

EXPERT INSIGHT

Artificial Intelligence By Example

Acquire advanced AI, machine learning,
and deep learning design skills

Second Edition



Denis Rothman

Packt

Artificial Intelligence By Example

Second Edition

Acquire advanced AI, machine learning, and deep learning design skills

Denis Rothman

Packt>

BIRMINGHAM - MUMBAI

Artificial Intelligence By Example

Second Edition

Copyright © 2020 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.

Producer: Tushar Gupta

Acquisition Editor - Peer Reviews: Divya Mudaliar

Content Development Editor: Dr. Ian Hough

Technical Editor: Saby D'silva

Project Editor: Kishor Rit

Proofreader: Safis Editing

Indexer: Tejal Daruwale Soni

Presentation Designer: Pranit Padwal

First published: May 2018

Second edition: February 2020

Production reference: 2140420

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-83921-153-9

www.packt.com



packt.com

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

Why subscribe?

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

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

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

Contributors

About the author

Denis Rothman graduated from Sorbonne University and Paris-Diderot University, writing one of the very first word2matrix embedding solutions. He began his career authoring one of the first AI cognitive **natural language processing (NLP)** chatbots applied as a language teacher for Moët et Chandon and other companies. He authored an AI resource optimizer for IBM and apparel producers. He then authored an **advanced planning and scheduling (APS)** solution used worldwide.

"I want to thank the corporations who trusted me from the start to deliver artificial intelligence solutions and share the risks of continuous innovation. I also thank my family, who believed I would make it big at all times."

About the reviewers

Carlos Toxtli is a human-computer interaction researcher who studies the impact of artificial intelligence in the future of work. He studied a Ph.D. in Computer Science at the University of West Virginia and a master's degree in Technological Innovation and Entrepreneurship at the Monterrey Institute of Technology and Higher Education. He has worked for some international organizations such as Google, Microsoft, Amazon, and the United Nations. He has also created companies that use artificial intelligence in the financial, educational, customer service, and parking industries. Carlos has published numerous research papers, manuscripts, and book chapters for different conferences and journals in his field.

"I want to thank all the editors who helped make this book a masterpiece."

Kausthub Raj Jadhav graduated from the University of California, Irvine, where he specialized in intelligent systems and founded the Artificial Intelligence Club. In his spare time, he enjoys powerlifting, rewatching Parks and Recreation, and learning how to cook. He solves hard problems for a living.

Table of Contents

Preface	xiii
Chapter 1: Getting Started with Next-Generation Artificial Intelligence through Reinforcement Learning	1
Reinforcement learning concepts	2
How to adapt to machine thinking and become an adaptive thinker	4
Overcoming real-life issues using the three-step approach	5
Step 1 – describing a problem to solve: MDP in natural language	7
Watching the MDP agent at work	8
Step 2 – building a mathematical model: the mathematical representation of the Bellman equation and MDP	10
From MDP to the Bellman equation	10
Step 3 – writing source code: implementing the solution in Python	14
The lessons of reinforcement learning	16
How to use the outputs	18
Possible use cases	20
Machine learning versus traditional applications	23
Summary	24
Questions	24
Further reading	25
Chapter 2: Building a Reward Matrix – Designing Your Datasets	27
Designing datasets – where the dream stops and the hard work begins	28
Designing datasets	29
Using the McCulloch-Pitts neuron	29
The McCulloch-Pitts neuron	31
The Python-TensorFlow architecture	35

Logistic activation functions and classifiers	35
Overall architecture	35
Logistic classifier	36
Logistic function	37
Softmax	38
Summary	42
Questions	43
Further reading	43
Chapter 3: Machine Intelligence – Evaluation Functions and Numerical Convergence	45
Tracking down what to measure and deciding how to measure it	46
Convergence	48
Implicit convergence	49
Numerically controlled gradient descent convergence	49
Evaluating beyond human analytic capacity	56
Using supervised learning to evaluate a result that surpasses human analytic capacity	60
Summary	64
Questions	65
Further reading	65
Chapter 4: Optimizing Your Solutions with K-Means Clustering	67
Dataset optimization and control	68
Designing a dataset and choosing an ML/DL model	69
Approval of the design matrix	70
Implementing a k-means clustering solution	74
The vision	74
The data	75
The strategy	76
The k-means clustering program	77
The mathematical definition of k-means clustering	78
The Python program	80
Saving and loading the model	84
Analyzing the results	85
Bot virtual clusters as a solution	86
The limits of the implementation of the k-means clustering algorithm	87
Summary	88
Questions	88
Further reading	89
Chapter 5: How to Use Decision Trees to Enhance K-Means Clustering	91
Unsupervised learning with KMC with large datasets	92

Identifying the difficulty of the problem	94
NP-hard – the meaning of P	94
NP-hard – the meaning of non-deterministic	95
Implementing random sampling with mini-batches	95
Using the LLN	96
The CLT	96
Using a Monte Carlo estimator	97
Trying to train the full training dataset	98
Training a random sample of the training dataset	98
Shuffling as another way to perform random sampling	100
Chaining supervised learning to verify unsupervised learning	102
Preprocessing raw data	103
A pipeline of scripts and ML algorithms	103
Step 1 – training and exporting data from an unsupervised ML algorithm	105
Step 2 – training a decision tree	106
Step 3 – a continuous cycle of KMC chained to a decision tree	110
Random forests as an alternative to decision trees	114
Summary	118
Questions	118
Further reading	119
Chapter 6: Innovating AI with Google Translate	121
Understanding innovation and disruption in AI	123
Is AI disruptive?	123
AI is based on mathematical theories that are not new	124
Neural networks are not new	124
Looking at disruption – the factors that are making AI disruptive	125
Cloud server power, data volumes, and web sharing of the early 21st century	125
Public awareness	126
Inventions versus innovations	126
Revolutionary versus disruptive solutions	127
Where to start?	127
Discover a world of opportunities with Google Translate	128
Getting started	128
The program	128
The header	128
Implementing Google's translation service	129
Google Translate from a linguist's perspective	130
Playing with the tool	131
Linguistic assessment of Google Translate	131
AI as a new frontier	135
Lexical field and polysemy	135
Exploring the frontier – customizing Google Translate with a Python program	137

k-nearest neighbor algorithm	138
Implementing the KNN algorithm	139
The knn_polysemy.py program	142
Implementing the KNN function in Google_Translate_Customized.py	144
Conclusions on the Google Translate customized experiment	152
The disruptive revolutionary loop	153
Summary	153
Questions	154
Further reading	154
Chapter 7: Optimizing Blockchains with Naive Bayes	157
Part I – the background to blockchain technology	158
Mining bitcoins	159
Using cryptocurrency	160
PART II – using blockchains to share information in a supply chain	161
Using blockchains in the supply chain network	164
Creating a block	165
Exploring the blocks	166
Part III – optimizing a supply chain with naive Bayes in a blockchain process	167
A naive Bayes example	167
The blockchain anticipation novelty	169
The goal – optimizing storage levels using blockchain data	170
Implementation of naive Bayes in Python	173
Gaussian naive Bayes	173
Summary	177
Questions	177
Further reading	178
Chapter 8: Solving the XOR Problem with a Feedforward Neural Network	179
The original perceptron could not solve the XOR function	180
XOR and linearly separable models	181
Linearly separable models	181
The XOR limit of a linear model, such as the original perceptron	182
Building an FNN from scratch	184
Step 1 – defining an FNN	184
Step 2 – an example of how two children can solve the XOR problem every day	185
Implementing a vintage XOR solution in Python with an FNN and backpropagation	189
A simplified version of a cost function and gradient descent	191
Linear separability was achieved	194

Applying the FNN XOR function to optimizing subsets of data	196
Summary	202
Questions	203
Further reading	203
Chapter 9: Abstract Image Classification with Convolutional Neural Networks (CNNs)	205
<hr/>	
Introducing CNNs	206
Defining a CNN	207
Initializing the CNN	209
Adding a 2D convolution layer	210
Kernel	210
Shape	215
ReLU	215
Pooling	218
Next convolution and pooling layer	219
Flattening	220
Dense layers	220
Dense activation functions	221
Training a CNN model	221
The goal	222
Compiling the model	223
The loss function	223
The Adam optimizer	225
Metrics	226
The training dataset	226
Data augmentation	227
Loading the data	227
The testing dataset	228
Data augmentation on the testing dataset	228
Loading the data	228
Training with the classifier	229
Saving the model	230
Next steps	230
Summary	231
Questions	231
Further reading and references	231
Chapter 10: Conceptual Representation Learning	233
<hr/>	
Generating profit with transfer learning	234
The motivation behind transfer learning	235
Inductive thinking	235
Inductive abstraction	235
The problem AI needs to solve	236

The Γ gap concept	237
Loading the trained TensorFlow 2.x model	238
Loading and displaying the model	238
Loading the model to use it	242
Defining a strategy	245
Making the model profitable by using it for another problem	246
Domain learning	247
How to use the programs	247
The trained models used in this section	248
The trained model program	248
Gap – loaded or underloaded	249
Gap – jammed or open lanes	251
Gap datasets and subsets	253
Generalizing the Γ (the gap conceptual dataset)	253
The motivation of conceptual representation learning	
metamodels applied to dimensionality	254
The curse of dimensionality	254
The blessing of dimensionality	255
Summary	256
Questions	257
Further reading	257
Chapter 11: Combining Reinforcement Learning and Deep Learning	259
Planning and scheduling today and tomorrow	260
A real-time manufacturing process	262
Amazon must expand its services to face competition	262
A real-time manufacturing revolution	263
CRLMM applied to an automated apparel manufacturing process	266
An apparel manufacturing process	267
Training the CRLMM	269
Generalizing the unit training dataset	269
Food conveyor belt processing – positive $p\gamma$ and negative $n\gamma$ gaps	270
Running a prediction program	274
Building the RL-DL-CRLMM	274
A circular process	275
Implementing a CNN-CRLMM to detect gaps and optimize	276
Q-learning – MDP	277
MDP inputs and outputs	278
The optimizer	281
The optimizer as a regulator	281
Finding the main target for the MDP function	284
A circular model – a stream-like system that never starts nor ends	286

Summary	291
Questions	291
Further reading	292
Chapter 12: AI and the Internet of Things (IoT)	293
The public service project	294
Setting up the RL-DL-CRLMM model	295
Applying the model of the CRLMM	297
The dataset	298
Using the trained model	300
Adding an SVM function	301
Motivation – using an SVM to increase safety levels	302
Definition of a support vector machine	303
Python function	305
Running the CRLMM	307
Finding a parking space	307
Deciding how to get to the parking lot	310
Support vector machine	311
The itinerary graph	313
The weight vector	314
Summary	315
Questions	316
Further reading	316
Chapter 13: Visualizing Networks with TensorFlow 2.x and TensorBoard	317
Exploring the output of the layers of a CNN in two steps with TensorFlow	318
Building the layers of a CNN	319
Processing the visual output of the layers of a CNN	323
Analyzing the visual output of the layers of a CNN	327
Analyzing the accuracy of a CNN using TensorBoard	334
Getting started with Google Colaboratory	334
Defining and training the model	336
Introducing some of the measurements	339
Summary	341
Questions	342
Further reading	342
Chapter 14: Preparing the Input of Chatbots with Restricted Boltzmann Machines (RBMs) and Principal Component Analysis (PCA)	343
Defining basic terms and goals	344

Introducing and building an RBM	345
The architecture of an RBM	346
An energy-based model	347
Building the RBM in Python	350
Creating a class and the structure of the RBM	350
Creating a training function in the RBM class	350
Computing the hidden units in the training function	351
Random sampling of the hidden units for the reconstruction and contractive divergence	352
Reconstruction	353
Contrastive divergence	354
Error and energy function	354
Running the epochs and analyzing the results	355
Using the weights of an RBM as feature vectors for PCA	357
Understanding PCA	362
Mathematical explanation	363
Using TensorFlow's Embedding Projector to represent PCA	367
Analyzing the PCA to obtain input entry points for a chatbot	370
Summary	372
Questions	373
Further reading	373
Chapter 15: Setting Up a Cognitive NLP UI/CUI Chatbot	375
Basic concepts	376
Defining NLU	376
Why do we call chatbots "agents"?	376
Creating an agent to understand Dialogflow	377
Entities	378
Intents	382
Context	387
Adding fulfillment functionality to an agent	392
Defining fulfillment	393
Enhancing the cogfilmdr agent with a fulfillment webhook	394
Getting the bot to work on your website	397
Machine learning agents	398
Using machine learning in a chatbot	398
Speech-to-text	398
Text-to-speech	399
Spelling	401
Why are these machine learning algorithms important?	403
Summary	404
Questions	405
Further reading	405

Chapter 16: Improving the Emotional Intelligence Deficiencies of Chatbots	407
From reacting to emotions, to creating emotions	408
Solving the problems of emotional polysemy	408
The greetings problem example	409
The affirmation example	410
The speech recognition fallacy	410
The facial analysis fallacy	411
Small talk	412
Courtesy	412
Emotions	415
Data logging	415
Creating emotions	418
RNN research for future automatic dialog generation	423
RNNs at work	424
RNN, LSTM, and vanishing gradients	425
Text generation with an RNN	426
Vectorizing the text	426
Building the model	427
Generating text	429
Summary	431
Questions	432
Further reading	432
Chapter 17: Genetic Algorithms in Hybrid Neural Networks	433
Understanding evolutionary algorithms	434
Heredity in humans	434
Our cells	435
How heredity works	435
Evolutionary algorithms	436
Going from a biological model to an algorithm	437
Basic concepts	437
Building a genetic algorithm in Python	440
Importing the libraries	440
Calling the algorithm	441
The main function	441
The parent generation process	442
Generating a parent	442
Fitness	443
Display parent	444
Crossover and mutation	445
Producing generations of children	447
Summary code	450
Unspecified target to optimize the architecture of a neural network with a genetic algorithm	451

A physical neural network	451
What is the nature of this mysterious S-FNN?	452
Calling the algorithm cell	453
Fitness cell	454
ga_main() cell	455
Artificial hybrid neural networks	456
Building the LSTM	457
The goal of the model	458
Summary	459
Questions	460
Further reading	460
Chapter 18: Neuromorphic Computing	461
Neuromorphic computing	462
Getting started with Nengo	463
Installing Nengo and Nengo GUI	464
Creating a Python program	466
A Nengo ensemble	466
Nengo neuron types	467
Nengo neuron dimensions	468
A Nengo node	468
Connecting Nengo objects	470
Visualizing data	470
Probes	475
Applying Nengo's unique approach to critical AI research areas	479
Summary	482
Questions	483
References	483
Further reading	483
Chapter 19: Quantum Computing	485
The rising power of quantum computers	486
Quantum computer speed	487
Defining a qubit	490
Representing a qubit	490
The position of a qubit	491
Radians, degrees, and rotations	492
The Bloch sphere	493
Composing a quantum score	494
Quantum gates with Quirk	494
A quantum computer score with Quirk	496
A quantum computer score with IBM Q	497
A thinking quantum computer	500
Representing our mind's concepts	500

Expanding MindX's conceptual representations	500
The MindX experiment	501
Preparing the data	501
Transformation functions – the situation function	501
Transformation functions – the quantum function	504
Creating and running the score	504
Using the output	506
Summary	507
Questions	507
Further reading	508
Appendix: Answers to the Questions	509
<hr/>	
Chapter 1 – Getting Started with Next-Generation Artificial Intelligence through Reinforcement Learning	509
Chapter 2 – Building a Reward Matrix – Designing Your Datasets	511
Chapter 3 – Machine Intelligence – Evaluation Functions and Numerical Convergence	512
Chapter 4 – Optimizing Your Solutions with K-Means Clustering	513
Chapter 5 – How to Use Decision Trees to Enhance K-Means Clustering	515
Chapter 6 – Innovating AI with Google Translate	516
Chapter 7 – Optimizing Blockchains with Naive Bayes	518
Chapter 8 – Solving the XOR Problem with a Feedforward Neural Network	519
Chapter 9 – Abstract Image Classification with Convolutional Neural Networks (CNNs)	521
Chapter 10 – Conceptual Representation Learning	522
Chapter 11 – Combining Reinforcement Learning and Deep Learning	524
Chapter 12 – AI and the Internet of Things	525
Chapter 13 – Visualizing Networks with TensorFlow 2.x and TensorBoard	527
Chapter 14 – Preparing the Input of Chatbots with Restricted Boltzmann Machines (RBMs) and Principal Component Analysis (PCA)	528
Chapter 15 – Setting Up a Cognitive NLP UI/CUI Chatbot	529
Chapter 16 – Improving the Emotional Intelligence Deficiencies of Chatbots	530
Chapter 17 – Genetic Algorithms in Hybrid Neural Networks	531
Chapter 18 – Neuromorphic Computing	532
Chapter 19 – Quantum Computing	534
Other Books You May Enjoy	537
Index	541

Preface

This second edition of *Artificial Intelligence By Example* will take you through the main aspects of present-day **artificial intelligence (AI)** and beyond!

This book contains many revisions and additions to the key aspects of AI described in the first edition:

- The theory of machine learning and deep learning including hybrid and ensemble algorithms.
- Mathematical representations of the main AI algorithms including natural language explanations making them easier to understand.
- Real-life case studies taking the reader inside the heart of e-commerce: manufacturing, services, warehouses, and delivery.
- Introducing AI solutions that combine IoT, **convolutional neural networks (CNN)**, and **Markov decision process (MDP)**.
- Many open source Python programs with a special focus on the new features of TensorFlow 2.x, TensorBoard, and Keras. Many modules are used, such as scikit-learn, pandas, and more.
- Cloud platforms: Google Colaboratory with its free VM, Google Translate, Google Dialogflow, IBM Q for quantum computing, and more.
- Use of the power of **restricted Boltzmann machines (RBM)** and **principal component analysis (PCA)** to generate data to create a meaningful chatbot.
- Solutions to compensate for the emotional deficiencies of chatbots.

- Genetic algorithms, which run faster than classical algorithms in specific cases, and genetic algorithms used in a hybrid deep learning neural network.
- Neuromorphic computing, which reproduces our brain activity with models of selective spiking ensembles of neurons in models that reproduce our biological reactions.
- Quantum computing, which will take you deep into the tremendous calculation power of qubits and cognitive representation experiments.

This second edition of *Artificial Intelligence By Example* will take you to the cutting edge of AI and beyond with innovations that improve existing solutions. This book will make you a key asset not only as an AI specialist but a visionary. You will discover how to improve your AI skills as a consultant, developer, professor, a curious mind, or any person involved in artificial intelligence.

Who this book is for

This book contains a broad approach to AI, which is expanding to all areas of our lives.

The main machine learning and deep learning algorithms are addressed with real-life Python examples extracted from hundreds of AI projects and implementations.

Each AI implementation is illustrated by an open source program available on GitHub and cloud platforms such as Google Colaboratory.

Artificial Intelligence By Example, Second Edition is for developers who wish to build solid machine learning programs that will optimize production sites, services, IoT and more.

Project managers and consultants will learn how to build input datasets that will help the reader face the challenges of real-life AI.

Teachers and students will have an overview of the key aspects of AI, along with many educational examples.

What this book covers

Chapter 1, Getting Started with Next-Generation Artificial Intelligence through Reinforcement Learning, covers reinforcement learning through the Bellman equation based on the MDP. A case study describes how to solve a delivery route problem with a human driver and a self-driving vehicle. This chapter shows how to build an MDP from scratch in Python.

Chapter 2, Building a Reward Matrix – Designing Your Datasets, demonstrates the architecture of neural networks starting with the McCulloch-Pitts neuron. The case study describes how to use a neural network to build the reward matrix used by the Bellman equation in a warehouse environment. The process will be developed in Python using logistic, softmax, and one-hot functions.

Chapter 3, Machine Intelligence – Evaluation Functions and Numerical Convergence, shows how machine evaluation capacities have exceeded human decision-making. The case study describes a chess position and how to apply the results of an AI program to decision-making priorities. An introduction to decision trees in Python shows how to manage decision-making processes.

Chapter 4, Optimizing Your Solutions with K-Means Clustering, covers a k-means clustering program with Lloyd's algorithm and how to apply it to the optimization of automatic guided vehicles. The k-means clustering program's model will be trained and saved.

Chapter 5, How to Use Decision Trees to Enhance K-Means Clustering, begins with unsupervised learning with k-means clustering. The output of the k-means clustering algorithm will provide the labels for the supervised decision tree algorithm. Random forests will be introduced.

Chapter 6, Innovating AI with Google Translate, explains the difference between a revolutionary innovation and a disruptive innovation. Google Translate will be described and enhanced with an innovative k-nearest neighbors-based Python program.

Chapter 7, Optimizing Blockchains with Naive Bayes, is about mining blockchains and describes how blockchains function. We use naive Bayes to optimize the blocks of **supply chain management (SCM)** blockchains by predicting transactions to anticipate storage levels.

Chapter 8, Solving the XOR Problem with a Feedforward Neural Network, is about building a **feedforward neural network (FNN)** from scratch to solve the XOR linear separability problem. The business case describes how to group orders for a factory.

Chapter 9, Abstract Image Classification with Convolutional Neural Networks (CNNs), describes CNN in detail: kernels, shapes, activation functions, pooling, flattening, and dense layers. The case study illustrates the use of a CNN using a webcam on a conveyor belt in a food-processing company.

Chapter 10, Conceptual Representation Learning, explains **conceptual representation learning (CRL)**, an innovative way to solve production flows with a CNN transformed into a **CRL metamodel (CRLMM)**. The case study shows how to use a CRLMM for transfer and domain learning, extending the model to other applications.

Chapter 11, Combining Reinforcement Learning and Deep Learning, combines a CNN with an MDP to build a solution for automatic planning and scheduling with an optimizer with a rule-based system.

The solution is applied to apparel manufacturing showing how to apply AI to real-life systems.

Chapter 12, AI and the Internet of Things (IoT), explores a **support vector machine (SVM)** assembled with a CNN. The case study shows how self-driving cars can find an available parking space automatically.

Chapter 13, Visualizing Networks with TensorFlow 2.x and TensorBoard, extracts information of each layer of a CNN and displays the intermediate steps taken by the neural network. The output of each layer contains images of the transformations applied.

Chapter 14, Preparing the Input of Chatbots with Restricted Boltzmann Machines (RBM) and Principal Component Analysis (PCA), explains how to produce valuable information using an RBM and a PCA to transform raw data into chatbot-input data.

Chapter 15, Setting Up a Cognitive NLP UI/CUI Chatbot, describes how to build a Google Dialogflow chatbot from scratch using the information provided by an RBM and a PCA algorithm. The chatbot will contain entities, intents, and meaningful responses.

Chapter 16, Improving the Emotional Intelligence Deficiencies of Chatbots, explains the limits of a chatbot when dealing with human emotions. The **Emotion** options of Dialogflow will be activated along with **Small Talk** to make the chatbot friendlier.

Chapter 17, Genetic Algorithms in Hybrid Neural Networks, enters our chromosomes, finds our genes, and helps you understand how our reproduction process works. From there, it is shown how to implement an evolutionary algorithm in Python, a **genetic algorithm (GA)**. A hybrid neural network will show how to optimize a neural network with a GA.

Chapter 18, Neuromorphic Computing, describes what neuromorphic computing is and then explores Nengo, a unique neuromorphic framework with solid tutorials and documentation.

This neuromorphic overview will take you into the wonderful power of our brain structures to solve complex problems.

Chapter 19, Quantum Computing, will show quantum computers are superior to classical computers, what a quantum bit is, how to use it, and how to build quantum circuits. An introduction to quantum gates and example programs will bring you into the futuristic world of quantum mechanics.

Appendix, Answers to the Questions, provides answers to the questions listed in the *Questions* section in all the chapters.

To get the most out of this book

Artificial intelligence projects rely on three factors:

- **Understanding the subject the AI project will be applied to.** To do so, go through a chapter to pick up the key ideas. Once you understand the key ideas of a case study described in the book, try to see how an AI solution can be applied to real-life examples around you.
- **The mathematical foundations of the AI algorithms.** Do not skip the mathematics equations if you have the energy to study them. AI relies heavily on mathematics. There are plenty of excellent websites that explain the mathematics used in this book.
- **Development.** An artificial intelligence solution can be directly used on an online cloud platform machine learning site such as Google. We can access these platforms with APIs. In the book, Google Cloud is used several times. Try to create an account of your own to explore several cloud platforms to understand their potential and their limits. Development remains critical for AI projects.

Even with a cloud platform, scripts and services are necessary. Also, sometimes, writing an algorithm is mandatory because the ready-to-use online algorithms are insufficient for a given problem. Explore the programs delivered with the book. They are open source and free.

Technical requirements

The following is a non-exhaustive list of the technical requirements for running the codes in this book. For a more detailed chapter-wise list, please refer to this link: <https://github.com/PacktPublishing/Artificial-Intelligence-By-Example-Second-Edition/blob/master/Technical%20Requirements.csv>.

Package	Website
Python	https://www.python.org/
NumPy	https://pypi.org/project/numpy/
Matplotlib	https://pypi.org/project/matplotlib/
pandas	https://pypi.org/project/pandas/
SciPy	https://pypi.org/project/scipy/
scikit-learn	https://pypi.org/project/scikit-learn/
PyDotPlus	https://pypi.org/project/pydotplus/
Google API	https://developers.google.com/docs/api/quickstart/python
html	https://pypi.org/project/html/
TensorFlow 2	https://pypi.org/project/tensorflow/
Keras	https://pypi.org/project/Keras/
Pillow	https://pypi.org/project/Pillow/
Imageio	https://pypi.org/project/imageio/
Pathlib	https://pypi.org/project/pathlib/
OpenCV-Python	https://pypi.org/project/opencv-python/
Google Dialogflow	https://dialogflow.com/
DEAP	https://pypi.org/project/deap/
bitstring	https://pypi.org/project/bitstring/
nengo	https://pypi.org/project/nengo/
nengo-gui	https://pypi.org/project/nengo-gui/
IBM Q	https://www.research.ibm.com/ibm-q/
Quirk	http://algassert.com/2016/05/22/quirk.html

Download the example code files

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

You can download the code files by following these steps:

1. Log in or register at <http://www.packt.com>.
2. Select the **Support** tab.
3. Click on **Code Downloads**.
4. Enter the name of the book in the **Search** box and follow the 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/Artificial-Intelligence-By-Example-Second-Edition>. In case there's an update to the code, it will be updated on the existing GitHub repository.

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

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: https://static.packt-cdn.com/downloads/9781839211539_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; "The decision tree program, `decision_tree.py`, reads the output of the KMC predictions, `ckmc.csv`."

A block of code is set as follows:

```
# load dataset
col_names = ['f1', 'f2', 'label']
df = pd.read_csv("ckmc.csv", header=None, names=col_names)
if pp==1:
    print(df.head())
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
for i in range(0,1000):
    xf1=dataset.at[i,'Distance']
    xf2=dataset.at[i,'location']
    X_DL = [[xf1,xf2]]
    prediction = kmeans.predict(X_DL)
```

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

```
Selection: BnVYkFcRK Fittest: 0 This generation Fitness: 0 Time
Difference: 0:00:00.000198
```

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: "When you click on **SAVE**, the **Emotions** progress bar will jump up."



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

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

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

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

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

Reviews

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

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

1

Getting Started with Next-Generation Artificial Intelligence through Reinforcement Learning

Next-generation AI compels us to realize that machines do indeed think. Although machines do not think like us, their thought process has proven its efficiency in many areas. In the past, the belief was that AI would reproduce human thinking processes. Only neuromorphic computing (see *Chapter 18, Neuromorphic Computing*), remains set on this goal. Most AI has now gone beyond the way humans think, as we will see in this chapter.

The **Markov decision process (MDP)**, a **reinforcement learning (RL)** algorithm, perfectly illustrates how machines have become intelligent in their own unique way. Humans build their decision process on experience. MDPs are memoryless. Humans use logic and reasoning to think problems through. MDPs apply random decisions 100% of the time. Humans think in words, labeling everything they perceive. MDPs have an unsupervised approach that uses no labels or training data. MDPs boost the machine thought process of self-driving cars (SDCs), translation tools, scheduling software, and more. This memoryless, random, and unlabeled machine thought process marks a historical change in the way a former human problem was solved.

With this realization comes a yet more mind-blowing fact. AI algorithms and hybrid solutions built on IoT, for example, have begun to surpass humans in strategic areas. Although AI cannot replace humans in every field, AI combined with classical automation now occupies key domains: banking, marketing, supply chain management, scheduling, and many other critical areas.

As you will see, starting with this chapter, you can occupy a central role in this new world as an adaptive thinker. You can design AI solutions and implement them. There is no time to waste. In this chapter, we are going to dive quickly and directly into reinforcement learning through the MDP.

Today, AI is essentially mathematics translated into source code, which makes it difficult to learn for traditional developers. However, we will tackle this approach pragmatically.

The goal here is not to take the easy route. We're striving to break complexity into understandable parts and confront them with reality. You are going to find out right from the outset how to apply an adaptive thinker's process that will lead you from an idea to a solution in reinforcement learning, and right into the center of gravity of the next generation of AI.

Reinforcement learning concepts

AI is constantly evolving. The classical approach states that:

- AI covers all domains
- Machine learning is a subset of AI, with clustering, classification, regression, and reinforcement learning
- Deep learning is a subset of machine learning that involves neural networks

However, these domains often overlap and it's difficult to fit neuromorphic computing, for example, with its sub-symbolic approach, into these categories (see *Chapter 18, Neuromorphic Computing*).

In this chapter, RL clearly fits into machine learning. Let's have a brief look into the scientific foundations of the MDP, the RL algorithm we are going to explore. The main concepts to keep in mind are the following:

- **Optimal transport:** In 1781, Gaspard Monge defined transport optimizing from one location to another using the shortest and most cost-effective path; for example, mining coal and then using the most cost-effective path to a factory. This was subsequently generalized to any form of path from point A to point B.

- **Boltzmann equation and constant:** In the late 19th century, Ludwig Boltzmann changed our vision of the world with his probabilistic distribution of particles beautifully summed up in his entropy formula:

$$S = k * \log W$$

S represents the entropy (energy, disorder) of a system expressed. k is the Boltzmann constant, and W represents the number of microstates. We will explore Boltzmann's ideas further in *Chapter 14, Preparing the Input of Chatbots with Restricted Boltzmann Machines (RBMs) and Principal Component Analysis (PCA)*.

- **Probabilistic distributions advanced further:** Josiah Willard Gibbs took the probabilistic distributions of large numbers of particles a step further. At that point, probabilistic information theory was advancing quickly. At the turn of the 19th century, Andrey Markov applied probabilistic algorithms to language, among other areas. A modern era of information theory was born.
- **When Boltzmann and optimal transport meet:** 2011 Fields Medal winner, Cédric Villani, brought Boltzmann's equation to yet another level. Villani then went on to unify optimal transport and Boltzmann. Cédric Villani proved something that was somewhat intuitively known to 19th century mathematicians but required proof.

Let's take all of the preceding concepts and materialize them in a real-world example that will explain why reinforcement learning using the MDP, for example, is so innovative.

Analyzing the following cup of tea will take you right into the next generation of AI:



Figure 1.1: Consider a cup of tea

You can look at this cup of tea in two different ways:

1. **Macrostates:** You look at the cup and content. You can see the volume of tea in the cup and you could feel the temperature when holding the cup in your hand.
2. **Microstates:** But can you tell how many molecules are in the tea, which ones are hot, warm, or cold, their velocity and directions? Impossible right?

Now, imagine, the tea contains 2,000,000,000+ Facebook accounts, or 100,000,000+ Amazon Prime users with millions of deliveries per year. At this level, we simply abandon the idea of controlling every item. We work on trends and probabilities.

Boltzmann provides a probabilistic approach to the evaluation of the features of our real world. Materializing Boltzmann in logistics through optimal transport means that the temperature could be the ranking of a product, the velocity can be linked to the distance to delivery, and the direction could be the itineraries we will study in this chapter.

Markov picked up the ripe fruits of microstate probabilistic descriptions and applied it to his MDP. Reinforcement learning takes the huge volume of elements (particles in a cup of tea, delivery locations, social network accounts) and defines the probable paths they take.

The turning point of human thought occurred when we simply could not analyze the state and path of the huge volumes facing our globalized world, which generates images, sounds, words, and numbers that exceed traditional software approaches.

With this in mind, we can start exploring the MDP.

How to adapt to machine thinking and become an adaptive thinker

Reinforcement learning, one of the foundations of machine learning, supposes learning through trial and error by interacting with an environment. This sounds familiar, doesn't it? That is what we humans do all our lives – in pain! Try things, evaluate, and then continue; or try something else.

In real life, you are the agent of your thought process. In reinforcement learning, the agent is the function calculating randomly through this trial-and-error process. This thought process function in machine learning is the MDP agent. This form of empirical learning is sometimes called Q-learning.

Mastering the theory and implementation of an MDP through a three-step method is a prerequisite.

This chapter will detail the three-step approach that will turn you into an AI expert, in general terms:

1. Starting by describing a problem to solve with real-life cases
2. Then, building a mathematical model that considers real-life limitations
3. Then, writing source code or using a cloud platform solution

This is a way for you to approach any project with an adaptive attitude from the outset. This shows that a human will always be at the center of AI by explaining how we can build the inputs, run an algorithm, and use the results of our code. Let's consider this three-step process and put it into action.

Overcoming real-life issues using the three-step approach

The key point of this chapter is to avoid writing code that will never be used. First, begin by understanding the subject as a subject matter expert. Then, write the analysis with words and mathematics to make sure your reasoning reflects the subject and, most of all, that the program will make sense in real life. Finally, in step 3, only write the code when you are sure about the whole project.

Too many developers start writing code without stopping to think about how the results of that code are going to manifest themselves within real-life situations. You could spend weeks developing the perfect code for a problem, only to find out that an external factor has rendered your solution useless. For instance, what if you coded a solar-powered robot to clear snow from the yard, only to discover that during winter, there isn't enough sunlight to power the robot!

In this chapter, we are going to tackle the MDP (Q function) and apply it to reinforcement learning with the Bellman equation. We are going to approach it a little differently to most, however. We'll be thinking about practical application, not simply code execution. You can find tons of source code and examples on the web. The problem is, much like our snow robot, such source code rarely considers the complications that come about in real-life situations. Let's say you find a program that finds the optimal path for a drone delivery. There's an issue, though; it has many limits that need to be overcome due to the fact that the code has not been written with real-life practicality in mind. You, as an adaptive thinker, are going to ask some questions:

- What if there are 5,000 drones over a major city at the same time? What happens if they try to move in straight lines and bump into each other?

- Is a drone-jam legal? What about the noise over the city? What about tourism?
- What about the weather? Weather forecasts are difficult to make, so how is this scheduled?
- How can we resolve the problem of coordinating the use of charging and parking stations?

In just a few minutes, you will be at the center of attention among theoreticians who know more than you, on one hand, and angry managers who want solutions they cannot get on the other. Your real-life approach will solve these problems. To do that, you must take the following three steps into account, starting with really getting involved in the real-life subject.

In order to successfully implement our real-life approach, comprised of the three steps outlined in the previous section, there are a few prerequisites:

- **Be a subject matter expert (SME):** First, you have to be an SME. If a theoretician geek comes up with a hundred TensorFlow functions to solve a drone trajectory problem, you now know it is going to be a tough ride in which real-life parameters are constraining the algorithm. An SME knows the subject and thus can quickly identify the critical factors of a given field. AI often requires finding a solution to a complex problem that even an expert in a given field cannot express mathematically. Machine learning sometimes means finding a solution to a problem that humans do not know how to explain. Deep learning, involving complex networks, solves even more difficult problems.
- **Have enough mathematical knowledge to understand AI concepts:** Once you have the proper natural language analysis, you need to build your abstract representation quickly. The best way is to look around and find an everyday life example and make a mathematical model of it. Mathematics is not an option in AI, but a prerequisite. The effort is worthwhile. Then, you can start writing a solid piece of source code or start implementing a cloud platform ML solution.
- **Know what source code is about as well as its potential and limits:** MDP is an excellent way to go and start working on the three dimensions that will make you adaptive: describing what is around you in detail in words, translating that into mathematical representations, and then implementing the result in your source code.

With those prerequisites in mind, let's look at how you can become a problem-solving AI expert by following our practical three-step process. Unsurprisingly, we'll begin at step 1.

Step 1 – describing a problem to solve: MDP in natural language

Step 1 of any AI problem is to go as far as you can to understand the subject you are asked to represent. If it's a medical subject, don't just look at data; go to a hospital or a research center. If it's a private security application, go to the places where they will need to use it. If it's for social media, make sure to talk to many users directly. The key concept to bear in mind is that you have to get a "feel" for the subject, as if you were the real "user."

For example, transpose it into something you know in your everyday life (work or personal), something you are an SME in. If you have a driver's license, then you are an SME of driving. You are certified. This is a fairly common certification, so let's use this as our subject matter in the example that will follow. If you do not have a driver's license or never drive, you can easily replace moving around in a car by imagining you are moving around on foot; you are an SME of getting from one place to another, regardless of what means of transport that might involve. However, bear in mind that a real-life project would involve additional technical aspects, such as traffic regulations for each country, so our imaginary SME does have its limits.

Getting into the example, let's say you are an e-commerce business driver delivering a package in a location you are unfamiliar with. You are the operator of a self-driving vehicle. For the time being, you're driving manually. You have a GPS with a nice color map on it. The locations around you are represented by the letters **A** to **F**, as shown in the simplified map in the following diagram. You are presently at **F**. Your goal is to reach location **C**. You are happy, listening to the radio. Everything is going smoothly, and it looks like you are going to be there on time. The following diagram represents the locations and routes that you can cover:

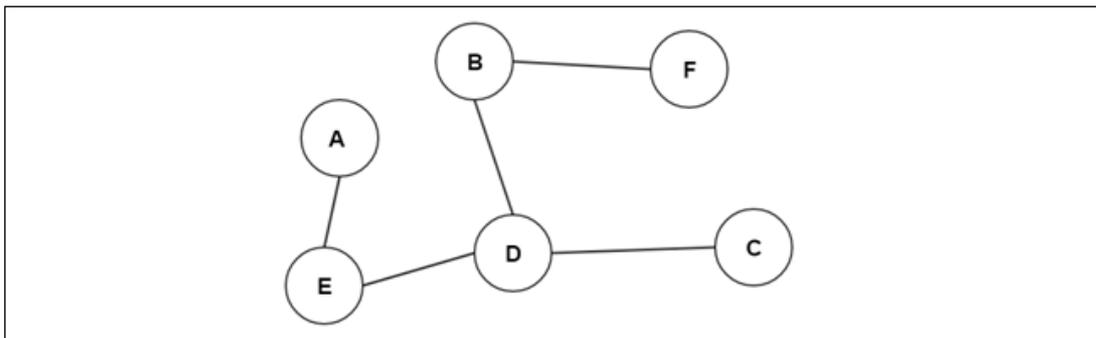


Figure 1.2: A diagram of delivery routes

The guidance system's state indicates the complete path to reach **C**. It is telling you that you are going to go from **F** to **B** to **D**, and then to **C**. It looks good!

To break things down further, let's say:

- The present state is the letter s . s is a variable, not an actual state. It can be one of the locations in L , the set of locations:

$$L = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}\}$$

We say *present state* because there is no sequence in the learning process. The memoryless process goes from one present state to another. In the example in this chapter, the process starts at location \mathbf{F} .

- Your next action is the letter a (action). This action a is not location \mathbf{A} . The goal of this action is to take us to the next possible location in the graph. In this case, only \mathbf{B} is possible. The goal of a is to take us from s (present state) to s' (new state).
- The action a (not location \mathbf{A}) is to go to location \mathbf{B} . You look at your guidance system; it tells you there is no traffic, and that to go from your present state, \mathbf{F} , to your next state, \mathbf{B} , will take you only a few minutes. Let's say that the next state \mathbf{B} is the letter \mathbf{B} . This next state \mathbf{B} is s' .

At this point, you are still quite happy, and we can sum up your situation with the following sequence of events:

$$s, a, s'$$

The letter s is your present state, your present situation. The letter a is the action you're deciding, which is to go to the next location; there, you will be in another state, s' . We can say that thanks to the action a , you will go from s to s' .

Now, imagine that the driver is not you anymore. You are tired for some reason. That is when a self-driving vehicle comes in handy. You set your car to autopilot. Now, you are no longer driving; the system is. Let's call that system the **agent**. At point \mathbf{F} , you set your car to autopilot and let the self-driving agent take over.

Watching the MDP agent at work

The self-driving AI is now in charge of the vehicle. It is acting as the MDP agent. This now sees what you have asked it to do and checks its **mapping environment**, which represents all the locations in the previous diagram from \mathbf{A} to \mathbf{F} .

In the meantime, you are rightly worried. Is the agent going to make it or not? You are wondering whether its strategy meets yours. You have your **policy** P —your way of thinking—which is to take the shortest path possible. Will the agent agree? What's going on in its machine mind? You observe and begin to realize things you never noticed before.

Since this is the first time you are using this car and guidance system, the agent is **memoryless**, which is an MDP feature. The agent doesn't know anything about what went on before. It seems to be happy with just calculating from this state s at location **F**. It will use machine power to run as many calculations as necessary to reach its goal.

Another thing you are watching is the total distance from **F** to **C** to check whether things are OK. That means that the agent is calculating all the states from **F** to **C**.

In this case, state **F** is state 1, which we can simplify by writing s_1 ; **B** is state 2, which we can simplify by writing s_2 ; **D** is s_3 ; and **C** is s_4 . The agent is calculating all of these possible states to make a decision.

The agent knows that when it reaches **D**, **C** will be better because the reward will be higher for going to **C** than anywhere else. Since it cannot eat a piece of cake to reward itself, the agent uses numbers. Our agent is a real number cruncher. When it is wrong, it gets a poor reward or nothing in this model. When it's right, it gets a reward represented by the letter R , which we'll encounter during step 2. This action-value (reward) transition, often named the Q function, is the core of many reinforcement learning algorithms.

When our agent goes from one state to another, it performs a *transition* and gets a reward. For example, the transition can be from **F** to **B**, state 1 to state 2, or s_1 to s_2 .

You are feeling great and are going to be on time. You are beginning to understand how the machine learning agent in your self-driving car is thinking. Suddenly, you look up and see that a traffic jam is building up. Location **D** is still far away, and now you do not know whether it would be good to go from **D** to **C** or **D** to **E**, in order to take another road to **C**, which involves less traffic. You are going to see what your agent thinks!

The agent takes the traffic jam into account, is stubborn, and increases its reward to get to **C** by the shortest way. Its policy is to stick to the initial plan. You do not agree. You have another policy.

You stop the car. You both have to agree before continuing. You have your opinion and policy; the agent does not agree. Before continuing, your views need to **converge**. **Convergence** is the key to making sure that your calculations are correct, and it's a way to evaluate the quality of a calculation.

A mathematical representation is the best way to express this whole process at this point, which we will describe in the following step.

Step 2 – building a mathematical model: the mathematical representation of the Bellman equation and MDP

Mathematics involves a whole change in your perspective of a problem. You are going from words to functions, the pillars of source coding.

Expressing problems in mathematical notation does not mean getting lost in academic math to the point of never writing a single line of code. Just use mathematics to get a job done efficiently. Skipping mathematical representation will fast-track a few functions in the early stages of an AI project. However, when the real problems that occur in all AI projects surface, solving them with source code alone will prove virtually impossible. The goal here is to pick up enough mathematics to implement a solution in real-life companies.

It is necessary to think through a problem by finding something familiar around us, such as the itinerary model covered early in this chapter. It is a good thing to write it down with some abstract letters and symbols as described before, with a meaning an action, and s meaning a state. Once you have understood the problem and expressed it clearly, you can proceed further.

Now, mathematics will help to clarify the situation by means of shorter descriptions. With the main ideas in mind, it is time to convert them into equations.

From MDP to the Bellman equation

In step 1, the agent went from **F**, or state 1 or s , to **B**, which was state 2 or s' .

A strategy drove this decision—a policy represented by P . One mathematical expression contains the MDP state transition function:

$$P_a(s, s')$$

P is the policy, the strategy made by the agent to go from **F** to **B** through action a . When going from **F** to **B**, this state transition is named the **state transition function**:

- a is the action
- s is state 1 (**F**), and s' is state 2 (**B**)

The reward (right or wrong) matrix follows the same principle:

$$R_a(s, s')$$

That means R is the reward for the action of going from state s to state s' . Going from one state to another will be a random process. Potentially, all states can go to any other state.

Each line in the matrix in the example represents a letter from **A** to **F**, and each column represents a letter from **A** to **F**. All possible states are represented. The 1 values represent the nodes (vertices) of the graph. Those are the possible locations. For example, line 1 represents the possible moves for letter **A**, line 2 for letter **B**, and line 6 for letter **F**. On the first line, **A** cannot go to **C** directly, so a 0 value is entered. But, it can go to **E**, so a 1 value is added.

Some models start with -1 for impossible choices, such as **B** going directly to **C**, and 0 values to define the locations. This model starts with 0 and 1 values. It sometimes takes weeks to design functions that will create a reward matrix (see *Chapter 2, Building a Reward Matrix – Designing Your Datasets*).

The example we will be working on inputs a reward matrix so that the program can choose its best course of action. Then, the agent will go from state to state, learning the best trajectories for every possible starting location point. The goal of the MDP is to go to **C** (line 3, column 3 in the reward matrix), which has a starting value of 100 in the following Python code:

```
# Markov Decision Process (MDP) - The Bellman equations adapted to
# Reinforcement Learning
import numpy as ql
# R is The Reward Matrix for each state
R = ql.matrix([ [0,0,0,0,1,0],
                [0,0,0,1,0,1],
                [0,0,100,1,0,0],
                [0,1,1,0,1,0],
                [1,0,0,1,0,0],
                [0,1,0,0,0,0] ])
```

Somebody familiar with Python might wonder why I used `ql` instead of `np`. Some might say "convention," "mainstream," "standard." My answer is a question. Can somebody define what "standard" AI is in this fast-moving world! My point here for the MDP is to use `ql` as an abbreviation of "Q-learning" instead of the "standard" abbreviation of NumPy, which is `np`. Naturally, beyond this special abbreviation for the MDP programs, I'll use `np`. Just bear in mind that conventions are there to break so as to set ourselves free to explore new frontiers. Just make sure your program works well!

There are several key properties of this decision process, among which there is the following:

- **The Markov property:** The process does not take the past into account. It is the memoryless property of this decision process, just as you do in a car with a guidance system. You move forward to reach your goal.
- **Unsupervised learning:** From this memoryless Markov property, it is safe to say that the MDP is not supervised learning. Supervised learning would mean that we would have all the labels of the reward matrix R and learn from them. We would know what A means and use that property to make a decision. We would, in the future, be looking at the past. MDP does not take these labels into account. Thus, MDP uses unsupervised learning to train. A decision has to be made in each state without knowing the past states or what they signify. It means that the car, for example, was on its own at each location, which is represented by each of its states.
- **Stochastic process:** In step 1, when state D was reached, the agent controlling the mapping system and the driver didn't agree on where to go. A random choice could be made in a trial-and-error way, just like a coin toss. It is going to be a heads-or-tails process. The agent will toss the coin a significant number of times and measure the outcomes. That's precisely how MDP works and how the agent will learn.
- **Reinforcement learning:** Repeating a trial-and-error process with feedback from the agent's environment.
- **Markov chain:** The process of going from state to state with no history in a random, stochastic way is called a Markov chain.

To sum it up, we have three tools:

- $P_a(s, s')$: A **policy**, P , or strategy to move from one state to another
- $T_a(s, s')$: A T , or stochastic (random) **transition**, function to carry out that action
- $R_a(s, s')$: An R , or **reward**, for that action, which can be negative, null, or positive

T is the transition function, which makes the agent decide to go from one point to another with a policy. In this case, it will be random. That's what machine power is for, and that is how reinforcement learning is often implemented.

Randomness

Randomness is a key property of MDP, defining it as a stochastic process.

The following code describes the choice the **agent** is going to make:

```
next_action = int(ql.random.choice(PossibleAction,1))
return next_action
```

The code selects a new random action (state) at each episode.

The Bellman equation

The Bellman equation is the road to programming reinforcement learning.

The Bellman equation completes the MDP. To calculate the value of a state, let's use Q , for the Q action-reward (or value) function. The pseudo source code of the Bellman equation can be expressed as follows for one individual state:

$$Q(s) = R(s) + \gamma * \max(s')$$

The source code then translates the equation into a machine representation, as in the following code:

```
# The Bellman equation
Q[current_state, action] = R[current_state, action] +
    gamma * MaxValue
```

The source code variables of the Bellman equation are as follows:

- $Q(s)$: This is the value calculated for this state—the total reward. In step 1, when the agent went from **F** to **B**, the reward was a number such as 50 or 100 to show the agent that it's on the right track.
- $R(s)$: This is the sum of the values up to that point. It's the total reward at that point.
- $\gamma = \mathbf{gamma}$: This is here to remind us that trial and error has a price. We're wasting time, money, and energy. Furthermore, we don't even know whether the next step is right or wrong since we're in a trial-and-error mode. **Gamma** is often set to 0.8. What does that mean? Suppose you're taking an exam. You study and study, but you don't know the outcome. You might have 80 out of 100 (0.8) chances of clearing it. That's painful, but that's life. The **gamma** penalty, or learning rate, makes the Bellman equation realistic and efficient.
- $\max(s')$: s' is one of the possible states that can be reached with $P_a(s, s')$; \max is the highest value on the line of that state (location line in the reward matrix).

At this point, you have done two-thirds of the job: understanding the real-life (process) and representing it in basic mathematics. You've built the mathematical model that describes your learning process, and you can implement that solution in code. Now, you are ready to code!

Step 3 – writing source code: implementing the solution in Python

In step 1, a problem was described in natural language to be able to talk to experts and understand what was expected. In step 2, an essential mathematical bridge was built between natural language and source coding. Step 3 is the software implementation phase.

When a problem comes up – and rest assured that one always does – it will be possible to go back over the mathematical bridge with the customer or company team, and even further back to the natural language process if necessary.

This method guarantees success for any project. The code in this chapter is in Python 3.x. It is a reinforcement learning program using the Q function with the following reward matrix:

```
import numpy as ql
R = ql.matrix([ [0,0,0,0,1,0],
                [0,0,0,1,0,1],
                [0,0,100,1,0,0],
                [0,1,1,0,1,0],
                [1,0,0,1,0,0],
                [0,1,0,0,0,0] ])

Q = ql.matrix(ql.zeros([6,6]))

gamma = 0.8
```

R is the reward matrix described in the mathematical analysis.

Q inherits the same structure as R, but all values are set to 0 since this is a learning matrix. It will progressively contain the results of the decision process. The gamma variable is a double reminder that the system is learning and that its decisions have only an 80% chance of being correct each time. As the following code shows, the system explores the possible actions during the process:

```
agent_s_state = 1

# The possible "a" actions when the agent is in a given state
```

```

def possible_actions(state):
    current_state_row = R[state,]
    possible_act = ql.where(current_state_row >0)[1]
    return possible_act

# Get available actions in the current state
PossibleAction = possible_actions(agent_s_state)

```

The agent starts in state 1, for example. You can start wherever you want because it's a random process. Note that the process only takes values > 0 into account. They represent possible moves (decisions).

The current state goes through an analysis process to find possible actions (next possible states). You will note that there is no algorithm in the traditional sense with many rules. It's a pure random calculation, as the following `random.choice` function shows:

```

def ActionChoice(available_actions_range):
    if (sum(PossibleAction)>0):
        next_action = int(ql.random.choice(PossibleAction,1))
    if (sum(PossibleAction)<=0):
        next_action = int(ql.random.choice(5,1))
    return next_action

# Sample next action to be performed
action = ActionChoice(PossibleAction)

```

Now comes the core of the system containing the Bellman equation, translated into the following source code:

```

def reward(current_state, action, gamma):
    Max_State = ql.where(Q[action,] == ql.max(Q[action,]))[1]

    if Max_State.shape[0] > 1:
        Max_State = int(ql.random.choice(Max_State, size = 1))
    else:
        Max_State = int(Max_State)
    MaxValue = Q[action, Max_State]

    # Q function
    Q[current_state, action] = R[current_state, action] +
        gamma * MaxValue

# Rewarding Q matrix
reward(agent_s_state,action,gamma)

```

You can see that the agent looks for the maximum value of the next possible state chosen at random.

The best way to understand this is to run the program in your Python environment and `print()` the intermediate values. I suggest that you open a spreadsheet and note the values. This will give you a clear view of the process.

The last part is simply about running the learning process 50,000 times, just to be sure that the system learns everything there is to find. During each iteration, the agent will detect its present state, choose a course of action, and update the Q function matrix:

```
for i in range(50000):
    current_state = q1.random.randint(0, int(Q.shape[0]))
    PossibleAction = possible_actions(current_state)
    action = ActionChoice(PossibleAction)
    reward(current_state, action, gamma)

# Displaying Q before the norm of Q phase
print("Q :")
print(Q)

# Norm of Q
print("Normed Q :")
print(Q/q1.max(Q)*100)
```

The process continues until the learning process is over. Then, the program will print the result in Q and the normed result. The normed result is the process of dividing all values by the sum of the values found. `print(Q/q1.max(Q)*100)` norms Q by dividing Q by `q1.max(Q)*100`. The result comes out as a normed percentage.

You can run the process with `mdp01.py`.

The lessons of reinforcement learning

Unsupervised reinforcement machine learning, such as the MDP-driven Bellman equation, is toppling traditional decision-making software location by location. Memoryless reinforcement learning requires few to no business rules and, thus, doesn't require human knowledge to run.

Being an adaptive next-generation AI thinker involves three prerequisites: the effort to be an SME, working on mathematical models to think like a machine, and understanding your source code's potential and limits.

Machine power and reinforcement learning teach us two important lessons:

- **Lesson 1:** Machine learning through reinforcement learning can beat human intelligence in many cases. No use fighting! The technology and solutions are already here in strategic domains.
- **Lesson 2:** A machine has no emotions, but you do. And so do the people around you. Human emotions and teamwork are an essential asset. Become an SME for your team. Learn how to understand what they're trying to say intuitively and make a mathematical representation of it for them. Your job will never go away, even if you're setting up solutions that don't require much development, such as AutoML. AutoML, or automated machine learning, automates many tasks. AutoML automates functions such as the dataset pipeline, hyperparameters, and more. Development is partially or totally suppressed. But you still have to make sure the whole system is well designed.

Reinforcement learning shows that no human can solve a problem the way a machine does. 50,000 iterations with random searching is not an option for a human. The number of empirical episodes can be reduced dramatically with a numerical convergence form of gradient descent (see *Chapter 3, Machine Intelligence – Evaluation Functions and Numerical Convergence*).

Humans need to be more intuitive, make a few decisions, and see what happens, because humans cannot try thousands of ways of doing something. Reinforcement learning marks a new era for human thinking by surpassing human reasoning power in strategic fields.

On the other hand, reinforcement learning requires mathematical models to function. Humans excel in mathematical abstraction, providing powerful intellectual fuel to those powerful machines.

The boundaries between humans and machines have changed. Humans' ability to build mathematical models and ever-growing cloud platforms will serve online machine learning services.

Finding out how to use the outputs of the reinforcement learning program we just studied shows how a human will always remain at the center of AI.

How to use the outputs

The reinforcement program we studied contains no trace of a specific field, as in traditional software. The program contains the Bellman equation with stochastic (random) choices based on the reward matrix. The goal is to find a route to C (line 3, column 3) that has an attractive reward (100):

```
# Markov Decision Process (MDP) - The Bellman equations adapted to
# Reinforcement Learning with the Q action-value(reward) matrix
import numpy as ql
# R is The Reward Matrix for each state
R = ql.matrix([ [0,0,0,0,1,0],
                [0,0,0,1,0,1],
                [0,0,100,1,0,0],
                [0,1,1,0,1,0],
                [1,0,0,1,0,0],
                [0,1,0,0,0,0] ])
```

That reward matrix goes through the Bellman equation and produces a result in Python:

```
Q :
[[ 0.  0.  0.  0. 258.44 0. ]
 [ 0.  0.  0. 321.8 0. 207.752]
 [ 0.  0. 500. 321.8 0.  0. ]
 [ 0. 258.44 401. 0. 258.44 0. ]
 [ 207.752 0. 0. 321.8 0. 0. ]
 [ 0. 258.44 0. 0. 0. 0. ]]
Normed Q :
[[ 0.  0.  0.  0. 51.688 0. ]
 [ 0.  0.  0. 64.36 0. 41.5504]
 [ 0.  0. 100. 64.36 0.  0. ]
 [ 0. 51.688 80.2 0. 51.688 0. ]
 [ 41.5504 0. 0. 64.36 0. 0. ]
 [ 0. 51.688 0. 0. 0. 0. ]]
```

The result contains the values of each state produced by the reinforced learning process, and also a normed Q (the highest value divided by other values).

As Python geeks, we are overjoyed! We made something that is rather difficult work, namely, reinforcement learning. As mathematical amateurs, we are elated. We know what MDP and the Bellman equation mean.

However, as natural language thinkers, we have made little progress. No customer or user can read that data and make sense of it. Furthermore, we cannot explain how we implemented an intelligent version of their job in the machine. We didn't.

We hardly dare say that reinforcement learning can beat anybody in the company, making random choices 50,000 times until the right answer came up.

Furthermore, we got the program to work, but hardly know what to do with the result ourselves. The consultant on the project cannot help because of the matrix format of the solution.

Being an adaptive thinker means knowing how to be good in all steps of a project. To solve this new problem, let's go back to step 1 with the result. Going back to step 1 means that if you have problems either with the results themselves or understanding them, it is necessary to go back to the SME level, the real-life situation, and see what is going wrong.

By formatting the result in Python, a graphics tool, or a spreadsheet, the result can be displayed as follows:

	A	B	C	D	E	F
A	-	-	-	-	258.44	-
B	-	-	-	321.8	-	207.752
C	-	-	500	321.8	-	-
D	-	258.44	401.	-	258.44	-
E	207.752	-	-	321.8	-	-
F	-	258.44	-	-	-	-

Now, we can start reading the solution:

- Choose a starting state. Take **F**, for example.
- The **F** line represents the state. Since the maximum value is 258.44 in the **B** column, we go to state **B**, the second line.
- The maximum value in state **B** in the second line leads us to the **D** state in the fourth column.
- The highest maximum of the **D** state (fourth line) leads us to the **C** state.

Note that if you start at the **C** state and decide not to stay at **C**, the **D** state becomes the maximum value, which will lead you back to **C**. However, the MDP will never do this naturally. You will have to force the system to do it.

You have now obtained a sequence: **F**->**B**->**D**->**C**. By choosing other points of departure, you can obtain other sequences by simply sorting the table.

A useful way of putting it remains the normalized version in percentages, as shown in the following table:

	A	B	C	D	E	F
A	-	-	-	-	51.68%	-
B	-	-	-	64.36%	-	41.55%
C	-	-	100%	64.36%	-	-
D	-	51.68%	80.2%	-	51.68%	-
E	41.55%	-	-	64.36%	-	-
F	-	51.68%	-	-	-	-

Now comes the very tricky part. We started the chapter with a trip on the road. But I made no mention of it in the results analysis.

An important property of reinforcement learning comes from the fact that we are working with a mathematical model that can be applied to anything. No human rules are needed. We can use this program for many other subjects without writing thousands of lines of code.

Possible use cases

There are many cases to which we could adapt our reinforcement learning model without having to change any of its details.

Case 1: optimizing a delivery for a driver, human or not

This model was described in this chapter.

Case 2: optimizing warehouse flows

The same reward matrix can apply to go from point F to C in a warehouse, as shown in the following diagram:

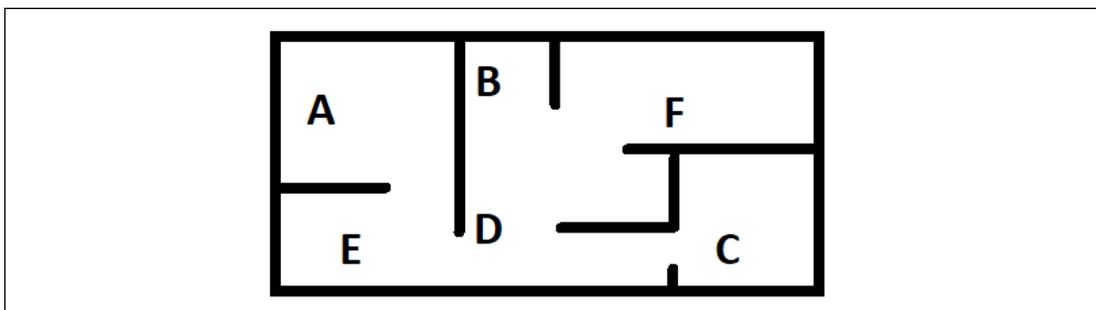


Figure 1.3: A diagram illustrating a warehouse flow problem

In this warehouse, the **F->B->D->C** sequence makes visual sense. If somebody goes from point **F** to **C**, then this physical path makes sense without going through walls.

It can be used for a video game, a factory, or any form of layout.

Case 3: automated planning and scheduling (APS)

By converting the system into a scheduling vector, the whole scenery changes. We have left the more comfortable world of physical processing of letters, faces, and trips. Though fantastic, those applications are social media's tip of the iceberg. The real challenge of AI begins in the abstract universe of human thinking.

Every single company, person, or system requires automatic planning and scheduling (see *Chapter 12, AI and the Internet of Things (IoT)*). The six **A** to **F** steps in the example of this chapter could well be six tasks to perform in a given unknown order represented by the following vector x :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

The reward matrix then reflects the weights of constraints of the tasks of vector x to perform. For example, in a factory, you cannot assemble the parts of a product before manufacturing them.

In this case, the sequence obtained represents the schedule of the manufacturing process.

Cases 4 and more: your imagination

By using physical layouts or abstract decision-making vectors, matrices, and tensors, you can build a world of solutions in a mathematical reinforcement learning model. Naturally, the following chapters will enhance your toolbox with many other concepts.

Before moving on, you might want to imagine some situations in which you could use the **A** to **F** letters to express some kind of path.

To help you with these mind experiment simulations, open `mdp02.py` and go to line 97, which starts with the following code that enables a simulation tool. `nextc` and `nextci` are simply variables to remember where the path begins and will end. They are set to `-1` so as to avoid `0`, which is a location.

The primary goal is to focus on the expression "concept code." The locations have become any concept you wish. A could be your bedroom, and C your kitchen. The path would go from where you wake up to where you have breakfast. A could be an idea you have, and F the end of a thinking process. The path would go from A (How can I hang this picture on the wall?) to E (I need to drill a hole) and, after a few phases, to F (I hung the picture on the wall). You can imagine thousands of paths like this as long as you define the reward matrix, the "concept code," and a starting point:

```
"""# Improving the program by introducing a decision-making process"""
nextc=-1
nextci=-1
conceptcode=["A", "B", "C", "D", "E", "F"]
```

This code takes the result of the calculation, labels the result matrix, and accepts an input as shown in the following code snippet:

```
origin=int(input(
    "index number origin(A=0,B=1,C=2,D=3,E=4,F=5): "))
```

The input only accepts the label numerical code: A=0, B=1 ... F=5. The function then runs a classical calculation on the results to find the best path. Let's take an example.

When you are prompted to enter a starting point, enter 5, for example, as follows:

```
index number origin(A=0,B=1,C=2,D=3,E=4,F=5): 5
```

The program will then produce the optimal path based on the output of the MDP process, as shown in the following output:

Concept Path

```
-> F
-> B
-> D
-> C
```

Try multiple scenarios and possibilities. Imagine what you could apply this to:

- An e-commerce website flow (visit, cart, checkout, purchase) imagining that a user visits the site and then resumes a session at a later time. You can use the same reward matrix and "concept code" explored in this chapter. For example, a visitor visits a web page at 10 a.m., starting at point A of your website. Satisfied with a product, the visitor puts the product in a cart, which is point E of your website. Then, the visitor leaves the site before going to the purchase page, which is C. D is the critical point. Why didn't the visitor purchase the product? What's going on?

You can decide to have an automatic email sent after 24 hours saying: "There is a 10% discount on all purchases during the next 48 hours." This way, you will target all the visitors stuck at D and push them toward C.

- A sequence of possible words in a sentence (subject, verb, object). Predicting letters and words was one of Andrey Markov's first applications 100+ years ago! You can imagine that B is the letter "a" of the alphabet. If D is "t," it is much more probable than F if F is "o," which is less probable in the English language. If an MDP reward matrix is built such as B leads to D or F, B can thus either go to D or to F. There are thus two possibilities, D or F. Andrey Markov would suppose, for example, that B is a variable that represents the letter "a," D is a variable that represents the letter "t" and F is a variable that represents the letter "o." After studying the structure of a language closely, he would find that the letter "a" would more likely be followed by "t" than by "o" in the English language. If one observes the English language, it is more likely to find an "a-t" sequence than an "a-o" sequence. In a Markov decision process, a higher probability will be awarded to the "a-t" sequence and a lower one to "a-o." If one goes back to the variables, the B-D sequence will come out as more probable than the B-F sequence.
- And anything you can find that fits the model that works is great!

Machine learning versus traditional applications

Reinforcement learning based on stochastic (random) processes will evolve beyond traditional approaches. In the past, we would sit down and listen to future users to understand their way of thinking.

We would then go back to our keyboard and try to imitate the human way of thinking. Those days are over. We need proper datasets and ML/DL equations to move forward. Applied mathematics has taken reinforcement learning to the next level. In my opinion, traditional software will soon be in the museum of computer science. The complexity of the huge volumes of data we are facing will require AI at some point.

An artificial adaptive thinker sees the world through applied mathematics translated into machine representations.

Use the Python source code example provided in this chapter in different ways. Run it and try to change some parameters to see what happens. Play around with the number of iterations as well. Lower the number from 50,000 down to where you find it fits best. Change the reward matrix a little to see what happens. Design your reward matrix trajectory. This can be an itinerary or decision-making process.