Rafael Ris-Ala

Fundamentals of Reinforcement Learning

Springer

Fundamentals of Reinforcement Learning

Rafael Ris-Ala

Fundamentals of Reinforcement Learning

Springer

Rafael Ris-Ala
Computing Institute
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

ISBN 978-3-031-37344-2 ISBN 978-3-031-37345-9 (eBook)
https://doi.org/10.1007/978-3-031-37345-9

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

This book is dedicated to the new generations of software engineers who are enthusiastic about designing ever smarter systems for improving the quality of human life.

Preface

This book covers basic concepts of Artificial Intelligence, going through Machine Learning and deepening RL, in a theoretical and practical way.

Throughout the chapters, there are blocks of information in gray with a dashed line that presents links to tools and complementary information that work as tips and extend the resources of this book.

The organization of the book, at all times, goes from an overview to a more specific view. This also happens in each chapter. Thus, students can follow a sequential reading or refer to its chapters promptly, as follows:

Chapter 1 introduces the research area of Artificial Intelligence, as well as distinguishing between the various Machine Learning approaches and the types of problems they solve. The meaning of Reinforcement Learning is playfully introduced with examples, and its framework is explained. Then, relevant historical milestones that permeate several sciences and that have contributed to the development of this line of research are addressed.

Chapter 2 covers the fundamental knowledge needed to understand the entire system that involves Reinforcement Learning. Concepts such as agent, environment, actions, rewards, policies, and value function are discussed. Examples and analogies are presented to help illustrate each of these concepts, from structuring problems starting from the Markov Chain through Watkins and Dayan's proposal and unfolding in the Bellman Equation. Finally, the classes and particularities of algorithms that have been successful in this innovative field of research are presented.

Chapter 3 illustrates the step-by-step operation of one of the most widely used algorithms in Reinforcement Learning, the Q-Learning algorithm. The meaning of each component of the algorithm and its demonstration through pseudocode is presented. Then, a detailed explanation of how the algorithm works is described through a visual example of an agent interacting in an environment, from the initialization of the Q-Table to the agent's decision-making based on its experiences with the environment, through the construction of a policy to be followed.

Chapter 4 deals with practical tools for developing solutions in Reinforcement Learning. Some main libraries and frameworks are available for implementing RL

algorithms, such as TensorFlow, Keras, and OpenAI Gym. Some useful data sources for conducting your RL experiments are also discussed.

In Chap. 5, a practical case of developing an autonomous cab with AI in Python is proposed. The details of the environment are discussed, and the action of the agent without the use of AI is exemplified. As a counterpoint, next, how to implement an RL algorithm is demonstrated in a simplified way. The code is commented on and explained in detail, illustrating the differences and advantages of using RL in these types of problems. The system is made available for further testing and implementation.

Chapter 6 presents the most recent applications of how Reinforcement Learning is impacting various areas of knowledge. Examples of RL applications in areas such as robotics, games, education, and quantum mechanics are presented. The main advantages and challenges of RL applications in different fields are also discussed, as well as perspectives for the future of RL use in each of these areas.

Rio de Janeiro, Brazil Rafael Ris-Ala

Acknowledgments

The creation of this book also required similar minds.

I am grateful for the support of the entire academic group at the Instituto de Computação at the Universidade Federal do Rio de Janeiro (UFRJ), especially professors Adriana Vivacqua, Mônica da Silva, and Carla Delgado, who made this book possible.

To Bruna Santiago, for always being ready to help personally and technically and for all comprehension.

To Priscila de Medeiros, for her cognitive understanding and guidance.

To my family and friends for their deep assistance throughout this project.

To the entire editorial team at Springer Nature, and expertise, especially to Paul Drougas, Kritheka Elango, Thomas Hempfling, Celine Chang, Jorge Nakahara Jr., Lakshmanan Radha, and Andrea Gonçalves.

For initial support of Djonathan Quadras, from the Universidade Federal de Santa Catarina (UFSC), André Ottoni, from the Universidade Federal da Bahia (UFBA), Marcos dos Santos, from the Instituto Militar de Engenharia (IME), in addition to Francine de Oliveira, Luciana Freesz, Haroldo Santiago, and Marcelo Manhães.

To Faculdade XP Educação (XPe) for encouragement and trust.

To the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for supporting research and development of technological solutions in Brazil.

Contents

About the Author

Rafael Ris-Ala is a professor and researcher in Artificial Intelligence, Machine Learning, and Research Methodology at the Universidade Federal do Rio de Janeiro (UFRJ), at the Pontifícia Universidade Católica de Minas Gerais (PUC-Minas), and at the Faculdade XP Educação (XPe). He has a master's degree in Data Science from UFRJ and is currently pursuing his Ph.D. in Artificial Intelligence at the same institution.He is the author of several papers in Software Engineering. He has supervised more than 50 academic papers and evaluated more than 100 projects. He is a recognized journal reviewer for Elsevier and Clarivate and participates in the review of IEEE scientific papers.He worked as Infrastructure Project Manager at Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) and was also responsible for the creation of a Data Center. He has more than 10 years of experience in Software Development in the Brazilian Navy.

Abbreviations

ACKTR	Actor-Critic using Kronecker-Factored Trust Region
ALE	Arcade Learning Environment
API	Application Programming Interface
AI	Artificial Intelligence
ANI	Artificial Narrow Intelligence
ASI	Artificial Superintelligence
AGI	Artificial General Intelligence
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DMs	Diffusion Models
DP	Dynamic Programming
IoT	Internet of Things
ML	Machine Learning
MDP	Markov Decision Process
MCM	Monte Carlo Method
MuJoCo	Multi-Joint dynamics with Contact
NLP	Natural Language Processing
NPC	Non-Player Character
OR	Operations Research
RL	Reinforcement Learning
3DSIM	RoboCup 3D Soccer Simulation League
SC2LE	StarCraft II Learning Environment
SARSA	State-Action-Reward-State-Action

Chapter 1
Introduction

Reinforcement Learning (RL) technology has shown numerous successes in recent years in solving complex human tasks (Dulac-Arnold et al., 2019). Many robotic systems equipped with RL, such as aerial, aquatic, and terrestrial systems have been successful in solving challenging activities (Singh et al., 2022).

RL allows an agent to discover optimal behavior through trial-and-error interactions with its environment. An agent explores possible strategies and receives feedback on the outcome of the choices made (Kober et al., 2013). This is a technique that resembles the learning mechanism of the human brain (Singh et al., 2022).

A variety of societal challenges can be formulated as a sequential decision problem, such as a chess game, and solved as a Markov Decision Process (MDP) driven by RL. In these cases, the goal is to design a system that is intended for long-term strategic planning, estimating future rewards, in ways similar to how to obtain human skills needed to achieve success in real life, also keeping similarities with most games. In this way, games are a natural platform to validate these Artificial Intelligence (AI) projects. Notice in Fig. 1.1 how each action leads the player to a new decision state.

The advantage of using RL for solving complex problems lies in the possibility of discovering the best actions, even without knowing all the properties of the environment.

An AI-enabled robotic system is suited to act in scenarios that pose risks to human action or require a faster response time. They can be designed to have autonomy in rugged terrain, hard-to-reach places, or toxic atmospheres. An example can be observed in space exploration, as in the Mars reconnaissance missions that rely on the Perseverance rover, situations in which the robot needs to have the autonomy to decide its movements because round-trip commands issued by the ground base have communication delays and could make the whole mission unfeasible (Alibay et al., 2022).

It is thus possible to design autonomous robots that learn to balance and walk so that they can move like humans in diverse environments.

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_1

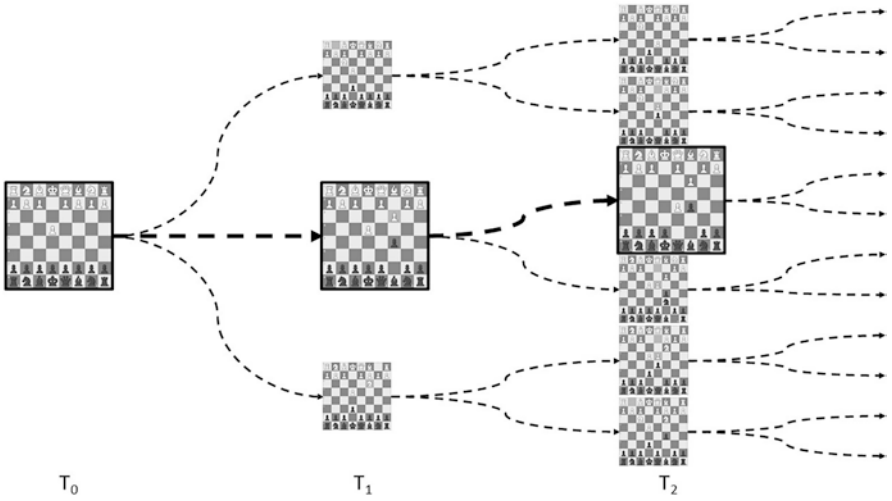


Fig. 1.1 With each action, the agent reaches a new state

The industry has sought to develop robots for these scenarios. Spot, formerly called SpotMini, was the first robot to be commercialized by the company Boston Dynamics. Meanwhile, Tesla is developing its Optimus model, with advanced computer vision for spatial recognition and displacement. The organizations do not disclose the technologies employed. Nevertheless, on Boston Dynamics' video channel, it is possible to glimpse part of their development, such as the training stages in which the robots learn to balance, walk, and even practice parkour.

Link to Boston Dynamics' disclosure channel:
https://youtube.com/@BostonDynamics

Recently, embedded Artificial Intelligence systems have gained prominence due to their ability to build original images from a textual description (Rombach et al., 2022). Graphic content-generating systems are forerunners in the way they use AI, some of them having gained prominence, such as Stable Diffusion, Midjourney, DALL-E 2, eDiffi, and Muse, which can craft artistic quality images (Balaji et al., 2022; Chang et al., 2023; Holz, 2022; Ramesh et al., 2022; Rombach et al., 2022). The images are generated by a process that starts with noise conditioned by the text and gradually takes the desired shape. To do this, these systems use diffusion models (DMs), which are Markov Chains with variational inference and, as will be explained later, Markov Chains are the basis of RL. For now, these systems are more of a tool than an artist. They facilitate the creative process, advancing much of the designer's work, but the human expert is still essential to adjust the details of the final artwork.

In the field of Natural Language Processing (NLP), an RL algorithm has been used to create a reward model (OpenAI, 2022). Researchers have developed ChatGPT, an Artificial Intelligence that participates in conversations in an advanced mode, capable of autonomously crafting text, tracking the context of conversations, and answering queries with direct and original answers (OpenAI, 2023). This technology opens opportunities for new prompt engineers, such as optimizing search engines, such as Google or Bing; extending our interactions with intelligent assistants, such as Alexa; implementing solutions in games; updating Non-Player Character (NPC) for entities with more freedom; and bringing more realism to interactions.

ChatGPT is available for free at:
https://chat.openai.com/
It is good to remember that there are also tools that check if the texts were generated by a machine or by a human. The detection works by checking whether the sequence of words is among the most likely to be recommended by intelligent systems; if so, it signals that the text was written by Artificial Intelligence. Here are some detection tools:
https://contentatscale.ai/ai-content-detector/
https://copyleaks.com/ai-content-detector

The possibility of building solutions in a virtual environment is very advantageous until the algorithm is perfected, as this avoids risks of environments and resource costs until the model is actually in its production version, building embedded robotic systems. Some options for embedding these solutions into hardware devices are becoming possible thanks to the advancement of the Internet of Things (IoT), such as the Arduino ESP32 cloud platform. It can read sensors and drive mechanisms, including being able to load them with Artificial Intelligence technologies, and can be implemented in physical devices, such as the design of a motorcycle capable of maintaining balance with the use of RL (Govindarajan & Salunkhe, 2022) or the design for the detection of people with ML (Ferraz, 2022).

The ESP32 cloud platform for developing smart devices on Arduino is available at:
https://create.arduino.cc/iot/

In turn, the discipline of Operations Research (OR) has a close connection with Reinforcement Learning, both are applications in decision-making problems but have different approaches. Operations Research relies on mathematical techniques to find optimal solutions to decision-making problems. It is widely used in many fields and involves the use of mathematical models to represent the variables and

constraints involved in a problem, as well as algorithms to find the optimal solution. Reinforcement Learning, on the other hand, relies on principles of reinforcement theory to train agents to perform tasks by experimenting and obtaining rewards. The main idea is that an agent learns to perform a task efficiently through trial and error, adjusting its actions based on the rewards obtained. RL is widely used in control systems, robotics, and games. By combining the two disciplines, researchers can support enterprises in making better decisions and achieving their desired results more efficiently.

The robotics market is hot and very promising. Applications for home automation and the development of autonomous vehicles have great appeal and are activities led by companies such as Amazon and Tesla. In addition, Artificial Intelligence has realized several solutions for humanity, in projects that generate social value, provided by a Human-Centered AI (HCAI), and in creating intelligible decision-making models, through an Explained AI (XAI).

This book is an invitation to students, professors, and academic researchers, as well as industry professionals from various fields, to rethink the challenges they face in the real world and design intelligent systems.

This first chapter highlights Artificial Intelligence techniques and the historical frameworks of Reinforcement Learning. Chapter 2 presents the concepts essential to understanding this approach. Chapter 3 illustrates the step-by-step operation of one of the most widely used algorithms. Chapter 4 presents useful tools for developing RL solutions. Chapter 5 demonstrates commented code with and without the use of AI. Finally, Chap. 6 presents the most recent applications of how RL impacts several areas of knowledge.

1.1 Artificial Intelligence

Initially, computing was successful and notorious for automating mechanical tasks. Later, computing started automating intelligent tasks. Referring to this first group, we establish the success of conventional computing, while this second group can be understood by intelligent systems, which established the field of Artificial Intelligence.

In turn, concerning coding techniques, traditional programming has been useful in building conventional systems but also in developing intelligent systems. Along with this, Machine Learning techniques have been very successful in building AI systems, as they perform very well in the ability to generalize human abilities. Figure 1.2 illustrates the layout of computer science and its programming techniques.

The term Artificial Intelligence became known due to a lecture given by John McCarthy during a conference on technology at Dartmouth College (USA) in 1956 (Dartmouth, 2022). Literature and fiction films have contributed to an incorrect interpretation of what Artificial Intelligence is. First, it is necessary to clarify that Artificial Intelligence refers to computer systems that perform activities similar to

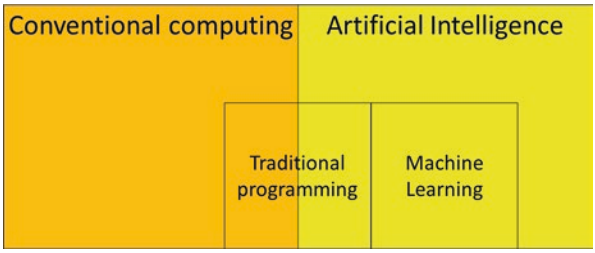


Fig. 1.2 Disposition of computer science

humans. What most movies end up dealing with is "Artificial Consciousness", which is much closer to the concept of Artificial Superintelligence (ASI).

We have defined Artificial Intelligence as a computer system that performs an activity with a similar performance as a human, be it intellectual or behavioral. As we will see, in many cases, this performance has already proven to be superior.

According to this definition, and as advocated by Russell & Norvig (2021), it is important to realize whether the system can use Machine Learning technology. In other words, it is considered Artificial Intelligence any system that presents intelligent behavior similar to human behavior, and for this, it may use traditional programming or a Machine Learning (ML) technique.

The integration of Reinforcement Learning agents in the real world is revolutionary in Artificial Intelligence applications to amplify and extend human capabilities. Although an ASI is still far from the real world, nothing is stopping us from continuing to try to build one.

The terms Artificial Narrow Intelligence (ANI), Artificial General Intelligence (AGI), and Artificial Superintelligence (ASI) have emerged independently at particular times in research but are currently used to determine the cognitive level that AI has reached. These terms can be understood as follows:

- ANI: machine intelligence is equivalent to human ability, but in specific abilities, i.e., they do not have the ability to perform tasks other than those for which they have been programmed. The term is often attributed to the American philosopher and mathematician John Searle in his article entitled "Minds, Brains, and Programs" published in 1980 (Searle, 1980). Since then, the term has been widely used to refer to Artificial Intelligence systems that are designed to perform many specific tasks in a variety of human activities.
- AGI: machine intelligence is equivalent to human ability in a variety of skills, i.e., they are capable of performing a variety of complex cognitive tasks, including reasoning and multiple problem solving. The term was coined in the 1970s by a group of researchers led by Marvin Minsky and Seymour Papert. At the time, researchers were working on Artificial Intelligence systems intended to perform a wide variety of tasks. Minsky and his colleagues discussed creating an Artificial General Intelligence system, capable of performing complex cognitive

tasks as the ultimate goal of their Artificial Intelligence research (Minsky & Papert, 2017). They believed that once AGI was achieved, machines could be considered intelligent in a general sense and no longer limited to specific tasks.

- ASI: machine intelligence is equivalent to human capacity in various abilities and extrapolates the ability to self-develop and even achieve artificial consciousness. The term "Artificial Superintelligence" was popularized starting in 2014 with the publication of the book "Superintelligence: Paths, Dangers, Strategies" by Nick Bostrom. The book explores the history of Artificial Intelligence and the development of technological growth, describing how AI is becoming increasingly advanced (Bostrom, 2014).

An example of ANI is a speech recognition system that is trained to accurately recognize words in a certain language or product recommendation. It is important to note that thus far, only ANI is part of our everyday life.

Although the idea of an AGI has been much discussed, many researchers believe that the creation of Artificial General Intelligence is still a distant goal. An example of an AGI is an AI system capable of playing different games, learning from the opponent, improvising moves, and eventually creating new strategies to win the game.

An example of ASI, on the other hand, would be an AI system capable of understanding and learning to solve complex problems on its own and in any field, such as biology, physics, mathematics, and even philosophy, as well as creating other technologies and scientific discoveries autonomously. However, ASI is currently only a theoretical concept, and there are no AI systems with this capability. It is not yet clear whether artificial superintelligence can be achieved, and if so, how it will be achieved. Some researchers believe that this can be done by improving Machine Learning algorithms and techniques, while others believe that it will be necessary to develop new frameworks and other approaches to Artificial Intelligence research, such as creating systems that mimic the mental functioning of the human brain.

Table 1.1 shows some examples of AI and its levels of complexity in fiction and reality.

Table 1.1 Example of Artificial Intelligence systems and their levels in fiction and reality

	Artificial Narrow Intelligence	Artificial General Intelligence	Artificial Superintelligence
Fiction	?	J.A.R.V.I.S. (Iron Man, 2008); and HAL 9000 (2001: A Space Odyssey, 1968)	Vision (Avengers: Age of Ultron, 2015); and David (A.I. – Artificial Intelligence, 2001)
Reality	Alexa Siri Smartwatch Personal assistants Autonomous cars Midjourney ChatGPT	?	?

Interestingly, while there are several examples of advanced AI in fiction, we still have no examples of AGI or ASI in reality. In turn, currently, the real world seems to be more endowed with constrained AIs than in fiction.

Some of these technologies have proven to be very versatile and revolutionary, such as ChatGPT. However, it is still considered a constrained AI, having been trained only to interpret and generate text quite efficiently.

It is natural that whenever a new technology arrives, people are impacted; it was so with electricity and the Internet. Potentially, the success of these devices lies in the fact that these solutions came out of research centers, whether industrial or academic, and were designed to be conveyed in a useful and friendly way into people's hands.

In addition, it is important to note that the creation of artificial superintelligence brings with it ethical and safety issues that need to be considered carefully to ensure that these systems are developed and used responsibly and safely.

To achieve more advanced AI systems, the paradigm seems to be: how can these technologies integrate multiple pieces of knowledge? With all this potential for more data, more algorithms, and more processing, it seems that we are headed in the right direction for the development of general-purpose intelligence.

Software engineering can be applied to all kinds of data, such as text, audio, image, and video. In this way, the development of many intelligent applications can benefit, as follows:

- Natural Language Processing (NLP): This is the area of Artificial Intelligence that is dedicated to making machines understand and process human language. Some examples of applications are chatbots and virtual assistants, which can help answer questions and perform simple tasks, or question complexity classifiers (Jardim, 2022);
- Speech recognition and speech processing: These areas of AI enable machines to recognize and process human speech. Some examples of applications are virtual assistants such as Siri and Google Assistant;
- Computer vision: This is the area of Artificial Intelligence that is dedicated to making machines understand and process images. Some examples of applications are facial recognition and medical image analysis; and
- Video analysis and human action recognition: These areas enable machines to analyze surveillance video and recognize human actions. Some examples of applications are the detection of suspicious movements in public areas.

There are many other applications in development and others that you can design yourself.

As software engineers, to develop intelligent systems, we can identify a task that requires human intelligence and think of ways to automate it. RL techniques have achieved success in many of these tasks. Thus, activities such as playing chess, driving a car, or getting around in environments with obstacles are excellent opportunities for applying AI. In addition, a major advantage is that the computational environment allows you to build and refine each feature virtually before risking its implementation on real-world hardware devices.

1.2 Machine Learning

Initially, computing began by explicitly programming machines to perform human tasks that could be automated. Now, machines can learn by example in a field of study called Machine Learning.

This technology has been very successful in processing text, audio, image, and video files. Software engineers are always looking for more efficient ways to design systems. Machine Learning can help humans make better-automated decisions (perform actions) across a range of Industry 4.0 challenges. Moreover, it is one of the most promising fields of Artificial Intelligence. It is a technology that will facilitate the work of experts and take civilization to the next stage of development.

Traditionally, computers follow rules explicitly programmed by the developer. However, the advance of technologies allowed them to perform actions without the rules being explicitly programmed; that is, machines were able to learn these rules through pattern discovery. Thus, ML can be understood as the area of computing in which the machine learns the rules through pattern recognition.

As illustrated in Fig. 1.3, in traditional programming, the programmer establishes the rules according to which the system should act. In ML, there is a previous training stage to define these rules, consolidated into a predictive model. Then, these rules are actually implemented so that the system follows them in a validation or production stage.

We can think of Machine Learning as a means for the machine to recognize patterns in data, i.e., all it takes is a set of data for the machine to learn these rules. According to Samuel (2000), Machine Learning is essentially a field of study that gives computers the ability to learn without being explicitly programmed.

There are three core categories within Machine Learning: Supervised Learning; Unsupervised Learning; and Reinforcement Learning. Figure 1.4 illustrates one way for the data scientist to decide which type of ML technique is best suited for each project by looking at how the data are available and what the goal is. Its concepts are best understood by comparing it to the other types of learning.

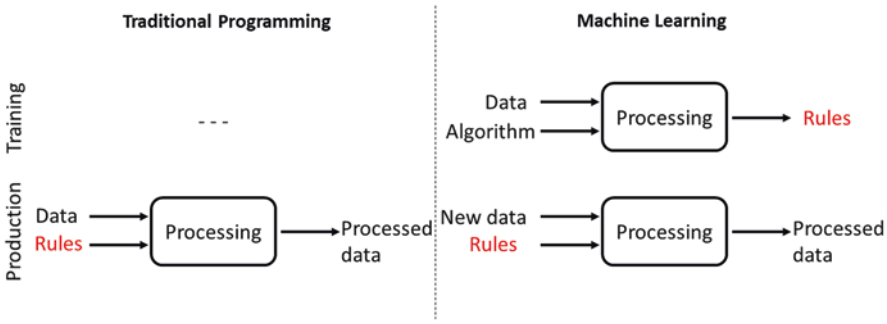


Fig. 1.3 Comparison between traditional programming and Machine Learning

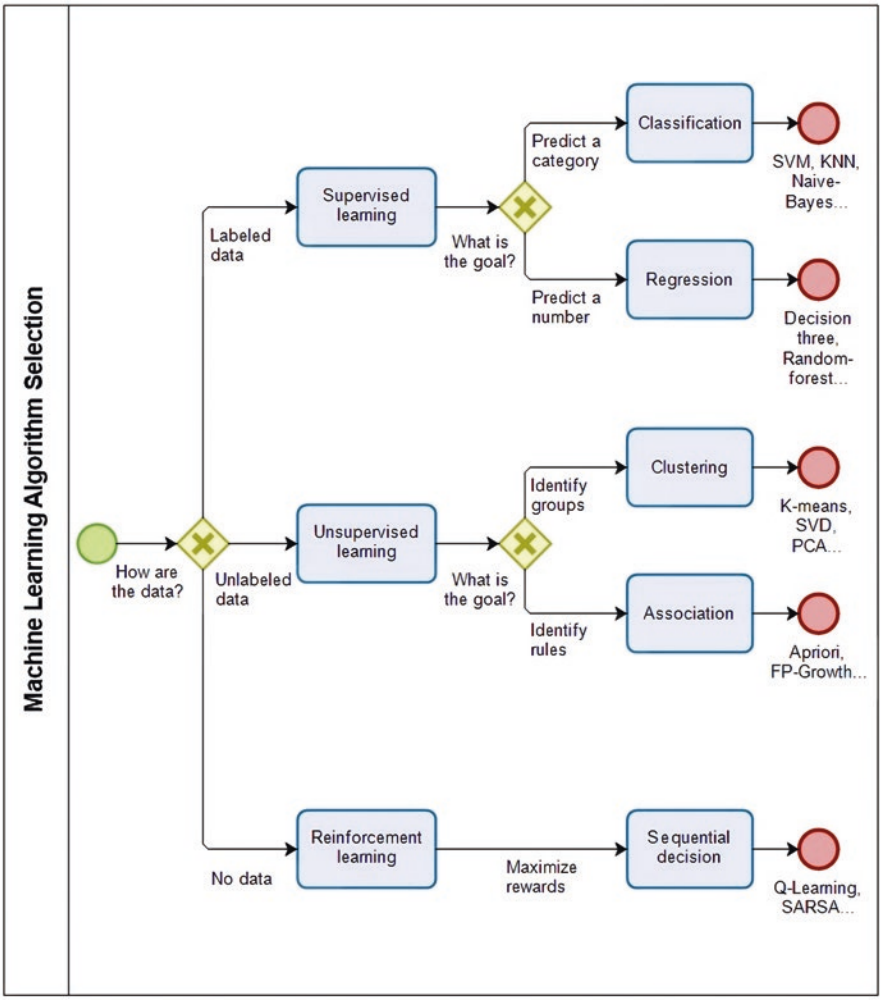


Fig. 1.4 Algorithm selection process by a Data Scientist.

In Supervised Learning, the machine learns from a labeled dataset, that is, the labeling of that data has been previously supervised by a human or another system. The machine should learn this pattern, these correlations, and these rules. The decisions it makes do not affect future choices.

In Unsupervised Learning, the machine learns from an unlabeled dataset, that is, the labeling of these data has not been previously supervised. The goal is for the machine to identify correlations between these data, and it can group similar instances or identify associations between them. Thus, the machine learns the rules of the correlations between the data in question.

In Reinforcement Learning there is no preexisting dataset. The machine must interact with the environment to collect these data and, through the rewards obtained, learn the pattern of actions that return the best rewards, i.e., the machine learns by seeking to maximize rewards. The algorithm constantly receives feedback from the environment about how positive its actions are. The decisions it makes will affect future choices.

Therefore, what all these techniques have in common is that they allow the machine to learn by recognizing patterns in the data. What differentiates them, in turn, is how they use this correlation and the data to be employed. Imagine that a programmer needs to build a movie recommendation system. They could use traditional programming, in which they would define the rules that the system must follow to recommend other movies to the customer. If the developer used Supervised Learning technology, they would use an algorithm that would recommend movies from the data of thousands of other users. If they used Unsupervised Learning, they could analyze the movies the person has watched and suggest similar movies. Whereas with Reinforcement Learning, the programmer could design a system that receives feedback from the user and understands their preferences, thus builds a knowledge base about them and suggests new movies. Similarly, a complex robotic system can have several mechanisms, each consisting of different technologies.

The use of ML is recommended in high adversity scenarios that require the ability to adapt, such as in identifying muddy car license plates, wine labels stained with booze, and uneven terrain on Mars. ML can be applied in solving various human problems, such as the following:

- Classification problems: these are those in which items are organized within existing classes. For example, classification of images into distinct groups; classification of texts, organizing them into classes of sentiment or question complexity (Jardim et al., 2022a, b).
- Regression problems: are those in which you want to predict a numerical value. It refers to the identification of real values. Examples include predicting a person's height, estimating the price of a product, and predicting the size of a property.
- Clustering problems: this is the organization of items that present similar characteristics. In this case, there are no previously defined groups. For example, grouping customers into profiles.
- Association problems: refer to the identification of rules in which one item is related to another. For example: identification of rules in purchasing, as in the case of who buys a product and normally buys another one in the same process.
- Sequential decision problems: these are problems in which the actions we choose in the present affect the rewards we will obtain in the future. They are problems that aim to maximize future rewards. They can be used when there is a specific value to be optimized and a function that can be discovered within a problem to optimize its solution. It is concerned with the long-term impact of its actions, aiming for optimization, maximum gain, or minimum loss. For example, the optimization of a product's delivery route considers several factors.

In short, if the challenge posed can be modeled as a sequential decision-making problem, it can be solved by RL (Sutton & Barto, 2020). As we will see, formulating a problem as a Markov Decision Process (MDP) is key to solving it. A Reinforcement Learning approach also allows for balancing the choice between immediate rewards with the consequences of long-term actions.

It is important to remember that it is not enough for the machine to perform well on training data (training dataset). It must perform well for new data (test dataset), i.e., it must learn to generalize these solutions to data it has never seen before (Moreira et al., 2022a, b).

1.3 Reinforcement Learning

The term "Reinforcement Machine Learning", or simply "Reinforcement Learning", is given because there is a reinforcement of the agent's actions. By way of analogy, imagine a dog being trained in a park. Every time it correctly performs a trick, such as sitting, lying down, or rolling on the ground, it gets a cookie. This reward reinforces the action, and the animal learns to repeat the act based on the order it is given. In this analogy, the dog is the agent, who by performing the trick, a correct action, receives a cookie, which is the reward. In other words, the correct action is reciprocated and reinforced, employing a reward; hence, it is considered Reinforcement Learning. Similarly, a hostile or incorrect action can be negatively rewarded as a punishment by the trainer.

Before we obtain the details, it is important to consider the following definitions in RL:

- Agent: the entity that will interact with the environment; it is the decision maker. It can be a robot, an autonomous car, etc.
- Environment: the universe in which the agent will interact. The environment is the outside world; it comprises everything outside the agent. For example, it can be a maze, a house, a park, etc. It does not necessarily have to be a physical space.
- Action (a): behavior that the agent can perform. Most often, it is associated with a movement, for example, toward the north, toward the south, or picking up or dropping an object, and so on.
- State (s): conjuncture of the agent with the environment. For example, consider the case of a robot in a position in a given maze.
- Reward (r): retribution (positive or negative) that the agent gets when reaching the next state (s'). For example, winning 10 points.
- Policy (Pi - π): strategy of action that promotes state change in the expectation of obtaining the best result. In other words, it is the objective that you have for the training, what the agent will learn. It can be, for example, which trajectory should be followed most efficiently.
- Episode: a complete set of actions that ends when reaching a goal. For example, going from one point to another, going from the house to the castle, etc. Thinking

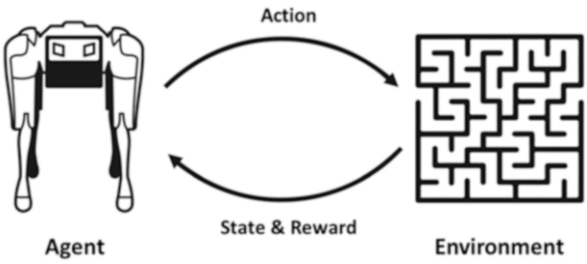


Fig. 1.5 Framework of an agent's interaction with the environment

that each movement is a single step, the set of steps to the final state is what is considered an episode.

Therefore, it can be said that the agent learns through experience, trial and error, or through obtaining rewards in an environment. These rewards can be positive or negative. Going back to the analogy of the dog, the cookie would be a positive reward, whereas getting its attention or using a whistle with an annoying noise, for example, would be a negative reward.

The goal, therefore, is to learn a policy that maximizes the accumulation of rewards. Here, it is not about getting immediate rewards but aiming for an accumulation of rewards at the end. The RL framework is composed of states, actions, and rewards (Sutton & Barto, 2020), as shown in Fig. 1.5.

In the framework, there is an agent that is in a certain initial state. In this state (s), the agent acts (a) on the environment, which promotes the change to the next state (s'), and it then receives a reward (r).

In RL, the agent records the data of its interaction and learns from the consequences of its actions. The system increments the agent with positive or negative rewards. The machine learns the consequences of its actions and what to do to achieve the goals. The idea is that the machine can design its own strategies to achieve its ultimate goal, even if circumstances change. In Reinforcement Learning, the agent learns to perform a task through interaction, or rather through experience, trial and error, cause and effect, or obtaining rewards from the environment.

The first time an agent enters an environment, it does not know how its world works. It is through interaction that the agent collects data, both the state and the action taken, and obtains rewards, which we can consider as labels. Once in possession of these data, it will be possible to perform better actions.

The purpose of the agent is to maximize the cumulative reward. The agent seeks to find a set of actions to achieve the highest cumulative rewards. Therefore, through interaction, the agent learns the rules of the environment, and then, after training, can choose a set of actions (policy) to achieve its goal.

The agent is not told which actions to take but instead must discover which actions bring the most benefits by trying them out. Thus, the agent learns from its own experience.

The agent's actions can affect not only the immediate reward but also the next situation and, through it, all subsequent rewards. Two factors characterize Reinforcement Learning: the search by trial and error and the existence of a delayed reward.

The world is not a static environment, and RL can perform well in these frequently changing scenarios. An embedded RL agent encounters these new situations and performs well.

Exploitation or exploration behavior

Learning from the environment is challenging. One such challenge is the dilemma of the agent's behavior of "exploitation" versus "exploration". The agent must take advantage of the knowledge it has already experienced by obtaining rewards, but it must also explore to discover possible better actions.

Exploitation and Exploration refer to the trade-off of the agent's decision between "treading an already known path" or "trying a new path," respectively. It is similar to the decision to go out to lunch: you can go to the restaurant you already know and receive the reward you already know or explore new restaurants and be surprised, either for better or worse. These behaviors can be described as follows:

Exploitation

Leverages the same actions in the environment, seeking known rewards (traveling through known states). It takes advantage of the agent's current estimated values; it takes advantage of knowledge for short-term benefits. It chooses the greedy action to try to obtain the maximum reward; however, precisely because it is greedy about estimated values, it may not obtain the highest rewards from the entire environment.

Exploration

Explores new actions in the environment, seeking greater rewards (traveling through new states). It allows the agent to improve its knowledge about each action in the expectation of obtaining greater benefits in the long run. Exploring allows for decreasing the uncertainty of these values, but it is worth remembering that it will inevitably lead to some negative reward from the environment. In any case, by improving the accuracy of the estimated values of actions, the agent will be able to make better-informed decisions in the future.

This behavior is a dilemma because at each step the agent must decide whether it will have an Exploitation or an Exploration behavior, that is, it must choose at each state.

Since the environment is unknown, the agent must try different actions and states. If we take advantage of the acquired knowledge indefinitely, it may be the

case that only good actions are taken, but not the best actions. Similarly, if we explore indefinitely, without taking advantage of the acquired knowledge, we may end up with a low total reward.

Therefore, how and when do you choose to exploit and explore? How do you balance these competing behaviors to achieve the best results? Ideally, the behavior should alternate between Exploitation and Exploration.

RL Application
To better understand the logic behind Reinforcement Learning, we can model a game of chess as an MDP problem. The goal of the game is to achieve a checkmate. A single game can be understood as an episode. The system can be modeled so that the agent is the pawn and the environment is the board. It can only move forward one space at a time or diagonally to attack an opponent's piece, and these are its possible actions. For each action, the pawn assumes a new state on the board, and in general, a positive reward is given for eliminating an opponent's piece. Losing a pawn is bad. However, there are rewarding situations even with the loss, such as losing a pawn to gain a queen. Note that an intelligent agent is not looking for the best immediate reward but for maximizing rewards. The move of sacrificing a pawn gets a negative immediate reward; however, it gets a very positive cumulative reward.

In this sense, the policy is the strategy of action that the pawn must learn to reach the best end result. The complete set of actions of the pawn until it reaches the final goal (the maximum reward), in this case, of knocking out the king, is the episode.

This approach will be demonstrated in the chapters ahead, with the demonstration of taking better actions within a policy and implementing the mathematical formula that makes it possible.

1.4 History

The knowledge area of Reinforcement Learning combines influences from disciplines related to psychology, neuroscience, operations research, and computer science. The following are the developments that most impacted the development of this line of research.

In the psychology of animal learning, the Russian physiologist Ivan Pavlov observed that dogs were treated for several days with a sound signal followed by a bowl of food. On subsequent occasions, beeping was maintained but without the provision of food. On these occasions, it was observed that the sound signal alone aroused the dogs' salivating behavior, demonstrating the animal conditioning that associated the sound signal with the moment of feeding. His research on the psychology of animal behavior earned him the Nobel Prize in Medicine in 1904 (Nobel Prizes, 2022).

The American psychologist Edward Thorndike studied the theory of learning by trial and error, in which several attempts were made to arrive at a solution. An action is performed, and if the reward is not achieved, another action is performed. This

procedure is repeated until a favorable solution is reached. Thorndike created an experiment to study how cats learn to exit the box in a situation in which the animals had to solve some puzzles to exit the box. He concluded that cats learned by trial and error, determining that learning happened in this way rather than from a sudden flash of understanding (Thorndike, 1911).

Psychologist John B. Watson (1913) established behaviorist theory. Watson conducted a series of studies and demonstrated that experience with the environment determines human behavior, even overriding genetic factors. Drawing a parallel with current computing, we can observe the behavior of an agent being guided by knowledge acquired from the environment (experiences) versus purely programmed behavior (genetic).

In the late 1940s, John von Neumann and Stanislaw Ulam developed the Monte Carlo Method (MCM), which solves deterministic problems using probabilities, or rather, massive random sampling of data (Neumann & Richtmyer, 1947). It is understood as a method for solving problems by approximating the solution.

In 1950, Alan Turing published "Computing Machinery and Intelligence" with the following propositions (Turing, 1950, p. 457): putting intelligence into computers by developing an assessment to ascertain whether a machine was intelligent (Turing Test) and, although not defined as Reinforcement Learning at the time, proposed an approach to teaching computers that "the use of punishments and rewards can, at best, be part of the teaching process." Later, he also suggested that it would be easier to create human-level AI by developing learning algorithms and then teaching the machine, rather than programming intelligence.

The term Artificial Intelligence was coined by John McCarthy during a conference on technology at Dartmouth College (USA) in 1956 (Dartmouth, 2022).

In the field of Operations Research, the Russian mathematician Andrei Andreyevich Markov formalized the MDP (Bellman, 1957).

The term "optimal control" was introduced in the late 1950s to describe the problem of designing a controller to minimize or maximize the behavior of a dynamic system over time. An approach to this problem was developed in the mid-1950s by Richard Bellman and others, extending the Hamiltonian and Jacobian theories of the nineteenth century. This approach uses the concepts of the state of a dynamical system and a value function, or "optimal return function," to define a functional equation, now commonly known as the Bellman Equation. This class of methods for solving optimal control problems by solving this equation has been called "dynamic programming" (Bellman, 1957).

In 1959, Arthur Samuel coined the term "Machine Learning" (Samuel, 2000). He also used methods that we now call Reinforcement Learning, whereby his program learned to play checkers. By doing this, he refuted the idea that computers can only do what they are programmed to do. In fact, his program learned to play checkers better than its creator. Samuel proposed the more modern ideas of Reinforcement Learning, covering temporal difference learning, function approximation, and multilayer representations of value functions.

Hilgard and Bower observed that the change in an organism's behavior in a given situation was based on repeated experiences of the organism in that situation

(Hilgard & Bower, 1975). For the first time, the relationship between Markov Decision Processes and Reinforcement Learning was determined (Werbos, 1977).

In 1988, an important paper by Rich Sutton related Reinforcement Learning to MDP (Sutton, 1988), referencing Samuel's checkers game program. This work enabled the formalization of Reinforcement Learning within a decision-theoretic framework and provided a mathematical understanding of temporal difference methods. Following this, several papers related AI research to MDPs, acquiring deep theoretical grounding, and the field of Reinforcement Learning proved prolific in applications in robotics and process control.

The DYNA architecture was proposed by Sutton (1990), in which a combination of time-difference learning and simulated experience models is used. The Q-learning algorithm was developed by Chris Watkins in his Ph.D. (Watkins & Dayan, 1992). The SARSA algorithm, by Rummery and Niranjan, was revealed in a technical report (Rummery & Niranjan, 1994).

Narendra and Thathachar (1974) published the first paper on learning automata, originally described explicitly as finite state automata. Similar to Reinforcement Learning, a learning automaton algorithm also has the advantage of solving a problem when the probabilities or rewards are unknown (Narendra & Thathachar, 1974).

Neuroscience often inspires Reinforcement Learning and confirms the value of this approach. A study using single-cell recording suggests that the primate brain's dopamine system performs something similar to value function learning (Schultz et al., 1997).

Barto et al. (1995) discuss different exploratory methods for the sequential decision problem. Brafman and Tennenholtz (2000) and Kearns and Singh (2002) describe algorithms that explore unknown environments and guarantee near-optimal policy convergence. Dietterich (2000) introduced the concept of additive decomposition of Q-functions caused by the hierarchy of subroutines. Bayesian Reinforcement Learning provides another view for uncertainty and model exploration (Dearden et al., 1998, 1999).

The book Reinforcement Learning: An Introduction (1998) by Sutton and Barton consolidates the research area and shows how Reinforcement Learning combines the ideas of learning, planning, and acting.

The availability of open-source simulation environments to develop and test agents has also leveraged Reinforcement Learning studies.

The University of Alberta, Canada, launched the Arcade Learning Environment (ALE) (Bellemare et al., 2013), a platform for 55 classic Atari video game games. The pixels on the screen are passed to the agent as an observation along with the game score. The DeepMind team used ALE to implement DQN (Deep Q-Network) learning and test the generality of the system on different games (Mnih et al., 2015), in which the DQN system uses a deep network to learn the Q function.

DeepMind, in turn, has opened up the code for several agent platforms, including the DeepMind Lab (Beattie et al., 2016); the AI Safety Gridworlds (Leike et al., 2017); the Unity gaming platform (Juliani et al., 2018); the DM Control Suite (Tassa et al., 2018); and, together with Blizzard, launched the StarCraft II Learning Environment (SC2LE), a Reinforcement Learning environment based on the game StarCraft II (Vinyals et al., 2017).

The SYNTHIA system (Ros et al., 2016) is a simulation environment designed to improve computer vision for autonomous cars. The OpenAI Gym (Brockman et al., 2016) provides various environments for Reinforcement Learning agents that are compatible with other simulators, such as Google Research Football (Kurach et al., 2020).

Facebook's HORIZON platform enables Reinforcement Learning in large-scale production systems (Gauci et al., 2018), and its AI Habitat simulation provides a photorealistic virtual environment for robot tasks (Savva et al., 2019).

A recent study has enabled an Artificial Intelligence system to improve the way we perform matrix multiplication, impacting mathematics, computing, and the Reinforcement Learning technique itself (Fawzi et al., 2022).

As we have seen, RL technology is very new, and we currently find a prolific scenario for expanding its potential. This is just the beginning of what is to come.

References

Alibay, F., Koch, J., Verma, V., Bean, K., Toupet, O., Petrizzo, D., Chamberlain-Simon, B., Lange, R., & Hogg, R. (2022). *On the operational challenges of coordinating a helicopter and Rover Mission on Mars*. 2022 IEEE Aerospace Conference (AERO), pp. 1–17. https://doi.org/10.1109/AERO53065.2022.9843670

Balaji, Y., Nah, S., Huang, X., Vahdat, A., Song, J., Kreis, K., Aittala, M., Aila, T., Laine, S., Catanzaro, B., Karras, T., & Liu, M. -Y. (2022). eDiff-I: Text-to-image diffusion models with an ensemble of Expert Denoisers. https://doi.org/10.48550/arXiv.2211.01324.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1–2), 81–138. https://doi.org/10.1016/0004-3702(94)00011-O

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., … Petersen, S. (2016). DeepMind Lab.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research, 47*, 253–279. https://doi.org/10.1613/jair.3912

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 679–684.

Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, Inc.

Brafman, R. I., & Tennenholtz, M. (2000). A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence, 121*(1–2), 31–47. https://doi.org/10.1016/S0004-3702(00)00039-4

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. https://doi.org/10.48550/arXiv.1606.01540.

Chang, H., Zhang, H., Barber, J., Maschinot, A., Lezama, J., Jiang, L., Yang, M. -H., Murphy, K., Freeman, W. T., Rubinstein, M., Li, Y., & Krishnan, D. (2023). *Muse: Text-to-image generation via masked generative transformers*. https://doi.org/10.48550/arxiv.2301.00704

Dartmouth, V. (2022). *Artificial Intelligence (AI) Coined at Dartmouth*. Vox of Dartmouth. https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth

Dearden, R., Friedman, N., & Russell, S. (1998). *Bayesian Q-learning*.

Dearden, R., Friedman, N., & Andre, D. (1999). *Model based Bayesian Exploration*. https://doi.org/10.5555/2073796.2073814

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227–303.

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). *Challenges of real-world reinforcement learning*.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, R., F. J, Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., & Kohli, P. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature, 610*(7930), 47–53. https://doi.org/10.1038/s41586-022-05172-4

Ferraz, A. (2022). *Embedded machine learning for person detection*.

Gauci, J., Conti, E., Liang, Y., Virochsiri, K., He, Y., Kaden, Z., Narayanan, V., Ye, X., Chen, Z., & Fujimoto, S. (2018). *Horizon: Facebook's open source applied reinforcement learning platform*.

Govindarajan, M., & Salunkhe, S. (2022). *Reinforcement Learning for a self-balancing motorcycle*. https://create.arduino.cc/projecthub/metrowest_aug/reinforcement-learning-for-a-self-balancing-motorcycle-719b40

Hilgard, E. R., & Bower, G. H. (1975). *Theories of learning* (4th ed.). Prentice-Hall.

Holz, D. (2022). *Midjourney*. https://www.midjourney.com/

Jardim, R. R. J. (2022). *Desenvolvimento de um modelo classificador de questões para o cenário educacional brasileiro fundamentado em ciência de dados*. Universidade Federal do Rio de Janeiro (UFRJ).

Jardim, R., Delgado, C., & Schneider, D. (2022a). Data science supporting a question classifier model. *Procedia Computer Science, 199*, 1237–1243. https://doi.org/10.1016/J.PROCS.2022.01.157

Jardim, R. R. J., Delgado, C., & Silva, M. F. (2022b). CLIQ! Intelligent Question Classifier for the elaboration of exams. *Software Impacts, 13*. https://doi.org/10.1016/J.SIMPA.2022.100345

Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). Unity: A general platform for intelligent agents.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2), 209–232. https://doi.org/10.1023/A:1017984413808

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238–1274. https://doi.org/10.1177/0278364913495721

Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., & Gelly, S. (2020). Google research football: A novel reinforcement learning environment. *Proceedings of the AAAI Conference on Artificial Intelligence, 34*(04), 4501–4510. https://doi.org/10.1609/aaai.v34i04.5878

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., & Legg, S. (2017). AI safety Gridworlds.

Minsky, M., & Papert, S. A. (2017). *Perceptrons*. The MIT Press. https://doi.org/10.7551/mitpress/11301.001.0001

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236

Moreira, M. Â. L., Gomes, C. F. S., dos Santos, M., da Silva Júnior, A. C., & de Araújo Costa, I. P. (2022a). Sensitivity analysis by the PROMETHEE-GAIA method: Algorithms evaluation for COVID-19 prediction. *Procedia Computer Science, 199*, 431–438. https://doi.org/10.1016/J.PROCS.2022.01.052

Moreira, M. Â. L., de Junior, C. S. R., de Silva, D. F. L., de Castro Junior, M. A. P., de Costa, I. P. A., Gomes, C. F. S., & dos Santos, M. (2022b). Exploratory analysis and implementation of machine learning techniques for predictive assessment of fraud in banking systems. *Procedia Computer Science, 214*, 117–124. https://doi.org/10.1016/j.procs.2022.11.156

Narendra, K. S., & Thathachar, M. A. L. (1974). Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-4*(4), 323–334. https://doi.org/10.1109/TSMC.1974.5408453

Nobel Prizes. (2022). *Ivan Pavlov - Biography*. http://www.nobelprize.org/nobel_prizes/medicine/laureates/1904/pavlov-bio.html

OpenAI. (2022). *ChatGPT: Optimizing language models for dialogue*. https://openai.com/blog/chatgpt/

OpenAI. (2023, March). *GPT-4 technical report*. https://doi.org/10.48550/arXiv.2303.08774

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). Hierarchical text-conditional image generation with CLIP Latents. https://doi.org/10.48550/arXiv.2204.06125

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models, 10674–10685. https://doi.org/10.1109/cvpr52688.2022.01042

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., & Lopez, A. M. (2016). *The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes*.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. Undefined.

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th-Glob ed.). Perarson.

Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 44*(1–2), 207–219. https://doi.org/10.1147/rd.441.0206

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., & Batra, D. (2019). *Habitat: A platform for embodied AI research*.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science, 275*(5306), 1593–1599. https://doi.org/10.1126/science.275.5306.1593

Searle, J. R. (1980). Minds, brains, and programs. *The Behavioral and Brain Sciences, 3*(3), 417–424. https://doi.org/10.1017/S0140525X00005756

Singh, B., Kumar, R., & Singh, V. P. (2022). Reinforcement learning in robotic applications: A comprehensive survey. *Artificial Intelligence Review, 55*(2), 945–990. https://doi.org/10.1007/s10462-021-09997-9

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*(1), 9–44. https://doi.org/10.1007/BF00115009

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990* (pp. 216–224). Elsevier. https://doi.org/10.1016/B978-1-55860-141-3.50030-4

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. A Bradford Book.

Sutton, R. S., & Barto, A. G. (2020). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. de Las, B. D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., & Riedmiller, M. (2018). *DeepMind Control Suite*.

Thorndike, E. (1911). *Animal intelligence: experimental studies*. The Macmillan Company.

Turing, A. M. (1950). Computer machinery and intelligence. *Mind, LIX*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., … Tsing, R. (2017). *StarCraft II: A new challenge for reinforcement learning*.

von Neumann, J., & Richtmyer, R. D. (1947). Statistical methods in neutron diffusion. In *Analogies between analogies* (pp. 17–36). University of California Press. https://doi.org/10.1525/9780520322929-004

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292. https://doi.org/10.1023/A:1022676722315

Watson, J. B. (1913). Psychology as the behaviorist views it. *Psychological Review, 20*(2), 158–177. https://doi.org/10.1037/h0074428

Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General System Yearbook, XXII*, 25–38.

Chapter 2
Concepts

The first step in developing systems based on Reinforcement Learning will be to model your problem as a Markov Decision Process (MDP) (Bellman, 1957). To do this, we will understand the fundamentals of formulating and solving such problems.

2.1 Markov Chain

The origins of RL have their foundations in Markov Chains. Markov Chains refer to probabilistic systems in which the future state depends on the present state and not on past states. This means that it does not matter which states the agent passed through until the present state; what matters are the relationships and probabilities of the next actions. Markov Chains allow the study of state transitions from the present to the future.

To obtain an example, let us say that in a given situation, I can only have two states: "sunny" or "rainy". Let us also say that we performed some observations and noticed that the sunny days were followed by sunshine 70% of the time, and therefore, 30% of them were rainy. We also observed that on a rainy day, the next day was followed by sunshine 60% of the time and by rain 40% of the time. This change of states can be understood as the probability of transition from one state to another. Thus, it is possible to model the states and their transitions, as presented in Fig. 2.1.

Notice that this problem in Fig. 2.1 can be modeled as a graph (a representation of objects and their relationships). We can read the graph as follows: there is a 70% chance that a sunny day will be followed by another sunny day and a 30% chance that it will be followed by a rainy day; and from a rainy day, there is a 40% chance that it will be followed by a rainy day and a 60% chance that it will be followed by a sunny day.

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_2

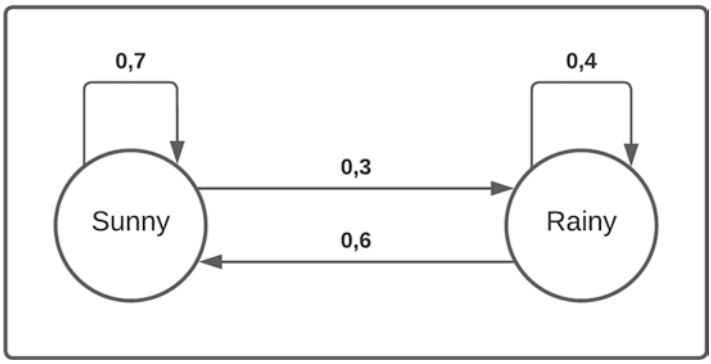


Fig. 2.1 Graph of the transitions between sunny and rainy states

Table 2.1 Matrix of current and future states

		Future States	
		Sunny	*Rainy*
Present States	*Sunny*	0.7	0.3
	Rainy	0.6	0.4

We can also represent the graph as a matrix, which can be organized by considering the rows as present states and the columns as future states, so we can assemble the state transition matrix in Table 2.1.

In this way, we can draw up the following matrix:

$$M = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix}$$

This is a simplified demonstration. However, imagine that in a complex system, we can have numerous states. For these more complex cases, we can use matrix calculus to make the operation easier (coming from linear algebra).

Therefore, let us demonstrate the importance of this technique. Imagine that today is a rainy Monday and we want to know the probability of rain on Wednesday. Note that we can only have two possible paths, according to Table 2.2.

For this case, we could perform the following calculations for the state transitions:

1°: Rainy-Sunny × Sunny-Rainy = 0.6 × 0.3 = 0.18
2°: Rainy-Rainy × Rainy-Rainy = 0.4 × 0.4 = 0.16

Table 2.2 Possible paths from the current state

Paths	Monday	Tuesday	Wednesday
1°	Rainy	Sunny	Rainy
2°	Rainy	Rainy	Rainy

Fig. 2.2 Resulting matrix

In other words, we only have these two possible paths. Therefore, since it is raining on Monday, the probability of rain on Wednesday is the sum of these two probabilities, which is 0.18 + 0.16 = 0.34.

Note that if one transition is the matrix by itself, two transitions equal the matrix multiplied by the matrix itself, which results in another 2 × 2 matrix with the following results:

$$M \times M = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} = \begin{bmatrix} (0.7*0.7+0.3*0.6) & (0.7*0.3+0.3*0.4) \\ (0.6*0.7+0.4*0.6) & (0.6*0.3+0.4*0.4) \end{bmatrix}$$

$$= \begin{bmatrix} 0.67 & 0.33 \\ 0.66 & 0.34 \end{bmatrix}$$

Additionally, notice that if the initial state was a rainy Monday and our claim for the final state will be a rainy Wednesday (Fig. 2.2), we obtain the result in the matrix:

Note also that if Monday is rainy and we want to know the probability of getting a sunny Wednesday, this result is also calculated in the matrix, which is a probability of 0.66. Similarly, if Monday is sunny and we want to know the probability of having a Wednesday that is also sunny, we obtain 0.67.

The matrix calculation comprises the operation of all possible paths, in this case, two, obtaining the resulting matrix.

In turn, if we also want to know the probability of reaching a Thursday with rain, we make the calculation referring to $M \times M \times M$, obtaining a new matrix, and the value obtained is a probability.

$$M \times M \times M = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.667 & 0.333 \\ 0.666 & \mathbf{0.334} \end{bmatrix}$$

Therefore, notice that in this probabilistic system, the future state depends only on the present state and not on the past states, culminating in a Markov Chain.

2.2 Markov Decision Process

The Markov Decision Process (MDP) is an extension of the Markov Chain with the inclusion of a reward function. It provides a logic for modeling decision-making in situations where outcomes are partially random and partially controlled by a decision-maker.

They are called "Markov processes" because they follow Markov's property, which states that the effect of an action depends only on the action and the current state of the system and not on how the process reaches that state. They are also called decision processes because they deal with the possibility of the agent, the decision maker, interfering in the system through actions, unlike Markov Chains, which do not consider how an agent can interfere in the process.

As discussed earlier, most Reinforcement Learning problems can be modeled as an MDP (Sutton & Barto, 2020). This is because MDP is a way of formulating processes in which the transitions between states are probabilistic; it is possible to observe what state the process is in and intervene at decision points through actions. In turn, each action has a reward, either positive or negative, depending on the state of the process.

The agent tries to choose the optimal actions from the many possibilities at each step of the process. An agent can try many different policies, although we are looking for the one that gives us the most utility.

One can conclude that some states are more desired than others. The issue is that the agent should maximize the reward by avoiding states that return negative values and choosing those that return positive values. The solution is to find a policy that selects the action with the highest reward.

However, an important feature of Reinforcement Learning is that the agent's actions can affect the state. For example, defending a bishop or surrendering a knight in a chess game impacts the success of the next moves. This creates a feedback loop in which the agent's actions can change the next state of the system, which in turn can affect the action the agent must choose in the next step.

The goal of an MDP or Reinforcement Learning problem is the expected cumulative total return. This is achieved by properly choosing a decision policy that will dictate how agents should act in every possible state. Since future states cannot be known in advance, we must now decide how we will act in all possible future situations to maximize the expected cumulative return on average.

MDP can be seen as a framework for formalizing sequential decision-making problems, in which decisions are broken down into several subproblems.

Five fundamental elements characterize an MDP: state, action, reward function, transition function, and discount factor. Respectively, a conventional description of an MDP is the tuple (S, A, R, P, γ). These elements are detailed below.

1. *State (s)*

A set of states that the agent can be in. The state contains the data needed to make decisions, determine rewards, and guide transitions.

2. *Action (a)*

A set of actions that the agent can take to move from one state to another. It is the set that contains all the possible actions that can be performed in the environment.

3. *Reward Function (r)*

The probability of obtaining a reward when an agent moves from one state to another. Indicates the direct reward for acting "*a*" in state "*s*". In some problems, the reward is not explicit, requiring the modeling of the reward to achieve the goal of solving the problem. Poor formulation of the reward can result in inappropriate behavior.

4. *Transition Function (t)*

The probability of an agent moving from one state to another by acting. It refers to the function that drives the dynamics of the system over time, moving the agent from state "*s*" to a new state, "*s′*". Usually, a transition involves both a stochastic and deterministic component. Transitions can be viewed from an external environment without being explicitly modeled.

5. *Gamma Discount Factor (γ)*

Controls the importance of immediate and future rewards. Defines the degree to which future rewards affect current decisions. Generally, the discount rate is set according to the level of uncertainty in the environment. The higher the uncertainty, the higher the discount rate is set, demonstrating a low influence of today's actions on future performance.

Overall, it indicates the extent to which future rewards are considered while a decision is being made now, with 0 completely ruling out future compensation and 1 considering all future compensation equally. It is worth noting that a very low value encourages short-term behavior, while a higher value emphasizes seeking long-term rewards. In this way, the discount factor controls how much the immediate reward scenario is preferable to the future reward.

Finally, each building block can present major challenges. The effectiveness of the algorithm used can be significantly impacted by how the MDP is modeled. In addition, the ability to clearly define MDPs is crucial to uniformly communicate academic and business problem descriptions. Note that it only depends on the current state and future actions. This means that it exhibits the Markov property.

We can model sequential decision-making problems to be solved as an MDP. Imagine that instead of using traditional programming, with all the rules defined, we want a robotic arm to learn to open an egg through Reinforcement Learning. Note that there are many ways to formalize this task; this is one of them: you must establish your goal and formalize states, actions, and rewards. For example:

1. *States:*
The sensor readings, such as movement speed and arm angle.
2. *Actions:*
The voltage applied to each motor.
3. *Rewards:*
+50 on reaching the goal; and −1 with each move.

Episodic or Continuous Task

The tasks to be automated may have a well-defined end or not, called episodic or continuous tasks, respectively. This identification helps in the most appropriate formulation of MDP problems.

Episodic tasks are activities that have a well-defined ending, for example, the match in a computer game that brings a clear ending, with the hero's victory or death.

The episode is understood by the sequence of actions from the initial state to the end. Then, the agent can review the cumulative reward it received during the episode, learn from this information, and adjust its parameters. Over the episodes, the agent should be able to choose a strategy where the total reward is higher.

The activities are finished in each episode, which ends in a terminal state. The next episode begins regardless of how the last episode ended. The return on the action is the sum of the rewards at the end of the episode. The game of chess is an example of an episodic problem because there is an end.

Continuous tasks are activities where the interaction continues without an endpoint. An example is robotic arms in the automation industry that must perform the same task indefinitely.

In these environments, the agent's interaction continues indefinitely, performing the same operation without end. In these cases, tasks are not finished, and there are no terminal states.

In such cases, we could design an intelligent thermostat capable of catering to the preference of each user. For this, each time there is a human intervention, there would be a negative reward.

Deterministic or Stochastic Policy

The policy (π) is a function that takes the current state (s) and leads it to action (a). If the policy follows a conventional function of the state (s), the outcome of action (a) will be a single number and is thus called a deterministic policy. If the policy is represented by a probability distribution instead of a function, it is considered a stochastic policy. We can draw the following differences between deterministic and stochastic policy:

- Deterministic policies will always obtain the same response for a given state. In general, they depend on previous states and actions. In MDP, where state transitions depend only on the current state, deterministic policies will also depend only on the current state;

- Stochastic policies: are random. They generalize the deterministic policies. In MDP cases with known transition probabilities, only deterministic policies need to be considered. In cases of unknown transition probabilities, the randomness of the actions will allow exploration for better model estimation.

It is worth noting that in the MDP, there is not always a deterministic optimal policy, so the task will be to identify all possible deterministic policies.

The problem formulation; as an episodic or continuous MDP can be used to model many real-world problems, such as teaching a robot to walk, driving an autonomous car, or developing a robotic arm. Although at first, each has a distinct goal, all can be understood as maximizing the cumulative reward.

2.3 Bellman Equation

Sequential decision problems can be solved using a mathematical modeling technique developed by Richard Bellman (1954) known as dynamic programming (DP).

Dynamic programming is a class of algorithms that attempt to simplify complex problems by breaking them down into subproblems and solving them iteratively. This means that instead of solving one complex problem at a time, it is divided into simple subproblems, and for each subproblem, the solution is calculated. If the same subproblem occurs, it is not recalculated, but the solution already calculated is used.

The Bellman equation (Bellman, 1954) is the fundamental element for solving MDP problems. It is present in most applications of Reinforcement Learning. Solving this problem means finding the optimal policy and value functions to obtain the maximum reward or minimum loss. The Bellman equation solves the following question: from the state the agent is in, what long-term reward can it expect?

In Bellman's equation, the value of the current state equals the action that maximizes the reward value of the best action in the current state and applies a discount factor multiplied by the value of the next state. This formulation helps determine what reward can be expected relative to the advantages or disadvantages of each state.

Deterministic or Stochastic Environment

A deterministic system describes what "will happen", while a stochastic system describes what "will probably happen". The advantage of the stochastic system over the deterministic system is that it is not necessary to know all the variables involved in the process. It is possible to make a function modeled on data where all possible situations will fall within the elaborate statistic.

Therefore, we can consider these two types of environments:

- Deterministic environment: an environment is considered deterministic if we know the outcome based on the current state. For example, in a game of chess, we know exactly the outcome of any move.

- Stochastic environment: an environment is considered stochastic if we do not know the outcome based on the current state; the uncertainty is large. For example, in dice throwing, if the conditions of the throw are not known, we will never know the outcome of a throw.

Thus, we can apply the Bellman Equation according to the type of environment, deterministic or stochastic. Consider, then, the Bellman Equation for a deterministic environment:

$$V(s) = \max_a \left(R(s,a) + \gamma V(s') \right) \tag{2.1}$$

$V(s)$ is the value of the current state. $V(s')$ is the value for reaching the next state after acting "a". $R(s, a)$ is the reward the agent receives after acting "a" in state "s". Since the agent can perform different actions, it uses the maximum value because it seeks the optimal state. γ is the discount factor, mentioned earlier.

For a stochastic environment, in turn, consider the following Bellman Equation:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s' \mid, s \mid, a) V(s') \right) \tag{2.2}$$

In a stochastic environment, when we act, there is no guarantee that we will reach a specific state, but there is a probability that we will reach a certain state. $P(s' \mid s, a)$ is the probability of ending state "s'" by performing action "a", which is equal to the total number of future states.

The value of the given state is equal to the maximum action. In other words, among all possible actions in the current state, the action that maximizes value is selected. We take the reward for the optimal action "a" in state "s" and apply a discount factor that reduces the reward over time, which is the *gamma* factor. Each time an action is performed, the agent advances to the next state "s". This is where dynamic programming comes in. Since it is recursive, it takes the value of state "s'" and puts it back into the value of the function "$V(s)$". You continue like this until you reach the final state and know the value of the state the agent is in, assuming it has chosen the optimal actions. The point is that at each step, it is necessary to know the optimal action that maximizes the expected reward in the long run. Thus, RL allows relating the current state to the value of future states without observing all future rewards.

There are a few options to obtain an optimal solution for MDP, for example, value iteration; policy iteration; SARSA (*State-Action-Reward-State-Action*); and *Q-Learning*. We will look at these strategies next.

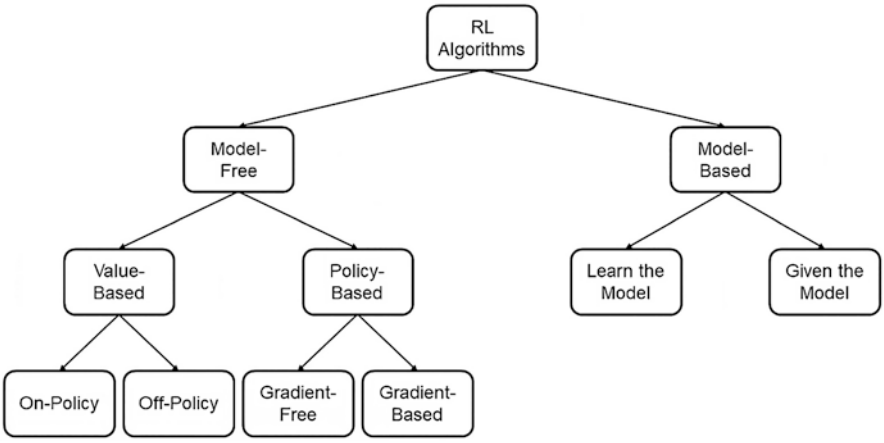


Fig. 2.3 Approaches to RL algorithms

2.4 Algorithm Approaches

The main approaches of RL algorithms are organized as shown in Fig. 2.3, adapted from Zhang and Yu (2020). For these cases, the changes in the environment are assumed to be zero or minimal.

The distinction between these approaches is presented below:

- Model-free: They do not use the transitions of the environment, in which case a model does not need to be learned. They directly estimate the optimal value function or policy for each action. More interactions with the environment are needed. In turn, Model-Free approaches can be divided into:
  - Value-based: learns the value of the state or state action. It acts by choosing the best action in the state. Exploration is needed. Can be subdivided into on-policy and off-policy. On-policy methods assume that the experience used for learning came from the policy being corrected, for example, the SARSA algorithm. Off-policy methods do not assume that the experience came from the policy. Learning can take place with the experience collected from any other policy, including interactions with human experts, such as the Q-Learning and DQN (Deep Q-Network) algorithms.
  - Policy-based: directly learns the stochastic policy function that maps state to action. It acts by sampling policy. In the expectation of obtaining the best parameters that will optimize the given function, the optimization method tries to find the best parameters close to the initial parameters iteratively and can be gradient-free or gradient-based. Gradient-free methods use a gradient to obtain the optimization parameters, such as the SAMUEL algorithm. Gradient-based methods do not use a gradient to obtain the optimization parameters, such as the ACKTR algorithm (Actor-Critic using Kronecker-Factored Trust Region).

- Model-based: make use of the transitions of the environment; in this case, a model needs to be learned. They assume that the agent can learn a model that tells how the environment evolves from the current state and the chosen action. They can be divided into:
  - Learn the model: focuses mainly on how to build the environment model. For example, World Models.
  - Given the model: methods concerned with how to apply the acquired model. For example, AlphaZero.

One of the most widespread algorithms is Q-Learning (Watkins & Dayan, 1992), which is an example of a model-free, value-based, off-policy algorithm. In the following chapter, we will explore solving sequential decision problems from this algorithm.

References

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society, 60*(6), 503–515. https://doi.org/10.1090/S0002-9904-1954-09848-8

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics, 6*(5), 679–684.

Sutton, R. S., & Barto, A. G. (2020). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292. https://doi.org/10.1023/A:1022676722315

Zhang, H., & Yu, T. (2020). Taxonomy of reinforcement learning algorithms. In *Deep reinforcement learning: Fundamentals, research and applications* (pp. 125–133). Springer Singapore. https://doi.org/10.1007/978-981-15-4095-0_3

Chapter 3
Q-Learning Algorithm

The Q-Learning algorithm is one of the most efficient Reinforcement Learning algorithms (Sutton & Barto, 2020). The following demonstrates step-by-step how it works.

3.1 Operation of the Algorithm

This algorithmwas first proposed as a theorem by Chris Watkins in 1989 and further developed by Watkins himself and Peter Dayan in 1992 (Watkins & Dayan, 1992). The authors have made a significant advance in Reinforcement Learning research.

Being in widespread use, Q-Learning works by successively improving evaluations of the quality of certain actions in certain states. See Fig. 3.1 for the formula that represents how this algorithm works:

To predict the next actions in the context of learning in a complex system, it is not possible to rely only on the next rewards, as this would be a limited view. Thus, Watkins and Dayan's (1992) proposal is to look at the quality of the action. The new quality of action would result from the immediate reward added to the future reward. Thus, we have

- $Q'(s, a)$: new quality to be obtained.
- $Q(s, a)$: the quality value of the state-action.
- α: learning rate (what is the desired relevance of what will be learned). For example, if the value of α is 1, which is the maximum value, the intention is for the machine to learn the maximum. However, there is a threshold between maximum and minimum learning, since by establishing maximum learning quickly—in a single operation—the resulting intensity is lower than in the case of repeating minimum learning several times.
- $R(s, a)$: reward for the current action.

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_3

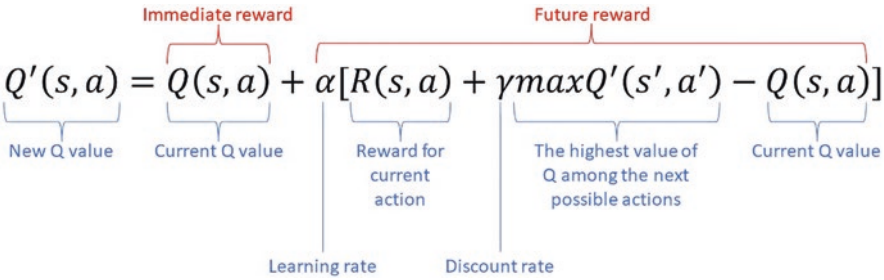
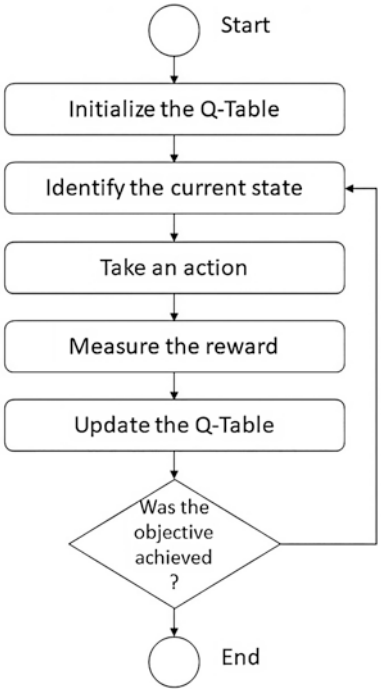


Fig. 3.1 Q-Learning algorithm

Fig. 3.2 Pseudocode of the Q-Learning algorithm



- γ: discount factor.
- max$Q'(s', a')$: the greatest value of Q among the possible actions.

The pseudocode of the Q-Learning algorithm is schematized in Fig. 3.2 to simplify its understanding.

The algorithm can be interpreted as follows:

1. Initialize the Q-value table (i.e., the stock quality table).
2. Observe the current state(s).

3. Based on the selection policy, choose an action (a) to be performed.
4. Take action (a), reach the new state (s'), and obtain the reward (r).
5. Update the Q value for the next state, using the observed reward and the maximum possible reward for the next state.
6. Repeat the process until a terminal state is reached.

The idea of the Q-Learning algorithm is that an agent interacts with a given environment to obtain data that are not previously presented. The agent will then map the set of states, actions, and rewards obtained into a table (Q-Table). This combination of state, action, and reward is called "quality" (Q-Value).

The construction of the Q-Table occurs during the training phase, in which the agent's actions vary between Exploration and Exploitation. Once the Q-Table is learned, it becomes the agent's policy. In other words, the data contained in the Q-Table will dictate the policy of actions. Later, in the test step, the agent will choose the best action from this policy based on the values of Q.

An agent concerned with the best immediate benefit would choose the greatest immediate reward. However, considering the long-term impact of its actions, another better decision could be made. An illustrative case could be losing a pawn to gain a queen in chess. For example, in Fig. 3.3, the agent wants to obtain the function $Q(s, a)$ that predicts the best action "a" in a state "s" to maximize the cumulative reward.

Consider that the agent can only go forward in its actions, as in the previous image. The agent would get +3 if it went south, which is a better immediate reward than −1 if it went east. However, by going south, the cumulative reward will be less, totaling −47 instead of +49. In this sense, the action of moving east will result in a better cumulative reward.

In Fig. 3.4, "Q" illustrates the quality of each movement, that is, each "step" taken by the agent.

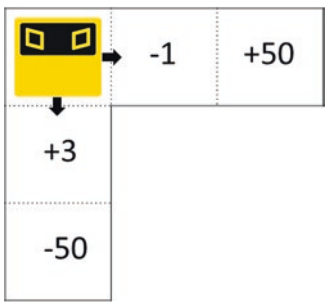


Fig. 3.3 Possible movements of the agent

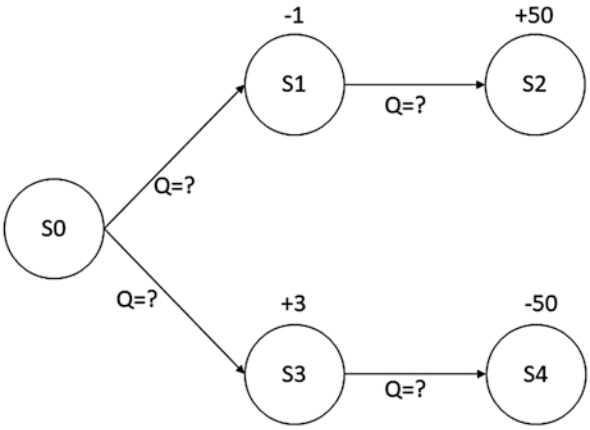


Fig. 3.4 Quality graph of the agent's movements

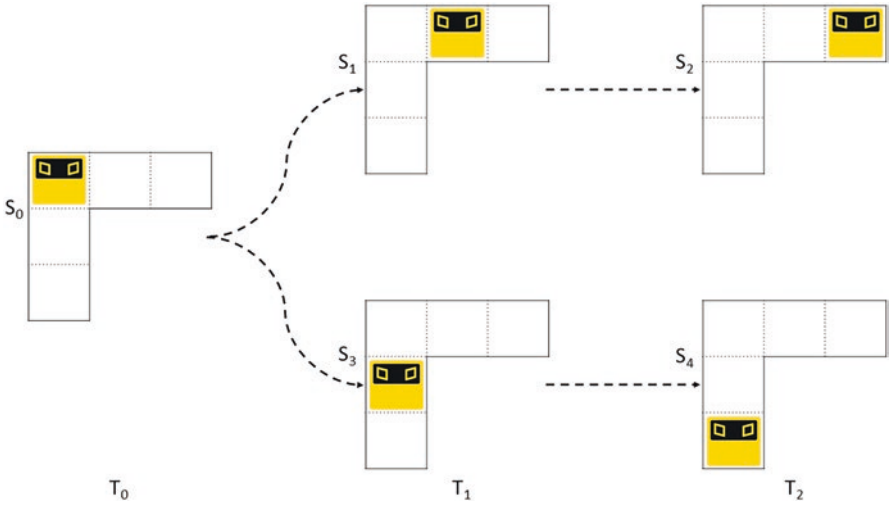


Fig. 3.5 Demonstration of the possible states at each moment

Figure 3.5 illustrates the transition between states depending on each action at each time for the proposed experiment.

3.2 Construction of the Q-Table

Let us illustrate the behavior of a reward-seeking agent in an unknown environment to understand the construction of the Q-Table in Fig. 3.6.

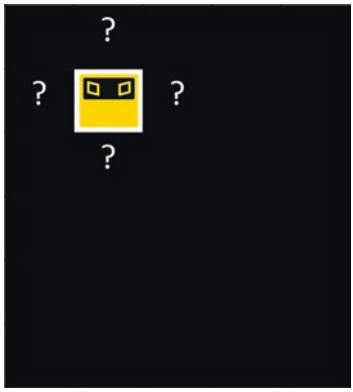


Fig. 3.6 The agent is unaware of the rewards

The goal is to find the sequence of actions that will generate the best result and to maximize the sum of the rewards. We can set some values for the system to meet the requirements of the RL *framework:*

1. states: 11 states;
2. stocks: 4 (north, south, west, and east);
3. rewards:

   - reach the green position: +50;
   - to reach the red position: −50;
   - any action: −1.

In addition, the robot must decide between Exploitation or Exploration behavior.

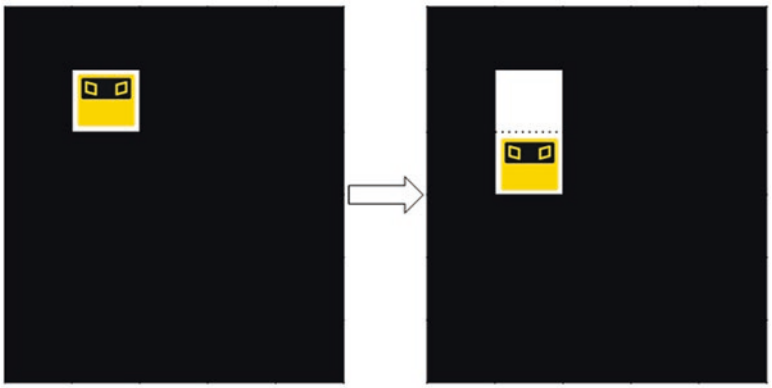
Episode 1, step 1

In step 1, the agent is in state S_0 (0,0), and its action will be to move south (Fig. 3.7). Remember: there is no data in Q-Table; yet, in this training phase, the agent will interact with the environment to get it. The goal is that in the future, intelligent actions will be taken (Russell & Norvig, 2021).

With the first action, the agent goes to state S_3 (1,0), obtaining a reward of −1 (Fig. 3.8).

Let us consider the default reward of −1 for each move, the learning rate of 0.5, and the discount rate of 0.9. By applying the algorithm, you can insert the first piece of data in the Q-Table. Note that the filling of the Q-Table follows the formula of the algorithm:

$$\begin{aligned}\text{Quality} = &\left(1 - \text{“ LearningRate”}\right) * \text{“ Current}Q\left(s,a\right)\text{”} + \text{“ LearningRate”} \\ &* \left(\text{“CurrentReward”} + \text{“ DiscountRate”} * \text{“ max}\, Q\left(s',a'\right)\text{”}\right)\end{aligned} \tag{3.1}$$



Current state	S0 (0,0)
Action	South
Current Q(s,a)	0

Future state	S3 (1,0)
MaxQ(s',a')	0
Reward	-1

Fig. 3.7 Episode 1, step 1

Q-Table	South	North	East	West
State S0 (0,0)	-0.5	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	0	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	0	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	0	0
State S9 (3,1)	0	0	0	0
State S10 (3,2)	0	0	0	0

Fig. 3.8 Table of the quality of the movements in step 1

Therefore, with the current values, you have:

Reward = (1 –C1)*B8 +C1*(B6 +C2*B7)
Reward = (1 –0.5)*0 +0.5*(−1 +0.9*0)
Reward = −0.5

For each action executed, a piece of information is inserted in the Q-Table. Another way to read the stored information would be that, in state S_0, by taking the southern action, the agent obtained −0.5.

After this action, state S_3 (1,0) was reached.

Episode 1, step 2

Taking the southern action once again, the agent goes to state S_5 (2,0) (Fig. 3.9).

Thus far, the maximum quality remains at zero, as well as the current quality (Fig. 3.10).

From the calculation, one more piece of information is stored in the Q-Table.

Episode 1, step 3

The agent is now in state S_5 (2,0) (Fig. 3.11).

Again, moving southward, the agent reaches the next state, designated State S_8 (3,0). The maximum quality and the current quality remain at zero (Fig. 3.12).

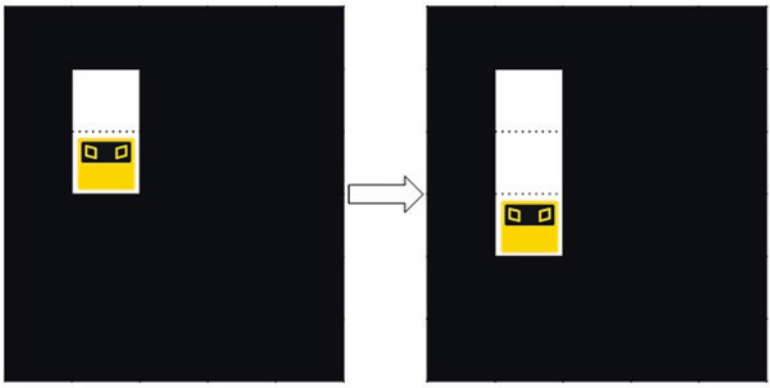
Episode 1, step 4

Starting from State S_8 (3,0), the agent will then move eastward (Fig. 3.13).

Thus, the agent arrives at state S_9 (3,1) after obtaining reward −1. In the Q-Table, one more piece of information is stored from the algorithm's formula (Fig. 3.14).

Episode 1, step 5

Moving once again in an easterly direction, from state S_9 (3,1), the agent reaches state S_{10} (3,2) (Fig. 3.15).



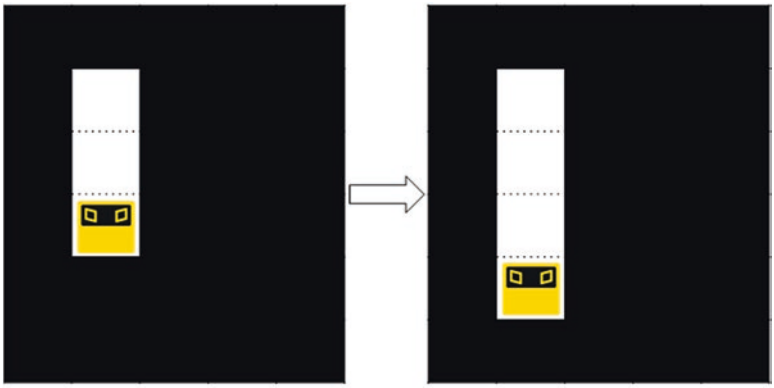
Current state	S3 (1,0)
Action	South
Current Q(s,a)	0

Future state	S5 (2,0)
MaxQ(s',a')	0
Reward	-1

Fig. 3.9 Episode 1, step 2

Q-Table	South	North	East	West
State S0 (0,0)	-0.5	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.5	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	0	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	0	0
State S9 (3,1)	0	0	0	0
State S10 (3,2)	0	0	0	0

Fig. 3.10 Table of the quality of movements in step 2



Current state	S5 (2,0)
Action	South
Current Q(s,a)	0

Future state	S8 (3,0)
MaxQ(s',a')	0
Reward	-1

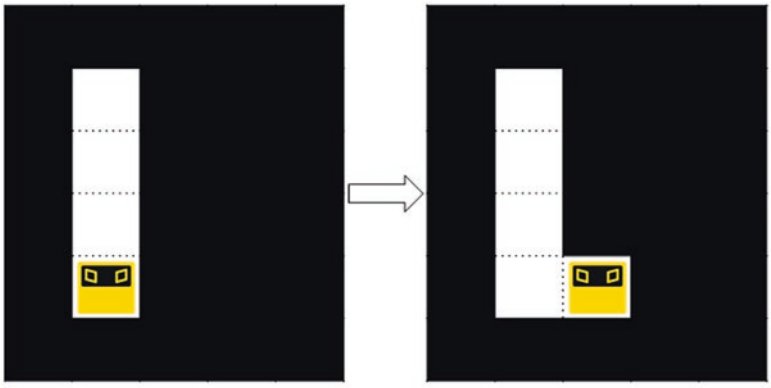
Fig. 3.11 Episode 1, step 3

The value of the current action is zero, as well as the maximum value of future actions. This state marks the end of Episode 1.

With this move, the agent reaches state S_{10} (3,2), in the green position, obtaining the +50 reward (Fig. 3.16).

Q-Table	South	North	East	West
State S0 (0,0)	-0.5	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.5	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.5	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	0	0
State S9 (3,1)	0	0	0	0
State S10 (3,2)	0	0	0	0

Fig. 3.12 Table of the quality of movements in step 3



Current state	S8 (3,0)
Action	East
Current Q(s,a)	0

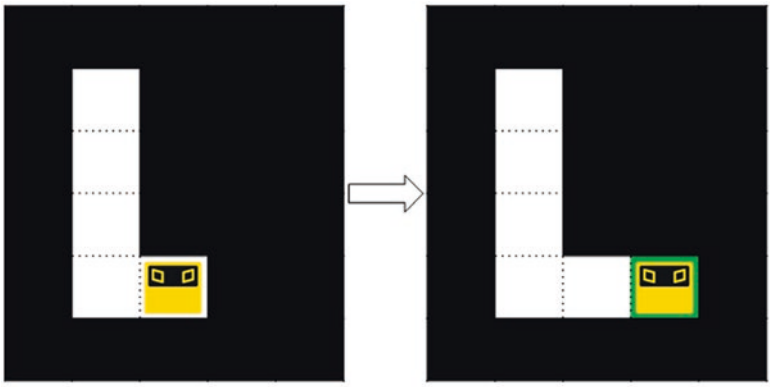
Future state	S9 (3,1)
MaxQ(s',a')	0
Reward	-1

Fig. 3.13 Episode 1, step 4

Note that the reward is +50, and when you apply the calculation described above, you get that the quality of the action was +25. The Q-Table notes the +25; after all, this is not a reward table but a table that records the quality of the actions.

Q-Table	South	North	East	West
State S0 (0,0)	-0.5	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.5	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.5	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	-0.5	0
State S9 (3,1)	0	0	0	0
State S10 (3,2)	0	0	0	0

Fig. 3.14 Table of the quality of movements in step 4



Current state	S9 (3,1)
Action	East
Current Q(s,a)	0

Future state	S10 (3,2)
MaxQ(s',a')	0
Reward	50

Fig. 3.15 Episode 1, step 5

All actions, up to this point, have been Exploration since there is not enough information for Exploitation.

Since it has reached an end state, the action will start again.

Q-Table	South	North	East	West
State S0 (0,0)	-0.5	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.5	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.5	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	-0.5	0
State S9 (3,1)	0	0	25	0
State S10 (3,2)	0	0	0	0

Fig. 3.16 Table of the quality of movements in step 5

Episode 2, step 6

In this episode, the agent starts over at the initial position, in state S_0 (0,0), but already with some knowledge regarding the data entered in Q-Table throughout Episode 1.

Episode 2 then begins at step six. The agent will take the southern action (Fig. 3.17).

In this case, the same actions of the previous episode will be taken to demonstrate how the valuation of these actions in question occurs, that is, how the quality of the actions improves (or decreases) as the agent repeats them. For this reason, the process is called Reinforcement Learning.

The agent takes the southern action and reaches the next state, which will be state S_3 (1,0). The reward for each action is -1, and the maximum quality remains at zero, but according to the data in Q-Table, the current quality $Q(s, a)$ is -0.5 (Fig. 3.18).

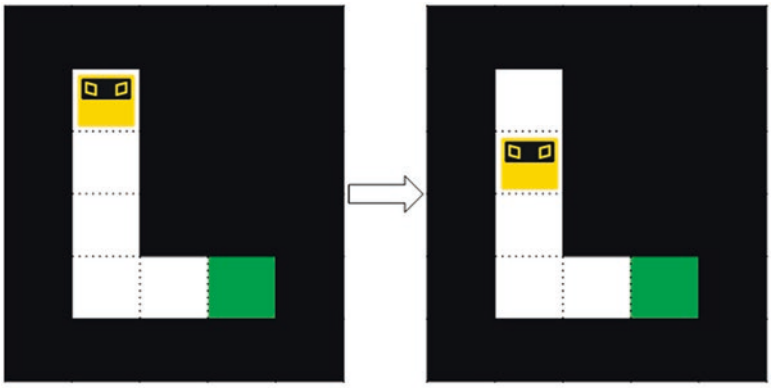
When applying the Q-Learning algorithm, according to the equation, the Q-Table is updated as a result of the action.

Thus, the value of state S_0 (0,0) to the south is updated to -0.75, indicating a worsening in quality. In other words, this action can be understood as something not very good.

Episode 2, step 7

After the action, the agent is in state S_3 (1,0). Upon taking the next step, its future state will be S_5 (2,0) (Fig. 3.19).

Again, based on the data obtained by applying the algorithm, the updated information indicates a worsening or reduction in a reward. This is not a very good action either (Fig. 3.20).



Current state	S0 (0,0)
Action	South
Current Q(s,a)	-0.5

Future state	S3 (1,0)
MaxQ(s',a')	0
Reward	-1

Fig. 3.17 Episode 2, step 6

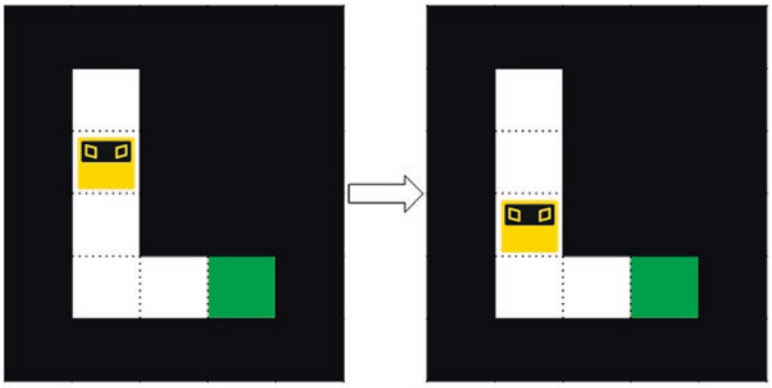
Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.5	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.5	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	-0.5	0
State S9 (3,1)	0	0	25	0
State S10 (3,2)	0	0	0	0

Fig. 3.18 Table of the quality of movements in step 6

Episode 2, step 8

Starting from state S_5 (2,0), the agent moves southward again (Fig. 3.21).

Being updated with the quality value of this action, the Q-Table will also indicate a reduced number (Fig. 3.22).



Current state	S3 (1,0)
Action	South
Current Q(s,a)	-0.5

Future state	S5 (2,0)
MaxQ(s',a')	0
Reward	-1

Fig. 3.19 Episode 2, step 7

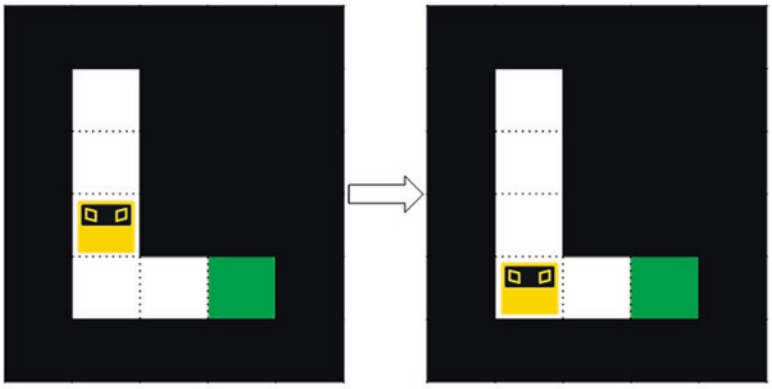
Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.5	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	-0.5	0
State S9 (3,1)	0	0	25	0
State S10 (3,2)	0	0	0	0

Fig. 3.20 Table of the quality of movements in step 7

Episode 2, step 9

The agent thus reaches state S_8 (3,0). From there, it moves eastward (Fig. 3.23).

Starting from state S_8 (3,0) and moving east, there is a change. It is already known that the agent will arrive at state S_9 (3,1) (Fig. 3.24). Additionally, the maximum



Current state	S5 (2,0)
Action	South
Current Q(s,a)	-0.5

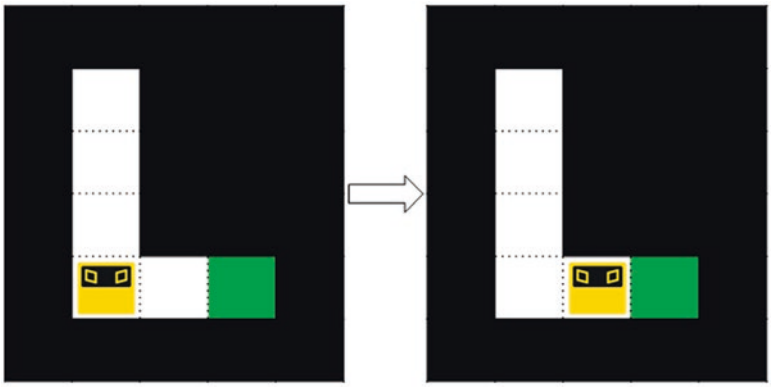
Future state	S8 (3,0)
MaxQ(s',a')	0
Reward	-1

Fig. 3.21 Episode 2, step 8

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	-0.5	0
State S9 (3,1)	0	0	25	0
State S10 (3,2)	0	0	0	0

Fig. 3.22 Table of the quality of movements in step 8

quality value of the action is 25. The algorithm updates the Q-Table, and the quality value of this action is 10.5. It is important to note that this is not the value of the reward but the quality of the action in state S_8 (3,0) as it moves east. In other words, the algorithm qualifies the action.



Current state	S8 (3,0)
Action	East
Current Q(s,a)	-0.5

Future state	S9 (3,1)
MaxQ(s',a')	25
Reward	-1

Fig. 3.23 Episode 2, step 9

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	25	0
State S10 (3,2)	0	0	0	0

Fig. 3.24 Table of the quality of movements in step 9

At this point, notice that the benefits of the reward begin to radiate into the "*q*" values. The quality of this action, which was −0.5, has gone to +10.5, that is, that future reward is already indicating that this is a good action.

Episode 2, step 10

Starting from state S_9 (3,1), the agent will move east again (Fig. 3.25). In advance, the reward for this action is known to be +50. The maximum action quality value in state S_{10} thus far is 0, and the current action quality is 25.

Thus, applying the algorithm, Q-Table presents the updated value of the action quality for the East when the agent is in state S_9 (3,1) (Fig. 3.26).

It can be seen that by retracing the route, there is a reinforcement of learning, both from the negative values and from the propagation of the reward in the specific actions to the east, which are positive. As the movement is repeated, the agent reinforces the states and echoes the information to the movement before the one that receives the positive reward.

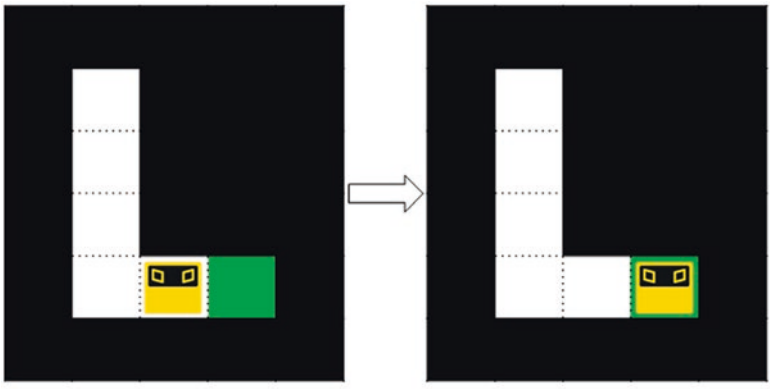
Episode 3, step 11

This episode involves a scenario where there is already a presence of data, and it is already possible to get some insight from what has been constructed in the Q-Table. In Episode 3, the agent is in S-State$_0$ (0,0) but will not repeat the moves of the previous episodes. Thus, starting from state S_0 (0,0), its action will be in the east direction (Fig. 3.27).

By applying the algorithm's formula, considering the maximum value of the action quality as 0, the Q-Table can be updated (Fig. 3.28).

Episode 3, step 12

Moving once more to the east, one obtains the value of the action quality starting from state S_1 (0,1) (Fig. 3.29).



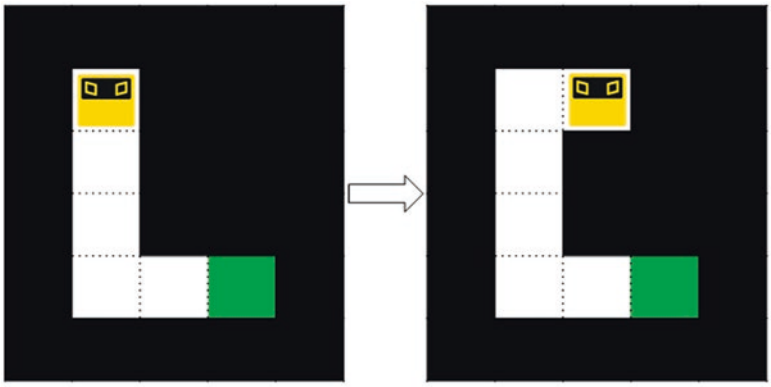
Current state	S9 (3,1)
Action	East
Current Q(s,a)	25

Future state	S10 (3,2)
MaxQ(s',a')	0
Reward	50

Fig. 3.25 Episode 2, step 10

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	0	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.26 Table of the quality of movements in step 10



Current state	S0 (0,0)
Action	East
Current Q(s,a)	0

Future state	S1 (0,1)
MaxQ(s',a')	0
Reward	-1

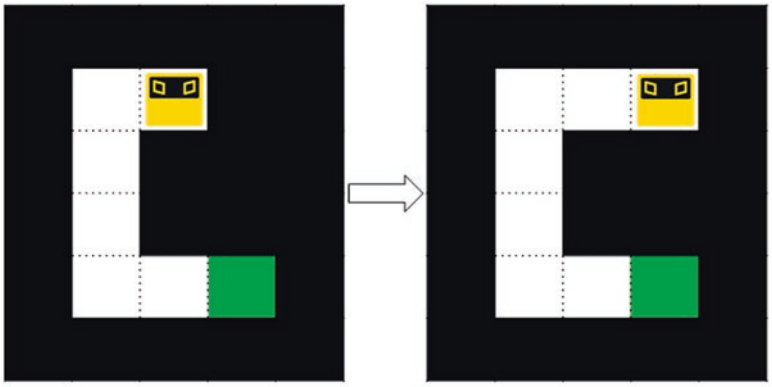
Fig. 3.27 Episode 3, step 11

The Q-Table is updated as shown (Fig. 3.30).

In practice, it is possible to establish that the agent takes random actions, both Exploration and Exploitation.

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	-0.5	0
State S1 (0,1)	0	0	0	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.28 Table of the quality of movements in step 11



Current state	S1 (0,1)
Action	East
Current Q(s,a)	0

Future state	S2 (0,2)
MaxQ(s',a')	0
Reward	-1

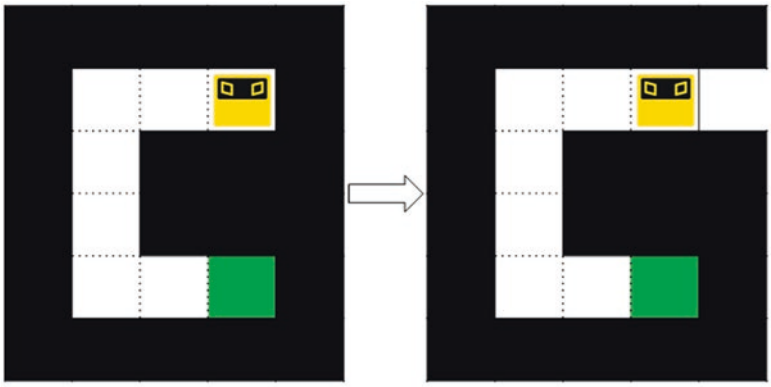
Fig. 3.29 Episode 3, step 12

Episode 3, step 13

Moving again in an easterly direction, it can be seen that starting from state S_2 (0,2), the agent encounters a "wall" and stays in that state, as illustrated below (Fig. 3.31).

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	-0.5	0
State S1 (0,1)	0	0	-0.5	0
State S2 (0,2)	0	0	0	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.30 Table of the quality of movements in step 12



Current state	S2 (0,2)
Action	East
Current Q(s,a)	0

Future state	S2 (0,2)
MaxQ(s',a')	0
Reward	-1

Fig. 3.31 Episode 3, step 13

With the application of the algorithm, the Q-Table is once again updated. In this sense, it is possible to see that by following the eastward direction, the agent is not able to go very far. In other words, the agent stayed in state S_2 (0,2), and the reward quality was kept at −0.5 (Fig. 3.32).

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	-0.5	0
State S1 (0,1)	0	0	-0.5	0
State S2 (0,2)	0	0	-0.5	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.32 Table of the quality of movements in step 13

Episode 3, step 14

Continuing on, from state S_2 (0,2), the agent will take the action southward (Fig. 3.33).

The agent then reaches state S_4 (1,2). Recall that the reward for each action is −1, with the maximum quality of the action still at a value of 0 and the current quality also 0 (Fig. 3.34).

Episode 3, step 15

Starting from state S_4 (1,2), the agent will once again move south (Fig. 3.35). Its future position will be in state S_7 (2,2), in which a change will occur since this "house" corresponds to the red position, for which the reward is −50.

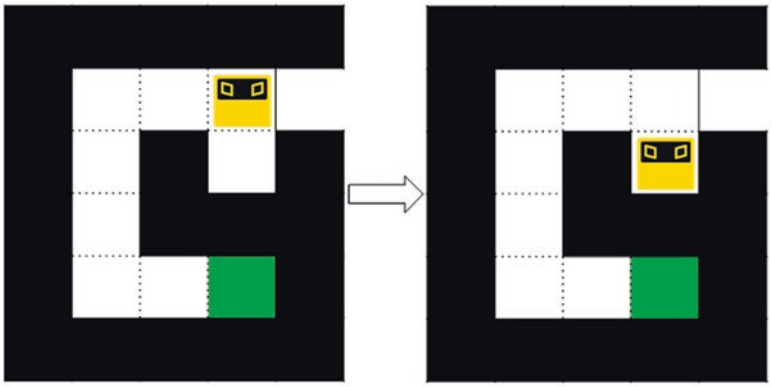
Applying the formula of the algorithm, the Q-Table will be filled with the value of the quality of the executed action—starting from state S_4 (1,2) to the south, the quality of the action is negative, at a value of −25 (Fig. 3.36).

200 episodes later...

After approximately 200 episodes, the agent mapped the entire environment (Fig. 3.37).

After the agent moved and traced several paths on the map, varying between Exploration and Exploitation, it was possible to collect a series of data to build the Q-Table. The values indicated in the table are approximate for the purpose of illustrating the process (Fig. 3.38).

The best actions according to the application of the Q-Learning algorithm are indicated in bold. There are two situations, indicated in red, where the quality of the action is at its lowest value, starting from state S_4 (1,2) to the south and from state S_6 (2,1) to the east. In these, the value −50 is obtained. On the other hand, when the agent is in state S_9 (3,1) and takes the action toward the east, the quality of the action is maximum, obtaining 50, according to the highlighted green box in Q-Table.



Current state	S2 (0,2)
Action	South
Current Q(s,a)	0

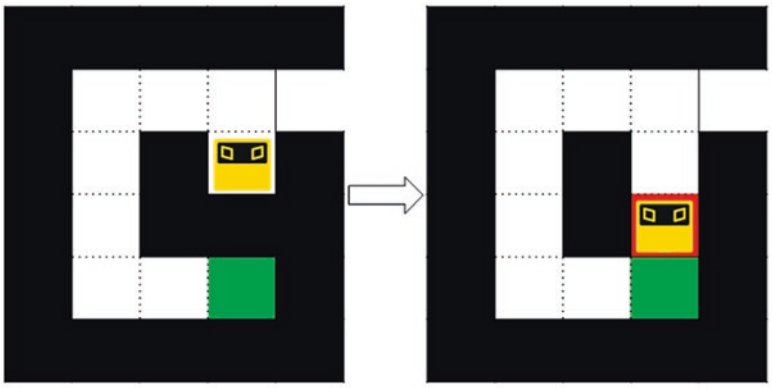
Future state	S4 (1,2)
MaxQ(s',a')	0
Reward	-1

Fig. 3.33 Episode 3, step 14

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	-0.5	0
State S1 (0,1)	0	0	-0.5	0
State S2 (0,2)	-0.5	0	-0.5	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	0	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.34 Table of the quality of movements in step 14

There are other ways to represent these actions and their respective qualities. One of them is called the "Grid World" (Oh et al., 2020), which shows the values according to each direction illustrated (Fig. 3.39).



Current state	S4 (1,2)
Action	South
Current Q(s,a)	0

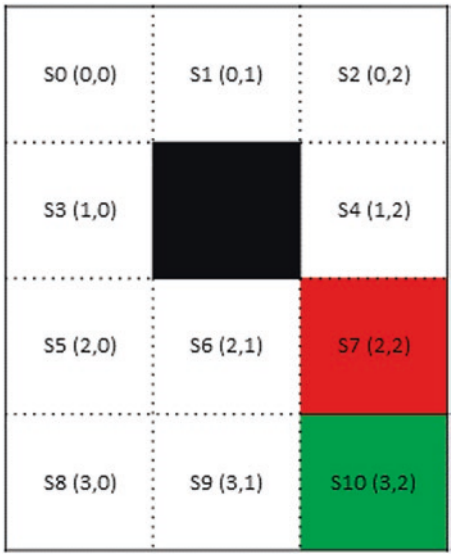
Future state	S7 (2,2)
MaxQ(s',a')	0
Reward	-50

Fig. 3.35 Episode 3, step 15

Q-Table	South	North	East	West
State S0 (0,0)	-0.75	0	-0.5	0
State S1 (0,1)	0	0	-0.5	0
State S2 (0,2)	-0.5	0	-0.5	0
State S3 (1,0)	-0.75	0	0	0
State S4 (1,2)	-25	0	0	0
State S5 (2,0)	-0.75	0	0	0
State S6 (2,1)	0	0	0	0
State S7 (2,2)	0	0	0	0
State S8 (3,0)	0	0	10.5	0
State S9 (3,1)	0	0	37.5	0
State S10 (3,2)	0	0	0	0

Fig. 3.36 Table of the quality of the movements in step 15

Fig. 3.37 Episode 200



Q-Table	South	North	East	West
State S0 (0,0)	**21**	-6.3	-6.3	-6.3
State S1 (0,1)	-6.3	-6.3	-6.3	**-2.7**
State S2 (0,2)	-6.3	-6.3	-6.3	**-2.7**
State S3 (1,0)	**26**	-6.3	-6.3	-6.3
State S4 (1,2)	-50	**-2.7**	-6.3	-6.3
State S5 (2,0)	**41**	-6.3	-6.3	-6.3
State S6 (2,1)	**20**	-6.3	-50	-6.3
State S7 (2,2)	0	0	0	0
State S8 (3,0)	-6.3	-6.3	**46**	-6.3
State S9 (3,1)	-6.3	-6.3	50	-6.3
State S10 (3,2)	0	0	0	0

Fig. 3.38 Final Q-Table

In addition, it is possible to represent the quality of these actions with the highlighted policy (Fig. 3.40). This policy was discovered by the algorithm, with the application of Q-Learning, which was able to build the Q-Table during the training phase, enabling the other representations. By learning the policy, the agent knows which actions to take, starting from each state, to obtain the best rewards.

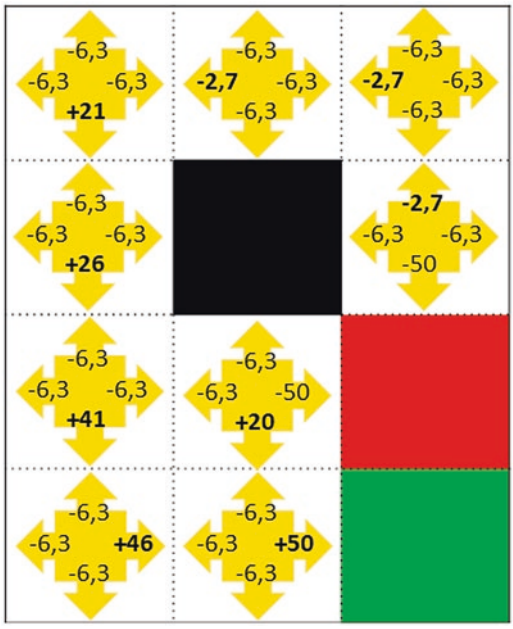


Fig. 3.39 Grid world

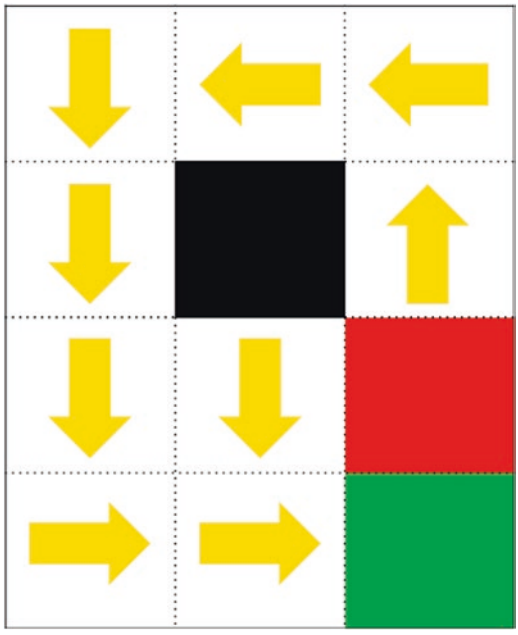


Fig. 3.40 Policy

From what has been learned, when moving on to a testing phase, the knowledge registered in the Q-Table will be applied. If, during training, the agent was performing both Exploration and Exploitation, in the testing stage, the objective is that the intelligent agent only performs Exploitation actions.

References

Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H., Singh, S., & Silver, D. (2020). *Discovering reinforcement learning algorithms*. Advances in neural information processing systems, 2020-Decem. https://doi.org/10.5555/3495724.3495814

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th-Glob ed.). Perarson.

Sutton, R. S., & Barto, A. G. (2020). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292. https://doi.org/10.1023/A:1022676722315

Chapter 4
Development Tools

Some tools allow interactions and applications of Reinforcement Learning (Sutton & Barto, 2020). Because they are already built, they are suitable for exercising the implementation of algorithms such as Q-Learning and the development of code in a dynamic way.

These tools include OpenAI Gym, TF-Agents, Keras, and Reinforcement Learning Toolbox. In addition, we will look at some related data repositories.

4.1 OpenAI Gym

OpenAI Gym is an open-source toolkit for developing and comparing algorithms in Machine Learning Python (Brockman et al., 2016). It supports teaching agents in a variety of areas, from walking or manipulating objects to playing Atari games.

> Learning more:
> https://www.gymlibrary.dev
> On the official OpenAI Gym GitHub:
> https://github.com/openai/gym

The platform provides several Atari games that can be used to test programming skills.

> Some Atari games are available for testing at:
> https://www.gymlibrary.dev/environments/atari/

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_4

To train algorithms and code, you can access five classic control environments: Acrobot, CartPole, Mountain Car, Continuous Mountain Car, and Pendulum.

These classic controls can be checked out in:
https://www.gymlibrary.dev/environments/classic_control/

The code used for the example in the Taxi environment is also available on the page in the "Toy Text" section.

Toy Text:
https://www.gymlibrary.dev/environments/toy_text/

In Multi-Joint dynamics with Contact (MuJoCo), the agent must learn to walk, with limitations already determined according to each case.

The MuJoCo environment is available from the link:
https://www.gymlibrary.dev/environments/mujoco/

The Q-Learning algorithm can be employed in several cases made available on the site. In addition, other algorithms can also be tested and applied.

4.2 TF-Agents

TensorFlow was developed by Google and is an open-source platform. TF-Agents makes it easy to design, implement, and test new Reinforcement Learning algorithms in Python by providing well-tested modular components that can be modified. It allows rapid code interaction, with good test integration and benchmarking.

Learning more:
https://www.tensorflow.org/agents

It is possible to train with the DQN algorithm, which is an evolution of Q-Learning (Mnih et al., 2015). On the page, comments are available, making navigation easier.

Try the tutorial at the address:
https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

4.3 Reinforcement Learning Toolbox

MATLAB software (MATrix LABoratory) has brought, in its recent version, several improvements and advances in the area of Reinforcement Learning. The Reinforcement Learning Toolbox software offers functions to train agents and validate the training results through simulation.

You can get it from this address:
https://www.mathworks.com/products/reinforcement-learning.html

Video tutorials and courses are also available on the main page.

Learning more:
https://www.mathworks.com/videos.html

4.4 Keras

Keras is an Application Programming Interface (API) that aims to simplify Machine Learning work, from data management to hyperparameter definition and deployment solutions. This API is written in Python and runs on top of TensorFlow. Keras is used in many other scientific organizations around the world.

Some examples of using Keras to build applications based on Reinforcement Learning can be seen at:
https://keras.io/examples/rl/

Some code is provided for training in RL. Among the challenges is the CartPole problem, in which the agent must keep a stick balanced on top of a cart on a rail. In this way, the agent has to apply an appropriate force to move the cart to the right or left. For each step in which the stick remains in an upright position, the agent is rewarded. Two different ways to solve this problem using Keras are demonstrated in the given address.

4.5 Data Sources

Some online repositories also make datasets available for experimentation.

Kaggle is a free platform that provides datasets and competitions in data science around the world and an environment for development on Jupyter Notebooks.

Learn more about Kaggle at:
https://www.kaggle.com/

OpenML is an open platform for sharing datasets, algorithms, and experiments for research and learning. It makes it easy to import and export datasets, tasks, and experiments into your Machine Learning environments and libraries (Russell & Norvig, 2021).

Learn more about OpenML at:
https://www.openml.org/

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. https://doi.org/10.48550/arXiv.1606.01540

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th-Glob ed.).

Sutton, R. S., & Barto, A. G. (2020). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Chapter 5
Practice with Code

For this practice, we will use the OpenAI Gym environment with the Taxi experiment in its third version. The objective is to teach the machine to pick up the person at the blue dot and take them to the purple dot, which was originally proposed by Dietterich (2000).

> Taxi v3 environment, proposed by OpenAI:
> https://www.gymlibrary.dev/environments/toy_text/taxi

The focus of this practice is to build an AI that knows how to operate within the environment in question since it is already properly structured. The developer's job is to program an Artificial Intelligence based on Reinforcement Learning (Sutton & Barto, 2020).

Figure 5.1 presents the proposed scenario, with a 5 × 5 matrix.

The cab is the agent. There are four locations, marked with letters, at which the passenger can be picked up or dropped off. According to RL's framework, this problem can be interpreted as follows:

1. States: the environment is 5 × 5, i.e., the cab can be in 25 positions. The passenger can be in 5 positions, i.e., R, G, Y, B, or inside the cab. In turn, the passenger can be dropped off at the 4 identified destinations. Therefore, the size of the environment is 25*5*4, so there are 500 possible states.
2. Actions:
   - move north;
   - move south;
   - move to the east;
   - move west;
   - pick up the passenger; and
   - leave the passenger.

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_5

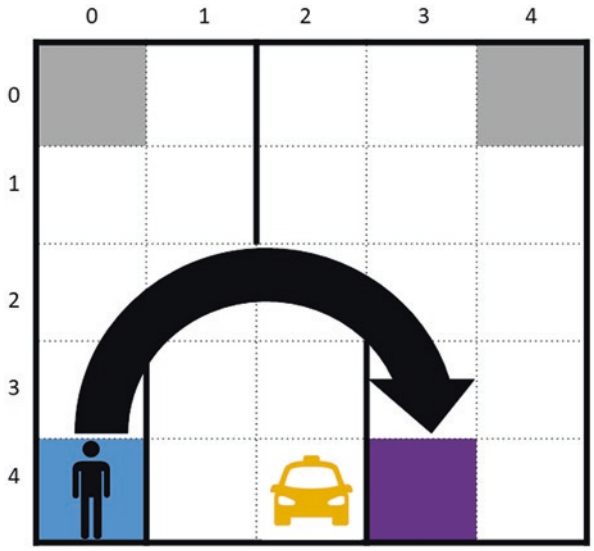


Fig. 5.1 Taxi v3 world grid

Fig. 5.2 Environment rendered in Google Colab

```
  0 1 2 3 4
+---------+
0 |R: | : :G|
1 | : | : : |
2 | : : : : |
3 | | : | : |
4 |Y| :■|B: |
+---------+
```

3. Rewards:

- moving in the environment (living penalty): −1;
- catch the passenger in the wrong place: −10;
- leaving the passenger in the wrong place: −10;
- pick up and drop off the passenger at the correct place: +20.

In the code, the environment is rendered as shown in Fig. 5.2.

It is worth clarifying that there are four boarding and alighting locations: (0,0); (0,4); (4,0); and (4,3). They are referred to as R, G, Y, and B, respectively. The passenger (represented by the letter in blue, in this case, "Y") is to be taken to the landing (represented by the letter in purple, in this case, "B"). The cab is shown in yellow when it is empty (no passenger). When it has a passenger, the taxi is shown in green.

In short, there is a scenario with multiple state possibilities in which the agent positions itself in the environment, undergoing changes. The purpose is to find the maximum quality, building an intelligence capable of performing this action.

Notice in Fig. 5.3 that a poorly trained agent performs many actions to get the passenger from the blue point (Y) to the purple point (R), and only at step 188 achieves the reward of +20, i.e., the maximum reward. This is a 188-step episode, which begins with the agent's first action and ends when the goal of dropping the passenger at the correct place is achieved.

By acting randomly, without intelligence, many steps are taken until an episode is completed. By performing a large number of actions in an exploitative manner, the agent will ultimately obtain a negative reward, which is not interesting for solving the problem.

5.1 Getting to Know the Environment

The code is available in *Python* language and can be run in *Jupyter Notebook* on Google Colab (2022) servers, which support running the code on their GPUs, or it can be run in their Integrated Development Environment (IDE) on the local machine. For this practice, the code is commented on below (Kansal & Martin, 2021).

The code is available at:
https://github.com/rafaeldigital2/RL

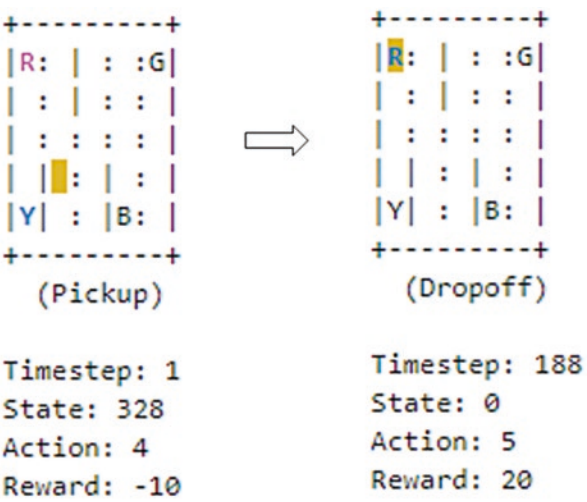


Fig. 5.3 Demonstration of the first and last movement of the taxi

```
!pip install gym==0.21.0
```

Fig. 5.4 Installing library

```
import gym
import random
```

Fig. 5.5 Importing library

```
env = gym.make('Taxi-v3')
```

Fig. 5.6 Setting the environment

```
env.reset()
```

```
483
```

Fig. 5.7 Resetting the environment

```
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B:  |
+---------+
```

Fig. 5.8 Representation of the environment

Installation of the gym library, in this case, version 0.21, which was tested for this practice (Fig. 5.4).

The first step in working on the code is to import the OpenAI Gym library to create the Taxi-v3 environment (Fig. 5.5).

You need to load the Taxi-v3 environment as indicated below (Fig. 5.6).

As noted earlier, there are 500 possible states. Each time the code is executed, a new random state is started. It is necessary to reset the current state of the environment. When it is reset, it will randomly present a new state. Note, in Fig. 5.7, that after resetting the environment, it is in state 483:

When rendered, it presents the configuration shown below (Fig. 5.8). Remember that the empty cab is represented in yellow. In the image, it is in position Y. Your goal is to pick up the passenger at the point represented by the blue letter (B) and drop them off at the point represented by the purple letter (G). In state 483,

```
# 0 = south, 1 = north, 2 = east, 3 = west, 4 = pickup, 5 = dropoff
print("Total number of actions: {}".format(env.action_space))
```

```
Total number of actions: Discrete(6)
```

Fig. 5.9 Quantity of actions

```
print("Total number of states: {}".format(env.observation_space))
```

```
Total number of states: Discrete(500)
```

Fig. 5.10 Quantity of states

```
env.P
```

```
{0: {0: [(1.0, 100, -1, False)],
  1: [(1.0, 0, -1, False)],
  2: [(1.0, 20, -1, False)],
  3: [(1.0, 0, -1, False)],
  4: [(1.0, 16, -1, False)],
  5: [(1.0, 0, -10, False)]},
 1: {0: [(1.0, 101, -1, False)],
  1: [(1.0, 1, -1, False)],
  2: [(1.0, 21, -1, False)],
  3: [(1.0, 1, -1, False)],
  4: [(1.0, 17, -1, False)],
  5: [(1.0, 1, -10, False)]},
 ...
```

Fig. 5.11 Dictionary of the states

according to the image after rendering, it is possible to observe that the passenger is at point G and should be dropped off at point R. There was an interaction between the agent and environment, promoting the change of state.

Note the number of possible actions (Fig. 5.9). We then give the command to print the actions, indicated by numbers. There are six possibilities in total: (0) south; (1) north; (2) east; (3) west; (4) pick up the passenger; and (5) drop off the passenger. The number of actions is shown below.

You can also check the number of state possibilities with the command below (Fig. 5.10).

You can view the 500 states and their respective actions. These states are listed individually, as shown below (Fig. 5.11). For each state, the six possible actions are indicated with their respective information: the probability of performing that action, the next state that action would lead to, the reward to be obtained by that action, and the information on whether the operation achieved the purpose of dropping off the passenger at the correct location. That is, the states are in dictionary form, following the structure "{action: [(probability, next-state, reward, done)]}".

```
env.encode(4, 2, 2, 3)
```

```
451
```

Fig. 5.12 Defining a state

```
env.s = 451
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Fig. 5.13 Rendering the state 451

```
env.P[451]
```

```
{0: [(1.0, 451, -1, False)],
 1: [(1.0, 351, -1, False)],
 2: [(1.0, 451, -1, False)],
 3: [(1.0, 431, -1, False)],
 4: [(1.0, 451, -10, False)],
 5: [(1.0, 451, -10, False)]}
```

Fig. 5.14 Displaying possible actions of state 451

Another way to observe the environment in a defined state would be to indicate the location of each of the elements (Fig. 5.12).

In the case above, the cab is in row 4 and column 2. The passenger leaves point 2 (Y in blue) and should be dropped off at point 3 (B in purple). This is state 451. From the render command, you can see the indicated configuration (Fig. 5.13).

For example, for state 451, the table of rewards for the respective states and actions would be displayed as follows: for each possible action, starting from state 451, the probability of performing that action; the next state if it is taken; the reward is obtained; and whether the end goal has been reached are listed (Fig. 5.14). Then, for example, if the agent takes the action south, starting from state 451, whose probability is 1.0, it will encounter a "wall" and remain in the same state. His reward for the action will be −1, and it will not have reached the end goal.

After executing several actions, in search of the one that would complete the goal, one arrives, for example, at state 479, whose rendering is shown in Fig. 5.15.

To reach the state in question, starting from state 451, a total of 15 steps must be performed. Note that on the 14th step, the taxi reaches state 479, the state in which the cab has reached the point at which the passenger is to be dropped off. Since reaching the goal gives the agent a reward of +20, with the action that cost −14, the agent completes the process with a total reward of +6.

```
env.s = 479
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Fig. 5.15 Rendering the state 479

```
env.P[479]
```

```
{0: [(1.0, 479, -1, False)],
 1: [(1.0, 379, -1, False)],
 2: [(1.0, 499, -1, False)],
 3: [(1.0, 479, -1, False)],
 4: [(1.0, 479, -10, False)],
 5: [(1.0, 475, 20, True)]}
```

Fig. 5.16 Displaying possible actions of state 479

When displaying the history indicators, we notice that starting from state 479 and taking the action of leaving the passenger (5), it is possible to read "True", that is, it is the final movement, which will take you to state 475, reaching your goal (Fig. 5.16).

5.2 Taking Random Actions

We can try to find a solution in the environment without using Reinforcement Learning, using random actions. In this case, the agent has no intelligence yet. This is a force-brutal strategy.

Starting with any state, such as 484, we leave the indication of zero epoch or period and the penalty also zero. The "frames", i.e., the frames for each movement, will be displayed in "[]". The final state has not yet been reached, so "done = False" (Fig. 5.17).

As long as the state is false and the goal has not been reached, a random sample action will be executed, as per command: "action = env.action_space.sample()" (Fig. 5.18). Based on the system, a random action will be taken from among those possible. The randomness indicator is the term "sample". After each move, the result should be presented in the format "state, reward, done, info = env.step(action)". If the reward is −10, add the penalty by 1. All the information is saved to later create the animation.

```
env.s = 484

epochs = 0
penalties = 0

frames = []

done = False
```

Fig. 5.17 Defining variables

```
while not done:
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

    if reward == -10:
        penalties += 1

    frames.append({
        'frame': env.render(mode='ansi'),
        'state': state,
        'action': action,
        'reward': reward
        }
    )

    epochs += 1
```

Fig. 5.18 Taking random actions

Finally, we add the command for the system to print the number of periods or seasons. When the scenario described is executed, the total number of actions executed (528) and the total number of penalties received (145) will be displayed until the final objective is reached (Fig. 5.19).

The movements performed can be demonstrated in the form of an animation (Fig. 5.20).

As you can see in the image, step 528 is precisely the one where the cab drops off the passenger.

5.3 Training with the Algorithm

With the algorithm formula, the import must be performed to create the Q-Table, with 500 rows (corresponding to the states) and 6 columns (corresponding to the actions) (Fig. 5.21).

```
print("Total amount of executed actions: {}".format(epochs))
print("Total amount of penalties received: {}".format(penalties))
```

```
Total amount of executed actions: 528
Total amount of penalties received: 145
```

Fig. 5.19 Quantity of actions and penalties

```
from IPython.display import clear_output
from time import sleep

def print_frames(frames):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print(f"Passo: {i + 1}")
        print(f"Estado: {frame['state']}")
        print(f"Ação: {frame['action']}")
        print(f"Recompensa: {frame['reward']}")
        sleep(.1)

print_frames(frames)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Passo: 578
Estado: 0
Ação: 5
Recompensa: 20
```

Fig. 5.20 Animation of the executed actions

Note that the Q-Table is initialized with zero, without information (Fig. 5.22).

To build the Q-Table, the algorithm must be applied for the agent to build its policy, becoming intelligent.

Notice, below, that the values in, for example, step 451 all initialize to zero (Fig. 5.23).

In the code, you will find the commands to display the processing time in the end and to import a clear output—so that the system does not display each of the outputs.

The hyperparameters refer to the training parameters (Fig. 5.24): learning rate (alpha—α) is how much one wants intelligence to learn; discount factor (gamma—γ) deals with the influence of the future state; and random choice chance (epsilon—ϵ).

```
import numpy as np
q_table = np.zeros([env.observation_space.n, env.action_space.n])
q_table.shape
```

```
(500, 6)
```

Fig. 5.21 Establishing the Q-Table

```
q_table
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

Fig. 5.22 Blank Q-Table

```
q_table[451]
```

```
array([0., 0., 0., 0., 0., 0.])
```

Fig. 5.23 Zeroed Q-Table values

```
alpha = 0.1
gamma = 0.6
epsilon = 0.1
```

Fig. 5.24 Defining hyperparameters

The latter must be a low rate so that the system does not randomly choose a single value frequently.

In this case, we set the values of the hyperparameters manually, but there are other ways to adjust them, such as through an Automated Reinforcement Learning system, as proposed by Souza and Ottoni (2021).

Next, the variables for penalty and reward are created (Fig. 5.25). While the action is not completed, it is established that when the value of "random.uniform(0,1)" is less than the chance of random choice, i.e., 0.1, the system should perform a random action, defined as Exploration.

Otherwise, the action with the highest value should be performed as recorded in the Q-Table that is being built. This is a Nesting action. The focus here is on Exploitation being greater than Exploration, which will still be performed to obtain

```
%%time
for i in range(100000):
  state = env.reset()

  penalties, reward = 0, 0
  done = False
  while not done:
    # When the value of "random.uniform(0, 1)" is less than 0.1 (epsilon),
    if random.uniform(0, 1) < epsilon:
      # will execute a random action (Exploration).
      action = env.action_space.sample()
    # otherwise,
    else:
      # will execute the action with the highest value, according to q_table
      # and informed state (Exploitation)
      action = np.argmax(q_table[state])

    next_state, reward, done, info = env.step(action)

    old_q = q_table[state, action]
    next_max = np.max(q_table[next_state])

    new_q = (1 - alpha) * old_q + alpha * (reward + gamma * next_max)
    q_table[state, action] = new_q

    if reward == -10:
      penalties += 1

    state = next_state

  if i % 100 == 0:
    clear_output(wait=True)
    print('Episode: ', i)

print('Training completed')
```

```
Episode:  99900
Training completed
CPU times: user 1min 30s, sys: 9.5 s, total: 1min 39s
Wall time: 1min 34s
```

Fig. 5.25 Running the training

new data, but on a smaller scale. The next command is for the system to move to the next state once the action is taken. For example, if the agent (cab) was in state 451 and took the north action, it will go to state 351.

The old Q-value will be displayed according to the Q-Table, and the system will search for the next maximum value recorded. Notice, in line 29, the actual application of the Q-Learning algorithm. In this case, it is an indication of updating the

Q-value according to the formula. Thus, the Q-Table will also be updated with the new Q:

$$new\_q = (1 - alpha) * old\_q + alpha * (reward + gamma * next\_max)$$

Next, the penalties are determined. As in the random action process, if the reward is −10, a penalty value of 1 must be added.

On the next line, the command points the update from the current state to the next one. Additionally, the value will be displayed every 100 episodes out of a total of 100,000 episodes as determined in the code. Remember that each episode ends when the agent reaches their goal.

By executing the code, the training will be performed over 100,000 episodes. It is possible to follow the evolution of the process every 100 episodes that, when completed, will be duly registered in the Q-Table with its updated data. Thus, it starts with zero and gradually fills up.

With the Q-Table, instead of having the reward as the metric, you use the quality of the movement estimated by it (Fig. 5.26).

This allows the system to recognize that there are actions that are worse than others and to make decisions intelligently. Look at the data for state 451 below (Fig. 5.27).

5.4 Testing the Q-Table

The following is a step-by-step demonstration of the taxi actions for the best understanding.

```
q_table
```

```
array([[  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ],
       [ -2.41837063,  -2.3639511 ,  -2.41837061,  -2.3639511 ,
         -2.27325184, -11.36394325],
       [ -1.87014398,  -1.45024006,  -1.87014399,  -1.45024001,
         -0.7504    , -10.4502399 ],
       ...,
       [ -1.14795804,   0.41599971,  -1.12348986,  -1.27655957,
         -4.04700446,  -5.67255546],
       [ -2.16963011,  -2.12207299,  -2.13218725,  -2.12207369,
         -6.60750755,  -4.42788627],
       [  3.83805059,   1.25378757,   2.79363391,  11.        ,
         -2.47019099,  -2.25208983]])
```

Fig. 5.26 Q-Table filled in

```
q_table[451]
```

```
array([-2.48617312, -2.48236806, -2.48625681, -2.48236806, -8.10946759,
       -9.68281298])
```

Fig. 5.27 Action quality values in state 451

```
env.s = 451
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```

Fig. 5.28 Rendering the state 451

```
# Sendo: 0 = south, 1 = north, 2 = east, 3 = west, 4 = pickup, 5 = dropoff
q_table[451]
```

```
array([ -2.48783664,  -2.48236806,  -2.48851972,  -2.48236806,
       -10.51500477, -10.51102972])
```

Fig. 5.29 Q values in state 451

Step 1

Having performed the training, the Q-Table was learned. The proposed initial state was 451, as rendered below (Fig. 5.28).

Among the possible actions, the values of the qualities of each are already known. Based on the data, it would be better for the agent to move north (1) or west (3) (Fig. 5.29).

Going north, you get the following result (Fig. 5.30):

Step 2

The agent is in state 351 (Fig. 5.31).

In this state, Q-Table presents the information that, once again, the highest quality stocks are to the north or west (Fig. 5.32).

By choosing to move north, the agent reaches a new state (Fig. 5.33).

Step 3

Here, the agent is in the current state, 251 (Fig. 5.34).

From it, the Q-Table reports that the highest quality action this time is westward (Fig. 5.35).

Reaching the next state (Fig. 5.36):

```
env.step(1)
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | :  | : |
|Y| : |B: |
+---------+
  (North)
```

Fig. 5.30 Taking action 1 and rendering the state

```
print(env.s)
```

```
351
```

Fig. 5.31 Displaying the current state

```
q_table[env.s]
```

```
array([ -2.48749897,  -2.47061344,  -2.48040604,  -2.47061344,
       -11.00736724, -10.70045825])
```

Fig. 5.32 Q values in the current state

```
env.step(1)
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : :  : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
```

Fig. 5.33 Taking action 1 and rendering the state

```
print(env.s)
```

```
251
```

Fig. 5.34 Displaying the current state

```
q_table[env.s]
```

```
array([ -2.48236806,  -2.48236806,  -2.48236806,  -2.4510224 ,
       -11.47061341, -11.47061343])
```

Fig. 5.35 Q values in the current state

```
env.step(3)
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| :▒: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
```

Fig. 5.36 Taking action 3 and rendering the state

```
env.s = 479
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |▒: |
+---------+
  (South)
```

Fig. 5.37 Rendering the state 479

```
q_table[479]
```

```
array([11. ,  5.6,  5.6, 11. ,  2. , 20. ])
```

Fig. 5.38 Q values in the current state 479

Going to the 14th step

Conducting itself in this way, the agent (cab) proceeds, successively, in search of the highest rewards (or the lowest penalties), until it picks up the passenger and drops them off at their destination, which occurs from state 479 in Fig. 5.37.

From Q-Table, it can be seen that in the state in question, the highest quality action is that of dropping off the passenger (5) (Fig. 5.38).

By taking this action, you reach a new state (Fig. 5.39).

Note that, at this instant, the agent has achieved its final objective (Fig. 5.40).

In this state, the Q-Table is reset to zero and a new episode can start (Fig. 5.41).

```
env.step(5)
print(env.render(mode='ansi'))
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```

Fig. 5.39 Taking action 3 and rendering the state

```
print(env.s)
```

```
475
```

Fig. 5.40 Displaying the current state

```
q_table[475]
```

```
array([0., 0., 0., 0., 0., 0.])
```

Fig. 5.41 Q values in state 475

5.5 Testing the Trained Agent

With the learning obtained, the problem is solved more easily. In a sequence of 50 episodes, storing the list of states, the code can be built based on the Q-Table data, according to the maximum value of the quality of each action (Fig. 5.42).

In this way, when executing the 50 episodes, the agent will suffer 0 penalty (Fig. 5.43).

You can also observe each of the executed steps in the list of *frames*. For example, frame 0 is presented as follows (Fig. 5.44):

When you print each episode, the state, the action, and the reward obtained are displayed (Fig. 5.45).

Thus, the system follows the Q-Table policy, and the agent takes the best path to reach the final goal.

```
total_penalties = 0
episodes = 50
frames = []
```

Fig. 5.42 Defining variables

```
for ep in range(episodes):
  state = env.reset()
  penalties, reward = 0, 0
  done = False
  while not done:
    action = np.argmax(q_table[state])
    state, reward, done, info = env.step(action)

    if reward == -10:
      penalties += 1

    frames.append({
        'frame': env.render(mode='ansi'),
        'episode': ep,
        'state': state,
        'action': action,
        'reward': reward
    })

  total_penalties += penalties

print('Episodes', episodes)
print('Penalties', total_penalties)
```

```
Episodes 50
Penalties 0
```

Fig. 5.43 Testing the agent for 50 episodes

```
frames[0]
```

```
{'frame': '+---------+\n|\x1b[34;1mR\x1b[0m: | : :G|\n| : | : : |\n|\x1b[43m \x:
(North)\n',
 'episode': 0,
 'state': 203,
 'action': 1,
 'reward': -1}
```

Fig. 5.44 Registered frame

```
from time import sleep

for frame in frames:
  clear_output(wait=True)
  print(frame['frame'])
  print(f"Episode: {frame['episode']}")
  print('State:', frame['state'])
  print('Action:', frame['action'])
  print('Reward:', frame['reward'])
  sleep(1)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Episode: 49
State: 410
Action: 5
Reward: 20
```

Fig. 5.45 Animation of the episodes

References

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227–303.

Google Colab. (2022). Welcome to Colaboratory - Colaboratory. Getting Started - Introduction. https://colab.research.google.com/

Kansal, S., & Martin, B. (2021). *Reinforcement Q-learning from scratch in Python with OpenAI Gym*. Web Page. https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-opena

Souza, G. K. B., & Ottoni, A. L. C. (2021). AutoRL-TSP-RSM: sistema de aprendizado por reforço automatizado com metodologia de superfície de resposta para o problema do caixeiro viajante. *Revista Brasileira de Computação Aplicada, 13*(3), 86–100. https://doi.org/10.5335/rbca.v13i3.12653

Sutton, R. S., & Barto, A. G. (2020). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Chapter 6
Recent Applications and Future Research

The techniques of Reinforcement Learning have been gaining prominence in many recent studies. Below, we present some pioneering research in major areas of knowledge and their impacts. In some of them, its implementation remains a challenge.

6.1 Artificial General Intelligence

Currently, human designers define in the code the tasks that the agents should optimize. Looking ahead, it will be necessary to design agents that make their own choices, not only about what action to take but also decide which tasks they intend to master. This opens the way for the development of Artificial General Intelligence (AGI).

One such challenge is raised in the article stating that "Reward is enough" by David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton, published in the journal *Artificial Intelligence* in October 2021 (Silver et al., 2021). This paper has had great repercussions in the field since it was developed by renowned authors. Researchers suggest that reward is sufficient to drive intelligent behavior, both natural and artificial. Thus, intelligence and the skills associated with it can be developed through reward maximization. The proposition is that agents that seek to maximize reward may constitute a solution to AGI. This contrasts with the current paradigm of Artificial Intelligence, in which problem-specific formulations are needed.

Although many people subscribe to the hypothesis that reward is enough, there are researchers who disagree, such as Walid Saba, who, in response to the aforementioned article, published "Reward is NOT Enough, and Neither is (Machine) Learning" (Saba, 2021)—"Reward is NOT Enough, and Neither is (Machine) Learning," in free translation. The author argues that reward is not enough and that learning, in general, is not the solution to achieve AGI.

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9_6

For Saba, RL works well for knowledge that can be learned incrementally, such as looking for food when one is hungry. However, the flaw of the reward hypothesis is that it starts from false premises by not considering the difference between learned knowledge and acquired knowledge and assuming that learning survival skills is equivalent to acquiring intelligent behavior. This is because, in more complex problems, knowledge cannot be learned incrementally.

In contrast, Silver et al. (Vamplew et al., 2022) claim that rewards are insufficient to explain some aspects of biological and computational intelligence because reward maximization imposes constraints on behavior. Furthermore, these scholars claim that even if scalar reward functions could trigger intelligent behavior in specific cases, this type of reward is insufficient for the development of human-aligned AGI due to risks of unsafe or unethical behavior. The authors take the opportunity to propose a multiobjective approach for AGI development.

The presentation of disagreements and counterpoints that leads to new research is very important for the technology to advance. To date, AGI has not been reached, so more investment in its development is essential.

6.2 Board Games

RL has also proven effective in developing planning methods for strategic applications such as chess and Go. The model of the environment can be known from the rules of the game or provided by human designers.

Go is a game of conquering territory where the strategy is used to progress on a board. In 2016, the AlphaGo system (Silver et al., 2017) exploited Reinforcement Learning in a competition against Lee Sedol, considered the greatest Go player and who held the title of world champion. The clash took place over five rounds, of which AlphaGo won four. Unfortunately, the film does not delve into the aspects involving the technology itself and how it was developed. The fact that Sedol won one of the rounds demonstrates his skill and also that there is room for improvement in the computational algorithm.

> The AlphaGo documentary can be watched at:
> https://www.alphagomovie.com/

AlphaGo later evolved into the AlphaZero system (Silver et al., 2018), which received only the rules of the game and learned to play against itself, not only Go but also shogi and chess.

More recently, the system has advanced to a version that does not know the rules of games, learning a model of the world, called MuZero (Schrittwieser et al., 2020); in addition to these games, it has also learned some Atari games. This technique allows planning winning strategies in unknown scenarios.

6.3 Digital Games

Digital games present even more challenging systematics. StarCraft II is a famous real-time strategy space game that requires a range of complex skills. AlphaStar was the first AI system to join the top league of the StarCraft II game in 2019. The paper regarding the system and how it works was published in the journal *Nature* (Vinyals et al., 2019).

Using various Machine Learning techniques, including Reinforcement Learning, DeepMind researchers trained AlphaStar to play StarCraft II on the official platform. The system was able to achieve a rating higher than 99.8% of the active players on that platform, reaching the "Grandmaster" level.

More details about the development of AI and the processes until it reaches the top level of the game have been pointed out by DeepMind at https://www.deepmind.com/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning

Dota 2 is a multiplayer strategic battle game. In 2018, the OpenAI Five system beat the champion team, consisting of five players. In this case, the advancement was to the point where the system was able to beat a team of individuals rather than confront a single player.

Details about the AI developed by OpenAI are available at:
https://openai.com/five/

In all 57 games on the Atari console, Agent57 achieved top scores, outperforming humans (Badia et al., 2020). It combines an efficient algorithm that adapts the agent's long- and short-term exploration and behavior.

A more detailed explanation of Agent57 can be found on the DeepMind page:
https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark

6.4 Robotics

OpenAI has also trained a set of artificial neural networks to solve Rubik's cube with a robotic hand. The project involves simulation-trained models that can be used to solve problems in the physical world, including those that require greater dexterity.

A full explanation of Rubik's cube solving is available at:
https://openai.com/blog/solving-rubiks-cube/

A team from the Allen Institute for AI recently announced the release of ProcTHOR (Deitke et al., 2022). ProcTHOR aims to train robots within a virtual environment and then apply the learning to real life. It is a framework for procedurally and arbitrarily generating large sets of interactive 3D houses. It allows large datasets of diverse, interactive, customizable, and high-performance virtual environments to be generated to train and evaluate embedded agents in navigation, interaction, and manipulation tasks. It is an open-source tool in Python, providing a notebook that can be run on Google Colab.

The demo and the ProcTHOR code can be checked on the website:
https://procthor.allenai.org/

One innovative robot simulation platform is the RoboCup 3D Soccer Simulation League (3DSIM), which runs a robot soccer competition with high-quality autonomous humanoid agents. This environment makes it possible to compete against each other in a realistic simulation of the rules and physics of a soccer game and provides a test bench for robotics and Artificial Intelligence.

The 3DSIM can be checked out at:
https://ssim.robocup.org/3d-simulation/

The work of Muzio et al. (2022) used Deep Reinforcement Learning (DRL) techniques, due to its success in continuous control tasks, to develop movements in the RoboCup environment. The authors were dedicated to learning the behaviors of completing the race track as quickly as possible and dodging a single opponent. This approach used a hierarchical controller in which a model-free policy learns to interact with the model-based walking algorithm, and then DRL algorithms were used for the agent to learn to perform these behaviors. In addition, the learned dribbling policy was evaluated within the RoboCup environment.

A team from the BahiaRT project has made available two tools that contribute to this competition (Simões et al., 2022): the BahiaRT Setplays Collecting Toolkit is a software tool that allows soccer fans or experts to watch the robots playing soccer and flag the moves that would suggest a better set play for that robot team, and BahiaRT Gym is a tool for comparing and developing Reinforcement Learning algorithms within this scenario. This makes the software useful to other teams in training their robot teams.

The development of a versatile bipedal walking motor is considered one of the most difficult problems in mobile robotics. They need to be able to dynamically restore their balance when subjected to certain types of external disturbances. The research by Melo et al. (2022) implemented a controller that improves the performance of the walking motor used by a simulated humanoid agent in the 3DSIM environment. The agent learns a motion policy that works well over a wide variety of perturbations.

6.5 Education

We noticed five main lines of research in education that make use of technologies based on Reinforcement Learning (Singla et al., 2021). These applications are mainly focused on personalizing curriculum across tasks, providing hints and quizzes, conducting adaptive experimentation and A/B testing on educational platforms, modeling human student behavior, and generating educational content. All of them are interested in improving human learning.

6.6 Quantum Mechanics

The consolidation of the 2022 Nobel Prize in Physics (NobelPrize.org, 2022) recognizes Quantum Mechanics as a complete theory, overriding the Einstein–Podolsky–Rosen Paradox (EPR) (Einstein et al., 1935). Simm et al. (2020) integrated knowledge from Quantum Mechanics and Reinforcement Learning. The researchers designed the reward function based on the properties of quantum physics. Thus, they were able to design new chemical compounds and made available the molecular design environment MolGym, enabling new theoretical and practical applications.

6.7 Mathematics

Matrix multiplication is an operation used in many computational applications. The current algorithms have been designed and used by humans for more than 50 years (Strassen, 1969). Mathematicians search for ways to optimize these operations and

can use computation to discover new algorithms, but it is not appropriate to search for solutions randomly by brute force.

An initiative by DeepMind designed the search for matrix multiplication algorithms in a game and then used AlphaTensor to play it until they discovered more efficient algorithms (Fawzi et al., 2022). The result was the discovery of matrix multiplication algorithms superior to those designed by humans. Artificial Intelligence itself is already providing ways to become more efficient.

Ultimately, we are also the users of the machine and must strive to design intelligent and safe systems. Policies derived from Reinforcement Learning can guide human decision-makers in managing our society.

Moreover, controversial questions remain paradigms in Artificial Intelligence, such as should an agent act in a way that benefits an individual rather than a group? These questions open opportunities for new engineers and the use of Reinforcement Learning technologies.

References

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., & Blundell, C. (2020). *Agent57: Outperforming the Atari human benchmark*. 37th international conference on machine learning, ICML 2020, PartF168147–1, pp. 484–494.

Deitke, M., VanderBilt, E., Herrasti, A., Weihs, L., Salvador, J., Ehsani, K., Han, W., Kolve, E., Farhadi, A., Kembhavi, A., & Mottaghi, R. (2022). *ProcTHOR: Large-scale embodied AI using procedural generation*. https://doi.org/10.48550/arxiv.2206.06994.

Einstein, A., Podolsky, B., & Rosen, N. (1935). Can quantum-mechanical description of physical reality be considered complete? *Physical Review, 47*(10), 777–780. https://doi.org/10.1103/PHYSREV.47.777/FIGURE/1/THUMB

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, R., F. J, Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., & Kohli, P. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature, 610*(7930), 47–53. https://doi.org/10.1038/s41586-022-05172-4

Melo, D. C., Maximo, M. R. O. A., & da Cunha, A. M. (2022). Learning push recovery behaviors for humanoid walking using deep reinforcement learning. *Journal of Intelligent & Robotic Systems, 106*(1), 8. https://doi.org/10.1007/s10846-022-01656-7

Muzio, A. F. V., Maximo, M. R. O. A., & Yoneyama, T. (2022). Deep reinforcement learning for humanoid robot behaviors. *Journal of Intelligent & Robotic Systems, 105*(1), 12. https://doi.org/10.1007/s10846-022-01619-y

NobelPrize.org. (2022). *The Nobel Prize in Physics 2022*. Nobel Prize Outreach AB 2022. Retrieved 11 Oct 2022, from https://www.nobelprize.org/prizes/physics/2022/summary/

Saba, W. (2021). *Reward is NOT enough, and neither is (Machine) Learning* | by Walid Saba, Ph.D. | ONTOLOGIK | Medium. https://medium.com/ontologik/reward-is-not-enough-and-neither-is-machine-learning-6f9896274995

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., & Silver, D. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature, 588*(7839), 604–609. https://doi.org/10.1038/s41586-020-03051-4

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general rein-

forcement learning algorithm that masters chess, shogi, and Go through self-play. *Science, 362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature, 550*(7676), 354–359. https://doi.org/10.1038/nature24270

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence, 299*, 103535. https://doi.org/10.1016/J.ARTINT.2021.103535

Simm, G. N. C., Pinsler, R., & Hernández-Lobato, J. M. (2020). Reinforcement learning for molecular design guided by quantum mechanics. In *37th international conference on machine learning, ICML 2020* (Vol. PartF16814, pp. 8906–8916). PMLR.

Simões, M. A. C., Mascarenhas, G., Fonseca, R., dos Santos, V. M. P., Mascarenhas, F., & Nogueira, T. (2022). BahiaRT Setplays collecting toolkit and BahiaRT Gym. *Software Impacts, 14*, 100401. https://doi.org/10.1016/j.simpa.2022.100401

Singla, A., Rafferty, A. N., Radanovic, G., & Heffernan, N. T. (2021). *Reinforcement learning for education: Opportunities and challenges overview of the RL4ED workshop at EDM 2021 conference* *.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik, 13*(4), 354–356. https://doi.org/10.1007/BF02165411

Vamplew, P., Smith, B. J., Källström, J., Ramos, G., Rădulescu, R., Roijers, D. M., Hayes, C. F., Heintz, F., Mannion, P., Libin, P. J. K., Dazeley, R., & Foale, C. (2022). Scalar reward is not enough: A response to Silver, Singh, Precup and Sutton (2021). *Autonomous Agents and Multi-Agent Systems, 36*(2), 1–19. https://doi.org/10.1007/S10458-022-09575-5/FIGURES/2

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature, 575*(7782), 350–354. https://doi.org/10.1038/s41586-019-1724-z

Index

R. Ris-Ala, *Fundamentals of Reinforcement Learning*,
https://doi.org/10.1007/978-3-031-37345-9