Book Collection

# Learning Path

# Python Reinforcement Learning

Solve complex real-world problems by mastering reinforcement learning algorithms using OpenAI Gym and TensorFlow

Sudharsan Ravichandiran, Sean Saito,
Rajalingappaa Shanmugamani and Yang Wenzhuo

Packt
www.packt.com

# Python Reinforcement Learning

Solve complex real-world problems by mastering reinforcement learning algorithms using OpenAI Gym and TensorFlow

**Sudharsan Ravichandiran**
**Sean Saito**
**Rajalingappaa Shanmugamani**
**Yang Wenzhuo**

Packt

**BIRMINGHAM - MUMBAI**

# Python Reinforcement Learning



First published: April 2019

Production reference: 1170419

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

ISBN 978-1-83864-977-7

www.packtpub.com

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

# Why subscribe?

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

# Packt.com

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

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

# Contributors

## About the authors

**Sudharsan Ravichandiran** is a data scientist, researcher, artificial intelligence enthusiast, and YouTuber (search for Sudharsan reinforcement learning). He completed his bachelors in information technology at Anna University. His area of research focuses on practical implementations of deep learning and reinforcement learning, which includes natural language processing and computer vision. He used to be a freelance web developer and designer and has designed award-winning websites. He is an open source contributor and loves answering questions on Stack Overflow.

**Sean Saito** is the youngest ever Machine Learning Developer at SAP and the first bachelor hired for the position. He currently researches and develops machine learning algorithms that automate financial processes. He graduated from Yale-NUS College in 2017 with a Bachelor of Science degree (with Honours), where he explored unsupervised feature extraction for his thesis. Having a profound interest in hackathons, Sean represented Singapore during Data Science Game 2016, the largest student data science competition. Before attending university in Singapore, Sean grew up in Tokyo, Los Angeles, and Boston.

*Writing this book is a daunting task for any 23-year-old, and hence I would like to thank many people who made this possible. My greatest words of gratitude belong to my mother and brother for giving me as much love, understanding, and guidance as anyone can fathom. Many thanks also goes to my closest friends and mentors, all from whom I've acquired much knowledge and wisdom, for their encouragement and advice.*

**Rajalingappaa Shanmugamani** is currently working as an Engineering Manager for a Deep learning team at Kairos. Previously, he worked as a Senior Machine Learning Developer at SAP, Singapore and worked at various startups in developing machine learning products. He has a Masters from Indian Institute of Technology—Madras. He has published articles in peer-reviewed journals and conferences and submitted applications for several patents in the area of machine learning. In his spare time, he coaches programming and machine learning to school students and engineers.

*I thank my spouse Ezhil, mom, dad, family and friends for their immense support. I thank all the teachers, colleagues and mentors from whom I have learned a lot. I thank the co-authors Wen and Sean making their contributions a pleasure to read. I thank the publishing team from Packt especially Snehal for encouraging at difficult times.*

**Yang Wenzhuo** works as a Data Scientist at SAP, Singapore. He got a bachelor's degree in computer science from Zhejiang University in 2011 and a Ph.D. in machine learning from the National University of Singapore in 2016. His research focuses on optimization in machine learning and deep reinforcement learning. He has published papers on top machine learning/computer vision conferences including ICML and CVPR, and operations research journals including Mathematical Programming.

# Packt is searching for authors like you

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

# Table of Contents

# Preface

**Reinforcement Learning** (**RL**) is the trending and most promising branch of artificial intelligence. This course will help you master not only the basic reinforcement learning algorithms but also the advanced deep reinforcement learning algorithms.

The course starts with an introduction to Reinforcement Learning followed by OpenAI Gym, and TensorFlow. You will then explore various RL algorithms and concepts, such as Markov Decision Process, Monte Carlo methods, and dynamic programming, including value and policy iteration. As you make your way through the book, you'll work on various datasets including image, text, and video. This example-rich guide will introduce you to deep reinforcement learning algorithms, such as Dueling DQN, DRQN, A3C, PPO, and TRPO. You will gain experience in several domains, including gaming, image processing, and physical simulations. You'll explore technologies such as TensorFlow and OpenAI Gym to implement deep learning reinforcement learning algorithms that also predict stock prices, generate natural language, and even build other neural networks. You will also learn about imagination-augmented agents, learning from human preference, DQfD, HER, and many more of the recent advancements in reinforcement learning.

By the end of the course, you will have all the knowledge and experience needed to implement reinforcement learning and deep reinforcement learning in your projects, and you will be all set to enter the world of artificial intelligence to solve various problems in real-life.

This Learning Path includes content from the following Packt products:

- Hands-On Reinforcement Learning with Python by Sudharsan Ravichandiran
- Python Reinforcement Learning Projects by Sean Saito, Yang Wenzhuo, and Rajalingappaa Shanmugamani

## Who this book is for

If you're a machine learning developer or deep learning enthusiast interested in artificial intelligence and want to learn about reinforcement learning and deep reinforcement learning from scratch, this Learning Path is for you. You will be all ready to build a better performing, automated, and optimized self-learning agent. Some knowledge of linear algebra, calculus, basic DL approaches, and Python will help you understand the concepts.

# What this book covers

`Chapter 1`, *Introduction to Reinforcement Learning*, helps us understand what reinforcement learning is and how it works. We will learn about various elements of reinforcement learning, such as agents, environments, policies, and models, and we will see different types of environments, platforms, and libraries used for reinforcement learning. Later in the chapter, we will see some of the applications of reinforcement learning.

`Chapter 2`, *Getting Started with OpenAI and TensorFlow*, helps us set up our machine for various reinforcement learning tasks. We will learn how to set up our machine by installing Anaconda, Docker, OpenAI Gym, Universe, and TensorFlow. Then we will learn how to simulate agents in OpenAI Gym, and we will see how to build a video game bot. We will also learn the fundamentals of TensorFlow and see how to use TensorBoard for visualizations.

`Chapter 3`, *The Markov Decision Process and Dynamic Programming*, starts by explaining what a Markov chain and a Markov process is, and then we will see how reinforcement learning problems can be modeled as Markov Decision Processes. We will also learn about several fundamental concepts, such as value functions, Q functions, and the Bellman equation. Then we will see what dynamic programming is and how to solve the frozen lake problem using value and policy iteration.

`Chapter 4`, *Gaming with Monte Carlo Methods*, explains Monte Carlo methods and different types of Monte Carlo prediction methods, such as first visit MC and every visit MC. We will also learn how to use Monte Carlo methods to play blackjack. Then we will explore different on-policy and off-policy Monte Carlo control methods.

`Chapter 5`, *Temporal Difference Learning*, covers temporal-difference (TD) learning, TD prediction, and TD off-policy and on-policy control methods such as Q learning and SARSA. We will also learn how to solve the taxi problem using Q learning and SARSA.

`Chapter 6`, *Multi-Armed Bandit Problem*, deals with one of the classic problems of reinforcement learning, the multi-armed bandit (MAB) or k-armed bandit problem. We will learn how to solve this problem using various exploration strategies, such as epsilon-greedy, softmax exploration, UCB, and Thompson sampling. Later in the chapter, we will see how to show the right ad banner to the user using MAB.

`Chapter 7`, *Playing Atari Games*, will get us creating our first deep RL algorithm to play ATARI games.

`Chapter 8`, *Atari Games with Deep Q Network*, covers one of the most widely used deep reinforcement learning algorithms, which is called the deep Q network (DQN). We will learn about DQN by exploring its various components, and then we will see how to build an agent to play Atari games using DQN. Then we will look at some of the upgrades to the DQN architecture, such as double DQN and dueling DQN.

`Chapter 9`, *Playing Doom with a Deep Recurrent Q Network*, explains the deep recurrent Q network (DRQN) and how it differs from a DQN. We will see how to build an agent to play Doom using a DRQN. Later in the chapter, we will learn about the deep attention recurrent Q network, which adds the attention mechanism to the DRQN architecture.

`Chapter 10`, *The Asynchronous Advantage Actor Critic Network*, explains how the Asynchronous Advantage Actor Critic (A3C) network works. We will explore the A3C architecture in detail, and then we will learn how to build an agent for driving up the mountain using A3C.

`Chapter 11`, *Policy Gradients and Optimization*, covers how policy gradients help us find the right policy without needing the Q function. We will also explore the deep deterministic policy gradient method. Later in the chapter, we will see state of the art policy optimization methods such as trust region policy optimization and proximal policy optimization.

`Chapter 12`, *Balancing CartPole*, will have us implement our first RL algorithms in Python and TensorFlow to solve the cart pole balancing problem.

`Chapter 13`, *Simulating Control Tasks*, provides a brief introduction to actor-critic algorithms for continuous control problems. We will learn how to simulate classic control tasks, look at how to implement basic actor-critic algorithms, and understand the state-of-the-art algorithms for control.

`Chapter 14`, *Building Virtual Worlds in Minecraft*, takes the advanced concepts covered in previous chapters and applies them to Minecraft, a game more complex than those found on ATARI.

`Chapter 15`, *Learning to Play Go*, will have us building a model that can play Go, the popular Asian board game that is considered one of the world's most complicated games.

`Chapter 16`, *Creating a Chatbot*, will teach us how to apply deep RL in natural language processing. Our reward function will be a future-looking function, and we will learn how to think in terms of probability when creating this function.

`Chapter 17`, *Generating a Deep Learning Image Classifier*, introduces one of the latest and most exciting advancements in RL: generating deep learning models using RL. We explore the cutting-edge research produced by Google Brain and implement the algorithms introduced.

`Chapter 18`, *Predicting Future Stock Prices*, discusses building an agent that can predict stock prices.

`Chapter 19`, *Capstone Project – Car Racing Using DQN*, provides a step-by-step approach for building an agent to win a car racing game using dueling DQN.

`Chapter 20`, *Looking Ahead*, concludes the book by discussing some of the real-world applications of reinforcement learning and introducing potential areas of future academic work.

# To get the most out of this book

The examples covered in this book can be run on Windows, Ubuntu, or macOS. All the installation instructions are covered. A basic knowledge of Python and machine learning is required. It's preferred that you have GPU hardware, but it's not necessary.

You need the following software for this book:

- Anaconda
- Python
- Any web browser
- Docker

# 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.packt.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

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

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

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

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Python-Reinforcement-Learning`. 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!

# Conventions used

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

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The `gym-minecraft` package has the same interface as other Gym environments."

A block of code is set as follows:

```
import logging
import minecraft_py
logging.basicConfig(level=logging.DEBUG)
```

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

```
python3 -m pip install gym
python3 -m pip install pygame
```

**Bold**: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an 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**: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

**Errata**: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.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@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
# Introduction to Reinforcement Learning

**Reinforcement learning** (**RL**) is a branch of machine learning where the learning occurs via interacting with an environment. It is goal-oriented learning where the learner is not taught what actions to take; instead, the learner learns from the consequence of its actions. It is growing rapidly with a wide variety of algorithms and it is one of the most active areas of research in **artificial intelligence** (**AI**).

In this chapter, you will learn about the following:

- Fundamental concepts of RL
- RL algorithm
- Agent environment interface
- Types of RL environments
- RL platforms
- Applications of RL

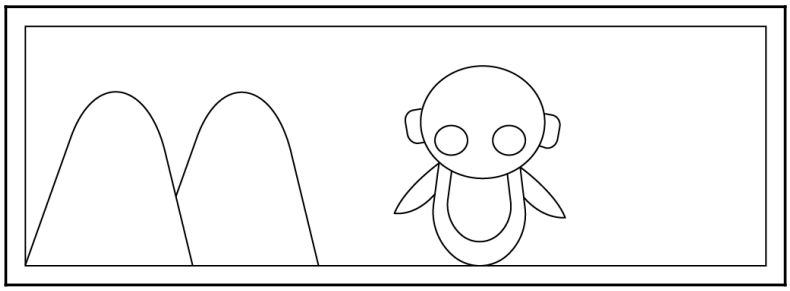
## What is RL?

Consider that you are teaching the dog to catch a ball, but you cannot teach the dog explicitly to catch a ball; instead, you will just throw a ball, and every time the dog catches the ball, you will give it a cookie. If it fails to catch the ball, you will not give a cookie. The dog will figure out what actions made it receive a cookie and will repeat those actions.

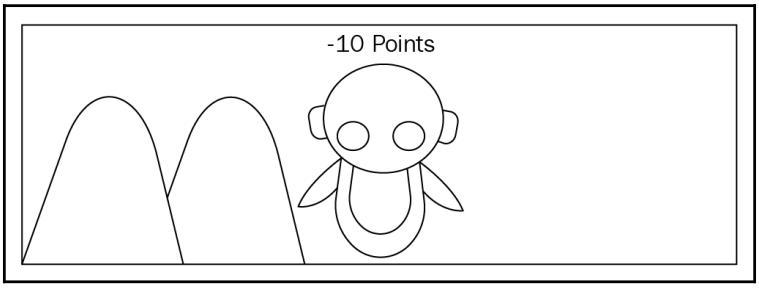
Similarly, in a RL environment, you will not teach the agent what to do or how to do instead, you will give a reward to the agent for each action it does. The reward may be positive or negative. Then the agent will start performing actions which made it receive a positive reward. Thus, it is a trial and error process. In the previous analogy, the dog represents the agent. Giving a cookie to the dog upon catching the ball is a positive reward, and not giving a cookie is a negative reward.

There might be delayed rewards. You may not get a reward at each step. A reward may be given only after the completion of a task. In some cases, you get a reward at each step to find out that whether you are making any mistakes.

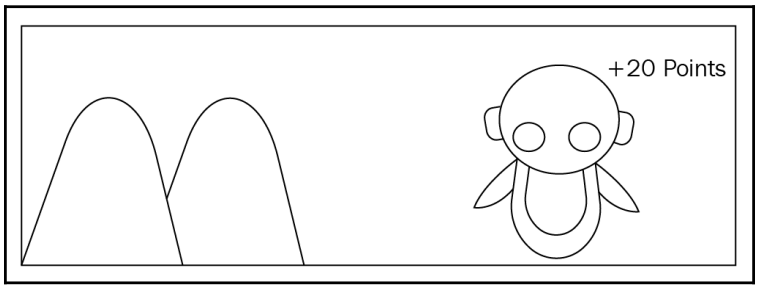
Imagine you want to teach a robot to walk without getting stuck by hitting a mountain, but you will not explicitly teach the robot not to go in the direction of the mountain:



Instead, if the robot hits and get stuck on the mountain, you will take away ten points so that robot will understand that hitting the mountain will result in a negative reward and it will not go in that direction again:



You will give 20 points to the robot when it walks in the right direction without getting stuck. So the robot will understand which is the right path and will try to maximize the rewards by going in the right direction:




The RL agent can **explore** different actions which might provide a good reward or it can **exploit** (use) the previous action which resulted in a good reward. If the RL agent explores different actions, there is a great possibility that the agent will receive a poor reward as all actions are not going to be the best one. If the RL agent exploits only the known best action, there is also a great possibility of missing out on the best action, which might provide a better reward. There is always a trade-off between exploration and exploitation. We cannot perform both exploration and exploitation at the same time. We will discuss the exploration-exploitation dilemma in detail in the upcoming chapters.

# RL algorithm

The steps involved in typical RL algorithm are as follows:

1. First, the agent interacts with the environment by performing an action
2. The agent performs an action and moves from one state to another
3. And then the agent will receive a reward based on the action it performed
4. Based on the reward, the agent will understand whether the action was good or bad
5. If the action was good, that is, if the agent received a positive reward, then the agent will prefer performing that action or else the agent will try performing an other action which results in a positive reward. So it is basically a trial and error learning process

# How RL differs from other ML paradigms

In supervised learning, the machine (agent) learns from training data which has a labeled set of input and output. The objective is that the model extrapolates and generalizes its learning so that it can be well applied to the unseen data. There is an external supervisor who has a complete knowledge base of the environment and supervises the agent to complete a task.

Consider the dog analogy we just discussed; in supervised learning, to teach the dog to catch a ball, we will teach it explicitly by specifying turn left, go right, move forward five steps, catch the ball, and so on. But instead in RL we just throw a ball, and every time the dog catches the ball, we give it a cookie (reward). So the dog will learn to catch the ball that meant it received a cookie.

In unsupervised learning, we provide the model with training data which only has a set of inputs; the model learns to determine the hidden pattern in the input. There is a common misunderstanding that RL is a kind of unsupervised learning but it is not. In unsupervised learning, the model learns the hidden structure whereas in RL the model learns by maximizing the rewards. Say we want to suggest new movies to the user. Unsupervised learning analyses the similar movies the person has viewed and suggests movies, whereas RL constantly receives feedback from the user, understands his movie preferences, and builds a knowledge base on top of it and suggests a new movie.

There is also another kind of learning called semi-supervised learning which is basically a combination of supervised and unsupervised learning. It involves function estimation on both the labeled and unlabeled data, whereas RL is essentially an interaction between the agent and its environment. Thus, RL is completely different from all other machine learning paradigms.

# Elements of RL

The elements of RL are shown in the following sections.

## Agent

Agents are the software programs that make intelligent decisions and they are basically learners in RL. Agents take action by interacting with the environment and they receive rewards based on their actions, for example, Super Mario navigating in a video game.

# Policy function

A policy defines the agent's behavior in an environment. The way in which the agent decides which action to perform depends on the policy. Say you want to reach your office from home; there will be different routes to reach your office, and some routes are shortcuts, while some routes are long. These routes are called policies because they represent the way in which we choose to perform an action to reach our goal. A policy is often denoted by the symbol $\pi$. A policy can be in the form of a lookup table or a complex search process.

# Value function

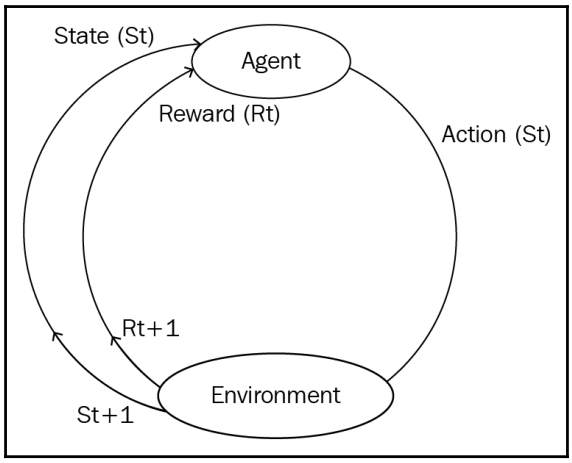
A value function denotes how good it is for an agent to be in a particular state. It is dependent on the policy and is often denoted by $v(s)$. It is equal to the total expected reward received by the agent starting from the initial state. There can be several value functions; the optimal value function is the one that has the highest value for all the states compared to other value functions. Similarly, an optimal policy is the one that has the optimal value function.

# Model

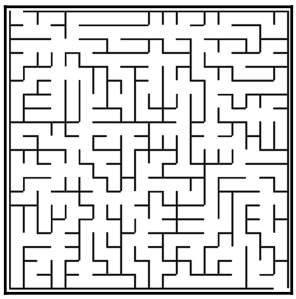
Model is the agent's representation of an environment. The learning can be of two types—model-based learning and model-free learning. In model-based learning, the agent exploits previously learned information to accomplish a task, whereas in model-free learning, the agent simply relies on a trial-and-error experience for performing the right action. Say you want to reach your office from home faster. In model-based learning, you simply use a previously learned experience (map) to reach the office faster, whereas in model-free learning you will not use a previous experience and will try all different routes and choose the faster one.

# Agent environment interface

Agents are the software agents that perform actions, $A_t$, at a time, $t$, to move from one state, $S_t$, to another state $S_{t+1}$. Based on actions, agents receive a numerical reward, $R$, from the environment. Ultimately, RL is all about finding the optimal actions that will increase the numerical reward:




Let us understand the concept of RL with a maze game:



The objective of a maze is to reach the destination without getting stuck on the obstacles. Here's the workflow:

- The agent is the one who travels through the maze, which is our software program/ RL algorithm
- The environment is the maze

- The state is the position in a maze that the agent currently resides in
- An agent performs an action by moving from one state to another
- An agent receives a positive reward when its action doesn't get stuck on any obstacle and receives a negative reward when its action gets stuck on obstacles so it cannot reach the destination
- The goal is to clear the maze and reach the destination

# Types of RL environment

Everything agents interact with is called an environment. The environment is the outside world. It comprises everything outside the agent. There are different types of environment, which are described in the next sections.

## Deterministic environment

An environment is said to be deterministic when we know the outcome based on the current state. For instance, in a chess game, we know the exact outcome of moving any player.

## Stochastic environment

An environment is said to be stochastic when we cannot determine the outcome based on the current state. There will be a greater level of uncertainty. For example, we never know what number will show up when throwing a dice.

## Fully observable environment

When an agent can determine the state of the system at all times, it is called fully observable. For example, in a chess game, the state of the system, that is, the position of all the players on the chess board, is available the whole time so the player can make an optimal decision.

# Partially observable environment

When an agent cannot determine the state of the system at all times, it is called partially observable. For example, in a poker game, we have no idea about the cards the opponent has.

# Discrete environment

When there is only a finite state of actions available for moving from one state to another, it is called a discrete environment. For example, in a chess game, we have only a finite set of moves.

# Continuous environment

When there is an infinite state of actions available for moving from one state to another, it is called a continuous environment. For example, we have multiple routes available for traveling from the source to the destination.

# Episodic and non-episodic environment

The episodic environment is also called the **non-sequential** environment. In an episodic environment, an agent's current action will not affect a future action, whereas in a non-episodic environment, an agent's current action will affect a future action and is also called the **sequential** environment. That is, the agent performs the independent tasks in the episodic environment, whereas in the non-episodic environment all agents' actions are related.

# Single and multi-agent environment

As the names suggest, a single-agent environment has only a single agent and the multi-agent environment has multiple agents. Multi-agent environments are extensively used while performing complex tasks. There will be different agents acting in completely different environments. Agents in a different environment will communicate with each other. A multi-agent environment will be mostly stochastic as it has a greater level of uncertainty.

# RL platforms

RL platforms are used for simulating, building, rendering, and experimenting with our RL algorithms in an environment. There are many different RL platforms available, as described in the next sections.

## OpenAI Gym and Universe

OpenAI Gym is a toolkit for building, evaluating, and comparing RL algorithms. It is compatible with algorithms written in any framework like TensorFlow, Theano, Keras, and so on. It is simple and easy to comprehend. It makes no assumption about the structure of our agent and provides an interface to all RL tasks.

OpenAI Universe is an extension to OpenAI Gym. It provides an ability to train and evaluate agents on a wide range of simple to real-time complex environments. It has unlimited access to many gaming environments. Using Universe, any program can be turned into a Gym environment without access to program internals, source code, or APIs as Universe works by launching the program automatically behind a virtual network computing remote desktop.

## DeepMind Lab

DeepMind Lab is another amazing platform for AI agent-based research. It provides a rich simulated environment that acts as a lab for running several RL algorithms. It is highly customizable and extendable. The visuals are very rich, science fiction-style, and realistic.

## RL-Glue

RL-Glue provides an interface for connecting agents, environments, and programs together even if they are written in different programming languages. It has the ability to share your agents and environments with others for building on top of your work. Because of this compatibility, reusability is greatly increased.

## Project Malmo

Project Malmo is the another AI experimentation platform from Microsoft which builds on top of Minecraft. It provides good flexibility for customizing the environment. It is integrated with a sophisticated environment. It also allows overclocking, which enables programmers to play out scenarios faster than in standard Minecraft. However, Malmo currently only provides Minecraft gaming environments, unlike Open AI Universe.

## ViZDoom

ViZDoom, as the name suggests, is a doom-based AI platform. It provides support for multi-agents and a competitive environment to test the agent. However, ViZDoom only supports the Doom game environment. It provides off-screen rendering and single and multiplayer support.

# Applications of RL

With greater advancements and research, RL has rapidly evolved everyday applications in several fields ranging from playing computer games to automating a car. Some of the RL applications are listed in the following sections.

## Education

Many online education platforms are using RL for providing personalized content for each and every student. Some students may learn better from video content, some may learn better by doing projects, and some may learn better from notes. RL is used to tune educational content personalized for each student according to their learning style and that can be changed dynamically according to the behavior of the user.

## Medicine and healthcare

RL has endless applications in medicine and health care; some of them include personalized medical treatment, diagnosis based on a medical image, obtaining treatment strategies in clinical decision making, medical image segmentation, and so on.

# Manufacturing

In manufacturing, intelligent robots are used to place objects in the right position. If it fails or succeeds in placing the object at the right position, it remembers the object and trains itself to do this with greater accuracy. The use of intelligent agents will reduce labor costs and result in better performance.

# Inventory management

RL is extensively used in inventory management, which is a crucial business activity. Some of these activities include supply chain management, demand forecasting, and handling several warehouse operations (such as placing products in warehouses for managing space efficiently). Google researchers in DeepMind have developed RL algorithms for efficiently reducing the energy consumption in their own data center.

# Finance

RL is widely used in financial portfolio management, which is the process of constant redistribution of a fund into different financial products and also in predicting and trading in commercial transactions markets. JP Morgan has successfully used RL to provide better trade execution results for large orders.

# Natural Language Processing and Computer Vision

With the unified power of deep learning and RL, **Deep Reinforcement Learning** (**DRL**) has been greatly evolving in the fields of **Natural Language Processing** (**NLP**) and **Computer Vision** (**CV**). DRL has been used for text summarization, information extraction, machine translation, and image recognition, providing greater accuracy than current systems.

# Summary

In this chapter, we have learned the basics of RL and also some key concepts. We learned different elements of RL and different types of RL environments. We also covered the various available RL platforms and also the applications of RL in various domains.

In `Chapter 2`, *Getting Started with OpenAI and TensorFlow*, we will learn the basics of and how to install OpenAI and TensorFlow, followed by simulating environments and teaching the agents to learn in the environment.

# Questions

The question list is as follows:

1. What is reinforcement learning?
2. How does RL differ from other ML paradigms?
3. What are agents and how do agents learn?
4. What is the difference between a policy function and a value function?
5. What is the difference between model-based and model-free learning?
6. What are all the different types of environments in RL?
7. How does OpenAI Universe differ from other RL platforms?
8. What are some of the applications of RL?

# Further reading

**Overview of RL**: `https://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html`.