

EXPERT INSIGHT

**HADELIN
DE PONTEVES**

A composite image featuring a man's face on the left and a digital head profile on the right. The digital head is composed of a grid of colorful lines (red, green, blue, yellow) and dots, resembling a data visualization or a neural network structure. The background is dark, making the colors of the digital head stand out.

AI CRASH COURSE

A fun and hands-on introduction to machine learning, reinforcement learning, deep learning, and artificial intelligence with Python

Packt

AI Crash Course

A fun and hands-on introduction to machine learning, reinforcement learning, deep learning, and artificial intelligence with Python

Hadelin de Ponteves

Packt

BIRMINGHAM - MUMBAI

AI Crash Course

Copyright © 2019 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Acquisition Editor: Jonathan Malysiak

Acquisition Editor – Peer Reviews: Suresh Jain

Content Development Editor: Alex Patterson

Project Editor: Kishor Rit

Technical Editor: Aniket Shetty

Proofreader: Safis Editing

Indexer: Rekha Nair

Presentation Designer: Sandip Tadge

First published: November 2019

Production reference: 3060120

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-83864-535-9

www.packt.com

Table of Contents

Preface	vii
Chapter 1: Welcome to the Robot World	1
Beginning the AI journey	1
Four different AI models	2
The models in practice	3
Fundamentals	3
Thompson Sampling	3
Q-learning	3
Deep Q-learning	3
Deep convolutional Q-learning	4
Where can learning AI take you?	4
Energy	4
Healthcare	4
Transport and logistics	5
Education	5
Security	5
Employment	5
Smart homes and robots	5
Entertainment and happiness	6
Environment	6
Economy, business, and finance	6
Summary	7
Chapter 2: Discover Your AI Toolkit	9
The GitHub page	9
Colaboratory	11
Summary	15

Chapter 3: Python Fundamentals – Learn How to Code in Python	17
Displaying text	18
Exercise	18
Variables and operations	19
Exercise	20
Lists and arrays	20
Exercise	22
if statements and conditions	22
Exercise	23
for and while loops	24
Exercise	27
Functions	27
Exercise	28
Classes and objects	29
Exercise	31
Summary	31
Chapter 4: AI Foundation Techniques	33
What is Reinforcement Learning?	33
The five principles of Reinforcement Learning	34
Principle #1 – The input and output system	34
Principle #2 – The reward	35
Principle #3 – The AI environment	37
Principle #4 – The Markov decision process	37
Principle #5 – Training and inference	38
Training mode	38
Inference mode	39
Summary	40
Chapter 5: Your First AI Model – Beware the Bandits!	41
The multi-armed bandit problem	41
The Thompson Sampling model	42
Coding the model	43
Understanding the model	47
What is a distribution?	48
Tackling the MABP	52
The Thompson Sampling strategy in three steps	55
The final touch of shaping your Thompson Sampling intuition	56
Thompson Sampling against the standard model	57
Summary	58

Chapter 6: AI for Sales and Advertising – Sell like the Wolf of AI Street	59
Problem to solve	59
Building the environment inside a simulation	61
Running the simulation	64
Recap	66
AI solution and intuition refresher	66
AI solution	67
Intuition	68
Implementation	68
Thompson Sampling vs. Random Selection	69
Performance measure	69
Let's start coding	69
The final result	74
Summary	76
Chapter 7: Welcome to Q-Learning	77
The Maze	78
Beginnings	78
Building the environment	79
The states	79
The actions	80
The rewards	81
Building the AI	85
The Q-value	85
The temporal difference	86
The Bellman equation	87
Reinforcement intuition	88
The whole Q-learning process	88
Training mode	89
Inference mode	89
Summary	90
Chapter 8: AI for Logistics – Robots in a Warehouse	91
Building the environment	94
The states	94
The actions	95
The rewards	95
AI solution refresher	96
Initialization (first iteration)	96
Next iterations	96
Implementation	97
Part 1 – Building the environment	98
Part 2 – Building the AI Solution with Q-learning	101

Part 3 – Going into production	103
Improvement 1 – Automating reward attribution	105
Improvement 2 – Adding an intermediate goal	108
Summary	111
Chapter 9: Going Pro with Artificial Brains – Deep Q-Learning	113
Predicting house prices	114
Uploading the dataset	114
Importing libraries	116
Excluding variables	117
Data preparation	119
Scaling data	119
Building the neural network	122
Training the neural network	123
Displaying results	123
Deep learning theory	125
The neuron	125
Biological neurons	125
Artificial neurons	127
The activation function	128
The threshold activation function	129
The sigmoid activation function	130
The rectifier activation function	131
How do neural networks work?	133
How do neural networks learn?	135
Forward-propagation and back-propagation	136
Gradient Descent	137
Batch gradient descent	140
Stochastic gradient descent	143
Mini-batch gradient descent	145
Deep Q-learning	145
The Softmax method	147
Deep Q-learning recap	149
Experience replay	149
The whole deep Q-learning algorithm	150
Summary	151
Chapter 10: AI for Autonomous Vehicles – Build a Self-Driving Car	153
Building the environment	153
Defining the goal	156
Setting the parameters	160
The input states	163

The output actions	165
The rewards	166
AI solution refresher	168
Implementation	169
Step 1 – Importing the libraries	170
Step 2 – Creating the architecture of the neural network	171
Step 3 – Implementing experience replay	175
Step 4 – Implementing deep Q-learning	177
The demo	188
Installing Anaconda	189
Creating a virtual environment with Python 3.6	190
Installing PyTorch	192
Installing Kivy	194
Summary	205
Chapter 11: AI for Business –	
Minimize Costs with Deep Q-Learning	207
Problem to solve	207
Building the environment	208
Parameters and variables of the server environment	208
Assumptions of the server environment	209
Assumption 1 – We can approximate the server temperature	209
Assumption 2 – We can approximate the energy costs	210
Simulation	211
Overall functioning	212
Defining the states	214
Defining the actions	214
Defining the rewards	215
Final simulation example	216
AI solution	220
The brain	221
Implementation	223
Step 1 – Building the environment	224
Step 2 – Building the brain	232
Without dropout	233
With dropout	237
Step 3 – Implementing the deep reinforcement learning algorithm	238
Step 4: Training the AI	245
No early stopping	246
Early stopping	254
Step 5 – Testing the AI	256
The demo	258

Recap – The general AI framework/Blueprint	268
Summary	270
Chapter 12: Deep Convolutional Q-Learning	271
What are CNNs used for?	271
How do CNNs work?	273
Step 1 – Convolution	275
Step 2 – Max pooling	277
Step 3 – Flattening	280
Step 4 – Full connection	282
Deep convolutional Q-learning	284
Summary	285
Chapter 13: AI for Games – Become the Master at Snake	287
Problem to solve	288
Building the environment	288
Defining the states	289
Defining the actions	290
Defining the rewards	292
AI solution	293
The brain	293
The experience replay memory	295
Implementation	295
Step 1 – Building the environment	296
Step 2 – Building the brain	303
Step 3 – Building the experience replay memory	307
Step 4 – Training the AI	309
Step 5 – Testing the AI	315
The demo	317
Installation	317
The results	323
Summary	325
Chapter 14: Recap and Conclusion	327
Recap – The general AI framework/blueprint	327
Exploring what's next for you in AI	329
Practice, practice, and practice	329
Networking	330
Never stop learning	331
Other Books You May Enjoy	333
Index	337

Preface

Hello, data scientists and AI enthusiasts. For many years I've created online courses on Artificial Intelligence (AI), which have been very successful and contributed well to the AI community. However, something essential was missing. At one point, so many AI courses were made that most of my students asked me for guidance on how to take the courses. So instead of providing an order in which to take the courses, I decided to create an all-in-one full guide to AI as a book, which would include in a perfect structure all the best explanations and real-world practical activities from my courses.

You see, my goal is to democratize AI and raise awareness among everyone of the fact that AI is an accessible technology that can make a difference for the better in this world. I am trying my best to spread knowledge around the world to get people prepared for the future jobs and opportunities of this 21st century. And I thought some people would learn AI much more efficiently from an all-in-one book they can take anywhere, rather than completing tens of online courses that can be hard to navigate. That being said, this book is also a great additional resource for those people who do prefer, and take, online courses.

My simple hope for this book is that more people learn AI the right way, as a result of me offering them this efficient alternative to online courses. I've succeeded at the challenge of including the best of my training in a single book, and today I'm truly happy to release it. I sincerely hope it will help more people land their dream job, grow an amazing career in data science or AI, and bring beautiful solutions to the tough challenges of this 21st century.

Who this book is for

Anyone interested in machine learning, deep learning, or AI.

People who aren't that comfortable with coding, but who are interested in AI and want to apply it easily to real-world problems.

College or university students who want to start a career in data science or AI.

Data analysts who want to level up in AI.

Anyone who isn't satisfied with their job and wants to take the first steps toward a career in data science.

Business owners who want to add value to their business by using powerful AI tools.

Entrepreneurs who are eager to learn how to leverage AI to optimize their business, maximize profitability, and increase efficiency.

AI practitioners who want to know what projects they can offer to their employees.

Aspiring data scientists, looking for business cases to add to their portfolio.

Technology enthusiasts interested in leveraging machine learning and AI to solve business problems.

Consultants who want to transition companies into being AI-driven businesses.

Students with at least high school knowledge in math, who want to start learning AI.

What this book covers

Chapter 1, Welcome to the Robot World, introduces you to the world of Artificial Intelligence.

Chapter 2, Discover Your AI Toolkit, uncovers an easy-to-use toolkit of all the AI models as Python files, ready to run thanks to the amazing Google Colaboratory platform.

Chapter 3, Python Fundamentals – Learn How to Code in Python, provides the right Python fundamentals and teaches you how to code in Python.

Chapter 4, AI Foundation Techniques, introduces you to reinforcement learning and its five fundamental principles.

Chapter 5, Your First AI Model – Beware the Bandits!, teaches the theory of the multi-armed bandit problem and how to solve it in the best way with the Thompson Sampling AI model.

Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street, applies the Thompson Sampling AI model of *Chapter 5* to solve a real-world business problem related to sales and advertising.

Chapter 7, Welcome to Q-Learning, introduces the theory of the Q-learning AI model.

Chapter 8, AI for Logistics – Robots in a Warehouse, applies the Q-learning AI model of *Chapter 7* to solve a real-world business problem related to logistics optimization.

Chapter 9, Going Pro with Artificial Brains – Deep Q-Learning, introduces the fundamentals of deep learning and the theory of the deep Q-learning AI model.

Chapter 10, AI for Autonomous Vehicles – Build a Self-Driving Car, applies the deep Q-learning AI model of *Chapter 9* to build a virtual self-driving car.

Chapter 11, AI for Business – Minimize Cost with Deep Q-Learning, applies the deep Q-learning AI model of *Chapter 9* to solve a real-world business problem related to cost optimization.

Chapter 12, Deep Convolutional Q-Learning, introduces the fundamentals of convolutional neural networks and the theory of the deep convolutional Q-learning AI model.

Chapter 13, AI for Games – Become the Master at Snake, applies the deep convolutional Q-learning AI model of *Chapter 12* to beat the famous *Snake* video game

Chapter 14, Recap and Conclusion, concludes the book with a recap of how to create an AI framework and some final words from the author about your future in the world of AI.

To get the most out of this book

- You don't need to know much before we begin; the book contains refreshers on all the prerequisites needed to understand the AI models. There's also a full chapter on Python fundamentals to help you learn, if you need to, how to code in Python.
- There are no required prior installations, since all the practical instructions are provided from scratch in the book. You only need to have your computer ready and switched on.

- I recommend you have Google open while reading the book, so that you can visit the links provided in the book as resources, and to check out the math concepts behind the AI models of this book in more detail.

Download the example code files

You can download the example code files for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://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 <http://www.packtpub.com>.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the on-screen 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/AI-Crash-Course>. 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: https://static.packt-cdn.com/downloads/9781838645359_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. For example; "To get these numbers you can add together the lists `nPosReward` and `nNegReward`."

A block of code is set as follows:

```
# Creating the dataset
X = np.zeros((N, d))
for i in range(N):
    for j in range(d):
        if np.random.rand() < conversionRates[j]:
            X[i][j] = 1
```

When we wish to draw your attention to a particular line in a code block, we have included the line numbers so that we can refer to them with precision:

```
80         self.last_state = new_state
81         self.last_action = new_action
82         self.last_reward = new_reward
83         return new_action
```

Any command-line input or output is written as follows:

```
conda install -c conda-forge keras
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. For example: "Select **System info** from the **Administration** panel."



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: Email feedback@packtpub.com, and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at questions@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, <http://www.packtpub.com/submit-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@packtpub.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 <http://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 packtpub.com.

1

Welcome to the Robot World

"We are truly living in the most exciting time to be alive!" These words, by the great tech entrepreneur Peter Diamandis, are even more true for people working in the **artificial intelligence (AI)** ecosystem. There is a reason why AI jobs are considered the sexiest jobs of the 21st century: besides being very well paid, AI is a fantastic topic to work on.

AI is taking a more and more important place in the world, and today we can find applications of it in almost all industries. This is not a temporary trend; AI is here to stay. As the top AI leader and influencer Andrew Ng said, AI is the new electricity. Just like the industrial revolution transformed lives and jobs in the 19th century, AI is about to do the same in this 21st century. Hence, the more you understand and know how to use it, the more opportunities will open up to you.

To give you some important figures, according to a study done by **PricewaterhouseCoopers (PwC)**, AI could contribute up to \$15.7 trillion to the global economy by 2030, which is more than the current output of China and India combined. So, you've definitely made a great choice to study this field. Welcome to the incredible world of Artificial Intelligence!

In this chapter, you will begin your AI journey with a top-level view of everything you'll learn from this book as you read and work through the chapters ahead with me. Then, I'll help you understand where learning AI can take you, by going through a variety of top industry applications for Artificial Intelligence.

Beginning the AI journey

Being a young AI scientist, I remember my first days in AI very well. This is important because this book is a crash course in AI. You don't need any prior knowledge of the field to work through the chapters.

In this book, I will explain the solid foundations of AI, while making sure to answer all the questions that I had back when I started in this field in detail. This means that everything will be explained step by step, and your learning process will follow a smooth path, supported by the relevant logic.

Having the right information at your fingertips is not enough to successfully break into the AI world. What you also need is energy, enthusiasm, and excitement. Even better, you need passion, and ideally obsession, about the subject. As an experienced tutor of online courses, I hope to pass on my knowledge and, most importantly, my passion.

In this book, you will go on a journey together with me, taking a path through a world of exciting AI applications, including many real-world case studies in the chapters. The applications will follow an increasing level of difficulty, from the simplest model in AI, to a much more advanced level.

For each of the AI applications, I will focus mostly on the intuition needed to understand them, and then, for those interested in the mathematics and pure theory behind the application, I will provide those as an option. The reason why I choose to focus on intuition rather than math is not only because I want to make this book easy to understand for everyone, but also because, in order to perform well in AI today, it is extremely important to have the right intuition. When you're solving a problem with AI, you have to figure out which model best fits your problem environment, and you can only do that when you have the proper intuition of how each AI model works.

Four different AI models

These AI models were chosen to be part of this book because they are used in a great variety of industry applications and can solve many different real-world problems. I'll just reveal their names here before we study them in depth across the book. The four AI models you will learn everything about in this book are the following:

1. Thompson Sampling
2. Q-learning
3. Deep Q-learning
4. Deep convolutional Q-learning

For each of these four models, we will follow the same three-step approach:

1. Get an intuitive understanding of how it works.
2. Get all the math behind the theory.
3. Implement the model from scratch in Python.

I have followed this structure many times with my students, and I can tell you that it works the best. The idea is simple: because you start with your intuition, you won't get overwhelmed by the math, but will instead understand it more easily. You'll also feel comfortable coding some models of which you both have an intuitive understanding and in-depth theoretical knowledge.

The models in practice

All the way through this book you'll find practical examples to learn from or implement yourself. Here's a list of the AI implementations you'll find in the chapters of this course, which start in *Chapter 3* after you get the tools you need for your AI journey in *Chapter 2*.

Fundamentals

Chapter 3, Python Fundamentals – Learn How to Code in Python, contains the Python coding fundamentals you'll need for this book. You can remind yourself, or learn from scratch, how to code in Python.

Chapter 4, AI Foundation Techniques, contains a pseudocode example to illustrate the five core principles of Artificial Intelligence.

Thompson Sampling

Chapter 5, Your First AI Model – Beware the Bandits!, contains introductory code to illustrate the theory behind the Thompson Sampling AI model.

Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street, contains a real-world implementation of the Thompson Sampling model, applied to online advertising.

Q-learning

Chapter 7, Welcome to Q-Learning, contains pseudocode to illustrate the theory of the Q-learning AI model.

Chapter 8, AI for Logistics – Robots in a Warehouse, contains a real-world implementation of the Q-learning model, applied to process automation and optimization.

Deep Q-learning

Chapter 9, Going Pro with Artificial Brains – Deep Q-Learning, contains introductory code to illustrate the theory behind Artificial Neural Networks.

Chapter 10, AI for Autonomous Vehicles – Build a Self-Driving Car, contains a real-world implementation of the deep Q-learning model, applied to self-driving cars.

Chapter 11, AI for Business – Minimize Costs with Deep Q-Learning, contains another real-world implementation of the deep Q-learning model, applied to energy and business.

Deep convolutional Q-learning

Chapter 12, Deep Convolution Q-Learning, contains introductory code to illustrate the implementation of a **Convolutional Neural Network (CNN)**.

Chapter 13, AI for Video Games – Become the Master at Snake, contains a real-world implementation of the deep convolutional Q-learning model applied to a game.

As you can see, every time you're introduced to a new model, you learn the intuition first, then the math, and then you move to an implementation of the model. So, why is learning how to implement these models worth your while?

Where can learning AI take you?

I'd like to motivate you by showing you that you made the right choice to learn AI. To do this, I'll take you on a tour of all the incredible applications AI can and will have in the 21st century. I have a vision of how AI can transform the world, and this vision is structured around 10 areas.

Energy

In 2016, Google used AI to reduce energy consumption in its data centers by more than 30%. If Google has done it for data centers, it could be done for an entire city. By building a smart AI platform using **Internet of Things (IoT)** technology, the consumption and distribution of energy can be optimized on a large scale.

Healthcare

AI has enormous promise for healthcare. It can already diagnose diseases, make prescriptions, and design new drug formulas. Combining all these skills into a smart healthcare platform will allow people to benefit from truly personalized medical care. This would be amazing for society. The challenges in achieving this are not only present in the technology, but also in getting access to anonymous patient data, which so far is protected by regulations.

Transport and logistics

Self-driving vehicles are becoming a reality. There is still a lot to achieve, but the technology is constantly improving. By building smart digital infrastructures, AI will help reduce the number of accidents and considerably reduce traffic. Also, self-driving delivery trucks and drones will speed up logistic processes, therefore boosting the economy; mostly through one of its bigger engines, the e-commerce industry.

Education

Today, we live in the era of Massive Open Online Courses. Anyone can learn anything online. This is great because the whole world can get access to an education; but it's definitely not enough. A significant improvement would be the personalization of education; everyone learns differently, and at different paces. Some, namely extroverts, will prefer the classroom, while others, introverts, will learn better at home. Some are more visual, while others are more auditory. Taking these and other factors into account, AI is a powerful technology that could deliver personalized training, optimizing everyone's learning curve.

Security

Computer vision has made tremendous technological progress. AI can now detect faces with a high level of accuracy. Not only that, the number of security cameras is increasing significantly. All this could be integrated into a global security platform to reduce crime, increase public safety, and disincentivize people from breaking the law. Besides this, AI and Machine Learning are powerful technologies already used in fraud detection and prevention.

Employment

AI can build powerful recommender systems. We already see platforms of digital recruitment, where AI matches the best candidates to jobs. This not only has a positive impact on the economy, but also on people's happiness, since work makes up more than half of a person's life.

Smart homes and robots

Smart homes, IoT, and connected objects are developing massively. Robots will assist people in their homes, allowing humans to focus on more important activities like their work or spending quality time with their family. They will also help elderly people to live in their home independently, or even allow them to stay active at work, for much longer.

Entertainment and happiness

One downside of technology today is that despite the fact people are so virtually connected, they feel more and more lonely. Loneliness is something we must fight against in this century, as it is very unhealthy for people. AI has a great role to play in this fight, since it is again a powerful recommender system, which can not only recommend relevant movies and songs to users, but also connect people through recommended activities based on their past experiences and common interests.

Through a global smart platform of entertainment, AI technology could help like-minded people to socialize and meet physically instead of virtually.

Another idea to fight loneliness is companion robots, which will be entering homes more and more over the next decade. One branch of AI in the Research and Development phase is emotion creation. This is the branch of AI that will allow robots to show emotions and empathy, and therefore interact more successfully with humans.

Environment

Using computer vision, machines could optimize waste sorting and redistribute the cycles of trash more efficiently. Combining pure AI models with IoT can optimize power and water consumption by individuals. Programs already exist on some platforms that allow people to track their consumption in real time, therefore collecting data. Integrating AI could minimize this consumption, or optimize the distribution cycles for beneficial reuse. Combined with traffic reduction and the development of autonomous vehicles, this will considerably reduce pollution, which will create a healthier environment.

Economy, business, and finance

AI is taking the business world by storm. Earlier, I mentioned the study done by PwC showing how AI could contribute up to \$15 trillion to the global economy in 2030 (<https://www.pwc.com/gx/en/issues/data-and-analytics/publications/artificial-intelligence-study.html>). But how can AI generate so much income? AI can bring significant added value to businesses in three different ways: process automation, profit optimization, and innovation. In my vision of an AI-driven economy, I see the majority of companies adopting at least one AI technology, or having an AI department. In finance, we can already see some jobs being replaced by robots. For example, the number of financial traders was significantly reduced after the development of trading robots that perform well on high-frequency trades.

As you can see, the robot world has a lot of great directions for you to take. AI is already in a dynamic place and it's picking up strong momentum as it moves forward. My professional purpose is to democratize AI and incentivize people to make a positive impact in this world thanks to AI— who knows, perhaps your purpose will be to work with AI for the good of humanity. I'm sure that at least one of these 10 applications resonates in you; if that's the case, work hard to become an AI master and you will have the chance to make a difference.

If you are ready to break into AI, or simply want to increase your knowledge, let's begin!

Summary

In this chapter, you began your AI journey and saw the vast land of opportunities that will open to you. Perhaps you can already think of which industry application might resonate the most in you, so you can become even more passionate about what you do with AI and understand why you're doing it. In the next chapter, you will uncover the AI toolkit you will use in this book.

2

Discover Your AI Toolkit

In the previous chapter, you began your AI journey. Before you continue it, you need your AI toolkit. This book is not just theory; it also contains an easy-to-use toolkit of all the AI models as Python files, ready to run thanks to the amazing Google Colaboratory platform that you will also be introduced to in this chapter.

To fill your AI toolkit, I've prepared a GitHub page containing all the AI implementations for you to download, and Google Colab links of the Python notebooks containing the implementations, all ready to execute via an easy plug and play process.

The GitHub page

You will find all the code for this book ready for you to download from the following GitHub page:

<https://github.com/PacktPublishing/AI-Crash-Course>

To download the code, you simply need to click the **Clone or download** button, and then **Download Zip**:

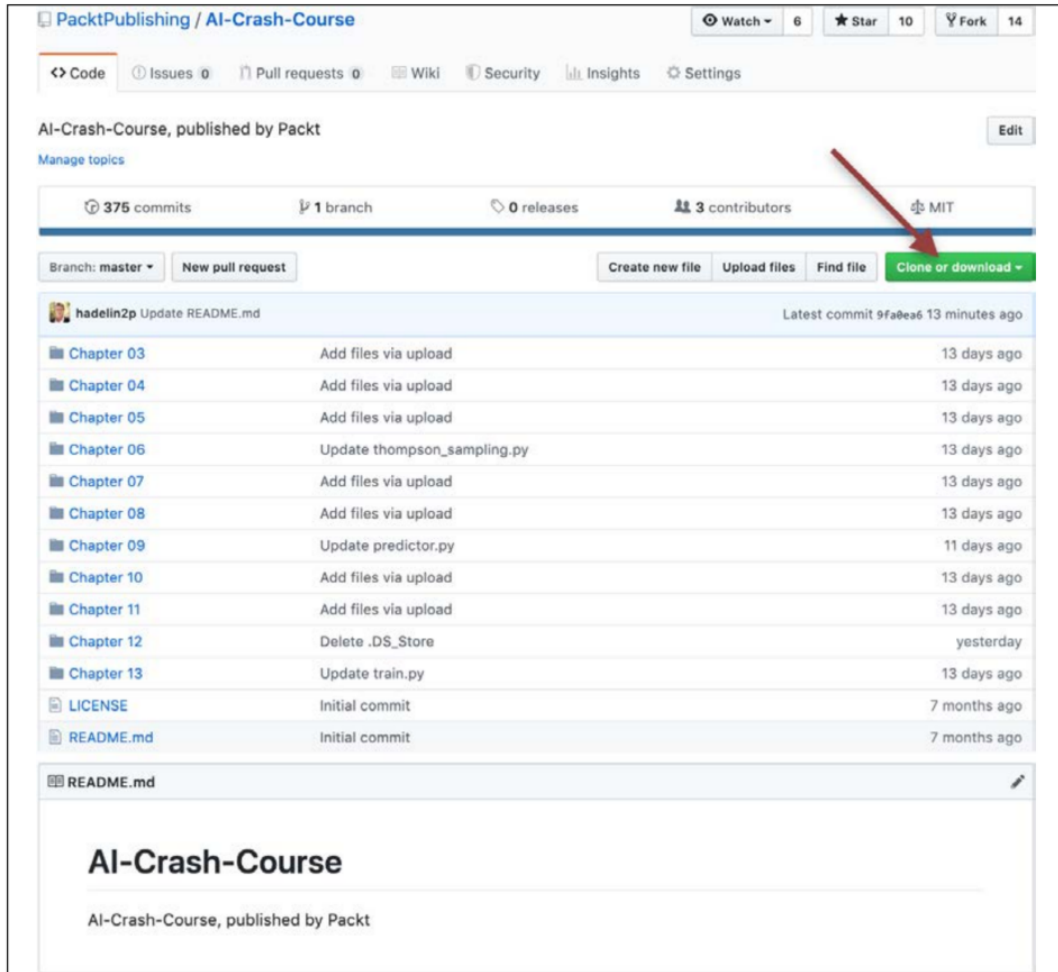


Figure 1: The GitHub repository

Then, once you've downloaded these codes, feel free to open them with your favorite Python **Integrated Development Environment (IDE)**, whether it's Jupyter Notebook, Spyder, a simple text editor, or even your terminal.

If you've never coded with Python before and have no idea of how to open the files with a Python editor, then no problem; I've prepared the best and simplest solution for you: Colaboratory (or Google Colab).

Colaboratory

Colaboratory is a free and open source environment for Python development that requires no setup and runs entirely on the cloud. It contains all the pre-installed packages required for your AI implementations so that they are ready to run with a simple plug and play process. By plug, I just mean to copy and paste the code inside a new Colab file (I'll explain how to open one next), and by play, I just mean to click on the play button (an example of that follows).

Here is the link to the main page of Colaboratory:

<https://colab.research.google.com/notebooks/welcome.ipynb>

You should get a page like this:

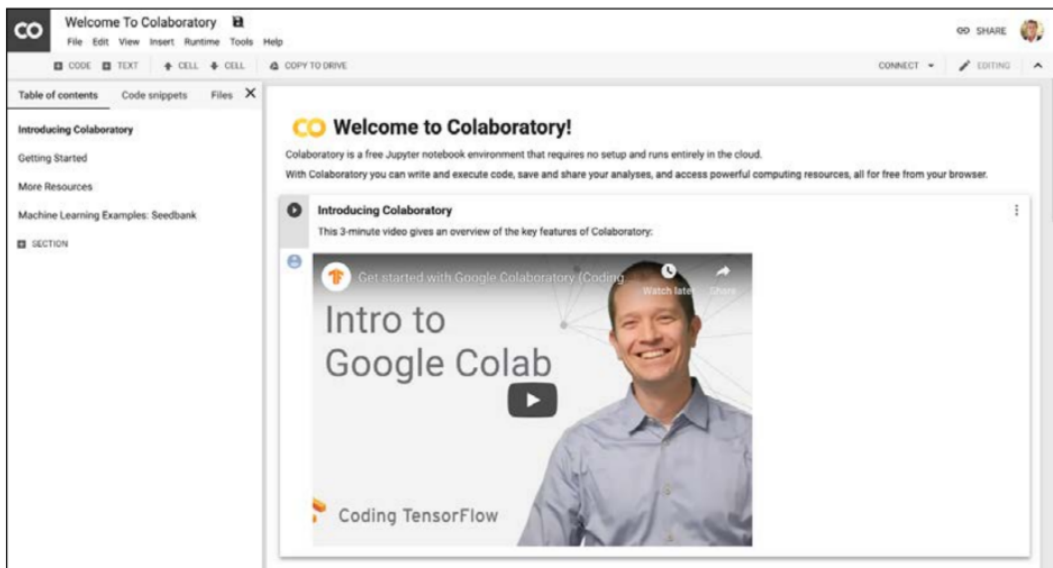


Figure 2: Colaboratory - main page

Click **File** in the upper left, and then click **New Python 3 notebook**:

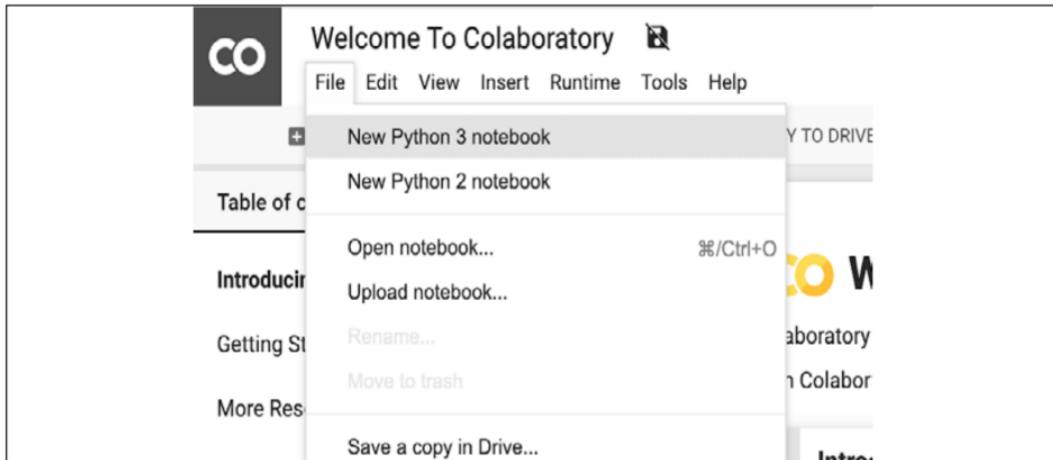


Figure 3: Colaboratory – opening a notebook

Then you will get this view. Paste your Python code inside the cell (red arrow). That's the "plug" part:

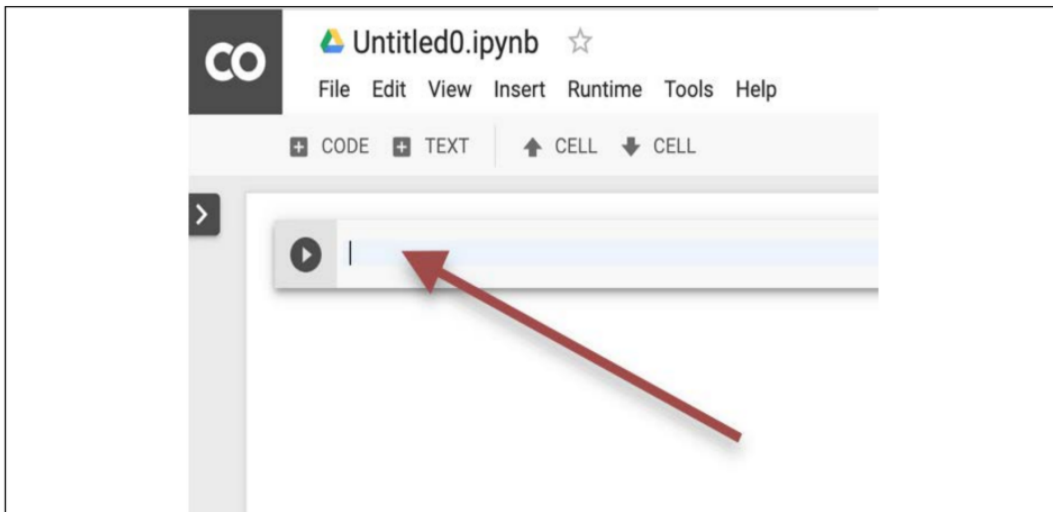


Figure 4: Colaboratory – the "plug" part

I recommend using separate Colaboratory notebooks for each model in this book.

Now let's see the "play" part. Open the Thompson Sampling model in the Chapter 06 folder, implemented inside the `thompson_sampling.py` file:

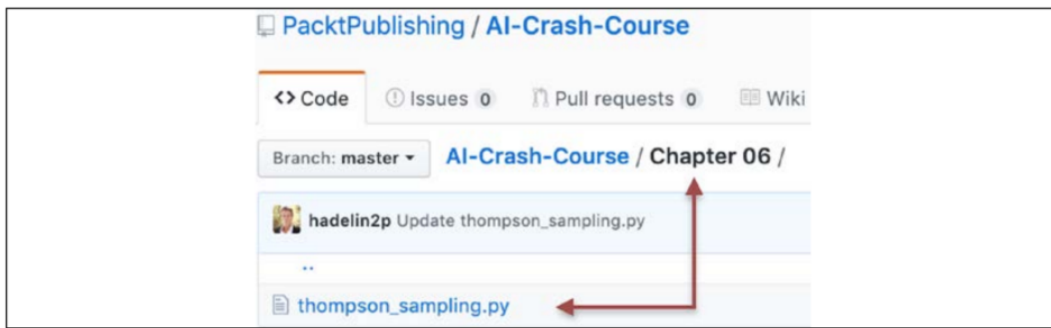


Figure 5: GitHub - opening Thompson Sampling

Copy the whole code from inside the Python file; don't worry about understanding the code (or the results) for now. It will all be explained, step by step, in *Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street*:

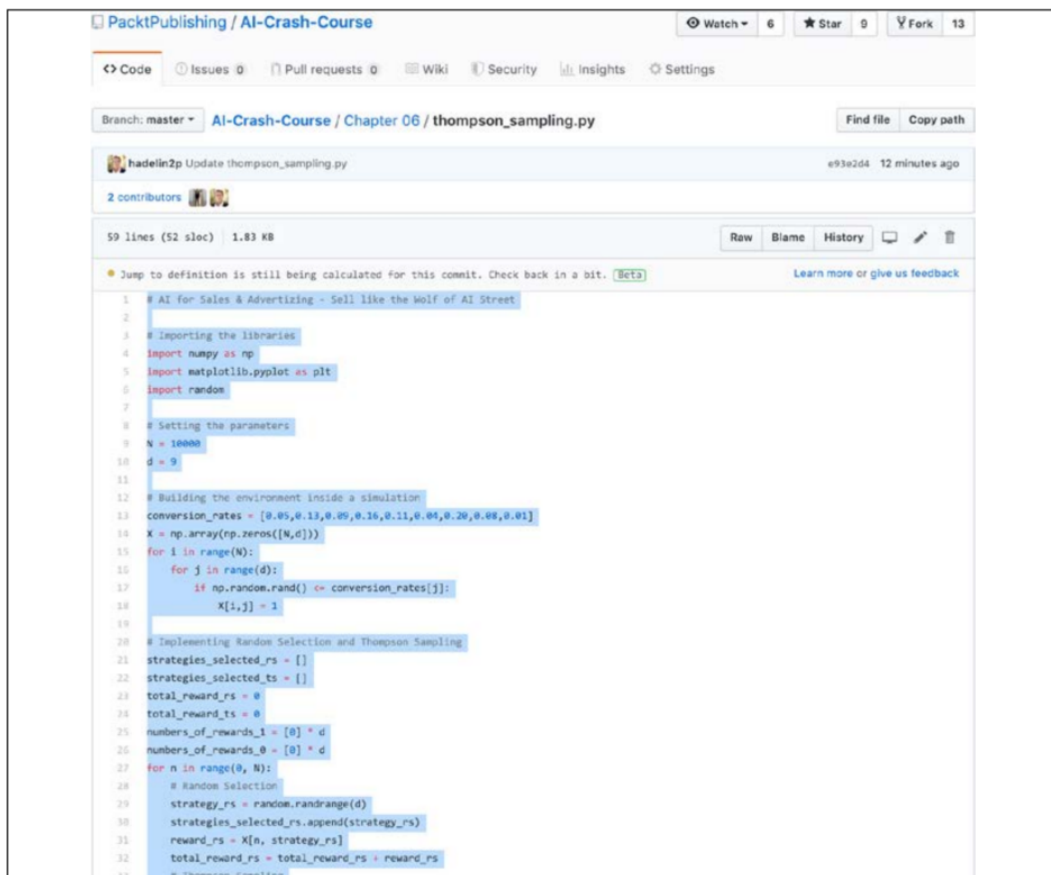
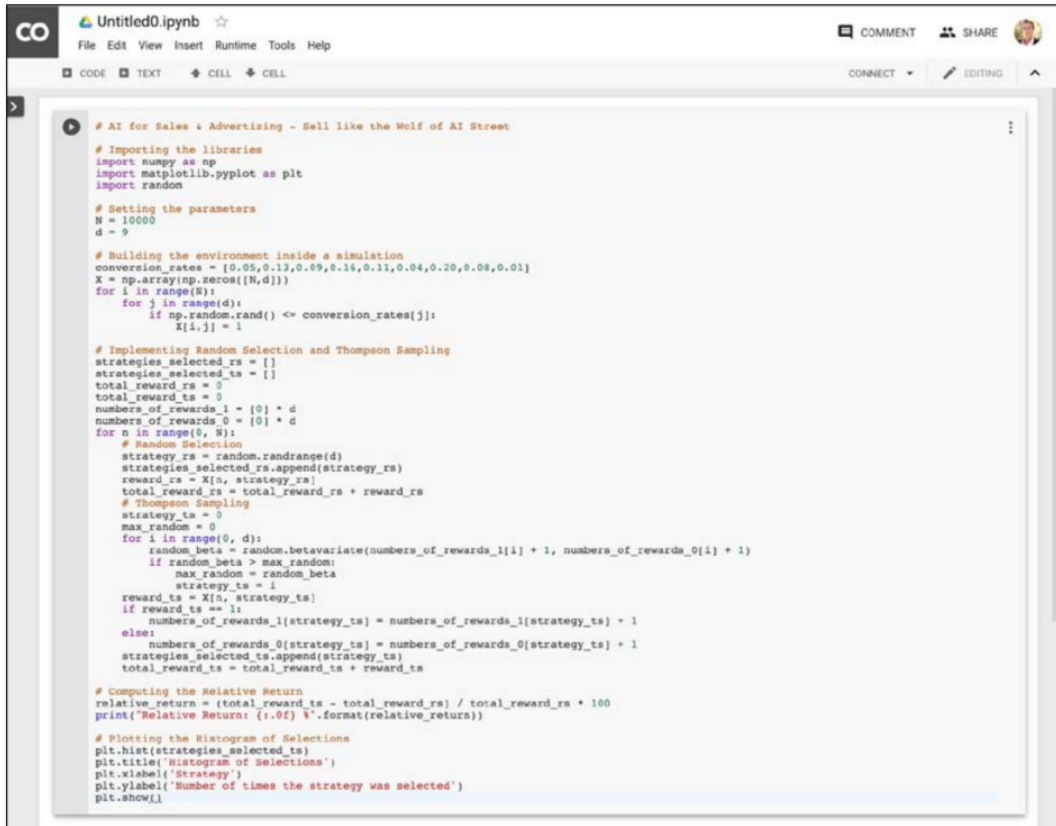


Figure 6: GitHub - copying Thompson Sampling

Next, paste it into Colaboratory (in the cell highlighted by the arrow in Figure 4). Then we get this:



```
# AI for Sales & Advertising - Sell like the Wolf of AI Street

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import random

# Setting the parameters
N = 10000
d = 2

# Building the environment inside a simulation
conversion_rates = [0.05, 0.13, 0.09, 0.16, 0.11, 0.04, 0.20, 0.08, 0.01]
x = np.array(np.zeros([N,d]))
for i in range(N):
    for j in range(d):
        if np.random.rand() <= conversion_rates[j]:
            x[i,j] = 1

# Implementing Random Selection and Thompson Sampling
strategies_selected_rs = []
total_reward_rs = 0
total_reward_ts = 0
numbers_of_rewards_1 = [0] * d
numbers_of_rewards_0 = [0] * d
for n in range(0, N):
    # Random Selection
    strategy_rs = random.randrange(d)
    strategies_selected_rs.append(strategy_rs)
    reward_rs = x[n, strategy_rs]
    total_reward_rs = total_reward_rs + reward_rs
    # Thompson Sampling
    strategy_ts = 0
    max_random = 0
    for i in range(0, d):
        random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, numbers_of_rewards_0[i] + 1)
        if random_beta > max_random:
            max_random = random_beta
            strategy_ts = i
    reward_ts = x[n, strategy_ts]
    if reward_ts == 1:
        numbers_of_rewards_1[strategy_ts] = numbers_of_rewards_1[strategy_ts] + 1
    else:
        numbers_of_rewards_0[strategy_ts] = numbers_of_rewards_0[strategy_ts] + 1
    strategies_selected_ts.append(strategy_ts)
    total_reward_ts = total_reward_ts + reward_ts

# Computing the Relative Return
relative_return = (total_reward_ts - total_reward_rs) / total_reward_rs * 100
print("Relative Returns: {:.0f} %".format(relative_return))

# Plotting the Histogram of Selections
plt.hist(strategies_selected_ts)
plt.title("Histogram of Selections")
plt.xlabel("Strategy")
plt.ylabel("Number of times the strategy was selected")
plt.show()
```

Figure 7: Pasting Thompson Sampling

And now we are ready for the "play" part! Just click the "play" button below:

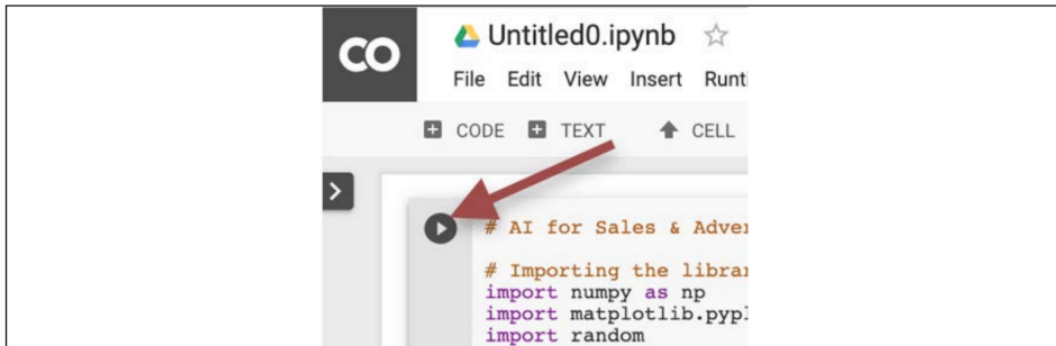


Figure 8: The "play" part

And the code will execute. Don't pay attention to the result now, as this will all be explained in *Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street*.

You are all set! You now have an AI toolkit that will enable you to follow along with every example in the book.

Before you begin your AI journey in earnest, you must make sure that you have the right basic coding knowledge. This is truly important before becoming a master at AI. If you have little or no experience with Python, make sure that you learn Python in *Chapter 3, Python Fundamentals – Learn How to Code in Python*, as a last preparation phase before you begin exploring the robot world.

Summary

In this chapter, you packed your luggage with our AI toolkit, which included not only the many AI models of this book, but also the very user-friendly Google Colaboratory environment. You saw how easy it was to plug and play our models from GitHub to Colaboratory. Now you just need coding skills to make you ready to begin the real journey. In the next chapter, you will have a chance to learn – or brush up on – your Python fundamentals.

3

Python Fundamentals – Learn How to Code in Python

This chapter is for people who have little or no experience with the Python programming language. If you already know how to use `for/while` loops, methods, and classes in Python, you can skip this chapter and you shouldn't have any problems later on.

If, however, you have not used Python before, or have only barely used it, I strongly recommend that you follow this guide. You'll learn how to code the elements of Python I mentioned in the previous paragraph, you'll fully understand the codes included in this book and you'll be able to code in Python on your own. I'll also give you some additional exercises, called "homework" throughout the chapter, which I strongly recommend that you do.

Before you begin, open your Python editor. I recommend using the Google Colab notebook, introduced to you as part of your AI Toolkit in the previous chapter. All the code, along with homework solutions, are provided on the GitHub page of this book in `Chapter 3` in their corresponding section folders. Inside them, you will find two Python files: one (named the same as the section) is the code used in this book, while the `homework.py` file is the solution to the exercise. Instructions for each homework exercise will be provided at the end of each section.

In this chapter, we'll cover the following topics:

- Displaying text
- Variables and operations
- Lists and arrays
- `if` statements and conditions
- `for` and `while` loops

- Functions
- Classes and objects

Especially if you're starting from scratch, cover each section in the order they're presented here, and remember to try your hand at the homework. Let's get started!

Displaying text

We'll begin with the most popular way of introducing any programming language; you'll learn how to display some text in the Python console. The console is a tool that's part of every Python editor, which shows the information we want or displays any errors that occurred (let's hope not to get any!).

The easiest way to show something in our console is to use the `print()` method, just like this:

```
# Displaying text
print('Hello world!')
```

The text above `print`, starting with `#`, is called a comment. Comments are excluded when executing code and are only visible to you.

After running this short code in Google Colab, you'll see this displayed:

```
Hello world!
```

In conclusion, just put what you want to display into the brackets of the `print` method – text surrounded by quotes, as in this example, or variables.

If you're curious about what variables are, that's great – you'll learn about them after this exercise.

Exercise

Using only one `print()` method, try to display two or more lines.

Hint: Try using the `\n` symbol.

The solution is provided in the `Chapter 03/Displaying Text/homework.py` file on the GitHub page.

Variables and operations

Variables are simply values that are allocated somewhere in the memory of our computer. They are similar to variables in mathematics. They can be anything: text, integers, or floats (a number with precision after the decimal point, such as 2.33).

To create a new variable, you only need to write this:

```
x = 2
```

In this case, we have named a variable `x` and set its value to 2.

As in mathematics, you can perform some operations on these variables. The most common operations are addition, subtraction, multiplication, and division. The way to write them in Python is like this:

```
x = x + 5    #x += 5
x = x - 3    #x -= 3
x = x * 2.5  #x *= 2.5
x = x / 3    #x /= 3
```

If you look at it for the first time, it doesn't make much sense—how can we write that `x = x + 5`?

In Python, and in most code, the "=" notation doesn't mean the two terms are equal. It means that we associate the new `x` value with the value of the old `x`, plus 5. It is crucial to understand that this is not an equation, but rather the creation of a new variable with the same name as the previous one.

You can also write these operations as shown on the right side, in the comments. You'll usually see them written in this way, since it's more space efficient.

You can also perform these operations on other variables, for example:

```
y = 3
x += y
print(x)
```

Here, we created a new variable `y` and set it to 3. Then, we added it to our existing `x`. Also, `x` will be displayed when you run this code.

So, what does `x` turn out to be after all these operations? If you run the code, you'll get this:

```
6.333333333333334
```

If you calculate these operations by hand, you will see that `x` does indeed equal `6.33`.

Exercise

Try to find a way to raise one number to the power of another.

Hint: Try using the `pow()` built-in function for Python.

The solution is provided in the `Chapter 03/Variables/homework.py` file on the GitHub page.

Lists and arrays

Lists and arrays can be represented with a table. Imagine a **one-dimensional (1D)** vector or a matrix, and you have just imagined a list/array.

Lists and arrays can contain data in them. Data can be anything – variables, other lists or arrays (these are called multi-dimensional lists/arrays), or objects of some classes (we will learn about them later).

For example, this is a 1D list/array containing integers:

	3	
	4	
	1	
	6	
	7	
	5	

And this is an example of a **two-dimensional (2D)** list/array, also containing integers:

	2	9	-5	
	-1	0	4	
	3	1	2	

In order to create a 2D list, you have to create a list of lists. Creating a list is very simple, just like this:

```
L1 = list()
L2 = []

L3 = [3,4,1,6,7,5]
L4 = [[2, 9, -5], [-1, 0, 4], [3, 1, 2]]
```

Here we create four lists: L1, L2, L3 and L4. The first two lists are empty – they have zero elements. The two subsequent lists have some predefined values in them. L3 is a one-dimensional list, same as the one in the first image. L4 is a two-dimensional list, the same as in the second image. As you can see, L4 actually consists of three smaller 1D lists.

Whenever I mention an array, I usually mean a "NumPy" array. NumPy is a Python library (a library is a collection of pre-coded programs that allows you to perform many actions without writing your code from scratch), widely used for list/array operations. You can think of a NumPy array as a special kind of list, with lots of additional functions.

To create a NumPy array, you have to specify a size and use an initialization method. Here's an example:

```
import numpy as np
nparray = np.zeros((5,5))
```

In the first line, we import the NumPy library (as you can see, to import a library, you need to write `import`) and by using `as`, we give NumPy the abbreviation `np` to make it easier to use. Then, we create a new array that we call `nparray`, which is a 2D array of size 5 x 5, full of zeros. The initialization method is the part after the `.`; in this case, we initialize this array as full of zeros, by using the function `zeros`.

In order to get access to the values in a list or array, you need to give the index of this value. For example, if you wanted to change the first element in the `L3` list, you would have to get its index. In Python, indexes start at 0, so you would need to write `L3[0]`. In fact, you can write `print(L3[0])` and execute it, and you will see that, as you might hope, the number 3 will be displayed.

Accessing a single value in a multi-dimensional list/array requires you to input as many indexes as there are dimensions. For example, to get 0 from our `L4` list, we would have to write `L4[1][1]`. `L4[1]` would return the entire second row, which is a list.

Exercise

Try to find the mean of all the numbers in the `L4` list. There are multiple solutions.

Hint: The simplest solution makes use of the NumPy library. Check out some of its functions here: <https://docs.scipy.org/doc/numpy/reference/>

The solution is provided in the `Chapter 03/Lists and Arrays/homework.py` file on the GitHub page.

if statements and conditions

Now I would like to introduce you to a very useful tool in programming – `if` conditions!

They are widely used to check whether a statement is true or not. If the given statement is true, then some instructions for our code are followed.

I'll present this subject to you with some simple code that will tell us whether a number is positive, negative, or equal to 0. The code's very short, so I'll show you all of it at once:

```
a = 5
if a > 0:
    print('a is greater than 0')
elif a == 0:
    print('a is equal to 0')
else:
    print('a is lower than 0')
```

In the first line, we introduce a new variable called `a` and we give it a value of 5. This is the variable whose value we are going to check.

In the next line we check if this variable is greater than 0. We do this by using an `if` condition. If `a` is greater than 0, then we follow the instructions written in the indented block; in this case, it is only displaying the message `a is greater than 0`.

Then, if the first condition fails, that is, if `a` is lower than or equal to 0, we go to the next condition, which is introduced with `elif` (which is short for `else if`). This statement will check whether `a` is equal to zero or not. If it is, we follow the indented instruction, which will display a message displaying: `a is equal to 0`.

The final condition is introduced via `else`. Instructions included in an `else` condition will always be followed when all other conditions fail. In this case, failing both conditions would mean that `a < 0`, and therefore we would display `a is lower than 0`.

It's easy to predict what our code will return. It will be the first instruction, `print('a is greater than 0')`. And, in fact, once you run this code, this is what you will get:

```
a is greater than 0
```

In brief, `if` is used to introduce statement checking and the first condition, `elif` is used to check as many further conditions as we want, and `else` is a true statement when all other statements fail.

It's also important to know that once one condition is true, no other conditions are checked. So, in this case, once we enter the first condition and we see that it is true, we no longer check other statements. If you would like to check other conditions, you would need to replace the `elif` and `else` statements with new `if` statements. A new `if` always checks a new condition; therefore, a condition included in an `if` is always checked.

Exercise

Build a condition that will check if a number is divisible by 3 or not.

Hint: You can use a mathematical expression called modulo, which when used, returns the remainder from the division between two numbers. In Python, modulo is represented by `%`. For example:

```
5 % 3 = 2
```

```
71 % 5 = 1
```

The solution is provided in the `Chapter 03/If Statements/homework.py` file on the GitHub page.

for and while loops

You can think of a loop as continuously repeating the same instructions over and over until some condition is satisfied that breaks this loop. For example, the previous code was not a loop; since it was only executed once, we only checked a once.

There are two types of loops in Python:

- `for` loops
- `while` loops

`for` loops have a specific number of iterations. You can think of an iteration as a single execution of the specific instructions included in the `for` loop. The number of iterations tells the program how many times the instruction inside the loop should be performed.

So, how do you create a `for` loop? Simply, just like this:

```
for i in range(1, 20):  
    print(i)
```

We initialize this loop by writing `for` to specify the type of loop. Then we create a variable `i`, that will be associated with integer values from `range (1, 20)`. This means that when we enter this loop for the first time, `i` will be equal to 1, the second time it will be equal to 2, and so on, all the way to 19. Why 19? That's because in Python, upper bounds are excluded, so at the final iteration `i` will be equal to 19. As for our instruction, in this case it's just showing the current `i` in our console by using the `print()` method. It's also important to understand that the main code does not progress until the `for` loop is finished.

This is what we get once we execute our code:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

```
12
13
14
15
16
17
18
19
```

You can see that our code displayed every integer higher than 0 and lower than 20.

You can also use a `for` loop to iterate through elements of a list, in the following way:

```
L3 = [3,4,1,6,7,5]
for element in L3:
    print(element)
```

Here we come back to our `L3` 1D list. This code iterates through every element in the `L3` list and displays it. If you run it, you will see all the elements of this array from 3 to 5.

`while` loops, on the other hand, need a condition to stop. They go on as long as the given condition is satisfied. Take this `while` loop, for example:

```
stop = False
i = 0
while stop == False: # alternatively it can be "while not stop:"
    i += 1
    print(i)
    if i >= 19:
        stop = True
```

Here, we create a new variable called `stop`. This type of variable is called a `bool`, since it can be assigned only two values - `True` or `False`. Then, we create a variable called `i` that we'll use to count how many times our `while` loop is executed. Next, we create a `while` loop that will go on as long as the variable `stop` is `False`; only once `stop` is changed to `True` will the loop stop.

In the loop, we increase `i` by 1, display it, and check if it is greater or equal to 19. If it is greater or equal to 19, we change `stop` to `True`; and as soon as we change `stop` to `True`, the loop will break!

After executing this code, you will see the exact same output as in the `for` loop example, that is:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

It's also very important to know that you can stack `for` and `while` loops inside each other. For example, to display all the elements from the 2D list `L4` we created previously, one after another, you would have to make one `for` loop that iterates through every row, and then another `for` loop (inside the previous one) that iterates through every value in this row. Something like this:

```
L4 = [[2, 9, -5], [-1, 0, 4], [3, 1, 2]]
for row in L4:
    for element in row:
        print(element)
```

And running this yields the following output:

```
2
9
-5
-1
```

```
0
4
3
1
2
```

This matches the L4 list.

In conclusion, `for` and `while` loops let us perform repetitive tasks with ease. `for` loops always work on a predefined range; you know exactly when they will stop. `while` loops work on an undefined range; just by looking at their `stop` condition, you may not be able to judge how many iterations will happen. `while` loops work as long as their particular condition is satisfied.

Exercise

Build both `for` and `while` loops that can calculate the factorial of a positive integer variable.

Hint: Factorial is a mathematical function that returns the product of all positive integers lower or equal to the argument of this function. This is the equation:

$$f(n) = n * (n - 1) * (n - 2) * ... * 1$$

Where:

- $f(n)$ - the factorial function
- n - the integer in question, the factorial of which we are searching for

This function is represented by `!` in mathematics, for example:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

$$4! = 4 * 3 * 2 * 1 = 24$$

The solution is provided in the `Chapter 03/For and While Loops/homework.py` file on the GitHub page.

Functions

Functions are incredibly useful when you want to increase code readability. You can think of them as blocks of code outside the main flow of code. Functions are executed once they are called in the main code.

You write a function like this:

```
def division(a, b):
    result = a / b
    return result

d = division(3, 5)
print(d)
```

The first three lines are a newly created function called `division`, and the last two lines are part of the main code.

You can create a function by writing `def` and then writing the function's name. After the name, you put brackets and within them write the arguments of the function; these are some variables that you will be able to use inside of your function and are a part of the connection between the main code and the function. In this case, our function takes two arguments: `a` and `b`.

Then, once we enter our function, what we do is calculate `a` divided by `b` and call this `division` `result`. Then, in the last line of our function, we say `return` so that when we call this function in code, it will return a value. In this case, the returned value is `result`.

Next, we go back to our main code and call our function. We do that by writing `division` and then in the brackets we input two numbers that we would like to divide. Remember, the `division` function returns a `result` of this division; therefore, we create a variable, `d`, that will hold this returned value. In the last line, we simply display `d` to see whether this code really works. If you run it, you'll get the output:

```
0.6
```

As you can confirm by hand, 3 divided by 5 is indeed 0.6; you can test it on other numbers as well.

In real-world code, functions can be much longer, and sometimes even call other functions. You will see them used a lot, even in the other chapters of this book. They also increase code readability, as you will see later; the code I've provided would be impossible to understand without functions.

Exercise

Build a function to calculate the distance between two points on an x,y plane: one with coordinates x_1 and y_1 , and the other with coordinates x_2 and y_2 .

Hint: You can use the following formula:

$$distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

The solution is provided in the `Chapter 03/Functions/homework.py` file on the GitHub page.

Classes and objects

Classes, like functions, are another part of code that sits outside of the main code, executed only when called in the main flow of code. Objects are instances of a corresponding class, existing within the main flow of our code. To better understand it, think of a class as a plan of something, for example, a plan of a car. It contains information on how certain components look and work with each other. A class in Python is a general plan of something.

You can think of objects as real-life constructions based on the plan. For example, a real, working, and self-driving car would be an example of an object. You create a plan of a car (which is a class) and then you build a car based on this plan (which is an object). And of course, when you have a plan of something, you can create as many copies as you want; for example, you can run a production line to produce cars.

To give you more insight into classes, we will create a simple bot. We begin with writing a class, like this:

```
class Bot():

    def __init__(self, posx, posy):
        self.posx = posx
        self.posy = posy

    def move(self, speedx, speedy):
        self.posx += speedx
        self.posy += speedy
```

We write `class` to specify that we are creating a new class, which we name `Bot`. Then, a very important step is to write an `__init__()` method, which is a necessity when creating a class. This function is called automatically whenever an object of this class is created in the main flow of the code.

All functions in a class need to take `self` as one argument. So, what is `self`? This parameter specifies that this function and its variables, whose names are preceded by `self`, are a part of this class. We will be able to call the `self` variables once we have an object of this class. Our bot's `__init__()` method also takes two arguments, `posx` and `posy`, which will be the initial position of our bot.

We have also created a method that will move our bot, by increasing or decreasing its `posx` and `posy`. A method is a function tucked inside a class. You can think of it as an instruction on how something has to work when we have a plan. For example, going back to the example of a car, a method could define the way our engine or gearbox works.

Now, you can create an object of this class. Remember, this will be a real-life object, constructed on the basis of a plan (`class`). Before, the class was predefined and didn't work along with your code. After you create an object, the class becomes an integral part of your main code. We can achieve this by doing:

```
bot = Bot(3, 4)
```

This will create a new object of class `Bot`; we called this object `bot`. We also need to specify the two arguments that the `__init__()` method of class `Bot` takes, which are `posx` and `posy`. This isn't optional; when creating an object, you always have to specify all the arguments given in the `__init__()` method.

Now, in the main code, you can move the bot and display its new position, like this:

```
bot.move(2, -1)
print(bot.posx, bot.posy)
```

In the first line, we use the `move` method from our `Bot` class. As you can see in its definition, `move` takes two arguments. These two arguments specify, respectively, by how much we will increase `posx` and `posy`. Then we just display the new `posx` and `posy`. This is where `self` comes into action; if the variables `posx` and `posy` were not preceded by `self` in our `Bot` class, we wouldn't have access to them via the method. Running this code gives us this result:

```
5 3
```

As you can see from the result, our bot moved two units forward on the x axis and one unit backward on the y axis. Remember, `posx` was set to 3 initially and has now been increased by 2 using the `move` method from the `Bot` class; `posy` was set to 4 initially and has now been decreased by 1, with the use of the same `move` method.

One great advantage of taking the time to code a `Bot` class is that now we are able to create as many bots as we want without making our code any longer. Simply put, objects are copies of a class and we can create as many of them as we want.

In conclusion, you can think of a class as a collection of predefined instructions and closed in methods, and you can think of an object as an instance of this class that is accessible in our code and that runs along with it.

Exercise

Your final challenge will be to build a very simple car class. As arguments, a car object should take the maximum velocity at which the car can move (unit in m/s), as well as the acceleration at which the car is accelerating (unit in m/s²). I also challenge you to build a method that will calculate the time it will take for the car to accelerate from the current speed to the maximum speed, knowing the acceleration (use the current speed as the argument of this method).

Hint: To calculate the time required, you can use the following equation:

$$t = \frac{(V_{max} - V_{current})}{a}$$

Where:

- t - time required to achieve the top speed
- V_{max} - maximum speed
- $V_{current}$ - current speed
- a - acceleration

The solution is provided in the `Chapter 03/Classes/homework.py` file on the GitHub page.

Summary

In this chapter, we covered the Python fundamentals that you'll need to keep up with the code presented in this book, from sending a simple text display to the console to writing your very first class in Python. You've now got all the skills you need to continue on your AI journey; in *Chapter 4, AI Foundation Techniques*, we will begin to study the foundational techniques of AI.

4

AI Foundation Techniques

In this chapter, you'll begin your study of AI theory in earnest. You'll start with an introduction to a major branch of AI, called Reinforcement Learning, and the five principles that underpin every Reinforcement Learning model. Those principles will give you the theoretical understanding to make sense of every forthcoming AI model in this book.

What is Reinforcement Learning?

When people refer to AI today, some of them think of Machine Learning, while others think of Reinforcement Learning. I fall into the second category. I always saw Machine Learning as statistical models that have the ability to learn some correlations, from which they make predictions without being explicitly programmed.

While this is, in some way, a form of AI, Machine Learning does not include the process of taking actions and interacting with an environment like we humans do. Indeed, as intelligent human beings, what we constantly keep doing is the following:

1. We observe some input, whether it's what we see with our eyes, what we hear with our ears, or what we remember in our memory.
2. These inputs are then processed in our brain.
3. Eventually, we make decisions and take actions.

This process of interacting with an environment is what we are trying to reproduce in terms of Artificial Intelligence. And to that extent, the branch of AI that works on this is Reinforcement Learning. This is the closest match to the way we think; the most advanced form of Artificial Intelligence, if we see AI as the science that tries to mimic (or surpass) human intelligence.

Reinforcement Learning also has the most impressive results in business applications of AI. For example, Alibaba leveraged Reinforcement Learning to increase its ROI in online advertising by 240% without increasing their advertising budget (see <https://arxiv.org/pdf/1802.09756.pdf>, page 9, Table 1 last row (DCMAB)). We'll tackle the same industry application in this book!

The five principles of Reinforcement Learning

Let's begin building the first pillars of your intuition into how Reinforcement Learning works. These are the fundamental principles of Reinforcement Learning, which will get you started with the right, solid basics in AI.

Here are the five principles:

1. **Principle #1:** The input and output system
2. **Principle #2:** The reward
3. **Principle #3:** The AI environment
4. **Principle #4:** The Markov decision process
5. **Principle #5:** Training and inference

In the following sections, you can read about each one in turn.

Principle #1 – The input and output system

The first step is to understand that today, all AI models are based on the common principle of inputs and outputs. Every single form of Artificial Intelligence, including Machine Learning models, ChatBots, recommender systems, robots, and of course Reinforcement Learning models, will take something as input, and will return another thing as output.

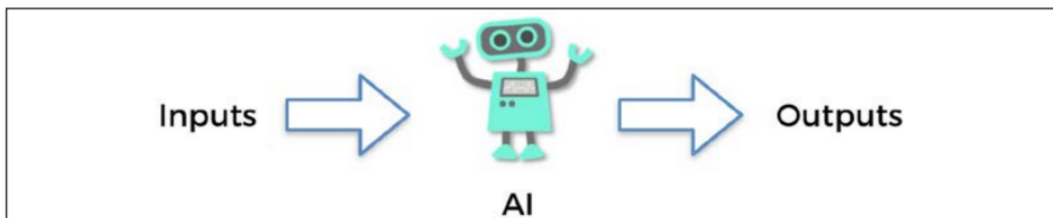


Figure 1: The input and output system

In Reinforcement Learning, these inputs and outputs have a specific name: the input is called the state, or input state. The output is the action performed by the AI. And in the middle, we have nothing other than a function that takes a state as input and returns an action as output. That function is called a policy. Remember the name, "policy," because you will often see it in AI literature.

As an example, consider a self-driving car. Try to imagine what the input and output would be in that case.

The input would be what the embedded computer vision system sees, and the output would be the next move of the car: accelerate, slow down, turn left, turn right, or brake. Note that the output at any time (t) could very well be several actions performed at the same time. For instance, the self-driving car can accelerate while at the same time turning left. In the same way, the input at each time (t) can be composed of several elements: mainly the image observed by the computer vision system, but also some parameters of the car such as the current speed, the amount of gas remaining in the tank, and so on.

That's the very first important principle in Artificial Intelligence: it is an intelligent system (a policy) that takes some elements as input, does its magic in the middle, and returns some actions to perform as output. Remember that the inputs are also called the **states**.

The next important principle is the reward.

Principle #2 – The reward

Every AI has its performance measured by a reward system. There's nothing confusing about this; the reward is simply a metric that will tell the AI how well it does over time.

The simplest example is a binary reward: 0 or 1. Imagine an AI that has to guess an outcome. If the guess is right, the reward will be 1, and if the guess is wrong, the reward will be 0. This could very well be the reward system defined for an AI; it really can be as simple as that!

A reward doesn't have to be binary, however. It can be continuous. Consider the famous game of *Breakout*:



Figure 2: The Breakout game

Imagine an AI playing this game. Try to work out what the reward would be in that case. It could simply be the score; more precisely, the score would be the accumulated reward over time in one game, and the rewards could be defined as the derivative of that score.

This is one of the many ways we could define a reward system for that game. Different AIs will have different reward structures; we will build five rewards systems for five different real-world applications in this book.

With that in mind, remember this as well: the ultimate goal of the AI will always be to maximize the accumulated reward over time.

Those are the first two basic, but fundamental, principles of Artificial Intelligence as it exists today; the input and output system, and the reward. The next thing to consider is the AI environment.

Principle #3 – The AI environment

The third principle is what we call an "AI environment." It is a very simple framework where you define three things at each time (t):

- The input (the state)
- The output (the action)
- The reward (the performance metric)

For each and every single AI based on Reinforcement Learning that is built today, we always define an environment composed of the preceding elements. It is, however, important to understand that there are more than these three elements in a given AI environment.

For example, if you are building an AI to beat a car racing game, the environment will also contain the map and the gameplay of that game. Or, in the example of a self-driving car, the environment will also contain all the roads along which the AI is driving and the objects that surround those roads. But what you will always find in common when building any AI, are the three elements of state, action, and reward. The next principle, the Markov decision process, covers how they work in practice.

Principle #4 – The Markov decision process

The Markov decision process, or MDP, is simply a process that models how the AI interacts with the environment over time. The process starts at $t = 0$, and then, at each next iteration, meaning at $t = 1, t = 2, \dots, t = n$ units of time (where the unit can be anything, for example, 1 second), the AI follows the same format of transition:

1. The AI observes the current state, s_t .
2. The AI performs the action, a_t .
3. The AI receives the reward, $r_t = R(s_t, a_t)$.
4. The AI enters the following state, s_{t+1} .

The goal of the AI is always the same in Reinforcement Learning: it is to maximize the accumulated rewards over time, that is, the sum of all the $r_t = R(s_t, a_t)$ received at each transition.

The following graphic will help you visualize and remember an MDP better, the basis of Reinforcement Learning models:

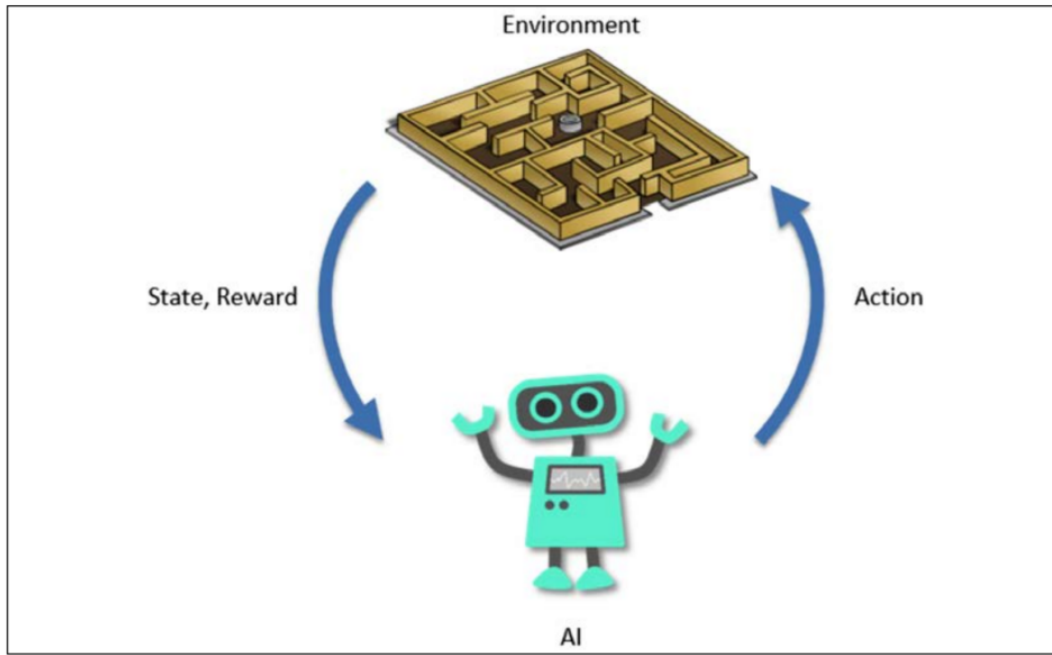


Figure 3: The Markov decision process

Now four essential pillars are already shaping your intuition of AI. Adding a last important one completes the foundation of your understanding of AI. The last principle is training and inference; in training, the AI learns, and in inference, it predicts.

Principle #5 – Training and inference

The final principle you have to understand is the difference between training and inference. When building an AI, there is a time for the training mode, and a separate time for inference mode. I'll explain what that means starting with the training mode.

Training mode

Now you understand, from the three first principles, that the very first step of building an AI is to build an environment in which the input states, the output actions, and a system of rewards are clearly defined. From the fourth principle, you also understand that inside this environment we will build an AI to interact with it, trying to maximize the total reward accumulated over time.

To put it simply, there will be a preliminary (and long) period of time during which the AI will be trained to do that. That period of time is called the training; we can also say that the AI is in training mode. During that time, the AI tries to accomplish a certain goal over and over again until it succeeds. After each attempt, the parameters of the AI model are modified in order to do better at the next attempt.

For example, let's say you're building a self-driving car and you want it to go from point *A* to point *B*. Let's also imagine that there are some obstacles that you want your self-driving car to avoid. Here is how the training process happens:

1. You choose an AI model, which can be Thompson Sampling (*Chapters 5 and 6*), Q-learning (*Chapters 7 and 8*), deep Q-learning (*Chapters 9, 10, and 11*) or even deep convolutional Q-learning (*Chapters 12 and 13*).
2. You initialize the parameters of the model.
3. Your AI tries to go from *A* to *B* (by observing the states and performing its actions). During this first attempt, the closer it gets to *B*, the higher reward you give to the AI. If it fails reaching *B* or hits an obstacle, you give the AI a very bad reward. If it manages to reach *B* without hitting any obstacle, you give the AI an extremely good reward. It's just like you would train a dog to sit: you give the dog a treat or say "good boy" (positive reward) if the dog sits. And you give the dog whatever small punishment you need to if the dog disobeys (negative reward). That process is training, and it works the same way in Reinforcement Learning.
4. At the end of the attempt (also called an episode), you modify the parameters of the model in order to do better next time. The parameters are modified intelligently, either iteratively through equations (Q-Learning), or by using Machine Learning and Deep Learning techniques such as stochastic gradient descent or backpropagation. All these techniques will be covered in this book.
5. You repeat steps 3 and 4 again, and again, until you reach the desired performance; that is, until you have your fully non-dangerous autonomous car!

So, that's training. Now, how about inference?

Inference mode

Inference mode simply comes after your AI is fully trained and ready to perform well. It will simply consist of interacting with the environment by performing the actions to accomplish the goal the AI was trained to achieve before in training mode. In inference mode, no parameters are modified at the end of each episode.

For example, imagine you have an AI company that builds customized AI solutions for businesses, and one of your clients asked you to build an AI to optimize the flows in a smart grid. First, you'd enter an R&D phase during which you would train your AI to optimize these flows (training mode), and as soon as you reached a good level of performance, you'd deliver your AI to your client and go into production. Your AI would regulate the flows in the smart grid only by observing the current states of the grid and performing the actions it has been trained to do. That's inference mode.

Sometimes, the environment is subject to change, in which case you have to alternate fast between training and inference modes so that your AI can adapt to the new changes in the environment. An even better solution is to train your AI model every day, and go into inference mode with the most recently trained model.

That was the last fundamental principle common to every AI. Congratulations – now you already have a solid basic understanding of Artificial Intelligence! Since you have that, you are ready to tackle your very first AI model in the next chapter: a simple yet very powerful one, still widely used today in business and marketing, to solve a problem that has the delightful name of the multi-armed bandit problem.

Summary

In this chapter, you learned the five fundamental principles of Artificial Intelligence from a Reinforcement Learning perspective. Firstly, an AI is a system that takes an observation (values, images, or any data) as input, and returns an action to perform as output (principle #1). Then, there is a reward system that helps it measure its performance. The AI will learn through trial and error based on the reward it gets over time (principle #2). The input (state), the output (action), and the reward system define the AI environment (principle #3). The AI interacts with this environment through the Markov decision process (principle #4). Finally, in training mode, the AI learns how to maximize its total reward by updating its parameters through the iterations, and in inference mode, the AI simply performs its actions over full episodes without updating any of its parameters – that is to say, without learning (principle #5).

In the next chapter, you will learn about Thompson Sampling, a simple Reinforcement Learning model, and use it to solve the multi-armed bandit problem.

5

Your First AI Model – Beware the Bandits!

In this chapter, you'll get to grips with your very first AI model! You're going to make a model that will solve the very well-known multi-armed bandit problem. This is a classic problem in AI, and it's also widely encountered in many real-world business problems.

The multi-armed bandit problem

Imagine you are in Las Vegas, in your favorite casino. You are in a room containing five slot machines. For each of them the game is the same: you bet a certain amount of money, say 1 dollar, you pull the arm, and then the machine will either take your money, or give you twice your money back. Remember the rewards we talked about in the previous chapter? Let's say that if the machine takes your money, your reward is -1, and if the machine returns you twice your money, your reward is +1.

As you can see, you're already starting to define an AI environment, which I'll remind you is absolutely fundamental when solving a problem with AI. So far, the AI isn't there, but it will come soon. You always start by defining the environment.

You've defined the rewards; you'll define the states (inputs) and actions (outputs) later. Now, still in the process of defining the environment, let's say that you know, somehow, that one of these machines has a higher probability of giving you a +1 reward than the others when you pull its arm. It doesn't matter how you know this info, but it must be part of the problem assumptions. Rest assured, this assumption is always naturally verified in the real-world business problems mentioned above where the multi-armed bandit problem can be applied.

Your goal, as in any AI environment, is to obtain the highest accumulated reward during your time of play. Let's say you are going to bet 1,000 dollars in total, meaning that you are going to bet 1 dollar, 1,000 times, each time by pulling the arm of any of these five slot machines. The question is:

What should be your strategy, so that after having played 1,000 times, you get the maximum amount of money to take home with you?

The first step of your strategy must be to figure out, in the minimum number of plays, which of these five slot machines has the highest chance of giving you a 1 reward. In other words, you have to quickly figure out the slot machine with the highest success rate. Then, as soon as you figure it out, you simply need to keep playing on that most successful slot machine.

Finding the most successful slot machine is not hard; one simple strategy could be to play 100 times on each of these five slot machines and then, at the end, look at which of them gave you more money. Statistically, this gives you a good chance of finding that most generous slot machine.

All the challenge is in "quickly". The hardest part is to find the best slot machine **in a minimum number of trials**. This is where your first AI model comes into play.

The Thompson Sampling model

You're going to build this model straight away. Right now, you'll build a simple implementation of this method, and later you will be shown the theory behind it. Let's get right into it!

As we defined previously, our problem is trying to find the best slot machine with the highest winning chance out of many. A not-so-optimal solution would be to play 100 rounds on each of our slot machines and see which one has the highest winning rate. A better solution is a method called Thompson Sampling.

I won't go too deeply into the theory behind it; we'll cover that later. For now, it is enough to say that Thompson Sampling uses a distribution function (distributions will be explained further in this chapter), called Beta, that takes two arguments. For simplicity's sake, let's say that the higher the first argument is, the better our slot machine is, and the higher the second argument is, the worse our slot machine is.

Therefore, we can define this function as:

$$x = \beta(a, b)$$

where:

- x - a random choice from our Beta distribution
- β - our Beta function
- a - the first argument
- b - the second argument

Don't worry if you don't understand this entirely quite yet; you'll read all about it later.

Coding the model

Let's start coding our solution. All this code is also available on the GitHub page of this book in the `Chapter 05` folder. Here we go with the first code section:

```
# Importing the libraries
import numpy as np
```

You'll only need one library, called NumPy. This is a very useful library, helping when we are dealing with multi-dimensional arrays and lists in general. Give it the abbreviation `np`, which is the industry standard, so that it will be easier to use.

Now we have to understand something very important. You are creating a simulation whose aim is to simulate real-life situations. In reality, every slot machine gives us some chance of winning, and some machines have it higher than others. Therefore, when simulating this environment, you have to do the same thing. It is important to remember, however, that our AI will not know these predefined winning rates. It cannot just read them and judge, based on these rates, which machine is the best.

For this example, let's call this list of winning chances `conversionRates`.

```
# Setting conversion rates and the number of samples
conversionRates = [0.15, 0.04, 0.13, 0.11, 0.05]
N = 10000
d = len(conversionRates)
```

Here, you have five slot machines. They have some win chance; for example, slot machine no. 1 offers a 15% chance of a win. Then you create a number of samples, `N`. Remember, you are performing a simulation, so you need to have a predefined dataset that will tell you whether you won or not when you're playing. You also introduce a variable, `d`, which is the length of your conversion rates list; that is, the number of slot machines. It's useful to use short variable names like that, because the code would be longer and less readable otherwise.

Do you have an idea of what you should do next? You are running a simulation, so you need to have a predefined set of wins and losses for every slot machine for every sample. I highly recommend that you try to do this on your own. You need to have a set that will tell you if at some timestep i you have won or not by playing a certain slot machine. The answer is in the next snippet of code.

```
# Creating the dataset
X = np.zeros((N, d))
for i in range(N):
    for j in range(d):
        if np.random.rand() < conversionRates[j]:
            X[i][j] = 1
```

In the first line, you create a 2d-array full of zeros, of size $N * d$. This means that you've created an array with N (in this case 10000) rows and d (in this case 5) columns. Then, in a `for` loop, you iterate through every row in that 2d-array x . In a nested `for` loop, you iterate through each column in that row. In line 5 of the preceding code snippet, for each slot machine (each column), we check if a random float number from range (0,1) is smaller than the conversion rate for the corresponding slot machine.

That's just like playing the slot machine; since there is an equal chance of getting any float number from this range, the chances of getting a number smaller than x (where x is also in range (0,1)) is equal to x . For example, for $d = 0.15$, there are 15 instances out of 100 of getting a smaller float number than 0.15, and thus a 15% chance of returning a high reward for slot machine 1. In other words, if the random float is smaller, then that means you will win if you play this certain machine at this certain timestep.

To make sure you understand, if one of the N samples from your dataset x looks like this: $[0, 1, 0, 0, 1]$, you would win at that point in time by playing slot machine no. 2 or no. 5.

Next, you need to create two arrays that will count how many times you have lost and won by playing each slot machine, like this:

```
# Making arrays to count our losses and wins
nPosReward = np.zeros(d)
nNegReward = np.zeros(d)
```

Name them `nPosReward` (number of wins) and `nNegReward` (number of losses).

Now that you have made a simulation set and these two counters, you can start coding some Thompson Sampling. Keep in mind that the theory, as well as another example, will be covered later.

Next, initialize a `for` loop that will iterate through every sample in our dataset and choose the best slot machine. Initially, only create two variables, one called `selected`, which will tell you which slot machine was chosen, and `maxRandom`, which you will use to get the highest Beta distribution guess across all slot machines:

```
# Taking our best slot machine through beta distribution and updating
its losses and wins
for i in range(N):
    selected = 0
    maxRandom = 0
```

So now you can get to the core of Thompson Sampling. You'll take random guesses from our Beta distribution and find the highest value across all your slot machines.

You can use a method taken from NumPy, called `np.random.beta(a, b)`, that returns this random guess. Knowing that, try to find the highest guess and the best machine on your own! It is totally fine if you fail—we haven't covered the theory yet—and I will provide you with an answer. Good luck!

I hope you've given it a try. Whether it's worked out for you or not, here's my answer:

```
for j in range(d):
    randomBeta = np.random.beta(nPosReward[j] + 1, nNegReward[j] +
1)
    if randomBeta > maxRandom:
        maxRandom = randomBeta
        selected = j
```

You haven't missed anything – this is all the code needed for this task. You create a `for` loop to iterate through every slot machine and find the best one. For each slot machine of index `j` (remember that you are still in the bigger `for` loop with index `i`), you take a random draw, called `randomBeta`, from our Beta distribution, and check if it is greater than `maxRandom`.

If it is, then you reassign `maxRandom` to be equal to `randomBeta`, and set `selected` to be equal to the index of this new highest-guess slot machine `j`. It is also worth mentioning what the `a` and `b` arguments of the Beta function are in this case; they're the number of wins and losses we've had on the specific slot machine. Remember, the bigger the first argument, the better, and the higher our random guess will be; the bigger the second argument, the worse, and the lower our random guess will be.

Now that you have selected the best slot machine, what do you think you should do next?

You have to update your `nPosReward` or `nNegReward` depending on whether you have won or not. We can do that with this code:

```
if X[i][selected] == 1:
    nPosReward[selected] += 1
else:
    nNegReward[selected] += 1
```

Here, you can see the use of the `X` array you created earlier. You check if you have won this round by checking if there's a `1` in the appropriate place in your `X` array. If you win, you update the index corresponding to the selected machine in `nPosReward` by adding `1`. If you lose, however, you update `nNegReward` by adding `1` in the same index there. You can clearly see that if you win, next time, your random guess from the Beta distribution for that machine will be higher; and if you lose, it will be lower.

This code works already, although it is worth adding a few lines of code to display which slot machine your code considers the best:

```
# Showing which slot machine is considered the best
nSelected = nPosReward + nNegReward
for i in range(d):
    print('Machine number ' + str(i + 1) + ' was selected ' +
          str(nSelected[i]) + ' times')
print('Conclusion: Best machine is machine number ' + str(np.
      argmax(nSelected) + 1))
```

Here, you simply display how many times each slot machine was chosen by your algorithm. To get these numbers you can add together the lists `nPosReward` and `nNegReward`. In the final line, you show which machine was chosen the highest number of times, making it the slot machine that is considered the best.

Now, you can just run your code and see the results:

```
Machine number 1 was selected 7927.0 times
Machine number 2 was selected 82.0 times
Machine number 3 was selected 1622.0 times
Machine number 4 was selected 306.0 times
Machine number 5 was selected 63.0 times
Conclusion: Best machine is machine number 1
```

As we can see, your algorithm **quickly** found out that machine no. 1 is the best. It did it in around 2,000 rounds (2,000 samples in your `x` set).

Understanding the model

Thompson Sampling is, by far, the best model for this kind of problem; at the end of this chapter, you will see a comparison with another method. Here's how it works its magic. The first thing we do, when finding the best slot machine, is obviously to play the arm of each of the five slot machines one by one. So here we go:

Round 1: We play the arm of slot machine number 1. Let's say we get reward 0.

Round 2: We play the arm of slot machine number 2. Let's say we get reward 1.

Round 3: We play the arm of slot machine number 3. Let's say we get reward 0.

Round 4: We play the arm of slot machine number 4. Let's say we get reward 0.

Round 5: We play the arm of slot machine number 5. Let's say we get reward 1.

Now, why do you think we had to do this? We only did that to collect some starting information from each of the slot machines. This information will be needed in future rounds.

Now, things start to get interesting. What are we going to do at round 6? Which arm are we going to play?

Well, we need to look back at what happened during the first five rounds. For each slot machine, we introduce two new variables, one that counts the number of times the slot machine returned a 0 reward, and another one that counts the number of times the slot machine returned a 1 reward.

Let's denote these variables as $N_i^0(n)$ and $N_i^1(n)$, where $N_i^0(n)$ is the number of times slot machine number i returned reward 0 up to round n , and $N_i^1(n)$ is the number of times slot machine number i returned reward 1 up to round n . These two variables are denoted by `nNegReward` and `nPosReward` in our code. So, based on what we've obtained so far at round 5, let's give some values examples of these variables:

$N_1^0(1)=1$ means that slot machine 1 has returned 1 loss over 1 round.

$N_1^1(1)=0$ means that slot machine 1 has returned 0 wins over 1 round.

$N_2^0(1)=0$ means that slot machine 2 has returned 0 losses over 1 round.

$N_2^1(1)=1$ means that slot machine 2 has returned 1 win over 1 round.

$N_5^0(4)=0$ means that slot machine 5 has returned 0 losses over 4 rounds.

$N_5^1(4)=0$ means that slot machine 5 has returned 0 wins over 4 rounds.

$N_5^0(5)=0$ means that slot machine 5 has returned 0 losses over 5 rounds.

$N_5^1(5)=1$ means that slot machine 5 has returned 1 win over 5 rounds.

Alright, that was the easy part. The good news is that we've created all the variables we needed for our AI. The bad news is that now comes the hard part, the math. If you think math is good news, I like your spirit; but don't worry if you don't like math, I won't let you down.

What is a distribution?

The next step of our AI journey is to introduce distributions in mathematics. For this, I'll give you a simple definition with my own words, not the very formal ones you find in math books. I want to make sure everybody understands. Here it is: the distribution of a variable is a function that will give, for each value in the possible range of values the variable could take, the probability that this variable is equal to that value.

Let's really understand what it is through an example:

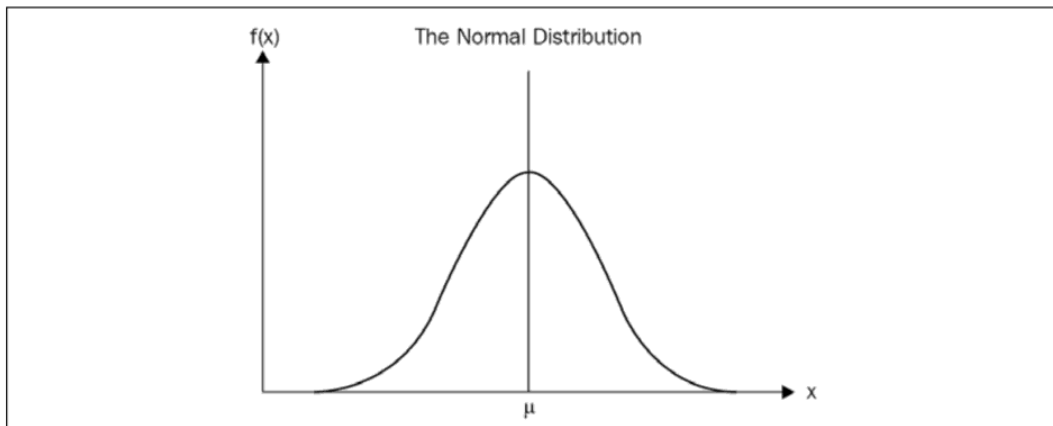


Figure 1: The normal distribution

In the preceding graph, you can see an example of a distribution. Now, remember in the definition I gave you, I mentioned two measures: "range of values the variable could take", and "probability that this variable is equal to that value". In any distribution, on the x -axis you have the range of values the variable could take, and on the y -axis you have the probability that the variable is equal to each value.

Don't worry if this isn't clear yet. To extend our example, let's say that on the preceding graph, this variable is the annual salary people have in a specific country.

On the x -axis, we would have the range of annual salaries from the minimum wage to the maximum wage, let's say from 15,000 dollars to 150,000 dollars. And on the y -axis, we would have the probabilities that a person would have that salary.

Now it should make more sense. For the low salaries, the curve is low, meaning that the probability that an individual earns a salary of around 15,000 dollars is low.

Then, up to the center of the x -axis, marked as μ , which is the average of the salaries, the probabilities of people's salaries increase. Let's say that μ is equal to 45,000 dollars. We intuitively understand that the probability that an individual in a specific country earns 45,000 dollars per year is the highest, simply because the majority of people earn something in the region of 45,000 dollars per year. And that's exactly why the distribution in the graph is the highest at this salary.

The higher we go above an annual salary of 45,000 dollars, the fewer people we'll find earning such salaries, and therefore the probability of people earning such salaries will decrease, until we go beyond an annual salary of 150,000 dollars, where very few people earn that much, therefore leading to a close-to-zero probability.

Alright, that was the distribution explained intuitively. Now, you have to know that there are many types of distributions: Gaussian distributions (that look like the preceding graph), normal distributions (Gaussian distribution of mean 0 and variance 1), Beta distributions, and many more.

That's the next step: **Beta distributions**. The Beta distribution is at the heart of the AI we built to solve our bandit problem. Here are what Beta distributions look like:

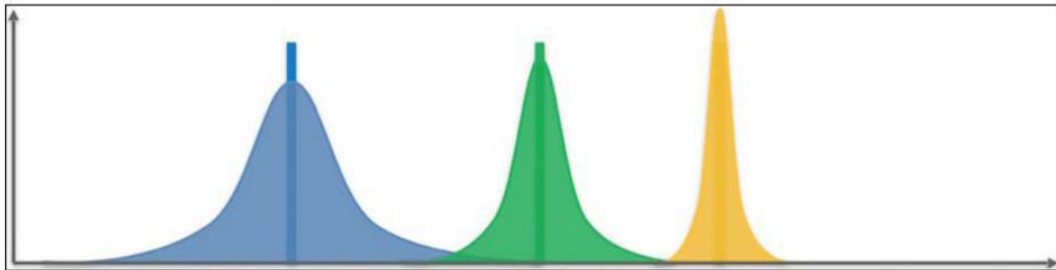


Figure 2: Three Beta distributions

Let's do some practice to make sure you understand how distributions work. Imagine these three distributions correspond to three different countries, and again let's say that they are the distributions of salaries in these countries. Which country has the highest salaries? Is it the purple one, the green one, or the yellow one? The answer is the yellow one, of course! It is in this country that we have positive probabilities for the highest salaries (remember, the salaries are on the x -axis, and the probabilities are on the y -axis).

That was just a quick test to make sure you were with me. Now, you don't have to remember the exact formula of a Beta distribution, but you do have to know that it has two parameters and how they impact the distribution. Don't forget that this was already mentioned when we solved the problem in practice, now it is explained in much more detail.

If we denote these two parameters as a and b again, we can denote the Beta distribution with the following:

$$y = \beta(x, a, b)$$

You might be asking what just happened – Why did x appear? Don't worry, we will demystify all this. In the formula above, y is the probability, β is a function of x only, x is the salary, and a, b are the two parameters present in any Beta distribution. Again, you don't have to know the exact definition of the function β , but just keep in mind the shape of its curve as given in the preceding graph.

However, what is really important for you to understand now is the role of the two parameters a and b . Following are the two points that you must know and visualize in your head:

1. Given two Beta distributions with the same parameter b , the one having a larger parameter a will be shifted more to the right.
2. Given two Beta distributions with the same parameter a , the one having a larger parameter b will be shifted more to the left.

That's it! That's enough to have an intuitive understanding of how our AI will solve the Bandit problem. In other words, the larger the parameter a , the more it will shift the Beta distribution to the right, and the larger the parameter b , the more it will shift the Beta distribution to the left.

Let's practice this! If I give you the following three Beta distributions:

1. $\beta(1,5)$
2. $\beta(5,1)$
3. $\beta(3,3)$

Could you tell me which of the three Beta distributions in the following graph they would approximately look like?

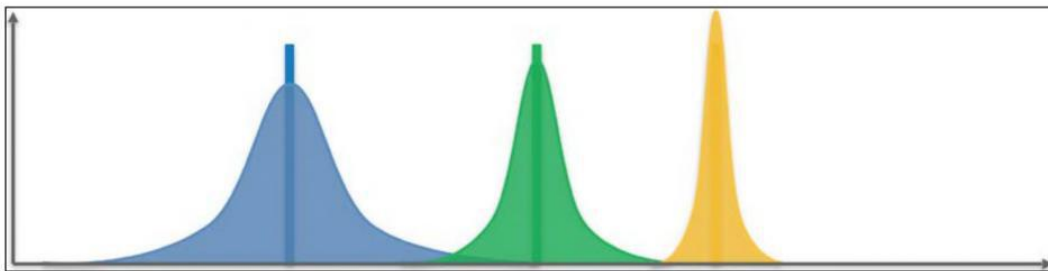


Figure 3: Three Beta distributions

Based on the two statements above, $\beta(1,5)$ is the purple one, $\beta(5,1)$ is the yellow one, and $\beta(3,3)$ is the green one. Congratulations to you if you guessed that right!

Now you are ready to solve our bandit problem. But let me ask you a question first, which might lead you to understand the magic faster than this book:

If, instead of the salaries in a country, the x -axis contained the success rates of the machines in the casino, and if each of the three Beta distributions represented one particular slot machine, which one would you choose to bet your 1,000 dollars?

You would choose the yellow one!

Of course! This distribution has positive probabilities for the highest conversion rates, since it is the one most shifted to the right.

This was already discussed in the previous code section of this chapter; I told you there that the higher the first parameter, the better the slot machine. Indeed, the Beta distribution will be shifted more to the right, meaning that this slot machine has a higher chance of giving us a win. Additionally, the higher the second parameter, the worse the slot machine is and now, the Beta distribution will be shifted to the left, meaning that this machine has a lower chance of us winning.

And now another question, before we solve our bandit problem. Remembering that you have five slot machines to play with, try to answer this question: if the five slot machines are associated with the following five Beta distributions of success rates:

$\beta(1,3)$, $\beta(1,5)$, $\beta(3,3)$, $\beta(5,3)$, and $\beta(5,1)$,

Which one would you pick to bet your 1,000 dollars?

The answer is $\beta(5,1)$!

Of course, again! Because it is the one with the largest parameter a and the lowest parameter b , therefore the most shifted to the right, and hence the one having the positive probabilities for the highest conversion rates.

If you are still with me, you are definitely ready to understand the AI magic. If not, please read through this section again. In the next section, I will finally reveal what happens next after Round 5.

Tackling the MABP

What we are going to do from now on before playing each round is to associate each slot machine with a specific Beta distribution. At each round n , the slot machine number i ($i=1,2,3,4,5$) will be associated with the following Beta distribution:

$$\beta(N_i^1(n)+1, N_i^0(n)+1)$$

Here, you should recall the following:

- $N_i^1(n)$ is the number of times the slot machine number i returned a 1 reward up to round n .
- $N_i^0(n)$ is the number of times the slot machine number i returned a 0 reward up to round n .

Remember, in the Beta distribution $\beta(a,b)$, the higher the parameter a , the more that shifts the distribution to the right. The higher the parameter b , the more that shifts the distribution to the left. Therefore, since at each round n and for each slot machine, the parameter a is the number of times (plus 1) it returned 1 up to round n , and the parameter b is the number of times (plus 1) it returned 0 up to round n , then that means the following: the more the slot machine returns 1 (success), the more its distribution will be shifted to the right; and the more the slot machine returns 0 (failure), the more its distribution will be shifted to the left.

Congratulations if you figured out what a and b should be on your own. We already used them in the practical tutorial above; we had two arrays, `nPosReward` and `nNegReward`, that correspond to $N_i^1(n)$ and $N_i^0(n)$ respectively.

Once you understand this, try to figure out the strategy before I give you the solution.

Alright, you are about to see the magic. What we are going to do, before playing the arm at each round, is take a random draw from each of the five distributions corresponding to the five slot machines. In case you're not clear what that means, I'll explain. Let me show you again the graph of the three Beta distributions:

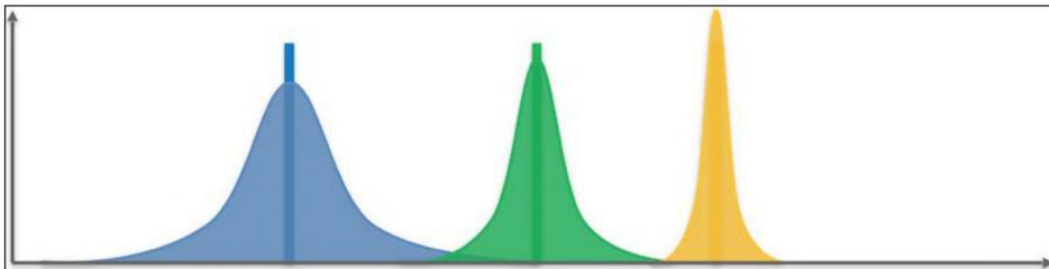


Figure 4: Three Beta distributions

What did I mean by taking a random draw? First, remember that for our bandit problem, on the x -axis, we have the success rates from 0 to 1. For example, $x = 0.25$ means that the machine returns a 1 reward (success) 25% of the time. Then, on the y -axis, we still have the probabilities to have these success rates.

Let's focus on one distribution, for example, the purple one. What would it mean to take a random draw from that distribution? That would mean very simply that we randomly pick a value on the x -axis where the distribution is positive, such that the x values where the probability is the highest will get the highest chance to be picked. For example, let's say the top of the purple curve corresponds to $x = 0.2$ and $y = 0.35$.

Then, taking a random draw from that purple distribution means that we will have a 35% chance to pick a success rate of 20%. To generalize this, let's say that $y = \beta_{\text{purple}}(x)$ is the function associated with the purple distribution, so taking a random draw from that purple distribution means that for each success rate x on the x -axis, we will have $\beta_{\text{purple}}(x)$ chance of picking x . That is what "to take a random draw from a distribution" means, and this is also called "to sample a distribution".

Now that you understand this, let's see where we left off. We said that before playing the arm at each round, we were going to take a random draw from each of the five distributions corresponding to the five slot machines. We thus obtain five values on the x -axis, each one corresponding to each of the five slot machines. Then, here comes the crucial question, the one that will tell whether you have the right intuition about the strategy.

According to you, which slot machine are you going to play, based on the observation of these five values? I really want you to take some time to answer this question, because right now, we are at the heart of the strategy (you can also have a look at our previously written code). The answer can be found in the next paragraph.

I really hope you tried figuring this out by yourself: the slot machine that you are going to play next is the one for which we got the highest of the five random draws. Why? Because the highest random draws correspond to the highest success rate, and for this highest success rate, the Beta distribution associated with the slot machine picked has positive probabilities around that highest success rate.

Since we want to maximize the success rate of the machines we play (because we want to make money), we must pick the slot machine for which the Beta distribution has positive probabilities around the highest success rates. In the following graph, that's the yellow distribution.

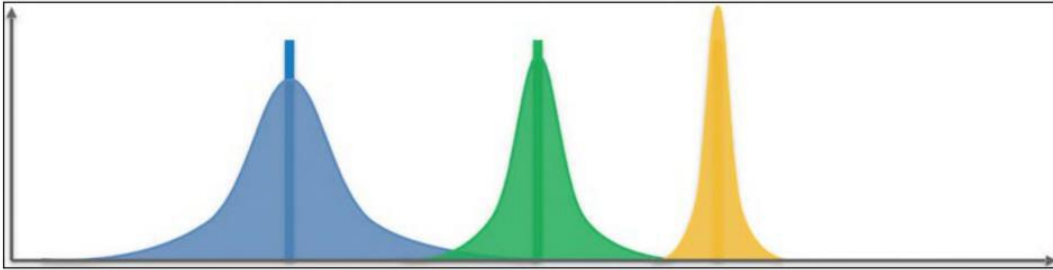


Figure 5: Three Beta distributions

Now, we must take a step back. I've been in your situation many times when I am learning something new and technical, which sometimes felt overwhelming. In that case, the best move is to take a step back, which is exactly what we are going to do now by giving a recap of the strategy and its intuition.

The Thompson Sampling strategy in three steps

After we play each of the five slot machines over the first five rounds, here's what the AI will do at each round n :

1. For each slot machine i ($i=1,2,3,4,5$), we take a random draw $\theta_i(n)$ from its Beta distribution:

$$\theta_i(n) \sim \beta(N_i^1(n)+1, N_i^0(n)+1)$$

where:

$N_i^1(n)$ is the number of times the slot machine number i returned a 1 reward up to round n .

$N_i^0(n)$ is the number of times the slot machine number i returned a 0 reward up to round n .

2. We pull the arm of the slot machine $s(n)$ that has the highest sampled $\theta_i(n)$:

$$s(n) = \operatorname{argmax}_{i=1,2,3,4,5} (\theta_i(n))$$

3. We don't forget to update $N_{s(n)}^1(n)$ or $N_{s(n)}^0(n)$:

If the played slot machine $s(n)$ returned a 1 reward:

$$N_{s(n)}^1(n) := N_{s(n)}^1(n) + 1$$

If the played slot machine $s(n)$ returned a 0 reward:

$$N_{s(n)}^0(n) := N_{s(n)}^0(n) + 1$$

Then, we repeat these three steps at each round until we spend our 1,000 dollars. This strategy, called Thompson Sampling, is a basic but powerful model of a specific branch of AI, called Reinforcement Learning.

The final touch of shaping your Thompson Sampling intuition

Your intuition about why and how this works should be as follows (try to keep it in mind or visualize it on the graphic):

Each slot machine has its own Beta distribution. Over the rounds, the Beta distribution of the slot machine with the highest conversion rate will be progressively shifted to the right, and the Beta distributions of the strategies with lower conversion rates will be progressively shifted to the left (Steps 1 and 3). Therefore, because of Step 2, the slot machine with the highest conversion rate will be selected more and more.

And voilà! Congratulations – you just learned about a powerful AI model, a massive step in your journey. To see Thompson Sampling in action and check that it indeed works, I won't force you to go to a casino and try it out; We'll apply it to another real-life model in *Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street*.

Finally, let me finish this theory tutorial with a question for you. Remember earlier in the book I told you that any AI we build today takes as input a state, returns as output an action to play, and, after playing the action, gets a reward (positive or negative). **For this particular bandit problem, what are the input states, the actions played, and the rewards received?** Think about this before reading the next paragraph.

Here we go with the answer:

- The input state is the exact round we've reached, including the information of the two parameters $N_{s(n)}^1(n)$ and $N_{s(n)}^0(n)$.
- The output action is the arm we pull from the selected slot machine.
- The reward is 1 or 0, 1 if the slot machine returns twice our dollar invested, and 0 if we lose our dollar.

Congratulations to you if you answered that one correctly, and for tackling this first AI model, Thompson Sampling. And don't forget, in *Chapter 6, AI for Sales and Advertising – Sell like the Wolf of AI Street*, we put this into practice to solve a real-world business problem.

Thompson Sampling against the standard model

When I learned Thompson Sampling for the first time, I had one main question in my mind: is it really that good? In fact, if you were to run the standard model (by "standard model" I mean playing every slot machine a certain number of times) and Thompson Sampling separately you might not see much difference; you would likely come to the conclusion that they work pretty much as well as each other.

To check whether it is true that Thompson Sampling isn't any better, I implemented a code to test both solutions on many different scenarios. The changes included: number of samples (200 or 1,000 or 5,000), number of slot machines (from 3 to 20), and conversion rate ranges (ranges in which conversion rates could be set: 0-0.1; 0-0.3; 0-0.5).

Every scenario was tested 100 times to compute the accuracy of each model.

The results and the code used are provided in the `resultsModified.xlsx` and `comparison.py` files, respectively, in Chapter 05 of this book's GitHub page. Here, you can see some graphs taken from this Excel file that show the performance of both models:

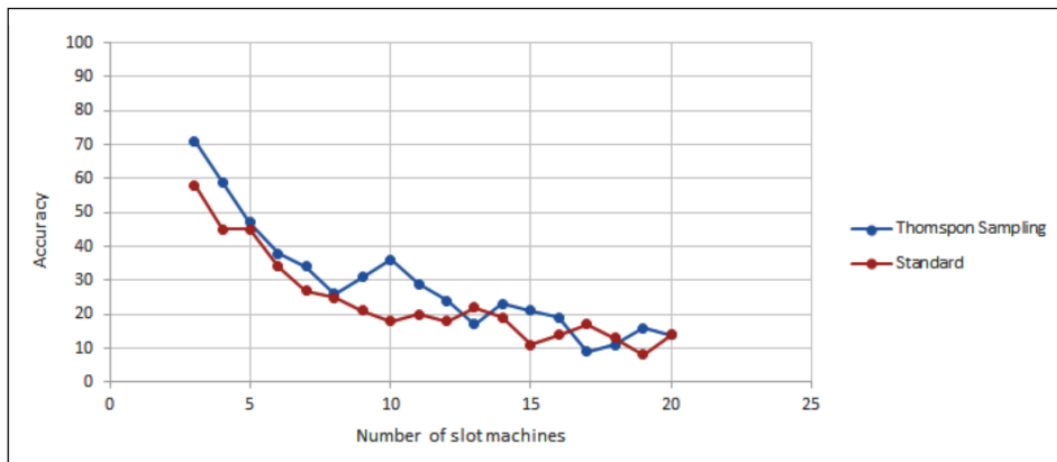


Figure 6: Accuracy vs. Number of slot machines (200 samples)

This first graph in *Figure 6* illustrates the accuracy of both models depending on the number of slot machines. The number of samples was set to 200 and the conversion rate ranges were set to 0-0.1, meaning that the differences between these rates were minor. This is the toughest setting for this comparison. Overall, Thompson Sampling performed better than the standard model (22% better).

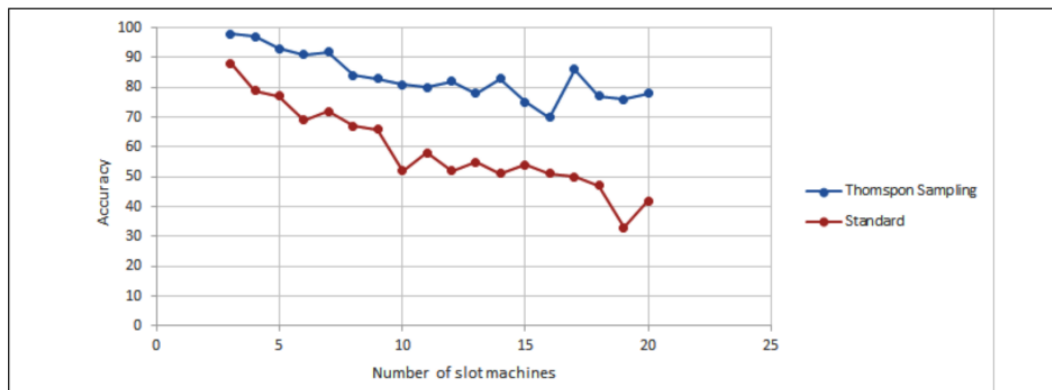


Figure 7: Accuracy vs. Number of slot machines (5,000 samples)

This second graph in *Figure 7* shows the performance under the easiest conditions. The number of samples was set to 5,000 and the conversion rate ranges were set to 0-0.5, meaning that the differences were clearly visible. The overall drop of accuracy for Thompson Sampling is smaller than the drop in accuracy for the standard solution. Thompson Sampling performed significantly better this time (41% better).

Taking all scenarios into consideration, Thompson Sampling achieved a mean accuracy of 57% and the standard model achieved 43% accuracy. This is a significant difference taking into account the fact that very tough scenarios were tested (for example, only 200 samples, a range of 0-0.1, and 20 slot machines).

Summary

Thompson Sampling is a powerful sampling technique that enables you to quickly figure out the highest of a number of unknown conversion rates. It is always applied in the same frame, called the multi-armed bandit problem, which in the classic sense is composed of several slot machines, each one having a different conversion rate of positive outcomes. We had a first glance at how this AI solves this problem better and faster than standard methods.

In the next chapter, we will perform a full practical activity where we will see how the multi-armed bandit frame can easily model a business problem – online advertising – and how Thompson Sampling can bring significant added value.