert degree to radian
    dx, dy = np.sin(rad)*step, np.cos(rad)*step
    df = (f(x+dx, y+dy) - f(x,y))/step
    if df > maxdf:
        maxdf, maxangle = df, angle
    print(f”Rate of change at {angle} degrees = {df}”)

dx, dy = np.sin(maxangle*np.pi/180), np.cos(maxangle*np.pi/180)
gradx, grady = dx*maxdf, dy*maxdf
print(f”Max rate of change at {maxangle} degrees”)
print(f”Gradient vector at ({x},{y}) is ({dx*maxdf},{dy*maxdf})”)
```

Program 18.3: Finding the direction of maximum rate of change

```
Rate of change at 0 degrees = 2.0009999999999195
Rate of change at 5 degrees = 2.1677008816789467
Rate of change at 10 degrees = 2.3179118613576577
Rate of change at 15 degrees = 2.4504897427832795
Rate of change at 20 degrees = 2.564425528222891
Rate of change at 25 degrees = 2.658852097554565
Rate of change at 30 degrees = 2.7330508075689153
Rate of change at 35 degrees = 2.786456961280326
Rate of change at 40 degrees = 2.8186641056113793
Rate of change at 45 degrees = 2.829427124746431
Rate of change at 50 degrees = 2.8186641056113793
Rate of change at 55 degrees = 2.786456961280326
Rate of change at 60 degrees = 2.7330508075689153
...
Rate of change at 345 degrees = 1.4152135623732853
Rate of change at 350 degrees = 1.62331915069025
Rate of change at 355 degrees = 1.8190779106879162
Max rate of change at 45 degrees
Gradient vector at (1,1) is (2.0007071067813564,2.000707106781357)
```

Output 18.2: Finding the direction of maximum rate of change

18.4 Gradient vectors in machine learning

The gradient vector is very important and used frequently in machine learning algorithms. In classification and regression problems, we normally define the mean square error function. Following the negative direction of the gradient of this function will lead us to finding the point where this function has a minimum value.

Similar is the case for functions, where maximizing them leads to achieving maximum accuracy. In this case we'll follow the direction of the maximum rate of increase of this function or the positive direction of the gradient vector.

18.5 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

18.6 Summary

In this tutorial, you discovered what are functions of several variables, partial derivatives, and the gradient vector. Specifically, you learned:

- ▷ Function of several variables
    - ○ Contours of a function of several variables
    - ○ Level sets of a function of several variables
- ▷ Partial derivatives of a function of several variables
- ▷ Gradient vector and its meaning

In the next chapter, we will learn about the derivative of a derivative.

Higher-Order Derivatives

19

Higher-order derivatives can capture information about a function that first-order derivatives on their own cannot capture.

First-order derivatives can capture important information, such as the rate of change, but on their own they cannot distinguish between local minima or maxima, where the rate of change is zero for both. Several optimization algorithms address this limitation by exploiting the use of higher-order derivatives, such as in Newton's method where the second-order derivatives are used to reach the local minimum of an optimization function.

In this tutorial, you will discover how to compute higher-order univariate and multivariate derivatives.

After completing this tutorial, you will know:

- ▷ How to compute the higher-order derivatives of univariate functions.
- ▷ How to compute the higher-order derivatives of multivariate functions.
- ▷ How the second-order derivatives can be exploited in machine learning by second-order optimization algorithms.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ Higher-Order Derivatives of Univariate Functions
- ▷ Higher-Order Derivatives of Multivariate Functions
- ▷ Application in Machine Learning

19.1 Higher-order derivatives of univariate functions

In addition to first-order derivatives, which we have seen can provide us with important information about a function, such as its instantaneous rate of change, higher-order derivatives can also be equally useful. For example, the second derivative can measure the acceleration of a moving object, or it can help an optimization algorithm distinguish between a local maximum and a local minimum.

Computing higher-order (second, third or higher) derivatives of univariate functions is not that difficult.

> The second derivative of a function is just the derivative of its first derivative. The third derivative is the derivative of the second derivative, the fourth derivative is the derivative of the third, and so on.
>
> — Page 147, *Calculus For Dummies*, 2016.

Hence, computing higher-order derivatives simply involves differentiating the function repeatedly. In order to do so, we can simply apply our knowledge of the power rule. Let's consider the function, $f(x) = x^3 + 2x^2 - 4x + 1$, as an example. Then:

- First derivative: $f'(x) = 3x^2 + 4x - 4$
- Second derivative: $f''(x) = 6x + 4$
- Third derivative: $f'''(x) = 6$
- Fourth derivative: $f^{(4)}(x) = 0$
- Fifth derivative: $f^{(5)}(x) = 0$, etc.

What we have done here is that we have first applied the power rule to $f(x)$ to obtain its first derivative, $f'(x)$, then applied the power rule to the first derivative in order to obtain the second, and so on. The derivative will, eventually, go to zero as differentiation is applied repeatedly.

In SymPy, the higher-order derivatives can be found using the same `diff()` function:

```
from sympy.abc import x
from sympy import diff, pprint

f = x**3 + 2*x**2 - 4*x + 1
df1 = diff(f, x)
df2 = diff(f, x, x)
df3 = diff(f, x, x, x)
df4 = diff(f, x, x, x, x)
df5 = diff(f, x, x, x, x, x)
print(”Function”)
pprint(f)
print(”First derivative”)
pprint(df1)
print(”Second derivative”)
pprint(df2)
print(”Third derivative”)
pprint(df3)
print(”Fourth derivative”)
pprint(df4)
print(”Fifth derivative”)
pprint(df5)
```

Program 19.1: Finding higher-order derivatives

```
Function
 3      2
x  + 2⋅x  - 4⋅x + 1
First derivative
   2
3⋅x  + 4⋅x - 4
Second derivative
2⋅(3⋅x + 2)
Third derivative
6
Fourth derivative
0
Fifth derivative
0
```

Output 19.1: Higher-order derivatives of $f(x)$

The application of the product and quotient rules also remains valid in obtaining higher-order derivatives, but their computation can become messier and messier as the order increases. The general Leibniz rule simplifies the task in this aspect, by generalizing the product rule to:

$$(fg)^{(n)} = \sum_{k=0}^{n} \binom{n}{k} f^{(n-k)} g^{(k)} = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} f^{(n-k)} g^{(k)}$$

Here, the term, $\frac{n!}{k!(n-k)!}$, is the binomial coefficient from the binomial theorem, while $f^{(k)}$ and $g^{(k)}$ denote the k-th derivative of the functions, f and g, respectively.

Therefore, finding the first and second derivatives (and, hence, substituting for $n = 1$ and $n = 2$, respectively), by the general Leibniz rule, gives us:

$$(fg)^{(1)} = (fg)' = f^{(1)}g + fg^{(1)}$$

$$(fg)^{(2)} = (fg)'' = f^{(2)}g + 2f^{(1)}g^{(1)} + fg^{(2)}$$

Notice the familiar first derivative as defined by the product rule. The Leibniz rule can also be used to find higher-order derivatives of rational functions, since the quotient can be effectively expressed into a product of the form, fg^{-1}

19.2 Higher-order derivatives of multivariate functions

The definition of higher-order partial derivatives of multivariate functions is analogous to the univariate case: the n-th order partial derivative for $n > 1$, is computed as the partial derivative of the $(n-1)$-th order partial derivative. For example, taking the second partial derivative of a function with two variables results in four, second partial derivatives: two *own* partial derivatives, f_{xx} and f_{yy}, and two cross partial derivatives, f_{xy} and f_{yx}.

> “ To take a “derivative,” we must take a partial derivative with respect to x or y, and there are four ways to do it: x then x, x then y, y then x, y then y. ”
>
> — Page 371, *Single and Multivariable Calculus*, 2020.

Let's consider the multivariate function, $f(x, y) = x^2 + 3xy + 4y^2$, for which we would like to find the second partial derivatives. The process starts with finding its first-order partial derivatives, first:

$$\frac{\partial f}{\partial x} = f_x = 2x + 3y$$
$$\frac{\partial f}{\partial y} = f_y = 3x + 8y$$

The four, second-order partial derivatives are then found by repeating the process of finding the partial derivatives, of the partial derivatives. The *own* partial derivatives are the most straightforward to find, since we simply repeat the partial differentiation process, with respect to either x or y, a second time:

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}(2x + 3y) = f_{xx} = 2$$
$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}(3x + 8y) = f_{yy} = 8$$

The cross partial derivative of the previously found f_x (that is, the partial derivative with respect to x is found by taking the partial derivative of the result with respect to y, giving us f_{xy}. Similarly, taking the partial derivative of f_y with respect to x, gives us f_{yx}:

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial y}(2x + 3y) = f_{xy} = 3$$
$$\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial x}(3x + 8y) = f_{yx} = 3$$

It is not by accident that the cross partial derivatives give the same result. This is defined by Clairaut's theorem, which states that as long as the cross partial derivatives are continuous, then they are equal. The above can be verified using SymPy as follows:

```
from sympy.abc import x, y
from sympy import diff, pprint

f = x**2 + 3*x*y + 4*y**2
fx = diff(f, x)
fy = diff(f, y)
fxx = diff(fx, x)
fyy = diff(fy, y)
fxy = diff(fx, y)
fyx = diff(fy, x)
print(”Function”)
pprint(f)
print(”f_x =”)
pprint(fx)
print(”f_y =”)
pprint(fy)
print(”f_xx =”)
```

```
pprint(fxx)
print("f_yy =")
pprint(fyy)
print("f_xy =")
pprint(fxy)
print("f_yx =")
pprint(fyx)
```

Program 19.2: Finding higher-order derivatives of multivariate function $f(x, y)$

```
Function
 2              2
x  + 3·x·y + 4·y
f_x =
2·x + 3·y
f_y =
3·x + 8·y
f_xx =
2
f_yy =
8
f_xy =
3
f_yx =
3
```

Output 19.2: Higher-order derivatives of $f(x, y)$

19.3 Application in machine learning

In machine learning, it is the second-order derivative that is mostly used. We had previously mentioned that the second derivative can provide us with information that the first derivative on its own cannot capture. Specifically, it can tell us whether a critical point is a local minimum or maximum (based on whether the second derivative is greater or smaller than zero, respectively), for which the first derivative would, otherwise, be zero in both cases.

There are several *second-order* optimization algorithms that leverage this information, one of which is Newton's method.

> Second-order information, on the other hand, allows us to make a quadratic approximation of the objective function and approximate the right step size to reach a local minimum ...
>
> — Page 87, *Algorithms for Optimization*, 2019.

In the univariate case, Newton's method uses a second-order Taylor series expansion to perform the quadratic approximation around some point on the objective function. The update rule for Newton's method, which is obtained by setting the derivative to zero and solving for the root, involves a division operation by the second derivative. If Newton's method is extended to multivariate optimization, the derivative is replaced by the gradient, while the reciprocal of the second derivative is replaced with the inverse of the Hessian matrix.

We shall be covering the Hessian and Taylor Series approximations, which leverage the use of higher-order derivatives, in separate tutorials.

19.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.
https://amzn.to/3je8O1J

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

19.5 Summary

In this tutorial, you discovered how to compute higher-order univariate and multivariate derivatives. Specifically, you learned:

- How to compute the higher-order derivatives of univariate functions.
- How to compute the higher-order derivatives of multivariate functions.
- How the second-order derivatives can be exploited in machine learning by second-order optimization algorithms.

In the next chapter, we will study chain rule. With it, we can find the derivatives of a function with respect to implicit variables.

20 The Chain Rule

The chain rule allows us to find the derivative of composite functions.

It is computed extensively by the backpropagation algorithm, in order to train feedforward neural networks. By applying the chain rule in an efficient manner while following a specific order of operations, the backpropagation algorithm calculates the error gradient of the loss function with respect to each weight of the network.

In this tutorial, you will discover the chain rule of calculus for univariate and multivariate functions.

After completing this tutorial, you will know:

- A composite function is the combination of two (or more) functions.
- The chain rule allows us to find the derivative of a composite function.
- The chain rule can be generalized to multivariate functions, and represented by a tree diagram.
- The chain rule is applied extensively by the backpropagation algorithm in order to calculate the error gradient of the loss function with respect to each weight.

Let's get started.

Overview

This tutorial is divided into four parts; they are:

- Composite Functions
- The Chain Rule
- The Generalized Chain Rule
- Application in Machine Learning

20.1 Prerequisites

For this tutorial, we assume that you already know what are:

- ▷ Multivariate functions (Chapter 16)
- ▷ The power rule (Chapter 11)
- ▷ The gradient of a function (Chapter 18)

20.2 Composite functions

We have, so far, met functions of single and multiple variables (so called, *univariate* and *multivariate* functions, respectively). We shall now extend both to their *composite* forms. We will, eventually, see how to apply the chain rule in order to find their derivative, but more on this shortly.

> “ A composite function is the combination of two functions. ”
>
> — Page 49, *Calculus For Dummies*, 2016.

Consider two functions of a single independent variable, $f(x) = 2x - 1$ and $g(x) = x^3$. Their composite function can be defined as follows:

$$h = g(f(x))$$

In this operation, g is a function of f. This means that g is applied to the result of applying the function, f, to x, producing h.

Let's consider a concrete example using the functions specified above to understand this better. Suppose that $f(x)$ and $g(x)$ are two systems in cascade, receiving an input $x = 5$:

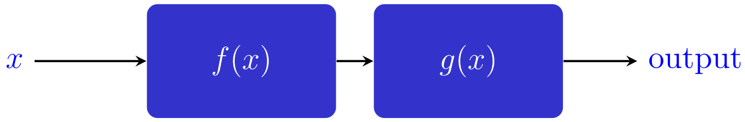


Figure 20.1: Two systems in cascade representing a composite function

Since $f(x)$ is the first system in the cascade (because it is the inner function in the composite), its output is worked out first:

$$f(5) = (2 \times 5) - 1 = 9$$

This result is then passed on as input to $g(x)$, the second system in the cascade (because it is the outer function in the composite) to produce the net result of the composite function:

$$g(9) = 9^3 = 729$$

We could have, alternatively, computed the net result at one go, if we had performed the following computation:

$$h = g(f(x)) = (2x - 1)^3 = 729$$

The composition of functions can also be considered as a *chaining* process, to use a more familiar term, where the output of one function feeds into the next one in the chain.

> “ With composite functions, the order matters. ”
>
> — Page 49, *Calculus For Dummies*, 2016.

Keep in mind that the composition of functions is a *non-commutative* process, which means that swapping the order of $f(x)$ and $g(x)$ in the cascade (or chain) does not produce the same results. Hence:

$$g(f(x)) \neq f(g(x))$$

The composition of functions can also be extended to the multivariate case:

$$h = g(r, s, t) = g(r(x, y), s(x, y), t(x, y)) = g(\mathbf{f}(x, y))$$

Here, $\mathbf{f}(x, y)$ is a vector-valued function of two independent variables (or inputs), x and y. It is made up of three components (for this particular example) that are $r(x, y)$, $s(x, y)$, $t(x, y)$, and which are also known as the *component* functions of $\mathbf{f}$. This means that $\mathbf{f}(x, y)$ will map two inputs to three outputs, and will then feed these three outputs into the consecutive system in the chain, $g(r, s, t)$, to produce h.

20.3 The chain rule

The chain rule allows us to find the derivative of a composite function.

Let's first define how the chain rule differentiates a composite function, and then break it into its separate components to understand it better. If we had to consider again the composite function, $h = g(f(x))$, then its derivative as given by the chain rule is:

$$\frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx}$$

Here, u is the output of the inner function f (hence, $u = f(x)$), which is then fed as input to the next function g to produce h (hence, $h = g(u)$). Notice, therefore, how the chain rule relates the net output, h, to the input, x, through an *intermediate variable*, u.

Recall that the composite function is defined as follows:

$$h(x) = g(f(x)) = (2x - 1)^3$$

The first component of the chain rule, $\frac{dh}{du}$, tells us to start by finding the derivative of the outer part of the composite function, while ignoring whatever is inside. For this purpose, we shall apply the power rule:

$$((2x - 1)^3)' = 3(2x - 1)^2$$

The result is then multiplied to the second component of the chain rule, $\frac{du}{dx}$, which is the derivative of the inner part of the composite function, this time ignoring whatever is outside:

$$((2x - 1)')^3 = 2$$

The derivative of the composite function as defined by the chain rule is, then, the following:

$$h' = 3(2x - 1)^2 \times 2 = 6(2x - 1)^2$$

We have, hereby, considered a simple example, but the concept of applying the chain rule to more complicated functions remains the same.

20.4 The generalized chain rule

We can generalize the chain rule beyond the univariate case.

Consider the case where $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$, which means that the inner function, f, maps m inputs to n outputs, while the outer function, g, receives n inputs to produce an output, h. For $i = 1, \ldots, m$ the generalized chain rule states:

$$\frac{\partial h}{\partial x_i} = \frac{\partial h}{\partial u_1} \cdot \frac{\partial u_1}{\partial x_i} + \frac{\partial h}{\partial u_2} \cdot \frac{\partial u_2}{\partial x_i} + \cdots + \frac{\partial h}{\partial u_n} \cdot \frac{\partial u_n}{\partial x_i}$$

or in its more compact form, for $j = 1, \ldots, n$:

$$\frac{\partial h}{\partial x_i} = \sum_j \frac{\partial h}{\partial u_j} \cdot \frac{\partial u_j}{\partial x_i}$$

Recall that we employ the use of partial derivatives when we are finding the gradient of a function of multiple variables.

We can also visualize the workings of the chain rule by a tree diagram. Suppose that we have a composite function of two independent variables, x_1 and x_2, defined as follows:

$$h = g(\mathbf{f}(x_1, x_2)) = g(u_1(x_1, x_2), u_2(x_1, x_2))$$

Here, u_1 and u_2 act as the intermediate variables. Its tree diagram would be represented as follows:

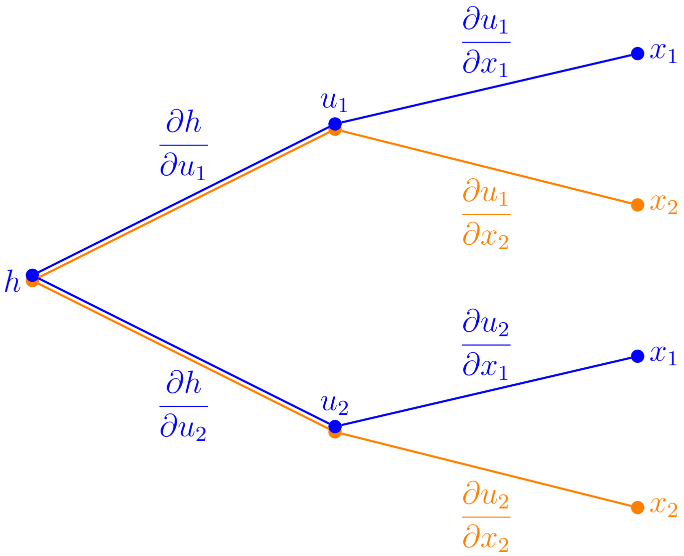

$\frac{\partial u_1}{\partial x_1}$
x_1
u_1
$\frac{\partial h}{\partial u_1}$
$\frac{\partial u_1}{\partial x_2}$
x_2
h
$\frac{\partial u_2}{\partial x_1}$
x_1
$\frac{\partial h}{\partial u_2}$
u_2
$\frac{\partial u_2}{\partial x_2}$
x_2

Figure 20.2: Representing the chain rule by a tree diagram

In order to derive the formula for each of the inputs, x_1 and x_2, we can start from the left hand side of the tree diagram, and follow its branches rightwards. In this manner, we find that we form the following two formulae (the branches being summed up have been color coded for simplicity):

$$\frac{\partial h}{\partial x_1} = \frac{\partial h}{\partial u_1} \cdot \frac{\partial u_1}{\partial x_1} + \frac{\partial h}{\partial u_2} \cdot \frac{\partial u_2}{\partial x_1}$$

$$\frac{\partial h}{\partial x_2} = \frac{\partial h}{\partial u_1} \cdot \frac{\partial u_1}{\partial x_2} + \frac{\partial h}{\partial u_2} \cdot \frac{\partial u_2}{\partial x_2}$$

Notice how the chain rule relates the net output, h, to each of the inputs, x_i, through the intermediate variables, u_j. This is a concept that the backpropagation algorithm applies extensively to optimize the weights of a neural network.

20.5 The chain rule on univariate functions

We have already discovered the chain rule for univariate and multivariate functions, but we have only seen a few simple examples so far. Let's see a few more challenging ones here. We will be starting with univariate functions first, and then apply what we learn to multivariate functions.

Example 1

Let's raise the bar a little by considering the following composite function:

$$h = g(f(x)) = \sqrt{x^2 - 10}$$

We can separate the composite function into the inner function, $u = f(x) = x^2 - 10$, and the outer function, $g(u) = \sqrt{u} = u^{1/2}$. The output of the inner function is denoted by the intermediate variable, u, and its value will be fed into the input of the outer function.

The first step is to find the derivative of the outer part of the composite function, while ignoring whatever is inside. For this purpose, we can apply the power rule:

$$\frac{dh}{du} = \frac{1}{2}u^{-\frac{1}{2}} = \frac{1}{2}(x^2 - 10)^{-\frac{1}{2}}$$

The next step is to find the derivative of the inner part of the composite function, this time ignoring whatever is outside. We can apply the power rule here too:

$$\frac{du}{dx} = 2x$$

Putting the two parts together and simplifying, we have:

$$\frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx} = \frac{1}{2}(x^2 - 10)^{-\frac{1}{2}}(2x) = \frac{x}{\sqrt{x^2 - 10}}$$

We can verify this result with SymPy:

```python
from sympy.abc import x, y
from sympy import diff, sqrt, pprint

f = x**2 - 10
g = sqrt(f)
result = diff(g, x)
print("Function")
pprint(g)
print("has derivative")
pprint(result)
```

Program 20.1: Find derivative of $g(f(x))$

```
Function
   _________
  ╱  2
╲╱  x  - 10
has derivative with respect to x
     x
────────────
   _________
  ╱  2
╲╱  x  - 10
```

Output 20.1: Derivative of $g(f(x))$

Example 2

Let's repeat the procedure, this time with a different composite function:

$$h = \cos(x^3 - 1)$$

We will again use, u, the output of the inner function, as our intermediate variable. The outer function in this case is, $\cos x$. Finding its derivative, again ignoring the inside, gives us:

$$\frac{dh}{du} = (\cos(x)^3 - 1))' = -\sin(x^3 - 1)$$

The inner function is, $x^3 - 1$ Hence, its derivative becomes:

$$\frac{du}{dx} = (x^3 - 1)' = 3x^2$$

Putting the two parts together, we obtain the derivative of the composite function:

$$\frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx} = -3x^2 \sin(x^3 - 1)$$

We can verify this result with SymPy:

```
from sympy.abc import x, y
from sympy import diff, cos, pprint

u = x**3 - 1
h = cos(u)
result = diff(h, x)
print(”Function”)
pprint(h)
print(”has derivative”)
pprint(result)
```

Program 20.2: Find derivative of $h = \cos(x^3 - 1)$

```
Function
   ⎛ 3    ⎞
cos⎝x  - 1⎠
has derivative
    2    ⎛ 3    ⎞
-3⋅x ⋅sin⎝x  - 1⎠
```

Output 20.2: Derivative of $h = \cos(x^3 - 1)$

Example 3

Let’s now raise the bar a little further by considering a more challenging composite function:

$$h = \cos(x\sqrt{x^2 - 10})$$

If we observe this closely, we realize that not only do we have nested functions for which we will need to apply the chain rule multiple times, but we also have a product to which we will need to apply the product rule.

We find that the outermost function is a cosine. In finding its derivative by the chain rule, we shall be using the intermediate variable, u:

$$\frac{dh}{du} = (\cos(x\sqrt{x^2 - 10}))' = -\sin(x\sqrt{x^2 - 10})$$

Inside the cosine, we have the product, $x\sqrt{x^2 - 10}$, to which we will be applying the product rule to find its derivative (notice that we are always moving from the outside to the inside, in order to discover the operation that needs to be tackled next):

$$\frac{du}{dx} = (x\sqrt{x^2 - 10})' = \sqrt{x^2 - 10} + x(\sqrt{x^2 - 10})'$$

One of the components in the resulting term is, $(\sqrt{x^2 - 10})'$, to which we shall be applying the chain rule again. Indeed, we have already done so above, and hence we can simply re-utilise the result:

$$(\sqrt{x^2 - 10})' = x(x^2 - 10)^{-\frac{1}{2}}$$

Putting all the parts together, we obtain the derivative of the composite function:

$$\frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx} = -\sin(x\sqrt{x - 10}) \cdot \left(\sqrt{x^2 - 10} + \frac{x^2}{\sqrt{x^2 - 10}}\right)$$

This can be simplified further into:

$$\frac{dh}{dx} = -\sin(x\sqrt{x-10}) \cdot \left(\frac{2x^2-10}{\sqrt{x^2-10}}\right)$$

We can verify this result with SymPy:

```python
from sympy.abc import x, y
from sympy import diff, sqrt, cos, simplify, pprint

u = x * sqrt(x**2 - 10)
h = cos(u)
result = diff(h, x)
print("Function")
pprint(h)
print("has derivative")
pprint(simplify(result))
```

Program 20.3: Find derivative of $h = \cos(x\sqrt{x^2-10})$

```
Function
   ⎛    __________⎞
   ⎜   ╱  2       ⎟
cos⎝x⋅╲╱  x  - 10 ⎠
has derivative
              ⎛    __________⎞
  ⎛     2⎞    ⎜   ╱  2       ⎟
2⋅⎝5 - x ⎠⋅sin⎝x⋅╲╱  x  - 10 ⎠
──────────────────────────────
           __________
          ╱  2
         ╲╱  x  - 10
```

Output 20.3: Derivative of $h = \cos(x\sqrt{x^2-10})$

20.6 The chain rule on multivariate functions

Example 4

Suppose that we are now presented by a multivariate function of two independent variables, s and t, with each of these variables being dependent on another two independent variables, x and y:

$$h = g(s,t) = s^2 + t^3$$

Where the functions, $s = xy$ and $t = 2x - y$.

Implementing the chain rule here requires the computation of partial derivatives, since we are working with multiple independent variables. Furthermore, s and t will also act as our intermediate variables. The formulae that we will be working with, defined with respect to each input, are the following:

$$\begin{aligned}\frac{\partial h}{\partial x} &= \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial x} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial x} \\ \frac{\partial h}{\partial y} &= \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial y} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial y}\end{aligned}$$

From these formulae, we can see that we will need to find six different partial derivatives:

$$\begin{aligned}\frac{\partial h}{\partial s} &= 2s & \qquad \frac{\partial s}{\partial x} &= y \\ \frac{\partial h}{\partial t} &= 3t^2 & \qquad \frac{\partial t}{\partial x} &= 2 \\ \frac{\partial s}{\partial y} &= x & \qquad \frac{\partial t}{\partial y} &= -1\end{aligned}$$

We can now proceed to substitute these terms in the formulae for $\frac{\partial h}{\partial x}$ and $\frac{\partial h}{\partial y}$:

$$\begin{aligned}\frac{\partial h}{\partial x} &= \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial x} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial x} = 2sy + 6t^2 \\ \frac{\partial h}{\partial y} &= \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial y} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial y} = 2sx - 3t^2\end{aligned}$$

And subsequently substitute for s and t to find the derivatives:

$$\begin{aligned}\frac{\partial h}{\partial x} &= 2(xy)y + 6(2x-y)^2 = 2xy^2 + 24x^2 - 24xy + 6y^2 \\ \frac{\partial h}{\partial y} &= 2(xy)x - 3(2x-y)^2 = 2x^2y - 12x^2 - 12xy - 3y^2\end{aligned}$$

We can verify this result with SymPy:

```python
from sympy.abc import x, y
from sympy import diff, pprint

s = x*y
t = 2*x - y
h = s**2 + t**3
dhdx = diff(h, x)
dhdy = diff(h, y)
print(”Function”)
pprint(h)
print(”Derivative with respect to x”)
pprint(dhdx)
print(”Derivative with respect to y”)
pprint(dhdy)
```

Program 20.4: Find derivatives of $h = g(s,t)$

```
Function
 2  2           3
x ·y  + (2·x − y)
Derivative with respect to x
     2            2
2·x·y  + 6·(2·x − y)
Derivative with respect to y
   2              2
2·x ·y − 3·(2·x − y)
```

Output 20.4: Derivative of $h = g(s, t)$

Example 5

Let's repeat this again, this time with a multivariate function of three independent variables, r, s and t, with each of these variables being dependent on another two independent variables, x and y:

$$h = g(r, s, t) = r^2 - rs + t^3$$

Where the functions, $r = x\cos y$, $s = xe^y$, and $t = x + y$.

This time round, r, s and t will act as our intermediate variables. The formulae that we will be working with, defined with respect to each input, are the following:

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial r} \cdot \frac{\partial r}{\partial x} + \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial x} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial x}$$

$$\frac{\partial h}{\partial y} = \frac{\partial h}{\partial r} \cdot \frac{\partial r}{\partial y} + \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial y} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial y}$$

From these formulae, we can see that we will now need to find nine different partial derivatives:

$$\frac{\partial h}{\partial r} = 2r - s \qquad \frac{\partial r}{\partial x} = \cos y \qquad \frac{\partial h}{\partial s} = -r$$

$$\frac{\partial s}{\partial x} = e^y \qquad \frac{\partial h}{\partial t} = 3t^2 \qquad \frac{\partial t}{\partial x} = 1$$

$$\frac{\partial r}{\partial y} = -x\sin y \qquad \frac{\partial s}{\partial y} = xe^y \qquad \frac{\partial t}{\partial y} = 1$$

Again, we proceed to substitute these terms in the formulae for $\frac{\partial h}{\partial x}$ and $\frac{\partial h}{\partial y}$:

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial r} \cdot \frac{\partial r}{\partial x} + \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial x} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial x} = (2r - s)\cos y - re^y + 3t^2$$

$$\frac{\partial h}{\partial y} = \frac{\partial h}{\partial r} \cdot \frac{\partial r}{\partial y} + \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial y} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial y} = (2r - s)(-x\sin y) - r(xe^y) + 3t^2$$

And subsequently substitute for r, s and t to find the derivatives:

$$\frac{\partial h}{\partial x} = (2x\cos y - xe^y)\cos y - (x\cos y)e^y + 3(x + y)^2$$

$$\frac{\partial h}{\partial y} = (2x\cos y - xe^y)(-x\sin y) - (x\cos y)(xe^y) + 3(x + y)^2$$

Which may be simplified a little further (hint: apply the trigonometric identity $2\sin y \cos y = \sin 2y$ to $\partial h/\partial y$):

$$\frac{\partial h}{\partial x} = 2x\cos y(\cos y - e^y) + 3(x+y)^2$$

$$\frac{\partial h}{\partial y} = -x^2(\sin 2y - e^y \sin y + e^y \cos y) + 3(x+y)^2$$

We can verify this result with SymPy:

```python
from sympy.abc import x, y
from sympy import diff, cos, exp, pprint

r = x*cos(y)
s = x*exp(y)
t = x + y
h = r**2 - r*s + t**3
dhdx = diff(h, x)
dhdy = diff(h, y)
print(”Function”)
pprint(h)
print(”Derivative with respect to x”)
pprint(dhdx)
print(”Derivative with respect to y”)
pprint(dhdy)
```

Program 20.5: Find derivative of $g(r, s, t)$

```
Function
   2  y           2    2             3
- x ⋅e ⋅cos(y) + x ⋅cos (y) + (x + y)
Derivative with respect to x
       y                 2               2
- 2⋅x⋅e ⋅cos(y) + 2⋅x⋅cos (y) + 3⋅(x + y)
Derivative with respect to y
 2  y           2  y             2                          2
x ⋅e ⋅sin(y) - x ⋅e ⋅cos(y) - 2⋅x ⋅sin(y)⋅cos(y) + 3⋅(x + y)
```

Output 20.5: Derivative of $g(r, s, t)$

No matter how complex the expression is, the procedure to follow remains similar:

> “Your last computation tells you the first thing to do.”
>
> — Page 143, *Calculus For Dummies*, 2016.

Hence, start by tackling the outer function first, then move inwards to the next one. You may need to apply other rules along the way, as we have seen for Example 3. Do not forget to take the partial derivatives if you are working with multivariate functions.

20.7 Application in machine learning

Observe how similar the tree diagram is to the typical representation of a neural network (although we usually represent the latter by placing the inputs on the left hand side and the outputs on the right hand side). We can apply the chain rule to a neural network through the use of the backpropagation algorithm, in a very similar manner as to how we have applied it to the tree diagram above.

> “An area where the chain rule is used to an extreme is deep learning, where the function value y is computed as a many-level function composition.”
>
> — Page 159, *Mathematics for Machine Learning*, 2020.

A neural network can, indeed, be represented by a massive nested composite function. For example:

$$\mathbf{y} = f_K(f_{K-1}(\dots(f_1(\mathbf{x}))\dots))$$

Here, $\mathbf{x}$ are the inputs to the neural network (for example, the images) whereas $\mathbf{y}$ are the outputs (for example, the class labels). Every function, f_i for $i = 1, \dots, K$, is characterized by its own weights.

Applying the chain rule to such a composite function allows us to work backwards through all of the hidden layers making up the neural network, and efficiently calculate the error gradient of the loss function with respect to each weight, w_i, of the network until we arrive to the input.

20.8 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Mark Ryan. *Calculus For Dummies*. 2nd ed. Wiley, 2016.
https://www.amazon.com/dp/1119293499/

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge, 2020.
https://www.amazon.com/dp/110845514X

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

20.9 Summary

In this tutorial, you discovered the chain rule of calculus for univariate and multivariate functions.

Specifically, you learned:

- A composite function is the combination of two (or more) functions.
- The chain rule allows us to find the derivative of a composite function.
- The chain rule can be generalized to multivariate functions, and represented by a tree diagram.

- ▷ The chain rule is applied extensively by the backpropagation algorithm in order to calculate the error gradient of the loss function with respect to each weight.

In next chapter, we will see how the partial derivatives of a vector-valued function can be presented in matrix form.

21 The Jacobian

In the literature, the term *Jacobian* is often interchangeably used to refer to both the Jacobian matrix or its determinant.

Both the matrix and the determinant have useful and important applications: in machine learning, the Jacobian matrix aggregates the partial derivatives that are necessary for backpropagation; the determinant is useful in the process of changing between variables.

In this tutorial, you will review a gentle introduction to the Jacobian.

After completing this tutorial, you will know:

- ▷ The Jacobian matrix collects all first-order partial derivatives of a multivariate function that can be used for backpropagation.
- ▷ The Jacobian determinant is useful in changing between variables, where it acts as a scaling factor between one coordinate space and another.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ Partial Derivatives in Machine Learning
- ▷ The Jacobian Matrix
- ▷ Other Uses of the Jacobian

21.1 Partial derivatives in machine learning

We have thus far mentioned gradients and partial derivatives as being important for an optimization algorithm to update, say, the model weights of a neural network to reach an optimal set of weights. The use of partial derivatives permits each weight to be updated independently of the others, by calculating the gradient of the error curve with respect to each weight in turn.

Many of the functions that we usually work with in machine learning are multivariate, vector-valued functions, which means that they map multiple real inputs, n, to multiple real outputs, m:

$$\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$$

For example, consider a neural network that classifies grayscale images into several classes. The function being implemented by such a classifier would map the n pixel values of each single-channel input image, to m output probabilities of belonging to each of the different classes.

In training a neural network, the backpropagation algorithm is responsible for sharing back the error calculated at the output layer, among the neurons comprising the different hidden layers of the neural network, until it reaches the input.

> “The fundamental principle of the backpropagation algorithm in adjusting the weights in a network is that each weight in a network should be updated in proportion to the sensitivity of the overall error of the network to changes in that weight.”
>
> — Page 222, *Deep Learning*, 2016.

This sensitivity of the overall error of the network to changes in any one particular weight is measured in terms of the rate of change, which, in turn, is calculated by taking the partial derivative of the error with respect to the same weight.

For simplicity, suppose that one of the hidden layers of some particular network consists of just a single neuron, k. We can represent this in terms of a simple computational graph:

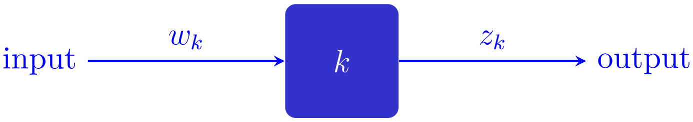


Figure 21.1: A neuron with a single input and a single output

Again, for simplicity, let's suppose that a weight, w_k, is applied to an input of this neuron to produce an output, z_k, according to the function that this neuron implements (including the nonlinearity). Then, the weight of this neuron can be connected to the error at the output of the network as follows (the following formula is formally known as the *chain rule of calculus*, see Chapter 20):

$$\frac{d(\text{error})}{dw_k} = \frac{d(\text{error})}{dz_k} \cdot \frac{dz_k}{dw_k}$$

Here, the derivative, $\frac{dz_k}{dw_k}$, first connects the weight, w_k, to the output, z_k, while the derivative, $\frac{d(\text{error})}{dz_k}$, subsequently connects the output, z_k, to the network error.

It is more often the case that we'd have many connected neurons populating the network, each attributed a different weight. Since we are more interested in such a scenario, then we can generalize beyond the scalar case to consider multiple inputs and multiple outputs:

$$\frac{d(\text{error})}{dw_k^{(i)}} = \frac{d(\text{error})}{dz_k^{(1)}} \cdot \frac{dz_k^{(1)}}{dw_k^{(i)}} + \frac{d(\text{error})}{dz_k^{(2)}} \cdot \frac{dz_k^{(2)}}{dw_k^{(i)}} + \frac{d(\text{error})}{dz_k^{(3)}} \cdot \frac{dz_k^{(3)}}{dw_k^{(i)}} + \cdots$$

This sum of terms can be represented more compactly as follows:

$$\frac{d(\text{error})}{dw_k^{(i)}} = \sum_j \frac{d(\text{error})}{dz_k^{(j)}} \cdot \frac{dz_k^{(j)}}{dw_k^{(i)}}$$

The above equation is indeed for all different i. If we list them out, we can make the left side a vector spanning each i. Similarly, the first fraction after the summation sight on the right can also be represented as a vector spanning each j. The second fraction after the summation sign, however, can be represented as a matrix in which each row is for a different i and each column is for a different j. We can use the vector notation and introduce the del operator, ∇, to represent the gradient of the error with respect to the weights $\mathbf{w}_k$ or the outputs $\mathbf{z}_k$. Then, if vectors are presented as columns, we can rewrite the above into the form of matrix multiplication:

$$\nabla_{\mathbf{w}_k}(\text{error}) = \left(\frac{\partial \mathbf{z}_k}{\partial \mathbf{w}_k}\right)^\top \nabla_{\mathbf{z}_k}(\text{error})$$

> The back-propagation algorithm consists of performing such a Jacobian-gradient product for each operation in the graph.
>
> — Page 207, *Deep Learning*, 2016.

This means that the backpropagation algorithm can relate the sensitivity of the network error to changes in the weights, through a multiplication by the *Jacobian matrix*, $\left(\frac{\partial z_k}{\partial w_k}\right)^\top$.

Hence, what does this Jacobian matrix contain?

21.2 The Jacobian matrix

The Jacobian matrix collects all first-order partial derivatives of a multivariate function.

Specifically, consider first a function that maps u real inputs, to a single real output:

$$f : \mathbb{R}^u \mapsto \mathbb{R}$$

Then, for an input vector, $\mathbf{x}$, of length, u, the Jacobian vector of size, $u \times 1$, can be defined as follows:

$$\mathbf{J} = \frac{df(x)}{dx} = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \cdots & \frac{\partial f(x)}{\partial x_u} \end{bmatrix}$$

Now, consider another function that maps u real inputs, to v real outputs:

$$\mathbf{f} : \mathbb{R}^u \mapsto \mathbb{R}^v$$

Then, for the same input vector, x, of length, u, the Jacobian is now a $v \times u$ matrix, $\mathbf{J} \in \mathbb{R}^{v\times u}$, that is defined as follows:

$$\mathbf{J} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_u} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_u} \\ \vdots & & \vdots \\ \frac{\partial f_v(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_v(\mathbf{x})}{\partial x_u} \end{bmatrix}$$

Reframing the Jacobian matrix into the machine learning problem considered earlier, while retaining the same number of u real inputs and v real outputs, we find that this matrix would contain the following partial derivatives:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial z_k^{(1)}}{\partial w_k^{(1)}} & \cdots & \dfrac{\partial z_k^{(1)}}{\partial w_k^{(u)}} \\ \vdots & & \vdots \\ \dfrac{\partial z_k^{(v)}}{\partial w_k^{(1)}} & \cdots & \dfrac{\partial z_k^{(v)}}{\partial w_k^{(u)}} \end{bmatrix}$$

As an example, consider a case that has a neural network with sigmoid activation function. Hence, in a layer with two input (x, y) and three outputs, the outputs can be represented as a vector-valued function:

$$f(x,y) = \begin{bmatrix} \dfrac{1}{1+e^{-(px+qy)}} & \dfrac{1}{1+e^{-(rx+sy)}} & \dfrac{1}{1+e^{-(tx+uy)}} \end{bmatrix}^\top$$

where p, q, r, s, t, u are the weights in that layer. The Jacobian can be found using SymPy:

```python
from sympy.abc import x, y, p, q, r, s, t, u
from sympy import exp, Matrix, simplify, pprint

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

# Vector-valued function
f = Matrix([sigmoid(p*x+q*y), sigmoid(r*x+s*y), sigmoid(t*x+u*y)])
variables = Matrix([x,y])
# Find and print the Jacobian
pprint(f.jacobian(variables))
```

Program 21.1: Finding Jacobian

```
⎡     -p⋅x - q⋅y          -p⋅x - q⋅y   ⎤
⎢  p⋅ℯ                 q⋅ℯ             ⎥
⎢──────────────────  ──────────────────⎥
⎢                 2                   2⎥
⎢⎛ -p⋅x - q⋅y    ⎞   ⎛ -p⋅x - q⋅y    ⎞ ⎥
⎢⎝ℯ           + 1⎠   ⎝ℯ           + 1⎠ ⎥
⎢                                      ⎥
⎢     -r⋅x - s⋅y          -r⋅x - s⋅y   ⎥
⎢  r⋅ℯ                 s⋅ℯ             ⎥
⎢──────────────────  ──────────────────⎥
⎢                 2                   2⎥
⎢⎛ -r⋅x - s⋅y    ⎞   ⎛ -r⋅x - s⋅y    ⎞ ⎥
⎢⎝ℯ           + 1⎠   ⎝ℯ           + 1⎠ ⎥
⎢                                      ⎥
⎢     -t⋅x - u⋅y          -t⋅x - u⋅y   ⎥
⎢  t⋅ℯ                 u⋅ℯ             ⎥
⎢──────────────────  ──────────────────⎥
⎢                 2                   2⎥
```

```
│⎛ -t⋅x - u⋅y    ⎞   ⎛ -t⋅x - u⋅y    ⎞ │
⎣⎝ℯ           + 1⎠   ⎝ℯ           + 1⎠ ⎦
```

Output 21.1: Finding Jacobian

which is:

$$\mathbf{J} = \begin{bmatrix} \dfrac{pe^{-(px+qy)}}{(1+e^{-(px+qy)})^2} & \dfrac{qe^{-(px+qy)}}{(1+e^{-(px+qy)})^2} \\ \dfrac{re^{-(rx+sy)}}{(1+e^{-(rx+sy)})^2} & \dfrac{se^{-(rx+sy)}}{(1+e^{-(rx+sy)})^2} \\ \dfrac{te^{-(tx+uy)}}{(1+e^{-(tx+uy)})^2} & \dfrac{ue^{-(tx+uy)}}{(1+e^{-(tx+uy)})^2} \end{bmatrix}$$

21.3 Other uses of the Jacobian

An important technique when working with integrals involves the *change of variables* (also referred to as, *integration by substitution* or *u-substitution*), where an integral is simplified into another integral that is easier to compute.

In the single variable case, substituting some variable, x, with another variable, u, can transform the original function into a simpler one for which it is easier to find an antiderivative. In the two variable case, an additional reason might be that we would also wish to transform the region of terms over which we are integrating, into a different shape.

> In the single variable case, there's typically just one reason to want to change the variable: to make the function "nicer" so that we can find an antiderivative. In the two variable case, there is a second potential reason: the two-dimensional region over which we need to integrate is somehow unpleasant, and we want the region in terms of u and v to be nicer—to be a rectangle, for example.
>
> — Page 412, *Single and Multivariable Calculus*, 2020.

When performing a substitution between two (or possibly more) variables, the process starts with a definition of the variables between which the substitution is to occur. For example, $x = f(u, v)$ and $y = g(u, v)$. This is then followed by a conversion of the integral limits depending on how the functions, f and g, will transform the u-v plane into the x-y plane. Finally, the absolute value of the *Jacobian determinant* is computed and included, to act as a scaling factor between one coordinate space and another.

21.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge, 2020.
https://www.amazon.com/dp/110845514X

John D. Kelleher. *Deep Learning*. Illustrated edition. The MIT Press Essential Knowledge series. MIT Press, 2019.
https://www.amazon.com/dp/0262537559/

Articles

Jacobian matrix and determinant. Wikipedia.
https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Integration by substitution. Wikipedia.
https://en.wikipedia.org/wiki/Integration_by_substitution

21.5 Summary

In this tutorial, you discovered a gentle introduction to the Jacobian. Specifically, you learned:

- ▷ The Jacobian matrix collects all first-order partial derivatives of a multivariate function that can be used for backpropagation.
- ▷ The Jacobian determinant is useful in changing between variables, where it acts as a scaling factor between one coordinate space and another.

In the next chapter, we will see another matrix notation in calculus that is very similar to Jacobian.

22 Hessian Matrices

Hessian matrices belong to a class of mathematical structures that involve second order derivatives. They are often used in machine learning and data science algorithms for optimizing a function of interest.

In this tutorial, you will discover Hessian matrices, their corresponding discriminants, and their significance. All concepts are illustrated via an example.

After completing this tutorial, you will know:

- ▷ Hessian matrices
- ▷ Discriminants computed via Hessian matrices
- ▷ What information is contained in the discriminant

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ Definition of a function's Hessian matrix and the corresponding discriminant
- ▷ Example of computing the Hessian matrix, and the discriminant
- ▷ What the Hessian and discriminant tell us about the function of interest

22.1 Prerequisites

For this tutorial, we assume that you already know:

- ▷ Derivative of functions (Chapter 7)
- ▷ Function of several variables, partial derivatives and gradient vectors (Chapter 18)
- ▷ Higher order derivatives (Chapter 19)

22.2 What is a Hessian matrix?

The Hessian matrix is a matrix of second order partial derivatives. Suppose we have a function f of n variables, i.e.,

$$f : \mathbb{R}^n \to \mathbb{R}$$

The Hessian of f is given by the following matrix on the left. The Hessian for a function of two variables is also shown below on the right.

$$f : \mathbb{R}^n \mapsto \mathbb{R} \qquad\qquad \text{for } f(x,y): \quad f : \mathbb{R}^2 \mapsto \mathbb{R}$$

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \qquad H_{f(x,y)} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{bmatrix}$$

Figure 22.1: Hessian a function of n variables (left). Hessian of $f(x,y)$ (right)

We already know from our tutorial on gradient vectors that the gradient is a vector of first order partial derivatives. The Hessian is similarly, a matrix of second order partial derivatives formed from all pairs of variables in the domain of f.

22.3 What is the discriminant?

The *determinant* of the Hessian matrix is also called the discriminant of f. For a two variable function $f(x,y)$, it is given by:

$$\det H_f = \begin{vmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{vmatrix} = f_{xx} f_{yy} - f_{xy}^2$$

The definition of determinant for any square matrix in general can be found in many linear algebra books.

22.4 Examples of Hessian matrices and discriminants

Suppose we have the following function:

$$g(x,y) = x^3 + 2y^2 + 3xy^2$$

Then the Hessian H_g and the discriminant D_g are given by:

$$H_g = \begin{bmatrix} 6x & 6y \\ 6y & 4+6x \end{bmatrix}$$

$$D_g = \begin{vmatrix} 6x & 6y \\ 6y & 4+6x \end{vmatrix} = 6x(4+6x) - 36y^2 = 36x^2 + 24x - 36y^2$$

Let's evaluate the discriminant at different points:

$$D_g(0,0) = 0$$
$$D_g(1,0) = 36 + 24 = 60$$
$$D_g(0,1) = -36$$
$$D_g(-1,0) = 12$$

All the above can be verified with SymPy:

```python
from sympy.abc import x, y
from sympy import pprint, hessian

g = x**3 + 2*y**2 + 3*x*y**2
variables = [x, y]
h = hessian(g, variables)
d = h.det()
print(”Function”)
pprint(g)
print(”Hessian”)
pprint(h)
print(”Discriminant”)
pprint(d)
for xval,yval in [(0,0), (1,0), (0,1), (-1,0)]:
    val = d.subs([(x,xval),(y,yval)])
    print(f”Discriminant at ({xval},{yval}) = {val}”)
```

Program 22.1: Finding Hessian and discriminant of $g(x, y)$

```
Function
 3        2      2
x  + 3⋅x⋅y  + 2⋅y
Hessian
⎡6⋅x    6⋅y  ⎤
⎢            ⎥
⎣6⋅y  6⋅x + 4⎦
Discriminant
    2              2
36⋅x  + 24⋅x - 36⋅y
Discriminant at (0,0) = 0
Discriminant at (1,0) = 60
Discriminant at (0,1) = -36
Discriminant at (-1,0) = 12
```

Output 22.1: Hessian and discriminant of $g(x, y)$

22.5 What do the Hessian and discriminant signify?

The Hessian and the corresponding discriminant are used to determine the local extreme points of a function. Evaluating them helps in the understanding of a function of several variables. Here are some important rules for a point (a, b) where the discriminant is $D(a, b)$:

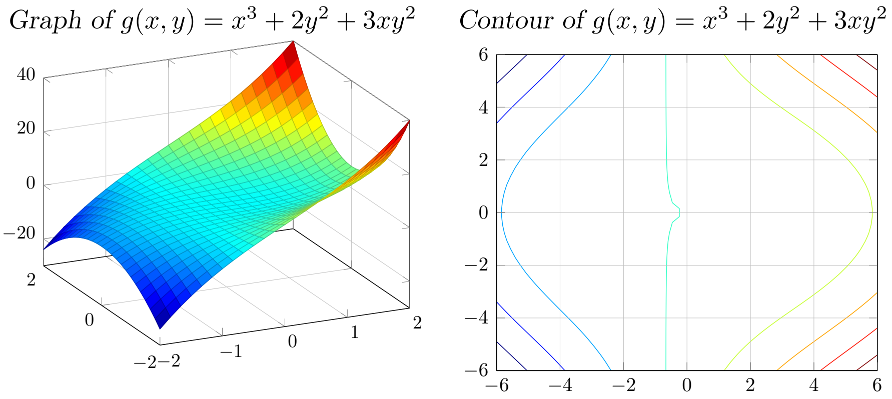
1. The function f has a *local minimum* if $f_{xx}(a, b) > 0$ and the discriminant $D(a, b) > 0$
2. The function f has a *local maximum* if $f_{xx}(a, b) < 0$ and the discriminant $D(a, b) > 0$
3. The function f has a saddle point if $D(a, b) < 0$
4. We cannot draw any conclusions if $D(a, b) = 0$ and need more tests

Example: $g(x, y)$

For the function $g(x, y)$:

1. We cannot draw any conclusions for the point $(0, 0)$
2. $f_{xx}(1, 0) = 6 > 0$ and $D_g(1, 0) = 60 > 0$, hence $(1, 0)$ is a local minimum
3. The point $(0, 1)$ is a saddle point as $D_g(0, 1) < 0$
4. $f_{xx}(-1, 0) = -6 < 0$ and $D_g(-1, 0) = 12 > 0$, hence $(-1, 0)$ is a local maximum

The figure below shows a graph of the function $g(x, y)$ and its corresponding contours.


Graph of $g(x, y) = x^3 + 2y^2 + 3xy^2$

Contour of $g(x, y) = x^3 + 2y^2 + 3xy^2$

Figure 22.2: Graph of $g(x, y)$ and contours of $g(x, y)$

22.6 Hessian matrix in machine learning

The Hessian matrix plays an important role in many machine learning algorithms, which involve optimizing a given function. While it may be expensive to compute, it holds some key information about the function being optimized. It can help determine the saddle points, and the local extremum of a function. It is used extensively in training neural networks and deep learning architectures.

22.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Optimization
- Eigenvalues of the Hessian matrix
- Inverse of Hessian matrix and neural network training

Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

22.8 Summary

In this tutorial, you discovered what are Hessian matrices. Specifically, you learned:

- Hessian matrix
- Discriminant of a function

While we list out the entire matrix in these two chapters, in the next chapter, we will learn about the Laplacian operator that can make our notation more concise.

23 The Laplacian

The Laplace operator was first applied to the study of celestial mechanics, or the motion of objects in outer space, by Pierre-Simon de Laplace, and as such has been named after him.

The Laplace operator has since been used to describe many different phenomena, from electric potentials, to the diffusion equation for heat and fluid flow, and quantum mechanics. It has also been recasted to the discrete space, where it has been used in applications related to image processing and spectral clustering.

In this tutorial, you will discover a gentle introduction to the Laplacian.

After completing this tutorial, you will know:

- The definition of the Laplace operator and how it relates to divergence.
- How the Laplace operator relates to the Hessian.
- How the continuous Laplace operator has been recasted to discrete-space, and applied to image processing and spectral clustering.

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- The Laplacian
  - The Concept of Divergence
  - The Continuous Laplacian
- The Discrete Laplacian

23.1 Prerequisites

For this tutorial, we assume that you already know what are:

- The gradient of a function (Chapter 18)
- Higher-order derivatives (Chapter 19)

- ▷ Multivariate functions (Chapter 16)
- ▷ The Hessian matrix (Chapter 22)

23.2 The Laplacian

The Laplace operator (or Laplacian, as it is often called) is the divergence of the gradient of a function.

In order to comprehend the previous statement better, it is best that we start by understanding the concept of *divergence*.

The concept of divergence

Divergence is a vector operator that operates on a vector field. The latter can be thought of as representing a flow of a liquid or gas, where each vector in the vector field represents a velocity vector of the moving fluid.

> Roughly speaking, divergence measures the tendency of the fluid to collect or disperse at a point ...
>
> — Page 432, *Single and Multivariable Calculus*, 2020.

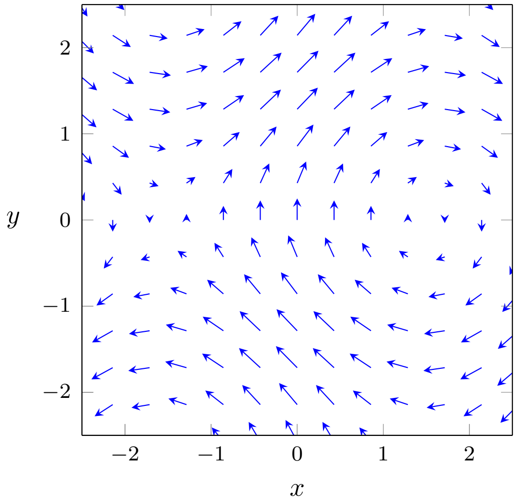


Figure 23.1: Part of the vector field of $(\cos x, \sin y)$

Using the nabla (or del) operator, ∇, the divergence is denoted by $\nabla\cdot$ and produces a scalar value when applied to a vector field, measuring the quantity of *fluid* at each point. In Cartesian coordinates, the divergence of a vector field, $\mathbf{F} = \langle f, g, h \rangle$, is given by:

$$\nabla \cdot \mathbf{F} = \left\langle \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right\rangle \cdot \langle f, g, h \rangle = \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} + \frac{\partial h}{\partial z}$$

Although the divergence computation involves the application of the divergence operator (rather than a multiplication operation), the dot in its notation is reminiscent of the dot product, which involves the multiplication of the components of two equal-length sequences (in this case, ∇ and $\mathbf{F}$) and the summation of the resulting terms.

The continuous Laplacian

Let's return back to the definition of the Laplacian.

Recall that the gradient of a two-dimensional function, f, is given by:

$$\nabla f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle$$

Then, the Laplacian (that is, the divergence of the gradient) of f can be defined by the sum of unmixed second partial derivatives:

$$\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

It can, equivalently, be considered as the trace (tr) of the function's Hessian, $H(f)$. The trace defines the sum of the elements on the main diagonal of a square $n \times n$ matrix, which in this case is the Hessian, and also the sum of its *eigenvalues*. As you may notice from Chapter 22 or in particular Figure 22.1 that the Hessian matrix contains the own (i.e., unmixed) second partial derivatives on the diagonal:

$$\nabla^2 f = \mathrm{tr}(H(f))$$

An important property of the trace of a matrix is its invariance to a *change of basis*. We have already defined the Laplacian in Cartesian coordinates. In polar coordinates, we would define it as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial^2 f}{d\theta^2}$$

The invariance of the trace to a change of basis means that the Laplacian can be defined in different coordinate spaces, but it would give the same value at some point (x, y) in the Cartesian coordinate space, and at the same point (r, θ) in the polar coordinate space.

Recall that we had also mentioned that the second derivative can provide us with information regarding the curvature of a function. Hence, intuitively, we can consider the Laplacian to also provide us with information regarding the local curvature of a function, through this summation of second derivatives.

The continuous Laplace operator has been used to describe many physical phenomena, such as electric potentials, and the diffusion equation for heat flow.

23.3 The discrete Laplacian

Analogous to the continuous Laplace operator, is the discrete one, so formulated in order to be applied to a discrete grid of, say, pixel values in an image, or to a graph.

Let's have a look at how the Laplace operator can be recasted for both applications.

In image processing, the Laplace operator is realized in the form of a digital filter that, when applied to an image, can be used for edge detection. In a sense, we can consider the Laplacian operator used in image processing to, also, provide us with information regarding the manner in which the function curves (or *bends*) at some particular point, (x, y).

In this case, the discrete Laplacian operator (or filter) is constructed by combining two, one-dimensional second derivative filters, into a single two-dimensional one:

$$\nabla^2 f(x,y) = f_{xx}(x,y) + f_{yy}(x,y)$$

In machine learning, the information provided by the discrete Laplace operator as derived from a graph can be used for the purpose of data clustering.

Consider a graph $G = (V, E)$, having a finite number of V vertices and E edges (i.e., an abstract structure of edges connecting vertices). Its Laplacian matrix, L, can be defined in terms of the degree matrix, D, containing information about the connectivity of each vertex, and the adjacency matrix, A, which indicates pairs of vertices that are adjacent in the graph:

$$L = D - A$$

Spectral clustering can be carried out by applying some standard clustering method (such as k-means) on the eigenvectors of the Laplacian matrix, hence partitioning the graph nodes (or the data points) into subsets.

One issue that can arise in doing so relates to a problem of scalability with large datasets, where the eigen-decomposition (or the extraction of the eigenvectors) of the Laplacian matrix may be prohibitive. The use of deep learning has been proposed[1] to address this problem, where a deep neural network is trained such that its outputs approximate the eigenvectors of the graph Laplacian. The neural network, in this case, is trained using a constrained optimization approach, to enforce the orthogonality of the network outputs.

23.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

David Guichard et al. *Single and Multivariable Calculus: Early Transcendentals*. 2020.
https://www.whitman.edu/mathematics/multivariable/multivariable.pdf

Al Bovik, ed. *Handbook of Image and Video Processing*. 2nd ed. Academic Press, 2005.
https://www.amazon.com/dp/0121197921

Articles

Laplace operator. Wikipedia.
https://en.wikipedia.org/wiki/Laplace_operator

Divergence. Wikipedia.
https://en.wikipedia.org/wiki/Divergence

Discrete Laplace operator. Wikipedia.
https://en.wikipedia.org/wiki/Discrete_Laplace_operator

Laplacian matrix. Wikipedia.
https://en.wikipedia.org/wiki/Laplacian_matrix

[1] https://arxiv.org/pdf/1801.01587.pdf

Spectral clustering. Wikipedia.
https://en.wikipedia.org/wiki/Spectral_clustering

Papers

Uri Shaham et al. “SpectralNet: Spectral Clustering Using Deep Neural Networks”. In: *Proc. ICLR*. 2018.
https://arxiv.org/pdf/1801.01587.pdf

23.5 Summary

In this tutorial, you discovered a gentle introduction to the Laplacian. Specifically, you learned:

- ▷ The definition of the Laplace operator and how it relates to divergence.
- ▷ How the Laplace operator relates to the Hessian.
- ▷ How the continuous Laplace operator has been recasted to discrete-space, and applied to image processing and spectral clustering.

This concludes our study in multivariate calculus. The next chapter will start our journey in exploring one particular use of calculus in optimization.

IV
Mathematical Programming

Introduction to Optimization and Mathematical Programming

24

Whether it is a supervised learning problem or an unsupervised problem, there will be some optimization algorithm working in the background. Almost any classification, regression or clustering problem can be cast as an optimization problem.

In this tutorial, you will discover what is optimization and concepts related to it. After completing this tutorial, you will know:

- What is Mathematical programming or optimization
- Difference between a maximization and minimization problems
- Difference between local and global optimal solutions
- Difference between constrained and unconstrained optimization
- Difference between linear and nonlinear programming
- Examples of optimization

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- Various introductory topics related to optimization
  - Constrained vs. unconstrained optimization
  - Equality vs. inequality constraints
  - Feasible region
- Examples of optimization in machine learning

24.1 What is optimization or mathematical programming?

In calculus and mathematics, the optimization problem is also termed as mathematical programming. To describe this problem in simple words, it is the mechanism through which we can find an element, variable or quantity that best fits a set of given criterion or constraints.

24.2 Maximization vs. minimization problems

The simplest cases of optimization problems are minimization or maximization of scalar functions. If we have a scalar function of one or more variables, $f(x_1, x_2, \ldots x_n)$ then the following is an optimization problem:

$$\text{Find } x_1, x_2, \ldots, x_n \text{ where } f(x) \text{ is minimum}$$

or we can have an equivalent maximization problem.

When we define functions quantifying errors or penalties, we apply a minimization problem. On the other hand, if a learning algorithm constructs a function modeling the accuracy of a method, we would maximize this function.

Many automated software tools for optimization, generally implement either a maximization problem or a minimization task but not both. Hence, we can convert a maximization problem to a minimization problem (and vice versa) by adding a negative sign to $f(x)$, i.e., "maximize $f(x)$ with respect to x" is equivalent to "minimize $-f(x)$ with respect to x"

As the two problems are equivalent, we'll only talk about either minimization or maximization problems in the rest of the tutorial. The same rules and definitions apply to its equivalent.

24.3 Global vs. local optimum points

In machine learning, we often encounter functions, which are highly nonlinear with a complex landscape. It is possible that there is a point where the function has the lowest value within a small or local region around that point. Such a point is called a local minimum point.

This is opposed to global minimum point, which is a point where the function has the least value over its entire domain. The following figure shows local and global minimum points.

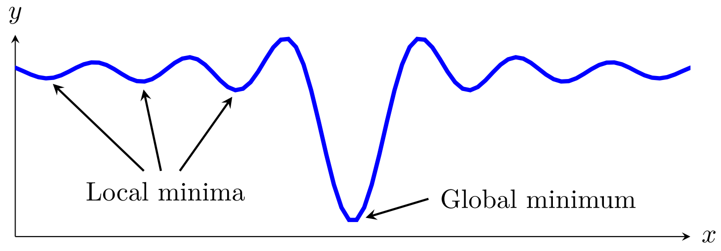


Figure 24.1: Local and global minimum points

24.4 Unconstrained vs. constrained optimization

There are many problems in machine learning, where we are interested in finding the global optimum point without any constraints or restrictions on the region in space. Such problems are called unconstrained optimization problems.

At times we have to solve an optimization problem subject to certain constraints. Such optimization problems are termed as constrained optimization problems. For example:

$$\text{minimize } x^2 + y^2 \qquad \text{subject to} \qquad x + y \leq 1$$

Examples of constrained optimization are:

1. Find minimum of a function when the sum of variables in the domain must sum to one
2. Find minimum of a function such that certain vectors are normal to each other
3. Find minimum of a function such that certain domain variables lie in a certain range.

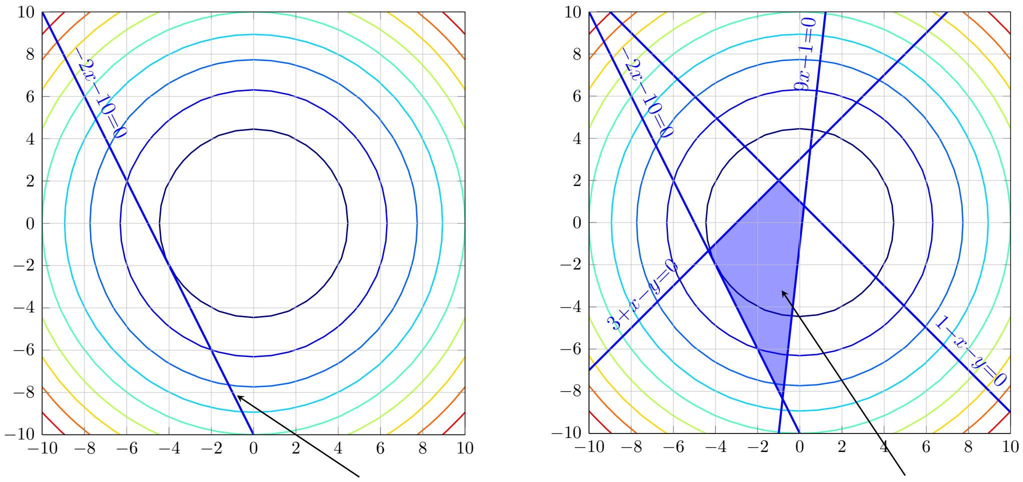
Feasible region

All the points in space where the constraints on the problem hold true comprise the feasible region. An optimization algorithm searches for optimal points in the feasible region. The feasible region for the two types of constraints is shown in the figure of the next section.

For an unconstrained optimization problem, the entire domain of the function is a feasible region.

Equality vs. inequality constraints

The constraints imposed in an optimization problem could be equality constraints or inequality constraints. The figure below shows the two types of constraints.



Equality constraint:
$-2x - 10 = 0$

Feasible region:
all points on this line

Inequality constraint:
$3 + x - y > 0$
$-2x - 10 < 0$
$9x - 1 < 0$
$1 - x - y > 0$

Feasible region

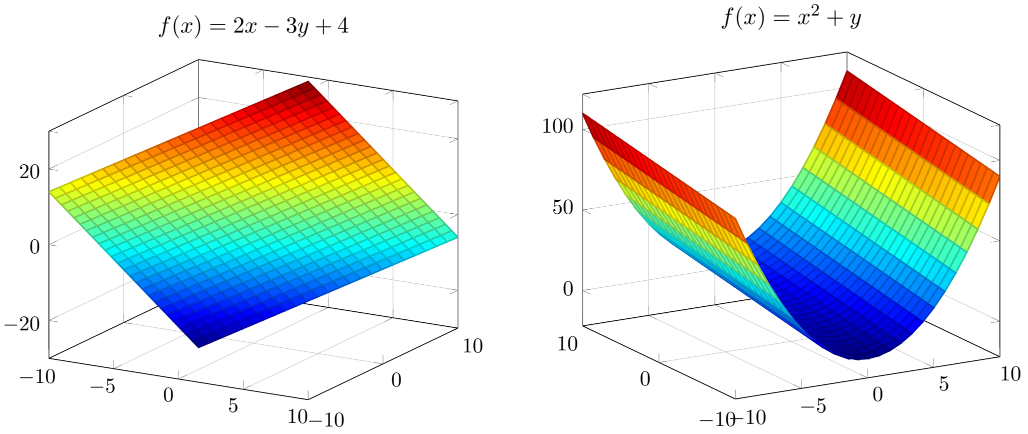
Figure 24.2: Equality vs. inequality constraints

24.5 Linear vs. nonlinear programming

An optimization problem where the function is linear and all equality or inequality constraints are also linear constraints is called a linear programming problem.

If either the objective function is nonlinear or one or more than one constraints is nonlinear, then we have a nonlinear programming problem.

To visualize the difference between linear and nonlinear functions you can check out the figure below.



$f(x) = 2x - 3y + 4$

$f(x) = x^2 + y$

Linear function

Nonlinear function

Figure 24.3: Linear vs. nonlinear functions

24.6 Examples of optimization in machine learning

Listed below are some well known machine learning algorithms that employ optimization. You should keep in mind that almost all machine learning algorithms employ some kind of optimization.

1. Gradient descent in neural networks (unconstrained optimization).
2. Method of Lagrange multipliers in support vector machines (constrained optimization).
3. Principal component analysis (constrained optimization)
4. Clustering via expectation maximization algorithm (constrained optimization)
5. Logistic regression (unconstrained optimization)
6. Genetic algorithms in evolutionary learning algorithms (different variants exist to solve both constrained and unconstrained optimization problems).

24.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- ▷ Method of Lagrange multipliers
- ▷ Nonlinear optimization techniques
- ▷ The simplex method

24.8 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

24.9 Summary

In this tutorial, you discovered what is mathematical programming or optimization problem. Specifically, you learned:

- ▷ Maximization vs. minimization
- ▷ Constrained vs. unconstrained optimization
- ▷ Why optimization is important in machine learning

In the next chapter, we will see how an optimization problem can be solved.

The Method of Lagrange Multipliers

25

The method of Lagrange multipliers is a simple and elegant method of finding the local minima or local maxima of a function subject to equality or inequality constraints. Lagrange multipliers are also called undetermined multipliers. In this tutorial we'll talk about this method when given equality constraints.

In this tutorial, you will discover the method of Lagrange multipliers and how to find the local minimum or maximum of a function when equality constraints are present.

After completing this tutorial, you will know:

▷ How to find points of local maximum or minimum of a function with equality constraints

▷ Method of Lagrange multipliers with equality constraints

Let's get started.

Overview

This tutorial is divided into two parts; they are:

▷ Method of Lagrange multipliers with equality constraints

▷ Two solved examples

25.1 Prerequisites

For this tutorial, we assume that you already know what are:

▷ Derivative of functions (Chapter 7)

▷ Function of several variables, partial derivatives and gradient vectors (Chapter 18)

▷ Introduction to optimization (Chapter 24)

▷ Gradient descent (Chapter 29)

25.2 The method of Lagrange multipliers with equality constraints

Suppose we have the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_1(x) = 0 \\ & g_2(x) = 0 \\ & \vdots \\ & g_n(x) = 0 \end{aligned}$$

The method of Lagrange multipliers first constructs a function called the Lagrange function as given by the following expression.

$$L(x, \lambda) = f(x) + \lambda_1 g_1(x) + \lambda_2 g_2(x) + \ldots + \lambda_n g_n(x)$$

Here λ represents a vector of Lagrange multipliers, i.e.,

$$\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_n]^\top$$

To find the points of local minimum of $f(x)$ subject to the equality constraints, we find the stationary points of the Lagrange function $L(x, \lambda)$, i.e., we solve the following equations:

$$\begin{aligned} \nabla \times L &= 0 \\ \frac{\partial L}{\partial \lambda_i} &= 0 \qquad \text{for } i = 1, \cdots, n \end{aligned}$$

Hence, we get a total of $m + n$ equations to solve, where

- ▷ $m =$ number of variables in domain of f
- ▷ $n =$ number of equality constraints.

In short, the points of local minimum would be the solution of the following equations:

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= 0 \qquad \text{for } j = 1, \cdots, m \\ g_i(x) &= 0 \qquad \text{for } i = 1, \cdots, n \end{aligned}$$

25.3 Solved examples

This section contains two solved examples. If you solve both of them, you'll get a pretty good idea on how to apply the method of Lagrange multipliers to functions of more than two variables, and a higher number of equality constraints.

Example 1: One equality constraint

Let's solve the following minimization problem:

$$\begin{aligned} \text{minimize} \qquad & f(x) = x^2 + y^2 \\ \text{subject to} \qquad & x + 2y - 1 = 0 \end{aligned}$$

The first step is to construct the Lagrange function:

$$L(x, y, \lambda) = x^2 + y^2 + \lambda(x + 2y - 1)$$

We have the following three equations to solve:

$$\frac{\partial L}{\partial x} = 2x + \lambda = 0 \tag{25.1}$$

$$\frac{\partial L}{\partial y} = 2y + 2\lambda = 0 \tag{25.2}$$

$$\frac{\partial L}{\partial \lambda} = x + 2y - 1 = 0 \tag{25.3}$$

Using (25.1) and (25.2), we get:

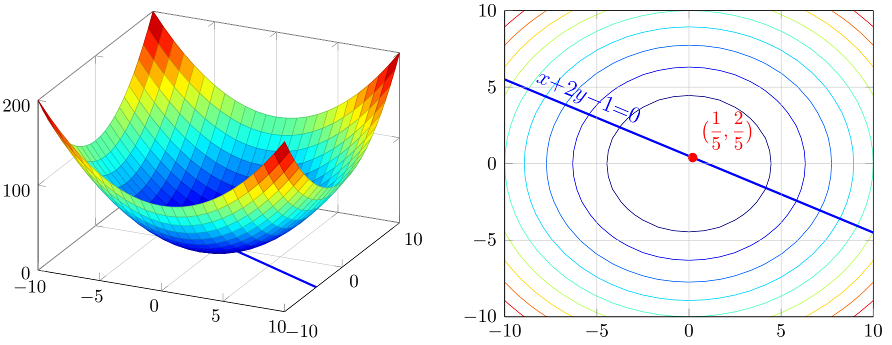
$$\lambda = -2x = -y$$

Plugging this in (25.3) gives us:

$$x = \frac{1}{5}$$

$$y = \frac{2}{5}$$

Hence, the local minimum point lies at $(\frac{1}{5}, \frac{2}{5})$ as shown in the right figure. The left figure shows the graph of the function.



Graph of $f(x, y)$ and the constraint *Contour of $f(x, y)$ and the constraint*

Figure 25.1: Graph of function (left). Contours, constraint and local minima (right)

This constrained optimization problem can also be solved numerically using SciPy, as follows:

```python
import numpy as np
from scipy.optimize import minimize

def objective(x):
    return x[0]**2 + x[1]**2

def constraint(x):
    return x[0]+2*x[1]-1

# initial guesses
x0 = np.array([3,3])

# optimize
bounds = ((-10,10), (-10,10))
constraints = [{”type”:”eq”, ”fun”:constraint}]
solution = minimize(objective, x0, method='SLSQP',
                    bounds=bounds, constraints=constraints)
x = solution.x

# show solution
print('Objective:', objective(x))
print('Solution:', x)
```

Program 25.1: Solving the optimization problem of Example 1

SciPy has several algorithms for constrained optimization. The above uses SLSQP (Sequential Least-Squares Programming). The objective function can be defined with a single argument of a vector of arbitrary length. The constraints are similarly defined as a function, and only those arguments where it produced zero return value will be considered as feasible. The SLSQP algorithm requires a range for each element of the vector argument to search on, as well as an initial "guessed" solution to start. The above code will produce the following solution:

```
Objective: 0.19999999999999998
Solution: [0.19999999 0.4       ]
```

Output 25.1: Solution to Example 1

The numerical solution matched the one found by the method of Lagrange multiplier. However, not all problems can be solved using the SLSQP algorithm (e.g., when the problem is not in the form of quadratic programming). The method of Lagrange multiplier, however, can be applied to a wider range of problems.

Example 2: Two equality constraints

Suppose we want to find the minimum of the following function subject to the given constraints:

$$\begin{aligned} \text{minimize} \qquad & g(x,y) = x^2 + 4y^2 \\ \text{subject to} \qquad & x + y = 0 \\ & x^2 + y^2 - 1 = 0 \end{aligned}$$

The solution of this problem can be found by first constructing the Lagrange function:

$$L(x, y, \lambda_1, \lambda_2) = x^2 + 4y^2 + \lambda_1(x + y) + \lambda_2(x^2 + y^2 - 1)$$

We have 4 equations to solve:

$$\begin{aligned}
\frac{\partial L}{\partial x} &= 2x + \lambda_1 + 2x\lambda_2 = 0 \\
\frac{\partial L}{\partial y} &= 8y + \lambda_1 + 2y\lambda_2 = 0 \\
\frac{\partial L}{\partial \lambda_1} &= x + y = 0 \\
\frac{\partial L}{\partial \lambda_2} &= x^2 + y^2 - 1 = 0
\end{aligned}$$

Solving the above system of equations gives us two solutions for (x, y), i.e. we get the two points:

$$\Big(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\Big)$$
$$\Big(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\Big)$$

This problem can also be solved using SciPy, but the algorithm will produce only one of the solutions:

```
import numpy as np
from scipy.optimize import minimize

def objective(x):
    return x[0]**2 + 4*x[1]**2

def constraint1(x):
    return x[0]+x[1]

def constraint2(x):
    return x[0]**2 + x[1]**2 - 1

# initial guesses
x0 = np.array([3,3])

# optimize
bounds = ((-10,10), (-10,10))
constraints = [{”type”:”eq”, ”fun”:constraint1}, {”type”:”eq”, ”fun”:constraint2}]
solution = minimize(objective, x0, method='SLSQP',
                    bounds=bounds, constraints=constraints)
x = solution.x

# show solution
```

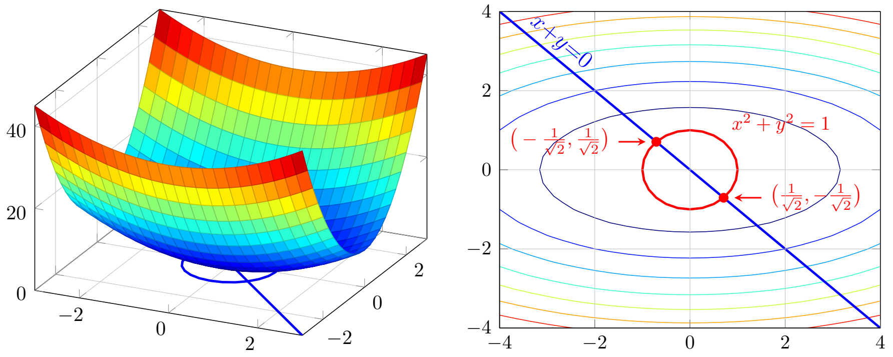
```
print('Objective:', objective(x))
print('Solution:', x)
```

Program 25.2: Solving the optimization problem of Example 2

```
Objective: 2.5000000000173994
Solution: [-0.70710678  0.70710678]
```

Output 25.2: Solution to Example 2

The function along with its constraints and local minimum are shown below.



Graph of $f(x, y)$ and the constraint *Contour of $f(x, y)$ and the constraint*

Figure 25.2: Graph of function (left). Contours, constraint and local minima (right)

25.4 Relationship to maximization problems

If you have a function to maximize, you can solve it in a similar manner, keeping in mind that maximization and minimization are equivalent problems, i.e.,

$$\text{maximize } f(x) \qquad \text{is equivalent to} \qquad \text{minimize } -f(x)$$

25.5 The method of Lagrange multipliers in machine learning

Many well known machine learning algorithms make use of the method of Lagrange multipliers. For example, the theoretical foundations of principal components analysis (PCA) are built using the method of Lagrange multipliers with equality constraints. Similarly, the optimization problem in support vector machines SVMs is also solved using this method. However, in SVMS, inequality constraints are also involved.

25.6 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

25.7 Summary

In this tutorial, you discovered what is the method of Lagrange multipliers. Specifically, you learned:

- Lagrange multipliers and the Lagrange function
- How to solve an optimization problem when equality constraints are given

In the next chapter, we will see how the same method can be applied to the case of having inequalities as constraints.

Lagrange Multipliers with Inequality Constraints 26

In the previous chapter, we introduced the method of Lagrange multipliers to find local minima or local maxima of a function with equality constraints. The same method can be applied to those with inequality constraints as well.

In this tutorial, you will discover the method of Lagrange multipliers applied to find the local minimum or maximum of a function when inequality constraints are present, optionally together with equality constraints.

After completing this tutorial, you will know:

- How to find points of local maximum or minimum of a function with inequality constraints
- Method of Lagrange multipliers with inequality constraints

Let's get started.

Overview

This tutorial is divided into four parts; they are:

- Constrained optimization and Lagrangians
- The complementary slackness condition
- Example 1: Mean-variance portfolio optimization
- Example 2: Water-filling algorithm

26.1 Prerequisites

For this tutorial, we assume that you already have reviewed:

- Derivative of functions (Chapter 7)
- Function of several variables, partial derivatives and gradient vectors (Chapter 18)
- Introduction to optimization (Chapter 24)
- The method of Lagrange multipliers (Chapter 25)

26.2 Constrained optimization and Lagrangians

Extending from the previous chapter, a constrained optimization problem can be generally considered as:

$$\begin{aligned} \min \quad & f(X) \\ \text{subject to} \quad & g(X) = 0 \\ & h(X) \geq 0 \\ & k(X) \leq 0 \end{aligned}$$

where X is a scalar or vector values. Here, $g(X) = 0$ is the equality constraint, and $h(X) \geq 0$, $k(X) \leq 0$ are inequality constraints. Note that we always use $\geq$ and $\leq$ rather than $>$ and $<$ in optimization problems because the former defined a *closed set* in mathematics from where we should look for the value of X. These can be many constraints of each type in an optimization problem.

The equality constraints are easy to handle, but the inequality constraints are not. Therefore, one way to make it easier to tackle is to convert the inequalities into equalities by introducing *slack variables*:

$$\begin{aligned} \min \quad & f(X) \\ \text{subject to} \quad & g(X) = 0 \\ & h(X) - s^2 = 0 \\ & k(X) + t^2 = 0 \end{aligned}$$

When something is negative, adding a certain positive quantity into it will make it equal to zero and vice versa. That quantity is the slack variable; the s^2 and t^2 above are examples. We deliberately put the s^2 and t^2 terms there to denote that they must not be negative.

With the slack variables introduced, we can use the Lagrange multipliers approach to solve it, in which the Lagrange function or Lagrangian is defined as:

$$L(X, \lambda, \theta, \phi) = f(X) - \lambda g(X) - \theta(h(X) - s^2) + \phi(k(X) + t^2)$$

It is useful to know that, for the optimal solution X^* to the problem, the inequality constraints either have the equality holds (which the slack variable is zero) or not. Those inequality constraints with the equality hold are called the active constraints. Otherwise, the inactive constraints. In this sense, you can consider that the equality constraints are always active.

26.3 The complementary slackness condition

The reason we need to know whether a constraint is active or not is because of the Krush-Kuhn-Tucker (KKT) conditions. Precisely, the KKT conditions describe what happens when X^* is the optimal solution to a constrained optimization problem:

1. The gradient of the Lagrangian function is zero

2. All constraints are satisfied
3. The inequality constraints satisfy the complementary slackness condition

The most important of them is the complementary slackness condition. While we learned that an optimization problem with equality constraints can be solved using the Lagrange multiplier where the gradient of the Lagrangian is zero at the optimal solution, the complementary slackness condition extends this to the case of inequality constraint by saying that at the optimal solution X^*, either the Lagrange multiplier is zero, or the corresponding inequality constraint is active.

The use of the complementary slackness condition is to help us explore different cases in solving the optimization problem. It is best explained with an example.

26.4 Example 1: Mean-variance portfolio optimization

This is an example from finance. If we have 1 dollar and engage in two different investments, in which their return is modeled as a bivariate Gaussian distribution, how much should we invest in each to minimize the overall variance in return?

This optimization problem, also known as Markowitz mean-variance portfolio optimization, is formulated as:

$$\begin{aligned}
\text{min} \qquad & f(w_1, w_2) = w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1w_2\sigma_{12} \\
\text{subject to} \qquad & w_1 + w_2 = 1 \\
& w_1 \geq 0 \\
& w_1 \leq 1
\end{aligned}$$

where the last two are to bound the weight of each investment to between 0 and 1 dollars. Let's assume $\sigma_1^2 = 0.25$, $\sigma_2^2 = 0.10$, $\sigma_{12} = 0.15$. Then the Lagrangian function is defined as:

$$\begin{aligned}
L(w_1, w_2, \lambda, \theta, \phi) =& 0.25w_1^2 + 0.1w_2^2 + 2(0.15)w_1w_2 \\
& - \lambda(w_1 + w_2 - 1) \\
& - \theta(w_1 - s^2) - \phi(w_1 - 1 + t^2)
\end{aligned}$$

and we have the gradients:

$$\begin{aligned}
\frac{\partial L}{\partial w_1} &= 0.5w_1 + 0.3w_2 - \lambda - \theta - \phi \\
\frac{\partial L}{\partial w_2} &= 0.2w_2 + 0.3w_1 - \lambda \\
\frac{\partial L}{\partial \lambda} &= 1 - w_1 - w_2 \\
\frac{\partial L}{\partial \theta} &= s^2 - w_1 \\
\frac{\partial L}{\partial \phi} &= 1 - w_1 - t^2
\end{aligned}$$

From this point onward, the complementary slackness condition has to be considered. We have two slack variables, s and t, and the corresponding Lagrange multipliers are θ and ϕ. We now have to consider whether a slack variable is zero (where the corresponding inequality constraint is active), or the Lagrange multiplier is zero (the constraint is inactive). There are four possible cases:

1. $\theta = \phi = 0$ and $s^2 > 0$, $t^2 > 0$
2. $\theta \neq 0$ but $\phi = 0$, and $s^2 = 0$, $t^2 > 0$
3. $\theta = 0$ but $\phi \neq 0$, and $s^2 > 0$, $t^2 = 0$
4. $\theta \neq 0$ and $\phi \neq 0$, and $s^2 = t^2 = 0$

For case 1, using $\partial L/\partial\lambda = 0$, $\partial L/\partial w_1 = 0$, and $\partial L/\partial w_2 = 0$ we get:

$$\begin{aligned} w_2 &= 1 - w_1 \\ 0.5w_1 + 0.3w_2 &= \lambda \\ 0.3w_1 + 0.2w_2 &= \lambda \end{aligned}$$

after which we get $w_1 = -1$, $w_2 = 2$, $\lambda = 0.1$. But with $\partial L/\partial\theta = 0$, we get $s^2 = -1$, in which we cannot find a solution (s^2 cannot be negative). Thus this case is infeasible.

For case 2, with $\partial L/\partial\theta = 0$, we get $w_1 = 0$. Hence from $\partial L/\partial\lambda = 0$, we know $w_2 = 1$. And with $\partial L/\partial w_2 = 0$, we found $\lambda = 0.2$, and from $\partial L/\partial w_1$, we get $\phi = 0.1$. In this case, the objective function is 0.1

For case 3, with $\partial L/\partial\phi = 0$, we get $w_1 = 1$. Hence from $\partial L/\partial\lambda = 0$, we know $w_2 = 0$. And with $\partial L/\partial w_2 = 0$, we get $\lambda = 0.3$, and from $\partial L/\partial w_1$, we get $\theta = 0.2$. In this case, the objective function is 0.25

For case 4, we get $w_1 = 0$ from $\partial L/\partial\theta = 0$ but $w_1 = 1$ from $\partial L/\partial\phi = 0$. Hence this case is infeasible.

Comparing the objective function from case 2 and case 3, we see that the value from case 2 is lower. Therefore, that is our solution to the optimization problem, with the optimal solution attained at $w_1 = 0$, $w_2 = 1$.

This problem can also be solved by SciPy using the SLSQP method:

```python
import numpy as np
from scipy.optimize import minimize

def objective(x):
    return 0.25*x[0]**2 + 0.1*x[1]**2 + 0.3*x[0]*x[1]

def constraint1(x):
    # Equality constraint: The result required be zero
    return x[0] + x[1] - 1

def constraint2(x):
    # Inequality constraint: The result required be non-negative
    return x[0]
```

```
def constraint3(x):
    # Inequality constraint: The result required be non-negative
    return 1-x[0]

# initial guesses
x0 = np.array([0, 1])

# optimize
bounds = ((0,1), (0,1))
constraints = [
    {”type”:”eq”, ”fun”:constraint1},
    {”type”:”ineq”, ”fun”:constraint2},
    {”type”:”ineq”, ”fun”:constraint3},
]
solution = minimize(objective, x0, method='SLSQP',
                    bounds=bounds, constraints=constraints)
x = solution.x

# show solution
print('Objective:', objective(x))
print('Solution:', x)
```

Program 26.1: Solving the optimization problem of Example 1

```
Objective: 0.1
Solution: [0. 1.]
```

Output 26.1: Solution to Example 1

As an exercise, you can retry the above with $\sigma_{12} = -0.15$. The solution will be 0.0038, attained when $w_1 = \frac{5}{13}$, with the two inequality constraints inactive.

26.5 Example 2: Water-filling algorithm

This is an example from communication engineering. If we have a channel (say, a wireless bandwidth) in which the noise power is N and the signal power is S, the channel capacity (in terms of bits per second) is proportional to $\log_2(1+S/N)$. If we have k similar channels, each has its own noise and signal level, the total capacity of all channels is the sum $\sum_i \log_2(1+S_i/N_i)$.

Assume we are using a battery that can give only 1 watt of power, and this power has to distribute to the k channels (denoted as $p_1, \cdots, p_k$). Each channel may have different attenuation, so at the end, the signal power is discounted by a gain g_i for each channel. Then the maximum total capacity we can achieve by using these k channels is formulated as an optimization problem:

$$\begin{aligned} \max \qquad & f^*(p_1, \cdots, p_k) = \sum_{i=1}^{k} \log_2 \left(1 + \frac{g_i p_i}{n_i}\right) \\ \text{subject to} \qquad & \sum_{i=1}^{k} p_i = 1 \\ & p_1, \cdots, p_k \geq 0 \end{aligned}$$

For convenience of differentiation, we notice $\log_2 x = \log x / \log 2$ and $\log(1 + g_i p_i / n_i) = \log(n_i + g_i p_i) - \log(n_i)$. Maximize for one would maximize for another. Hence the objective function can be replaced with:

$$f(p_1, \cdots, p_k) = \sum_{i=1}^{k} \log(n_i + g_i p_i)$$

in the sense that if f attained its maximum, f^* also attained its maximum. Assume we have $k = 3$ channels, each has noise level of 1.0, 0.9, 1.0, respectively, and the channel gain is 0.9, 0.8, and 0.7, then the optimization problem is:

$$\begin{aligned} \max \qquad & f(p_1, p_2, p_k) = \log(1 + 0.9p_1) + \log(0.9 + 0.8p_2) + \log(1 + 0.7p_3) \\ \text{subject to} \qquad & p_1 + p_2 + p_3 = 1 \\ & p_1, p_2, p_3 \geq 0 \end{aligned}$$

We have three inequality constraints here. The Lagrangian function is defined as:

$$\begin{aligned} & L(p_1, p_2, p_3, \lambda, \theta_1, \theta_2, \theta_3) \\ = \; & \log(1 + 0.9p_1) + \log(0.9 + 0.8p_2) + \log(1 + 0.7p_3) \\ & - \lambda(p_1 + p_2 + p_3 - 1) \\ & - \theta_1(p_1 - s_1^2) - \theta_2(p_2 - s_2^2) - \theta_3(p_3 - s_3^2) \end{aligned}$$

The gradient is therefore:

$$\begin{aligned} \frac{\partial L}{\partial p_1} &= \frac{0.9}{1 + 0.9p_1} - \lambda - \theta_1 \\ \frac{\partial L}{\partial p_2} &= \frac{0.8}{0.9 + 0.8p_2} - \lambda - \theta_2 \\ \frac{\partial L}{\partial p_3} &= \frac{0.7}{1 + 0.7p_3} - \lambda - \theta_3 \\ \frac{\partial L}{\partial \lambda} &= 1 - p_1 - p_2 - p_3 \\ \frac{\partial L}{\partial \theta_1} &= s_1^2 - p_1 \end{aligned}$$

$$\frac{\partial L}{\partial \theta_2} = s_2^2 - p_2$$

$$\frac{\partial L}{\partial \theta_3} = s_3^2 - p_3$$

But now we have 3 slack variables, and we have to consider 8 cases:

1. $\theta_1 = \theta_2 = \theta_3 = 0$, hence none of s_1^2, s_2^2, s_3^2 are zero
2. $\theta_1 = \theta_2 = 0$ but $\theta_3 \neq 0$, hence only $s_3^2 = 0$
3. $\theta_1 = \theta_3 = 0$ but $\theta_2 \neq 0$, hence only $s_2^2 = 0$
4. $\theta_2 = \theta_3 = 0$ but $\theta_1 \neq 0$, hence only $s_1^2 = 0$
5. $\theta_1 = 0$ but θ_2, θ_3 are non-zero, hence only $s_2^2 = s_3^2 = 0$
6. $\theta_2 = 0$ but θ_1, θ_3 are non-zero, hence only $s_1^2 = s_3^2 = 0$
7. $\theta_3 = 0$ but θ_1, θ_2 are non-zero, hence only $s_1^2 = s_2^2 = 0$
8. all of $\theta_1, \theta_2, \theta_3$ are non-zero, hence $s_1^2 = s_2^2 = s_3^2 = 0$

Immediately we can tell case 8 is infeasible since from $\partial L/\partial \theta_i = 0$, we can make $p_1 = p_2 = p_3 = 0$, but it cannot make $\partial L/\partial \lambda = 0$.

For case 1, we have:

$$\frac{0.9}{1 + 0.9p_1} = \frac{0.8}{0.9 + 0.8p_2} = \frac{0.7}{1 + 0.7p_3} = \lambda$$

from $\partial L/\partial p_1 = \partial L/\partial p_2 = \partial L/\partial p_3 = 0$. Together with $p_3 = 1 - p_1 - p_2$ from $\partial L/\partial \lambda = 0$, we found the solution to be $p_1 = 0.444$, $p_2 = 0.430$, $p_3 = 0.126$, and the objective function $f(p_1, p_2, p_3) = 0.639$.

For case 2, we have $p_3 = 0$ from $\partial L/\partial \theta_3 = 0$. Further, using $p_2 = 1 - p_1$ from $\partial L/\partial \lambda = 0$, and

$$\frac{0.9}{1 + 0.9p_1} = \frac{0.8}{0.9 + 0.8p_2} = \lambda$$

from $\partial L/\partial p_1 = \partial L/\partial p_2 = 0$, we can solve for $p_1 = 0.507$ and $p_2 = 0.493$. The objective function $f(p_1, p_2, p_3) = 0.634$.

Similarly in case 3, $p_2 = 0$, and we solved $p_1 = 0.659$ and $p_3 = 0.341$, with the objective function $f(p_1, p_2, p_3) = 0.574$.

In case 4, we have $p_1 = 0$, $p_2 = 0.652$, $p_3 = 0.348$, and the objective function $f(p_1, p_2, p_3) = 0.570$.

Case 5 has $p_2 = p_3 = 0$ and hence $p_1 = 1$. Thus we have the objective function $f(p_1, p_2, p_3) = 0.536$.

Similarly in case 6 and case 7, we have $p_2 = 1$ and $p_1 = 1$ respectively. The objective function attained 0.531 and 0.425, respectively.

Comparing all these cases, we found that the maximum value that the objective function attained is in case 1. Hence the solution to this optimization problem is $p_1 = 0.444$, $p_2 = 0.430$, $p_3 = 0.126$, with $f(p_1, p_2, p_3) = 0.639$.

This problem is an example where SciPy cannot find the optimal solution. The issue lies with the use of logarithms in the objective function. Hence, it is more accurate if we can solve it for an exact solution using the method of Lagrange multipliers.

```python
import numpy as np
from scipy.optimize import minimize

def objective(x):
    return np.log(1+0.9*x[0]) + np.log(0.9+0.8*x[1]) + np.log(1+0.7*x[2])

# Equality constraint: The result required be zero
def constraint1(x):
    return x[0] + x[1] + x[2] - 1

# Inequality constraints: The result required be non-negative
def constraint2(x):
    return x[0]
def constraint3(x):
    return x[1]
def constraint4(x):
    return x[2]

# initial guesses
x0 = np.array([0.4, 0.4, 0.4])

# optimize
bounds = ((0,1), (0,1), (0,1))
constraints = [
    {”type”:”eq”, ”fun”:constraint1},
    {”type”:”ineq”, ”fun”:constraint2},
    {”type”:”ineq”, ”fun”:constraint3},
    {”type”:”ineq”, ”fun”:constraint4},
]
solution = minimize(objective, x0, method='SLSQP',
                    bounds=bounds, constraints=constraints)
x = solution.x

# show solution
print('Objective:', objective(x))
print('Solution:', x)
```

Program 26.2: Solving the optimization problem of Example 2

```
Objective: 0.4252677354043441
Solution: [0. 0. 1.]
```

Output 26.2: Sub-optimal solution for Example 2

26.6 Extensions and further reading

In the above example, we introduced the slack variables into the Lagrangian function; some books may prefer not to add the slack variables but to limit the Lagrange multipliers for inequality constraints as positive. In that case you may see the Lagrangian function written as

$$L(X, \lambda, \theta, \phi) = f(X) - \lambda g(X) - \theta h(X) + \phi k(X)$$

but requires $\theta \geq 0; \phi \geq 0$.

The Lagrangian function is also useful to apply to the primal-dual approach for finding the maximum or minimum. This is particularly helpful if the objectives or constraints are nonlinear, in which the solution may not be easily found.

Some books that cover this topic are:

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, 2004.
https://amzn.to/34mvCr1

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

26.7 Summary

In this tutorial, you discovered how the method of Lagrange multipliers can be applied to inequality constraints. Specifically, you learned:

- ▷ Lagrange multipliers and the Lagrange function in the presence of inequality constraints
- ▷ How to use KKT conditions to solve an optimization problem when inequality constraints are given

In the next chapter, we will see a different application of calculus.

V
Approximation

27 Approximation

When it comes to machine learning tasks such as classification or regression, approximation techniques play a key role in learning from the data. Many machine learning methods approximate a function or a mapping between the inputs and outputs via a learning algorithm.

In this tutorial, you will discover what is approximation and its importance in machine learning and pattern recognition. After completing this tutorial, you will know:

- ▷ What is approximation
- ▷ Importance of approximation in machine learning

Let's get started.

Overview

This tutorial is divided into 3 parts; they are:

- ▷ What is approximation?
- ▷ Approximation when the form of function is not known
- ▷ Approximation when the form of function is known

27.1 What is approximation?

We come across approximation very often. For example, the irrational number π can be approximated by the number 3.14. A more accurate value is 3.141593, which remains an approximation. You can similarly approximate the values of all irrational numbers like $\sqrt{3}$, $\sqrt{7}$, etc.

Approximation is used whenever a numerical value, a model, a structure or a function is either unknown or difficult to compute. In this chapter we'll focus on function approximation and describe its application to machine learning problems. There are two different cases:

1. The function is known but it is difficult or numerically expensive to compute its exact value. In this case approximation methods are used to find values, which are close to the function's actual values.

2. The function itself is unknown and hence a model or learning algorithm is used to closely find a function that can produce outputs close to the unknown function's outputs.

27.2 Approximation when form of function is known

If the form of a function is known, then a well known method in calculus and mathematics is approximation via Taylor series. The Taylor series of a function is the sum of infinite terms, which are computed using function's derivatives. The Taylor series expansion of a function is discussed in the next chapter.

Another well known method for approximation in calculus and mathematics is Newton's method[1]. It can be used to approximate the roots of polynomials, hence making it a useful technique for approximating quantities such as the square root of different values or the reciprocal of different numbers, etc.

27.3 Approximation when form of function is unknown

In data science and machine learning, it is assumed that there is an underlying function that holds the key to the relationship between the inputs and outputs. The form of this function is unknown. Here, we discuss several machine learning problems that employ approximation.

Approximation in regression

Regression involves the prediction of an output variable when given a set of inputs. In regression, the function that truly maps the input variables to outputs is not known. It is assumed that some linear or nonlinear regression model can approximate the mapping of inputs to outputs.

For example, we may have data related to consumed calories per day and the corresponding blood sugar. To describe the relationship between the calorie input and blood sugar output, we can assume a straight line relationship/mapping function. The straight line is therefore the approximation of the mapping of inputs to outputs. A learning method such as the method of least squares is used to find this line.

[1] https://en.wikipedia.org/wiki/Newton%27s_method

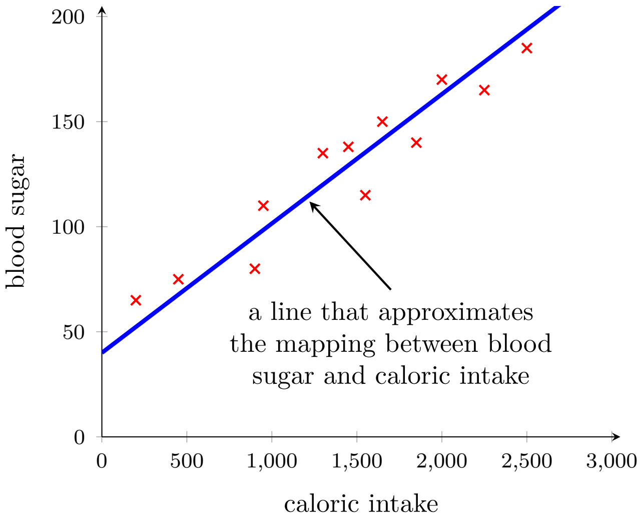


Figure 27.1: A straight line approximation to relationship between caloric count and blood sugar

Approximation in classification

A classic example of models that approximate functions in classification problems is that of neural networks. It is assumed that the neural network as a whole can approximate a true function that maps the inputs to the class labels. Gradient descent or some other learning algorithm is then used to learn that function approximation by adjusting the weights of the neural network.

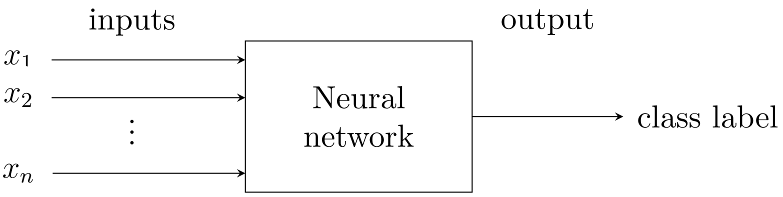


Figure 27.2: A neural network approximates an underlying function that maps inputs to outputs

Approximation in unsupervised learning

Below is a typical example of unsupervised learning. Here we have points in 2D space and the label of none of these points is given. A clustering algorithm generally assumes a model according to which a point can be assigned to a class or label. For example, k-means learns the labels of data by assuming that data clusters are circular, and hence, assigns the same label or class to points lying in the same circle or an n-sphere in case of multi-dimensional data. In the figure below we are approximating the relationship between points and their labels via circular functions.

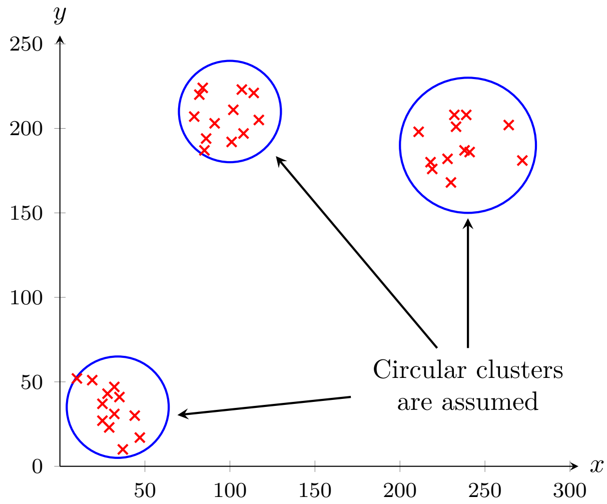


Figure 27.3: A clustering algorithm approximates a model that determines clusters or unknown labels of input points

27.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

27.5 Summary

In this tutorial, you discovered what is approximation. Specifically, you learned:

- ▷ Approximation
- ▷ Approximation when the form of a function is known
- ▷ Approximation when the form of a function is unknown

In the next chapter, we will see a concrete example, the Taylor series.

Taylor Series

28

Taylor series expansion is an awesome concept, not only the world of mathematics, but also in optimization theory, function approximation and machine learning. It is widely applied in numerical computations when estimates of a function's values at different points are required.

In this tutorial, you will discover Taylor series and how to approximate the values of a function around different points using its Taylor series expansion. After completing this tutorial, you will know:

- ▷ Taylor series expansion of a function
- ▷ How to approximate functions using Taylor series expansion

Let's get started.

Overview

This tutorial is divided into 3 parts; they are:

- ▷ Power series and Taylor series
- ▷ Taylor polynomials
- ▷ Function approximation using Taylor polynomials

28.1 What is a power series?

The following is a power series about the center $x = a$ and constant coefficients c_0, c_1, etc.

$$\sum_{n=0}^{\infty} c_n(x-a)^n = c_0 + c_1(x-a) + c_2(x-a)^2 + \cdots + c_k(x-a)^k + \cdots$$

28.2 What is a Taylor series?

It is an amazing fact that functions which are infinitely differentiable can generate a power series called the Taylor series. Suppose we have a function $f(x)$ and $f(x)$ has derivatives of all orders on a given interval, then the Taylor series generated by $f(x)$ at $x = a$ is given by:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + \cdots$$

$$c_k = \frac{f^{(k)}(a)}{k!}$$

The second line of the above expression gives the value of the k-th coefficient.

If we set $a = 0$, then we have an expansion called the Maclaurin series expansion of $f(x)$.

28.3 Examples of Taylor series expansion

Taylor series generated by $f(x) = 1/x$ can be found by first differentiating the function and finding a general expression for the k-th derivative.

$$f(x) = \frac{1}{x} \qquad f'(x) = \frac{-1}{x^2} \qquad f''(x) = \frac{2}{x^3} \qquad f^{(k)} = (-1)^k \frac{k!}{x^{k+1}}$$

The Taylor series about various points can now be found. For example:

Setting $a = 1$,

$$\sum_{n=0}^{\infty} \frac{(x-1)^n}{x^{n+1}} = 1 - (x-1) + (x-1)^2 + \cdots + (-1)^k(x-1)^k + \cdots$$

Setting $a = 3$,

$$\sum_{n=0}^{\infty} \frac{(x-3)^n}{x^{n+1}} = \frac{1}{3} - \frac{x-3}{3^2} + \frac{(x-3)^2}{3^3} + \cdots + (-1)^k \frac{(x-3)^k}{3^{k+1}} + \cdots$$

28.4 Taylor polynomial

A Taylor polynomial of order k, generated by $f(x)$ at $x = a$ is given by:

$$P_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k$$

That is, it is the Taylor series expansion clipped at the k-th order. For the example of $f(x) = 1/x$, the Taylor polynomial of order 2 is given by:

At $a = 1$,

$$P_2(x) = 1 - (x-1) - (x-1)^2$$

At $a = 3$,

$$P_3(x) = \frac{1}{3} - \frac{x-3}{3^2} - \frac{(x-3)^2}{3^3}$$

28.5 Approximation via Taylor polynomials

We can approximate the value of a function at a point $x = a$ using Taylor polynomials. The higher the order of the polynomial, the more the terms in the polynomial and the closer the approximation is to the actual value of the function at that point.

In the graph below, the function $\frac{1}{x}$ is plotted around the point $x = 1$ (left) and $x = 3$ (right). The line in green is the actual function $f(x) = \frac{1}{x}$. The pink line represents the approximation via an order 2 polynomial.

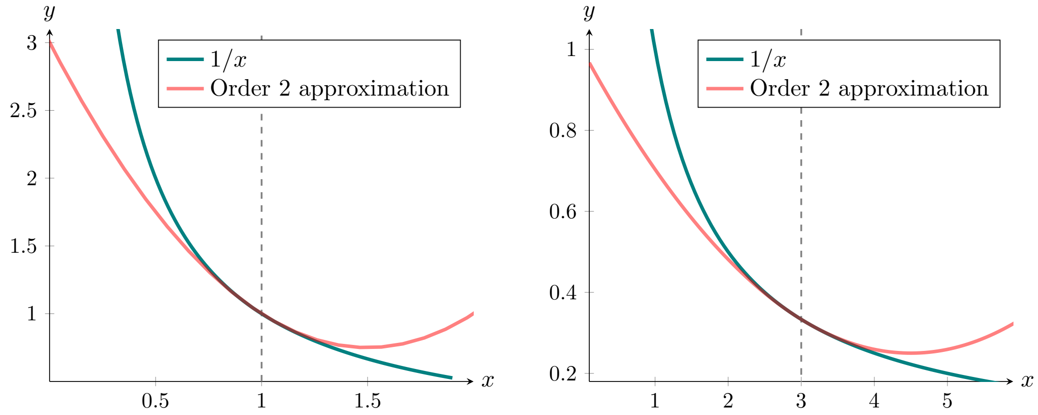


Figure 28.1: The actual function (green) and its approximation (red)

28.6 More examples of Taylor series

Let's look at the function $g(x) = e^x$. Noting the fact that the k-th order derivative of $g(x)$ is also $g(x)$, the expansion of $g(x)$ about $x = a$, is given by:

$$e^a + e^a(x-a) + \frac{e^a}{2!}(x-a)^2 + \cdots + \frac{e^a}{k!}(x-a)^k + \cdots$$

Hence, around $x = 0$, the series expansion of $g(x)$ is given by (obtained by setting $a = 0$):

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

The polynomial of order k generated for the function e^x around the point $x = 0$ is given by:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^k}{k!}$$

The plots below show polynomials of different orders that estimate the value of e^x around $x = 0$. We can see that as we move away from zero, we need more terms to approximate e^x more accurately. The green line representing the actual function is hiding behind the blue line of the approximating polynomial of order 7.

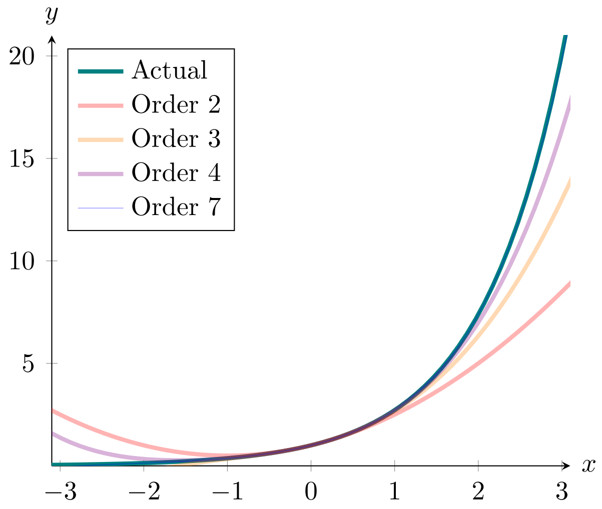


Figure 28.2: Polynomials of varying degrees that approximate e^x

28.7 Taylor series in machine learning

A popular method in machine learning for finding the optimal points of a function is the Newton's method. Newton's method uses the second order polynomials to approximate a function's value at a point. Such methods that use second order derivatives are called second order optimization algorithms.

28.8 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
https://amzn.to/3qSk3C2

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

28.9 Summary

In this tutorial, you discovered what is Taylor series expansion of a function about a point. Specifically, you learned:

- Power series and Taylor series
- Taylor polynomials
- How to approximate functions around a value using Taylor polynomials

In the next chapter, we will see some examples in machine learning that benefited directly from calculus.

VI
Calculus in Machine Learning

Gradient Descent Procedure 29

Gradient descent procedure is a method that holds paramount importance in machine learning. It is often used for minimizing error functions in classification and regression problems. It is also used in training neural networks, and deep learning architectures.

In this tutorial, you will discover the gradient descent procedure. After completing this tutorial, you will know:

- Gradient descent method
- Importance of gradient descent in machine learning

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- Gradient descent procedure
- Solved example of gradient descent procedure

29.1 Gradient descent procedure

The gradient descent procedure is an algorithm for finding the minimum of a function.

Suppose we have a function $f(x)$, where x is a tuple of several variables, i.e., $x = (x_1, x_2, \ldots x_n)$. Also, suppose that the gradient of $f(x)$ is given by $\nabla f(x)$. We want to find the value of the variables $(x_1, x_2, \ldots x_n)$ that give us the minimum of the function. At any iteration t, we'll denote the value of the tuple x by $x[t]$. So $x[t][1]$ is the value of x_1 at iteration t, $x[t][2]$ is the value of x_2 at iteration t, etc.

The notation

We have the following variables:

- t = Iteration number
- T = Total iterations

- n = Total variables in the domain of f (also called the dimensionality of x)
- j = Iterator for variable number, e.g., x_j represents the j-th variable
- η = Learning rate
- $\nabla f(x[t])$ = Value of the gradient vector of f at iteration t

The training method

The steps for the gradient descent algorithm are given below. This is also called the training method.

1. Choose a random initial point x_{initial} and set $x[0] = x_{\text{initial}}$
2. For iterations $t = 1 \ldots T$
    - Update $x[t] = x[t-1] - \eta \nabla f(x[t-1])$

It is as simple as that!

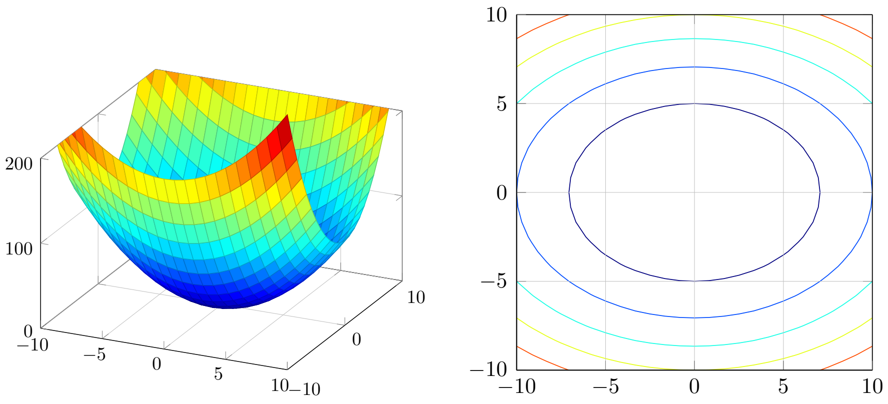
The learning rate η is a user defined variable for the gradient descent procedure. Its value lies in the range $[0, 1]$.

The above method says that at each iteration we have to update the value of x by taking a small step in the direction of the negative of the gradient vector. If $\eta = 0$,, then there will be no change in x. If $\eta = 1$, then it is like taking a large step in the direction of the negative of the gradient of the vector. Normally, η is set to a small value like 0.05 or 0.1. It can also be variable during the training procedure. So your algorithm can start with a large value (e.g. 0.8) and then reduce it to smaller values.

29.2 Example of gradient descent

Let's find the minimum of the following function of two variables, whose graphs and contours are shown in the figure below:

$$f(x, y) = x \times x + 2 \times y \times y$$



$f(x, y) = x^2 + 2y^2$

Contours of $f(x, y) = x^2 + 2y^2$

Figure 29.1: Graph and contours of $f(x, y) = x \times x + 2 \times y \times y$

The general fmrm of the gradient vector is given by:

$$\nabla f(x, y) = 2x\mathbf{i} + 4y\mathbf{j}$$

Two iterations of the algorithm, $T = 2$ and $\eta = 0.1$ are shown below

1. Initial $t = 0$,

   $$x[0] = (4, 3)$$

   (This is just a randomly chosen initial point)

2. At $t = 1$,

   $$\begin{aligned} x[1] &= x[0] - \eta\nabla f(x[0]) \\ &= (4, 3) - 0.1 \times (8, 12) \\ &= (3.2, 1.8) \end{aligned}$$

3. At $t = 2$,

   $$\begin{aligned} x[2] &= x[1] - \eta\nabla f(x[1]) \\ &= (3.2, 1.8) - 0.1 \times (6.4, 7.2) \\ &= (2.56, 1.08) \end{aligned}$$

If you keep running the above iterations, the procedure will eventually end up at the point where the function is minimum, i.e., (0,0). At iteration $t = 1$, the algorithm is illustrated in the figure below:

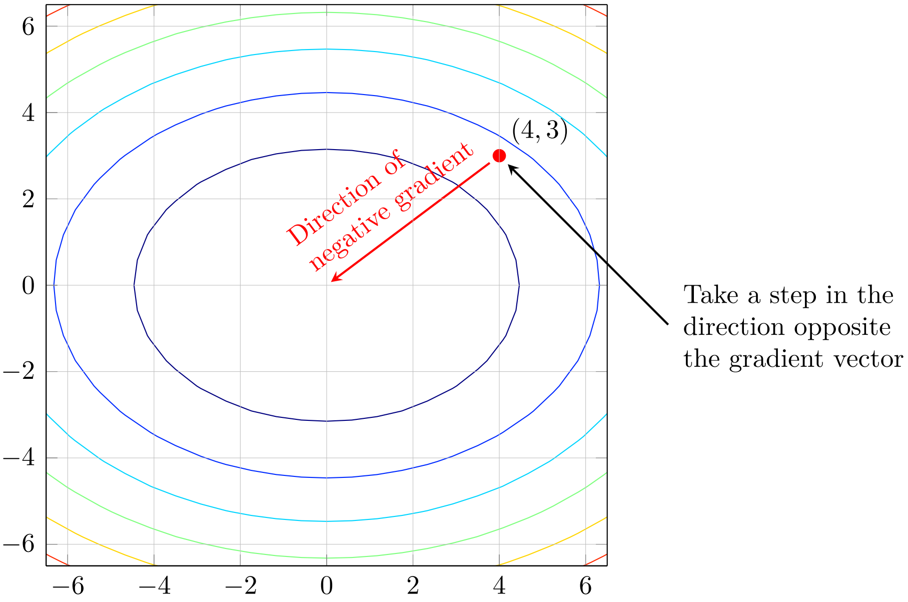


Figure 29.2: Illustration of gradient descent procedure

29.3 How many iterations to run?

Normally gradient descent is run till the value of x does not change or the change in x is below a certain threshold. The stopping criterion can also be a user defined maximum number of iterations (that we defined earlier as T).

29.4 Adding momentum

Gradient descent can run into problems such as:

1. Oscillate between two or more points
2. Get trapped in a local minimum
3. Overshoot and miss the minimum point

To take care of the above problems, a momentum term can be added to the update equation of gradient descent algorithm as:

$$x[t] = x[t-1] - \eta \times \nabla f(x[t-1]) + \alpha \times \Delta x[t-1]$$

where $\Delta x[t-1]$ represents the change in x, i.e.,

$$\Delta x[t] = x[t] - x[t-1]$$

The initial change at $t = 0$ is a zero vector. For this problem $\Delta x[0] = (0, 0)$.

29.5 About gradient ascent

There is a related gradient ascent procedure, which finds the maximum of a function. In gradient descent we follow the direction of the rate of maximum decrease of a function. It is the direction of the negative gradient vector. Whereas, in gradient ascent we follow the direction of maximum rate of increase of a function, which is the direction pointed to by the positive gradient vector. We can also write a maximization problem in terms of a maximization problem by adding a negative sign to $f(x)$, i.e.,

$$\text{maximize } f(x) \text{ with respect to } x$$

is equivalent to

$$\text{minimize } -f(x) \text{ with respect to } x$$

29.6 Why is the gradient descent important in machine learning?

The gradient descent algorithm is often employed in machine learning problems. In many classification and regression tasks, the mean square error function is used to fit a model to the data. The gradient descent procedure is used to identify the optimal model parameters that lead to the lowest mean square error.

Gradient ascent is used similarly, for problems that involve maximizing a function.

29.7 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Gilbert Strang. *Calculus*. 3rd ed. Wellesley-Cambridge Press, 2017.
https://amzn.to/3fqNSEB
(The 1991 edition is available online: https://ocw.mit.edu/resources/res-18-001-calculus-online-textbook-spring-2005/textbook/).

James Stewart. *Calculus*. 8th ed. Cengage Learning, 2013.
https://amzn.to/3kS9I52

29.8 Summary

In this tutorial, you discovered the algorithm for gradient descent. Specifically, you learned:

- Gradient descent procedure
- How to apply gradient descent procedure to find the minimum of a function
- How to transform a maximization problem into a minimization problem

In the next chapter, we will learn about neural networks, which is the famous use case of gradient descent.

30 Calculus in Neural Networks

An artificial neural network is a computational model that approximates a mapping between inputs and outputs. It is inspired by the structure of the human brain, in that it is similarly composed of a network of interconnected neurons that propagate information upon receiving sets of stimuli from neighboring neurons. Training a neural network involves a process that employs the backpropagation and gradient descent algorithms in tandem. As we will be seeing, both of these algorithms make extensive use of calculus.

In this tutorial, you will discover how aspects of calculus are applied in neural networks. After completing this tutorial, you will know:

- ▷ An artificial neural network is organized into layers of neurons and connections, where the latter are attributed a weight value each.
- ▷ Each neuron implements a nonlinear function that maps a set of inputs to an output activation.
- ▷ In training a neural network, calculus is used extensively by the backpropagation and gradient descent algorithms.

Let's get started.

Overview

This tutorial is divided into three parts; they are:

- ▷ An Introduction to the Neural Network
- ▷ The Mathematics of a Neuron
- ▷ Training the Network

30.1 An introduction to the neural network

Artificial neural networks can be considered as function approximation algorithms. In a supervised learning setting, when presented with many input observations representing the problem of interest, together with their corresponding target outputs, the artificial neural network will seek to approximate the mapping that exists between the two.

> A neural network is a computational model that is inspired by the structure of the human brain.
>
> — Page 65, *Deep Learning*, 2019.

The human brain consists of a massive network of interconnected neurons (around one hundred billion of them), with each comprising a cell body, a set of fibers called dendrites, and an axon:

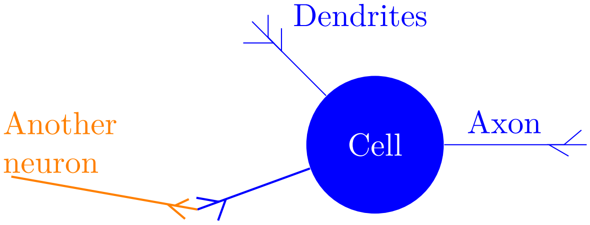


Figure 30.1: A neuron in the human brain

The dendrites act as the input channels to a neuron, whereas the axon acts as the output channel. Therefore, a neuron would receive input signals through its dendrites, which in turn would be connected to the (output) axons of other neighboring neurons. In this manner, a sufficiently strong electrical pulse (also called an action potential) can be transmitted along the axon of one neuron, to all the other neurons that are connected to it. This permits signals to be propagated along the structure of the human brain.

> So, a neuron acts as an all-or-none switch, that takes in a set of inputs and either outputs an action potential or no output.
>
> — Page 66, *Deep Learning*, 2019.

An artificial neural network is analogous to the structure of the human brain, because (1) it is similarly composed of a large number of interconnected neurons that, (2) seek to propagate information across the network by, (3) receiving sets of stimuli from neighboring neurons and mapping these to outputs, to be fed to the next layer of neurons.

The structure of an artificial neural network is typically organized into layers of neurons (recall the depiction of a tree diagram). For example, the following diagram illustrates a fully-connected neural network, where all the neurons in one layer are connected to all the neurons in the next layer:

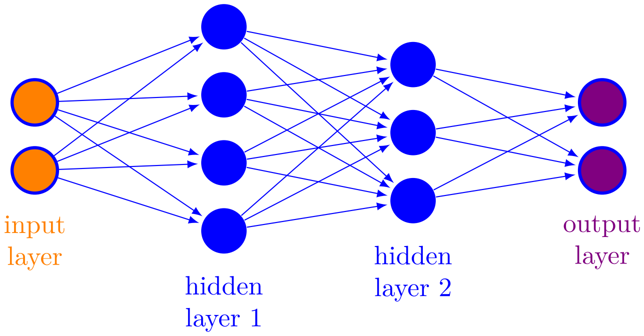


Figure 30.2: A fully-connected, feedforward neural network

The inputs are presented on the left hand side of the network, and the information propagates (or flows) rightward towards the outputs at the opposite end. Since the information is, hereby, propagating in the *forward* direction through the network, then we would also refer to such a network as a *feedforward neural network*.

The layers of neurons in between the input and output layers are called *hidden* layers, because they are not directly accessible. Each connection (represented by an arrow in the diagram) between two neurons is attributed a weight, which acts on the data flowing through the network, as we will see shortly.

30.2 The mathematics of a neuron

More specifically, let's say that a particular artificial neuron (or a *perceptron*, as Frank Rosenblatt had initially named it) receives n inputs, $[x_1, \dots, x_n]$, where each connection is attributed a corresponding weight, $[w_1, \dots, w_n]$.

The first operation that is carried out multiplies the input values by their corresponding weight, and adds a bias term, b, to their sum, producing an output, z:

$$z = ((x_1 \times w_1) + (x_2 \times w_2) + \ldots + (x_n \times w_n)) + b$$

We can, alternatively, represent this operation in a more compact form as follows:

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

This weighted sum calculation that we have performed so far is a linear operation. If every neuron had to implement this particular calculation alone, then the neural network would be restricted to learning only linear input-output mappings.

> However, many of the relationships in the world that we might want to model are nonlinear, and if we attempt to model these relationships using a linear model, then the model will be very inaccurate.
>
> — Page 77, *Deep Learning*, 2019.

Hence, a second operation is performed by each neuron that transforms the weighted sum by the application of a nonlinear activation function, $a(\cdot)$:

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

We can represent the operations performed by each neuron even more compactly, if we had to integrate the bias term into the sum as another weight, w_0 (notice that the sum now starts from 0):

$$y = a(z) = a\left(\sum_{i=0}^{n} x_i \times w_i \right)$$

The operations performed by each neuron can be illustrated as follows:

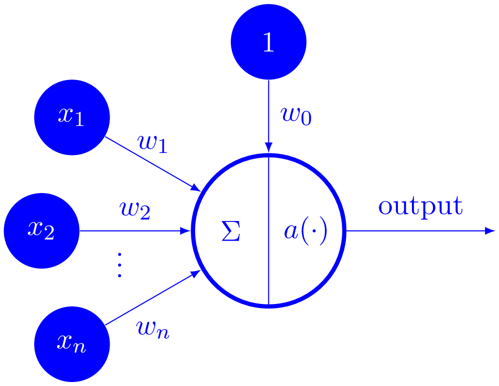


Figure 30.3: Nonlinear function implemented by a neuron

Therefore, each neuron can be considered to implement a nonlinear function that maps a set of inputs to an output activation.

30.3 Training the network

Training an artificial neural network involves the process of searching for the set of weights that model best the patterns in the data. It is a process that employs the backpropagation and gradient descent algorithms in tandem. Both of these algorithms make extensive use of calculus.

Each time that the network is traversed in the forward (or rightward) direction, the error of the network can be calculated as the difference between the output produced by the network and the expected ground truth, by means of a loss function (such as the sum of squared errors, SSE). The backpropagation algorithm, then, calculates the gradient (or the rate of change) of this error to changes in the weights. In order to do so, it requires the use of the chain rule and partial derivatives.

For simplicity, consider a network made up of two neurons connected by a single path of activation. If we had to break them open, we would find that the neurons perform the following operations in cascade:

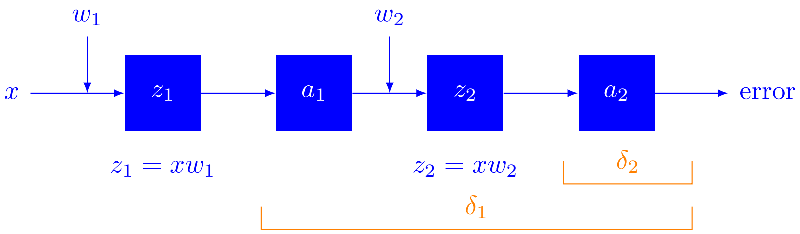


Figure 30.4: Operations performed by two neurons in cascade

The first application of the chain rule connects the overall error of the network to the input, z_2, of the activation function a_2 of the second neuron, and subsequently to the weight, w_2, as follows:

$$\frac{\partial(\text{error})}{\partial w_2} = \frac{\partial(\text{error})}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_2} = \delta_2 \times \frac{\partial z_2}{\partial w_2}$$

You may notice that the application of the chain rule involves, among other terms, a multiplication by the partial derivative of the neuron's activation function with respect to its input, z_2. There are different activation functions to choose from, such as the sigmoid or the logistic functions. If we had to take the logistic function as an example, then its partial derivative would be computed as follows:

$$\frac{\partial a_2}{\partial z_2} = \frac{\text{logistic}(z)}{\partial z_2} = \text{logistic}(z_2) \times (1 - \text{logistic}(z_2))$$

Hence, we can compute δ_2 as follows:

$$\delta_2 = \text{logistic}(z_2) \times (1 - \text{logistic}(z_2)) \times (t_2 - a_2)$$

Here, t_2 is the expected activation, and in finding the difference between t_2 and a_2 we are, therefore, computing the error between the activation generated by the network and the expected ground truth.

Since we are computing the derivative of the activation function, it should, therefore, be continuous and differentiable over the entire space of real numbers. In the case of deep neural networks, the error gradient is propagated backwards over a large number of hidden layers. This can cause the error signal to rapidly diminish to zero, especially if the maximum value of the derivative function is already small to begin with (for instance, the inverse of the logistic function has a maximum value of 0.25). This is known as the *vanishing gradient problem.* The ReLU function has been so popularly used in deep learning to alleviate this problem, because its derivative in the positive portion of its domain is equal to 1.

The next weight backwards is deeper into the network and, hence, the application of the chain rule can similarly be extended to connect the overall error to the weight, w_1, as follows:

$$\frac{\partial(\text{error})}{\partial w_1} = \frac{\partial(\text{error})}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1} = \delta_1 \times \frac{\partial z_1}{\partial w_1}$$

If we take the logistic function again as the activation function of choice, then we would compute δ_1 as follows:

$$\begin{aligned}\delta_1 &= \underbrace{\frac{\partial(\text{error})}{\partial a_2} \times \frac{\partial a_2}{\partial z_2}}_{\delta_2} \times \underbrace{\frac{\partial z_2}{\partial a_1}}_{w_2} \times \underbrace{\frac{\partial a_1}{\partial z_1}}_{\text{logistic}} \\ &= (\delta_2 \times w_2) \times \text{logistic}(z_1) \times (1 - \text{logistic}(z_1))\end{aligned}$$

Once we have computed the gradient of the network error with respect to each weight, then the gradient descent algorithm can be applied to update each weight for the next *forward propagation* at time, $t + 1$. For the weight, w_1, the weight update rule using gradient descent would be specified as follows:

$$w_1^{t+1} = w_1^t + \left(\eta \times \delta_1 \times \frac{\partial z_1}{\partial w_1}\right)$$

Even though we have hereby considered a simple network, the process that we have gone through can be extended to evaluate more complex and deeper ones, such convolutional neural networks (CNNs).

If the network under consideration is characterized by multiple branches coming from multiple inputs (and possibly flowing towards multiple outputs), then its evaluation would involve the summation of different derivative chains for each path, similarly to how we have previously derived the generalized chain rule.

30.4 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

John D. Kelleher. *Deep Learning*. Illustrated edition. The MIT Press Essential Knowledge series. MIT Press, 2019.
https://www.amazon.com/dp/0262537559/

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

30.5 Summary

In this tutorial, you discovered how aspects of calculus are applied in neural networks. Specifically, you learned:

- An artificial neural network is organized into layers of neurons and connections, where the latter are each attributed a weight value.
- Each neuron implements a nonlinear function that maps a set of inputs to an output activation.
- In training a neural network, calculus is used extensively by the backpropagation and gradient descent algorithms.

In the next chapter, we will combine what we learned in these two chapter and implement a neural network training process from scratch.

Implementing a Neural Network in Python 31

Differential calculus is an important tool in machine learning algorithms. In neural networks in particular, the gradient descent algorithm depends on the gradient, which is a quantity computed by differentiation.

In this tutorial, we will see how the backpropagation technique is used to find the gradients in neural networks.

After completing this tutorial, you will know:

- What is a total differential and total derivative
- How to compute the total derivatives in neural networks
- How backpropagation helped in computing the total derivatives

Let's get started.

Overview

This tutorial is divided into five parts; they are:

- Total differential and total derivatives
- Algebraic representation of a multilayer perceptron model
- Finding the gradient by backpropagation
- Matrix form of gradient equations
- Implementing backpropagation

31.1 Total differential and total derivatives

For a function such as $f(x)$, we can denote its derivative as $f'(x)$ or $\frac{df}{dx}$. But for a multivariate function, such as $f(u, v)$, we have a partial derivative of f with respect to u denoted as $\frac{\partial f}{\partial u}$, or sometimes written as f_u. A partial derivative is obtained by the differentiation of f with respect to u while assuming the other variable v is a constant. Therefore, we use ∂ instead of d as the symbol for differentiation to signify the difference.

However, what if the u and v in $f(u, v)$ are both functions of x? In other words, we can write $u(x)$ and $v(x)$ and $f(u(x), v(x))$. So x determines the value of u and v, and in turn, determines $f(u, v)$. In this case, it is perfectly fine to ask what is $\frac{df}{dx}$, as f is eventually determined by x.

This is the concept of total derivatives. In fact, for a multivariate function $f(t, u, v) = f(t(x), u(x), v(x))$, we always have:

$$\frac{df}{dx} = \frac{\partial f}{\partial t}\frac{dt}{dx} + \frac{\partial f}{\partial u}\frac{du}{dx} + \frac{\partial f}{\partial v}\frac{dv}{dx}$$

The above notation is called the total derivative because it is sum of the partial derivatives. In essence, it is applying the chain rule to find the differentiation.

If we take away the dx part in the above equation, what we get is an approximate change in f with respect to x, i.e.,

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial u}du + \frac{\partial f}{\partial v}dv$$

We call this notation the total differential.

31.2 Algebraic representation of a multilayer perceptron model

Consider the network in Figure 31.1. This is a simple, fully-connected, 4-layer neural network. Let's call the input layer as layer 0, the two hidden layers as layer 1 and 2, and the output layer as layer 3. In this picture, we see that we have $n_0 = 3$ input units, $n_1 = 4$ units in the first hidden layer and $n_2 = 2$ units in the second input layer. There are $n_3 = 2$ output units.

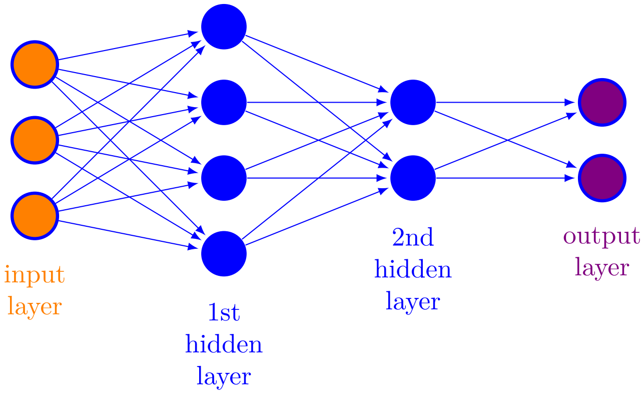


Figure 31.1: An example of neural network

If we denote the input to the network as x_i where $i = 1, \cdots, n_0$ and the network's output as $\hat{y}_i$ where $i = 1, \cdots, n_3$, then we can write:

$$
\begin{aligned}
h_{1i} &= f_1(\sum_{j=1}^{n_0} w_{ij}^{(1)} x_j + b_i^{(1)}) && \text{for } i = 1, \cdots, n_1 \\
h_{2i} &= f_2(\sum_{j=1}^{n_1} w_{ij}^{(2)} h_{1j} + b_i^{(2)}) && i = 1, \cdots, n_2 \\
\hat{y}_i &= f_3(\sum_{j=1}^{n_2} w_{ij}^{(3)} h_{2j} + b_i^{(3)}) && i = 1, \cdots, n_3
\end{aligned}
$$

Here the activation function at layer i is denoted as f_i. The outputs of the first hidden layer are denoted as h_{1i} for the i-th unit. Similarly, the outputs of the second hidden layer are denoted as h_{2i}. The weights and bias of unit i in layer k are denoted as $w_{ij}^{(k)}$ and $b_i^{(k)}$, respectively.

In the above, we can see that the output of layer $k-1$ will feed into layer k. Therefore, while $\hat{y}_i$ is expressed as a function of h_{2j}, h_{2i} is also a function of h_{1j}, and in turn, a function of x_j.

The above describes the construction of a neural network in terms of algebraic equations. Training a neural network would need to specify a *loss function* as well so we can minimize it in the training loop. Depending on the application, we commonly use cross entropy for categorization problems or mean squared error for regression problems. For example, with the target variables as y_i, the mean square error loss function is specified as:

$$L = \sum_{i=1}^{n_3} (y_i - \hat{y}_i)^2$$

31.3 Finding the gradient by backpropagation

In the above construct, x_i and y_i are from the dataset. The parameters to the neural network are w and b. While the activation functions f_i are by design, the outputs at each layer h_{1i}, h_{2i}, and $\hat{y}_i$ are dependent variables with respect to the dataset and parameters. In training the network, our goal is to update w and b in each iteration using the gradient descent update rule:

$$
\begin{aligned}
w_{ij}^{(k)} &= w_{ij}^{(k)} - \eta \frac{\partial L}{\partial w_{ij}^{(k)}} \\
b_i^{(k)} &= b_i^{(k)} - \eta \frac{\partial L}{\partial b_i^{(k)}}
\end{aligned}
$$

where η is the learning rate parameter to gradient descent.

From the equation of L, we know that L is not dependent on $w_{ij}^{(k)}$ or $b_i^{(k)}$ but on $\hat{y}_i$. However, $\hat{y}_i$ can be written as function of $w_{ij}^{(k)}$ or $b_i^{(k)}$ eventually. Let's see one by one how the weights and bias at layer k can be connected to $\hat{y}_i$ at the output layer.

We begin with the loss metric. If we consider the loss of a single data point, we have:

$$\begin{aligned} L &= \sum_{i=1}^{n_3}(y_i - \hat{y}_i)^2 \\ \frac{\partial L}{\partial \hat{y}_i} &= 2(\hat{y}_i - y_i) && \text{for } i = 1, \cdots, n_3 \end{aligned}$$

Here we see that the loss function depends on all outputs $\hat{y}_i$, and therefore, we can find a partial derivative $\frac{\partial L}{\partial \hat{y}_i}$.

Now let's look at the output layer:

$$\begin{aligned} \hat{y}_i &= f_3(\sum_{j=1}^{n_2} w_{ij}^{(3)} h_{2j} + b_i^{(3)}) && \text{for } i = 1, \cdots, n_3 \\ \frac{\partial L}{\partial w_{ij}^{(3)}} &= \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_{ij}^{(3)}} && i = 1, \cdots, n_3;\ j = 1, \cdots, n_2 \\ &= \frac{\partial L}{\partial \hat{y}_i} f_3'(\sum_{j=1}^{n_2} w_{ij}^{(3)} h_{2j} + b_i^{(3)}) h_{2j} \\ \frac{\partial L}{\partial b_i^{(3)}} &= \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial b_i^{(3)}} && i = 1, \cdots, n_3 \\ &= \frac{\partial L}{\partial \hat{y}_i} f_3'(\sum_{j=1}^{n_2} w_{ij}^{(3)} h_{2j} + b_i^{(3)}) \end{aligned}$$

Because the weight $w_{ij}^{(3)}$ at layer 3 applies to input h_{2j} and affects output $\hat{y}_i$ only. Hence we can write the derivative $\frac{\partial L}{\partial w_{ij}^{(3)}}$ as the product of two derivatives $\frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_{ij}^{(3)}}$. Similar case for the bias $b_i^{(3)}$ as well. In the above, we make use of $\frac{\partial L}{\partial \hat{y}_i}$, which we already derived previously.

But in fact, we can also write the partial derivative of L with respect to the output of the second layer h_{2j}. It is not used for the update of weights and bias on layer 3, but we will see its importance later:

$$\begin{aligned} \frac{\partial L}{\partial h_{2j}} &= \sum_{i=1}^{n_3} \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_{2j}} && \text{for } j = 1, \cdots, n_2 \\ &= \sum_{i=1}^{n_3} \frac{\partial L}{\partial \hat{y}_i} f_3'(\sum_{j=1}^{n_2} w_{ij}^{(3)} h_{2j} + b_i^{(3)}) w_{ij}^{(3)} \end{aligned}$$

This one is the interesting one and different from the previous partial derivatives. Note that h_{2j} is an output of layer 2. Each and every output in layer 2 will affect the output $\hat{y}_i$ in layer 3. Therefore, to find $\frac{\partial L}{\partial h_{2j}}$, we need to add up every output at layer 3. Thus the summation sign in the equation above. And we can consider $\frac{\partial L}{\partial h_{2j}}$ as the total derivative, in which we applied the chain rule $\frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_{2j}}$ for every output i and then summed them up.

If we move back to layer 2, we can derive the derivatives similarly:

$$
\begin{aligned}
h_{2i} &= f_2(\sum_{j=1}^{n_1} w_{ij}^{(2)} h_{1j} + b_i^{(2)}) && \text{for } i = 1, \cdots, n_2 \\
\frac{\partial L}{\partial w_{ij}^{(2)}} &= \frac{\partial L}{\partial h_{2i}} \frac{\partial h_{2i}}{\partial w_{ij}^{(2)}} && i = 1, \cdots, n_2;\ j = 1, \cdots, n_1 \\
&= \frac{\partial L}{\partial h_{2i}} f_2'(\sum_{j=1}^{n_1} w_{ij}^{(2)} h_{1j} + b_i^{(2)}) h_{1j} \\
\frac{\partial L}{\partial b_i^{(2)}} &= \frac{\partial L}{\partial h_{2i}} \frac{\partial h_{2i}}{\partial b_i^{(2)}} && i = 1, \cdots, n_2 \\
&= \frac{\partial L}{\partial h_{2i}} f_2'(\sum_{j=1}^{n_1} w_{ij}^{(2)} h_{1j} + b_i^{(2)}) \\
\frac{\partial L}{\partial h_{1j}} &= \sum_{i=1}^{n_2} \frac{\partial L}{\partial h_{2i}} \frac{\partial h_{2i}}{\partial h_{1j}} && j = 1, \cdots, n_1 \\
&= \sum_{i=1}^{n_2} \frac{\partial L}{\partial h_{2i}} f_2'(\sum_{j=1}^{n_1} w_{ij}^{(2)} h_{1j} + b_i^{(2)}) w_{ij}^{(2)}
\end{aligned}
$$

In the equations above, we are reusing $\frac{\partial L}{\partial h_{2i}}$ that we derived earlier. Again, this derivative is computed as a sum of several products from the chain rule. Also similar to the previous, we derived $\frac{\partial L}{\partial h_{1j}}$ as well. It is not used to train $w_{ij}^{(2)}$ nor $b_i^{(2)}$ but will be used for the layer prior. So for layer 1, we have:

$$
\begin{aligned}
h_{1i} &= f_1(\sum_{j=1}^{n_0} w_{ij}^{(1)} x_j + b_i^{(1)}) && \text{for } i = 1, \cdots, n_1 \\
\frac{\partial L}{\partial w_{ij}^{(1)}} &= \frac{\partial L}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{ij}^{(1)}} && i = 1, \cdots, n_1;\ j = 1, \cdots, n_0 \\
&= \frac{\partial L}{\partial h_{1i}} f_1'(\sum_{j=1}^{n_0} w_{ij}^{(1)} x_j + b_i^{(1)}) x_j
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial b_i^{(1)}} &= \frac{\partial L}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial b_i^{(1)}} && i = 1, \cdots, n_1 \\
&= \frac{\partial L}{\partial h_{1i}} f_1'(\sum_{j=1}^{n_0} w_{ij}^{(1)} x_j + b_i^{(1)})
\end{aligned}
$$

and this completes all the derivatives needed for training the neural network using the gradient descent algorithm.

Recall how we derived the above: We first start from the loss function L and find the derivatives one by one in the reverse order of the layers. We write down the derivatives on layer k and reuse it for the derivatives on layer $k-1$. While computing the output $\hat{y}_i$ from input x_i starts from layer 0 forward, computing gradients are in the reversed order. Hence the name "backpropagation."

31.4 Matrix form of gradient equations

While we did not use it above, it is cleaner to write the equations in vectors and matrices. We can rewrite the layers and the outputs as:

$$\mathbf{a}_k = f_k(\mathbf{z}_k) = f_k(\mathbf{W}_k\mathbf{a}_{k-1} + \mathbf{b}_k)$$

where $\mathbf{a}_k$ is a vector of outputs of layer k, and assume $\mathbf{a}_0 = \mathbf{x}$ is the input vector and $\mathbf{a}_3 = \hat{\mathbf{y}}$ is the output vector. Also denote $\mathbf{z}_k = \mathbf{W}_k\mathbf{a}_{k-1} + \mathbf{b}_k$ for convenience of notation.

Under such notation, we can represent $\dfrac{\partial L}{\partial \mathbf{a}_k}$ as a vector (so as $\mathbf{z}_k$ and $\mathbf{b}_k$) and $\dfrac{\partial L}{\partial \mathbf{W}_k}$ as a matrix. And then, if $\dfrac{\partial L}{\partial \mathbf{a}_k}$ is known, we have:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{z}_k} &= \frac{\partial L}{\partial \mathbf{a}_k} \odot f_k'(\mathbf{z}_k) \\
\frac{\partial L}{\partial \mathbf{W}_k} &= \left(\frac{\partial L}{\partial \mathbf{z}_k}\right)^\top \cdot \mathbf{a}_k \\
\frac{\partial L}{\partial \mathbf{b}_k} &= \frac{\partial L}{\partial \mathbf{z}_k} \\
\frac{\partial L}{\partial \mathbf{a}_{k-1}} &= \left(\frac{\partial \mathbf{z}_k}{\partial \mathbf{a}_{k-1}}\right)^\top \cdot \frac{\partial L}{\partial \mathbf{z}_k} = \mathbf{W}_k^\top \cdot \frac{\partial L}{\partial \mathbf{z}_k}
\end{aligned}$$

where $\dfrac{\partial \mathbf{z}_k}{\partial \mathbf{a}_{k-1}}$ is a Jacobian matrix as both $\mathbf{z}_k$ and $\mathbf{a}_{k-1}$ are vectors, and this Jacobian matrix happens to be $\mathbf{W}_k$.

31.5 Implementing backpropagation

We need the matrix form of equations because it will make our code simpler and avoid a lot of loops. Let's see how we can convert these equations into code and make a multilayer perceptron model for classification from scratch using numpy.

The first thing we need is to implement the activation function and the loss function. Both need to be differentiable functions or otherwise our gradient descent procedure will not work. Today, it is common to use ReLU activation in the hidden layers and sigmoid activation in the output layer. We define them as a function (which assumes the input as numpy array) as well as their differentiation:

```python
import numpy as np

# Find a small float to avoid division by zero
epsilon = np.finfo(float).eps

# Sigmoid function and its differentiation
def sigmoid(z):
    return 1/(1+np.exp(-z.clip(-500, 500)))
def dsigmoid(z):
    s = sigmoid(z)
    return 2 * s * (1-s)

# ReLU function and its differentiation
def relu(z):
    return np.maximum(0, z)
def drelu(z):
    return (z > 0).astype(float)
```

Program 31.1: Defining activation functions and their derivatives

We deliberately clip the input of the sigmoid function to between -500 to $+500$ to avoid overflow. Otherwise, these functions are trivial. Then for classification, we care about accuracy, but the accuracy function is not differentiable. Therefore, we use the cross entropy function as loss for training:

```python
# Loss function L(y, yhat) and its differentiation
def cross_entropy(y, yhat):
    """Binary cross entropy function
        L = - y log yhat - (1-y) log (1-yhat)

    Args:
        y, yhat (np.array): 1xn matrices which n are the number of data instances
    Returns:
        average cross entropy value of shape 1x1, averaging over the n instances
    """
    return ( -(y.T @ np.log(yhat.clip(epsilon)) +
               (1-y.T) @ np.log((1-yhat).clip(epsilon))
             ) / y.shape[1] )

def d_cross_entropy(y, yhat):
    """ dL/dyhat """
    return ( - np.divide(y, yhat.clip(epsilon))
             + np.divide(1-y, (1-yhat).clip(epsilon)) )
```

Program 31.2: Define the cross entropy loss function

In the above, we assume the output and the target variables are row matrices in numpy. Hence we use the dot product operator “@” to compute the sum and divide by the number of elements in the output. Note that this design is to compute the *average cross entropy* over a *batch* of samples.

Then we can implement our multilayer perceptron model. To make it easier to read, we want to create the model by providing the number of neurons at each layer as well as the activation function at the layers. But at the same time, we will also need the differentiation of the activation functions as well as the differentiation of the loss function for the training. The loss function itself, however, is not required but useful for us to track the progress. We create a class to encapsulate the entire model and define each layer k according to the formula:

$$\mathbf{a}_k = f_k(\mathbf{z}_k) = f_k(\mathbf{a}_{k-1}\mathbf{W}_k + \mathbf{b}_k)$$

```
class mlp:
    '''Multilayer perceptron using numpy
    '''
    def __init__(self, layersizes, activations, derivatives, lossderiv):
        ”””remember config, then initialize array to hold NN parameters
        without init”””
        # hold NN config
        self.layersizes = layersizes
        self.activations = activations
        self.derivatives = derivatives
        self.lossderiv = lossderiv
        # parameters, each is a 2D numpy array
        L = len(self.layersizes)
        self.z = [None] * L
        self.W = [None] * L
        self.b = [None] * L
        self.a = [None] * L
        self.dz = [None] * L
        self.dW = [None] * L
        self.db = [None] * L
        self.da = [None] * L

    def initialize(self, seed=42):
        np.random.seed(seed)
        sigma = 0.1
        for l, (n_in, n_out) in enumerate(zip(self.layersizes, self.layersizes[1:]), 1):
            self.W[l] = np.random.randn(n_in, n_out) * sigma
            self.b[l] = np.random.randn(1, n_out) * sigma

    def forward(self, x):
        self.a[0] = x
        for l, func in enumerate(self.activations, 1):
            # z = W a + b, with `a` as output from previous layer
            # `W` is of size rxs and `a` the size sxn with n the number of data
            # instances, `z` the size rxn, `b` is rx1 and broadcast to each
            # column of `z`
            self.z[l] = (self.a[l-1] @ self.W[l]) + self.b[l]
            # a = g(z), with `a` as output of this layer, of size rxn
            self.a[l] = func(self.z[l])
        return self.a[-1]
```

Program 31.3: Implementing a neural network using numpy

The variables in this class `z`, `W`, `b`, and `a` are for the forward pass, and the variables `dz`, `dW`, `db`, and `da` are their respective gradients that will be computed in the backpropagation. All these variables are presented as numpy arrays.

As we will see later, we are going to test our model using data generated by scikit-learn. Hence we will see our data in numpy array of shape “(number of samples, number of features).” Therefore, each sample is presented as a row on a matrix, and in function `forward()`, the weight matrix is right-multiplied to each input `a` to the layer. While the activation function and dimension of each layer can be different, the process is the same. Thus we transform the neural network’s input `x` to its output by a loop in the `forward()` function. The network’s output is simply the output of the last layer.

To train the network, we need to run the backpropagation after each forward pass. The backpropagation is to compute the gradient of the weight and bias of each layer, starting from the output layer to the input layer. With the equations we derived above, the backpropagation function is implemented as:

```python
class mlp:
    ...

    def backward(self, y, yhat):
        # first `da`, at the output
        self.da[-1] = self.lossderiv(y, yhat)
        for l, func in reversed(list(enumerate(self.derivatives, 1))):
            # compute the differentials at this layer
            self.dz[l] = self.da[l] * func(self.z[l])
            self.dW[l] = self.a[l-1].T @ self.dz[l]
            self.db[l] = np.mean(self.dz[l], axis=0, keepdims=True)
            self.da[l-1] = self.dz[l] @ self.W[l].T

    def update(self, eta):
        for l in range(1, len(self.W)):
            self.W[l] -= eta * self.dW[l]
            self.b[l] -= eta * self.db[l]
```

Program 31.4: Backpropagation function in neural network

The only difference here is that we compute `db` not for one training sample but for the entire batch. Since the loss function is the cross entropy averaged across the batch, we also compute `db` by averaging across the samples.

Up to here, we have completed our model. The `update()` function simply applies the gradients found by the backpropagation to the parameters `W` and `b` using the gradient descent update rule.

To test out our model, we make use of scikit-learn to generate a classification dataset:

```python
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

# Make data: Two circles on x-y plane as a classification problem
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
y = y.reshape(-1,1) # our model expects a 2D array of (n_sample, n_dim)
```

Program 31.5: Creating classification dataset

and then we build our model: Input is two-dimensional and output is one dimensional (logistic regression). We make two hidden layers of 4 and 3 neurons respectively:

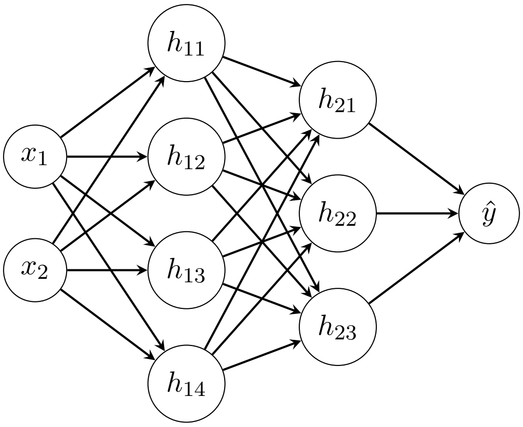


Figure 31.2: Neural network model for binary classification

```
# Build a model
model = mlp(layersizes=[2, 4, 3, 1],
            activations=[relu, relu, sigmoid],
            derivatives=[drelu, drelu, dsigmoid],
            lossderiv=d_cross_entropy)
model.initialize()
yhat = model.forward(X)
loss = cross_entropy(y, yhat)
score = accuracy_score(y, (yhat > 0.5))
print(f"Before training - loss value {loss} accuracy {score}")
```

Program 31.6: Create a neural network model and run one forward pass

We see that, under random weight, the accuracy is 50%:

```
Before training - loss value [[693.62972747]] accuracy 0.5
```

Output 31.1: Output of Program 31.6

Now we train our network. To make things simple, we perform full-batch gradient descent with fixed learning rate:

```
# train for each epoch
n_epochs = 150
learning_rate = 0.005
for n in range(n_epochs):
    model.forward(X)
    yhat = model.a[-1]
    model.backward(y, yhat)
    model.update(learning_rate)
    loss = cross_entropy(y, yhat)
    score = accuracy_score(y, (yhat > 0.5))
    print(f"Iteration {n} - loss value {loss} accuracy {score}")
```

Program 31.7: Training loop of the neural network

and the output is:

```
Iteration 0 - loss value [[693.62972747]] accuracy 0.5
Iteration 1 - loss value [[693.62166655]] accuracy 0.5
Iteration 2 - loss value [[693.61534159]] accuracy 0.5
Iteration 3 - loss value [[693.60994018]] accuracy 0.5
...
Iteration 145 - loss value [[664.60120828]] accuracy 0.818
Iteration 146 - loss value [[697.97739669]] accuracy 0.58
Iteration 147 - loss value [[681.08653776]] accuracy 0.642
Iteration 148 - loss value [[665.06165774]] accuracy 0.71
Iteration 149 - loss value [[683.6170298]] accuracy 0.614
```

Output 31.2: Output of Program 31.7

Although not perfect, we see the improvement by training. At least in the example above, we can see the accuracy was up to more than 80% at iteration 145, but then we saw the model diverged. That can be improved by reducing the learning rate, which we didn't implement above. Nonetheless, this shows how we computed the gradients by backpropagations and chain rules.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

The complete code is as follows:

```python
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
import numpy as np
np.random.seed(0)

# Find a small float to avoid division by zero
epsilon = np.finfo(float).eps

# Sigmoid function and its differentiation
def sigmoid(z):
    return 1/(1+np.exp(-z.clip(-500, 500)))
def dsigmoid(z):
    s = sigmoid(z)
    return 2 * s * (1-s)

# ReLU function and its differentiation
def relu(z):
    return np.maximum(0, z)
def drelu(z):
    return (z > 0).astype(float)

# Loss function L(y, yhat) and its differentiation
def cross_entropy(y, yhat):
    """Binary cross entropy function
        L = - y log yhat - (1-y) log (1-yhat)

    Args:
        y, yhat (np.array): nx1 matrices which n are the number of data instances
    Returns:
        average cross entropy value of shape 1x1, averaging over the n instances
    ”””
    return ( -(y.T @ np.log(yhat.clip(epsilon)) +
               (1-y.T) @ np.log((1-yhat).clip(epsilon))
              ) / y.shape[1] )

def d_cross_entropy(y, yhat):
    ””” dL/dyhat ”””
    return ( - np.divide(y, yhat.clip(epsilon))
             + np.divide(1-y, (1-yhat).clip(epsilon)) )

class mlp:
    '''Multilayer perceptron using numpy
    '''
    def __init__(self, layersizes, activations, derivatives, lossderiv):
        ”””remember config, then initialize array to hold NN parameters
        without init”””
        # hold NN config
        self.layersizes = tuple(layersizes)
        self.activations = tuple(activations)
        self.derivatives = tuple(derivatives)
        self.lossderiv = lossderiv
        assert len(self.layersizes)-1 == len(self.activations), \
            ”number of layers and the number of activation functions do not match”
        assert len(self.activations) == len(self.derivatives), \
            ”number of activation functions and number of derivatives do not match”
        assert all(isinstance(n, int) and n >= 1 for n in layersizes), \
            ”Only positive integral number of perceptons is allowed in each layer”
        # parameters, each is a 2D numpy array
        L = len(self.layersizes)
        self.z = [None] * L
        self.W = [None] * L
        self.b = [None] * L
        self.a = [None] * L
        self.dz = [None] * L
        self.dW = [None] * L
        self.db = [None] * L
        self.da = [None] * L

    def initialize(self, seed=42):
        ”””initialize the value of weight matrices and bias vectors with small
        random numbers.”””
        np.random.seed(seed)
        sigma = 0.1
        for l, (n_in, n_out) in enumerate(zip(self.layersizes, self.layersizes[1:]), 1):
            self.W[l] = np.random.randn(n_in, n_out) * sigma
            self.b[l] = np.random.randn(1, n_out) * sigma

    def forward(self, x):
        ”””Feed forward using existing `W` and `b`, and overwrite the result
        variables `a` and `z`
    Args:
      x (numpy.ndarray): Input data to feed forward
    ”””
    self.a[0] = x
    for l, func in enumerate(self.activations, 1):
      # z = W a + b, with `a` as output from previous layer
      # `W` is of size rxs and `a` the size sxn with n the number of data
      # instances, `z` the size rxn, `b` is rx1 and broadcast to each
      # column of `z`
      self.z[l] = (self.a[l-1] @ self.W[l]) + self.b[l]
      # a = g(z), with `a` as output of this layer, of size rxn
      self.a[l] = func(self.z[l])
    return self.a[-1]

  def backward(self, y, yhat):
    ”””back propagation using NN output yhat and the reference output y,
    generates dW, dz, db, da
    ”””
    assert y.shape[1] == self.layersizes[-1], \
      ”Output size doesn't match network output size”
    assert y.shape == yhat.shape, \
      ”Output size doesn't match reference”
    # first `da`, at the output
    self.da[-1] = self.lossderiv(y, yhat)
    for l, func in reversed(list(enumerate(self.derivatives, 1))):
      # compute the differentials at this layer
      self.dz[l] = self.da[l] * func(self.z[l])
      self.dW[l] = self.a[l-1].T @ self.dz[l]
      self.db[l] = np.mean(self.dz[l], axis=0, keepdims=True)
      self.da[l-1] = self.dz[l] @ self.W[l].T
      assert self.z[l].shape == self.dz[l].shape
      assert self.W[l].shape == self.dW[l].shape
      assert self.b[l].shape == self.db[l].shape
      assert self.a[l].shape == self.da[l].shape

  def update(self, eta):
    ”””Updates W and b

    Args:
      eta (float): Learning rate
    ”””
    for l in range(1, len(self.W)):
      self.W[l] -= eta * self.dW[l]
      self.b[l] -= eta * self.db[l]

# Make data: Two circles on x-y plane as a classification problem
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
y = y.reshape(-1,1) # our model expects a 2D array of (n_sample, n_dim)
print(X.shape)
print(y.shape)

# Build a model
model = mlp(layersizes=[2, 4, 3, 1],
      activations=[relu, relu, sigmoid],
            derivatives=[drelu, drelu, dsigmoid],
            lossderiv=d_cross_entropy)
model.initialize()
yhat = model.forward(X)
loss = cross_entropy(y, yhat)
score = accuracy_score(y, (yhat > 0.5))
print(f”Before training - loss value {loss} accuracy {score}”)

# train for each epoch
n_epochs = 150
learning_rate = 0.005
for n in range(n_epochs):
    model.forward(X)
    yhat = model.a[-1]
    model.backward(y, yhat)
    model.update(learning_rate)
    loss = cross_entropy(y, yhat)
    score = accuracy_score(y, (yhat > 0.5))
    print(f”Iteration {n} - loss value {loss} accuracy {score}”)
```

Program 31.8: Neural network for binary classification

31.6 Further readings

The backpropagation algorithm is the center of all neural network training, regardless of what variation of gradient descent algorithms you used. Textbook such as this one covers it:

▷ Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.
https://amzn.to/3qSk3C2

31.7 Summary

In this tutorial, you learned how differentiation is applied to training a neural network.

Specifically, you learned:

▷ What is a total differential and how it is expressed as a sum of partial differentials

▷ How to express a neural network as equations and derive the gradients by differentiation

▷ How backpropagation helped us to express the gradients of each layer in the neural network

▷ How to convert the gradients into code to make a neural network model

In the next chapter, we will see the use of calculus in a different machine learning model.

32 Training a Support Vector Machine: The Separable Case

This tutorial is designed for anyone looking for a deeper understanding of how Lagrange multipliers are used in building up the model for support vector machines (SVMs). SVMs were initially designed to solve binary classification problems and later extended and applied to regression and unsupervised learning. They have shown their success in solving many complex machine learning classification problems.

In this tutorial, we'll look at the simplest SVM that assumes that the positive and negative examples can be completely separated via a linear hyperplane.

After completing this tutorial, you will know:

- How the hyperplane acts as the decision boundary
- Mathematical constraints on the positive and negative examples
- What is the margin and how to maximize the margin
- Role of Lagrange multipliers in maximizing the margin
- How to determine the separating hyperplane for the separable case

Let's get started.

Overview

This tutorial is divided into three parts; they are:

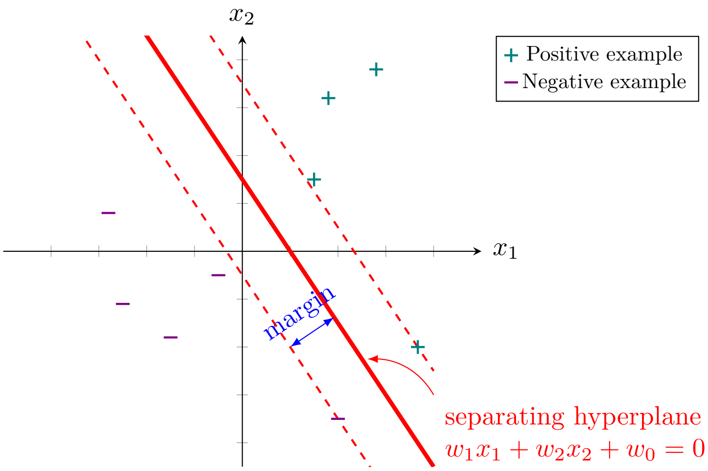
- Formulation of the mathematical model of SVM
- Solution of finding the maximum margin hyperplane via the method of Lagrange multipliers
- Solved example to demonstrate all concepts

32.1 Notations used in this tutorial

- m: Total training points.
- n: Total features or the dimensionality of all data points

- ▷ x: Data point, which is an n-dimensional vector.
- ▷ x^+: Data point labeled as $+1$.
- ▷ x^-: Data point labeled as -1.
- ▷ i: Subscript used to index the training points. $0 \leq i < m$
- ▷ j: Subscript used to index the individual dimension of a data point. $1 \leq j \leq n$
- ▷ t: Label of a data point.
- ▷ $\top$: Transpose operator.
- ▷ w: Weight vector denoting the coefficients of the hyperplane. It is also an n-dimensional vector.
- ▷ α: Lagrange multipliers, one per each training point. This is an m-dimensional vector.
- ▷ d: Perpendicular distance of a data point from the decision boundary.

32.2 The hyperplane as the decision boundary



The support vector machine is designed to discriminate data points belonging to two different classes. One set of points is labeled as $+1$ also called the positive class. The other set of points is labeled as -1 also called the negative class. For now, we'll make a simplifying assumption that points from both classes can be discriminated via linear hyperplane.

The SVM assumes a linear decision boundary between the two classes and the goal is to find a hyperplane that gives the maximum separation between the two classes. For this reason, the alternate term *maximum margin classifier* is also sometimes used to refer to an SVM. The perpendicular distance between the closest data point and the decision boundary is referred to as the *margin*. As the margin completely separates the positive and negative examples and does not tolerate any errors, it is also called the *hard margin*.

The mathematical expression for a hyperplane is given below with w_j being the coefficients and w_0 being the arbitrary constant that determines the distance of the hyperplane from the origin:

$$w^\top x_i + w_0 = 0$$

For the i-th 2-dimensional point (x_{i1}, x_{i2}) the above expression is reduced to:

$$w_1 x_{i1} + w_2 x_{i2} + w_0 = 0$$

Mathematical constraints on positive and negative data points

As we are looking to maximize the margin between positive and negative data points, we would like the positive data points to satisfy the following constraint:

$$w^\top x_i^+ + w_0 \geq +1$$

Similarly, the negative data points should satisfy:

$$w^\top x_i^- + w_0 \leq -1$$

We can use a neat trick to write a uniform equation for both set of points by using $t_i \in \{-1, +1\}$ to denote the class label of data point x_i:

$$t_i(w^\top x_i + w_0) \geq +1$$

32.3 The maximum margin hyperplane

The perpendicular distance d_i of a data point x_i from the margin is given by:

$$d_i = \frac{|w^\top x_i + w_0|}{\|w\|}$$

To maximize this distance, we can minimize the square of the denominator to give us a quadratic programming problem given by:

$$\begin{array}{lll} \text{minimize} & \frac{1}{2}\|w\|^2 & \\ \text{subject to} & t_i(w^\top x_i + w_0) \geq +1 & \forall i \end{array}$$

32.4 Solution via the method of Lagrange multipliers

To solve the above quadratic programming problem with inequality constraints, we can use the method of Lagrange multipliers. The Lagrange function is therefore:

$$L(w, w_0, \alpha) = \frac{1}{2}\|w\|^2 + \sum_i \alpha_i \Big(t_i(w^\top x_i + w_0) - 1\Big)$$

To solve the above, we set the following:

$$\frac{\partial L}{\partial w} = 0$$

$$\frac{\partial L}{\partial \alpha} = 0$$

$$\frac{\partial L}{\partial w_0} = 0$$

Plugging above in the Lagrange function gives us the following optimization problem, also called the dual:

$$L_d = -\frac{1}{2}\sum_i \sum_k \alpha_i \alpha_k t_i t_k (x_i)^\top (x_k) + \sum_i \alpha_i$$

We have to maximize the above subject to the following:

$$w = \sum_i \alpha_i t_i x_i$$

$$0 = \sum_i \alpha_i t_i$$

The nice thing about the above is that we have an expression for w in terms of Lagrange multipliers. The objective function involves no w term. There is a Lagrange multiplier associated with each data point. The computation of w_0 is also explained later.

32.5 Deciding the classification of a test point

The classification of any test point x can be determined using this expression:

$$y(x) = \sum_i \alpha_i t_i x^\top x_i + w_0$$

A positive value of $y(x)$ implies $x = +1$ and a negative value means $x = -1$

32.6 Karush-Kuhn-Tucker Conditions

Also, Karush-Kuhn-Tucker (KKT) conditions are satisfied by the above constrained optimization problem as given by:

$$\begin{aligned} \alpha_i &\geq 0 \\ t_i y(x_i) - 1 &\geq 0 \\ \alpha_i(t_i y(x_i) - 1) &= 0 \end{aligned}$$

The KKT conditions dictate that for each data point one of the following is true:

- ▷ The Lagrange multiplier is zero, i.e., $\alpha_i = 0$. This point, therefore, plays no role in classification; or,
- ▷ $t_i y(x_i) = 1$ and $\alpha_i > 0$: In this case, the data point has a role in deciding the value of w. Such a point is called a support vector.

For w_0, we can select any support vector x_s and solve

$$t_s y(x_s) = 1$$

giving us:

$$t_s(\sum_i \alpha_i t_i x_s^\top x_i + w_0) = 1$$

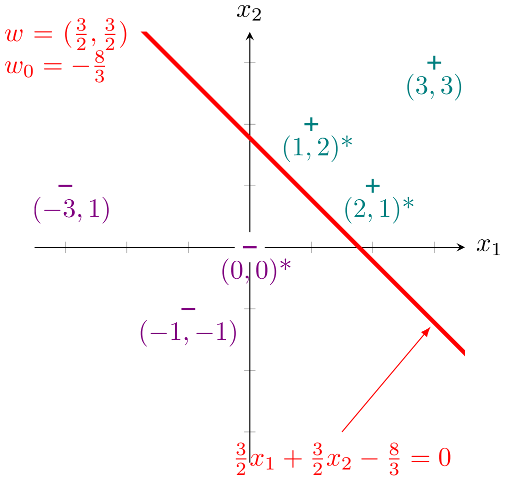
32.7 A solved example

To help you understand the above concepts, here is a simple arbitrarily solved example. Of course, for a large number of points you would use an optimization software to solve this. Also, this is one possible solution that satisfies all the constraints. The objective function can be maximized further but the slope of the hyperplane will remain the same for an optimal solution. Also, for this example, w_0 was computed by taking the average of w_0 from all three support vectors.

This example will show you that the model is not as complex as it looks.

i	data point x	label t	α
0	$(1,2)$	$+1$	0.5
1	$(2,1)$	$+1$	0.5
2	$(3,3)$	$+1$	0
3	$(0,0)$	-1	1
4	$(-1,-1)$	-1	0
5	$(-3,-1)$	-1	0

$\rightarrow$



For the above set of points, we can see that $(1,2)$, $(2,1)$ and $(0,0)$ are points closest to the separating hyperplane and hence, act as support vectors. Points far away from the boundary (e.g. $(-3,1)$) do not play any role in determining the classification of the points.

32.8 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

Joel Hass, Christopher Heil, and Maurice Weir. *Thomas' Calculus*. 14th ed. Based on the original works of George B. Thomas. Pearson, 2017.
https://www.amazon.com/dp/0134438981/

Articles

Christopher J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". *Data mining and Knowledge Discovery*, 2, 1998, pp. 121–167.
https://www.di.ens.fr/~mallat/papiers/svmtutorial.pdf

32.9 Summary

In this tutorial, you discovered how to use the method of Lagrange multipliers to solve the problem of maximizing the margin via a quadratic programming problem with inequality constraints.

Specifically, you learned:

- ▷ The mathematical expression for a separating linear hyperplane
- ▷ The maximum margin as a solution of a quadratic programming problem with inequality constraint
- ▷ How to find a linear hyperplane between positive and negative examples using the method of Lagrange multipliers

In the next chapter, we will consider the case that the SVM cannot separate the positive and negative classes perfectly.

Training a Support Vector Machine: The Non-Separable Case

33

This tutorial is an extension of the previous chapter and explains the non-separable case. In real life problems positive and negative training examples may not be completely separable by a linear decision boundary. This tutorial explains how a soft margin can be built that tolerates a certain amount of errors.

In this tutorial, we'll cover the basics of a linear SVM. We won't go into details of nonlinear SVMs derived using the kernel trick. The content is enough to understand the basic mathematical model behind an SVM classifier.

After completing this tutorial, you will know:

- Concept of a soft margin
- How to maximize the margin while allowing mistakes in classification
- How to formulate the optimization problem and compute the Lagrange dual

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- The solution of the SVM problem for the case where positive and negative examples are not linearly separable
  - The separating hyperplane and the corresponding relaxed constraints
  - The quadratic optimization problem for finding the soft margin
- A worked example

33.1 Notations used in this tutorial

This is a continuation of the previous chapter, so the same notations will be used.

- m: Total training points
- x: Data point, which is an n-dimensional vector. Each dimension is indexed by j.

- ▷ x^+: Positive example
- ▷ x^-: Negative example
- ▷ i: Subscript used to index the training points. $0 \le i < m$
- ▷ j: Subscript to index a dimension of the data point. $1 \le j \le n$
- ▷ t: Label of data points. It is an m-dimensional vector
- ▷ $\top$: Transpose operator
- ▷ w: Weight vector denoting the coefficients of the hyperplane. It is an n-dimensional vector
- ▷ α: Vector of Lagrange multipliers, an m-dimensional vector
- ▷ μ: Vector of Lagrange multipliers, again an m-dimensional vector
- ▷ ξ: Error in classification. An m-dimensional vector

33.2 The separating hyperplane and relaxing the constraints

Let's find a separating hyperplane between the positive and negative examples. Just to recall, the separating hyperplane is given by the following expression, with w_j being the coefficients and w_0 being the arbitrary constant that determines the distance of the hyperplane from the origin:

$$w^\top x_i + w_0 = 0$$

As we allow positive and negative examples to lie on the wrong side of the hyperplane, we have a set of relaxed constraints. Defining $\xi_i \ge 0, \forall i$, for positive examples we require:

$$w^\top x_i^+ + w_0 \ge 1 - \xi_i$$

Also for negative examples we require:

$$w^\top x_i^- + w_0 \le -1 + \xi_i$$

Combining the above two constraints by using the class label $t_i \in \{-1, +1\}$ we have the following constraint for all points:

$$t_i(w^\top x_i + w_0) \ge 1 - \xi_i$$

The variable ξ allows more flexibility in our model. It has the following interpretations:

- ▷ $\xi_i = 0$: This means that x_i is correctly classified and this data point is on the correct side of the hyperplane and away from the margin.
- ▷ $0 < \xi_i < 1$: When this condition is met, x_i lies on the correct side of the hyperplane but inside the margin.
- ▷ $\xi_i > 0$: Satisfying this condition implies that x_i is misclassified.

Hence, ξ quantifies the errors in the classification of training points. We can define the soft error as:

$$E_{\text{soft}} = \sum_i \xi_i$$

33.3 The quadratic programming problem

We are now in a position to formulate the objective function along with the constraints on it. We still want to maximize the margin, i.e., we want to minimize the norm of the weight vector. Along with this, we also want to keep the soft error as small as possible. Hence, now our new objective function is given by the following expression, with C being a user defined constant and represents the penalty factor or the regularization constant.

$$\frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

The overall quadratic programming problem is, therefore, given by the following expression:

$$\begin{aligned} &\min_w \frac{1}{2}\|w\|^2 + C\sum_i \xi_i \\ \text{subject to} \qquad & t_i(w^\top x_i + w_0) \geq +1 - \xi_i, \qquad && \forall i \\ & \xi_i \geq 0, && \forall i \end{aligned}$$

The role of regularization constant *C*

To understand the penalty factor C, consider the product term $C\sum_i \xi_i$, which has to be minimized. If C is kept large, then the soft margin $\sum_i \xi_i$ would automatically be small. If C is close to zero, then we are allowing the soft margin to be large making the overall product small.

In short, a large value of C means we have a high penalty on errors and hence our model is not allowed to make too many mistakes in classification. A small value of C allows the errors to grow.

33.4 Solution via the method of Lagrange multipliers

Let's use the method of Lagrange multipliers to solve the quadratic programming problem that we formulated earlier. The Lagrange function is given by:

$$L(w, w_0, \alpha, \mu) = \frac{1}{2}\|w\|^2 + \sum_i \alpha_i\Big(t_i(w^\top x_i + w_0) - 1 + \xi_i\Big) - \sum_i \mu_i \xi_i$$

To solve the above, we set the following:

$$\frac{\partial L}{\partial w} = 0$$

$$\frac{\partial L}{\partial \alpha} = 0$$

$$\frac{\partial L}{\partial w_0} = 0$$

$$\frac{\partial L}{\partial \mu} = 0$$

Solving the above gives us:

$$w = \sum_i \alpha_i t_i x_i$$

$$0 = C - \alpha_i - \mu_i$$

Substitute the above in the Lagrange function gives us the following optimization problem, also called the dual:

$$L_d = -\frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k t_i t_k x_i^\top x_k + \sum_i \alpha_i$$

We have to maximize the above subject to the following constraints:

$$\sum_i \alpha_i t_i = 0, \text{ and}$$

$$0 \leq \alpha_i \leq C, \qquad \forall i$$

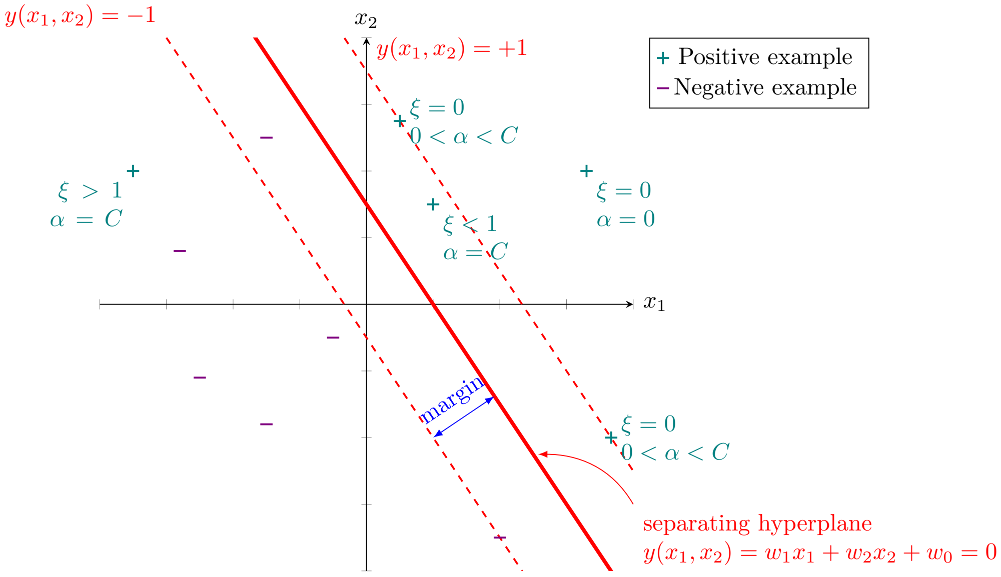
Similar to the separable case, we have an expression for w in terms of Lagrange multipliers. The objective function involves no w term. There is a Lagrange multiplier α and μ associated with each data point.

33.5 Interpretation of the mathematical model and computation of w_0

Following cases are true for each training data point x_i:

- ▷ $\alpha_i = 0$: The i-th training point lies on the correct side of the hyperplane away from the margin. This point plays no role in the classification of a test point.
- ▷ $0 < \alpha_i < C$: The i-th training point is a support vector and lies on the margin. For this point $\xi_i = 0$ and $t_i(w^\top x_i + w_0) = 1$ and hence it can be used to compute w_0. In practice w_0 is computed from all support vectors and an average is taken.
- ▷ $\alpha_i = C$: The i-th training point is either inside the margin on the correct side of the hyperplane or this point is on the wrong side of the hyperplane.

The picture below will help you understand the above concepts:



33.6 Deciding the classification of a test point

The classification of any test point x can be determined using this expression:

$$y(x) = \sum_i \alpha_i t_i x^\top x_i + w_0$$

A positive value of $y(x)$ implies $x = +1$ and a negative value means $x = -1$. Hence, the predicted class of a test point is the sign of $y(x)$.

33.7 Karush-Kuhn-Tucker conditions

Karush-Kuhn-Tucker (KKT) conditions are satisfied by the above constrained optimization problem as given by:

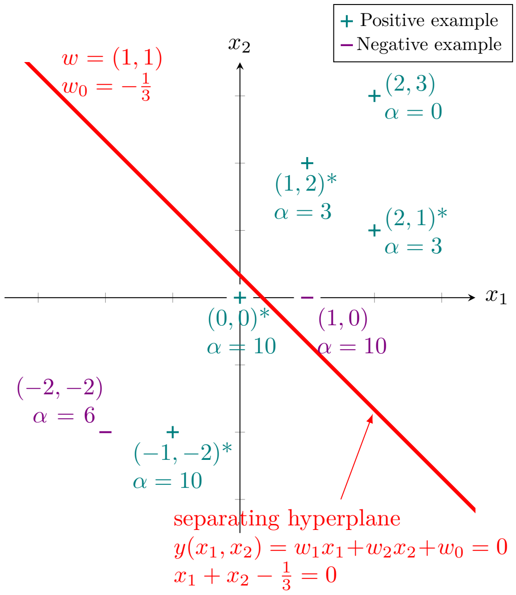
$$\begin{aligned} \alpha_i &\geq 0 \\ t_i y(x_i) - 1 + \xi_i &\geq 0 \\ \alpha_i (t_i y(x_i) - 1 + \xi_i) &= 0 \\ \mu_i &\geq 0 \\ \xi_i &\geq 0 \\ \mu_i \xi_i &= 0 \end{aligned}$$

33.8 A solved example

i	data point x	label t	α
0	$(1,2)$	$+1$	3
1	$(2,1)$	$+1$	3
2	$(0,0)$	$+1$	10
3	$(2,3)$	$+1$	0
4	$(-1,-2)$	$+1$	10
5	$(-2,-2)$	-1	6
6	$(1,0)$	-1	10

$$\sum_i \alpha_i t_i = 0$$

→



Points marked with * are support vectors

Compute w:

$$\begin{aligned} w &= 3 \times (1,2) + 3 \times (2,1) + 10 \times (0,0) + 10 \times (-1,-2) + 6 \times (-2,-1) + 10 \times (1,0) \\ &= (1,1) \end{aligned}$$

Compute w_0:

$$\begin{aligned} &\text{From } (1,2) & 1 + 2 + w_0 &= 1 \\ & & w_0 &= -2 \\ &\text{From } (2,1) & 2 + 1 + w_0 &= 1 \\ & & w_0 &= -2 \\ &\text{From } (-2,-2) & -2 - 2 + w_0 &= -1 \\ & & w_0 &= 3 \\ &\text{Take the average} & w_0 &= \frac{-2-2+3}{3} \\ & & &= -\frac{1}{3} \end{aligned}$$

Shown above is a solved example for 2D training points to illustrate all the concepts. A few things to note about this solution are:

- ▷ The training data points and their corresponding labels act as input
- ▷ The user defined constant C is set to 10

- The solution satisfies all the constraints, however, it is not the optimal solution
- We have to make sure that all the α lie between 0 and C
- The sum of alphas of all negative examples should equal the sum of alphas of all positive examples
- The points $(1, 2)$, $(2, 1)$ and $(-2, -2)$ lie on the soft margin on the correct side of the hyperplane. Their values have been arbitrarily set to 3, 3 and 6 respectively to balance the problem and satisfy the constraints.
- The points with $\alpha = C = 10$ lie either inside the margin or on the wrong side of the hyperplane

33.9 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

Articles

Christopher J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". *Data mining and Knowledge Discovery*, 2, 1998, pp. 121–167.
https://www.di.ens.fr/~mallat/papiers/svmtutorial.pdf

33.10 Summary

In this tutorial, you discovered the method of Lagrange multipliers for finding the soft margin in an SVM classifier.

Specifically, you learned:

- How to formulate the optimization problem for the non-separable case
- How to find the hyperplane and the soft margin using the method of Lagrange multipliers
- How to find the equation of the separating hyperplane for very simple problems

In the next chapter, we will see how we can implement what we learned above in Python.

34 Implementing a Support Vector Machine in Python

The mathematics that powers a support vector machine (SVM) classifier is beautiful. It is important to not only learn the basic model of an SVM but also know how you can implement the entire model from scratch. This is a continuation of our series of tutorials on SVMs. In the previous two chapters, we discussed the mathematical model behind a linear SVM. In this tutorial, we'll show how you can build an SVM linear classifier using the optimization routines shipped with Python's SciPy library.

After completing this tutorial, you will know:

- How to use SciPy's optimization routines
- How to define the objective function
- How to define bounds and linear constraints
- How to implement your own SVM classifier in Python

Let's get started.

Overview

This tutorial is divided into two parts; they are:

- The optimization problem of an SVM
- Solution of the optimization problem in Python
  - Define the objective function
  - Define the bounds and linear constraints
- Solve the problem with different C values

34.1 Notations and assumptions

A basic SVM machine assumes a binary classification problem. Suppose, we have m training points, each point being an n-dimensional vector. We'll use the following notations:

- m: Total training points

- n: Dimensionality of each training point
- x: Data point, which is an n-dimensional vector
- i: Subscript used to index the training points. $0 \le i < m$
- k: Subscript used to index the training points. $0 \le k < m$
- j: Subscript used to index each dimension of a training point
- t: Label of a data point. It is an m-dimensional vector, with $t_i \in \{-1, +1\}$
- $\top$: Transpose operator
- w: Weight vector denoting the coefficients of the hyperplane. It is also an n-dimensional vector
- α: Vector of Lagrange multipliers, also an m-dimensional vector
- C: User defined penalty factor/regularization constant

34.2 The SVM optimization problem

The SVM classifier maximizes the following Lagrange dual given by:

$$L_d = -\frac{1}{2}\sum_i \sum_k \alpha_i \alpha_k t_i t_k x_i^\top x_k + \sum_i \alpha_i$$

The above function is subject to the following constraints:

$$\begin{aligned} &0 \le \alpha_i \le C, && \forall i \\ &\sum_i \alpha_i t_i = 0 \end{aligned}$$

All we have to do is find the Lagrange multiplier α associated with each training point, while satisfying the above constraints.

34.3 Python implementation of SVM

We'll use the SciPy optimize package to find the optimal values of Lagrange multipliers, and compute the soft margin and the separating hyperplane.

Let's write the import section for optimization, plotting and synthetic data generation.

```
import numpy as np
# For optimization
from scipy.optimize import Bounds, BFGS
from scipy.optimize import LinearConstraint, minimize
# For plotting
import matplotlib.pyplot as plt
import seaborn as sns
# For generating dataset
import sklearn.datasets as dt
```

Program 34.1: Libraries to be used

We also need the following constant to detect all alphas numerically close to zero, so we need to define our own threshold for zero.

```
...
ZERO = 1e-7
```

Program 34.2: Define a small value deemed as zero

Next, let's define a very simple dataset, the corresponding labels and a simple routine for plotting this data. Optionally, if a string of alphas is given to the plotting function, then it will also label all support vectors with their corresponding alpha values. Just to recall support vectors are those points for which $\alpha > 0$.

```
dat = np.array([[0,3], [-1,0], [1,2], [2,1], [3,3], [0,0], [-1,-1], [-3,1], [3,1]])
labels = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1])

def plot_x(x, t, alpha=[], C=0):
    sns.scatterplot(x=dat[:,0], y=dat[:,1], style=labels,
                    hue=labels, markers=['s','P'], palette=['magenta','green'])
    if len(alpha) > 0:
        alpha_str = np.char.mod('%.1f', np.round(alpha, 1))
        ind_sv = np.where(alpha > ZERO)[0]
        for i in ind_sv:
            plt.gca().text(dat[i,0], dat[i, 1]-.25, alpha_str[i] )

plot_x(dat, labels)
plt.show()
```

Program 34.3: Define data points

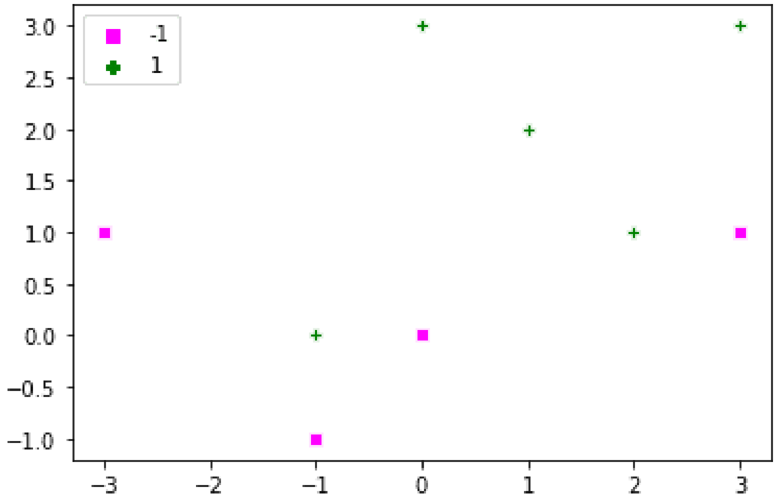


Figure 34.1: Data points generated by Program 34.3

34.4 The minimize() Function

Let's look at the `minimize()` function in `scipy.optimize` library. It requires the following arguments:

- The objective function to minimize. Lagrange dual in our case.
- The initial values of variables with respect to which the minimization takes place. In this problem, we have to determine the Lagrange multipliers α. We'll initialize all α randomly.
- The method to use for optimization. We'll use `trust-constr`.
- The linear constraints on α.
- The bounds on α.

Defining the objective function

Our objective function is L_d defined above, which has to be maximized. As we are using the `minimize()` function, we have to multiply L_d by (-1) to maximize it. Its implementation is given below. The first parameter for the objective function is the variable with respect to which the optimization takes place. We also need the training points and the corresponding labels as additional arguments.

You can shorten the code for the `lagrange_dual()` function given below by using matrices. However, in this tutorial, it is kept very simple to make everything clear.

```python
# Objective function
def lagrange_dual(alpha, x, t):
    result = 0
    ind_sv = np.where(alpha > ZERO)[0]
    for i in ind_sv:
        for k in ind_sv:
            result = result + alpha[i]*alpha[k]*t[i]*t[k]*np.dot(x[i, :], x[k, :])
    result = 0.5*result - sum(alpha)
    return result
```

Program 34.4: Lagrange dual function

Defining the linear constraints

The linear constraint on alpha for each point is given by:

$$\sum_i \alpha_i t_i = 0$$

We can also write this as:

$$\alpha_0 t_0 + \alpha_1 t_1 + \ldots \alpha_m t_m = 0$$

The `LinearConstraint()` class requires all constraints to be written as matrix form, which is:

$$0 = \begin{bmatrix} t_0 & t_1 & \dots t_m \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix} = 0$$

The first matrix is the first parameter in the `LinearConstraint()` class. The left and right bounds are the second and third arguments. This produce the `LinearConstraint` object that will be used later when we call `minimize()`.

```
...
linear_constraint = LinearConstraint(labels, [0], [0])
```

Program 34.5: Defining the `LinearConstraint` object

Defining the bounds

The bounds on alpha are defined using the `Bounds()` class. All alphas are constrained to lie between 0 and C. Here is an example for $C = 10$.

```
bounds_alpha = Bounds(np.zeros(dat.shape[0]), np.full(dat.shape[0], 10))
print(bounds_alpha)
```

Program 34.6: Defining the `Bounds` object

```
Bounds(array([0., 0., 0., 0., 0., 0., 0., 0., 0.]), array([10, 10, 10, 10, 10, 10, 10,  ⤸
↪   10, 10]))
```

Output 34.1: Output of Program 34.6

Defining the function to find α

Let's write the overall routine to find the optimal values of α when given the parameters `x`, `t`, and `C`. The objective function requires the additional arguments `x` and `t`, which are passed via arguments in `minimize()`.

```
def optimize_alpha(x, t, C):
    m, n = x.shape
    np.random.seed(1)
    # Initialize alphas to random values
    alpha_0 = np.random.rand(m)*C
    # Define the constraint
    linear_constraint = LinearConstraint(t, [0], [0])
    # Define the bounds
    bounds_alpha = Bounds(np.zeros(m), np.full(m, C))
    # Find the optimal value of alpha
    result = minimize(lagrange_dual, alpha_0, args = (x, t), method='trust-constr',
                      hess=BFGS(), constraints=[linear_constraint],
                      bounds=bounds_alpha)
    # The optimized value of alpha lies in result.x
    alpha = result.x
    return alpha
```

Program 34.7: Function to find the optimal α

Determining the hyperplane

The expression for the hyperplane is given by:

$$w^\top x + w_0 = 0$$

For the hyperplane, we need the weight vector w and the constant w_0. The weight vector is given by:

$$w = \sum_i \alpha_i t_i x_i$$

If there are too many training points, it's best to use only support vectors with $\alpha > 0$ to compute the weight vector.

For w_0, we'll compute it from each support vector s, for which $\alpha_s < C$, and then take the average. For a single support vector x_s, w_0 is given by:

$$w_0 = t_s - w^\top x_s$$

A support vector's α cannot be numerically exactly equal to C. Hence, we can subtract a small constant from C to find all support vectors with $\alpha_s < C$. This is done in the `get_w0()` function:

```
def get_w(alpha, t, x):
    m = len(x)
    # Get all support vectors
    w = np.zeros(x.shape[1])
    for i in range(m):
        w = w + alpha[i]*t[i]*x[i, :]
    return w

def get_w0(alpha, t, x, w, C):
    C_numeric = C-ZERO
    # Indices of support vectors with alpha<C
    ind_sv = np.where((alpha > ZERO)&(alpha < C_numeric))[0]
    w0 = 0.0
    for s in ind_sv:
        w0 = w0 + t[s] - np.dot(x[s, :], w)
    # Take the average
    w0 = w0 / len(ind_sv)
    return w0
```

Program 34.8: Functions to find w and w_0

Classifying test points

To classify a test point x_{test}, we use the sign of $y(x_{\text{test}})$ as:

$$\text{label}_{x_{\text{test}}} = \text{sign}(y(x_{\text{test}})) = \text{sign}(w^\top x_{\text{test}} + w_0)$$

Let's write the corresponding function that can take as argument an array of test points along with w and w_0 and classify various points. We have also added a second function for calculating the misclassification rate:

```python
def classify_points(x_test, w, w0):
    # get y(x_test)
    predicted_labels = np.sum(x_test*w, axis=1) + w0
    predicted_labels = np.sign(predicted_labels)
    # Assign a label arbitrarily a +1 if it is zero
    predicted_labels[predicted_labels==0] = 1
    return predicted_labels

def misclassification_rate(labels, predictions):
    total = len(labels)
    errors = sum(labels != predictions)
    return errors/total*100
```

Program 34.9: Functions to classify test points

Plotting the margin and hyperplane

Let's also define functions to plot the hyperplane and the soft margin.

```python
def plot_hyperplane(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    y_coord = -w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, y_coord, color='red')

def plot_margin(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    ypos_coord = 1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, ypos_coord, '--', color='green')
    yneg_coord = -1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, yneg_coord, '--', color='magenta')
```

Program 34.10: Functions for plotting

34.5 Powering up the SVM

It's now time to run the SVM. The function `display_SVM_result()` will help us visualize everything. We'll initialize α to random values, define C and find the best values of α in this function. We'll also plot the hyperplane, the margin and the data points. The support vectors would also be labeled by their corresponding α value. The title of the plot would be the percentage of errors and number of support vectors.

```
def display_SVM_result(x, t, C):
    # Get the alphas
    alpha = optimize_alpha(x, t, C)
    # Get the weights
    w = get_w(alpha, t, x)
    w0 = get_w0(alpha, t, x, w, C)
    plot_x(x, t, alpha, C)
    xlim = plt.gca().get_xlim()
    ylim = plt.gca().get_ylim()
    plot_hyperplane(w, w0)
    plot_margin(w, w0)
    plt.xlim(xlim)
    plt.ylim(ylim)
    # Get the misclassification error and display it as title
    predictions = classify_points(x, w, w0)
    err = misclassification_rate(t, predictions)
    title = 'C = ' + str(C) + ',  Errors: ' + '{:.1f}'.format(err) + '%'
    title = title + ',  total SV = ' + str(len(alpha[alpha > ZERO]))
    plt.title(title)

display_SVM_result(dat, labels, 100)
plt.show()
```

Program 34.11: Function to display the SVM

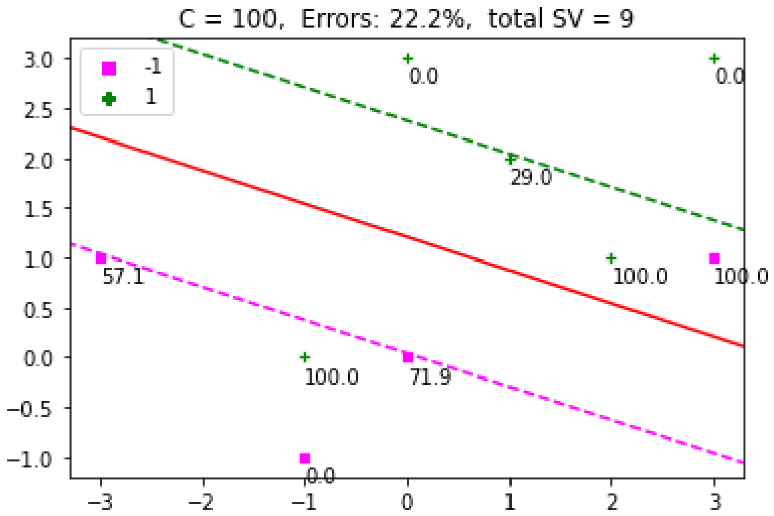


Figure 34.2: Plot produced by Program 34.11

Putting them together, the following is the complete code to produce a SVM from a given dataset:

```
import numpy as np
# For optimization
from scipy.optimize import Bounds, BFGS
from scipy.optimize import LinearConstraint, minimize
# For plotting
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
# For generating dataset
import sklearn.datasets as dt

ZERO = 1e-7

def plot_x(x, t, alpha=[], C=0):
    sns.scatterplot(x=dat[:,0], y=dat[:,1], style=labels,
                    hue=labels, markers=['s','P'], palette=['magenta','green'])
    if len(alpha) > 0:
        alpha_str = np.char.mod('%.1f', np.round(alpha, 1))
        ind_sv = np.where(alpha > ZERO)[0]
        for i in ind_sv:
            plt.gca().text(dat[i,0], dat[i, 1]-.25, alpha_str[i] )

# Objective function
def lagrange_dual(alpha, x, t):
    result = 0
    ind_sv = np.where(alpha > ZERO)[0]
    for i in ind_sv:
        for k in ind_sv:
            result = result + alpha[i]*alpha[k]*t[i]*t[k]*np.dot(x[i, :], x[k, :])
    result = 0.5*result - sum(alpha)
    return result

def optimize_alpha(x, t, C):
    m, n = x.shape
    np.random.seed(1)
    # Initialize alphas to random values
    alpha_0 = np.random.rand(m)*C
    # Define the constraint
    linear_constraint = LinearConstraint(t, [0], [0])
    # Define the bounds
    bounds_alpha = Bounds(np.zeros(m), np.full(m, C))
    # Find the optimal value of alpha
    result = minimize(lagrange_dual, alpha_0, args = (x, t), method='trust-constr',
                      hess=BFGS(), constraints=[linear_constraint],
                      bounds=bounds_alpha)
    # The optimized value of alpha lies in result.x
    alpha = result.x
    return alpha

def get_w(alpha, t, x):
    m = len(x)
    # Get all support vectors
    w = np.zeros(x.shape[1])
    for i in range(m):
        w = w + alpha[i]*t[i]*x[i, :]
    return w

def get_w0(alpha, t, x, w, C):
    C_numeric = C-ZERO
    # Indices of support vectors with alpha<C
    ind_sv = np.where((alpha > ZERO)&(alpha < C_numeric))[0]
    w0 = 0.0
    for s in ind_sv:
        w0 = w0 + t[s] - np.dot(x[s, :], w)
    # Take the average
    w0 = w0 / len(ind_sv)
    return w0

def classify_points(x_test, w, w0):
    # get y(x_test)
    predicted_labels = np.sum(x_test*w, axis=1) + w0
    predicted_labels = np.sign(predicted_labels)
    # Assign a label arbitrarily a +1 if it is zero
    predicted_labels[predicted_labels==0] = 1
    return predicted_labels

def misclassification_rate(labels, predictions):
    total = len(labels)
    errors = sum(labels != predictions)
    return errors/total*100

def plot_hyperplane(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    y_coord = -w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, y_coord, color='red')

def plot_margin(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    ypos_coord = 1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, ypos_coord, '-', color='green')
    yneg_coord = -1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, yneg_coord, '-', color='magenta')

def display_SVM_result(x, t, C):
    # Get the alphas
    alpha = optimize_alpha(x, t, C)
    # Get the weights
    w = get_w(alpha, t, x)
    w0 = get_w0(alpha, t, x, w, C)
    plot_x(x, t, alpha, C)
    xlim = plt.gca().get_xlim()
    ylim = plt.gca().get_ylim()
    plot_hyperplane(w, w0)
    plot_margin(w, w0)
    plt.xlim(xlim)
    plt.ylim(ylim)
    # Get the misclassification error and display it as title
    predictions = classify_points(x, w, w0)
    err = misclassification_rate(t, predictions)
    title = 'C = ' + str(C) + ',  Errors: ' + '{:.1f}'.format(err) + '%'
    title = title + ',  total SV = ' + str(len(alpha[alpha > ZERO]))
    plt.title(title)
```

```
dat = np.array([[0, 3], [-1, 0], [1, 2], [2, 1], [3,3], [0, 0], [-1, -1], [-3, 1], [3, 1]])
labels = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1])
plot_x(dat, labels)
plt.show()
display_SVM_result(dat, labels, 100)
plt.show()
```

Program 34.12: Fit a SVM on a given dataset

34.6 The effect of C

If you change the value of C to ∞, then the soft margin turns into a hard margin, with no toleration for errors. The problem we defined above is not solvable in this case. Let's generate an artificial set of points and look at the effect of C on classification. To understand the entire problem, we'll use a simple dataset, where the positive and negative examples are separable.

Below are the points generated via `make_blobs()`:

```
dat, labels = dt.make_blobs(n_samples=[20,20],
                            cluster_std=1,
                            random_state=0)
labels[labels==0] = -1
plot_x(dat, labels)
```

Program 34.13: Generate points and labels

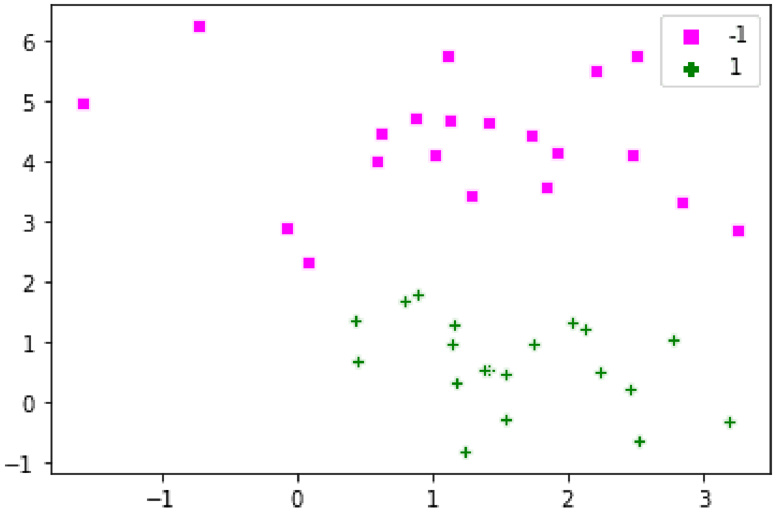


Figure 34.3: Data points generated by Program 34.13

Now let's define different values of C and run the code.

```
fig = plt.figure(figsize=(8,25))

i=0
C_array = [1e-2, 100, 1e5]

for C in C_array:
    fig.add_subplot(311+i)
    display_SVM_result(dat, labels, C)
    i = i + 1
```

Program 34.14: SVM with different values of C

The above is a nice example, which shows that increasing C, decreases the margin. A high value of C adds a stricter penalty on errors. A smaller value allows a wider margin and more misclassification errors. Hence, C defines a trade-off between the maximization of margin and classification errors.

34.7 Consolidated code

Here is the consolidated code, that you can save it in a file and run it at your end. This will take a while to run. You can experiment with different values of C and try out the different optimization methods given as arguments to the `minimize()` function. At the end of this program, you will see the plot as in Figure 34.2 but for the data points in Figure 34.3, one each for the different values of C.

```
import numpy as np
# For optimization
from scipy.optimize import Bounds, BFGS
from scipy.optimize import LinearConstraint, minimize
# For plotting
import matplotlib.pyplot as plt
import seaborn as sns
# For generating dataset
import sklearn.datasets as dt

ZERO = 1e-7

def plot_x(x, t, alpha=[], C=0):
    sns.scatterplot(x=dat[:,0], y=dat[:,1], style=labels,
                    hue=labels, markers=['s','P'], palette=['magenta','green'])
    if len(alpha) > 0:
        alpha_str = np.char.mod('%.1f', np.round(alpha, 1))
        ind_sv = np.where(alpha > ZERO)[0]
        for i in ind_sv:
            plt.gca().text(dat[i,0], dat[i, 1]-.25, alpha_str[i] )

# Objective function
def lagrange_dual(alpha, x, t):
    result = 0
    ind_sv = np.where(alpha > ZERO)[0]
    for i in ind_sv:
        for k in ind_sv:
            result = result + alpha[i]*alpha[k]*t[i]*t[k]*np.dot(x[i, :], x[k, :])
    result = 0.5*result - sum(alpha)
    return result

def optimize_alpha(x, t, C):
    m, n = x.shape
    np.random.seed(1)
    # Initialize alphas to random values
    alpha_0 = np.random.rand(m)*C
    # Define the constraint
    linear_constraint = LinearConstraint(t, [0], [0])
    # Define the bounds
    bounds_alpha = Bounds(np.zeros(m), np.full(m, C))
    # Find the optimal value of alpha
    result = minimize(lagrange_dual, alpha_0, args = (x, t), method='trust-constr',
                      hess=BFGS(), constraints=[linear_constraint],
                      bounds=bounds_alpha)
    # The optimized value of alpha lies in result.x
    alpha = result.x
    return alpha

def get_w(alpha, t, x):
    m = len(x)
    # Get all support vectors
    w = np.zeros(x.shape[1])
    for i in range(m):
        w = w + alpha[i]*t[i]*x[i, :]
    return w

def get_w0(alpha, t, x, w, C):
    C_numeric = C-ZERO
    # Indices of support vectors with alpha<C
    ind_sv = np.where((alpha > ZERO)&(alpha < C_numeric))[0]
    w0 = 0.0
    for s in ind_sv:
        w0 = w0 + t[s] - np.dot(x[s, :], w)
    # Take the average
    w0 = w0 / len(ind_sv)
    return w0

def classify_points(x_test, w, w0):
    # get y(x_test)
    predicted_labels = np.sum(x_test*w, axis=1) + w0
    predicted_labels = np.sign(predicted_labels)
    # Assign a label arbitrarily a +1 if it is zero
    predicted_labels[predicted_labels==0] = 1
    return predicted_labels

def misclassification_rate(labels, predictions):
    total = len(labels)
    errors = sum(labels != predictions)
    return errors/total*100
```

```
def plot_hyperplane(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    y_coord = -w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, y_coord, color='red')

def plot_margin(w, w0):
    x_coord = np.array(plt.gca().get_xlim())
    ypos_coord = 1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, ypos_coord, '-', color='green')
    yneg_coord = -1/w[1] - w0/w[1] - w[0]/w[1] * x_coord
    plt.plot(x_coord, yneg_coord, '-', color='magenta')

def display_SVM_result(x, t, C):
    # Get the alphas
    alpha = optimize_alpha(x, t, C)
    # Get the weights
    w = get_w(alpha, t, x)
    w0 = get_w0(alpha, t, x, w, C)
    plot_x(x, t, alpha, C)
    xlim = plt.gca().get_xlim()
    ylim = plt.gca().get_ylim()
    plot_hyperplane(w, w0)
    plot_margin(w, w0)
    plt.xlim(xlim)
    plt.ylim(ylim)
    # Get the misclassification error and display it as title
    predictions = classify_points(x, w, w0)
    err = misclassification_rate(t, predictions)
    title = 'C = ' + str(C) + ',  Errors: ' + '{:.1f}'.format(err) + '%'
    title = title + ',  total SV = ' + str(len(alpha[alpha > ZERO]))
    plt.title(title)

dat, labels = dt.make_blobs(n_samples=[20,20],
                            cluster_std=1,
                            random_state=0)
labels[labels==0] = -1
plot_x(dat, labels)

fig = plt.figure(figsize=(15,8))

i=0
C_array = [1e-2, 100, 1e5]

for C in C_array:
    fig.add_subplot(221+i)
    display_SVM_result(dat, labels, C)
    i = i + 1
plt.show()
```

Program 34.15: Building a SVM classifier with different values of C

34.8 Further reading

This section provides more resources on the topic if you are looking to go deeper.

Books

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
https://amzn.to/36yvG9w

Articles

Christopher J. C. Burges. “A Tutorial on Support Vector Machines for Pattern Recognition”. *Data mining and Knowledge Discovery*, 2, 1998, pp. 121–167.
https://www.di.ens.fr/~mallat/papiers/svmtutorial.pdf

APIs

SciPy’s optimization library.
https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
scikit-learn’s sample generation library (`sklearn.datasets`).
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets
NumPy random number generator.
https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html

34.9 Summary

In this tutorial, you discovered how to implement an SVM classifier from scratch.

Specifically, you learned:

- How to write the objective function and constraints for the SVM optimization problem
- How to write code to determine the hyperplane from Lagrange multipliers
- The effect of C on determining the margin

This marks the end of this book. Hope you now find yourself capable to understand the literature on machine learning algorithms when it mentions about calculus.

VII Appendix

Notations in Mathematics

A

If you are not from a suitable background, you may feel the notations in mathematics are confusing. Believe it or not, mathematics are fond of the rigorous formulation and logic but the notations used are sometimes ambiguous.

In the following, we list out all the notations you can find in the previous chapters and explain them. We hope this will make you feel easier to follow.

Delta. The Greek letter δ is usually to mean a change of something else. Therefore we usually see notations such as $\delta x = x_{n+1} - x_n$. Sometimes uppercase delta is used, for example Δx.

Multiplication. Multiplication so common that we preferred to omit this operator when there is no confusion. For example, in the equation $y = mx + c$, we write m and x together to mean m multiplied by x. Sometimes we would like to make the multiplication explicit, so we may write $m \times x$ or $m \cdot x$, or even $(m)(x)$.

Vectors. If we want to emphasize that a symbol is a vector, we may write it in bold or with an arrow, such as $\mathbf{x}$ or $\vec{x}$. But we may just write it as x if there is no confusion or ambiguity. The vectors in mathematics are analogous to one-dimensional array in programming. Hence we may sometimes write $\mathbf{w} = \langle w_0, w_1, w_2 \rangle$ but we may also use round or square brackets, such as (w_0, w_1, w_2) or $[w_0, w_1, w_2]$. Under the geometrical context, we may have unit vectors defined and all other vectors can be written using unit vectors. For example, we may see a vector in two-dimensional space as $x\mathbf{i} + y\mathbf{j}$ with $\mathbf{i}$ and $\mathbf{j}$ are unit vectors along the horizontal and vertical axes.

Norm of vectors. For a vector $\mathbf{v} = \langle x, y, z \rangle$, quite often we want to know how "long" it is, which is defined as $\sqrt{x^2 + y^2 + z^2}$. This operation is so common that we have a notation $\|\mathbf{v}\|_2$ to mean taking the square of each element of this vector, then sum it up, and take the square root of the sum. Hence $\|\mathbf{v}\|_2^2$ essentially is sum of the square of each element (without taking square root afterwards). Sometimes we write $\|\mathbf{v}\|$ instead of $\|\mathbf{v}\|_2$ for a cleaner notation. And if we write $\|\mathbf{v}\|_k$, it means to sum the k-th power of each element and take the k-th root. Hence $\|\mathbf{v}\|_1$ is just the sum of all elements, and we abuse this notation to make $\|\mathbf{v}\|_0$ to mean how many elements are there in vector $\mathbf{v}$.

Matrices. A matrix is usually represented with a uppercase letter, and sometimes we write it in bold, such as $\mathbf{W}$. If we write its elements out, we usually use square or round backets, such as

$$\begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} \quad \text{or} \quad \begin{pmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{pmatrix}.$$

Determinant of a matrix, however, is always written with a straight line on left and right, such as

$$\begin{vmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{vmatrix}.$$

Multiplication of vectors or matrix. If necessary, we will use a dot to mean the vector multiplication, such as $\mathbf{w}\cdot\mathbf{x}$. Because of the nature of matrix multiplication, we may sometimes consider vectors as column matrices and write $\mathbf{w}^\top\mathbf{x}$ instead. We are careful to write $\mathbf{w}\times\mathbf{x}$ for vectors as it means a different kind of multiplication (the cross product instead of dot product). On the contrary, we may see $\mathbf{A}\cdot\mathbf{B}$, $\mathbf{A}\times\mathbf{B}$, or even $\mathbf{AB}$ to mean the same multiplication for matrices. One special kind of multiplication for matrices is called the Hadamard product, which is to multiply elementwise and denoted as $\mathbf{A}\odot\mathbf{B}$ or $\mathbf{A}\otimes\mathbf{B}$.

Sets. We usually see symbols $\mathbb{R}$ to mean all real numbers. Similarly, we may use $\mathbb{Z}$ to mean all integers (positive and negative) and $\mathbb{N}$ to mean all natural numbers (positive integers only). When we want to represent a vector of n real numbers, we use $\mathbb{R}^n$ for that. When there is a set X of multiple elements, we can say x is one of them by $x \in X$. If we mean *for any* x in the set, we can write $\forall x \in X$ or simply $\forall x$.

Functions. We usually write functions as $f(x)$ to mean it takes value x, and hence $f(g(x))$ is a composite function that g takes the value x and f takes the result of g. But to define a function accurately, we may write $f : \mathbb{R}^k \mapsto \mathbb{R}^n$ to mean function f takes a vector of k real numbers as its input and produces a vector of n real numbers.

Summation, products, and factorials. We write $\displaystyle\sum_{k=0}^{3} f(k)$ to mean $f(0)+f(1)+f(2)+f(3)$ and $\displaystyle\prod_{j=2}^{4} x_j$ to mean $x_2 \times x_3 \times x_4$. This is the math expression to resemble a loop in programming. Factorial is a particular product that is used a lot and hence we use the notation 4! to mean $\prod k = 1^4 k = 1 \times 2 \times 3 \times 4$.

Differentiation. For a function $g(x)$, its derivative can be written as any of the following:

$$g'(x) \qquad \frac{dg}{dx} \qquad \frac{d}{dx}g(x) \qquad \dot{g}$$

Newton used $\dot{g}$ in his work but Leibniz used $\dfrac{dg}{dx}$. For a higher order derivative, we may use $g''(x), g'''(x)$ for second and third order derivatives, and subsequently, $g^{(n)}(x)$ for n-th order.

Leibniz's notation is more convenient, as we will write $\frac{d^n g}{dx^n}$ or $\frac{d^n}{dx^n}g(x)$ for n-th order. Newton's notation, however, will use $\ddot{g}$, $\dddot{g}$, $\ddddot{g}$ for second, third, and forth order derivatives respectively. For partial derivatives, we will use $\frac{\partial}{\partial x}g(x,y)$ and for higher order, we will use $\frac{\partial^n}{\partial x^n}g(x,y)$ or $\frac{\partial^n}{\partial x^m \partial y^{n-m}}g(x,y)$. But sometimes we will use a more convenient notation of g_x, or g_{xx}, g_{xy} for second order.

How to Setup a Workstation for Python

B

It can be difficult to install a Python machine learning environment on some platforms. Python itself must be installed first and then there are many packages to install, and it can be confusing for beginners. In this tutorial, you will discover how to setup a Python machine learning development environment using Anaconda. After completing this tutorial, you will have a working Python environment to begin learning, practicing, and developing machine learning software. These instructions are suitable for Windows, Mac OS X, and Linux platforms. I will demonstrate them on Windows, so you may see some Windows dialogs and file extensions.

B.1 Overview

In this tutorial, we will cover the following steps:

1. Download Anaconda
2. Install Anaconda
3. Start and Update Anaconda

> ℹ Note: The specific versions may differ as the software and libraries are updated frequently.

B.2 Download Anaconda

In this step, we will download the Anaconda Python package for your platform. Anaconda is a free and easy-to-use environment for scientific Python.

1. Visit the Anaconda homepage https://www.anaconda.com/

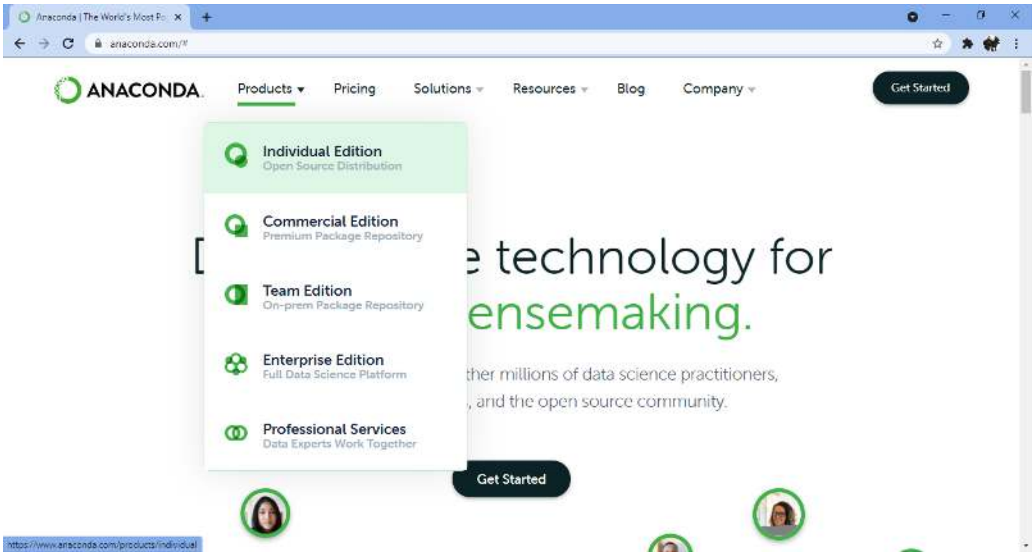
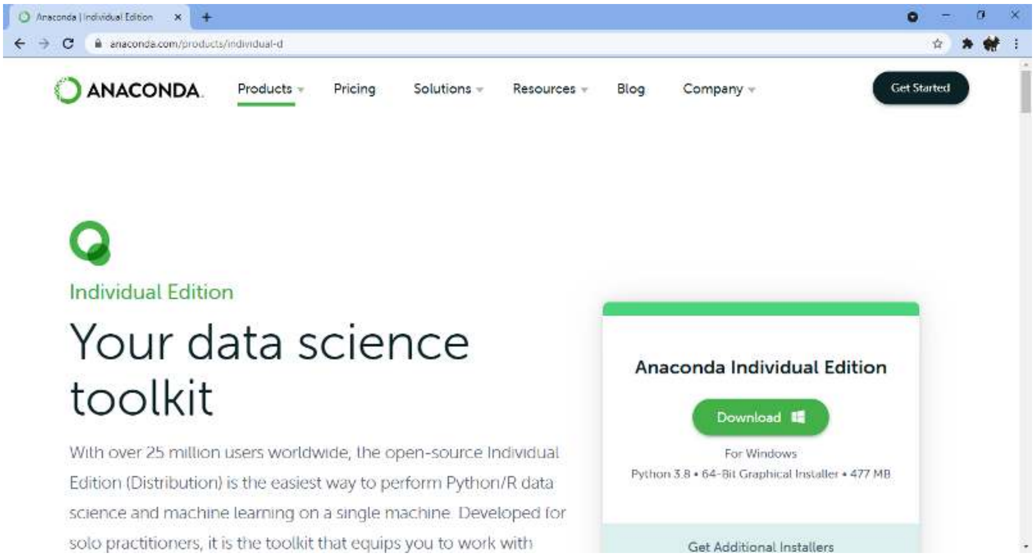

Anaconda | The World's Most Po
anaconda.com/#
ANACONDA
Products
Pricing
Solutions
Resources
Blog
Company
Get Started
Individual Edition
Open Source Distribution
Commercial Edition
Premium Package Repository
Team Edition
On-prem Package Repository
Enterprise Edition
Full Data Science Platform
Professional Services
Data Experts Work Together
[
e technology for
ensemaking.
ther millions of data science practitioners,
, and the open source community.
Get Started
https://www.anaconda.com/products/individual

Figure B.1: Click “Products” and “Individual Edition”

2. Click “Products” from the menu and click “Individual Edition” to go to the download page `https://www.anaconda.com/products/individual-d/`.


Anaconda | Individual Edition
anaconda.com/products/individual-d
ANACONDA
Products
Pricing
Solutions
Resources
Blog
Company
Get Started
Individual Edition
Your data science toolkit
Anaconda Individual Edition
Download
For Windows
Python 3.8 • 64-Bit Graphical Installer • 477 MB
Get Additional Installers

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with

Figure B.2: Click Download

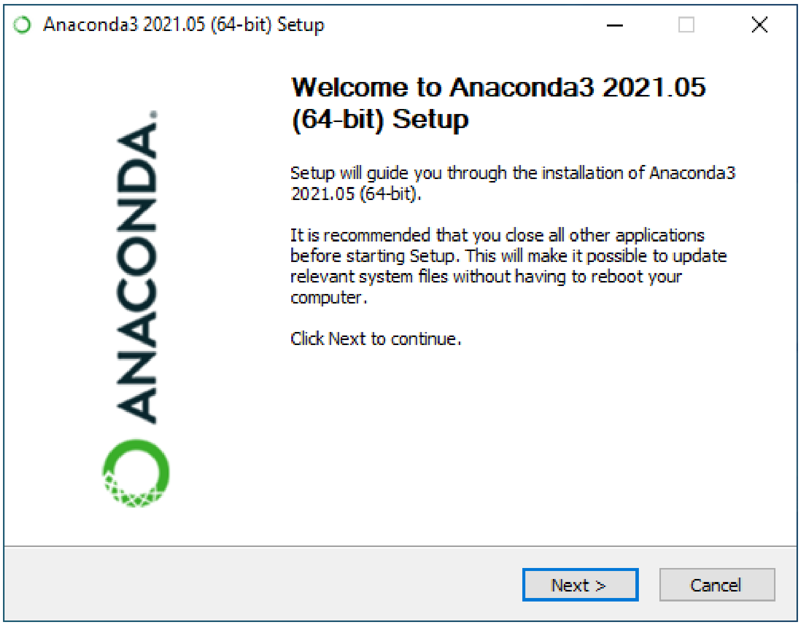
This will download the Anaconda Python package to your workstation. It will automatically give you the installer according to your OS (Windows, Linux, or MacOS). The file is about 480 MB. You should have a file with a name like:

```
Anaconda3-2021.05-Windows-x86_64.exe
```

B.3 Install Anaconda

In this step, we will install the Anaconda Python software on your system. This step assumes you have sufficient administrative privileges to install software on your system.

1. Double click the downloaded file.
2. Follow the installation wizard.



Anaconda3 2021.05 (64-bit) Setup

Welcome to Anaconda3 2021.05 (64-bit) Setup

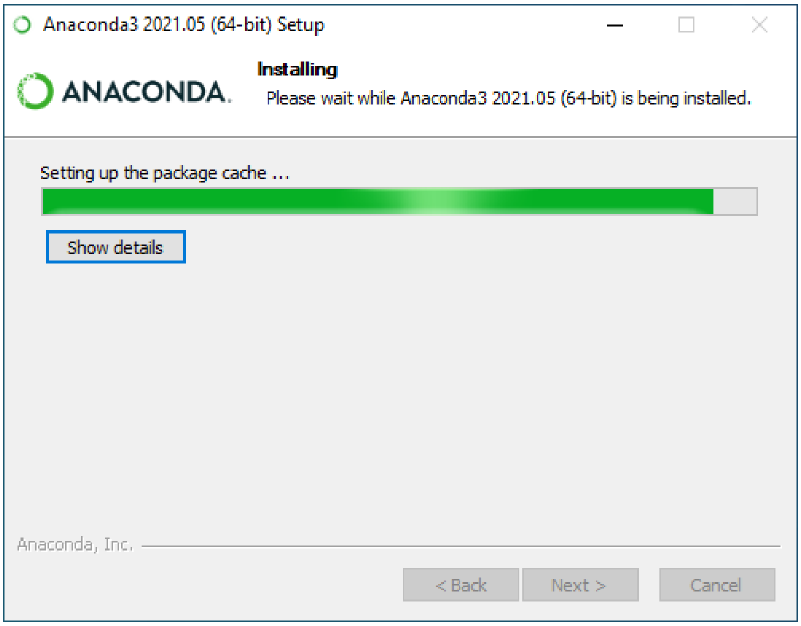
Setup will guide you through the installation of Anaconda3 2021.05 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure B.3: Anaconda Python Installation Wizard

Installation is quick and painless. There should be no tricky questions or sticking points.



Anaconda3 2021.05 (64-bit) Setup

Installing

Please wait while Anaconda3 2021.05 (64-bit) is being installed.

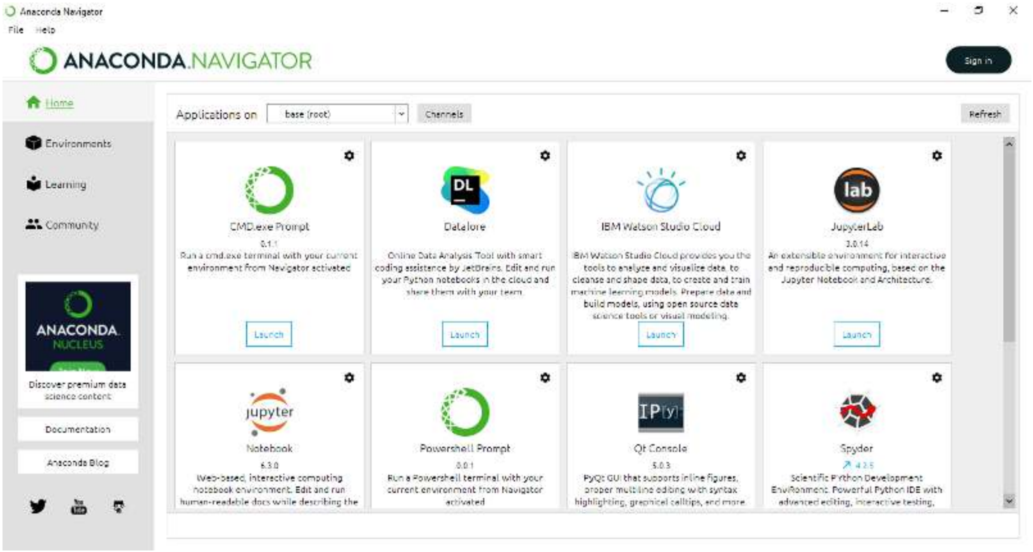
Setting up the package cache ...

Figure B.4: Anaconda Python Installation Wizard Writing Files

The installation should take less than 10 minutes and take a bit more than 5 GB of space on your hard drive.

B.4 Start and update Anaconda

In this step, we will confirm that your Anaconda Python environment is up to date. Anaconda comes with a suite of graphical tools called Anaconda Navigator. You can start Anaconda Navigator by opening it from your application launcher.



Run a cmd.exe terminal with your current environment from Navigator activated

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the

Run a Powershell terminal with your current environment from Navigator activated

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing,

Figure B.5: Anaconda Navigator GUI

You can use the Anaconda Navigator and graphical development environments later; for now, I recommend starting with the Anaconda command line environment called conda[1]. Conda is fast, simple, it's hard for error messages to hide, and you can quickly confirm your environment is installed and working correctly.

1. Open a terminal or CMD.exe prompt (command line window).
2. Confirm conda is installed correctly, by typing:

   ```
   conda -V
   ```

   You should see the following (or something similar):

   ```
   conda 4.10.1
   ```

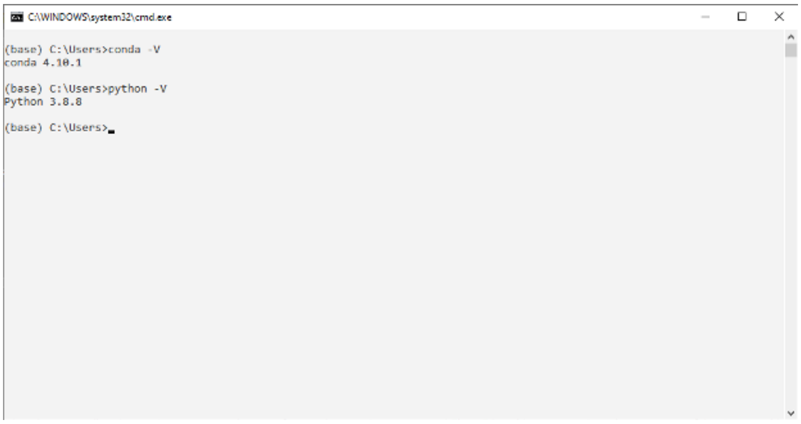
3. Confirm Python is installed correctly by typing:

   ```
   python -V
   ```

[1] https://conda.pydata.org/docs/index.html

You should see the following (or something similar):

```
Python 3.8.8
```



```
(base) C:\Users>conda -V
conda 4.10.1

(base) C:\Users>python -V
Python 3.8.8

(base) C:\Users>
```

Figure B.6: Confirm Conda and Python are Installed

If the commands do not work or have an error, please check the documentation for help for your platform. See some of the resources in the "Further Reading" section.

4. Confirm your conda environment is up-to-date, type:

```
conda update conda
conda update anaconda
```

You may need to install some packages and confirm the updates.

5. Confirm your SciPy environment.

The script below will print the version number of the key SciPy libraries you require for machine learning development, specifically: SciPy, NumPy, Matplotlib, Pandas, Statsmodels, and Scikit-learn. You can type "`python`" and type the commands in directly. Alternatively, I recommend opening a text editor and copy-pasting the script into your editor.

```python
# scipy
import scipy
print('scipy: %s' % scipy.__version__)
# numpy
import numpy
print('numpy: %s' % numpy.__version__)
# matplotlib
import matplotlib
print('matplotlib: %s' % matplotlib.__version__)
# pandas
import pandas
print('pandas: %s' % pandas.__version__)
# statsmodels
```

```
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)
# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)
```

Program B.1: Code to check that key Python libraries are installed

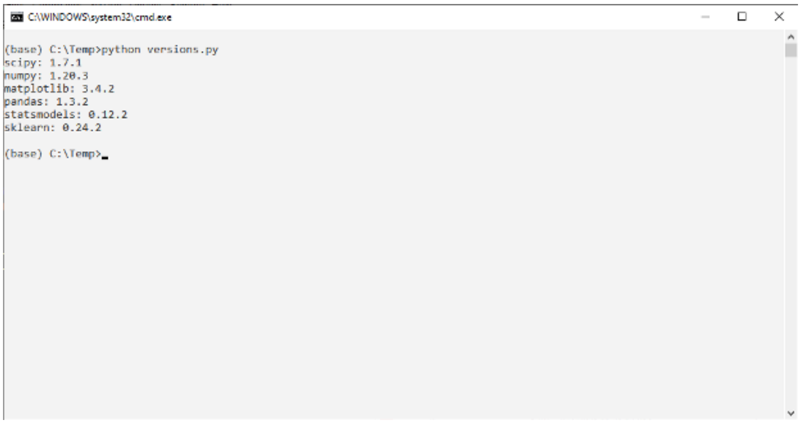
Save the script as a file with the name: `versions.py`. On the command line, change your directory to where you saved the script and type:

```
python versions.py
```

You should see output like the following:

```
scipy: 1.7.1
numpy: 1.20.3
matplotlib: 3.4.2
pandas: 1.3.2
statsmodels: 0.12.2
sklearn: 0.24.2
```

Output B.1: Sample output of thhe versions script



```
(base) C:\Temp>python versions.py
scipy: 1.7.1
numpy: 1.20.3
matplotlib: 3.4.2
pandas: 1.3.2
statsmodels: 0.12.2
sklearn: 0.24.2

(base) C:\Temp>
```

Figure B.7: Confirm Anaconda SciPy environment

B.5 Further reading

This section provides resources if you want to know more about Anaconda.

- Anaconda Documentation
  https://docs.continuum.io/
- Anaconda Documentation: Installation
  https://docs.continuum.io/anaconda/install
- Anaconda Navigator
  https://docs.continuum.io/anaconda/navigator.html

- The conda command line tool
  https://conda.pydata.org/docs/index.html
- Using conda
  https://conda.pydata.org/docs/using/

B.6 Summary

Congratulations, you now have a working Python development environment for machine learning. You can now learn and practice machine learning on your workstation.

How to Solve Calculus Problems C

Calculus is a topic of mathematics. For problems in calculus, we have various ways to check if your solution is correct. If you want to use a computer to verify your solution, or use a computer to solve a calculus problem for you, there are several approaches you can try.

C.1 Computer Algebra System

As you have seen in the earlier chapters of this book, there are several rules for differentiation and integration is just the reverse of the rules. Therefore, it is possible to analyze an expression and apply the rules to do differentiation and integration automatically. This is the idea of how a computer algebra system (CAS) work on a calculus problem.

Several famous CAS out there. The following are the proprietary commercial examples:

- Maple (https://www.maplesoft.com/products/Maple/)
- Mathematica (https://www.wolfram.com/mathematica/)

and the open source examples are:

- Maxima (https://maxima.sourceforge.io/)
- SymPy (https://www.sympy.org/en/index.html)

C.2 Wolfram Alpha

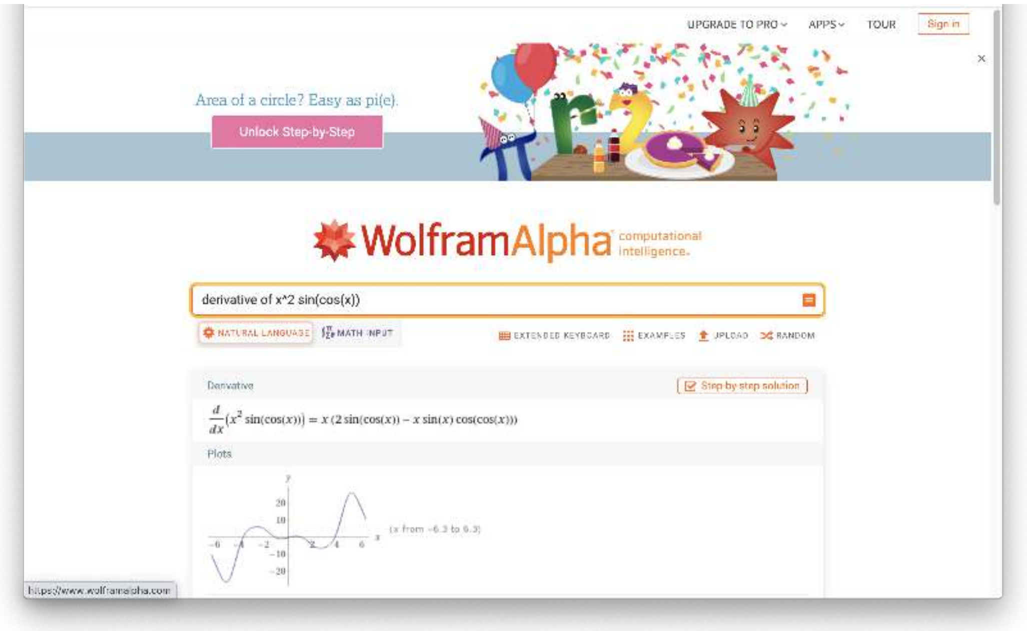
The personal license for commerical CAS can cost a few hundred dollars per year. But if you just want to try out a little, you can consider using WolframAlpha (https://www.wolframalpha.com/). It is like an online version of Mathematica, which you can type in your problem in the search box and it will give you the answer (and some related details). For example, solving

$$\frac{d}{dx}x^2\sin(\cos x)$$

can be done by typing

```
derivative of x^2 sin(cos(x))
```

as shown in the following screenshot:



derivative of x^2 sin(cos(x))

$$\frac{d}{dx}(x^2 \sin(\cos(x))) = x\,(2 \sin(\cos(x)) - x \sin(x) \cos(\cos(x)))$$

Figure C.1: Solution to a differentiation problem using WolframAlpha

For more complicated expression, you may need to learn to write in the Wolfram Language. The above example would be written as

```
D[x^2 Sin[Cos[x]], x]
```

C.3 SymPy

In Python, we have a CAS library, SymPy, that can do basic evaluations. If you haven't installed this library yet, you can do so using `pip`:

```
pip install sympy
```

SymPy is a library that allows you to define *symbols* in Python. Some common functions are also provided by SymPy to help you specify the problem. For example, the same differentiation problem as mentioned above can be solved using:

```
from sympy import *

x = Symbol("x")
expression = x**2 * sin(cos(x))
print(expression)
print(diff(expression))
```

Program C.1: Solving differentiation using SymPy which prints the following:

```
x**2*sin(cos(x))
-x**2*sin(x)*cos(cos(x)) + 2*x*sin(cos(x))
```

Output C.1: Solution to the differentiation problem from SymPy

In SymPy, we need to define variables as `Symbol` objects and use them to define an expression. The syntax in defining an expression is same as Python arithmatics: We need to explicitly use $*$ for all multiplications and exponents are introduced using $**$. Once you have the symbols defined, you can find limits using `limit()` function, differentiation using the `diff()` function and integration using `integrate()` function. Partial derivatives are also supported, for example,

$$y = \tanh(wx + b)$$

We can find $\partial y/\partial w$ and $\partial y/\partial b$ respectively as follows:

```
from sympy import *

w, x, b = symbols(”w x b”)
y = tanh(w*x + b)
print(y)
print(diff(y, w))
print(diff(y, b))
```

Program C.2: Partial differentiation using SymPy

which prints the following:

```
tanh(b + w*x)
x*(1 - tanh(b + w*x)**2)
1 - tanh(b + w*x)**2
```

Output C.2: Solution to the partial differentiation problem from SymPy

The symbols used to build an expression are not limited to a single letter. So you can write `w1 w2 w3 = symbols(”w1 w2 w3”)` if you prefer. Since single letter symbols are so common, you can avoid defining them but importing them from `sympy.abc` instead. Moreover, if you feel this notation is too clumpsy to read, you can choose to “pretty print” the SymPy expression using the `pprint()` function. This is illustrated in the following rewrite:

```
from sympy import *
from sympy.abc import w, x, b

y = tanh(w*x + b)
pprint(y)
pprint(diff(y, w))
pprint(diff(y, b))
```

Program C.3: Partial differentiation using SymPy

which prints the following if your console supports Unicode:

```
tanh(b + w⋅x)
  ⎛        2         ⎞
x⋅⎝1 - tanh (b + w⋅x)⎠
        2
1 - tanh (b + w⋅x)
```

Output C.3: Solution to the partial differentiation problem from SymPy

SymPy allows you to do much more than these. For example, solving an equation, simplify an expression, plotting are also supported. To know more, you should start with its tutorial and full documentation:

- ▷ SymPy tutorial (`https://docs.sympy.org/latest/tutorial/index.html`)
- ▷ SymPy documentation (`https://docs.sympy.org/latest/index.html`)

How Far You Have Come

You made it. Well done. Take a moment and look back at how far you have come. You now know:

- What is calculus and how it is introduced as a branch of mathematics
- What other branches of mathematics are related to calculus
- Two main topics of calculus are differentiation and integration, and they are reverse of each other
- Rate of change of a quantity or the slope in geometry can be represented by differentiation of a function
- Differentiation is done by taking limits, but there are a number of rules to help us do that faster
- Calculus is not only for a function with single parameter. We have multivariate calculus for vector-valued functions or functions with multiple variables
- With calculus, we can solve constrainted optimization problem using the method of Lagrange multipliers
- With calculus, we can approximate a function using Taylor series expansion
- How gradient descent uses differentiation of a function to determine direction of optimization
- How backpropagation procedure in neural networks gets its name from using the chain rule
- How the support vector machine find its solution using the method of Lagrange multiplier

Don't make light of this. You have come a long way in a short amount of time. You have developed the important and valuable foundational skills in calculus. You can now confidently:

- Understand the calculus notation in machine learning papers.
- Implement the calculus expressions of machine learning algorithms into code.
- Describe the calculus operations of your machine learning models.

The sky's the limit.

Thank You!

We want to take a moment and sincerely thank you for letting me help you start your calculus journey. We hope you keep learning and have fun as you continue to master machine learning.