Basics of Linear Algebra for Machine Learning

Discover the Mathematical Language of Data in Python

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

Acknowledgements

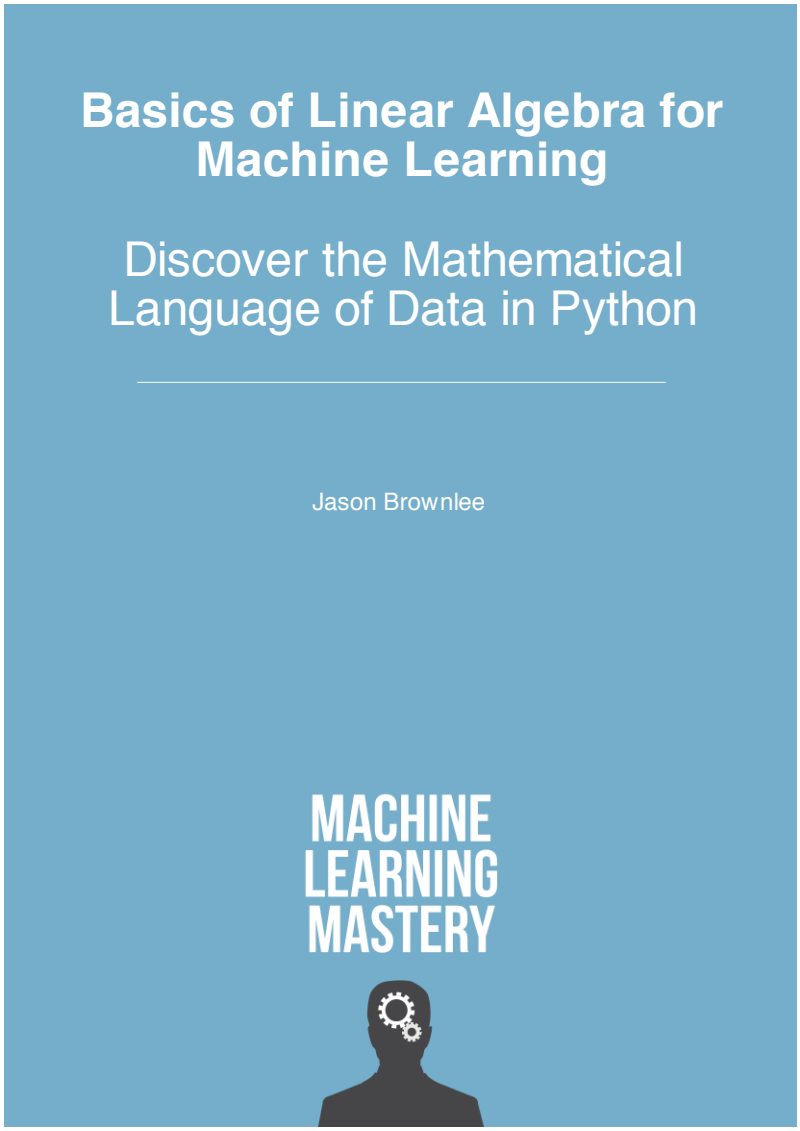
Special thanks to my copy editor Sarah Martin and my technical editors Arun Koshy and Andrei Cheremskoy.

Copyright

Edition: v1.1

This is Just a Sample

Thank-you for your interest in **Basics of Linear Algebra for Machine Learning**. This is just a sample of the full text. You can purchase the complete book online from: https://machinelearningmastery.com/linear_algebra_for_machine_learning/



Disclaimer

Acknowledgements

Special thanks to my copy editor Sarah Martin and my technical editors Arun Koshy and Andrei Cheremskoy.

Copyright

Edition: v1.1

Contents

Copyright i

Copyright iii

Contents iv

Preface v

I Introduction vii

Welcome viii
- Who Is This Book For? viii
- About Your Outcomes viii
- How to Read This Book ix
- About the Book Structure ix
- About Python Code Examples x
- About Further Reading xi
- About Getting Help xi
- Summary xi

II Foundations 1

1 Introduction to Linear Algebra 2

2 Linear Algebra and Machine Learning 3

3 Examples of Linear Algebra in Machine Learning 4
- 3.1 Overview 4
- 3.2 Dataset and Data Files 5
- 3.3 Images and Photographs 5
- 3.4 One Hot Encoding 6
- 3.5 Linear Regression 6
- 3.6 Regularization 6
- 3.7 Principal Component Analysis 7
- 3.8 Singular-Value Decomposition 7
- 3.9 Latent Semantic Analysis 7

3.10 Recommender Systems 8
3.11 Deep Learning 8
3.12 Summary 8

III NumPy 9

4 Introduction to NumPy Arrays 10

5 Index, Slice and Reshape NumPy Arrays 11

6 NumPy Array Broadcasting 12

IV Matrices 13

7 Vectors and Vector Arithmetic 14
7.1 Tutorial Overview 14
7.2 What is a Vector 14
7.3 Defining a Vector 15
7.4 Vector Arithmetic 15
7.5 Vector Dot Product 19
7.6 Vector-Scalar Multiplication 20
7.7 Extensions 21
7.8 Further Reading 21
7.9 Summary 22

8 Vector Norms 23

9 Matrices and Matrix Arithmetic 24

10 Types of Matrices 25

11 Matrix Operations 26

12 Sparse Matrices 27

13 Tensors and Tensor Arithmetic 28

V Factorization 29

14 Matrix Decompositions 30

15 Eigendecomposition 31

16 Singular Value Decomposition 32

VI Statistics 33

17 Introduction to Multivariate Statistics 34

18 Principal Component Analysis 35

19 Linear Regression 36

VII Appendix 37

A Getting Help 38

B How to Setup a Workstation for Python 39

C Linear Algebra Cheat Sheet 40

D Basic Math Notation 41

VIII Conclusions 42

How Far You Have Come 43

Preface

I wrote this book to help machine learning practitioners, like you, get on top of linear algebra, fast.

Linear Algebra Is Important in Machine Learning

There is no doubt that linear algebra is important in machine learning. Linear algebra is the mathematics of data. It's all vectors and matrices of numbers. Modern statistics is described using the notation of linear algebra and modern statistical methods harness the tools of linear algebra. Modern machine learning methods are described the same way, using the notations and tools drawn directly from linear algebra. Even some classical methods used in the field, such as linear regression via linear least squares and singular-value decomposition, are linear algebra methods, and other methods, such as principal component analysis, were born from the marriage of linear algebra and statistics. To read and understand machine learning, you must be able to read and understand linear algebra.

Practitioners Study Linear Algebra Too Early

If you ask how to get started in machine learning, you will very likely be told to start with linear algebra. We know that knowledge of linear algebra is critically important, but it does not have to be the place to start. Learning linear algebra first, then calculus, probability, statistics, and eventually machine learning theory is a long and slow bottom-up path. A better fit for developers is to start with systematic procedures that get results, and work back to the deeper understanding of theory, using working results as a context. I call this the top-down or results-first approach to machine learning, and linear algebra is not the first step, but perhaps the second or third.

Practitioners Study Too Much Linear Algebra

When practitioners do circle back to study linear algebra, they learn far more of the field than is required for or relevant to machine learning. Linear algebra is a large field of study that has tendrils into engineering, physics and quantum physics. There are also theorems and derivations for nearly everything, most of which will not help you get better skill from or a deeper understanding of your machine learning model. Only a specific subset of linear algebra is required, though you can always go deeper once you have the basics.

Practitioners Study Linear Algebra Wrong

Linear algebra textbooks will teach you linear algebra in the classical university bottom-up approach. This is too slow (and painful) for your needs as a machine learning practitioner. Like learning machine learning itself, take the top-down approach. Rather than starting with theorems and abstract concepts, you can learn the basics of linear algebra in a concrete way with data structures and worked examples of operations on those data structures. It's so much faster. Once you know how operations work, you can circle back and learn how they were derived.

A Better Way

This book was born out of my frustrations at seeing practitioner after practitioner diving into linear algebra textbooks and online courses designed for undergraduate students and giving up. The bottom-up approach is hard, especially if you already have a full time job. Linear algebra is not only important to machine learning, but it is also a lot of fun, or can be if it is approached in the right way. I put together this book to help you see the field the way I see it: as just another set of tools we can harness on our journey toward machine learning mastery.

Jason Brownlee
2018

Part I

Introduction

Welcome

Welcome to *Basics of Linear Algebra for Machine Learning*. Linear algebra is a pillar of machine learning.

The field started to be formalized about 150 years ago, but it was only about 70 years ago that modern linear algebra came into existence. It's a huge field of study that has made an impact on other areas of mathematics, such as statistics, as well as engineering and physics. Thankfully, we don't need to know the breadth and depth of the field of linear algebra in order to improve our understanding and application of machine learning.

I designed this book to teach you step-by-step the basics of linear algebra with concrete and executable examples in Python.

Who Is This Book For?

Before we get started, let's make sure you are in the right place. This book is for developers that may know some applied machine learning. Maybe you know how to work through a predictive modeling problem end-to-end, or at least most of the main steps, with popular tools. The lessons in this book do assume a few things about you, such as:

- You know your way around basic Python for programming.
- You may know some basic NumPy for array manipulation.
- You want to learn linear algebra to deepen your understanding and application of machine learning.

This guide was written in the top-down and results-first machine learning style that you're used to from MachineLearningMastery.com.

About Your Outcomes

This book will teach you the basics of linear algebra that you need to know as a machine learning practitioner. After reading and working through this book, you will know:

- What linear algebra is and why it is relevant and important to machine learning.
- How to create, index, and generally manipulate data in NumPy arrays.
- What a vector is and how to perform vector arithmetic and calculate vector norms.

- What a matrix is and how to perform matrix arithmetic, including matrix multiplication.
- A suite of types of matrices, their properties, and advanced operations involving matrices.
- What a tensor is and how to perform basic tensor arithmetic.
- Matrix factorization methods, including the eigendecomposition and singular-value decomposition.
- How to calculate and interpret basic statistics using the tools of linear algebra.
- How to implement methods using the tools of linear algebra such as principal component analysis and linear least squares regression.

This new basic understanding of linear algebra will impact your practice of machine learning in the following ways:

- Read the linear algebra mathematics in machine learning papers.
- Implement the linear algebra descriptions of machine learning algorithms.
- Describe your machine learning models using the notation and operations of linear algebra.

This book is not a substitute for an undergraduate course in linear algebra or a textbook for such a course, although it could complement to such materials. For a good list of top courses, textbooks, and other resources on linear algebra, see the *Further Reading* section at the end of each tutorial.

How to Read This Book

This book was written to be read linearly, from start to finish. That being said, if you know the basics and need help with a specific notation or operation, then you can flip straight to that section and get started. This book was designed for you to read on your workstation, on the screen, not on a tablet or eReader. My hope is that you have the book open right next to your editor and run the examples as you read about them.

This book is not intended to be read passively or be placed in a folder as a reference text. It is a playbook, a workbook, and a guidebook intended for you to learn by doing and then apply your new understanding with working Python examples. To get the most out of the book, I would recommend playing with the examples in each tutorial. Extend them, break them, then fix them. Try some of the extensions presented at the end of each lesson and let me know how you do.

About the Book Structure

This book was designed around major data structures, operations, and techniques in linear algebra that are directly relevant to machine learning algorithms. There are a lot of things you could learn about linear algebra, from theory to abstract concepts to APIs. My goal is to take

you straight to developing an intuition for the elements you must understand with laser-focused tutorials.

I designed the tutorials to focus on how to get things done with linear algebra. They give you the tools to both rapidly understand and apply each technique or operation. Each of the tutorials are designed to take you about one hour to read through and complete, excluding the extensions and further reading. You can choose to work through the lessons one per day, one per week, or at your own pace. I think momentum is critically important, and this book is intended to be read and used, not to sit idle. I would recommend picking a schedule and sticking to it.

The tutorials are divided into 5 parts:

- **Part 1: Foundation**. Discover a gentle introduction to the field of linear algebra and the relationship it has with the field of machine learning.
- **Part 2: NumPy**. Discover NumPy tutorials that show you how to create, index, slice, and reshape NumPy arrays, the main data structure used in machine learning and the basis for linear algebra examples in this book.
- **Part 3: Matrices**. Discover the key structures for holding and manipulating data in linear algebra in vectors, matrices, and tensors.
- **Part 4: Factorization**. Discover a suite of methods for decomposing a matrix into its constituent elements in order to make numerical operations more efficient and more stable.
- **Part 5: Statistics**. Discover statistics through the lens of linear algebra and its application to principal component analysis and linear regression.

Each part targets a specific learning outcome, and so does each tutorial within each part. This acts as a filter to ensure you are only focused on the things you need to know to get to a specific result and do not get bogged down in the math or near-infinite number of digressions.

The tutorials were not designed to teach you everything there is to know about each of the theories or techniques of linear algebra. They were designed to give you an understanding of how they work, how to use them, and how to interpret the results the fastest way I know how: to learn by doing.

About Python Code Examples

The code examples were carefully designed to demonstrate the purpose of a given lesson. Code examples are complete and standalone. The code for each lesson will run as-is with no code from prior lessons or third parties required beyond the installation of the required packages. A complete working example is presented with each tutorial for you to inspect and copy-and-paste. All source code is also provided with the book and I would recommend running the provided files whenever possible to avoid any copy-paste issues.

The provided code was developed in a text editor and intended to be run on the command line. No special IDE or notebooks are required. If you are using a more advanced development environment and are having trouble, try running the example from the command line instead. All code examples were tested on a POSIX-compatible machine with Python 3.

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, I try to list and link to the relevant API documentation for key functions used in each lesson so you can learn more about them. I have tried to link to books on Amazon so that you can learn more about them. I don't know everything, and if you discover a good resource related to a given lesson, please let me know so I can update the book.

About Getting Help

You might need help along the way. Don't worry; you are not alone.

- **Help with a Technique?** If you need help with the technical aspects of a specific operation or technique, see the *Further Reading* sections at the end of each lesson.
- **Help with NumPy?** If you need help with using the NumPy library, see the list of resources in the *Further Reading* section at the end of each lesson, and also see *Appendix A*.
- **Help with your workstation?** If you need help setting up your environment, I would recommend using Anaconda and following my tutorial in *Appendix B*.
- **Help with the math?** I provided a list of locations where you can search for answers and ask questions about linear algebra math in *Appendix A*. You can also see *Appendix D* for a crash course on math notation.
- **Help in general?** You can shoot me an email. My details are in *Appendix A*.

Summary

Are you ready? Let's dive in!

Next

Next up you will discover a gentle introduction to the field of linear algebra.

Part II

Foundations

Chapter 1

Introduction to Linear Algebra

Chapter 2

Linear Algebra and Machine Learning

Chapter 3

Examples of Linear Algebra in Machine Learning

Linear algebra is a sub-field of mathematics concerned with vectors, matrices and linear transforms. It is a key foundation to the field of machine learning from notations used to describe the operation of algorithms, to the implementation of algorithms in code. Although linear algebra is integral to the field of machine learning, the tight relationship is often left unexplained or explained using abstract concepts such as vector spaces or specific matrix operations. In this chapter, you will discover 10 common examples of machine learning that you may be familiar with that use, require and are really best understood using linear algebra. After reading this chapter, you will know:

- The use if linear algebra structures when working with data such as tabular datasets and images.
- Linear algebra concepts when working with data preparation such as one hot encoding and dimensionality reduction.
- The in-grained use of linear algebra notation and methods in sub-fields such as deep learning, natural language processing and recommender systems.

Let's get started.

3.1 Overview

In this chapter, we will review 10 obvious and concrete examples of linear algebra in machine learning. I tried to pick examples that you may be familiar with or have even worked with before. They are:

1. Dataset and Data Files
2. Images and Photographs
3. One Hot Encoding
4. Linear Regression

5. Regularization
6. Principal Component Analysis
7. Singular-Value Decomposition
8. Latent Semantic Analysis
9. Recommender Systems
10. Deep Learning

Do you have your own favorite example of linear algebra in machine learning? Let me know.

3.2 Dataset and Data Files

In machine learning, you fit a model on a dataset. This is the table like set of numbers where each row represents an observation and each column represents a feature of the observation. For example, below is a snippet of the Iris flowers dataset[1]:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
...
```

Listing 3.1: Sample output of the iris flowers dataset.

This data is in fact a matrix, a key data structure in linear algebra. Further, when you split the data into inputs and outputs to fit a supervised machine learning model, such as the measurements and the flower species, you have a matrix (X) and a vector (y). The vector is another key data structure in linear algebra. Each row has the same length, i.e. the same number of columns, therefore we can say that the data is vectorized where rows can be provided to a model one at a time or in batch and the model can be pre-configured to expect rows of a fixed width.

3.3 Images and Photographs

Perhaps you are more used to working with images or photographs in computer vision applications. Each image that you work with is itself a table structure with a width and height and one pixel value in each cell for black and white images or 3 pixel values in each cell for a color image. A photo is yet another example of a matrix from linear algebra. Operations on the image, such as cropping, scaling, shearing and so on are all described using the notation and operations of linear algebra.

[1] http://archive.ics.uci.edu/ml/datasets/Iris

3.4 One Hot Encoding

Sometimes you work with categorical data in machine learning. Perhaps the class labels for classification problems, or perhaps categorical input variables. It is common to encode categorical variables to make their easier to work with and learn by some techniques. A popular encoding for categorical variables is the one hot encoding. A one hot encoding is where a table is created to represent the variable with one column for each category and a row for each example in the dataset. A check or one-value is added in the column for the categorical value for a given row, and a zero-value is added to all other columns. For example, the variable color variable with the 3 rows:

```
red
green
blue
...
```

Listing 3.2: Example of a categorical variable.

Might be encoded as:

```
red, green, blue
  1,    0,    0
  0,    1,    0
  0,    0,    1
...
```

Listing 3.3: Example of a one hot encoded categorical variable.

Each row is encoded as a binary vector, a vector with zero or one values and this is an example of a sparse representation, a whole sub-field of linear algebra.

3.5 Linear Regression

Linear regression is an old method from statistics for describing the relationships between variables. It is often used in machine learning for predicting numerical values in simpler regression problems. There are many ways to describe and solve the linear regression problem, i.e. finding a set of coefficients that when multiplied by each of the input variables and added together results in the best prediction of the output variable. If you have used a machine learning tool or library, the most common way of solving linear regression is via a least squares optimization that is solved using matrix factorization methods from linear regression, such as an LU decomposition or an singular-value decomposition or SVD. Even the common way of summarizing the linear regression equation uses linear algebra notation:

$$y = A \cdot b \tag{3.1}$$

Where y is the output variable A is the dataset and b are the model coefficients.

3.6 Regularization

In applied machine learning, we often seek the simplest possible models that achieve the best skill on our problem. Simpler models are often better at generalizing from specific examples

to unseen data. In many methods that involve coefficients, such as regression methods and artificial neural networks, simpler models are often characterized by models that have smaller coefficient values. A technique that is often used to encourage a model to minimize the size of coefficients while it is being fit on data is called regularization. Common implementations include the L^2 and L^1 forms of regularization. Both of these forms of regularization are in fact a measure of the magnitude or length of the coefficients as a vector and are methods lifted directly from linear algebra called the vector norm.

3.7 Principal Component Analysis

Often a dataset has many columns, perhaps tens, hundreds, thousands or more. Modeling data with many features is challenging, and models built from data that include irrelevant features are often less skillful than models trained from the most relevant data. It is hard to know which features of the data are relevant and which are not. Methods for automatically reducing the number of columns of a dataset are called dimensionality reduction, and perhaps the most popular is method is called the principal component analysis or PCA for short. This method is used in machine learning to create projections of high-dimensional data for both visualization and for training models. The core of the PCA method is a matrix factorization method from linear algebra. The eigendecomposition can be used and more robust implementations may use the singular-value decomposition or SVD.

3.8 Singular-Value Decomposition

Another popular dimensionality reduction method is the singular-value decomposition method or SVD for short. As mentioned and as the name of the method suggests, it is a matrix factorization method from the field of linear algebra. It has wide use in linear algebra and can be used directly in applications such as feature selection, visualization, noise reduction and more. We will see two more cases below of using the SVD in machine learning.

3.9 Latent Semantic Analysis

In the sub-field of machine learning for working with text data called natural language processing, it is common to represent documents as large matrices of word occurrences. For example, the columns of the matrix may be the known words in the vocabulary and rows may be sentences, paragraphs, pages or documents of text with cells in the matrix marked as the count or frequency of the number of times the word occurred. This is a sparse matrix representation of the text. Matrix factorization methods such as the singular-value decomposition can be applied to this sparse matrix which has the effect of distilling the representation down to its most relevant essence. Documents processed in thus way are much easier to compare, query and use as the basis for a supervised machine learning model. This form of data preparation is called Latent Semantic Analysis or LSA for short, and is also known by the name Latent Semantic Indexing or LSI.

3.10 Recommender Systems

Predictive modeling problems that involve the recommendation of products are called recommender systems, a sub-field of machine learning. Examples include the recommendation of books based on previous purchases and purchases by customers like you on Amazon, and the recommendation of movies and TV shows to watch based on your viewing history and viewing history of subscribers like you on Netflix. The development of recommender systems is primarily concerned with linear algebra methods. A simple example is in the calculation of the similarity between sparse customer behavior vectors using distance measures such as Euclidean distance or dot products. Matrix factorization methods like the singular-value decomposition are used widely in recommender systems to distill item and user data to their essence for querying and searching and comparison.

3.11 Deep Learning

Artificial neural networks are nonlinear machine learning algorithms that are inspired by elements of the information processing in the brain and have proven effective at a range of problems not least predictive modeling. Deep learning is the recent resurged use of artificial neural networks with newer methods and faster hardware that allow for the development and training of larger and deeper (more layers) networks on very large datasets. Deep learning methods are routinely achieve state-of-the-art results on a range of challenging problems such as machine translation, photo captioning, speech recognition and much more.

At their core, the execution of neural networks involves linear algebra data structures multiplied and added together. Scaled up to multiple dimensions, deep learning methods work with vectors, matrices and even tensors of inputs and coefficients, where a tensor is a matrix with more than two dimensions. Linear algebra is central to the description of deep learning methods via matrix notation to the implementation of deep learning methods such as Google's TensorFlow Python library that has the word "tensor" in its name.

3.12 Summary

In this chapter, you discovered 10 common examples of machine learning that you may be familiar with that use and require linear algebra. Specifically, you learned:

- The use of linear algebra structures when working with data such as tabular datasets and images.
- Linear algebra concepts when working with data preparation such as one hot encoding and dimensionality reduction.
- The in-grained use of linear algebra notation and methods in sub-fields such as deep learning, natural language processing and recommender systems.

3.12.1 Next

This is the end of the first part, in the next part you will discover how to manipulate arrays of data in Python using NumPy.

Part III

NumPy

Chapter 4

Introduction to NumPy Arrays

Chapter 5

Index, Slice and Reshape NumPy Arrays

Chapter 6

NumPy Array Broadcasting

Part IV

Matrices

Chapter 7

Vectors and Vector Arithmetic

Vectors are a foundational element of linear algebra. Vectors are used throughout the field of machine learning in the description of algorithms and processes such as the target variable (y) when training an algorithm. In this tutorial, you will discover linear algebra vectors for machine learning. After completing this tutorial, you will know:

- What a vector is and how to define one in Python with NumPy.
- How to perform vector arithmetic such as addition, subtraction, multiplication and division.
- How to perform additional operations such as dot product and multiplication with a scalar.

Let's get started.

7.1 Tutorial Overview

This tutorial is divided into 5 parts; they are:

1. What is a Vector
2. Defining a Vector
3. Vector Arithmetic
4. Vector Dot Product
5. Vector-Scalar Multiplication

7.2 What is a Vector

A vector is a tuple of one or more values called scalars.

> Vectors are built from components, which are ordinary numbers. You can think of a vector as a list of numbers, and vector algebra as operations performed on the numbers in the list.
>
> — Page 69, *No Bullshit Guide To Linear Algebra*, 2017.

Vectors are often represented using a lowercase character such as v; for example:

$$v = (v_1, v_2, v_3) \tag{7.1}$$

Where v_1, v_2, v_3 are scalar values, often real values.
Vectors are also shown using a vertical representation or a column; for example:

$$v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{7.2}$$

It is common to represent the target variable as a vector with the lowercase y when describing the training of a machine learning algorithm. It is common to introduce vectors using a geometric analogy, where a vector represents a point or coordinate in an n-dimensional space, where n is the number of dimensions, such as 2. The vector can also be thought of as a line from the origin of the vector space with a direction and a magnitude.

These analogies are good as a starting point, but should not be held too tightly as we often consider very high dimensional vectors in machine learning. I find the vector-as-coordinate the most compelling analogy in machine learning. Now that we know what a vector is, let's look at how to define a vector in Python.

7.3 Defining a Vector

We can represent a vector in Python as a NumPy array. A NumPy array can be created from a list of numbers. For example, below we define a vector with the length of 3 and the integer values 1, 2 and 3.

```
# create a vector
from numpy import array
# define vector
v = array([1, 2, 3])
print(v)
```

Listing 7.1: Example of defining a vector.

The example defines a vector with 3 elements. Running the example prints the defined vector.

```
[1 2 3]
```

Listing 7.2: Sample output from defining a vector.

7.4 Vector Arithmetic

In this section will demonstrate simple vector-vector arithmetic, where all operations are performed element-wise between two vectors of equal length to result in a new vector with the same length

7.4.1 Vector Addition

Two vectors of equal length can be added together to create a new third vector.

$$c = a + b \tag{7.3}$$

The new vector has the same length as the other two vectors. Each element of the new vector is calculated as the addition of the elements of the other vectors at the same index; for example:

$$c = (a_1 + b_1, a_2 + b_2, a_3 + b_3) \tag{7.4}$$

Or, put another way:

$$\begin{aligned} c[0] &= a[0] + b[0] \\ c[1] &= a[1] + b[1] \\ c[2] &= a[2] + b[2] \end{aligned} \tag{7.5}$$

We can add vectors directly in Python by adding NumPy arrays.

```
# vector addition
from numpy import array
# define first vector
a = array([1, 2, 3])
print(a)
# define second vector
b = array([1, 2, 3])
print(b)
# add vectors
c = a + b
print(c)
```

Listing 7.3: Example of vector addition.

The example defines two vectors with three elements each, then adds them together. Running the example first prints the two parent vectors then prints a new vector that is the addition of the two vectors.

```
[1 2 3]

[1 2 3]

[2 4 6]
```

Listing 7.4: Sample output from vector addition.

7.4.2 Vector Subtraction

One vector can be subtracted from another vector of equal length to create a new third vector.

$$c = a - b \tag{7.6}$$

As with addition, the new vector has the same length as the parent vectors and each element of the new vector is calculated as the subtraction of the elements at the same indices.

$$c = (a_1 - b_1, a_2 - b_2, a_3 - b_3) \tag{7.7}$$

Or, put another way:

$$\begin{aligned} c[0] &= a[0] - b[0] \\ c[1] &= a[1] - b[1] \\ c[2] &= a[2] - b[2] \end{aligned} \tag{7.8}$$

The NumPy arrays can be directly subtracted in Python.

```
# vector subtraction
from numpy import array
# define first vector
a = array([1, 2, 3])
print(a)
# define second vector
b = array([0.5, 0.5, 0.5])
print(b)
# subtract vectors
c = a - b
print(c)
```

Listing 7.5: Example of vector subtraction.

The example defines two vectors with three elements each, then subtracts the first from the second. Running the example first prints the two parent vectors then prints the new vector that is the first minus the second.

```
[1 2 3]

[ 0.5 0.5 0.5]

[ 0.5 1.5 2.5]
```

Listing 7.6: Sample output from vector subtraction.

7.4.3 Vector Multiplication

Two vectors of equal length can be multiplied together.

$$c = a \times b \tag{7.9}$$

As with addition and subtraction, this operation is performed element-wise to result in a new vector of the same length.

$$c = (a_1 \times b_1, a_2 \times b_2, a_3 \times b_3) \tag{7.10}$$

or

$$c = (a_1 b_1, a_2 b_2, a_3 b_3) \tag{7.11}$$

Or, put another way:

$$\begin{aligned} c[0] &= a[0] \times b[0] \\ c[1] &= a[1] \times b[1] \\ c[2] &= a[2] \times b[2] \end{aligned} \tag{7.12}$$

We can perform this operation directly in NumPy.

```
# vector multiplication
from numpy import array
# define first vector
a = array([1, 2, 3])
print(a)
# define second vector
b = array([1, 2, 3])
print(b)
# multiply vectors
c = a * b
print(c)
```

Listing 7.7: Example of vector multiplication.

The example defines two vectors with three elements each, then multiplies the vectors together. Running the example first prints the two parent vectors, then the new vector is printed.

```
[1 2 3]

[1 2 3]

[1 4 9]
```

Listing 7.8: Sample output from vector multiplication.

7.4.4 Vector Division

Two vectors of equal length can be divided.

$$c = \frac{a}{b} \tag{7.13}$$

As with other arithmetic operations, this operation is performed element-wise to result in a new vector of the same length.

$$c = (\frac{a_1}{b_1}, \frac{a_2}{b_2}, \frac{a_3}{b_3}) \tag{7.14}$$

Or, put another way:

$$\begin{aligned} c[0] &= a[0]/b[0] \\ c[1] &= a[1]/b[1] \\ c[2] &= a[2]/b[2] \end{aligned} \tag{7.15}$$

We can perform this operation directly in NumPy.

```
# vector division
from numpy import array
# define first vector
a = array([1, 2, 3])
print(a)
# define second vector
b = array([1, 2, 3])
print(b)
# divide vectors
c = a / b
print(c)
```

Listing 7.9: Example of vector division.

The example defines two vectors with three elements each, then divides the first by the second. Running the example first prints the two parent vectors, followed by the result of the vector division.

```
[1 2 3]

[1 2 3]

[ 1.  1.  1.]
```

Listing 7.10: Sample output from vector division.

7.5 Vector Dot Product

We can calculate the sum of the multiplied elements of two vectors of the same length to give a scalar. This is called the dot product, named because of the dot operator used when describing the operation.

> The dot product is the key tool for calculating vector projections, vector decompositions, and determining orthogonality. The name dot product comes from the symbol used to denote it.
>
> — Page 110, *No Bullshit Guide To Linear Algebra*, 2017.

$$c = a \cdot b \tag{7.16}$$

The operation can be used in machine learning to calculate the weighted sum of a vector. The dot product is calculated as follows:

$$c = (a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3) \tag{7.17}$$

or

$$c = (a_1 b_1 + a_2 b_2 + a_3 b_3) \tag{7.18}$$

We can calculate the dot product between two vectors in Python using the `dot()` function on a NumPy array.

```
# vector dot product
from numpy import array
# define first vector
a = array([1, 2, 3])
print(a)
# define second vector
b = array([1, 2, 3])
print(b)
# multiply vectors
c = a.dot(b)
print(c)
```

Listing 7.11: Example of vector dot product.

The example defines two vectors with three elements each, then calculates the dot product. Running the example first prints the two parent vectors, then the scalar dot product.

```
[1 2 3]

[1 2 3]

14
```

Listing 7.12: Sample output from vector dot product.

7.6 Vector-Scalar Multiplication

A vector can be multiplied by a scalar, in effect scaling the magnitude of the vector. To keep notation simple, we will use lowercase s to represent the scalar value.

$$c = s \times v \tag{7.19}$$

or

$$c = sv \tag{7.20}$$

The multiplication is performed on each element of the vector to result in a new scaled vector of the same length.

$$c = (s \times v_1, s \times v_2, s \times v_3) \tag{7.21}$$

Or, put another way:

$$\begin{aligned} c[0] &= v[0] \times s \\ c[1] &= v[1] \times s \\ c[2] &= v[2] \times s \end{aligned} \tag{7.22}$$

We can perform this operation directly with the NumPy array.

```
# vector-scalar multiplication
from numpy import array
# define vector
a = array([1, 2, 3])
```

```
print(a)
# define scalar
s = 0.5
print(s)
# multiplication
c = s * a
print(c)
```

Listing 7.13: Example of vector-scalar multiplication.

The example first defines the vector and the scalar then multiplies the vector by the scalar. Running the example first prints the parent vector, then scalar, and then the result of multiplying the two together.

```
[1 2 3]

0.5

[ 0.5 1.  1.5]
```

Listing 7.14: Sample output from vector-scalar multiplication.

Similarly, vector-scalar addition, subtraction, and division can be performed in the same way.

7.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Create one example using each operation using your own small array data.
- Implement each vector arithmetic operation manually for vectors defined as lists.
- Search machine learning papers and find 1 example of each operation being used.

If you explore any of these extensions, I'd love to know.

7.8 Further Reading

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

7.8.1 Books

- Section 1.15, Vectors. *No Bullshit Guide To Linear Algebra*, 2017.
  http://amzn.to/2k76D4
- Section 2.2, Vector operations. *No Bullshit Guide To Linear Algebra*, 2017.
  http://amzn.to/2k76D4
- Section 1.1 Vectors and Linear Combinations, *Introduction to Linear Algebra*, Fifth Edition, 2016.
  http://amzn.to/2j2J0g4

- Section 2.1 Scalars, Vectors, Matrices and Tensors, *Deep Learning*, 2016.
  http://amzn.to/2j4oKuP
- Section 1.B Definition of Vector Space, *Linear Algebra Done Right*, Third Edition, 2015.
  http://amzn.to/2BGuEqI

7.8.2 API

- `numpy.array()` API.
  https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.array.html
- `numpy.dot()` API.
  https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dot.html

7.8.3 Articles

- Vector space on Wikipedia.
  https://en.wikipedia.org/wiki/Vector_space
- Dot product on Wikipedia.
  https://en.wikipedia.org/wiki/Dot_product

7.9 Summary

In this tutorial, you discovered linear algebra vectors for machine learning. Specifically, you learned:

- What a vector is and how to define one in Python with NumPy.
- How to perform vector arithmetic such as addition, subtraction, multiplication and division.
- How to perform additional operations such as dot product and multiplication with a scalar.

7.9.1 Next

In the next chapter you will discover vector norms for calculating the magnitude of vectors.

Chapter 8

Vector Norms

Chapter 9

Matrices and Matrix Arithmetic

Chapter 10

Types of Matrices

Chapter 11

Matrix Operations

Chapter 12

Sparse Matrices

Chapter 13

Tensors and Tensor Arithmetic

Part V

Factorization

Chapter 14

Matrix Decompositions

Chapter 15

Eigendecomposition

Chapter 16

Singular Value Decomposition

Part VI

Statistics

Chapter 17

Introduction to Multivariate Statistics

Chapter 18

Principal Component Analysis

Chapter 19

Linear Regression

Part VII

Appendix

Appendix A

Getting Help

Appendix B

How to Setup a Workstation for Python

Appendix C

Linear Algebra Cheat Sheet

Appendix D

Basic Math Notation

Part VIII

Conclusions

How Far You Have Come

This is Just a Sample

Thank-you for your interest in **Basics of Linear Algebra for Machine Learning**. This is just a sample of the full text. You can purchase the complete book online from: https://machinelearningmastery.com/linear_algebra_for_machine_learning/