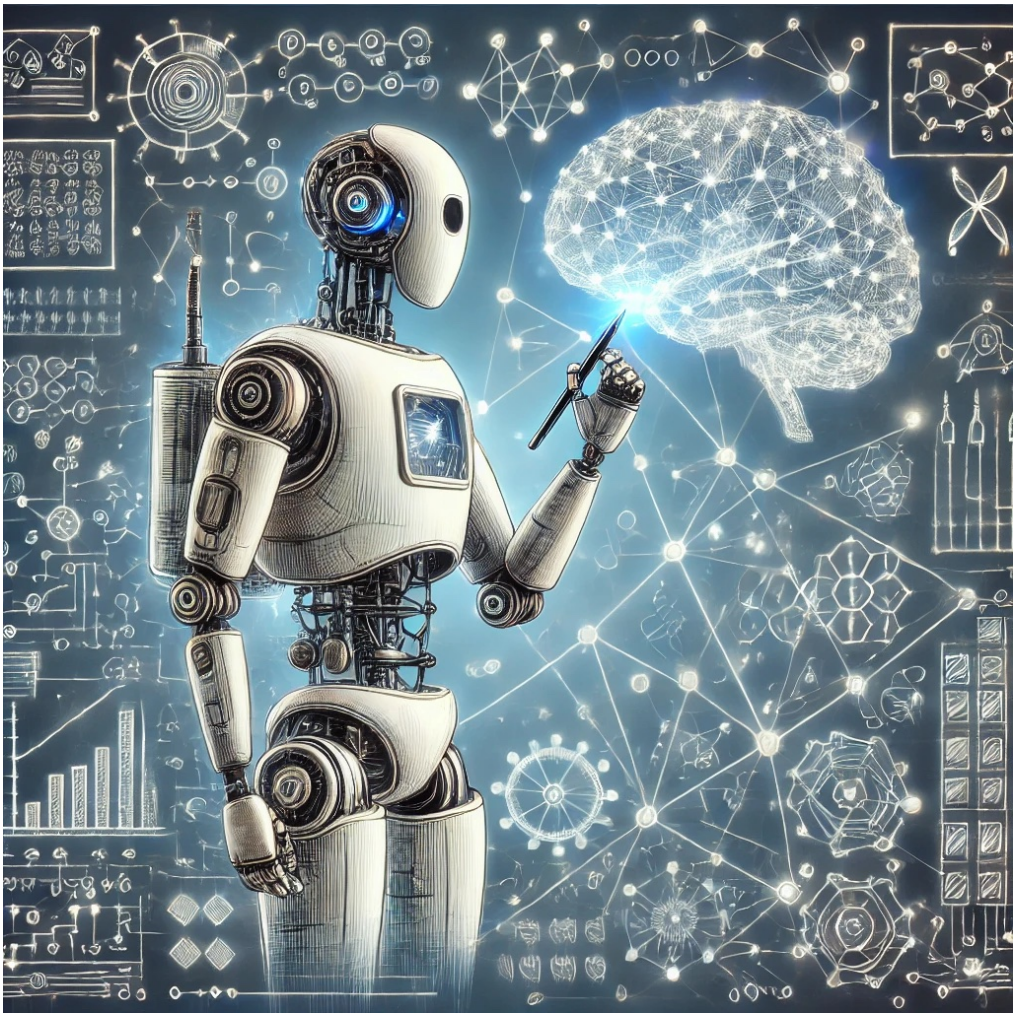
REINFORCEMENT LEARNING: ALGORITHMS & CONVERGENCE

CLEMENS HEITZINGER

mailto:Clemens.Heitzinger@TUWien.ac.at
http://Clemens.Heitzinger.name

Edition 0.1 — June 24, 2025



Contents

Preface

Reinforcement learning (RL) is an incredibly appealing subject. Firstly, it is a very general concept: An agent interacts with an environment with the goal to maximize the rewards it receives. The environment is random and communicates states and rewards to the agent, while the agent chooses actions according to a possibly random policy. The challenge is to find policies that maximize the expected value of all future rewards. Because reinforcement learning is such a general concept, it encompasses many real-world applications and is at the core of artificial intelligence and machine learning.

Secondly, the concepts and algorithms developed in reinforcement learning are leading to more and more general capabilities of artificial-intelligence (AI) systems and will very likely be part of artificial general intelligence. There are many similarities between reinforcement learning and how (human) brains work. These similarities must be explored in order to further our understanding of brains and human intelligence.

Thirdly, superhuman capabilities of RL learning algorithms have been demonstrated in various areas, and the list of extraordinary capabilities of AI systems built on reinforcement learning is continually expanding. Prominent examples are playing backgammon, Atari 2600 games, many more computer games, card games, chess, Go, and shogi at superhuman levels. Probably most famously, however, reinforcement learning is the last and crucial step in training large language models such as ChatGPT.

This book derives and describes the theory of reinforcement learning. The most fundamental as well as the most powerful RL algorithms are discussed. In addition to calculating optimal policies using powerful learning algorithms, we must also strive to provide guarantees regarding the quality of the learned results, i.e. the performance of the policies calculated by the algorithms. Hence the two main kinds of theoretical results concern the convergence of an algorithm to an optimal policy and performance guarantees for these policies. This book collects the theoretic foundations of reinforcement learning in view of these main questions.

To help translate theory into practice, this book also includes pseudocode and programming exercises that are concerned with the implementation of algorithms. The purpose of the programming exercises is to gain working knowledge, to try to exhibit both the advantages and disadvantages of certain methods, to show the challenges faced when developing new algorithms, and to inspire the reader to experiment with the algorithms. Both theoretic and programming exercises are an invitation to the reader to further explore this captivating subject.

Clearly, it was necessary to make choices – in many cases hard ones – about which algorithms, theorems, and proofs could be included in this book. Preference has been given to the more fundamental and general ones. In any case, care was taken to provide a complete and self-contained treatment. The appendix collects definitions, concepts, and results coming from outside reinforcement learning in a form that is expedient for our use here. Depending on the background of the reader, the appendix can be skipped in whole or in part.

This book can be used in various ways: It can be used for self-study, but also as the basis for courses on reinforcement learning. I hope that it is useful to various audiences such as anybody interested in AI and/or machine learning, to theoreticians interested in algorithms, theoretic results, and proofs, and to practitioners interested in the inner workings of the algorithms and seeking assurances in the quality of the computations.

I hope that you have as much fun reading the book as I had writing it.

Acknowledgments. I am happy to acknowledge the productive discussion with my students Markus Böck, Florian Chen, Patrizia Daxbacher, Sebastian Eresheim, Helmut Horvath, Lorenz Kapral, Tobias Kietreiber, Julien Malle, Daryna Oliynyk, Daniel Pasterk, Markus Peschl, Tobias Salzer, Jakob aus der Schmitten, Felix Wagner, and Richard Weiss as well as with my colleagues Felix Birkelbach, Rene Hofmann, and Carlotta Sophie von Tubeuf.

Furthermore, I am happy to acknowledge the support of the Austrian Research Promotion Agency (FFG) via the research project RELY (Reliable Reinforcement Learning for Sustainable Energy Systems).

Vienna, June 2025 *Clemens Heitzinger*

Chapter 1

Introduction

Here the basic concept of reinforcement learning and the fundamental notions used in formulating problems in reinforcement learning are introduced. Reinforcement learning is a very general concept and applies to time-dependent learning problems whenever an agent interacts with its environment. The main goal in reinforcement learning is to find optimal policies for the agent to follow. Examples of applications of reinforcement learning are given.

1.1 The Concept of Reinforcement Learning

One of the major appeals of reinforcement learning (RL) is that it applies to all situations where an agent interacts with an environment in a time-dependent manner.

The basic concept is illustrated in Figure 1.1. Firstly, the environment and the agent are possibly randomly initialized at the beginning of each episode. In each new time step t (which is increased in the top right in the figure), the environment and the agent enter the next state S_t and the agent receives the reward R_t. Then the agent chooses the next action A_t according to its possibly random policy. In the next iteration, this action affects the environment and the agent in a possibly random manner; the environment and the agent enter the next state; the agent receives a reward; and so forth.

In this manner, sequences of states S_t, actions A_t, and rewards R_t are generated. These sequences may be infinite (at least theoretically) or they end in a terminal state (which always happens in practice) after a finite number of time steps. A collection of a sequence of states S_t, a sequence of actions A_t, and a sequence of rewards R_t is called an episode. It is convenient to treat both cases, i.e., finite and infinite numbers of time steps, within the same theoretical

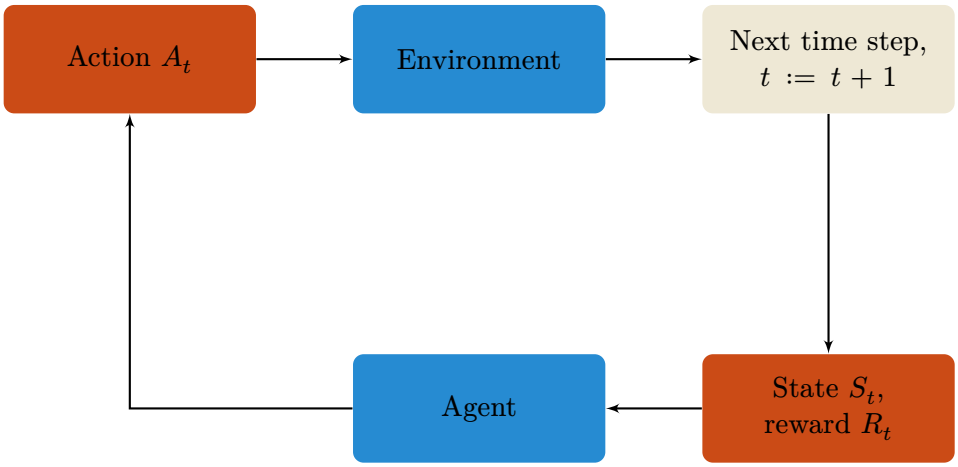


Figure 1.1: Environments, agents, and their interactions.

framework, which can be done under reasonable assumptions.

The return is the expected value of the sum of all (discounted) rewards the agent receives. We call a policy optimal if it maximizes the return.

The main goal in RL is to find optimal policies for the agent to follow. The input to a policy is the current state the agent finds itself in. The output of a deterministic policy is the action to take, and – more generally – the output of a random policy is a probability distribution that assigns probabilities to all actions that are possible in the state. Hence policies may be random, just as environments may be random.

It is necessary to allow random policies. An well-known example where the optimal policy must be random is the game of Rock Paper Scissors (Lizard Spock); if it were not random, it could easily be exploited by the opponent.

Because the main goal in RL is to find optimal policies, the main purpose of this book is to present and to discuss methods and algorithms that calculate such optimal policies.

Because of its generality, the concept of RL encompasses almost all real-world time-dependent problems whose solution requires foresight or intelligence. Therefore, it is at the core of artificial intelligence (AI). Any situation where an agent interacts with an environment and tries to optimize the sum of its rewards is a problem in the realm of RL, irrespective of the cardinality of the sets of spaces and actions and irrespective of whether the environment is deterministic or stochastic.

Depending on the problem, some information may be hidden from the agent. For example, in chess, there is no hidden information; the whole state of the

game is known to both players. On the other hand, in poker, the agents do not have access to all the information and it is hence called a hidden-information game. Therefore, there is observable and unobservable information. In this book, the information that is observable by the agent determines the state, i.e., unobservable information is not included in the definition of the state.

If the discount factor is zero, RL simplifies to supervised learning. In other words, RL is a generalization of supervised learning, which deals with problems that are not time-dependent. There is also a relation between RL and unsupervised learning. In RL, the initially unknown internal structure of the environment is learned and exploited in order to maximize the return. This is done implicitly, in contrast to unsupervised learning, whose goal is to bring hidden structures to light.

1.2 Examples of Applications

Clearly, the more complicated or random the environment is, the more challenging the RL problem is. Thanks to advanced algorithms and – in some cases – to gigantic amounts of data or computational resources or both, it has become possible to solve real-world problems at the level of human intelligence or above. Some applications, mentioned in alphabetical order, are discussed the following.

1.2.1 Autonomous Driving

In autonomous driving, the agent has to traverse a stochastic environment quickly and safely. The agent should also drive smoothly, except in dangerous situations, when it should act decisively. Sophisticated simulation environments including simulated sensor output have been developed, e.g. [1]. For the popular computer game Gran Turismo, agents that drive at a superhuman level have been developed [2]. RL agents can learn to drive on simulated highways and in simulated cities [3, 4], even in changing environments and with failing equipment [5].

1.2.2 Board Games: Backgammon, Chess, Shogi, and Go

Backgammon is a popular two-player board game that is played with counters and dice on tables boards. It is a member of the family of tables games, which date back nearly five thousand years. The use of dice implies that backgammon games include randomness.

In the 1990s, Gerald Tesauro developed the backgammon program TD-Gammon, which used temporal-difference learning and artificial neural networks

[6]. It achieved a level of play only slightly below that of the best human backgammon players at the time; in 1998, it lost a 100-game competition against the world champion by only eight points [7]. TD-Gammon also had strong impact on the backgammon community, since professional players soon adopted its evaluation of certain opening strategies.

In 1996, then world chess champion Garry Kasparov won a six-game match against IBM's Deep Blue chess program by 4:2. In the next year, an updated version of Deep Blue defeated Garry Kasparov by $3\frac{1}{2} : 2\frac{1}{2}$. In 2002, a match between then world chess champion Vladimir Kramnik and the chess program Deep Fritz ended in a 4:4 draw. Ever since that time, chess has been solved in the sense that the best chess programs play better than the best humans.

It is important to note that these chess programs were not self-learning, but they were based on tree search and fixed rules to assess the values of positions. Since self-learning is a defining characteristic of algorithms in machine learning and AI, these programs were not artificial intelligences.

After chess, Go was the only remaining classical board game which humans could play better than computers. Go games have a much larger search spaces than chess games, which is the reason why Go was the last unsolved board game and remained a formidable challenge. It was commonly believed at the time that it would take decades till Go can be solved.

In [8], an RL algorithm called AlphaGo Zero learned to beat the best humans consistently using no prior knowledge apart from the rules of the game, i.e., *tabula rasa,* albeit using vast computational resources. AlphaGo Zero is a truly self-learning algorithm.

In the next step, in [9], a more general RL algorithm called AlphaZero learned to play the three games of chess, shogi (Japanese chess), and Go again *tabula rasa*, but now using only self-play. It defeated world-champion programs in each of the three games.

1.2.3 Card Games: Poker and Schnapsen

Card games differ from board games regarding the amount of information that is available to the players. In board games, the players know the full state of the game at all times, whereas in card games some information is available to all players, some information only to certain players, and some information is known by no player. Such games are called hidden-information games.

In 2019, Brown and Sandholm reported on their poker program program, dubbed Pluribus, that plays six-player no-limit Texas hold'em [10]. Pluribus learned by self-play, playing against five copies of itself. In order to evaluate the performance of Pluribus, it had to compete with five elite professional poker

players or with five copies of itself playing against one professional. Over the course of 10 000 hands of poker, it was found that it performs significantly better than humans [10].

Schnapsen is trick-taking card game of the ace-ten family for two players. It is the national card game of Austria and Hungary, and it is very popular in Bavaria and Upper Silesia as well. Schnapsen is a both a point-trick and trick-and-draw game and involves lots of strategy. Schnapsen is played in many tournaments, and there are also variants for three and four players (Dreierschnapsen and Bauernschnapsen, respectively). In contrast to poker, it is not considered a game of luck, but a game of skill, according to Austrian law. Schnapsen can be traced back to the 1700s, and the earliest description of its predecessor Mariage dates back to 1715.

The aim of the game is to take tricks in order to collect 66 or more card points as quickly as possible in rounds. In each round, a player can win one, two, or three game points. The player who wins seven game points first wins the game.

In his master's thesis, Tobias Salzer developed a deep RL algorithm for Schnapsen, which beat the previously strongest Schnapsen program, playing above human expert level, in a tournament of twenty games by 12:8 [11].

1.2.4 Computer Games

The study of computer games in the context of AI dates back at least to Falken's Maze [12].

In the context of RL playing computer games, the game is the environment and the agent has to learn an optimal strategy. One of the great successes in the history of RL and a precursor to the single algorithm that solved chess, Go, and shogi (see Section 1.2.2) was a single deep RL algorithm that can learn to play many (but not all) Atari 2600 games at the human level or above [13].

A few years later, an RL algorithm learned to play Quake III Arena in a mode called "Capture the Flag" at the human level [14]. Also in 2019, a multi-agent RL algorithm learned to play StarCraft II, a contemporary computer game, at the grandmaster level. It did not achieve clearly superhuman capabilities, which may be due to the hidden information in this computer game.

1.2.5 Finance

Oftentimes in finance, the environment is a certain market, the state is given by the positions in the portfolio, and the profits become positive rewards, while a

risk measure may become a negative reward. In this manner, wealth is maximized, while the risk is managed.

1.2.6 Medicine

The main goal in medicine is to treat patients so that they recover as fully and as quickly as possible. In the nomenclature of RL, the patient is the environment, the agent is the medical doctor, the state is given by any measurements of the patient's condition, and the actions are the therapies or medications. In this manner, treating patient becomes an RL problem. Questions such as how to find optimal therapies, how to personalize therapies, and how to evaluate therapies thus become amenable to quantification and can be solved by RL algorithms. This approach to medicine paves the way towards truly personalized and predictive medicine.

Intensive care is particularly well-suited for RL, because many values are recorded over relatively short periods of time in intensive-care units. Typical episodes last a few days, and hundreds of values may be recorded every few hours.

The most prevalent condition and at the same time the most prevalent cause of death in intensive-care units is sepsis. Therefore sepsis is an obvious application of RL in medicine. After clustering of the patients' conditions, RL algorithms for a finite number of states can be utilized. When the actions correspond to two medications at different doses, it was found that RL policies reach or slightly surpass the performance of human doctors [15, 16, 17].

1.2.7 Optimal Control and Robotics

Reinforcement learning can be viewed as optimal control of stochastic environments or systems. It does not matter whether there is a model of the environment or not. If there is model, it is called a model-based control problem, also often called a planning problem; if there is no model, it is called a model-free control problem, also often called a learning problem.

There is a vast number of applications of RL in automation, optimal control, and robotics. Examples for the optimal control of industrial systems are chemical and biological reactors. The output of the product is to be maximized, while the reactor must be kept within safe operating conditions. In robotics, the robot interacts with its environment in order to solve the task it has been assigned by specifying the rewards and/or penalties. Ideally, optimal policies are then learned without any further help or interaction. Generally speaking, RL can used to automate any kind of equipment or machinery, a few examples being

[18, 19].

1.2.8 Recommendation Systems

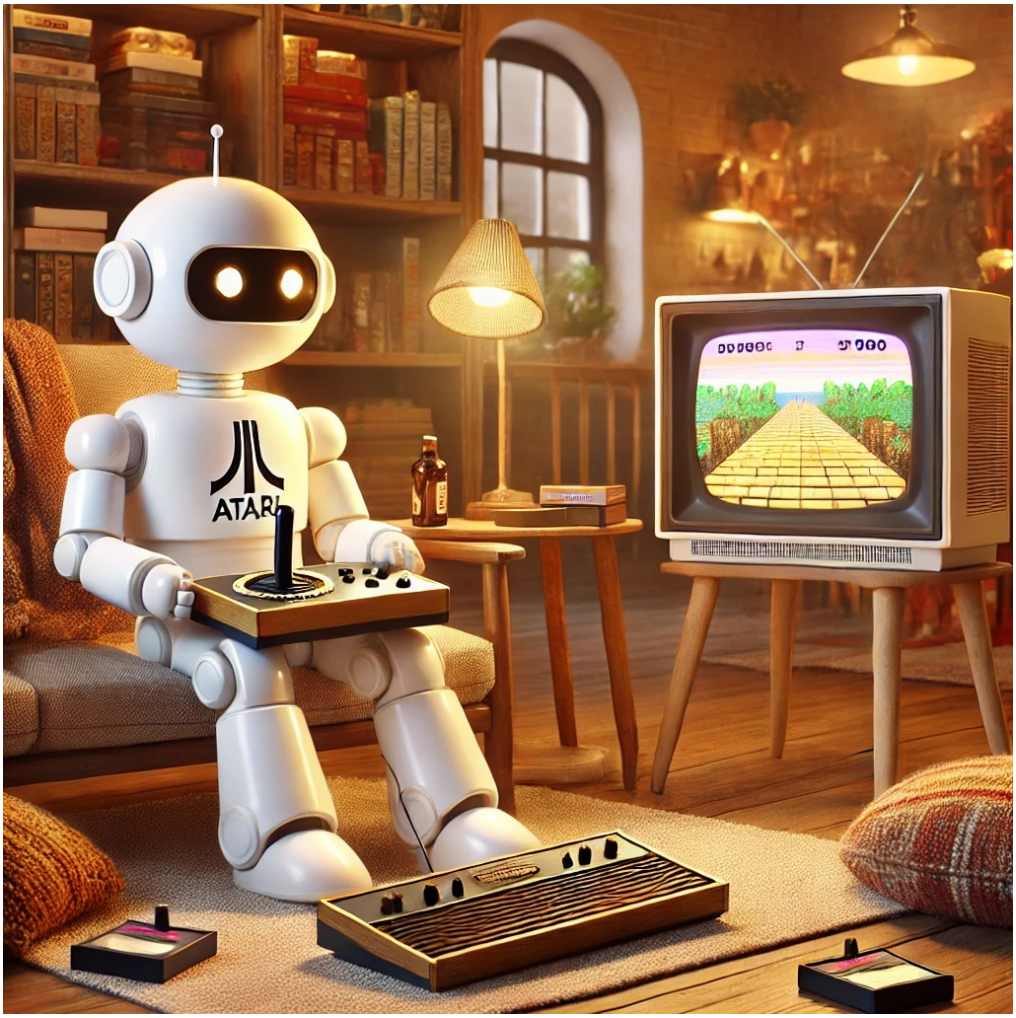
Although recommendation problems have traditionally often been considered to be classification or prediction problems, they are nowadays often formulated as sequential decision problems, which can better reflect the multiple interactions between users and the recommendation system. RL is used to directly optimize for user satisfaction or other metrics in various areas such as music and video playlists [20, 21, 22]. It is known that traffic is routed through RL systems on many large websites.

1.2.9 Supply Chains and Inventories

In supply-chain and inventory management, goods must be moved efficiently such that they arrive at their points of destination in sufficient, but not abundant quantity. Rewards are obtained whenever the goods arrive on time in the specified quantity at their points of destination, while penalties are obtained when too few or too many goods arrive or when they are delayed.

1.3 Bibliographical Remarks

Here textbooks on RL are mentioned. The most important and influential introductory textbook is [23]. RL is closely related to optimal control and often builds on dynamic programming; textbooks at the intersections of these subjects are [24, 25, 26]. [27]. A textbook on distributional RL is [28]. A comprehensive collection of approaches to dynamic programming, RL, and stochastic optimization can be found in [29]. Practical approaches to deep RL are discussed in [30, 31, 32, 33].


ATARI

Chapter 2

Markov Decision Processes and Dynamic Programming

> I was intrigued by dynamic programming. It was clear to me that there was a good deal of good analysis there. Furthermore, I could see many applications. It was a clear choice. I could either be a traditional intellectual, or a modern intellectual using the results of my research for the problems of contemporary society. This was a dangerous path. Either I could do too much research and too little application, or too little research and too much application. I had confidence that I could do this delicate activity, pie à la mode.
>
> Richard Bellman [34, p. 173].

This chapter starts with one of the simplest learning problems, the so-called multi-armed bandits. Multi-armed bandits serve to introduce basic notions and challenges in time-dependent learning. Then Markov decision processes are defined, as they serve as the foundation for the description of reinforcement-learning problems. Once the basic language to describe reinforcement-learning problems has been defined, a collection of standard environments is presented. Based on the definitions, the most important notions in and results of dynamic programming are presented, since they serve as a foundation that inspires learning algorithms.

2.1 Multi-Armed Bandits

A relatively simple, but illustrative example of a reinforcement-learning problem are multi-armed bandits. The name of the problem stems from slot machines.

There are $k = |\mathcal{A}|$ slot machines or bandits, and the action is to choose one machine and play it to receive a reward. The reward each slot machine or bandit distributes is taken from a stationary probability distribution, which is of course unknown to the agent and different for each machine.

In more abstract terms, the problem is to learn the best policy when being repeatedly faced with a choice among k different actions. After each action, the reward is chosen from the stationary probability distribution associated with each action. The objective of the agent is to maximize the expected total reward over some time period or for all times.

In time step t, the action is denoted by $A_t \in \mathcal{A}$ and the reward by R_t. In this example, we define the (true) value of an action a to be

$$q_* \colon \mathcal{A} \to \mathbb{R}, \qquad q_*(a) := \mathbb{E}[R_t \mid A_t = a], \qquad a \in \mathcal{A}.$$

(This definition is simpler than the general one, since the time steps are independent from one another. There are no states.) This function is called the action-value function.

Since the true value of the actions is unknown (at least in the beginning), we calculate an approximation called $Q_t \colon \mathcal{A} \to \mathbb{R}$ in each time step; it should be as close to q_* as possible. A reasonable approximation is the sample mean of the observed rewards, i.e.,

$$Q_t(a) := \frac{\text{sum of rewards obtained when action } a \text{ was taken prior to } t}{\text{number of times action } a \text{ was taken prior to } t}.$$

Based on this approximation, the greedy way to select an action is to choose

$$A_t :\in \arg\max_a Q_t(a),$$

which serves as a substitute for the ideal choice

$$\arg\max_a q_*(a).$$

Note that $\arg\max_a Q_t(a)$ is a set, since multiple actions may maximize the argument. Hence the notation "$x :\in X$" means that an element of X is chosen randomly with each element having the same probability and assigned to x, analogously to the notation ":=".

In summary, these simple considerations have led us to a first example of an action-value method. In general, an action-value method is a method which is based on (an approximation Q_t of) the action-value function q_*.

Choosing the actions in a greedy manner is called exploitation. However, there is a problem. In each time step, we only have the approximation Q_t at

our disposal. For some of the k bandits or actions, it may be a bad approximation, in the sense that it leads us to choose an action a whose estimated value $Q_t(a)$ is higher than its true value $q_*(a)$. Such an approximation error would be misleading and reduce or rewards.

In other words, exploitation by greedy actions is not enough. We also have to ensure that our approximations are sufficiently accurate; this process is called exploration. If we explore all actions sufficiently well, bad actions cannot hide behind high rewards obtained by chance.

The duality between exploitation and exploration is fundamental to reinforcement learning, and it is worthwhile to always keep these two concepts in mind. Here we saw how these two concepts are linked to the quality of the approximation of the action-value function q_*.

In general, a greedy policy always exploits the current knowledge (in the form of an approximation of the action-value function) in order to maximize the immediate reward, but it spends no time on the long-term picture. A greedy policy does not sample apparently worse actions to see whether their true action values are better or whether they lead to more desirable states. (Note that the multi-bandit problem is stateless.)

A common and simple way to combine exploitation and exploration into one policy in the case of finite action sets is to choose the greedy action most of the time, but any action with a (usually small) probability ϵ. This is the subject of the following definition.

Definition 2.1 (ϵ-greedy policy)**.** Suppose that the action set $\mathcal{A}$ is finite, that Q_t is an approximation of the action-value function, and that $\epsilon_t \in [0, 1]$. Then the policy defined by

$$A_t := \begin{cases} \arg\max_{a \in \mathcal{A}} Q_t(a) & \text{with probability } 1 - \epsilon_t \text{ breaking ties randomly,} \\ \text{a random action } a & \text{with probability } \epsilon_t \end{cases}$$

is called the ϵ-greedy policy.

In the first case, it is important to break ties randomly, because otherwise a bias towards certain actions would be introduced. The random action in the second case is usually chosen uniformly from all actions $\mathcal{A}$.

Learning methods that use ϵ-greedy policies are called ϵ-greedy methods. Intuitively speaking, every action will be sampled an infinite number of times as $t \to \infty$, which ensures convergence of Q_t to q_*.

Regarding the numerical implementation, it is clear that storing all previous actions and rewards becomes inefficient as the number of time steps increases. Can memory and the computational effort per time step be kept constant, which

would be the ideal case? Yes, it is possible to achieve this ideal case using a trick, which will lead to our first learning algorithm.

To simplify notation, we focus on the action-value function of a certain action. We denote the reward received after the k-th selection of this specific action by R_k and use the approximation

$$Q_n := \frac{1}{n-1} \sum_{k=1}^{n-1} R_k$$

of its action value after the action has been chosen $n-1$ times. This is called the sample-average method. The trick is to find a recursive formula for Q_n by calculating

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{k=1}^{n} R_k \\ &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{k=1}^{n-1} R_k \right) \\ &= \frac{1}{n} (R_n + (n-1) Q_n) \\ &= Q_n + \frac{1}{n} (R_n - Q_n) \qquad \forall n \geq 1. \end{aligned}$$

(If $n = 1$, this formula holds for arbitrary values of Q_1 so that the starting point Q_1 does not play a role.)

This yields our first learning algorithm, Algorithm 1. The implementation of this recursion requires only constant memory for n and Q_n and a constant amount of computation in each time step.

The recursion above has the form

$$\text{new estimate} := \text{old estimate} + \text{learning rate} \cdot (\text{target} - \text{old estimate}), \quad (2.1)$$

which is a common theme in reinforcement learning. Its intuitive meaning is that the estimate is updated towards a target value. Since the environment is stochastic, the learning rate only moves the estimate towards the observed target value. Updates of this form will occur many times in this book.

In the most general case, the learning rate α depends on the time step and the action taken, i.e., $\alpha = \alpha_t(a)$. In the sample-average method above, the learning rate is $\alpha_n(a) = 1/n$. It can be shown that the sample-average approximation Q_n above converges to the true action-value function q_* by using the law of large numbers.

Algorithm 1 a simple algorithm for the multi-bandit problem.

Initialization:
choose $\epsilon \in (0, 1)$,
initialize two vectors $\mathbf{q}$ and $\mathbf{n}$ of length $|\mathcal{A}|$ with zeros.

loop
  select an action a ϵ-greedily using $\mathbf{q}$ (see Definition 2.1)
  perform the action a and receive the reward r from the environment
  $n_a := n_a + 1$
  $q_a := q_a + \frac{1}{n_a}(r - q_a)$
end loop

return q

Sufficient conditions that yield convergence with probability one are

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty, \tag{2.2a}$$

$$\sum_{k=1}^{\infty} \alpha_k(a)^2 < \infty. \tag{2.2b}$$

They are well-known in stochastic-approximation theory and are a recurring theme in convergence proofs. The first condition ensures that the steps are sufficiently large to eventually overcome any initial conditions or random fluctuations. The second condition means the steps eventually become sufficiently small. Of course, these two conditions are satisfied for the learning rate

$$\alpha_k(a) := \frac{1}{k},$$

but they are not satisfied for a constant learning rate $\alpha_k(a) := \alpha$. However, a constant learning rate may be desirable when the environment is time-dependent, since then the updates continue to adjust the policy to changes in the environment.

Finally, we shortly discuss an important improvement over ϵ-greedy policies, namely action selection by upper confidence bounds. The disadvantage of an ϵ-greedy policy is that it chooses the non-greedy actions without any further consideration. It is however better to select the non-greedy actions according to their potential to actually being an optimal action and according to the

uncertainty in our estimate of the value function. This can be done using the so-called upper-confidence-bound action selection

$$A_t :\in \arg\max_{a \in \mathcal{A}} \left(Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right),$$

where $N_t(a)$ is the number of times that action a has been selected before time t. If $N_t(a) = 0$, the action a is treated as an optimal action. The constant c controls the amount of exploration.

The term $\sqrt{(\ln t)/N_t(a)}$ measures the uncertainty in the estimate $Q_t(a)$. When the action a is selected, $N_t(a)$ increases and the uncertainty decreases. On the other hand, if an action other than a is chosen, t increases and the uncertainty relative to other actions increases. Therefore, since $Q_t(a)$ is the approximation of the value and $c\sqrt{(\ln t)/N_t(a)}$ is the uncertainty, where c is the confidence level, the sum of these two terms acts as an upper bound for the true value $q_*(a)$.

Since the logarithm is unbounded, all actions are ensured to be chosen eventually. Actions with lower value estimates $Q_t(a)$ and actions that have often been chosen (large $N_t(a)$ and low uncertainty) are selected with lower frequency, just as they should in order to balance exploitation and exploration.

2.2 Markov Decision Processes

The mathematical language and notation for describing and solving reinforcement-learning problems is deeply rooted in Markov decision processes. Having discussed multi-armed bandits as a concrete example for a reinforcement-learning problem, we now generalize some notions and fix the notation for the rest of the book using the language of Markov decision processes.

As already discussed in Chapter 1, the whole world is divided into an agent and an environment. The agent interacts with the environment iteratively. The agent takes actions in the environment, which changes the state of the environment and for which it receives a reward (see Figure 1.1). The task of the agent is to learn optimal policies, i.e., to find the best action in order to maximize all future rewards it will receive. We will now formalize the problem of finding optimal policies.

We note the sequence of (usually discrete) time steps by $t \in \{0, 1, 2, \dots\}$. The state (or observation) that the agent receives in time step t from the environment is denoted by $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all states. We use capital letters to denote random variables. In general, $\mathcal{S} \subset \mathbb{R}^{d_s}$, $d_s \in \mathbb{N}$, but if there is a finite number of states, $\mathcal{S} \subset \mathbb{Z} \subset \mathbb{R}$ suffices.

The action performed by the agent in time step t is denoted by $A_t \in \mathcal{A}(s)$ if the environment is in state $s = S_t$. In general, $\mathcal{A} \subset \mathbb{R}^{d_a}$, $d_a \in \mathbb{N}$, but if there is a finite number of actions, $\mathcal{A}(s) \subset \mathbb{Z} \subset \mathbb{R}$ suffices. In general, the set $\mathcal{A}(s)$ of all available actions depends on the very state s, although this dependence is sometimes dropped to simplify notation. In the subsequent time step, the agent receives the reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and finds itself in the next state S_{t+1}. Then the iteration is repeated.

The whole information about these interactions between the agent and the environment can be recorded in sequences $\langle S_t \rangle_{t\in\mathbb{N}}$, $\langle A_t \rangle_{t\in\mathbb{N}}$, and $\langle R_t \rangle_{t\in\mathbb{N}}$ or in the sequence

$$\langle S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \dots \rangle$$

These (finite or infinite) sequences are called episodes.

The random variables S_t and R_t provided by the environment depend only on the preceding state and action. This is the Markov property, and the whole process is a Markov decision process (MDP). We assume that the Markov property is always satisfied.

In a finite MDP, all three sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ are finite, and hence the random variables S_t, A_t, and R_t are discrete.

The purpose of the following definitions is to describe the dynamics of the environment, i.e., the random variables S_{t+1} and R_{t+1} knowing $S_t = s$ and $A_t = a$, starting from the general case of states and rewards that are real vectors and ending with finite MDP.

The cumulative distribution function F_X of a random variable X is defined as

$$F_X(x) := \mathbb{P}[X \leq x]$$

in the univariate case and as

$$F_{X_1,\dots,X_d}(x_1, \dots, x_d) := \mathbb{P}[X_1 \leq x_1 \wedge \dots \wedge X_d \leq x_d]$$

in the multivariate case, where $d \in \mathbb{N}$. For vectors $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$, we define

$$\mathbf{x} \leq \mathbf{y} \quad :\Longleftrightarrow \quad x_k \leq y_k \quad \forall k \in \{1, \dots, d\}.$$

Then we can write

$$F_X(\mathbf{x}) = \mathbb{P}[X \leq \mathbf{x}].$$

In general, the expectation of a function h of a random variable X is defined as

$$\mathbb{E}[h(x)] := \int_{\mathbf{x}\in\mathbb{R}^d} h(\mathbf{x}) \mathrm{d}F_X(\mathbf{x}) = \int_{x_1=-\infty}^{\infty} \cdots \int_{x_d=-\infty}^{\infty} h(\mathbf{x}) \mathrm{d}F_{X_d}(x_d) \cdots \mathrm{d}F_{X_1}(x_1).$$

The probability of being put into state $s' \in \mathcal{S}$ and receiving a reward $r \in \mathcal{R}$ after starting from a state $s \in \mathcal{S}$ and performing action $a \in \mathcal{A}(s)$ is recorded by the transition probability

$$p\colon \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1],$$
$$p(s', r \mid s, a) := \mathbb{P}[S_t = s',\ R_t = r \mid S_{t-1} = s,\ A_{t-1} = a].$$

Despite the notation with the vertical bar in the argument list of p, which is reminiscent of a conditional probability, the function p is a deterministic function of four arguments.

The function p records the dynamics of the MDP, and it is therefore also called the dynamics function of the MDP. Since it is a probability distribution, the equality

$$\forall s \in \mathcal{S}\colon \quad \forall a \in \mathcal{A}(s)\colon \qquad \int_{\mathcal{S}} \int_{\mathcal{R}} p(s', r \mid s, a) \mathrm{d}s' \mathrm{d}r = 1$$

holds.

If there is a finite number of states and rewards, the integrals become sums and the equalities are

$$\forall s \in \mathcal{S}\colon \quad \forall a \in \mathcal{A}(s)\colon \qquad \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1.$$

The requirement that the Markov property holds is met by ensuring that the information recorded in the states $s \in \mathcal{S}$ is sufficient. This is an important point when translating an informal problem description into the framework of MDPs and reinforcement learning. In practice, this often means that the states become sufficiently long vectors that contain enough information about the past ensuring that the Markov property holds. This in turn has the disadvantage that the dimension of the state space may have to increase to ensure the Markov property.

It is important to note that the transition probability p, i.e., the dynamics of the MDP, is *unknown*. It is sometimes said that the term *learning* refers to problems where information about the dynamics of the system is absent. Learning algorithms face the task of calculating optimal strategies with only very little knowledge about the environment, i.e., just the sets $\mathcal{S}$ and $\mathcal{A}(s)$.

The dynamics function contains all relevant information about the MDP, and therefore other quantities can be derived from it. The first quantity are the state-transition probabilities

$$p\colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1],$$

$$p(s' \mid s, a) := \mathbb{P}\{S_t = s' \mid S_{t-1} = s,\ A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a),$$

also denoted by p, but taking only three arguments.

Next, the expected rewards for state-action pairs are

$$r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R},$$
$$r(s, a) := \mathbb{E}[R_t \mid S_{t-1} = s,\ A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a).$$

(Note that $\sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) = 1$ must hold.) Furthermore, the expected rewards for state–action–next-state triples are given by

$$r \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}, \tag{2.3a}$$
$$r(s, a, s') := \mathbb{E}[R_t \mid S_{t-1} = s,\ A_{t-1} = a,\ S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}. \tag{2.3b}$$

(Note that $\sum_{r \in \mathcal{R}} p(s', r \mid s, a)/p(s' \mid s, a) = 1$ must hold.)

MDPs can be visualized as directed graphs. The nodes are the states, and the edges starting from a state s correspond to the actions $\mathcal{A}(s)$. The edges starting from state s may split and end in multiple target nodes s'. The edges are labeled with the state-transition probabilities $p(s' \mid s, a)$ and the expected reward $r(s, a, s')$.

Further important data structures are the transition and the experience. A transition is a tuple

$$(s, a, s', r, \text{done?}),$$

which records all information in a time step. Therefore an episode can be viewed as a sequence of transitions. The binary variable done? records whether a terminal state has been reached and the episode has ended; this information is useful in certain learning algorithms. An experience (also often called an experience-replay buffer) is a set of transitions.

2.3 Rewards, Returns, and Episodes

We start with a remark on how rewards should be defined in practice when translating an informal problem description into a precisely defined environment. It is important to realize that the learning algorithms will learn to maximize the expected value of the discounted sum of all future rewards (defined below), nothing more and nothing less.

For example, if the agent should learn to escape a maze quickly, it is expedient to set $R_t := -1$ for all times t. This ensures that the task is completed quickly.

The obvious alternative to define $R_t := 0$ before escaping the maze and a positive reward when escaping fails to convey to the agent that the maze should be escaped quickly; there is no penalty for lingering in the maze.

Furthermore, the temptation to give out rewards for solving subproblems must be resisted. For example, when the goal is to play chess, there should be no rewards to taking opponents' pieces. Because otherwise the agent would become proficient in taking opponents' pieces, but not in checkmating the king.

From now on, we make the following assumption, which is needed for defining the return in the general case, which is the next concept we discuss.

Assumption 2.2 (bounded rewards)**.** The reward sequence $\langle R_t \rangle_{t \in \mathbb{N}}$ is bounded.

There are two cases to be discerned, namely whether the episodes are finite or infinite. We denote the time of termination, i.e., the time when the terminal state of an episode is reached, by T. The case of a finite episode is called episodic, and $T < \infty$ holds; the case of an infinite episode is called continuing, and $T = \infty$ holds.

Definition 2.3 (discounted return)**.** The *discounted return* is

$$G_t := \sum_{k=t+1}^{T} \gamma^{k-(t+1)} R_k = R_{t+1} + \gamma R_{t+2} + \cdots,$$

where $T \in \mathbb{N} \cup \{\infty\}$ is the terminal state of the episode and $\gamma \in [0, 1]$ is the discount rate.

From now on, we also make the following assumption.

Assumption 2.4 (finite returns)**.** $T = \infty$ and $\gamma = 1$ do not hold at the same time.

Assumptions 2.2 and 2.4 ensure that all returns are finite.

There is an important recursive formula for calculating the returns from the rewards of an episode. It is found by calculating

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-(t+1)} R_k \tag{2.4a}$$

$$= R_{t+1} + \gamma \sum_{k=t+2}^{T} \gamma^{k-(t+2)} R_k \tag{2.4b}$$

$$= R_{t+1} + \gamma G_{t+1}. \tag{2.4c}$$

The calculation also works when $T < \infty$, since $G_T = 0$, $G_{T+1} = 0$, and so forth since then the sum in the definition of G_t is empty and hence equal to zero. This formula is very useful to quickly compute returns from reward sequences.

At this point, we can formalize what (classical) problems in reinforcement learning are.

Definition 2.5 (reinforcement-learning problem)**.** Given the states $\mathcal{S}$, the actions $\mathcal{A}(s)$, and the opaque transition function $\mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{R}$ of the environment, a reinforcement-learning problem consists of finding policies for selecting the actions of an agent such that the expected discounted return is maximized.

The random transition provided by the MDP of the environment, namely going from a state-action pair to a state-reward pair, being opaque means that we consider it a black box. For any state-action pair as input, it is only required to yield a state-reward pair as output. In particular, the transition probabilities are considered to be unknown. Examples of such opaque environments are

- functions $\mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{R}$ defined in a programming language,
- more complex pieces of software such as computer games or simulation software,
- historic data from which states, actions, and rewards can be observed.

The fact that the problem class in Definition 2.5 is so large adds to the appeal of reinforcement learning.

2.4 Standard Environments

At this point, we have defined the basic concepts of RL that are necessary to translate applications into the language of RL. In this section, various standard environments are defined. They serve several purposes. First, they serve as leading examples of RL problems in different application areas. Several environments in this collection have a long history in RL. Second, most of these environments have well-known solutions, and thus they easily serve as test cases to verify implementations of algorithms. Third, these environments embody a wide variety of problems useful for the evaluation of algorithms. The challenge in developing learning algorithms is that they should be general; if a learning algorithm performs well on a wide variety of problems, a useful algorithm has been found.

The state and action spaces can be discrete or continuous. This is indicated in the titles below in this order; for example, "continuous/discrete" indicates a continuous state space and a discrete action space.

A popular type of environment is the grid world. A grid world is an environment that typically consists of a (two-dimensional) grid, i.e., the states, on which the agent moves. There are usually four actions: up, down, left, and right; sometimes there are eight, including the diagonal directions. If the agent is near the edge of the grid and the action would make it leave the grid, the state remains unchanged; in other words, the agents cannot leave the grid. Often the grid world is a maze, i.e., there is a start state and a goal or a terminal state. In the case of a maze, the goal state must be reached as quickly as possible. Therefore there is a reward of -1 in each step until the goal state is reached, and the learning task is episodic and undiscounted ($\gamma = 1$). At first glance, it would also make sense to give out zero rewards during the episode and a positive one when the goal is reached; however, such rewards provide no incentive to reach the goal state as quickly as possible.

Usually there are complications built into the environment, or additional tasks defined via rewards must be performed. In this manner, grid worlds become a versatile class of environments and are well-suited to elucidate certain behaviors of algorithms.

2.4.1 Simple Grid World (Discrete/Discrete)

Implement a 4×4 grid world with two terminal states in the upper left and lower right corners, resulting in 14 non-terminal states (see [23, Example 4.1]. The four actions $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$ act deterministically, the discount factor is $\gamma = 1$, and the reward is always equal to -1. Ensure that a maximum number of time steps can be specified.

2.4.2 Windy Grid World (Discrete/Discrete)

Windy Grid World is [23, Example 6.5]. The underlying grid has size 10×7, see Figure 2.1. The start is in the middle of the first column, and the goal is in the middle of the eighth column. The four actions are up, down, left, and right, and the learning task is an undiscounted ($\gamma = 1$) episodic task. As is usual in mazes, there is a reward of -1 in each time step until the goal or terminal state is reached.

This grid world is called windy because of its complication of a wind that sometimes displaces the agent (deterministically). In the middle region of the grid, the next states are shifted upward by the wind by one or two cells as

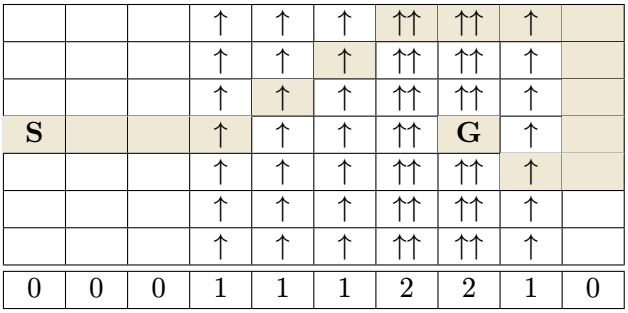


Figure 2.1: Windy Grid World. The colored cells indicate an optimal policy, resulting in an episode of length 15.

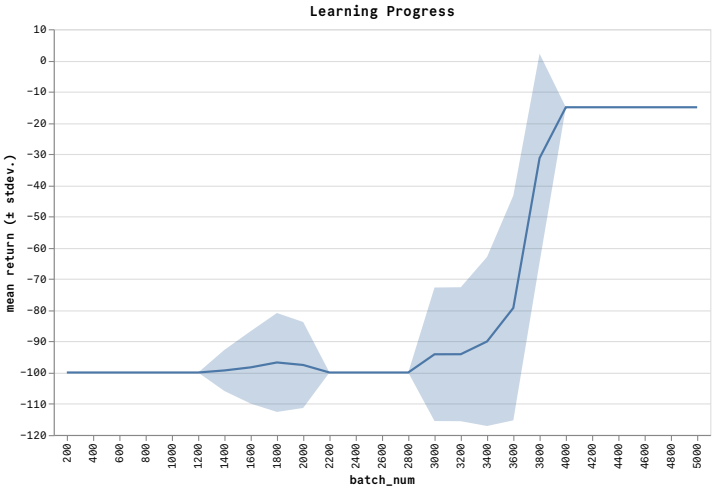


Figure 2.2: Learning progress of Q-learning for Windy Grid World.

indicated in the bottom row in the figure. The number below each column indicates the number of cells shifted upward. For example, if the agent is located to the right of the goal and the action is going left, then the agent is taken to the cell just above the goal. Note that the agent never leaves the grid world (as usual in grid worlds), even if it is displaced by the wind.

Learning progress is shown in Figure 2.2.

2.4.3 Cliff Walking (Discrete/Discrete)

Cliff Walking is another grid world [23, Example 6.6]. The underlying grid has size 12×4, see Figure 2.3. The start is in the bottom left, and the goal is in

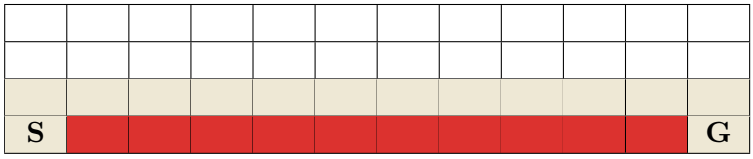


Figure 2.3: Cliff Walking. The cliff is located at the bottom between the start and goal states. The colored cells indicate an optimal policy, resulting in an episode of length 13.

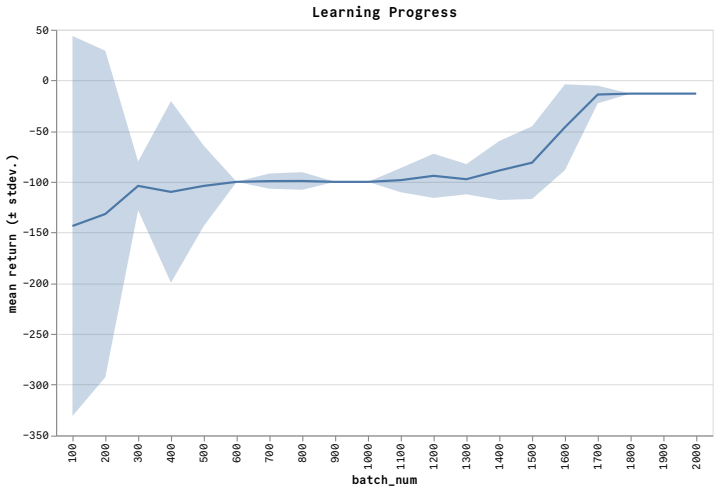


Figure 2.4: Learning progress of Q-learning for Cliff Walking.

the bottom right. Between the start and goal there is a cliff, representing the complication of this environment. The four actions are up, down, left, and right, and the learning task is an undiscounted ($\gamma = 1$) episodic task. The reward is -1 as usual in mazes, but falling down the cliff incurs a reward of -100 and instantly teleports the agent back to the start state.

Learning progress is shown in Figure 2.4.

2.4.4 Frozen Lake (Discrete/Discrete)

Frozen Lake[1] is a stochastic grid world. There are two versions regarding the size of the grid, namely 4×4 (see Figure 2.5) and 8×8 (see Figure 2.6). The start state is in the top left, and the goal state (a terminal state) is in the bottom right. The four actions are up, down, left, and right, and the learning task is an

[1] https://www.gymlibrary.dev/environments/toy_text/frozen_lake/

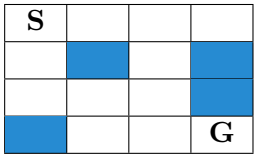


Figure 2.5: Frozen Lake 4×4. The colored cells are holes in the ice.

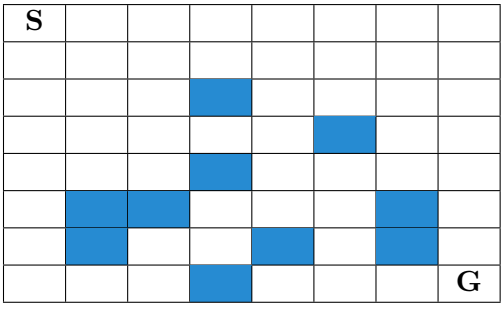


Figure 2.6: Frozen Lake 8×8. The colored cells are holes in the ice.

undiscounted ($\gamma = 1$) episodic task. There is a reward of $+1$ when reaching the goal, otherwise the rewards are zero. The complications are the holes in the ice, which are terminal states.

Another complication, which can be used optionally, is slippery ice. This is a complication that introduces lots of randomness. In the slippery version, the intended direction is followed with probability $1/3$ only. Otherwise, the agent will move in either direction perpendicular to the intended direction with equal probability of $1/3$ in both directions. For example, if the action is up, then the actions up, left, and right all have probability $1/3$.

Learning progresses are shown in Figure 2.7 and Figure 2.8.

2.4.5 Rock Paper Scissors (Discrete/Discrete)

Rock paper scissors is a well-known simultaneous, zero-sum game for two players. In the first variant considered here, the opponent uses a fixed, stochastic policy. Each action (rock, paper, or scissors) is chosen with equal probability $1/3$. As usual, the task of the agent is to learn an optimal policy.

In the second variant, the opponent uses a stochastic policy that varies with time. At time $t \in \mathbb{N}$, the probabilities are

$$\mathbb{P}[\text{rock}] = \sin^2 \alpha t \cos^2 \beta t,$$
$$\mathbb{P}[\text{paper}] = \sin^2 \alpha t \sin^2 \beta t,$$

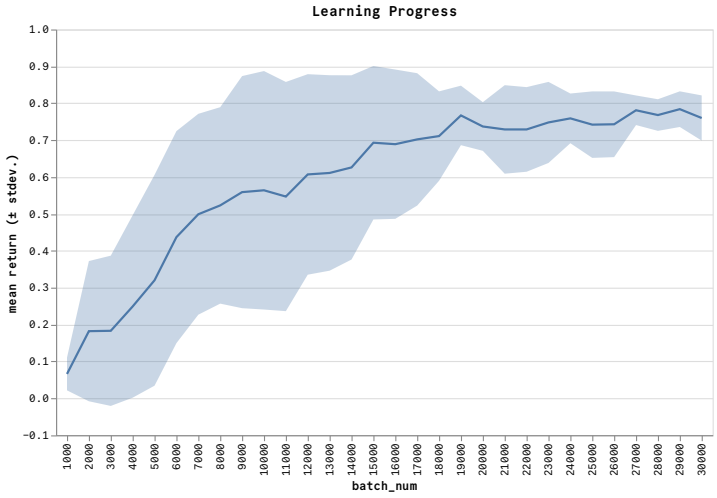


Figure 2.7: Learning progress of Q-learning for Frozen Lake (size 4×4, slippery).

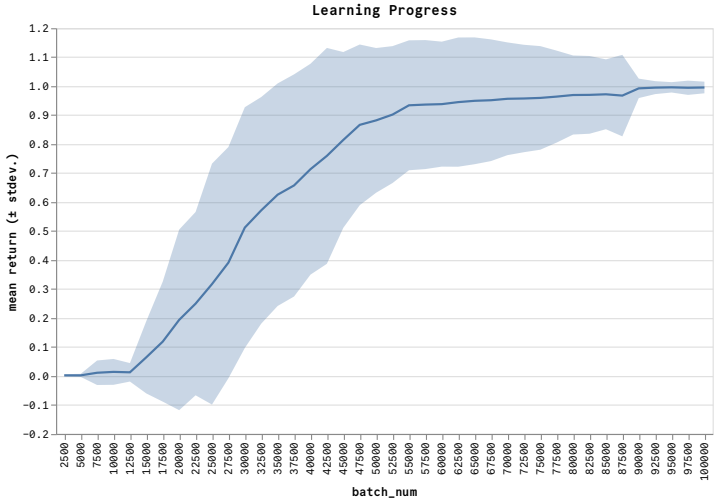


Figure 2.8: Learning progress of Q-learning for Frozen Lake (size 8×8, slippery).

$$\mathbb{P}[\text{scissors}] = \cos^2 \alpha t,$$

where $\alpha := 2\pi/100$ and $\beta := 2\alpha$. Again, the task of the agent is to learn an optimal policy.

2.4.6 Prisoner's Dilemma (Discrete/Discrete)

2.4.7 Blackjack (Discrete/Discrete)

Blackjack is the most widely played casino banking game. It is a descendent of the family of casino banking games known as "twenty-one," whose progenitor is recorded in Spain in the early 17th century. In the context of card games, a banking game is a gambling card game in which one or more players, the so-called punters, play against a banker or dealer who controls the game.

The objective of Blackjack is to draw cards such that the sum of their values is as great as possible without exceeding 21. An ace has a value of 1 or 11, whichever is more expedient to the player. All face cards (King, Queen, and Jack) have a value of 10. [23, Example 5.1] is the basic version of the game in which each player plays independently against the dealer and has only two choices, namely to "hit" or to "stick." In other versions, the player can also "double down," "split," or "surrender."

All cards are dealt from an infinite deck of cards, which means that there is no advantage in counting cards. At the start of a each game, two cards are dealt to both the player and the dealer. Both cards of the player are only known to the player, but one card of the dealer is dealt face up and the other face down.

If a player or the dealer has 21 immediately, it is called a "natural." If the player has a natural immediately, the game is a draw if the dealer also has a natural, or the player wins if the dealer does not have a natural. Later, if the player does not have a natural immediately, the player has two choices in each time step: he can request an additional card ("hits") or he can stop requesting additional cards ("sticks"). The episode continues until he sticks or exceeds 21 ("goes bust"). If he goes bust, he loses the game. Otherwise, if the sticks, it becomes the dealer's turn.

Whenever it is the dealer's turn, he also requests cards one by one, but his strategy or policy is fixed without any choice. The dealer sticks on any sum of 17 or greater and hits otherwise.

If the dealer goes bust, then the player wins. Otherwise, the result of the game (win, draw, or lose) and hence the terminal reward ($+1$, 0, -1, respectively) is determined by whose sum is closer to 21. All rewards are 0 except the terminal rewards. The learning task is an undiscounted ($\gamma = 1$) episodic task, and therefore the terminal rewards are the returns.

An ace that can count as 11 without going bust is called a "usable" ace. A usable ace always counts as 11.

Learning progress is shown in Figure 2.9.

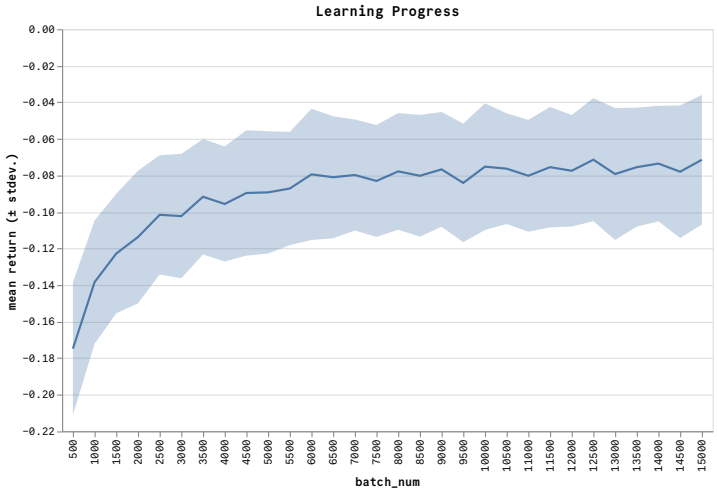


Figure 2.9: Learning progress of Q-learning for Blackjack.

2.4.8 Schnapsen (Discrete/Discrete)

The rules of Schnapsen[2] are available from the playing-card and board-game manufacturer Wiener Spielkartenfabrik Ferd. Piatnik & Söhne.

2.4.9 Autoregressive Trend Process (Continuous/Discrete)

Autoregressive trend processes such as the following serve as models for stock prices. We define the stochastic process

$$\begin{aligned} a_0 &:= 1, \\ b_0 &:= N(0,1), \\ a_k &:= a_{k-1} + b_{k-1} + \lambda N(0,1), \\ b_k &:= \kappa b_{k-1} + N(0,1). \end{aligned}$$

Here $N(\mu, \sigma^2)$ means drawing a random number from the normal distribution with mean μ and variance σ^2. We use $\lambda := 3$, $\kappa := 0.9$, and $k \in \{0, \dots, 99\}$. Having calculated 100 values $\{a_0, \dots, a_{99}\}$, the scaled values

$$p_k := \exp\left(\frac{a_k}{\max_{0 \le k < 100} a_k - \min_{0 \le k < 100} a_k}\right)$$

serve as simulated stock prices.

[2] https://www.piatnik.com/karten-spielregel

The actions in this episodic task are taking long, neutral, or short positions. The transactions costs are fractions of the stock prices; it is instructive to learn policies for cheap and expensive transaction costs.

2.5 Policies, Value Functions, and Bellman Equations

After the discussion of environments, rewards, returns, and episodes, we focus on concepts that underlie learning algorithms.

Learning optimal policies is the goal.

Definition 2.6 (policy)**.** A *policy* is a function $\mathcal{S} \times \mathcal{A}(s) \to [0,1]$. An agent is said to follow a policy π at time t, if $\pi(a|s)$ is the probability that the action $A_t = a$ is chosen if $S_t = s$.

Like the dynamics function p of the environment, a policy π is a function despite the notation that is reminiscent of a conditional probability. We denote the set of all policies by $\mathcal{P}$.

The following two functions, the (state-)value function and the action-value function, are useful for the agent because they indicate how expedient it is to be in a state or to be in a state and to take a certain action, respectively. Both functions depend on a given policy.

Definition 2.7 ((state-)value function)**.** The *value function* of a state s under a policy π is

$$v\colon \mathcal{P} \times \mathcal{S} \to \mathbb{R}, \qquad v_\pi(s) := \mathbb{E}_\pi[G_t \mid S_t = s],$$

i.e., it is the expected discounted return when starting in state s and following the policy π until the end of the episode.

Definition 2.8 (action-value function)**.** The *action-value function* of a state-action pair(s,a) under a policy π is

$$q\colon \mathcal{P} \times \mathcal{S} \times \mathcal{A}(s) \to \mathbb{R}, \qquad q_\pi(s,a) := \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a],$$

i.e., it is the expected discounted return when starting in state s, taking action a, and then following the policy π until the end of the episode.

Recursive formulae such as (2.4) are fundamental throughout reinforcement learning and dynamic programming. We now use (2.4) to find a recursive formula for the value function v_π by calculating

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{2.5a}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \tag{2.5b}$$
$$= \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r \mid s, a)(r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']) \tag{2.5c}$$
$$= \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r \mid s, a)(r + \gamma v_\pi(s')) \qquad \forall s \in \mathcal{S}. \tag{2.5d}$$

This equation is called the Bellman equation for v_π, and it is fundamental to computing, approximating, and learning v_π. The solution v_π exists uniquely if $\gamma < 1$ or all episodes are guaranteed to terminate from all states $s \in \mathcal{S}$ under policy π.

In the case of a finite MDP, the optimal policy is defined as follows. We start by noting that state-value functions can be used to define a partial ordering over the policies: a policy π is defined to be better than or equal to a policy π' if its value function v_π is greater than or equal to the value function $v_{\pi'}$ for all states. We write

$$\pi \geq \pi' \quad :\iff \quad v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}.$$

An optimal policy is a policy that is greater than or equal to all other policies. The optimal policy may not be unique. We denote optimal policies by π_*.

Optimal policies share the *same* state-value and action-value functions. The optimal state-value function is given by

$$v_*\colon \mathcal{S} \to \mathbb{R}, \qquad v_*(s) := \max_{\pi\in\mathcal{P}} v_\pi(s),$$

and the optimal action-value function is given by

$$q_*\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \qquad q_*(s, a) := \max_{\pi\in\mathcal{P}} q_\pi(s, a),$$

These two functions are related by the equation

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s,\ A_t = a] \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}(s). \tag{2.6}$$

Next, we find a recursion for these two optimal value functions similar to the Bellman equation above. Similarly to the derivation of (2.5), we calculate

$$v_*(s) = \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s, a) \tag{2.7a}$$
$$= \max_{a\in\mathcal{A}(s)} \mathbb{E}_{\pi_*}[G_t \mid S_t = s,\ A_t = a] \tag{2.7b}$$
$$= \max_{a\in\mathcal{A}(s)} \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s,\ A_t = a] \tag{2.7c}$$
$$= \max_{a\in\mathcal{A}(s)} \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.7d}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_*(s')) \qquad \forall s \in \mathcal{S}. \tag{2.7e}$$

The last two equations are both called the Bellman optimality equation for v_*. Analogously, two forms of the Bellman optimality equation for q_* are

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') \mid S_t = s,\ A_t = a] \tag{2.8a}$$

$$= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma \max_{a' \in \mathcal{A}(s)} q_*(s', a')) \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}(s). \tag{2.8b}$$

One can try to solve the Bellman optimality equations for v_* or q_*; they are just systems of algebraic equations. If the optimal action-value function q_* is known, an optimal policy π_* is easily found; we are still considering the case of a *finite* MDP. However, there are a few reasons why this approach is seldomly expedient for realistic problems:

- The Markov property may not hold.
- The dynamics of the environment, i.e., the function p, must be known.
- The system of equations may be huge.

2.6 On-Policy and Off-Policy Learning

2.7 Policy Evaluation (Prediction)

Policy evaluation means that we evaluate how well a policy π does by computing its state-value function v_π. The term "policy evaluation" is common in dynamic programming, while the term "prediction" is common in reinforcement learning. In policy evaluation, a policy π is given and its state-value function v_π is calculated.

Instead of solving the Bellman equation (2.5) for v_π directly, we follow an iterative approach. Starting from an arbitrary initial approximation $v_0 \colon \mathcal{S} \to \mathbb{R}$ (whose terminal state must have the value 0), we use (2.5) to define the iteration

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \tag{2.9a}$$

$$= \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_k(s')) \tag{2.9b}$$

for $v_{k+1} \colon \mathcal{S} \to \mathbb{R}$. This iteration is called iterative policy evaluation.

If $\gamma < 1$ or all episodes are guaranteed to terminate from all states $s \in \mathcal{S}$ under policy π, then this operator is a contraction and hence the approximations v_k converge to the state-value function v_π as $k \to \infty$, since v_π is the fixed point by the Bellman equation (2.5) for v_π.

The updates performed in (2.9) and in dynamic programming in general are called expected updates, since the expected value over all possible next states is computed in contrast to using a sample next state.

Algorithm 2 shows how this iteration can be implemented with updates performed in place. It also shows a common termination condition that uses the maximum norm and a prescribed accuracy threshold.

Algorithm 2 iterative policy evaluation for approximating $\mathbf{v} \approx v_\pi$ given $\pi \in \mathcal{P}$.

Initialization:
choose the accuracy threshold $\delta \in \mathbb{R}^+$,
initialize the vector $\mathbf{v}$ of length $|\mathcal{S}|$ arbitrarily
except that the value of the terminal state is 0.

repeat
  $d := 0$
  for all $s \in \mathcal{S}$ **do**
    $w := v_s$ ▷ save the old value
    $v_s := \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{s'})$
    $d := \max(d, |v_s - w|)$
  end for
until $d < \delta$

return v

2.8 Policy Improvement

Having seen how we can evaluate a policy, we now discuss how to improve it. To do so, the value functions show their usefulness. For now, we assume that the policies we consider here are deterministic.

Similarly to (2.6), the action value of selecting action a and then following policy π can be written as

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.10a}$$

$$= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_\pi(s')) \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}(s). \quad (2.10b)$$

This formula helps us determine if the action $\pi'(s)$ of another policy π' is an improvement over $\pi(s)$ in this time step. In order to be an improvement in this time step, the inequality

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

must hold. If this inequality holds, we also expect that selecting $\pi'(s)$ instead of $\pi(s)$ every time the state s occurs is an improvement. This is the subject of the following theorem.

Theorem 2.9 (policy improvement theorem)**.** *Suppose π and π' are two deterministic policies such that*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \qquad \forall s \in \mathcal{S}.$$

Then

$$\pi' \geq \pi,$$

i.e., the policy π' is greater than or equal to π.

Proof. By the definition of the partial ordering of policies, we must show that

$$v_{\pi'}(s) \geq v_\pi(s) \qquad \forall s \in \mathcal{S}.$$

Using the assumption and (2.10), we calculate

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s] \\
&\ \ \vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\
&= v_{\pi'}(s),
\end{aligned}$$

which concludes the proof. □

In addition to making changes to the policy in single states, we can define a new, greedy policy π' by selection the action that appears best in each state according to a given action-value function q_π. This greedy policy is given by

$$\pi'(s) :\in \arg\max_{a\in\mathcal{A}(s)} q_\pi(s,a), \tag{2.11a}$$

$$= \arg\max_{a\in\mathcal{A}(s)} \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.11b}$$

$$= \arg\max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r \mid s, a)(r + \gamma v_\pi(s')). \tag{2.11c}$$

Any ties in the arg max are broken in a random manner. By construction, the policy π' satisfies the assumption of Theorem 2.9, implying that it is better than or equal to π. This process is called policy improvement. We have created a new policy that improves on the original policy by making it greedy with respect to the value function of the original policy.

In the case that the $\pi' = \pi$, i.e., the new, greedy policy is as good as the original one, the equation $v_\pi = v_{\pi'}$ follows, and using the definition (2.11) of π', we find

$$v_{\pi'}(s) = \max_{a\in\mathcal{A}(s)} \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.12a}$$

$$= \max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{\pi'}(s')) \qquad \forall s \in \mathcal{S}, \tag{2.12b}$$

which is the Bellman optimality equation (2.7). Since $v_{\pi'}$ satisfies the optimality equation (2.7), $v_{\pi'} = v_*$ holds, meaning that both π and π' are optimal policies. In other words, policy improvement yields a strictly better policy unless the original policy is already optimal.

So far we have considered only deterministic policies, but these ideas can be extended to stochastic policies, and Theorem 2.9 also holds for stochastic policies. When defining the greedy policy, all maximizing actions can be assigned some nonzero probability.

2.9 Policy Iteration

Policy iteration is the process of using policy evaluation and policy improvement to define a sequence of monotonically improving policies and value functions. We start from a policy π_0 and evaluate it to find its state-value function v_{π_0}. Using v_{π_0}, we use policy improvement to define a new policy π_1. This policy is evaluated, and so forth, resulting in the sequence

$$\pi_0 \overset{\text{eval.}}{\longrightarrow} v_{\pi_0} \overset{\text{impr.}}{\longrightarrow} \pi_1 \overset{\text{eval.}}{\longrightarrow} v_{\pi_1} \overset{\text{impr.}}{\longrightarrow} \pi_2 \overset{\text{eval.}}{\longrightarrow} v_{\pi_2} \overset{\text{impr.}}{\longrightarrow} \cdots \overset{\text{impr.}}{\longrightarrow} \pi_* \overset{\text{eval.}}{\longrightarrow} v_*.$$

Unless a policy π_k is already optimal, it is a strict improvement over the previous policy π_{k-1}. In the case of a finite MDP, there is only a finite number of policies, and hence the sequence converges to the optimal policy and value function within a finite number of iterations.

A policy-iteration algorithm is shown in Algorithm 3. Each policy evaluation is started with the value function of the previous policy, which speeds up convergence. Note that the update of v_s has changed.

2.10 Value Iteration

A time-consuming property in Algorithm 3 is the fact that the repeat loop for policy evaluation is nested within the outer loop. Nesting these two loops may be quite time consuming. (The other inner loop is a for loop with a fixed number of iterations.) This suggests to try to get rid of these two nested loops while still guaranteeing convergence. An important simple case is to perform only one iteration policy evaluation, which makes it possible to combine policy evaluation and improvement into one loop. This algorithm is called value iteration, and it can be shown to convergence under the same assumptions that guarantee the existence of v_*.

Turning the fixed-point equation (2.12) into an iteration, value iteration becomes the update

$$\begin{aligned} v_{k+1}(s) &:= \max_{a \in \mathcal{A}(s)} \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s,\ A_t = a] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_k(s')). \end{aligned}$$

The algorithm is shown in Algorithm 4. Note that the update of v_s has changed again, now incorporating taking the maximum from the policy-improvement part.

2.11 Bibliographical and Historical Remarks

The classic textbook on dynamic programming is [35] by Richard Bellman. The most influential introductory textbook on RL is [23]. An excellent summary of the theory of Markov decision processes is [27].

Algorithm 3 policy iteration for calculating $\mathbf{v} \approx v_*$ and $\pi \approx \pi_*$.

Initialization:
choose the accuracy threshold $\delta \in \mathbb{R}^+$,
initialize the vector $\mathbf{v}$ of length $|\mathcal{S}|$ arbitrarily
  except that the value of the terminal state is 0,
initialize the vector π of length $|\mathcal{S}|$ arbitrarily.

loop
  policy evaluation:
  repeat
    $d := 0$
    for all $s \in \mathcal{S}$ **do**
      $w := v_s$ ▷ save the old value
      $v_s := \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, \pi(s))(r + \gamma v_{s'})$
        ▷ note the change: π_s is an optimal action
      $d := \max(d, |v_s - w|)$
    end for
  until $d < \delta$

  policy improvement:
  policyIsStable := true
  for all $s \in \mathcal{S}$ **do**
    old_action := $\pi[s]$ ▷ save the old value
    $\pi_s :\in \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{s'})$
      ▷ $\pi_s :\in \arg\max_{a \in \mathcal{A}(s)} q(s, a)$
    if old_action $\neq \pi(s)$ **then**
      policy_is_stable := false
    end if
  end for
  if policy_is_stable **then**
    return $\mathbf{v} \approx v_*$ and $\pi \approx \pi_*$
  end if
end loop

Algorithm 4 value iteration for calculating $\mathbf{v} \approx v_*$ and $\pi \approx \pi_*$.

Initialization:
choose the accuracy threshold $\delta \in \mathbb{R}^+$,
initialize the vector $\mathbf{v}$ of length $|\mathcal{S}|$ arbitrarily
  except that the value of the terminal state is 0,
initialize the vector π of length $|\mathcal{S}|$ arbitrarily.

policy evaluation and improvement:
repeat
  $d := 0$
  for all $s \in \mathcal{S}$ **do**
    $w := v_s$ ▷ save the old value
    $v_s := \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{s'})$
    ▷ note the change
    $d := \max(d, |v_s - w|)$
  end for
until $d < \delta$

calculate deterministic policy:
for all $s \in \mathcal{S}$ **do**
  $\pi_s :\in \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{s'})$
  ▷ $\pi_s :\in \arg\max_{a \in \mathcal{A}(s)} q(s, a)$
end for

return $\mathbf{v} \approx v_*$ and $\pi \approx \pi_*$

2.12 Exercises

2.12.1 Applications and Environments

Exercise 2.1. Think of another (preferably creative) application of reinforcement learning. Specify the environments, agents, states, actions, and rewards. Which assumptions are needed to satisfy the Markov property?

Exercise 2.2. Try to find a goal-directed learning task that cannot be represented by a Markov decision process.

Exercise 2.3 (Simple Grid World)**.** Implement Simple Grid World (see Section 2.4.1).

Exercise 2.4 (Windy Grid World)**.** Implement Windy Grid World (see Section 2.4.2).

Exercise 2.5 (Cliff Walking)**.** Implement Cliff Walking (see Section 2.4.3).

Exercise 2.6 (Frozen Lake)**.** Implement Frozen Lake (see Section 2.4.4).

Exercise 2.7 (variants of Frozen Lake)**.**

1. In Frozen Lake (see Section 2.4.4), there is a positive reward only when the goal is reached. However, in mazes, the rewards are usually -1 until the goal is reached. Implement the usual rewards as a variant and investigate any differences when learning the two variants.

2. Another variant is that holes do not end the episode, but incur a large negative reward and teleport the agent to the start. Implement this variant and investigate any differences in learning.

Exercise 2.8 (Rock paper scissors)**.** Implement Rock paper scissors (see Section 2.4.5).

Exercise 2.9 (Blackjack)**.** Implement Blackjack (see Section 2.4.7).

Exercise 2.10 (Schnapsen)**.** $*$ Implement Schnapsen (see Section 2.4.8).

Exercise 2.11 (autoregressive trend process)**.** Implement an autoregressive trend process (see Section 2.4.9).

Exercise 2.12 (ϵ-greedy action selection)**.** Assume that ϵ-greedy action selection is used and that there is a single greedy action.

1. Suppose $|\mathcal{A}| = 4$ and $\epsilon = 1/5$. What is the probability that the greedy action is selected?

2. Give a formula for calculating the probability of selecting the greedy action for any $|\mathcal{A}|$ and any ϵ.

2.12.2 Multi-Armed Bandits

Exercise 2.13 (multi-armed bandits with ϵ-greedy action selection (programming))**.** You play against a 10-armed bandit, where at the beginning of each episode the true value $q_*(a)$, $a \in \{1, \dots, 10\}$, of each of the 10 actions is chosen to be normally distributed with mean zero and unit variance. The rewards after choosing action/bandit a are normally distributed with mean $q_*(a)$ and unit variance. Using the simple bandit algorithm and ϵ-greedy action selection, you have 1000 time steps or tries in each episode to maximize the average reward starting from zero knowledge about the bandits.

Which value of ϵ maximizes the average reward? Which value of ϵ maximizes the percentage of optimal actions taken?

Exercise 2.14 (multi-armed bandits with upper-confidence-bound action selection (programming))**.** This exercise is the same as in Exercise 2.13, but now the actions

$$A_t :\in \arg\max_a \left(Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right)$$

are selected according to the upper-confidence bound.

Which value of c yields the largest average reward?

Exercise 2.15 (multi-armed bandits with soft-max action selection (programming))**.** This exercise is the same as Exercise 2.13, but now the actions $A_t \in \mathcal{A} = \{1, \dots, |\mathcal{A}|\}$ are selected with probability

$$\mathbb{P}[a] = \frac{\exp(Q_t(a)/\tau)}{\sum_{i=1}^{|\mathcal{A}|} \exp(Q_t(i)/\tau)},$$

where the parameter τ is called the temperature. This probability distribution is called the soft-max or Boltzmann distribution.

What are the effects of low and high temperatures, i.e., how does the temperature influence the probability distribution all else being equal? Which value of τ yields the largest average reward?

2.12.3 Step Sizes

Exercise 2.16 (harmonic step sizes)**.** Show that the step sizes

$$\alpha_n := \frac{1}{an + b}, \qquad a, b \in \mathbb{R},$$

(where $a \in \mathbb{R}^+$ and $b \in \mathbb{R}$ are chosen such that $an+b > 0$) satisfy the convergence conditions

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty.$$

Exercise 2.17 (unbiased step sizes)**.** We use the iteration

$$\begin{aligned} Q_1 &\in \mathbb{R}, \\ Q_{n+1} &:= Q_n + \alpha_n(R_n - Q_n), \qquad n \geq 1, \end{aligned}$$

to estimate Q_n using R_n, where

$$\alpha_n := \frac{\alpha}{\beta_n}, \qquad \alpha \in (0,1), \quad n \geq 1,$$

and

$$\begin{aligned} \beta_0 &:= 0, \\ \beta_n &:= \beta_{n-1} + \alpha(1 - \beta_{n-1}), \qquad n \geq 1. \end{aligned}$$

Show that the iteration for Q_n above yields an exponential recency-weighted average *without initial bias* (i.e., the Q_n do not depend on the initial value Q_1).

2.12.4 Basic Definitions

Exercise 2.18 (returns and episodes)**.** Suppose $\gamma := 1/2$ and the rewards $R_1 := 1$, $R_2 := -1$, $R_3 := 2$, $R_4 := -1$, and $R_5 := 2$ are received in an episode with length $T := 5$. What are $G_0, \ldots, G_5$?

Exercise 2.19 (returns and episodes)**.** Suppose $\gamma := 0.9$ and the reward sequence starts with $R_1 := -1$ and $R_2 := 2$ and is followed by an infinite sequence of 1s. What are G_0, G_1, and G_2?

Exercise 2.20 (change of return)**.** In episodic tasks and in continuing tasks, how does the return G_t change if a constant c is added to all rewards R_t? How does it change if all rewards R_t are multiplied by a constant c?

2.12.5 Dynamic Programming

Exercise 2.21 (equation for v_π)**.** Give an equation for v_π in terms of q_π and π.

Exercise 2.22 (equation for q_π)**.** Give an equation for q_π in terms of v_π and the four-argument p.

Exercise 2.23 (Bellman equation for q_π)**.** Analogous to the derivation of the Bellman equation for v_π, derive the Bellman equation for q_π.

Exercise 2.24 (equation for v_*)**.** Give an equation for v_* in terms of q_*.

Exercise 2.25 (equation for q_*)**.** Give an equation for q_* in terms of v_* and the four-argument p.

Exercise 2.26 (equation for π_*)**.** Give an equation for π_* in terms of q_*.

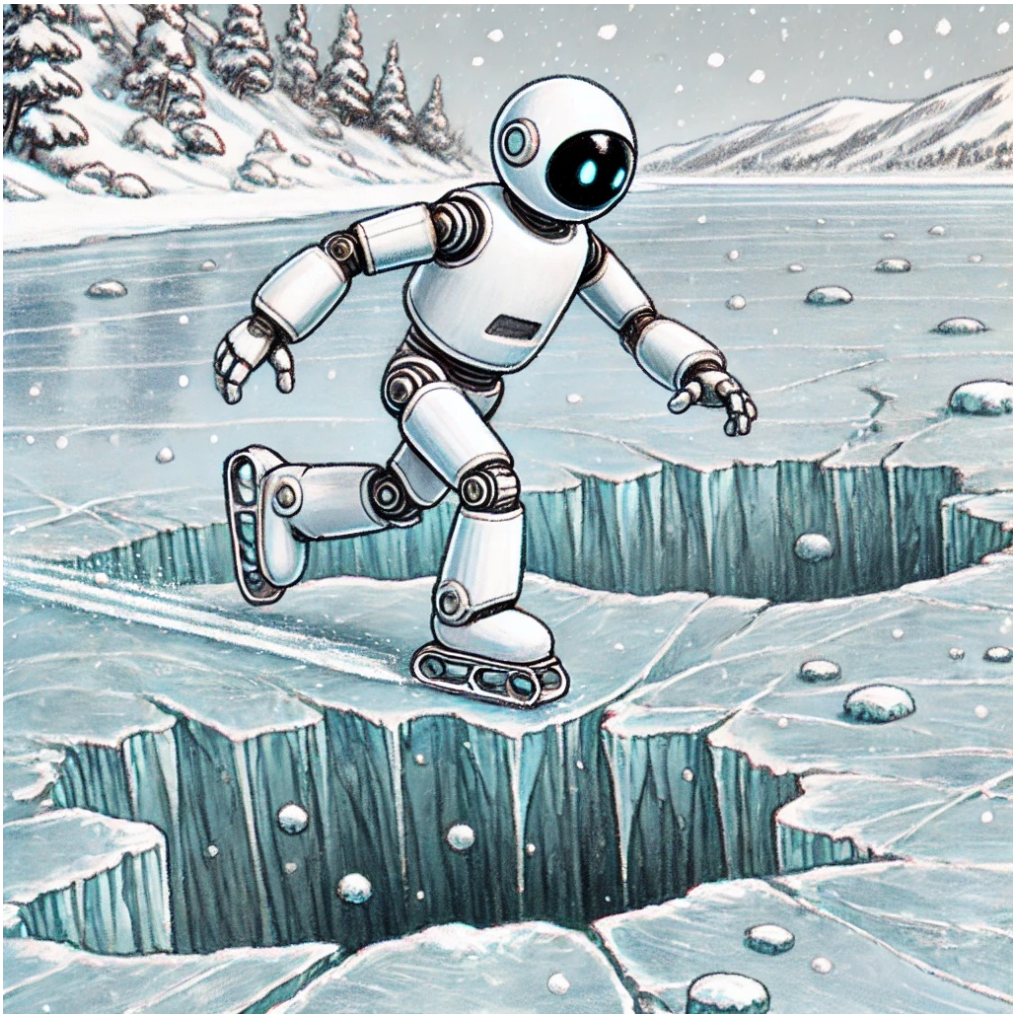
Exercise 2.27 (equation for π_*)**.** Give an equation for π_* in terms of v_* and the four-argument p.

Exercise 2.28 (update rule for q_π)**.** Using the Bellman equation for q_π (see Exercise 2.23), find an update rule for approximations q_{k+1} of q_π (in terms of q_k, π, and p) analogous to the update rule for v_{k+1}.

Exercise 2.29 (iterative policy evaluation (programming))**.** Implement iterative policy evaluation and use it to estimate v_π for the grid world in Exercise 2.3, where π is the equiprobable random policy.

Exercise 2.30 (policy iteration (programming))**.** Implement policy iteration and use it to estimate π_* for the grid world in Exercise 2.3.

Exercise 2.31 (value iteration (programming))**.** Implement value iteration and use it to estimate π_* for the grid world in Exercise 2.3.



Chapter 3

Taxonomy of Algorithms

Greek táxis: arrangement, order.

This chapter provides a taxonomy of reinforcement-learning algorithms for calculating state-value and action-value functions. In machine learning – and hence in reinforcement learning in particular –, an optimization and a generalization problem must be solved. The source of the optimization problem in reinforcement learning is obvious; we seek optimal policies. The generalization problem arises whenever function approximations on an infinite state or state-action space are used to represent the solutions. The type of function approximation must fit the properties of the environment and the state or state-action space such that an optimal solution can be represented well and overfitting is avoided at the same time. The approach in this chapter is to start from various errors and loss functions to be minimized and elucidate the choices that are made in order to arrive at various learning algorithms.

3.1 Introduction

Optimization clearly plays a fundamental role in reinforcement learning, as optimal policies are sought. In this chapter, we present and discuss optimization problems that occur on conjunction with function approximations of the state-value and action-value functions. In prediction, the state- and action-value functions of a given a policy are sought. In control, there are a indirect and direct ways to find an optimal policy. The indirect one is to calculate the action-value function of an optimal policy, which immediately yields an optimal policy when-

ever there is a finite number of actions; the direct one is to calculate the optimal policy.

The goal of the present taxonomy of algorithms is to elucidate the various choices that must be and can be made in order to arrive at learning algorithms starting from various choices of error expressions. In this way, a (hopefully) complete picture of learning algorithms and their interrelations obtained.

We start from a state space that is infinite; more precisely, it is a subset of a real vector space, i.e., $\mathcal{S} \subset \mathbb{R}^{\dim \mathcal{S}}$. Later, we will also discuss how the case of a finite state space can be considered a special case of the infinite case, unifying the theory of reinforcement learning for both finite and infinite state spaces. The following discussion applies to both prediction and control problems, i.e., to both state-value functions and action-value functions. The true state-value functions are denoted by v_π and v_*, and the true action-value functions are denoted by q_π and q_*. Their approximations are denoted by hats; for example,

$$\hat{V}(s, \mathbf{w}) \approx v_\pi(s),$$
$$\hat{Q}(s, a, \mathbf{w}) \approx q_\pi(s, a),$$

where the parameter vector

$$\mathbf{w} \in \mathbb{R}^d$$

and $d \ll \dim \mathcal{S}$ in general. If it would not be the case that $d \ll \dim \mathcal{S}$, the approximation and hence dimension reduction would hardly be worthwhile. We use capital letters for the approximations, since they are generally random variables, because they depend on the environment and on the policy.

3.2 Types of Errors

The first choice to be made in the derivation of a learning algorithm concerns the type of error. The error should be a norm, it is usually applied to the difference between the approximation and the true value, and it will be minimized.

We denote the L^p norm of a function f by $\|f\|_p$ and the weighted L^p norm of a function f with respect to a weight function μ by $\|f\|_{p,\mu}$.

3.2.1 The Mean Squared Value Error

The state distribution $\mu_\pi \colon \mathcal{S} \to [0, 1]$ of a policy π is the probability of visiting a state while following the policy π. State distributions are used in certain error definitions.

We consider episodic tasks first [23, Section 9.2]. We denote the initial state distribution by h, i.e., the probability that an episode begins in state s is $h(s)$.

We denote the average number of time steps spent in state s in a single episode by $\eta(s)$. Time is spent in state s' either because the episode starts there or because the policy and the environment result in the agent visiting state s' coming from any preceding state s, i.e.,

$$\eta(s') = h(s') + \gamma \int_{\mathcal{S}} \eta(s) \int_{\mathcal{A}(s)} \pi(a|s) p(s'|s,a) \mathrm{d}a \mathrm{d}s \qquad \forall s' \in \mathcal{S}.$$

Using η, we find the probabilities

$$\mu_\pi(s) = \frac{\eta(s)}{\int_{\mathcal{S}} \eta(s') \mathrm{d}s'} = \frac{\eta(s)}{\mathbb{E}_{s' \sim \eta}[1]}.$$

In continuing tasks ($\gamma = 1$), the state distribution is the stationary distribution under the policy π (and the environment) and therefore satisfies the equation

$$\begin{aligned} \mu_\pi(s') &= \int_{\mathcal{S}} \mu_\pi(s) \int_{\mathcal{A}(s)} \pi(a|s) p(s'|s,a) \mathrm{d}a \mathrm{d}s \\ &= \mathbb{E}_{s \sim \mu_\pi} \mathbb{E}_{a \sim \pi(s)}[p(s'|s,a)] \qquad \forall s' \in \mathcal{S}. \end{aligned}$$

Definition 3.1 (mean squared value error)**.** The *mean squared value error* is the function $\overline{\mathrm{VE}} \colon \mathcal{P} \times \mathbb{R}^d \to \mathbb{R}_0^+$,

$$\begin{aligned} \overline{\mathrm{VE}}_\pi(\mathbf{w}) &:= \|\hat{V}(.,\mathbf{w}) - v_\pi(.)\|_{2,\lambda}^2 = \mathbb{E}_{s \sim \lambda}[(\hat{V}(s,\mathbf{w}) - v_\pi(s))^2] \\ &= \int_{\mathcal{S}} \lambda(s) (\hat{V}(s,\mathbf{w}) - v_\pi(s))^2 \mathrm{d}s. \end{aligned}$$

There are two options for the weight function $\lambda \colon \mathcal{S} \to \mathbb{R}_0^+$:

1. All the weights are equal to one, i.e.,

$$\overline{\mathrm{VE}}_\pi(\mathbf{w}) = \|\hat{V}(.,\mathbf{w}) - v_\pi(.)\|_2^2.$$

2. The weights are the state distribution $\mu_\pi \colon \mathcal{S} \to [0,1]$ of the policy π.

3.2.2 The Mean Squared Return Error

The error between the value estimate at each time and the return from that time is always observable. The mean squared return error is the mean of the square of this error with respect to the state distribution.

Definition 3.2 (mean squared return error)**.** The *mean squared return error* is the function $\overline{\mathrm{RE}}\colon \mathcal{P} \times \mathbb{R}^d \to \mathbb{R}_0^+$,

$$\begin{aligned}\overline{\mathrm{RE}}_\pi(\mathbf{w}) &:= \mathbb{E}_{s\sim\mu_\pi}\mathbb{E}_\pi[(\hat{V}(S_t,\mathbf{w}) - G_t)^2 \mid S_t = s] \\ &= \int_{\mathcal{S}} \mu_\pi(s)\mathbb{E}_\pi[(\hat{V}(S_t,\mathbf{w}) - G_t)^2 \mid S_t = s]\mathrm{d}s.\end{aligned}$$

3.2.3 Mean Squared Bellman Errors

Definition 3.3 (mean squared Bellman error)**.** The *mean squared Bellman error* is the function $\overline{\mathrm{BE}}\colon \mathcal{P} \times \mathbb{R}^d \to \mathbb{R}_0^+$,

$$\overline{\mathrm{BE}}_\pi(\mathbf{w}) := \|B_\pi\hat{V}(\mathbf{w}) - \hat{V}(\mathbf{w})\|_{2,\mu_\pi}^2.$$

The projected version of this error depends on the projection Π that maps an arbitrary function v to the closest representable function

$$v_{\mathbf{w}_*} := \Pi[v],$$

which is determined by the approximation method used and the minimizing parameter vector

$$\mathbf{w}_* := \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \|v_\mathbf{w} - v\|_{2,\mu_\pi}^2 \in \mathbb{R}^d.$$

This projection is important, because computations can obviously only be performed on representable functions.

Definition 3.4 (mean squared projected Bellman error)**.** The *mean squared projected Bellman error* is the function $\overline{\mathrm{BE}}\colon \mathcal{P} \times \mathbb{R}^d \to \mathbb{R}_0^+$,

$$\overline{\mathrm{BE}}_\pi(\mathbf{w}) := \|\Pi[B_\pi\hat{V}(\mathbf{w}) - \hat{V}(\mathbf{w})]\|_{2,\mu_\pi}^2.$$

3.2.4 The Mean Squared Temporal-Difference Error

We start by defining the temporal-difference error

$$\delta_t := U_t(\mathbf{w}_t) - \hat{V}(S_t,\mathbf{w}_t),$$

where $U_t(\mathbf{w}_t)$ is any temporal-difference update target (see Chapter 4 and Chapter 5). By definition, a non-bootstrapping update target does not depend on any weight vector, while a bootstrapping one does. The simplest non-bootstrapping target is the return G_t, which results in the Monte-Carlo temporal-difference error

$$\delta_t = G_t - \hat{V}(S_t,\mathbf{w}_t).$$

Maybe the simplest of the bootstrapping update targets is the one-step temporal-difference target $R_{t+1}+\gamma\hat{V}(S_{t+1}, \mathbf{w}_t)$, resulting in the one-step temporal-difference error

$$\delta_t = R_{t+1} + \gamma\hat{V}(S_{t+1}, \mathbf{w}_t) - \hat{V}(S_t, \mathbf{w}_t).$$

Definition 3.5 (mean squared temporal-difference error)**.** The *mean squared temporal-difference error* is the function $\overline{\mathrm{TDE}}\colon \mathcal{P} \times \mathbb{R}^d \to \mathbb{R}_0^+$,

$$\begin{aligned}\overline{\mathrm{TDE}}_\pi(\mathbf{w}) &:= \mathbb{E}_\pi[\delta_t^2] = \mathbb{E}_{s\sim\mu_\pi}\mathbb{E}[\delta_t^2 \mid S_t = s, A_t \sim \pi] \\ &= \int_{\mathcal{S}} \mu_\pi(s)\mathbb{E}[\delta_t^2 \mid S_t = s, A_t \sim \pi]\mathrm{d}s.\end{aligned}$$

In off-policy learning (see Section 2.6), the behavior policy b differs from the target policy π. We can use the importance-sampling ratio

$$\rho_t := \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

(cf. Section 4.3) as a correction factor. This yields

$$\overline{\mathrm{TDE}}_\pi(\mathbf{w}) = \mathbb{E}_b[\rho_t\delta_t^2] = \mathbb{E}_{s\sim\mu_b}\mathbb{E}[\rho_t\delta_t^2 \mid S_t = s, A_t \sim b].$$

3.3 Learnability

Learnability is understood in the fundamental sense answering the question whether certain values can be calculated or learned given any amount of observations or experience. It does not concern the question whether it is too computationally expensive to compute certain values.

3.3.1 The Mean Squared Return Error

The mean squared return error is observable, since the return G_t is observable, and hence this error is learnable.

The mean squared value error $\overline{\mathrm{VE}}$ and the mean squared return error $\overline{\mathrm{RE}}$ are connected by

$$\overline{\mathrm{RE}}_\pi(\mathbf{w}) = \overline{\mathrm{VE}}_\pi(\mathbf{w}) + \mathbb{E}_{s\sim\mu_\pi}\mathbb{E}_\pi[(v_\pi(S_t) - G_t)^2] \tag{3.1}$$

in the on-policy case [23, Equation (11.24)]. The last term does not depend on the parameter vector $\mathbf{w}$, and therefore the two errors always have the same minimizer.

3.3.2 The Mean Squared Value Error

Figure 3.1 shows two different Markov decision processes that result in the same reward sequence [23, Section 11.6.]. The left process has a single state, a single action, and the reward is zero or one with probability 1/2. The right process has two states and a single action in each state. This process stays in the same state or switches to the other state with probability 1/2. The reward is zero from the first state and one from the second state, i.e., the reward is deterministic.

Figure 3.1: Two Markov decision processes that result in the same reward sequence, a uniformly distributed sequence of zeros and ones. In each state, there is only one possible action, which is hence not shown.

The reward sequences of both processes are identical; more precisely, they are sequences of zeros and ones, both values occurring with probability 1/2 in the sequence. Since there is only a single action available in each state, the policy is always trivial and hence not indicated in the following.

We assume that $\gamma = 0$. Then the value of the single state in the left process is $v(\mathrm{A}) = 1/2$, and the values of the states in the right process are $v(\mathrm{B}) = 0$ and $v(\mathrm{C}) = 1$.

We assume that all states appear the same after approximating the state-value function. In the left process, the value of the single state is approximated as $\hat{V}(\mathrm{A}, w) = w$. In the right process, the values of the two states are approximated as $\hat{V}(\mathrm{B}, w) = w = \hat{V}(\mathrm{C}, w)$. (This is the simplest special case of linear function approximation, see Section **??**. In the right process, the two states share a single feature.)

In the left process, the mean squared value error is $\overline{\mathrm{VE}}(w) = (\hat{V}(\mathrm{A}, w) - v(\mathrm{A}))^2 = (w - 1/2)^2$. Its minimizer $w_* := 1/2$ results in $\overline{\mathrm{VE}}(w_*) = 0$.

In the right process, the mean squared value error is $\overline{\mathrm{VE}}(w) = (1/2)(\hat{V}(\mathrm{B}, w) - v(\mathrm{B}))^2 + (1/2)(\hat{V}(\mathrm{C}, w) - v(\mathrm{C}))^2 = (1/2)w^2 + (1/2)(w-1)^2 = w^2 - w + 1/2$. Its minimizer $w_* := 1/2$, the same as before, now results in $\overline{\mathrm{VE}}(w_*) = 1/4$, which is different from before.

Of course, if the values of the states of the right process would be approximated as $\hat{V}(\mathrm{B}, \mathbf{w}) = w_1$ and $\hat{V}(\mathrm{C}, \mathbf{w}) = w_2$, then the mean squared value error would be $\overline{\mathrm{VE}}(\mathbf{w}) = (1/2)(\hat{V}(\mathrm{B}, \mathbf{w}) - v(\mathrm{B}))^2 + (1/2)(\hat{V}(\mathrm{C}, \mathbf{w}) - v(\mathrm{C}))^2 = (1/2)w_1^2 + (1/2)(w_2 - 1)^2$ and the minimizer would be $\mathbf{w}_* = (0, 1)$, which results

in $\overline{\mathrm{VE}}(\mathbf{w}_*) = 0$.

In order to summarize this example, there are two different Markov decision processes with the (statistically) same reward sequence. If the states cannot be observed or the approximation of the state-value function does not discern between the two states in the right process, then it is impossible to learn the mean squared value error; the error is different in the two processes, but the observations are identical.

In this sense, the mean squared value error is not learnable from the observations. It is no coincidence, however, that both processes in the example share the same minimizer of the error.

If two processes share the (statistically) same reward sequence, they have the same returns and hence the same mean squared return error, which is observable (see Section 3.3.1). Because they have the same mean squared return error, the minimizer is also the same. Next, due to (3.1), the mean squared return error and the mean squared value error have the same minimizer. Therefore, we have shown that two processes that have the (statistically) same reward sequence also have the same minimizer of the mean squared value error. In this sense, the mean squared value error is a viable objective for learning.

In short, the mean squared value error is not learnable, but its minimizer is.

3.3.3 The Mean Squared Temporal-Difference Error

The so-called A-split example is shown in Figure 3.2 [23, Example 11.2]. We use the one-step temporal difference update target. It can be shown that the minimal mean squared temporal-difference error is 1/16, while the error of the (true) value function (see Figure 3.2) is 1/8, i.e., larger.

Therefore, this example implies that minimizing the mean squared temporal-difference error is not expedient.

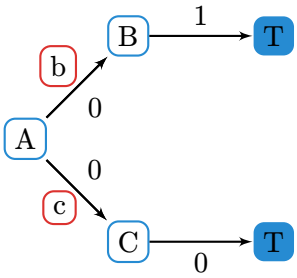


Figure 3.2: The A-split example. The policy π is given by $\pi(\mathrm{b}|\mathrm{A}) := 1/2$ and $\pi(\mathrm{c}|\mathrm{A}) := 1/2$. Its value function is $v_\pi(\mathrm{A}) = 1/2$, $v_\pi(\mathrm{B}) = 1$, and $v_\pi(\mathrm{C}) = 0$.

Chapter 4

Monte-Carlo Methods

> Monaco managed to keep up its veneer of sophistication only through the presence of its royal family and the high prices speculators, hoteliers, restaurateurs and shopkeepers charged. Even those had not created a safe buffer against some of the more garish encroachments of the 1980s. On his last visit, Bond had been horrified to find one-armed bandits installed in the exclusive Salles Privées of the Casino. Now he would not be surprised if there were space invader games there as well.
>
> John Gardner, *James Bond – Role of Honour,* Chapter 4.

In science and technology, Monte-Carlo methods approximate the stochastic unknown quantity of interest (e.g., an expected value) by generating many samples and calculating their statistics (e.g., the sample mean). In RL, the Monte-Carlo idea can be applied to the prediction and the control problem, and various algorithms are presented here. Convergence of the two main versions of Monte-Carlo prediction, namely first-visit and every-visit Monte-Carlo prediction, is shown, including the convergence rate of first-visit Monte-Carlo prediction.

4.1 Monte-Carlo Prediction

Prediction means learning the state-value function for a given policy π. The Monte-Carlo (MC) idea is to estimate the state-value function v_π at all states $s \in \mathcal{S}$ by averaging the returns obtained after the occurrences of each state s in many episodes. There are two variants:

- In first-visit MC, only the first occurrence of a state s in an episode is used to calculate the average and hence to estimate $v_\pi(s)$.

- In every-visit MC, all occurrences of a state s in an episode are used to calculate the average.

First-visit MC prediction is shown in Algorithm 5. In the every-visit variant, the check towards the end whether the occurrence is the first is left out.

Algorithm 5 first/every-visit MC prediction for calculating $v \approx v_*$ given the policy π.

initialization:
initialize the vector v of length $|\mathcal{S}|$ arbitrarily
initialize returns(s) to be an empty list for all $s \in \mathcal{S}$

loop ▷ for all episodes
  generate an episode $(S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T)$ following π
  $g := 0$
  for $t \in (T-1, T-2, \dots, 0)$ **do** ▷ for all time steps
    $g := \gamma g + R_{t+1}$
    if $S_t \notin \{S_0, \dots, S_{t-1}\}$ **then** ▷ remove check for every-visit MC
      append g to the list returns(S_t)
      $v(S_t) :=$ average(returns(S_t))
    end if
  end for
end loop

The converge of first-visit MC prediction to v_π as the number of visits goes to infinity follows from the law of large numbers, since each return is an independent and identically distributed estimate of $v_\pi(s)$ with finite variance. It is well-known that each average calculated in this manner is an unbiased estimate and that the standard deviation of the error is proportional to $n^{-1/2}$, where n is the number of occurrences of the state s.

The convergence proof for every-visit MC is more involved, since the occurrences are not independent.

The main advantage of MC methods is that they are simple methods. It is always possible to generate sample episodes.

Another feature of MC methods is that the approximations of $v_\pi(s)$ are independent from on another. The approximation for one state does not build on or depend on the approximation for another state, i.e., MC methods do not bootstrap.

Furthermore, the computational expense is independent of the number of states. Sometimes it is possible to steer the computation to interesting states

by choosing the starting states of the episodes suitably.

One can also try to estimate the action-value function q_π using MC. However, it is possible that many state-action pairs are never visited or only very seldomly. In other words, there may not be sufficient exploration. Sometimes it is possible to prescribe the starting state-action pairs of the episodes, which are then called exploring starts. It is then possible to remedy this problem, but it depends on the environment if this is possible or not. Exploring starts are not a general solution.

4.2 On-Policy Monte-Carlo Control

Control means to approximate optimal policies. The idea is the same as in Section 2.9, namely to iteratively perform policy evaluation and policy improvement. By Theorem 2.9, the sequences of value functions and policies converge to the optimal value functions and to the optimal policies under the assumption of exploring starts and under the assumption that infinitely many episodes are available.

An on-policy method is a method where the policy that is used to generate episodes is the same as the policy that is being improved. This is in contrast to off-policy methods, where these two policies are different ones (see Section 4.3).

But exploring starts are not available in general. Another important way to ensure sufficient exploration (and hence convergence) is to use ϵ-greedy policies as shown in Algorithm 6.

4.3 Off-Policy Methods and Importance Sampling

The dilemma between exploitation and exploration is a fundamental one in learning. The goal in action-value methods is to learn the correct action values, which depend on future optimal behavior. But at the same time, the algorithm must perform sufficient exploration to be able to discover optimal actions first.

The on-policy MC control algorithm, Algorithm 6 in the previous section comprises by using ϵ-greedy policies. Off-policy methods clearly distinguish between two policies:

- The behavior policy b is used to generate episodes. It is usually stochastic.
- The target policy π is the policy that is improved. It is usually the deterministic greedy policy with respect to the action-value function.

Algorithm 6 on-policy first-visit MC control for calculating $\pi \approx \pi_*$.

initialization:
choose $\epsilon \in (0, 1)$
initialize π to be an ϵ-greedy policy
initialize $q(s, a) \in \mathbb{R}$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$
initialize returns(s, a) to be an empty list for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$

loop ▷ for all episodes
  generate an episode $(S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T)$ following π
  $g := 0$
  for $t \in (T-1, T-2, \dots, 0)$ **do** ▷ for all time steps
    $g := \gamma g + R_{t+1}$
    if $S_t \notin \{S_0, \dots, S_{t-1}\}$ **then** ▷ remove check for every-visit MC
      append g to the list returns(S_t, A_t)
      $q(S_t, A_t) :=$ average(returns(S_t, A_t))
      $a_* :\in \arg\max_{a \in \mathcal{A}(s)} q(S_t, a)$ ▷ break ties randomly
      for all $a \in \mathcal{A}(S_t)$ **do**

$$\pi(a|S_t) := \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)|, & a = a_*, \\ \epsilon/|\mathcal{A}(S_t)|, & a \neq a_*. \end{cases}$$

      end for
    end if
  end for
end loop

The behavior and the target policies must satisfy the assumption of coverage, i.e., that $\pi(a|s) > 0$ implies $b(a|s) > 0$. In other words, every action that the target policy π performs must also be performed by the behavior policy b.

The by far most common technique used in off-policy methods is importance sampling. Importance sampling is a general technique that makes it possible to estimate the expected value under one probability distribution by using samples from another distribution. In off-policy methods it is of course used to adjust the returns from the behavior to the target policy.

We start by considering the probability

$$\mathbb{P}\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t,\ A_{t:T-1} \sim \pi\} = \prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1} \mid S_k, A_k)$$

that a trajectory $(A_t, S_{t+1}, A_{t+1}, \dots, S_T)$ occurs after starting from state S_t and following policy π. Then the relative probability

$$\rho_{t:T-1} := \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k) p(S_{k+1} \mid S_k, A_k)} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)}$$

of this trajectory under the target and behavior policies is called the importance-sampling ratio. Fortunately, the transition probabilities of the MDP cancel.

Now the importance-sampling ratio makes it possible to adjust the returns G_t from the behavior policy b to the target policy π. We are not interested in the state-value function

$$v_b(s) = \mathbb{E}[G_t \mid S_t = s]$$

of b, but in the state-value function

$$v_\pi(s) = \mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s]$$

of π.

Within a MC method, this means that we calculate averages of these adjusted returns. There are two variants of averages that are used for this purpose. First, we define some notation. It is convenient to number all time steps across all episodes consecutively. We denote the set of all time steps when state s occurs by $\mathcal{T}(s)$, the first time the termination state is reached after time t by $T(t)$, and the return after time t till the end of the episode at time $T(t)$ again by G_t. Then the set $\{G_t\}_{t \in \mathcal{T}(s)}$ contains all returns after visiting state s and the set $\{\rho_{t:T(t)-1}\}_{t \in \mathcal{T}(s)}$ contains the important-sampling ratios.

The first variant is called ordinary importance sampling. It is the mean of all adjusted returns $\rho_{t:T-1} G_t$, i.e.,

$$V_{\mathrm{o}}(s) := \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{|\mathcal{T}(s)|}.$$

In the second variant, the factors $\rho_{t:T-1}$ are interpreted as weights, and the weighted mean

$$V_{\mathrm{w}}(s) := \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1}}$$

is used. It is defined to be zero if the denominator is zero. This variant is called weighted importance sampling.

Although the weighted-average estimate has expectation $v_b(s)$ rather than the desired $v_\pi(s)$, it is much preferred in practice because it is much lower variance. On the other hand, ordinary importance sampling is easier to extend to approximate methods.

4.4 Convergence of First-Visit Monte-Carlo Prediction

In first-visit Monte Carlo, only the first visit to a state in every episode is used to estimate its value, while in every-visit Monte Carlo, all visits to a state in the episodes are used for calculating the value of the state. In the first-visit case, the theory is more straightforward due to the independence of the states in separate episodes, while an implementation needs to check in every episode whether a state has already been visited. First-visit Monte Carlo also generally requires more episodes to achieve the same confidence in estimating the value function, rendering it less data efficient.

Both first-visit and every-visit Monte-Carlo converge to the true value function. In this section, the convergence result for the first-visit Monte-Carlo prediction algorithm is stated and proved.

Theorem 4.1 (convergence of first-visit variant of Algorithm 5)**.** *Suppose that all episodes consist of a finite number of iterations, that the rewards are bounded, and that each state is visited an infinite number of times. Denote the number of returns used to calculate the sample mean V_n of the value function in the first-visit variant of Algorithm 5 by n. Then the sample mean V_n converges to the correct value function v_π for each state for a given policy π in distribution, more precisely,*

$$\forall s \in \mathcal{S}\colon \quad \exists \sigma_s \in \mathbb{R}^+\colon \quad \sqrt{n}(V_n(s) - v_\pi(s)) \xrightarrow[n\to\infty]{\mathrm{d}} N(0, \sigma_s^2).$$

Proof. The returns obtained in each state s by following the policy π are stored in the algorithm, and then their sample mean $V_n(s)$ is used as the estimate of the correct state value $v_\pi(s)$. Since only the first visit of any state is used in

each episode, each return is an independent and identically distributed random variable. Their expected values and their variances are finite, since the rewards are bounded by assumption. Therefore, by the central limit theorem, Theorem B.68, the sample means converge in distribution and the error decays as $1/\sqrt{n}$ for each state $s \in \mathcal{S}$. □

In summary, the proof is a straightforward application of the central limit theorem, which is proved in Chapter B together with the law of large numbers.

4.5 Convergence of Every-Visit Monte-Carlo Prediction

In this section, the convergence result for the every-visit Monte-Carlo prediction algorithm is stated and proved.

Theorem 4.2 (convergence of every-visit variant of Algorithm 5)**.** *Suppose that all episodes consist of a finite number of iterations, that the rewards are bounded, and that each state is visited an infinite number of times. Denote the number of returns used to calculate the sample mean V_n of the value function in the every-visit variant of Algorithm 5 by n. Then the sample mean V_n almost surely converges to the correct value function v_π for a given policy π, more precisely,*

$$\forall s \in \mathcal{S}\colon \quad V_n(s) \xrightarrow[n\to\infty]{\text{a. s.}} v_\pi(s).$$

The proof follows [36, Section 5.2].

Proof. We denote the integer-valued random variable that yields the number of visits to state $s \in \mathcal{S}$ in the k-th episode by $N_{s,k}$, and the $N_{s,k}$ samples of the return generated in the k-th episode are denoted by $R(s,k,m)$, $m \in \{1,\dots,N_{s,k}\}$.

Conditioned on $N_{s,k} \geq 1$, the random variables $N_{s,k}$ are non-negative, independent, and identically distributed. They are independent because the episodes are independent and they are identically distributed by the Markov property of the environment. By the same reasons, the random variables $\sum_{m=1}^{N_{s,k}} R(s,k,m)$ conditioned on $N_{s,k} \geq 1$ are also independent and identically distributed for different episodes k.

We denote the number of times that state s has been visited in all episodes by n_s, and the total number of visits to any state in all episodes by n. By assumption, all n_s, $s \in \mathcal{S}$, go to infinity as $n \to \infty$. The algorithm calculates

$$V_{n_s}(s) = \frac{\sum_{\{k\in\mathbb{N}:N_{s,k}\geq 1\}} \sum_{m=1}^{N_{s,k}} R(s,k,m)}{\sum_{\{k\in\mathbb{N}:N_{s,k}\geq 1\}} N_{s,k}},$$

where the nominator is the sum of all returns received in state s and the denominator is the number of all visits to state s up to that point. The quotient can be rewritten as

$$V_{n_s}(s) = \frac{\frac{1}{n_s}\sum_{\{k\in\mathbb{N}:N_{s,k}\geq 1\}}\sum_{m=1}^{N_{s,k}} R(s,k,m)}{\frac{1}{n_s}\sum_{\{k\in\mathbb{N}:N_{s,k}\geq 1\}} N_{s,k}}.$$

By the strong law of large numbers, Theorem B.67, applied to the nominator and the denominator, we have

$$V_{n_s}(s) \xrightarrow[n_s\to\infty]{\text{a. s.}} \frac{\mathbb{E}\left[\sum_{m=1}^{N_{s,k}} R(s,k,m) \;\middle|\; N_{s,k}\geq 1\right]}{\mathbb{E}\left[N_{s,k} \mid N_{s,k}\geq 1\right]}.$$

By Wald's equation, Theorem B.69, the quotient is equal to

$$\frac{\mathbb{E}\left[\sum_{m=1}^{N_{s,k}} R(s,k,m) \;\middle|\; N_{s,k}\geq 1\right]}{\mathbb{E}\left[N_{s,k} \mid N_{s,k}\geq 1\right]} = \mathbb{E}\left[R(s,k,1) \mid N_{s,k}\geq 1\right],$$

which is equal to $v_\pi(s)$ by the definition of the state-value function v_π.

In summary, we have shown that

$$V_n(s) \xrightarrow[n\to\infty]{\text{a. s.}} v_\pi(s),$$

which concludes the proof. □

4.6 Bibliographical and Historical Remarks

Further investigations into the convergence properties of first- and every-visit Monte Carlo can be found in [37].

4.7 Exercises

Exercise 4.1 (off-policy MC control (programming))**.** Implement off-policy MC control and use it to estimate π_* for the grid world in Exercise 2.3.

Exercise 4.2 (recursive formula for weighted importance sampling for off-policy learning)**.** In weighted importance sampling, we calculate the estimate

$$V_{n+1}^{\pi} := \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k}, \qquad n\geq 1,$$

given the returns $G_1, G_2, \dots$ and the weights $W_1, W_2, \dots$.

Show that the iteration

$$V_{n+1} := V_n + \frac{W_n}{C_n}(G_n - V_n), \qquad n \geq 1,$$

where

$$\begin{aligned} C_0 &= 0, \\ C_{n+1} &:= C_n + W_{n+1}, \qquad n \geq 0, \end{aligned}$$

yields the same values $V_n^\pi = V_n$ for all $n \in \{2, 3, \dots\}$.

Hint: Follow the derivation of the formula $Q_{n+1} = Q_n + (1/n)(R_n - Q_n)$.

Exercise 4.3 (importance-sampling ratio in off-policy MC control)**.** In the off-policy MC-control algorithm in [23, Section 5.7], the importance-sampling ratio is updated according to

$$W := \frac{W}{b(A_t|S_t)},$$

although the definition of the importance-sampling ratio $\rho_{t:T-1}$ implies

$$W := \frac{\pi(A_t|S_t)}{b(A_t|S_t)} W.$$

Why is the update in the algorithm nevertheless correct?


MONTE CARLO

Chapter 5

Temporal-Difference Learning

> In a number of important situations (in fact, in all decision-making situations), we face the problem of making decisions without a full knowledge of the basic workings of the underlying system. [...] It is necessary to learn and act at the same time. [...] Not only do we have to decide upon the allocation of time and other resources to the control activity; we also must decide how much time and effort to devote to studying the intrinsic nature of the system.
>
> Richard Bellman [38, p. 36].

5.1 Introduction

Temporal-difference (TD) methods are at the core of reinforcement learning. TD methods combine the advantages of dynamic programming (see Chapter 2) and MC (see Chapter 4). TD methods do not require any knowledge of the dynamics of the environments in contrast to dynamic programming, which requires full knowledge. The disadvantage of MC methods is that the end of an episode must be reached before any updates are performed; in TD updates are performed immediately or much earlier based on approximations learned earlier, i.e., they bootstrap approximations from previous approximations.

5.2 On-Policy Temporal-Difference Prediction: TD(0)

In prediction, an approximation V of the state-value function v_π is calculated for a given policy $\pi \in \mathcal{P}$. The simplest TD method is called TD(0) or one-step

TD. It performs the update

$$V(S_t) := V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{5.1}$$

after having received the reward R_{t+1}. Note that this equation has the form (2.1) with $R_{t+1} + \gamma V(S_{t+1})$ being the target value. The new approximation on the left side is based on the previous approximation on the right side. Therefore this method is a bootstrapping method.

The algorithm based on this update is shown in Algorithm 7.

Algorithm 7 TD(0) for calculating $V \approx v_\pi$ given π.

initialization:
choose learning rate $\alpha \in (0, 1]$
initialize the vector v of length $|\mathcal{S}|$ arbitrarily
    except that the value of the terminal state is 0

loop ▷ for all episodes
    initialize s
    repeat ▷ for all time steps
        set a to be the action given by π for s
        take action a and receive the new state s' and the reward r
        $v[s] := v[s] + \alpha(r + \gamma v[s'] - v[s])$
        $s := s'$
    until s is the terminal state and the episode is finished
end loop

From Chapter 2, we know about the state-value function that

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && \text{(5.2a)} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{(5.2b)} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. && \text{(5.2c)} \end{aligned}$$

These equations help us to interpret the differences between dynamic programming, MC, and TD. Considering the target values in the updates, the target in dynamic programming is an estimate because $v_\pi(S_{t+1})$ in (5.2c) is unknown and the previous approximation $V(S_{t+1})$ is used instead. The target in MC is an estimate, because the expected value in (5.2a) is unknown and a sample of the return is used instead. The target in TD is an estimate because of both reasons: $v_\pi(S_{t+1})$ in (5.2c) is replaced by the previous approximation $V(S_{t+1})$ and the expected value in (5.2c) is replaced by a sample of the return.

One advantage of TD methods is the fact that they do not require any knowledge of the environment just like MC methods. The advantage of TD methods over MC methods is that they perform an update immediately after having received a reward (or after some time steps in multi-step methods) instead of waiting till the end of the episode.

However, TD methods employ two approximations as discussed above. Do they still converge? The answer is yes. TD(0) converges to v_π (for given π) in the mean if the learning rate is constant and sufficiently small. It converges with probability one if the learning rate satisfies the stochastic-approximation conditions (2.2).

It has not been possible so far to show stringent results about the which method, MC or TD, converges faster. However, it has been found empirically that TD methods usually converge faster than MC methods with constant learning rates when the environment is stochastic.

5.3 On-Policy Temporal-Difference Control: SARSA

If we can approximate the optimal action-value function q_*, we can immediately find the best action $\arg\max_{a\in\mathcal{A}} q(s,a)$ in state s whenever the MDP is finite. In order to solve control problems, i.e., to calculate optimal policies, it therefore suffices to approximate the action-value function. In order to do so, we replace V in (5.1) by the approximation Q of the action-value function q_π to find the update

$$Q(S_t,A_t) := Q(S_t,A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)). \tag{5.3}$$

This method is called SARSA due to the appearance of the values S_t, A_t, R_{t+1}, S_{t+1}, and A_{t+1}.

The corresponding control algorithm is shown in Algorithm 8.

SARSA converges to an optimal policy and action-value function with probability one if all state-action pairs are visited an infinite number of times and the policy converges to the greedy policy. The convergence of the policy to the greedy policy can be ensured by using an ϵ-greedy policy and $\epsilon_t \to 0$.

5.4 On-Policy Temporal-Difference Control: Expected SARSA

Expected SARSA is derived from SARSA by replacing the target $R_{t+1}+\gamma Q(S_{t+1},A_{t+1})$ in the update by

$$R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1},A_{t+1}) \mid S_{t+1}].$$

Algorithm 8 SARSA for calculating $Q \approx q_*$ and π_*.

initialization:
choose learning rate $\alpha \in (0, 1]$
choose $\epsilon > 0$
initialize $q(s, a) \in \mathbb{R}$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$
except that the value of the terminal state is 0

loop ▷ for all episodes
initialize s
choose action a from s using an (ϵ-greedy) policy derived from q
repeat ▷ for all time steps
take action a and receive the new state s' and the reward r
choose action a' from s' using an (ϵ-greedy) policy derived from q
$q[s, a] := q[s, a] + \alpha(r + \gamma q[s', a'] - q[s, a])$
$s := s'$
$a := a'$
until s is the terminal state and the episode is finished
end loop

This means that the updates in expected SARSA moves in a deterministic manner into the same direction as the updates in SARSA move in expectation.

The update in expected SARSA hence is

$$\begin{aligned} Q(S_t, A_t) &:= Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)) \\ &= Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a \in \mathcal{A}(s)} \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)). \end{aligned}$$

Each update is more computationally expense than an update in SARSA. The advantage, however, is that the variance that is introduced due to the randomness in A_t is reduced.

5.5 Off-Policy Temporal-Difference Control: *Q*-Learning

One of the mainstays in reinforcement learning is Q-learning, which is an off-policy TD control algorithm [39]. Its update is

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a) - Q(S_t, A_t)).$$

Using this update, the approximation Q approximates q_* directly independently of the policy followed. Therefore, it is an off-policy method.

The policy that is being following still influences convergence and convergence speed. In particular, it must be ensured that all state-action pairs occur an infinite number of times. But this is a reasonable assumption, because their action values cannot be updated without visiting them.

The corresponding algorithm is shown in Algorithm 9.

Algorithm 9 Q-learning for calculating $Q \approx q_*$ and $\pi \approx \pi_*$.

initialization:
choose learning rate $\alpha \in (0, 1]$
choose $\epsilon > 0$
initialize $q(s, a) \in \mathbb{R}$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$
    except that the value of the terminal state is 0

loop ▷ for all episodes
    initialize s
    repeat ▷ for all time steps
        choose action a from s using an (ϵ-greedy) policy derived from q
        take action a and receive the new state s' and the reward r
        $q[s, a] := q[s, a] + \alpha(r + \gamma \max_{a \in \mathcal{A}(s')} q[s', a] - q[s, a])$
        $s := s'$
    until s is the terminal state and the episode is finished
end loop

Variants of Q-learning are presented in the following, and convergence results for Q-learning are given in Chapter 6.

5.6 Double Q-Learning

Since the target value in Q-learning involves the maximum over the estimate used for bootstrapping, a significant positive bias is introduced. It is often call a *maximization bias*.

How can the maximization bias be overcome? One method is called double learning. We note that having obtained one sample, the regular algorithm, Algorithm 9, uses it both to determine the maximizing action and to estimate its value. The idea of double learning is to partition the samples into two sets and to use them to learn two independent estimates Q_1 and Q_2. One estimate, say Q_1, is used to find the maximizing action

$$a_* := \arg\max_{a \in \mathcal{A}} Q_1(s, a);$$

the other estimate Q_2 is used to estimate its value

$$Q_2(s, a_*) = Q_2(s, \arg\max_{a \in \mathcal{A}} Q_1(s, a)).$$

Then this estimate is unbiased in the sense that

$$\mathbb{E}[Q_2(s, a_*)] = q(s, a_*).$$

The roles of Q_1 and Q_2 can of course then be switched.

In double learning, two estimates are calculated, which doubles the memory requirement. In each iteration, only one of the two approximations is updated so that the amount of computation per iteration remains the same.

The update for Q_1 is

$$Q_1(S_t, A_t) := Q_1(S_t, A_t) + \alpha_t(R_{t+1} + \gamma Q_2(S_{t+1}, \arg\max_{a \in \mathcal{A}(S_{t+1})} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)),$$

and the indices are switched in the update for Q_2.

In a double-learning algorithm, the choice whether Q_1 or Q_2 is updated is usually performed randomly. The resulting algorithm is shown in Algorithm 10.

5.7 Deep Q-Learning

In deep Q-learning, the optimal action-state function is represented by an artificial neural network. For example, in [13], a deep convolutional neural network was used. Convolutional neural networks are especially well suited for inputs that are images, which is the case in [13]. In [13], the state is the input of the neural network and there is a separate output neuron for each possible action. This has the advantage that the action-value function can be evaluated for all action in one forward pass.

Since artificial neural networks are nonlinear function approximators, convergence results are hard to obtain. In order to increase convergence speed or to ensure convergence at all, experience replay is usually used (cf. Remark 8.4 and Remark 6.8).

In [13], a separate neural network was used to generate the episodes. At regular intervals, i.e., after a fixed number of updates, the neural network was copied and this fixed copy was used to generate the next updates. This makes the algorithm more stable, since oscillations are prevented. In the Atari 2600 games played in [13], two consecutive states S_t and S_{t+1} are highly correlated hence oscillations are likely.

The algorithm in [13] is summarized in Algorithm 11.

Algorithm 10 double Q-learning for calculating $Q \approx q_*$ and $\pi \approx \pi_*$.

initialization:
choose learning rate $\alpha \in (0, 1]$
choose $\epsilon > 0$
initialize $Q_1[s, a] \in \mathbb{R}$ and $Q_2[s, a]$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$
except that the value of the terminal state is 0

loop ▷ for all episodes
initialize s
repeat ▷ for all time steps
choose action a from s using an (ϵ-greedy) policy derived from $Q := (Q_1 + Q_2)/2$
take action a and receive the new state s' and the reward r
if a random number chosen uniformly in $[0, 1)$ is less than $1/2$ **then**
$Q_1[s, a] := Q_1[s, a] + \alpha(r + \gamma Q_2[s', \arg\max_{a' \in \mathcal{A}(s')} Q_1[s', a')] - Q_1[s, a])$
else
$Q_2[s, a] := Q_2[s, a] + \alpha(r + \gamma Q_1[s', \arg\max_{a' \in \mathcal{A}(s')} Q_2[s', a')] - Q_2[s, a])$
end if
$s := s'$
until s is the terminal state and the episode is finished
end loop

Algorithm 11 deep Q-learning for calculating $Q \approx q_*$.

initialization:
initialize the replay memory M
initialize action-value function Q_θ with weights θ randomly
initialize target action-value function $\hat{Q}_\theta$ with weights θ randomly
initialize $N \in \mathbb{N}$

loop ▷ for all episodes
  initialize s
  repeat ▷ for all time steps
    choose action a from s using an (ϵ-greedy) policy derived from Q_θ
    take action a and receive the new state s' and the reward r
    store the transition (s, a, r, s') in the replay memory M
    sample a random minibatch of transitions $(s_i, a_i, r_{i+1}, s_{i+1})$ from M
    set the targets

$$y_i := \begin{cases} r_i, & \text{if } s_i \text{ is terminal state,} \\ r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_\theta(s_{i+1}, a'), & \text{otherwise} \end{cases}$$

    perform gradient descent step on $(y_i - Q_\theta(s_i, a_i))^2$
    every N steps reset $\hat{Q} := Q$
    $s := s'$
  until s is the terminal state and the episode is finished
end loop

5.8 On-Policy Multi-Step Temporal-Difference Prediction: n-step TD

The idea of multi-step temporal-difference methods is to perform not only one time step as in the methods in this chapter so far, but to perform multiple time steps and use the accumulated rewards as the target in the update. The multi-step methods will still be bootstrapping methods, of course.

Therefore we start by defining the return over n time steps being based on the state-value function afterwards.

Definition 5.1 (n-step return (using V))**.** The *n-step return* using V is defined as

$$G_{t:t+n} := \begin{cases} R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), & t+n < T, \\ G_t, & t+n \geq T \end{cases}$$

for all $n \geq 1$.

In the second case, when the time $t+n$ is equal to the last time T in an episode or larger, the n-step return $G_{t:t+n}$ is equal to the return G_t, which was defined to include all rewards up to the end of an episode.

Of course, the n-step return $G_{t:t+n}$ will only be available after having received the reward R_{t+n} in time step $t+n$. Hence no updates are possible in the first $n-1$ time steps of each episode. We also have to be careful when indexing the approximations V_t. Approximation V_{t+n} is available in time step $t+n$, it uses the n-step return $G_{t:t+n}$ as its target, and it bootstraps from the previous approximation $G_{t:t+n+1}$. The state value of state S_t, n steps in the past, is updated (with the information obtained in the future n steps), and the state values of all other states remain unchanged as usual in such update formulae.

Therefore we define the n-step TD update as

$$V_{t+n}(s) := \begin{cases} V_{t+n-1}(S_t) + \alpha(G_{t:t+n} - V_{t+n-1}(S_t)), & s = S_t, \\ V_{t+n-1}(s), & s \neq S_t \end{cases}$$

for all $0 \leq t < T$. In the case $n = 1$, we recover the one-step update (5.1).

The corresponding algorithm is shown in Algorithm 12. Some bookkeeping is required because the rewards for the updates must be accumulated first. The states S_t and rewards R_t are saved in vectors of length $n+1$. Since only their history of this length is required, S_t (and R_t) can be stored as the element number $t \mod n+1$.

Algorithm 12 n-step TD for calculating $V \approx v_\pi$ given π.

initialization:
choose number of steps n
choose learning rate $\alpha \in (0, 1]$
initialize the vector v of length $|\mathcal{S}|$ arbitrarily
    except that the value of the terminal state is 0

loop ▷ for all episodes
    initialize and store S_0
    $t := 0$
    $T := \infty$

    repeat ▷ for all time steps
        if $t < T$ **then**
            set a to be the action given by π for S_t
            take action a and receive the new state S_{t+1} and the reward R_{t+1}
            if S_{t+1} is terminal **then**
                $T := t + 1$
            end if
        end if

        $\tau := t - n + 1$ ▷ time step whose approximation is now updated
        if $\tau \geq 0$ **then**
            $G := \sum_{k=\tau+1}^{\min(\tau+n,T)} \gamma^{k-\tau-1} R_k$
            if $\tau + n < T$ **then**
                $G := G + \gamma^n v(S_{\tau+n})$
            end if
            $v[S_\tau] := v[S_\tau] + \alpha(G - v[S_\tau])$
        end if
        $t := t + 1$
    until $\tau + 1 = T$ ▷ corresponding to the final non-zero G, $G = R_{\tau+1} = R_T$
end loop

5.9 On-Policy Multi-Step Temporal-Difference Control: n-step SARSA

The multi-step version of SARSA is an extension of the one-step SARSA method in Section 5.3. Analogously to n-step TD, it uses n future rewards, but replaces the approximation V of the state-value function by the approximation Q of the action-value function. We therefore redefine the n-step return as follows.

Definition 5.2 (n-step return (using Q))**.** The *n-step return* using Q is defined as

$$G_{t:t+n} := \begin{cases} R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), & t+n < T, \\ G_t, & t+n \geq T \end{cases}$$

for all $n \geq 1$.

Therefore the n-step SARSA update is

$$Q_{t+n}(s,a) := \begin{cases} Q_{t+n-1}(S_t, A_t) + \alpha(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)), & (s,a) = (S_t, A_t), \\ Q_{t+n-1}(s,a), & (s,a) \neq (S_t, A_t) \end{cases}$$

for all $0 \leq t < T$. In the case $n = 1$, we recover the one-step update (5.3).

The corresponding algorithm is shown in Algorithm 13.

Algorithm 13 n-step SARSA for calculating $Q \approx q_*$ and $\pi \approx \pi_*$.

initialization:
choose number of steps n
choose learning rate $\alpha \in (0, 1]$
initialize $Q(s, a) \in \mathbb{R}$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$
    except that the value of the terminal state is 0
initialize π to be ϵ-greedy with respect to Q

loop ▷ for all episodes
    initialize and store S_0
    set A_0 to be the action given by π for S_0
    $t := 0$
    $T := \infty$

    repeat ▷ for all time steps
        if $t < T$ **then**
            take action a and receive the new state S_{t+1} and the reward R_{t+1}
            if S_{t+1} is terminal **then**
                $T := t + 1$
            else
                set a to be the action given by π for S_{t+1}
            end if
        end if

        $\tau := t - n + 1$ ▷ time step whose approximation is now updated
        if $\tau \geq 0$ **then**
            $G := \sum_{k=\tau+1}^{\min(\tau+n,T)} \gamma^{k-\tau-1} R_k$
            if $\tau + n < T$ **then**
                $G := G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
            end if
            $Q[S_\tau, A_\tau] := Q[S_\tau, A_\tau] + \alpha(G - Q[S_\tau, A_\tau])$
            set $\pi(\cdot|S_\tau)$ to be ϵ-greedy with respect to Q
        end if
        $t := t + 1$
    until $\tau + 1 = T$ ▷ corresponding to the final non-zero G, $G = R_{\tau+1} = R_T$
end loop

5.10 Bibliographical and Historical Remarks

5.11 Exercises

Chapter 6

Convergence of Discrete Q-Learning

> In any case, the mathematician sees hundreds and thousands of formidable new problems in dozens of blossoming areas, puzzles galore, and challenges to his heart's content. He may never resolve some of these, but he will never be bored. What more can he ask?
>
> Richard Bellman [38, p. 37].

In this chapter, it is shown that discrete Q-learning converges to the optimal action-value function under certain benign assumptions. Two different approaches and proofs are presented. The first is the original proof by Christopher Watkins and works by constructing another Markov decision process called the action-replay process, whose actions are optimal and which is sufficiently similar to the original process. The second approach uses a fixed-point argument to ensure the unique existence of the optimal action-value function and then uses the theory of stochastic processes to show that the difference between the learned policy and the optimal action-value function goes to zero as learning continues.

6.1 Introduction

Q-learning is an off-policy temporal-difference control method that directly approximates the optimal action-value function $q^*\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote the approximation in time step t of an episode by $Q_t\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The initial approximation Q_0 is initialized arbitrarily except that it vanishes for all terminal states. In each iteration, an action a_t is chosen from state s_t using a policy

derived from the previous approximation of the action-value function Q_t, e.g., using an ϵ-greedy policy, and a reward r_{t+1} is obtained and a new state s_{t+1} is achieved. The next approximation Q_{t+1} is defined as

$$Q_{t+1}(s,a) := \begin{cases} (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a)), & (s,a) = (s_t,a_t), \\ Q_t(s,a), & (s,a) \neq (s_t,a_t). \end{cases} \tag{6.1}$$

Here $\alpha_t \in [0,1]$ is the step size or learning rate and $\gamma \in [0,1]$ denotes the discount factor.

In this value-iteration update, only the value for (s_t,a_t) is updated. The update can be viewed as a weighted average of the old value $Q_t(s_t,a_t)$ and the new information $r_{t+1} + \gamma \max_a Q_t(s_{t+1},a)$, which is an estimate of the action-value function a time step later. Since the approximation Q_{t+1} depends on the previous estimate of q^*, Q-learning is a bootstrapping method.

The new value $Q_{t+1}(s_t,a_t)$ can also be written as

$$\underbrace{Q_{t+1}(s_t,a_t)}_{\text{new value}} := \underbrace{Q_t(s_t,a_t)}_{\text{old value}} + \alpha_t(\underbrace{r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q_t(s_{t+1},a)}_{\text{target value}} - \underbrace{Q_t(s_t,a_t)}_{\text{old value}}),$$

which is the form of a semigradient SGD method with a certain linear function approximation.

The learning rate α_t must be chosen appropriately, and convergence results hold only under certain conditions on the learning rate. If the environment is fully deterministic, the learning rate $\alpha_t := 1$ is optimal. If the environment is stochastic, then a necessary condition for convergence is that $\lim_{t\to\infty} \alpha_t = 0$.

If the initial approximation Q_0 is defined to have large values, exploration is encouraged at the beginning of learning. This kind of initialization is known as using optimistic initial conditions.

6.2 Convergence Proved Using Action Replay

Q-learning was introduced in [39, page 95], where an outline [39, Appendix 1] of the first convergence proof of one-step Q-learning was given as well. The original proof was extended and given in detail in [40]. It is presented in a summarized form in this section, because its approach is different from other, later proofs and because it gives intuitive insight into the convergence process.

The proof concerns the discrete or tabular method, i.e., the case of finite state and action sets. It is shown that Q-learning converges to the optimum action values with probability one if all actions are repeatedly sampled in all states.

Before we can state the theorem, a definition is required. We assume that all states and actions observed during learning have been numbered consecutively; then $N(i, s, a) \in \mathbb{N}$ is defined as the index of the i-th time that action a is taken in state s.

Theorem 6.1 (convergence of Q-learning proved by action replay)**.** *Suppose that the state and action spaces are finite. Suppose that the episodes that form the basis of learning include an infinite number of occurrences of each pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ (e.g., as starting state-action pairs). Given bounded rewards, the discount factor $\gamma < 1$, learning rates $\alpha_n \in [0, 1)$, and*

$$\sum_{i=1}^{\infty} \alpha_{N(i,s,a)} = \infty \quad \wedge \quad \sum_{i=1}^{\infty} \alpha_{N(i,s,a)}^2 < \infty \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A},$$

then

$$Q_n(s, a) \to q^*(s, a)$$

(as defined by (6.1)) as $n \to \infty$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ with probability one.

Sketch of the proof. The main idea is to construct an artificial Markov process called the action-replay process (ARP) from the sequence of episodes and the sequence of learning rates of the real process.

The ARP is defined as follows. It has the same discount factor as the real process. Its state space is (s, n), where s is either a state of the real process or a new, absorbing and terminal state, and $n \in \mathbb{N}$ is an index whose significance is discussed below. The action space is the same as the real process.

Action a at state (s, n_1) in the ARP is performed as follows. All transitions observed during learning are recorded in a sequence consisting of tuples

$$T_t := (s_t, a_t, s'_t, r'_t, \alpha_t).$$

Here a_t is the action taken in state s_t yielding the new state s'_t and the reward r'_t while using the learning rate α_t. To perform action a at state (s, n_1) in the ARP, all transitions after and including transition n_1 are eliminated and not considered further. Starting at transition $n_1 - 1$ and counting down, the first transition $T_t = (s_t, a_t, s'_t, r'_t, \alpha_t)$ before transition number n_1 whose starting state s_t and action a_t match (s, a) is found and its index is called n_2, i.e., n_2 is the largest index less than n_1 such that $(s, a) = (s_{n_2}, a_{n_2})$. With probability α_{n_2}, the transition $T_{n_2} = (s_{n_2}, a_{n_2}, s'_{n_2}, r'_{n_2}, \alpha_{n_2})$ is replayed. Otherwise, with probability $1 - \alpha_{n_2}$, the search is repeated towards the beginning. If, however, there is no such n_2, i.e., if there is no matching state-action pair, the reward

$r'_{n_1} = Q_0(s,a)$ of transition T_{n_1} is recorded and the episode in the ARP stops in an absorbing, terminal state.

Replaying a transition T_{n_2} in the ARP is defined to mean that the ARP state (s, n_1) is followed by the state $(s'_{n_2}, n_2 - 1)$; the state $s = s_{n_2}$ was followed by the state s'_{n_2} in the real process after taking action $a = a_{n_2}$.

The next transition in the ARP is found by following this search process towards the beginning of the transitions recorded during learning, and so forth. The ARP episode ultimately terminates after a finite number of steps, since the index n in its states (s, n) decreases strictly monotonically. Because of this construction, the ARP is a Markov decision process just as the real process.

Having constructed the ARP, the proof proceeds in two steps recorded as lemmata. The first lemma says that the approximations $Q_n(s,a)$ calculated during Q-learning are the optimal action values for the ARP states and actions. The rest of the lemmata say that the ARP converges to the real process. We start with the first lemma.

Lemma 6.2. *The optimal action values for ARP states (s,n) and ARP actions a are $Q_n(s,a)$, i.e.,*

$$Q_n(s,a) = Q^*_{\mathrm{ARP}}((s,n),a) \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \quad \forall n \in \mathbb{N}.$$

Proof. The proof is by induction with respect to n. Due to the construction of the ARP, the action value $Q_0(s,a)$ is optimal, since it is the only possible action value of $(s,0)$ and a. In other words, the induction basis

$$Q_0(s,a) = Q^*_{\mathrm{ARP}}((s,0),a)$$

holds true.

The show the induction step, we suppose that the action values Q_n as calculated by the Q-learning iteration are optimal action values for the ARP at level n, i.e.,

$$Q_n(s,a) = Q^*_{\mathrm{ARP}}((s,n),a) \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A},$$

which implies

$$V^*_{\mathrm{ARP}}((s,n)) = \max_a Q_n(s,a)$$

for the optimal state values V^*_{ARP} of the ARP at level n. (Note that the last equation motivates the use of the maximum in Q-learning.)

In order to perform action a in state $(s, n+1)$, we consider two cases. The first case is $(s,a) \neq (s_n, a_n)$. In this case, performing a in state $(s, n+1)$ is the same as performing a in state (s,n) by the definition of Q-learning; nothing changes in the second case in (6.1). Therefore we have

$$Q_{n+1}(s,a) = Q_n(s,a) = Q^*_{\mathrm{ARP}}((s,n),a) = Q^*_{\mathrm{ARP}}((s,n+1),a)$$
$$\forall (s,a) \in \mathcal{S} \times \mathcal{A} \setminus \{(s_n,a_n)\} \quad \forall n \in \mathbb{N}.$$

The second case is $(s,a) = (s_n,a_n)$. In this case, performing a_n in the state $(s_n,n+1)$ is equivalent

- to obtaining the reward r'_n and the new state (s'_n,n) with probability α_n or
- to performing a_n in the state (s_n,n) with probability $1-\alpha_n$.

The induction hypothesis and the definition of Q-learning hence yield

$$\begin{aligned} Q^*_{\mathrm{ARP}}((s_n,n+1),a_n) &= (1-\alpha_n)Q^*_{\mathrm{ARP}}((s_n,n),a_n) + \alpha_n(r'_n + \gamma V^*_{\mathrm{ARP}}((s'_n,n)) \\ &= (1-\alpha_n)Q_n(s_n,a_n) + \alpha_n(r'_n + \gamma \max_a Q_n(s'_n,a)) \\ &= Q_{n+1}(s_n,a_n) \qquad \forall n \in \mathbb{N}. \end{aligned}$$

Both cases together mean that

$$Q_{n+1}(s,a) = Q^*_{\mathrm{ARP}}((s,n+1),a) \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \quad \forall n \in \mathbb{N},$$

which concludes the proof of the induction step. □

The rest of the lemmata say that the ARP converges to the real process. Here we prove only the first and second one; the rest are shown in [40].

Lemma 6.3. *Consider a finite Markov process with a discount factor $\gamma < 1$ and bounded rewards. Also consider two episodes, both starting at state s and followed by the same finite sequence of k actions, before continuing with any other actions. Then the difference of the values of the starting state s in both episodes tends to zero as $k \to \infty$.*

Proof. The assertion follows because $\gamma < 1$ and the rewards are bounded. □

Lemma 6.4. *Consider the ARP defined above with states (s,n) and call n the level. Then, given any level $n_1 \in \mathbb{N}$, there exists another level $n_2 \in \mathbb{N}$, $n_2 > n_1$, such that the probability of reaching a level below n_1 after taking $k \in \mathbb{N}$ actions starting from a level above n_2 is arbitrarily small.*

In other words, the probability of reaching any fixed level n_1 after starting at level n_2 of the ARP tends to zero as $n_2 \to \infty$. Therefore there is a sufficiently high level n_2 from which k actions can be performed with an arbitrarily high probability of leaving the ARP episode above level n_1.

Proof. We start by determining the probability P of reaching a level below n_1 starting from a state (s, n) with $n > n_1$ and performing action a. Recall that $N(i, s, a)$ is the index of the i-th time that action a is taken in state s. We define i_1 to be the smallest index i such that $N(i, s, a) \geq n_1$ and i_2 to be the largest index i such that $N(i, s, a) \leq n_2$. We also define $\alpha_{N(0,s,a)} := 1$. Then the probability is

$$P = \underbrace{\left(\prod_{i=i_1}^{i_2} (1 - \alpha_{N(i,s,a)}) \right)}_{\text{continue above level } i_1} \underbrace{\sum_{j=0}^{i_1-1} \alpha_{N(j,s,a)} \prod_{k=j+1}^{i_1-1} (1 - \alpha_{N(k,s,a)})}_{\text{stop below level } i_1}.$$

The sum is less equal one, since it is the probability that the episode ends (below level i_1). Therefore we have the estimate

$$P \leq \prod_{i=i_1}^{i_2} (1 - \alpha_{N(i,s,a)}) \leq \exp\left(- \sum_{i=i_1}^{i_2} \alpha_{N(i,s,a)} \right),$$

where the second inequality holds because the inequality $1 - x \leq \exp(-x)$ for all $x \in [0, 1]$ has been applied to each term. Since the sum of the learning rates diverges by assumption, we find that $P \leq \exp\left(- \sum_{i=i_1}^{i_2} \alpha_{N(i,s,a)} \right) \to 0$ as $n_2 \to \infty$ and hence $i_2 \to \infty$.

Since the state and action spaces are finite, for each probability $\epsilon \in (0, 1]$, there exists a level n_2 such that starting above it from any state s and taking action a leads to a level above n_1 with probability at least $1 - \epsilon$. This argument can be applied k times for each action, and the probability ϵ can be chosen small enough such that the overall probability of reaching a level below n_1 after taking k actions becomes arbitrarily small, which concludes the proof. □

Before stating the next lemma, we denote the transition probabilities of the ARP by $p_{\text{ARP}}((s', n') \mid (s, n), a)$ and its expected rewards by $R_n(s, a)$. We also define the probability

$$p_n(s' \mid s, a) := \sum_{n'=1}^{n-1} p_{\text{ARP}}((s', n') \mid (s, n), a)$$

that performing action a at state (s, n) (at level n) in the ARP leads to the state s' at a lower level.

Lemma 6.5. *With probability one, the transition probabilities $p_n(s' \mid s, a)$ at level n and the expected rewards $R_n(s, a)$ at level n of the ARP converge to the transition probabilities and expected rewards of the real process as the level n tends to infinity.*

Sketch of the proof. The proof of this lemma [40, Lemma B.3] relies on a standard theorem in stochastic convergence (see, e.g., [41, Theorem 2.3.1]), which states that if random variables X_n are updated according to

$$X_{n+1} := X_n + \beta_n(\xi_n - X_n),$$

where $\beta_n \in [0,1)$, $\sum_{n=1}^{\infty} \beta_n = \infty$, $\sum_{n=1}^{\infty} \beta_n^2 < \infty$, and the random variables ξ_n are bounded and have mean ξ, then

$$X_n \to \xi \quad \text{as } n \to \infty \text{ with probability one.}$$

This theorem is applied to the two update formulae for the transition probabilities and expected rewards for going from occurrence $i+1$ to occurrence i. Since there is only a finite number of states and actions, the convergence is uniform. □

Lemma 6.6. *Consider episodes of $k \in \mathbb{N}$ actions in the ARP and in the real process. If the transition probabilities $p_n(s' \mid s,a)$ and the expected rewards $R_n(s,a)$ at appropriate levels of the ARP for each of the actions are sufficiently close to the transition probabilities $p(s' \mid s,a)$ and expected rewards $R(s,a)$ of the real process for all actions a, for all states s, and for all states s', then the value of the episode in the ARP is close to its value in the real process.*

Sketch of the proof. The difference in the action values of a finite number k of actions between the ARP and the real process grows at most quadratically with k. Therefore, if the transition probabilities and mean rewards are sufficiently close, the action values must also be close. □

Using these lemmata, we can finish the proof of the theorem. The idea is that the ARP tends towards the real process and hence its optimal action values do as well. The values $Q_n(s,a)$ are the optimal action values for level n of the ARP by Lemma 6.2, and therefore they tend to $Q^*(s,a)$.

More precisely, we denote the bound of the rewards by $R \in \mathbb{R}_0^+$ such that $|r_n| \leq R$ for all $n \in \mathbb{N}$. Without loss of generality, it can be assumed that $Q_0(s,a) < R/(1-\gamma)$ and that $R \geq 1$. For an arbitrary $\epsilon \in \mathbb{R}^+$, we choose $k \in \mathbb{N}$ such that

$$\gamma^k \frac{R}{1-\gamma} < \frac{\epsilon}{6}$$

holds.

By Lemma 6.5, with probability one, it is possible to find a sufficiently large $n_1 \in \mathbb{N}$ such that the inequalities

$$|p_n(s' \mid s,a) - p(s' \mid s,a)| < \frac{\epsilon}{3k(k+1)R},$$

$$|R_n(s,a) - R(s,a)| < \frac{\epsilon}{3k(k+1)}$$

hold for the differences between the transition probabilities and expected rewards of the ARP and the real process for all $n > n_1$ and for all actions a, for all states s, and for all states s'.

By Lemma 6.4, it is possible to find a sufficiently large $n_2 \in \mathbb{N}$ such that for all $n > n_2$ the probability of reaching a level lower than n_1 after taking k actions is less than $\min\{\epsilon(1-\gamma)/6kR, \epsilon/3k(k+1)R\}$. This implies that the inequalities

$$|p'_n(s' \mid s,a) - p(s' \mid s,a)| < \frac{2\epsilon}{3k(k+1)R},$$
$$|R'_n(s,a) - R(s,a)| < \frac{2\epsilon}{3k(k+1)}$$

hold, where the primes on the probabilities indicate that they are conditional on the level of the ARP after k actions being greater than n_1.

Then, by Lemma 6.6, the difference between the value $Q_{\mathrm{ARP}}((s,n),a_1,\dots,a_k)$ of performing actions $a_1,\dots,a_k$ at state (s,n) in the ARP and the value $Q(s,a_1,\dots,a_k)$ of performing these actions in the real process is bounded by the inequality

$$|Q_{\mathrm{ARP}}((s,n),a_1,\dots,a_k) - Q(s,a_1,\dots,a_k)| < \frac{\epsilon(1-\gamma)}{6kR}\frac{2kR}{1-\gamma} + \frac{2\epsilon}{3k(k+1)}\frac{k(k+1)}{2} = \frac{2\epsilon}{3}.$$

The first term is the difference if the conditions for Lemma 6.4 are not satisfied, since the cost of reaching a level below n_1 is bounded by $2kR/(1-\gamma)$. The second term is the difference from Lemma 6.6 stemming from imprecise transition probabilities and expected rewards.

By Lemma 6.3, the difference due to taking only k actions is less than $\epsilon/6$ for both the ARP and the real process.

Since the inequality above applies to any sequence of actions, it applies in particular to a sequence of actions optimal for either the ARP or the real process. Therefore the estimate

$$|Q^*_{\mathrm{ARP}}((s,n),a) - Q^*(s,a)| < \epsilon$$

holds. In conclusion, $Q_n(s,a) \to Q^*(s,a)$ as $n \to \infty$ with probability one, which concludes the proof of the theorem. □

Remark 6.7 (the non-discounted case $\gamma = 1$ with absorbing goal states). If the discount factor $\gamma = 1$, but the Markov process has absorbing goal states, the

proof can be modified [40, Section 4]. The certainty of being trapped in an absorbing goal state then plays the role of $\gamma < 1$ and ensures that the value of each state is bounded under any policy and that Lemma 6.3 holds.

Remark 6.8 (action replay and experience replay). The assumption that all pairs of states and actions occur an infinite number of times during learning is crucial for the proof. The proof also suggests that convergence is faster if the occurrences of states and actions are equidistributed. This fact motivates the use of action or experience replay. Experience replay is a method (not limited to be used in conjunction with Q-learning) which keeps a cache of states and actions that have been visited and which are replayed during learning in order to ensure that the whole space is sampled in an equidistributed manner. This is beneficial when the states of the Markov chain are highly correlated. Experience replay was used, e.g., in [13]. Cf. Remark 8.4.

6.3 Convergence Proved Using Stochastic Approximation

It is natural to consider the iterates of temporal-difference learning algorithms as a stochastic process. Then the theory of stochastic approximation (see Chapter C) may provide tools to show almost sure convergence or convergence in mean square (or in quadratic mean). Unfortunately, however, the theory of stochastic approximation for multidimensional stochastic processes requires an underlying Hilbert space (see Section C.2), while the empirical Bellman operator in the Q-learning iteration is a contraction only with respect to the maximum norm, and the maximum norm does not stem from any inner product, which would be necessary for a Hilbert space. This major technical hurdle must be overcome when applying the theory of stochastic approximation to the convergence of Q-learning. Furthermore, the noise terms may not satisfy standard assumptions made in stochastic approximation.

The general approach is to show that the empirical Bellman operator is a contraction with respect to the maximum norm first. Its unique fixed point is the supposed limit, i.e., the optimal action-value function. Then the iteration formula is split into a deterministic and a stochastic (or noise) term and it is matched to a stochastic process whose convergence is known by a lemma. This lemma is usually a generalization of a result known from stochastic approximation.

This is the basic approach taken in [42] for Q-learning and TD(λ) and in [43] and [36, Section 5.6] for Q-learning. By extending the same approach in

[44, 45], convergence theorems for Q-learning for environments that change over time, but whose accumulated changes remain bounded, were shown in [46].

Theorem 6.9 (stochastic approximation)**.** *The stochastic process*

$$w_{t+1}(s) := (1 - \alpha_t(s))w_t(s) + \beta_t(s)r_n(s),$$

where all random variables may depend on the past P_t, converges to zero with probability one if the following conditions are satisfied.

1. *The step sizes $\alpha_t(s)$ and $\beta_t(s)$ satisfy the equalities and inequalities $\sum_t \alpha_t(s) = \infty$, $\sum_t \alpha_t(s)^2 < \infty$, $\sum_t \beta_t(s) = \infty$, and $\sum_t \beta_t(s)^2 < \infty$ uniformly.*
2. *$\mathbb{E}[r_t(s) \mid P_t] = 0$ and there exists a constant $C \in \mathbb{R}^+$ such that $\mathbb{E}[r_t(s)^2 \mid P_t] \leq C$ with probability one.*

Lemma 6.10. *Consider the stochastic process*

$$X_{t+1}(s) = G_t(X_t, s),$$

where

$$G_t(\beta X_t, s) = \beta G_t(X_t, s).$$

Suppose that if the norm $\|X_t\|$ were kept bounded by scaling in all iterations, then X_t would converge to zero with probability one. Then the original stochastic process converges to zero with probability one.

Lemma 6.11. *The stochastic process*

$$X_{t+1}(s) = (1 - \alpha(s))X_t(s) + \gamma\beta_t(s)\|X_t\|$$

converges to zero with probability one if the following conditions are satisfied.

1. *The state space $\mathcal{S}$ is finite.*
2. *The step sizes $\alpha_t(s)$ and $\beta_t(s)$ satisfy the equalities and inequalities $\sum_t \alpha_t(s) = \infty$, $\sum_t \alpha_t(s)^2 < \infty$, $\sum_t \beta_t(s) = \infty$, and $\sum_t \beta_t(s)^2 < \infty$ uniformly.*
3. *The step sizes satisfy the inequality*

$$\mathbb{E}[\beta_t(s)] \leq \mathbb{E}[\alpha_t(s)]$$

uniformly with probability one.

Theorem 6.12 (convergence of a stochastic process [42]). *The stochastic process*

$$\Delta_{t+1}(s) := (1 - \alpha_t(s))\Delta_t(s) + \beta_t(s)F_t(s) \tag{6.2}$$

converges to zero almost surely if the following assumptions hold.

1. *The state space is finite.*
2. *The equalities and inequalities*

$$\begin{aligned}\sum_t \alpha_t &= \infty,\\ \sum_t \alpha_t^2 &< \infty,\\ \sum_t \beta_t &= \infty,\\ \sum_t \beta_t^2 &< \infty,\\ \mathbb{E}[\beta_t|P_t] &\leq \mathbb{E}[\alpha_t|P_t]\end{aligned}$$

   are satisfied uniformly and almost surely.

3. *The inequality*

$$\|\mathbb{E}[F_t(s)|P_t]\|_W \leq C\|\Delta_t\|_W \qquad \exists C \in (0,1)$$

   holds.

4. *The inequality*

$$\mathbb{V}[F_t(s)|P_t] \leq C(1 + \|\Delta_t\|_W)^2 \qquad \exists C \in \mathbb{R}^+$$

   holds.

Here $P_t := \{\Delta_t, \Delta_{t-1}, \dots, F_{t-1}, F_{t-2}, \dots, \alpha_{t-1}, \alpha_{t-2}, \dots, \beta_{t-1}, \beta_{t-2}, \dots\}$ *denotes the past in iteration* n. *The values* $F_t(s)$, α_t, *and* β_t *may depend on the past* P_t *as long as the assumptions are satisfied. Furthermore,* $\|\Delta_t(s)\|_W$ *denotes the weighted maximum norm* $\|\Delta_t\|_W := \max_s |\Delta_t(s)/W(s)|$.

Proof. The proof uses the three lemmata above. The stochastic process $\Delta_t(s)$ can be decomposed into two processes

$$\Delta_t(s) = \delta_t(s) + w_t(s)$$

by defining

$$\begin{aligned} r_t(s) &:= F_t(s) - \mathbb{E}[F_t(s) \mid P_t], \\ \delta_{t+1}(s) &:= (1-\alpha_t(s))\delta_t(s) + \beta_t(s)\mathbb{E}[F_t(s) \mid P_t], \\ w_{t+1}(s) &:= (1-\alpha_t(s))w_n(s) + \beta_t(s)r_t(s). \end{aligned}$$

□

When the last theorem is applied, the stochastic process Δ_t is usually the difference between the iterates generated by an algorithm and an optimal value such as the optimal action-value function characterized by the Bellman optimality equation.

The following is [43, Lemma 1].

Lemma 6.13 (convergence of a stochastic process)**.** *Let $\langle \mathcal{F}_t \rangle_{t\in\mathbb{N}}$ be an increasing sequence of σ-fields. For all $t \in \mathbb{N}$, let α_t, W_t, and B_t be $\mathcal{F}_t$-measurable scalar random variables. Suppose that the following hold with probability one:*

1. $\mathbb{E}[W_t|\mathcal{F}_t] = 0$,
2. $\mathbb{E}[W_t^2|\mathcal{F}_t] \le B_t$,
3. *the sequence $\langle B_t \rangle_{t\in\mathbb{N}}$ is bounded,*
4. $\alpha_t \in [0,1]$,
5. $\sum_{t\in\mathbb{N}} \alpha_t = \infty$,
6. $\sum_{t\in\mathbb{N}} \alpha_t^2 < \infty$.

Let X_t satisfy the recursion

$$X_{t+1} = (1-\alpha_t)X_t + \alpha_t W_t.$$

Then the iterates X_t converge to zero with probability one, i.e.,

$$\mathbb{P}\Big[\lim_{t\to\infty} X_t = 0\Big] = 1.$$

The first application of the last theorem is the convergence of Q-learning.

Theorem 6.14 (convergence of Q-learning)**.** *The Q-learning iterates*

$$Q_{t+1}(s,a) := (1-\alpha_t(s,a))Q_t(s_t,a_t) + \alpha_t(s,a)(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a)), \quad (6.3)$$

where $\alpha_t \colon \mathcal{S}\times\mathcal{A} \to [0,1)$, converge to the optimal action-value function q^* if the following assumptions hold.

1. *The state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite.*
2. *The equality $\sum_t \alpha_t(s,a) = \infty$ and the inequality $\sum_t \alpha_t(s,a)^2 < \infty$ hold for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.*
3. *The variance $\mathbb{V}[r_t]$ of the rewards is bounded.*
4. *In the case $\gamma = 1$, all episodes must almost surely terminate in a terminal state with zero reward.*

Remark 6.15. The difference between the two Q-learning iterations (6.1) and (6.3) is in the step sizes α_t. In the first form (6.1), the step size α_t is a real number and it is clear that Q_t and Q_{t+1} differ only in their values for a single argument pair (s,a). In the second form (6.3), the step size α_t is a function of s and a. The assumption $\sum_t \alpha_t(s,a) = \infty$ (and the bound $0 \le \alpha_t(s,a) < 1$ for all s and a) again ensures that each pair (s,a) is visited infinitely often (as in Theorem 6.1).

Proof. The basic idea in applying Theorem 6.12, where the stochastic process converges to zero, is to consider the difference between the stochastic process calculated by the algorithm and the supposed limit value, which is characterized here by the Bellman optimality equation.

We start by defining the operator $K \colon (\mathcal{S} \times \mathcal{A} \to \mathbb{R}) \to \mathbb{R}$ as

$$(Kq)(s,a) := \sum_{s'} p(s' \mid s,a)(r(s,a,s') + \gamma \max_{a'} q(s',a')).$$

Recall that the expected reward is given by

$$r(s,a,s') = \mathbb{E}[R_t \mid S_{t-1} = s,\ A_{t-1} = a,\ S_t = s'] = \sum_{r \in \mathcal{R}} \frac{p(s', r \mid s,a)}{p(s' \mid s,a)}$$

(see (2.3)). To show that K is a contraction with respect to the maximum norm (with respect to both s and a), we calculate

$$\begin{aligned}
\|Kq_1 - Kq_2\|_\infty &= \max_{s,\,a} \Big| \sum_{s'} p(s' \mid s,a)(r(s,a,s') + \gamma \max_{a'} q_1(s',a') - r(s,a,s') - \gamma \max_{a'} q_2 \\
&\le \gamma \max_{s,\,a} \sum_{s'} p(s' \mid s,a) \Big| \max_{a'} q_1(s',a') - \max_{a'} q_2(s',a') \Big| \\
&\le \gamma \max_{s,\,a} \sum_{s'} p(s' \mid s,a) \max_{s'',\,a'} |q_1(s'',a') - q_2(s'',a')| \\
&= \gamma \max_{s,\,a} \sum_{s'} p(s' \mid s,a) \|q_1 - q_2\|_\infty
\end{aligned}$$

$$= \gamma \|q_1 - q_2\|_\infty.$$

In summary,

$$\|Kq_1 - Kq_2\|_\infty \leq \gamma \|q_1 - q_2\|_\infty. \tag{6.4}$$

The Bellman optimality equation (2.8) for q^* implies that q^* is a fixed point of K, i.e.,

$$Kq^* = q^*. \tag{6.5}$$

In order to relate the iteration (6.3) to the stochastic process in Theorem 6.12, we define

$$\begin{aligned} \beta_t &:= \alpha_t, \\ \Delta_t(s,a) &:= Q_t(s,a) - q^*(s,a), \\ F_t(s,a) &:= r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - q^*(s,a). \end{aligned}$$

With these definitions, the Q-learning iteration (6.3) and the stochastic process (6.2) are identical.

The expected value of $F_t(s,a)$ given the past P_t as it appears in Theorem 6.12 is

$$\begin{aligned} \mathbb{E}[F_t(s,a) \mid P_t] &= \sum_{s'} p(s', r_{t+1} \mid s, a)(r_{t+1} + \gamma \max_a Q_t(s', a) - q^*(s,a)) \\ &= (KQ_t)(s,a) - q^*(s,a) \\ &= (KQ_t)(s,a) - (Kq^*)(s,a) \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}, \end{aligned}$$

where the last equation follows from (6.5).

Using (6.4), this yields

$$\|\mathbb{E}[F_t(s,a) \mid P_t]\|_\infty \leq \gamma \|Q_t - q^*\|_\infty = \gamma \|\Delta_t\|_\infty,$$

which means that the third assumption of Theorem 6.12 is satisfied if $\gamma < 1$.

In order to check the fourth assumption Theorem 6.12, we calculate

$$\begin{aligned} \mathbb{V}[F_t(s,a) \mid P_t] &= \mathbb{E}\big[\big(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - q^*(s,a) - ((KQ_t)(s,a) - q^*(s,a))\big)^2\big] \\ &= \mathbb{E}\big[\big(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - (KQ_t)(s,a)\big)^2\big] \\ &= \mathbb{V}[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) \mid P_t]. \end{aligned}$$

Since the variance $\mathbb{V}[r_t]$ of the rewards is bounded by the third assumption, we hence find

$$\mathbb{V}[F_t(s,a) \mid P_t] \leq C(1 + \|\Delta_t\|_W)^2 \qquad \exists C \in \mathbb{R}^+.$$

In the case $\gamma = 1$, the usual assumptions that ensure that all episodes are finite are necessary.

In summary, all assumptions of Theorem 6.12 are satisfied. □

The second application of Theorem 6.12 is the convergence of TD(λ).

Theorem 6.16 (convergence of TD(λ))**.** *Suppose that the lengths of the episodes are finite, that there is no inaccessible state in the distribution of the starting states, that the reward distribution has finite variance, that the step sizes satisfy* $\sum_t \alpha_t(s) = \infty$ *and* $\sum_t \alpha_t(s)^2 < \infty$*, and that* $\gamma\lambda < 1$ *holds for* $\gamma \in [0, 1]$ *and* $\lambda \in [0, 1]$*. Then the iterates* V_t *in the* TD(λ) *algorithm almost surely converge to the optimal prediction* v^**.*

6.4 Bibliographical and Historical Remarks

Q-learning was introduced in [39, page 95] and demonstrated at the example of a route-finding problem and a Skinner box. A first convergence proof for one-step Q-learning was given there as well [39, Appendix 1]. An extended, more detailed version of this proof was given in [40]. Convergence proofs based on stochastic processes and stochastic approximation were given in [42], [43], and [36, Section 5.6]. Further references [44, 45, 46].

6.5 Exercises

Chapter 7

On-Policy Prediction with Approximation

7.1 Introduction

Starting in this chapter, we consider the case of infinite state sets $\mathcal{S}$. Then it is obviously not possible anymore to store the state-value function (or the policy) in tabular form, but instead we have to approximate it. We write

$$\hat{v}_w(s) \approx v_\pi(s)$$

for the approximation $\hat{v}_w$ of the state-value function v_π based on the a weight vector $w \in W \subset \mathbb{R}^d$. Typically, the dimension of the weight vector is much smaller than the dimension of the state space, i.e.,

$$d \ll \dim \mathcal{S}.$$

This implies that change one element of the weight vector changes the estimated value of many states, resulting in *generalization*. This effect has advantages and disadvantages: it may make learning more efficient, but it also may make it more difficult if the type of approximation used does not support the kind of value functions required by the problem well, i.e., when the kind of approximation prevents generalization.

So far, in the tabular methods we have discussed for finite state spaces, the estimate for the value of a state was updated to be closer to a certain target value and all other estimates remained unchanged. Now, when function approximation is employed, updating the estimate of a state may affect the estimate values of many other states as well.

In principle, any kind of function approximation can be used. Whenever a new sample, i.e., a state (and action) and an estimate of its value becomes available, the function approximation can be updated globally. Of course, approximation methods that support such online updates are especially suitable.

7.2 Stochastic Gradient and Semi-Gradient Methods

Before we can start to calculate the weights in the approximations of the value functions, we must specify the optimization objective or the error we seek to minimize. One of the most popular errors is the *mean squared value error*

$$\overline{\text{VE}}(w) := \sum_{s \in \mathcal{S}} \mu_\pi(s)(v_\pi(s) - \hat{v}_w(s))^2,$$

where μ_π is the discounted state distribution (see Definition 8.1). The discounted state distribution acts as weight for the squared differences in the true and approximated values. Of course, other weights can be used whenever it makes sense to assign different importance to the states.

The goal is to find a global optimum w_*, i.e., a weight vector w_* such that $\overline{\text{VE}}(w_*) \leq \overline{\text{VE}}(w)$ for all $w \in W \subset \mathbb{R}^d$. It is sometimes possible to show that a global optimum is found when linear function approximations are used, but it becomes much harder in the case of nonlinear function approximation.

Short of finding a global optimum, the goal is to find a local optimum, i.e., a weight vector w_* such that $\overline{\text{VE}}(w_*) \leq \overline{\text{VE}}(w)$ holds for all w in a neighborhood of w_*.

The most popular method for function approximations is stochastic gradient descent (SGD), and it is very well suited for online learning. In SGD, it is assumed that the approximate value function $\hat{v}_w$ is a differentiable function of the weight vector w. The weight vector calculated in each iteration is denoted by w_t for $t \in \{0, 1, 2, ...\}$. We assume for now that in each iteration a new sample $v_\pi(S_t)$ becomes available having reached state S_t. SGD means improving the weight vector w_t by moving it slightly downhill with respect to the error $\overline{\text{VE}}$ in the direction of the greatest change in the error at w_t. This direction of greatest change is the gradient, and minimizing the error means adding a small multiple of the negative gradient. This results in the iteration

$$w_{t+1} := w_t - \frac{1}{2}\alpha_t \nabla_w (v_\pi(S_t) - \hat{v}_{w_t}(S_t))^2 \tag{7.1a}$$

$$= w_t + \alpha_t (v_\pi(S_t) - \hat{v}_{w_t}(S_t)) \nabla_w \hat{v}_{w_t}(S_t), \tag{7.1b}$$

where the learning rate $\alpha_t \in \mathbb{R}^+$. The sole purpose of the factor $1/2$ in the first line is to not have a factor 2 in the second line.

SGD is a *stochastic* gradient-descent method since the update is a random variable because it depends on the random variable S_t. Over many samples or iterations, the accumulated effect is to minimize the average of an objective function such as the mean squared value error. To ensure convergence, the learning rate α_t must tend to zero.

Unfortunately, the true value $v_\pi(S_t)$ is unknown, since v_π is to be calculated. Therefore, in fact, we can only use a random variable U_t instead of $v_\pi(S_t)$ in the iteration. Hence the general SGD method for the prediction of state values is the iteration

$$w_{t+1} := w_t + \alpha_t(U_t - \hat{v}_{w_t}(S_t))\nabla_w \hat{v}_{w_t}(S_t). \tag{7.2}$$

If U_t is an unbiased estimate of $v_\pi(S_t)$, i.e., if

$$\mathbb{E}[U_t \mid S_t = s] = v_\pi(S_t)$$

for all times t and if the learning rate α satisfy the conditions (2.2) for stochastic approximation, then the w_t converge to a local optimum.

Equipped with the SGD method, we can now develop algorithms for calculating w_* based on different choices for the target value U_t.

Probably the most obvious choice for an unbiased estimate of $v_\pi(S_t)$ is the Monte-Carlo target

$$U_t := G_t.$$

Based on the convergence results just mentioned, the general SGD method in conjunction with this estimate converges to a locally optimal approximation of $v_\pi(S_t)$. In other words, the algorithm for the Monte-Carlo state-value prediction can be shown to always find a locally optimal solution. The resulting algorithm is shown in Algorithm 14. Note that the episode must have ended so that G_t can be calculated in each time step.

Bootstrapping targets such as the n-step return $G_{t:t+n}$, which build on previously calculated approximations, do not provide the same convergence guarantees. By the definition of bootstrapping, the target U_t in a bootstrapping method depends on the current weight vector w_t, which means that the estimate is biased.

Bootstrapping methods are not even gradient-descent methods. This becomes clear by considering the derivative calculated in (7.1). While the derivative of $\hat{v}_{w_t}(S_t)$ appears in the equation, in a bootstrapping method the derivative of $U_t(w_t) \approx v_\pi(S_t)$ is nonzero, but does not appear in the equation. Because of this missing term, these methods are called *semigradient* methods.

Algorithm 14 gradient MC prediction for calculating $\hat{v}_w \approx v_\pi$ given the policy π.

initialization:
choose a representation for the state-value function $\hat{v}_w$
choose learning rate $\alpha_t \in \mathbb{R}^+$
initialize state-value parameter $w \in W \subset \mathbb{R}^d$

loop ▷ for all episodes
  generate an episode $(S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T)$ following π
  for $t \in (0, 1, \dots, T-1)$ **do** ▷ for all time steps
    $w := w + \alpha_t(G_t - \hat{v}_w(S_t))\nabla\hat{v}_w(S_t)$
  end for
end loop

On the other hand, semigradient methods often learn significantly faster and they converge reliably in the important case of linear function approximation. Additionally, they enable online learning in contrast to MC methods, which have to wait till the end of an episode.

The most straightforward semigradient method is probably the semigradient TD(0) method, which uses the target

$$U_t := R_{t+1} + \gamma\hat{v}_{w_t}(S_{t+1}).$$

The resulting algorithm is shown in Algorithm 15.

7.3 Linear Function Approximation

One of the most important cases when approximating the state-value function is the case of linear function approximation. Then the state-value function v_π is approximated by

$$v_\pi(s) \approx \hat{v}_w(s) := w^\top x(s) = \sum_{k=1}^{d} w_i x_i(s). \tag{7.3}$$

The vector valued function

$$x\colon \mathcal{S} \to \mathbb{R}^d, \qquad x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_d(s) \end{pmatrix}$$

gives the feature vectors or simply features, whose expedient choice is crucial.

Algorithm 15 semigradient TD(0) prediction for calculating $\hat{v}_w \approx v_\pi$ given the policy π.

initialization:
choose a representation for the state-value function $\hat{v}_w$
choose learning rate $\alpha_t \in \mathbb{R}^+$
initialize state-value parameter $w \in W \subset \mathbb{R}^d$

loop ▷ for all episodes
  initialize s
  repeat ▷ for all time steps
    choose action a using π
    take action a and receive the new state s' and the reward r
    $w := w + \alpha_t(r + \gamma\hat{v}_w(s') - \hat{v}_w(s))\nabla\hat{v}_w(s)$
    $s := s'$
  until s is the terminal state and the episode is finished
end loop

In other words, the features are the basis functions that span the subspace of all approximations of v_w. Unfortunately, the subspace is often rather small, i.e., $d \ll \dim \mathcal{S}$, due to problem size or computational limitations. Obviously, the choice of the subspace (which is equivalent to the choice of the features) is instrumental in being able to calculate good approximations to v_π at all.

When using linear approximations, the SGD iteration simplifies. The gradient of the approximated state-value function is just the feature function, i.e., $\nabla_w\hat{v}_w(s) = x(s)$. Hence the update (7.2) becomes

$$w_{t+1} := w_t + \alpha_t(U_t - \hat{v}_{w_t}(S_t))x(S_t).$$

Almost all convergence results that are known are for the case of linear function approximation. In the linear case, there is only one global optimum or – more precisely – a set of equally good optima. This means that convergence to a local optimum implies global convergence.

For example, the gradient MC algorithm Algorithm 14 when combined with linear function approximation converges to a local minimum of the mean squared value error if the learning rate satisfies the usual conditions (2.2).

The semigradient algorithm Algorithm 15 also converges, but in general to a different limit, and this fact does not follow from the considerations above.

Assuming that the algorithm converges, we can find the limit using the

following consideration. Using the notation $x_t := x(S_t)$, the iteration is

$$\begin{aligned} w_{t+1} &:= w_t + \alpha_t(R_{t+1} + \gamma w_t^\top x_{t+1} - w_t^\top x_t)x_t \\ &= w_t + \alpha_t(R_{t+1}x_t - x_t(x_t - \gamma x_{t+1})^\top w_t). \end{aligned}$$

Applying the expected value to both sides, we find

$$\begin{aligned} \mathbb{E}[w_{t+1} \mid w_t] &= w_t + \alpha_t(b - Aw_t), \\ A &:= \mathbb{E}[x_t(x_t - \gamma x_{t+1})^\top] \in \mathbb{R}^{d\times d}, \\ b &:= \mathbb{E}[R_{t+1}x_t] \in \mathbb{R}^d. \end{aligned}$$

Hence, if the w_t converge, the equation $Aw_t = b$ must hold, which means that the possible fixed point is

$$w_{TD} := A^{-1}b, \tag{7.4}$$

which is called the TD fixed point.

Theorem 7.1. *Suppose that $\gamma < 1$, that $\langle\alpha_t\rangle_{t\in\mathbb{N}}$ satisfies (2.2), that the feature vectors $x(s)$ are a basis of $\mathbb{R}^d$, and that the state distribution is positive for all states.*

Then the semigradient algorithm Algorithm 15 with the linear approximation (7.3) converges to the TD fixed point w_{TD} defined in (7.4) with probability one.

Sketch of the proof. The calculation above shows that

$$\mathbb{E}[w_{t+1} \mid w_t] = (I - \alpha_t A)w_t + \alpha b.$$

Therefore we define the function

$$K\colon \mathbb{R}^d \to \mathbb{R}^d, \qquad K(w) := (I - \alpha A)w + \alpha b.$$

In order to be able to use the Banach fixed-point theorem, we will show that K is a contraction for sufficiently small $\alpha \in \mathbb{R}^+$. It is a contraction if

$$\|K(v_2) - K(v_1)\| \le \|I - \alpha A\|\|v_2 - v_1\| \le \kappa\|v_2 - v_1\| \qquad \exists\kappa \in \mathbb{R}^+.$$

Therefore it suffices to show that A is positive definite, i.e., $v^\top Av > 0$ for all $v \in \mathbb{R}^d \setminus \{0\}$.

We can write the matrix A as

$$\begin{aligned} A &= \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a)x(s)(x(s) - \gamma x(s'))^\top \\ &= \sum_{s\in\mathcal{S}} \mu(s) \sum_{s'\in\mathcal{S}} p(s'|s)x(s)(x(s) - \gamma x(s'))^\top \end{aligned}$$

$$
\begin{aligned}
&= \sum_{s \in \mathcal{S}} \mu(s) x(s) \big(x(s) - \gamma \sum_{s' \in \mathcal{S}} p(s'|s) x(s')\big)^\top \\
&= X^\top D(I - \gamma P) X,
\end{aligned}
$$

where μ_π is the state distribution under π, D is the diagonal matrix with the entries $\mu(s)$, P is the $\dim \mathcal{S} \times \dim \mathcal{S}$ matrix of the transition probabilities $p(s'|s)$ from state s to state s' under π, and X is the $\dim \mathcal{S} \times d$ matrix that contains the $x(s)$ as rows.

Since X is a basis change, it suffices to show that $D(I - \gamma P)$ is positive definite. It can shown that it suffices to show that all of its column sums and all of its row sums are positive.

The row sums of $D(I - \gamma P)$ are all positive, since P is a matrix of probabilities and $\gamma < 1$.

To show that the columns sums of $D(I - \gamma P)$ are all positive, we calculate them as

$$
\begin{aligned}
\mathbf{1}^\top D(1 - \gamma P) &= \mu^\top (I - \gamma) P \\
&= \mu^\top - \gamma \mu^\top P \\
&= \mu^\top - \gamma \mu^\top \\
&= (1 - \gamma) \mu^\top.
\end{aligned}
$$

Each entry of this vector is positive, since the state distribution is positive everywhere by assumption.

□

Since the optimal weight vector w_* and the TD fixed point w_{TD} are different in general, the question arises how close they are. The following theorem means that the mean square value error of the TD fixed point is always within a factor of $1/(1 - \gamma)$ of the lowest possible error.

Theorem 7.2. *The TD fixed point w_{TD} in (7.4) satisfies the inequality*

$$
\overline{\mathrm{VE}}(w_{TD}) \leq \frac{1}{1 - \gamma} \min_{w \in \mathbb{R}^d} \overline{\mathrm{VE}}(w).
$$

7.4 Features for Linear Methods

7.4.1 Polynomials

But consider Weierstrass approximation theorem.

7.4.2 Fourier Basis

ith feature:

$$x_i(s) := \cos(\pi s^\top c^i),$$

where c^i is a constant vector.

7.4.3 Coarse Coding

Cover state space with circles. A feature has value 1 (or is said to be present), if it inside the corresponding circle. Otherwise it has value 0 (and is said to be absent).

7.4.4 Tile Coding

Cover state space with tiles. First construct tiling (a partition), the shift the tilings.

7.4.5 Radial Basis Functions

The features are

$$x_i(s) := \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right),$$

where c_i is called the center state and σ_i is called the feature width.

7.5 Nonlinear Function Approximation

7.5.1 Artificial Neural Networks

See bonus chapter.

7.5.2 Memory Based Function Approximation

Save training samples in memory, use a set of samples to compute the value estimate only when required. Also called lazy learning.

7.5.3 Kernel-Based Function Approximation

We denote the set of all stored examples by E. Then the state-value function is approximated as

$$\hat{v}_E(s) := \sum_{s' \in E} k(s, s') g(s'),$$

where $g(s')$ is the target for state s' and $k \colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a kernel function that assigns a weight to the known data about state s' when asked about state s.

7.6 Bibliographical and Historical Remarks

[47]

Problems

Chapter 8

Policy-Gradient Methods

8.1 Introduction

A large class of reinforcement-learning methods are action-value methods, i.e., methods that calculate action values and then choose the best action based on these action values. On the other hand, we present methods for calculating policies more directly in this chapter. The policies here are parameterized, i.e., we write them in the form

$$\pi \colon \mathcal{A}(s) \times \mathcal{S} \times \Theta \to [0,1],$$
$$\pi_\theta(a \mid s) := \pi(a \mid s, \theta) := \mathbb{P}\{A_t = a \mid S_t = s,\ \theta_t = \theta\},$$

where the parameter $\theta \in \Theta \subset \mathbb{R}^{d'}$ is a vector. We seek a parameter that corresponds to an optimal policy.

Commonly, the parameters are learned such that a scalar performance measure called

$$J \colon \Theta \to \mathbb{R},$$

defined on the parameters, is maximized. A leading example is the definition

$$J(\theta) := \mathbb{E}[v_{\pi_\theta}(S_0) \mid S_0 \sim \iota],$$

where $S_0 \sim \iota$ is the initial state chosen according to the probability distribution ι.

The general assumption is that the policy π and the performance measure J are differentiable with respect to θ such that gradient based optimization can be employed. Such methods are called *policy-gradient methods*. The performance can be maximized for example by using stochastic gradient ascent with respect to the performance measure J, i.e., we define the iteration

$$\theta_{t+1} := \theta_t + \alpha_t \mathbb{E}[\nabla_\theta J(\theta_t)].$$

The expected value of the gradient of the performance measure J in the last term is usually approximated.

The question whether action-value or policy-gradient methods are preferable depends on the problem. It may be the case that the action-value function has a simpler structure and is therefore easier to learn, or it may be the case that the policy itself has a simpler structure.

8.2 Finite and Infinite Action Sets

8.2.1 Finite Action Sets

If the action sets $\mathcal{A}(s)$ are finite, then a common form of the policy is based on the so-called *preference function*

$$h\colon \mathcal{S} \times \mathcal{A}(s) \times \Theta \to \mathbb{R}.$$

The preferences of state-action pairs (s, a) are translated into probabilities and hence into a policy via the exponential soft-max function

$$\pi(a \mid s, \theta) := \frac{\exp(h(s, a, \theta))}{\sum_{a' \in \mathcal{A}(s)} \exp(h(s, a', \theta))}.$$

Many choices for the representation of the preference functions are possible. Two popular ones are the following.

- The preference function is an artificial neural network. Then the parameter vector $\theta \in \Theta \subset \mathbb{R}^{d'}$ contains all weights and biases of the artificial neural network.

- The preference function has the linear form

  $$h(s, a, \theta) := \theta^\top x(s, a),$$

  where the *feature function*

  $$x\colon \mathcal{S} \times \mathcal{A}(s) \to \mathbb{R}^{d'}$$

  yields the feature vectors.

8.2.2 Infinite Action Sets

If the action sets $\mathcal{A}(s)$ are infinite, it is possible to simplify the problem of learning the probabilities of all actions by reducing it to learning the parameters of a probability distribution. The parameters of the distribution are represented by functions. For example, we define a policy of the form

$$\pi(a \mid s, \theta) := \frac{1}{\sqrt{2\pi}\sigma(s,\theta)} \exp\left(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}\right),$$

where now the two functions $\mu\colon \mathcal{S} \times \Theta \to \mathbb{R}$ and $\sigma\colon \mathcal{S} \times \Theta \to \mathbb{R}^+$ need to be learned. To do that, we split the parameter vector $\theta \in \Theta$ into two vectors θ_μ and θ_σ, i.e., $\theta = (\theta_\mu, \theta_\sigma)^\top$, and write the functions μ and σ as a linear and a positive function

$$\mu(s,\theta) := \theta_\mu^\top x_\mu(s),$$
$$\sigma(s,\theta) := \exp(\theta_\sigma^\top x_\sigma(s)),$$

respectively, where the features x_μ and x_σ are vector valued functions as usual.

Representing policies as the soft-max of preferences for actions makes it possible to approximate deterministic policies, which is not immediately possible when using ϵ-greedy policies. If the optimal policy is deterministic, the preferences of the optimal actions become unbounded, at least if this behavior is allowed by the kind of parameterization used.

Action preferences can represent optimal stochastic policies well in the sense that the probabilities of actions may be arbitrary. This is in contrast to action-value methods and it is an important feature whenever the optimal policy is stochastic such as in rock-paper-scissors and poker.

8.3 The Policy-Gradient Theorem

Before stating the policy-gradient theorem, we define the (discounted) state distribution. In the case of an episodic environment or learning task, we consider discount rates $\gamma < 1$. In the case of a continuing environment or learning task, it can be shown that a discount rate $\gamma < 1$ only results in the factor $1/(1-\gamma)$ in the performance measure and hence we assume that $\gamma = 1$ in this case without loss of generality.

Definition 8.1 (discounted state distribution)**.** The *discounted state distribution*

$$\mu_\pi\colon \mathcal{S} \to [0,1],$$

$$\mu_\pi(s) := \mathbb{E}_{\text{all episodes}}[\lim_{t\to\infty} \mathbb{P}\{S_t = s \mid S_0 \sim \iota,\ A_0, \dots, A_{t-1} \sim \pi\}]$$

is the probability of being in state s in all episodes under a given policy $\pi \in \mathcal{P}$ and a given initial state distribution ι and discounted by γ in the episodic case.

In order to simplify notation, we write μ_π instead of $\mu_{\pi,\iota,\gamma}$.

By definition, $\mu_\pi(s) \geq 0$ for all $s \in \mathcal{S}$ and $\sum_{s\in\mathcal{S}} \mu_\pi(s) = 1$. We discount the state distribution by the discount rate γ, because this is the form that is necessary for the calculations in Theorem 8.3 below. It is also consistent with the appearance of the discount rate in the definitions of the state- and action-value functions, which are also used in the calculations in the theorem.

If the environment or learning task is continuing, the state distribution is just the stationary distribution under the policy π. If the environment or learning task is episodic, however, the distribution of the initial distribution of the states plays a role as seen in the following lemma, which gives the state distribution in both cases.

Lemma 8.2 (discounted state distribution)**.** *If the environment is continuing, the* state distribution *is the stationary distribution under the policy π.*

If the environment is episodic, the discounted state distribution *is given by*

$$\mu_\pi(s) = \frac{\eta(s)}{\sum_{s'\in\mathcal{S}} \eta(s')} \qquad \forall s \in \mathcal{S},$$

where the $\eta(s)$ are the solution of the system of equations

$$\eta(s) = \iota(s) + \gamma \sum_{s'\in\mathcal{S}} \eta(s') \sum_{a\in\mathcal{A}(s)} \pi(a|s') p(s \mid s', a) \qquad \forall s \in \mathcal{S},$$

where $\iota\colon \mathcal{S} \to [0,1]$ is the initial distribution of the states in an episode.

To simplify notation, we write η instead of η_π or $\eta_{\pi,\iota,\gamma}$.

Proof. We start by noting that $\eta(s)$ is the average number of time steps spent in state s over all episodes discounted by γ. It consists of two terms, namely the probability $\iota(s)$ to start in state s and the discounted average number of times the state s occurs coming from all other states $s' \in \mathcal{S}$.

This linear system of equations has a unique solution, yielding the $\eta(s)$. In order to find the state distribution $\mu_\pi(s)$, the $\eta(s)$ must be scaled by the mean length

$$L := \sum_{s\in\mathcal{S}} \eta(s) \tag{8.1}$$

of all episodes. □

The following theorem [48] is fundamental for policy-gradient methods. In the continuing case, the performance measure is

$$r(\pi_\theta) := \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0 \sim \iota,\ A_0, \dots, A_{t-1} \sim \pi_\theta],$$

which is assumed to exist and to be independent of the initial state distribution ι, i.e., the stochastic process defined by the policy π_θ and the transition probability p is assumed to be ergodic.

Theorem 8.3 (policy-gradient theorem)**.** *Suppose that the performance measure is defined as*

$$J(\theta) := \begin{cases} \mathbb{E}[v_{\pi_\theta}(S_0) \mid S_0 \sim \iota], & \textit{episodic environment,} \\ r(\pi_\theta), & \textit{continuing environment,} \end{cases}$$

where $S_0 \sim \iota$ is the initial state chosen according to the probability distribution ι. Then its gradient is given by

$$\nabla_\theta J(\theta) = L \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a \mid s, \theta),$$

where

$$L := \begin{cases} \textit{mean episode length}, & \textit{episodic environment,} \\ 1, & \textit{continuing environment} \end{cases}$$

and μ is the state distribution.

Proof. We prove the episodic case first and start by differentiating the state-value function v_{π_θ}. By the definitions of the value functions v and q, we have

$$\nabla_\theta v_{\pi_\theta}(s) = \nabla_\theta \Big(\sum_{a\in\mathcal{A}(s)} \pi(a \mid s, \theta) q_{\pi_\theta}(s,a) \Big) \qquad \forall s \in \mathcal{S}.$$

By (2.10), we find

$$\begin{aligned} \nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(& \nabla_\theta \pi(a \mid s, \theta) q_{\pi_\theta}(s,a) \\ & + \pi(a \mid s, \theta) \nabla_\theta \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r \mid s, a)(r + \gamma v_{\pi_\theta}(s')) \Big) \qquad \forall s \in \mathcal{S}, \end{aligned}$$

which simplifies to

$$\nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(\nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) + \gamma\pi(a \mid s,\theta) \sum_{s'\in\mathcal{S}} p(s', r \mid s, a) \nabla_\theta v_{\pi_\theta}(s')\Big) \qquad \forall s \in \mathcal{S}.$$

Performing a second time step by applying the recursive formula we just found to $\nabla_\theta v_{\pi_\theta}(s')$, we find

$$\nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(\nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) + \gamma\pi(a \mid s,\theta) \sum_{s'\in\mathcal{S}} p(s', r \mid s, a) \sum_{a'\in\mathcal{A}(s)} \Big(\nabla_\theta \pi(a' \mid s',\theta) q_{\pi_\theta}(s',a') + \gamma\pi(a' \mid s',\theta) \sum_{s''\in\mathcal{S}} p(s'', r \mid s', a') \nabla_\theta v_{\pi_\theta}(s'')\Big)\Big) \qquad \forall s \in \mathcal{S}.$$

Hence the recursive expansion of this formula can be written as

$$\nabla_\theta v_{\pi_\theta}(s) = \sum_{s'\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}\{s' \mid s, k, \pi\} \sum_{a\in\mathcal{A}(s')} \nabla_\theta \pi(a \mid s',\theta) q_{\pi_\theta}(s',a),$$

where $\mathbb{P}\{s' \mid s, k, \pi\}$ is the probability of transitioning to state s' after following policy π for k steps after starting from state s.

The gradient of the performance measure J thus becomes

$$\begin{aligned}
\nabla_\theta J(\theta) &:= \nabla_\theta \mathbb{E}[v_{\pi_\theta}(S_0) \mid S_0 \sim \iota] \\
&= \mathbb{E}\Big[\sum_{s\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}\{s \mid S_0, k, \pi\} \sum_{a\in\mathcal{A}(s)} \nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \mid S_0 \sim \iota\Big] \\
&= \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}(s)} \nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \\
&= L \sum_{s\in\mathcal{S}} \frac{\eta(s)}{L} \sum_{a\in\mathcal{A}(s)} \nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \\
&= L \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} \nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a),
\end{aligned}$$

where we have used the definition of $\eta(s)$ in Lemma 8.2 and (8.1).

In the continuing case, similar calculations for the differential return

$$G_t := \sum_{k=1}^{\infty} (R_{t+k} - r(\pi_\theta))$$

can be done.

□

Interestingly enough, although the performance measure depends on the state distribution which depends on the policy parameter θ, the derivative of the state distribution does not appear in the expression found in the policy-gradient theorem. This is the usefulness of the theorem.

Remark 8.4 (experience replay). The expression for the gradient of the performance measure found in the theorem includes the state distribution μ and hence motivates experience replay. Experience replay is a method which keeps a cache of states and actions that have been visited and which are replayed during learning in order to ensure that the whole space is sampled in an equidistributed manner. Cf. Remark 6.8.

8.4 Monte-Carlo Policy-Gradient Method: REINFORCE

Having the gradient of the performance measure available from Theorem 8.3, we can use gradient based stochastic optimization. The most straightforward way is the iteration

$$\theta_{t+1} := \theta_t + \alpha \sum_{a \in \mathcal{A}(S_t)} \hat{q}_w(S_t, a) \nabla_\theta \pi_\theta(a \mid S_t, \theta),$$

where $\hat{q}_w$ is an approximation of q_{π_θ} and parameterized by a vector $w \in \mathbb{R}^d$. This iteration is called an all-actions method.

The more classical REINFORCE algorithm is derived as follows. We start from state S_t in time step t and use Theorem 8.3 to write

$$\begin{aligned} \nabla_\theta J(\theta) &= L\mathbb{E}_{\pi_\theta}\Big[\gamma^t \sum_{a \in \mathcal{A}(S_t)} q_{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a \mid s, \theta)\Big] \\ &= L\mathbb{E}_{\pi_\theta}\Big[\gamma^t \sum_{a \in \mathcal{A}(S_t)} \pi_\theta(a \mid S_t, \theta) q_{\pi_\theta}(S_t, a) \frac{\nabla_\theta \pi_\theta(a \mid S_t, \theta)}{\pi_\theta(a \mid S_t, \theta)}\Big], \end{aligned}$$

since the discounted state distribution μ_{π_θ} includes a factor of γ for each time step. Next, we replace the sum over all actions by the sample $A_t \sim \pi_\theta$. Then the gradient of the performance measure is approximately proportional to

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\Big[\gamma^t q_{\pi_\theta}(S_t, A_t) \frac{\nabla_\theta \pi_\theta(A_t \mid S_t, \theta)}{\pi_\theta(A_t \mid S_t, \theta)}\Big].$$

Having selected the action A_t, we use Definition 2.8 to find

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\gamma^t G_t \frac{\nabla_\theta \pi_\theta(A_t \mid S_t, \theta)}{\pi_\theta(A_t \mid S_t, \theta)}\right].$$

This yields the gradient used in the REINFORCE update

$$\begin{aligned}\theta_{t+1} &:= \theta_t + \alpha\gamma^t G_t \frac{\nabla_\theta \pi_\theta(A_t \mid S_t, \theta_t)}{\pi_\theta(A_t \mid S_t, \theta_t)} \\ &= \theta_t + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta_t).\end{aligned}$$

Since the return G_t until the end of an episode is used as the target, this is an MC method.

The algorithm for this update is shown in Algorithm 16.

Algorithm 16 REINFORCE for calculating $\pi_\theta \approx \pi_*$.

initialization:
choose a representation the policy π_θ
choose learning rate $\alpha \in \mathbb{R}^+$
initialize policy parameter $\theta \in \Theta \subset \mathbb{R}^{d'}$

loop ▷ for all episodes
  generate an episode $(S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T)$ following π_θ
  for $t \in (0, 1, \ldots, T-1)$ **do** ▷ for all time steps
    $G := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta := \theta + \alpha\gamma^t G \nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta)$
  end for
end loop

return θ

REINFORCE is a stochastic gradient ascent method. By the construction based on Theorem 8.3, the expected update over time is in the same direction as the performance measure J. Under the standard stochastic approximation conditions (2.2), the algorithm converges to a local optimum. On the other hand, REINFORCE is a MC algorithm and thus may be of high variance.

8.5 Monte-Carlo Policy-Gradient Method: REINFORCE with Baseline

The right side in Theorem 8.3 can be changed by subtracting an arbitrary so-called baseline b, a function or a random variable of the state, from the action-value function, i.e.,

$$\begin{aligned}\nabla_\theta J(\theta) &= L \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} q_{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a \mid s,\theta) \\ &= L \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} (q_{\pi_\theta}(s,a) - b(s))\nabla_\theta \pi_\theta(a \mid s,\theta).\end{aligned}$$

The last equation holds true because

$$\sum_{a\in\mathcal{A}(s)} b(s)\nabla_\theta \pi_\theta(a \mid s,\theta) = b(s)\nabla_\theta \underbrace{\sum_{a\in\mathcal{A}(s)} \pi_\theta(a \mid s,\theta)}_{=1} = 0.$$

With this change, the update becomes

$$\theta_{t+1} := \theta_t + \alpha\gamma^t (G_t - b(S_t))\nabla_\theta \ln \pi_\theta(A_t \mid S_t,\theta_t).$$

What is the purpose of adding a baseline? It leaves the expected value of the updates unchanged, but it is a method to reduce their variance. The natural choice is an approximation of the expected value of G_t, i.e., the state-value function. This approximation

$$\hat{v}_w(s) \approx v_{\pi_\theta}(s)$$

of the state-value function $v_{\pi_\theta}(s)$, where $w \in W \subset \mathbb{R}^d$ is a parameter vector, can be calculated by any suitable method, but since REINFORCE is an MC method, an MC method is used for calculating the approximation in Algorithm 17 as well.

The general rule of thumb for choosing the learning rate α_w is

$$\alpha_w := \frac{0.1}{\mathbb{E}[\|\nabla_w \hat{v}_w(S_t)\|_\mu^2]},$$

which is updated while the algorithm runs. Unfortunately, no such general rule is available for the learning rate α_θ, since the learning rate depends on the range of the rewards and on the parameterization of the policy.

REINFORCE with baselines is unbiased and its approximation of an optimal policy converges to a local minimum. As an MC method, it converges slowly with high variance and inconvenient to implement for continuing environments or learning tasks.

Algorithm 17 REINFORCE with baseline for calculating $\pi_\theta \approx \pi_*$.

initialization:
choose a representation for the policy π_θ
choose a representation for the state-value function $\hat{v}_w$
choose learning rate $\alpha_\theta \in \mathbb{R}^+$
choose learning rate $\alpha_w \in \mathbb{R}^+$
initialize policy parameter $\theta \in \Theta \subset \mathbb{R}^{d'}$
initialize state-value parameter $w \in W \subset \mathbb{R}^d$

loop ▷ for all episodes
  generate an episode $(S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T)$ following π_θ
  for $t \in (0, 1, \dots, T-1)$ **do** ▷ for all time steps
    $G := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta := G - \hat{v}_w(S_t)$
    $w := w + \alpha_w \delta \nabla_w \hat{v}_w(S_t)$
    $\theta := \theta + \alpha_\theta \gamma^t \delta \nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta)$
  end for
end loop

return θ and w

8.6 Temporal-Difference Policy-Gradient Methods: Actor-Critic Methods

REINFORCE with baseline is an MC method, since the target value in the update is the return till the end of the episode. In other words, no bootstrapping is performed, i.e., no previous approximation of a value function is used to update the policy. Although the state-value function is used in the iteration, it is used to only for the state that is currently being updated; it serves for variance reduction, not for bootstrapping.

Bootstrapping (e.g., by going from MC to TD methods) introduces a bias. Still, this is often useful as it reduces the variance in the value function or policy and hence accelerates learning. Going from MC methods to TD methods is analogous to going from REINFORCE with baseline to actor-critic methods. Just as TD methods, actor-critic methods use the return calculated over a certain number of time steps; the simplest case being to use the return $G_{t:t+1}$ using only one time step.

Here we introduce the one-step actor-critic method that uses the one-step return $G_{t:t+1}$ as the target in the update and that still uses the learned state-value function $\hat{v}_w$ as the baseline $b := \hat{v}_w$. This yields the iteration

$$\begin{aligned}\theta_{t+1} &:= \theta_t + \alpha\gamma^t(G_t - b(S_t))\nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta_t) \\ &= \theta_t + \alpha\gamma^t(R_t + \gamma\hat{v}_w(S_{t+1}) - \hat{v}_w(S_t))\nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta_t)\end{aligned}$$

The update of the baseline is now performed by TD(0) in order to be consistent in the methods that are used to learn the baseline and the policy.

The name comes from the intuition that the policy is the actor (answering the question what to do) and the baseline, i.e., the state-value function is the critic (answering the question how well it is done).

This one-step actor-critic method is shown in Algorithm 18.

Of course, instead of the one-step return, the n-step return $G_{t:t+n}$ or the λ-return G_t^λ can be used.

8.7 Bibliographical and Historical Remarks

Problems

Algorithm 18 one-step actor critic for calculating $\pi_\theta \approx \pi_*$.

initialization:
choose a representation for the policy π_θ
choose a representation for the state-value function $\hat{v}_w$
choose learning rate $\alpha_\theta \in \mathbb{R}^+$
choose learning rate $\alpha_w \in \mathbb{R}^+$
initialize policy parameter $\theta \in \Theta \subset \mathbb{R}^{d'}$
initialize state-value parameter $w \in W \subset \mathbb{R}^d$

loop ▷ for all episodes
  initialize $s \sim \iota$
  $G := 1$
  while s is not terminal **do** ▷ for all time steps
    choose action a according to $\pi_\theta(\cdot|s)$
    take action a and receive the new state s' and the reward r
    $\delta := R + \gamma \hat{v}_w(s') - \hat{v}_w(s)$
    $w := w + \alpha_w \delta \nabla_w \hat{v}_w(s)$
    $\theta := \theta + \alpha_\theta G \delta \nabla_\theta \ln \pi_\theta(a \mid s, \theta)$
    $G := \gamma G$
    $s := s'$
  end while
end loop

return θ and w

Chapter 9

Hamilton-Jacobi-Bellman Equations

9.1 Introduction

In certain cases, it is possible to model the environment by difference or differential equations. This may be the case when the environment depends on – for example – physical, chemical, or biological processes that can be described by such equations. The ability to describe the environment in such a manner usually has the advantage that the number of episodes available for learning is unlimited, since episodes can be rolled out by solving the equations. This is in contrast to problems in data science, where the available data may be limited. The purpose of this chapter is to take advantage of the knowledge about the environment encoded in the equations.

In this chapter, we study the case when the environment can be described by deterministic or stochastic ordinary differential equations, leading to problems in deterministic and stochastic optimal control. Here we follow the notation for RL problems used in the rest of the book.

We write the *state equation* in the form of the initial-value problem

$$\dot{s}(t) = f(s(t), \pi(t)) \qquad \forall t \in \mathbb{R}_0^+, \tag{9.1a}$$

$$s(0) = s_0 \in \mathcal{S}, \tag{9.1b}$$

where the function $s\colon \mathbb{R}_0^+ \to \mathcal{S}$, whose image is the set $\mathcal{S} \subset \mathbb{R}^{n_s}$ of all states, gives the state of the system at time $t \in \mathbb{R}_0^+$, the function $u\colon \mathbb{R}_0^+ \to \mathcal{A}(t)$, whose image is the set $\mathcal{A}(t) \subset \mathbb{R}^{n_a}$ of all actions available at time t, is the control applied at time $t \in \mathbb{R}_0^+$, and the vector valued function $f\colon \mathbb{R}^{n_s} \times \mathbb{R}^{n_a} \to \mathbb{R}^{n_s}$

describes the dynamics of the system as a deterministic or stochastic ordinary differential equation.

The control $u\colon \mathbb{R}_0^+ \to \mathcal{A}$ and the corresponding policy $\pi\colon \mathcal{S} \to \mathcal{A}$ are related simply by

$$u(t) = \pi(s(t)).$$

We assume that the state set or state space $\mathcal{S}$ is an open, bounded set with a sufficiently smooth boundary. We also assume that the function u is bounded and Lebesgue measurable and that its image is a compact set in $\mathcal{A}(t)$. Furthermore, we assume that the dynamics f of the system are Lipschitz continuous with respect to its first argument $s(t)$.

If the control function u is given, then the initial-value problem (9.1) has a unique solution as known from the standard theory of ordinary differential equations. Unfortunately, the solution may exit the state space $\overline{\mathcal{S}}$ at a certain time

$$\tau := \begin{cases} \infty, & s(t) \in \overline{\mathcal{S}} \quad \forall t \in \mathbb{R}_0^+, \\ \inf\{t \in \mathbb{R}_0^+ \mid s(t) \notin \overline{\mathcal{S}}\}, & \text{otherwise}, \end{cases}$$

the so-called exit time.

Next, we define the (deterministic) return as the functional

$$G\colon \mathcal{S} \times (\mathbb{R}_0^+ \to \mathcal{A}(t)) \to \mathbb{R},$$
$$G(s_0, u) := \int_0^\tau \mathrm{e}^{-\gamma t} r(s(t), u(t)) \mathrm{d}t + \mathrm{e}^{-\gamma \tau} R(s(\tau))$$

on the state space $\mathcal{S}$ and the set of all actions. The function $r\colon \mathcal{S} \times \mathcal{A}(t) \to \mathbb{R}$ is called the current reward, and the function $R\colon \partial\mathcal{S} \to \mathbb{R}$ is called the boundary reward. The discount rate $\gamma \in \mathbb{R}^+$ is constant. The return is the continuously discounted return over all times $[0, \tau]$ the trajectory $\{t \in [0, \tau] \mid s(t)\}$ remains within the set $\overline{\mathcal{S}}$ of admissible states.

The optimal-control problem consists in finding an initial state $s_0 \in \overline{\mathcal{S}}$ and an optimal control u_* that maximizes the return G.

9.2 The Hamilton-Jacobi-Bellman Equation

Similar to the discrete case, we can find an equation that is satisfied by optimal controls; this is the purpose of this section. We start by defining the optimal value function.

Definition 9.1 (optimal value function)**.** The *optimal value function* is defined as

$$v_*\colon \mathcal{S} \to \mathbb{R}, \qquad v_*(s_0) := \sup_{u \in \mathcal{P}} G(s_0, u),$$

and gives the maximal value of the return G for the initial state $s_0 \in \mathcal{S}$ over all controls $u \in \mathcal{P}$. The supremum is taken over the set $\mathcal{P}$ of all bounded, Lebesgue measurable functions $u \colon \mathbb{R}_0^+ \to \mathcal{A}(t)$.

The following lemma states that the optimal value function v_* can be split into a sum for the time interval $[0, \Delta t)$ and one for the rest $[\Delta t, \infty)$ of the time [49, Lemma I.7.1]. This is analogous to the Bellman optimality equation (2.7).

Lemma 9.2 (dynamic-programming principle)**.** *The equation*

$$v_*(s_0) = \sup_{u \in \mathcal{P}} \Big(\int_0^{\min(\Delta t, \tau)} \mathrm{e}^{-\gamma t} r(s(t), u(t)) \mathrm{d}t + \mathrm{e}^{-\gamma \tau} R(s(\tau)) [\tau < \Delta t] + \mathrm{e}^{-\gamma \Delta t} v_*(s(\Delta t)) [\tau \geq \Delta t] \Big) \qquad \forall s_0 \in \mathcal{S} \quad \forall \Delta t \in \mathbb{R}_0^+$$

holds. Here the Iverson notation means that [statement] $= 1$ *if the statement holds true and* [statement] $= 0$ *otherwise.*

In the following, we approximate the right side in Lemma 9.2 for small Δt. The first term can be approximated as

$$\int_0^{\min(\Delta t, \tau)} \mathrm{e}^{-\gamma t} r(s(t), u(t)) \mathrm{d}t = \Delta t r(s_0, u(0)) + o(\Delta t).$$

The second term tends to zero as $\Delta t \to 0$. The third term, the optimal value function becomes

$$\begin{aligned} v_*(s(\Delta t)) &= v_*(s_0) + \Delta t \nabla v_*(s_0) \cdot \dot{s}(0) + o(\Delta t) \\ &= v_*(s_0) + \Delta t \nabla v_*(s_0) \cdot f(s_0, u(0)) + o(\Delta t) \end{aligned}$$

using Taylor expansion and the state equation (9.1). Next, we divide the equation by Δt to find

$$\frac{1 - \mathrm{e}^{-\gamma \Delta t}}{\Delta t} v_*(s_0) = \sup_{u \in \mathcal{P}} \left(r(s_0, u(0)) + \mathrm{e}^{-\gamma \Delta t} \nabla v_*(s_0) \cdot f(s_0, u(0)) + \frac{o(\Delta t)}{\Delta t} \right)$$

for all $s_0 \in \mathcal{S}$ and for sufficiently small Δt. Finally, we obtain the Hamilton-Jacobi-Bellman (HJB) equation by letting Δt tend to zero.

Theorem 9.3 (Hamilton-Jacobi-Bellman equation)**.** *If the optimal value function v_* is in $C^1(\overline{\mathcal{S}})$, then it satisfies the* Hamilton-Jacobi-Bellman *equation*

$$\gamma v_*(s_0) = \sup_{a \in \mathcal{A}(0)} (r(s_0, a) + \nabla v_*(s_0) \cdot f(s_0, a)) \qquad \forall s_0 \in \mathcal{S} \tag{9.2}$$

with the boundary condition

$$v_*(s) \geq R(s) \qquad \forall s \in \partial \mathcal{S}. \tag{9.3}$$

Note that here the supremum is taken over all actions $a \in \mathcal{A}(0)$ available at time zero.

The inequality in the boundary conditions holds because there may be points $s \in \partial\mathcal{S}$ for which a control exists such that $G(s, u(t)) > R(s)$, implying the strict inequality. It is also possible that the trajectory immediately exits the state space after starting on the boundary $\partial\mathcal{S}$. If this is the optimal control, then the equality in the boundary condition holds.

The HJB equation in Theorem 9.3 is a necessary condition for its solution being the optimal value function. The following theorem states that it is also a sufficient condition [49, Theorem I.7.1].

Theorem 9.4 (sufficient condition)**.** *Suppose that $w \in C^1(\overline{\mathcal{S}})$ satisfies (9.2) and (9.3). If $\tau = \infty$, suppose that w also satisfies the equation $\lim_{t\to\infty} \mathrm{e}^{-\gamma t} w(s(t)) = 0$. Then the inequality $w(s) \geq v_*(s)$ holds for all $s \in \overline{\mathcal{S}}$.*

Furthermore, suppose that there exists a control u_* such that

$$u_*(t) \in \arg\max_{a\in\mathcal{A}(t)} \{r(s_*(t), a) + \nabla w(s_*(t)) \cdot f(s_*(t), a)\} \tag{9.4}$$

for almost all $t \in [0, \tau_*)$ and that $w(s_*(\tau_*)) = R(s_*(\tau_*))$ if $\tau_* < \infty$, where s_* is the solution of the state equation (9.1) for $u = u_*$ and τ_* is the corresponding exit time. Then u_* is optimal for the initial state s and the equation

$$w(s) = v_*(s) \qquad \forall s \in \mathcal{S}$$

holds.

Proof. Since $w \in C^1(\overline{\mathcal{S}})$, we can start from the equality

$$\begin{aligned} \mathrm{e}^{-\gamma t} w(s(t)) &= w(s) + \int_0^t \frac{\mathrm{d}}{\mathrm{d}t'} (\mathrm{e}^{-\gamma t'} w(s(t'))) \mathrm{d}t' \\ &= w(s) + \int_0^t \mathrm{e}^{-\gamma t'} (-\gamma w(s(t')) + \dot{s}(t') \cdot \nabla w(s(t'))) \mathrm{d}t'. \end{aligned}$$

Using the state equation (9.1), we find

$$\mathrm{e}^{-\gamma t} w(s(t)) = w(s) + \int_0^t \mathrm{e}^{-\gamma t'} (-\gamma w(s(t')) + f(s(t'), u(s)) \cdot \nabla w(s(t'))) \mathrm{d}t',$$

and using (9.2), we find

$$\mathrm{e}^{-\gamma t} w(s(t)) \leq w(s) - \int_0^t \mathrm{e}^{-\gamma t'} r(s(t'), u(t')) \mathrm{d}t' \qquad \forall u \in \mathcal{P} \quad \forall t \in [0, \tau).$$

Letting t tend to the exit time τ yields

$$w(s) \geq \int_0^\tau \mathrm{e}^{-\gamma t'} r(s(t'), u(t'))\mathrm{d}t' + \lim_{t\to\tau} \mathrm{e}^{-\gamma t} w(s(t)).$$

In the case $\tau < \infty$, the inequality $\lim_{t\to\tau} \mathrm{e}^{-\gamma t} w(s(t)) \geq \mathrm{e}^{-\gamma\tau} R(s(\tau))$ holds for the last term. In the case $\tau = \infty$, the limit is zero by assumption. In both cases, we thus have the inequality

$$w(s) \geq G(s, u) \qquad \forall s \in \mathcal{S}$$

for all controls $u \in \mathcal{P}$, which implies

$$w(s) \geq v_*(s) \qquad \forall s \in \mathcal{S}.$$

This concludes the proof of the first assertion.

The first assumption of the second assertion of the theorem means that the action $u_*(t)$ is always maximal. Together with the second assumption, the same calculations as above can be performed for u_* instead of u, but with equalities everywhere. Thus we have $w(s) = G(s, u_*)$ for all $s \in \mathcal{S}$ and hence the equality

$$w(s) = v_*(s) \qquad \forall s \in \mathcal{S}$$

holds, which concludes the proof. □

Knowing the dynamics f of the system, we solve (9.2) in Theorem 9.3 for the optimal value function v_*. Knowing v_*, we can then use (9.4) in Theorem 9.4 to find an optimal control. However, so far we have used controls as functions of time and not policies as functions of state. Both a control $u\colon \mathbb{R}_0^+ \to \mathcal{A}$ and a policy $\pi\colon \mathcal{S} \to \mathcal{A}$ are related simply by

$$u(t) = \pi(s(t)).$$

Therefore, we use (9.4) to find an optimal policy by choosing

$$\pi_*(s) \in \operatorname*{arg\,max}_{a\in\mathcal{A}(t)} \{r(s, a) + \nabla v_*(s) \cdot f(s, a)\}.$$

9.3 An Example of Optimal Control

The following, simple example shows that the optimal value function is in general not a classical solution of the HJB equation (9.2). Therefore the question arises, in which class of solutions the optimal value function is the *unique* solution of

the HJB equation, if such a class of solutions exists at all. The answer to this question are viscosity solutions. The main result will be that the optimal value function is the unique viscosity solution of the HJB equation.

The example a one-dimensional control problem [50]. The system dynamics are given by

$$\begin{aligned} \dot{s}(t) &= u(t) \qquad \forall t \in \mathbb{R}_0^+, \\ s(0) &= s_0, \end{aligned}$$

where $\mathcal{S} := [0,1]$ and $\mathcal{A} := \{\pm 1\}$, and hence $s \colon \mathbb{R}_0^+ \to [0,1]$ and $u \colon \mathbb{R}_0^+ \to \{\pm 1\}$. The interpretation in classical mechanics is that the velocity of a particle at position $s(t)$ is controlled to be either $+1$ or -1. We define the current reward r to always vanish and the boundary reward to be $R(0) := R_0 > 0$ and $R(1) := R_1 > 0$. Therefore the return is

$$G(s_0, u) = \begin{cases} \mathrm{e}^{-\gamma\tau} R_0, & s(\tau) = 0, \\ \mathrm{e}^{-\gamma\tau} R_1, & s(\tau) = 1. \end{cases}$$

The optimal value function is easily found. Since the current reward always vanishes, the best policy is to reach the boundary as quickly as possible because of the factor $\mathrm{e}^{-\gamma\tau}$. Since we can only go to the left or to the right, the exit time is $\tau = 1 - s$ or $\tau = s$, respectively, yielding the optimal value function

$$v_*(s) := \max(R_0 \mathrm{e}^{-\gamma s}, R_1 \mathrm{e}^{-\gamma(1-s)}). \tag{9.5}$$

The HJB equation (9.2) simplifies to

$$v_*(s) = \max_{a \in \mathcal{A} = \{\pm 1\}} a v_*'(s) = |v_*'(s)|$$

with the boundary conditions $v_*(0) \geq R_0$ and $v_*(1) \geq R_1$.

The first problem is that the optimal value function is not a classical solution of the HJB equation (9.2). Already in the case $R_0 := 1$, $R_1 := 1$, and $\gamma := 1$, the optimal value function is not differentiable (at one point, namely $s = 1/2$).

Therefore it is plausible to admit generalized solutions that are differentiable almost everywhere. However, the second problem is that there may be infinitely many solutions satisfying the HJB equation (9.2) almost everywhere.

The third problem is that the optimal value function may satisfy the boundary condition only as a strict inequality. For the case $R_0 := 1$, $R_1 := 5$, and $\gamma := 1$, the optimal value function is $v_* = R_1 \mathrm{e}^{s-1}$. The boundary condition at $s = 0$ is $V(0) > R_0$. The reason is that the reward R_1 for leaving the domain $[0,1]$ at $s = 1$ is so much larger that the reward for leaving at $s = 0$ that it is always the best policy to move to the right, even when starting at $s = 0$. The boundary condition is therefore satisfied at $s = 0$ as a strict inequality.

9.4 Viscosity Solutions

It turns out that the correct solution type for the HJB equation (9.2) are viscosity solutions in the sense that in this class of solutions, unique existence can be guaranteed. This is important: if the optimal value function is the unique solution of the HJB equation, we know that after solving the equation we can immediately solve the optimal-control problem by choosing the actions according to (9.4).

The name of viscosity solutions refers to the vanishing-viscosity method used to show their existence [51].

To state some properties of viscosity solutions, we write the first-order equation under consideration as the boundary-value problem

$$H(s, v, \nabla v) = 0 \qquad \forall s \in \overline{\mathcal{S}}, \tag{9.6a}$$

$$v(s) = w(s) \qquad \forall s \in \partial\mathcal{S}, \tag{9.6b}$$

where $\mathcal{S}$ is an open domain, $w\colon \partial\mathcal{S} \to \mathbb{R}$ is the boundary condition, and the given function H is called the Hamiltonian of the system. The HJB equation (9.2) corresponds to

$$H(s, v, p) := \gamma v - \sup_{a \in \mathcal{A}(s)} \left(r(s, a) + p \cdot f(s, a)\right)$$

Definition 9.5 (viscosity solution)**.** Suppose $v\colon \overline{\mathcal{S}} \to \mathbb{R}$ is continuous and that $v = w$ on $\partial\mathcal{S}$.

The function v is called a *viscosity subsolution* of (9.6) if the following statement holds for all $\phi \in C^1(\mathcal{S})$: If $v - \phi$ has a local maximum at $s_0 \in \mathcal{S}$, then $H(s_0, v(s_0), \nabla\phi(s_0)) \leq 0$.

The function v is called a *viscosity supersolution* of (9.6) if the following statement holds for all $\phi \in C^1(\mathcal{S})$: If $v - \phi$ has a local minimum at $s_0 \in \mathcal{S}$, then $H(s_0, v(s_0), \nabla\phi(s_0)) \geq 0$.

The function v is called a *viscosity solution* of (9.6) if it is both a viscosity subsolution and a viscosity supersolution.

Continuing the example in Section 9.3 for the case $R_0 := 1$, $R_1 := 1$, and $\gamma := 1$, it can be checked by elementary calculations that (9.5) is a viscosity solution of the HJB equation $H(s, v(s), v'(s)) := v(s) - |v'(s)| = 0$ as follows. The function v_* is a classical solution on both intervals $(0, 1/2)$ and $(1/2, 1)$ and hence also a viscosity solution on these intervals by Lemma 9.6. To establish that v_* is a viscosity solution, it suffices to check the conditions in the definition at $s = 1/2$.

Next, we summarize a few basic properties of viscosity solutions.

Lemma 9.6. *Suppose $v \in C^1(\mathcal{S})$ be a classical solution of (9.6). Then it is also a viscosity solution.*

Lemma 9.7. *Suppose $v \in C(\overline{\mathcal{S}})$ is a viscosity solution of (9.6) and suppose v is differentiable at $s_0 \in \mathcal{S}$. Then $H(s_0, v(s_0), \nabla v(s_0)) = 0$.*

Proof. [52, Section 10.1.2]. □

Theorem 9.8. *Under certain assumptions on the Hamiltonian H, the boundary-value problem (9.6) has at most one bounded viscosity solution.*

Proof. [51, Theorem III.1]. □

So far, we have seen that the solution of the boundary-value problem (9.6) is unique among the viscosity solutions under some assumptions. However, the boundary condition in Theorem 9.3 is the inequality (9.3) and not an equality such as the boundary condition in (9.6). The theorems below show that the HJB equation has a unique viscosity solution with the following inequality boundary conditions and with a further assumption.

Definition 9.9 (viscosity solution with inequality boundary conditions)**.** Suppose v is a viscosity solution of the equation

$$H(s, v(s), \nabla v(s)) = 0.$$

Then v is called a *viscosity solution with the inequality boundary conditions*

$$v(s) \geq w(s) \qquad \forall s \in \partial\mathcal{S}$$

if the following two conditions hold.

1. If $\phi \in C^1(\overline{\mathcal{S}})$ and the function $v - \phi$ has a local maximum at $s_0 \in \partial\mathcal{S}$, then
$$\min(H(s_0, v(s_0), \nabla\phi(s_0)), v(s_0) - w(s_0)) \leq 0.$$
2. If $\phi \in C^1(\overline{\mathcal{S}})$ and the function $v - \phi$ has a local minimum at $s_0 \in \partial\mathcal{S}$, then
$$\max(H(s_0, v(s_0), \nabla\phi(s_0)), v(s_0) - w(s_0)) \geq 0.$$

The following assumption means that at any point on the boundary of the state space there is at least one trajectory that is not tangential to the boundary.

Assumption 9.10. The following two assumptions hold for all $s \in \partial\mathcal{S}$, where $n(s)$ denotes the outward pointing normal vector of $\mathcal{S}$ at s.

1. If there exists an $a \in \mathcal{A}$ such that $f(s,a) \cdot n(s) \leq 0$, then there exists an $a' \in \mathcal{A}$ such that $f(s,a') \cdot n(s) < 0$.

2. If there exists an $a \in \mathcal{A}$ such that $f(s,a) \cdot n(s) \geq 0$, then there exists an $a' \in \mathcal{A}$ such that $f(s,a') \cdot n(s) > 0$.

Theorem 9.11. *Suppose that Assumption 9.10 holds. Then the equation (9.2) with the boundary condition (9.3) has a unique viscosity solution with an inequality boundary condition.*

Proof. [50, Theorem 4], [49, Section II.11 and II.13]. □

In summary, viscosity solutions are the right class of solutions for the HJB equation (9.2) in the sense that there always exists a unique solution for the purposes of Theorem 9.3 and solving the optimal-control problem.

9.5 Stochastic Optimal Control

If the environment is stochastic, the dynamics of the system are described by a stochastic ordinary differential equation. This case is realistic because of random fluctuations in the system and lack of precision in measurements.

In the case of additive and normally distributed noise, the dynamics of the system are given by the stochastic ordinary differential equation

$$\mathrm{d}s = f(s(t)), u(t))\mathrm{d}t + \sigma(s(t), u(t))\mathrm{d}\omega \qquad \forall t \in \mathbb{R}_0^+, \tag{9.7a}$$

$$s(0) = s_0 \in \mathcal{S} \tag{9.7b}$$

to be understood in the sense of Itô calculus. Here ω is a *Brownian motion* of dimension $d := \dim \mathcal{S}$, and σ is an $n \times d$ matrix, where $n := \dim \mathcal{S} + \dim \mathcal{A}$.

Definition 9.12 (stochastic process)**.** If $(\Omega, \mathcal{F}, P)$ is a probability space and (S, Σ) is a measurable space, then a *stochastic process* is a set $\{X_t \mid t \in T\}$ of random variables X_t with values in S.

Definition 9.13 (Brownian motion)**.** A *Brownian motion* or a *Wiener process* is a stochastic process ω that satisfies the following conditions.

1. $\omega(0) = 0$.

2. ω_t is almost surely continuous for all $t \in \mathbb{R}_0^+$.

3. The increments of ω are independent, i.e., the increments $\omega_{t_1} - \omega_{s_1}$ and ω_{t_2} and ω_{s_2} are independent random variables if $0 \leq s_1 < t_1 \leq s_2 < t_2$.

4. The increments are normally distributed; more precisely,

$$\forall t \in \mathbb{R}_0^+ : \quad \forall s \in [0,t] : \qquad \omega_t - \omega_s \sim N(0, t-s).$$

In the stochastic case, the return

$$G \colon \mathcal{S} \times (\mathbb{R}_0^+ \to \mathcal{A}(t)) \to \mathbb{R},$$
$$G(s_0, u) := \mathbb{E}\left[\int_0^\tau \mathrm{e}^{-\gamma t} r(s(t), u(t)) \mathrm{d}t + \mathrm{e}^{-\gamma\tau} R(s(\tau))\right].$$

is the expected value over all trajectories of the stochastic process that is the solution of the state equation. Then the definition of the optimal value function remains the same.

It can be shown that if the optimal value function v_* is in $C^2(\mathcal{S} \to \mathbb{R})$, then it satisfies the HJB equation

$$\gamma v_*(s) = \sup_{a \in \mathcal{A}(0)} \left(r(s,a) + \nabla v_*(s) \cdot f(s,a) + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n a_{ij}(s,a) \frac{\partial v_*}{\partial s_i \partial s_j}(s) \right) \qquad \forall s \in \mathcal{S}, \tag{9.8a}$$

$$v_*(s) = R(s) \qquad \forall s \in \partial\mathcal{S}, \tag{9.8b}$$

where $(a_{ij}) = A := \sigma\sigma^\top$. This HJB equation is a nonlinear, second-order partial differential equation.

If the matrix A is uniformly elliptic, then the HJB equation (9.8) has a unique classical solution. Otherwise, the concept of viscosity solutions extended to second-order equations [53] can be used.

9.6 Bibliographical and Historical Remarks

Problems

Chapter 10

Deep Reinforcement Learning

> That a computer can be programmed to play legal chess or checkers is hardly remarkable; that it can learn from experience to play master checkers is interesting and represents a feat of programming. It cannot, at present, learn from experience to play master chess, and an ability of this type would represent an astounding breakthrough in the theory of adaptive processes.
>
> Richard Bellman [38, p. 37].

Algorithmic advancements and huge computational resources have made it possible in recent years to train reinforcement-learning agents based on deep neural networks that can outperform humans in playing games such as Atari 2600, chess, and Go.

10.1 Introduction

Series of papers coming out of Google DeepMind: [13], [54], [8], [9], [14], [55].

10.2 Atari 2600 Games

In [13], the action-value function was represented by a deep neural network, termed a deep Q-network (DQN). The DQN agent received only the pixels and the game score as inputs and was able to achieve a level comparable to that of a professional human games tester. Importantly, the same algorithm, neural-network architecture, and hyperparameters were used across a set of 49 games,[1]

[1] DQN plays Breakout: https://www.youtube.com/watch?v=TmPfTpjtdgg. DQN plays Space Invaders: https://www.youtube.com/watch?v=W2CAghUiofY.

representing a diverse collection of tasks.

The algorithm is shown in Algorithm 19.

Algorithm 19 deep Q-network (DQN) with experience replay.

initialization:
initialize replay memory D
initialize action-value function $\hat{q}(\mathbf{w})$ with random weights $\mathbf{w}$
initialize target action-value function $\hat{q}(\mathbf{w}^-)$ with weights $\mathbf{w}^- := \mathbf{w}$

for episode $\in (0, 1, \dots, M)$ **do** ▷ for all episodes
  initialize $s_0 := x_0$ and preprocess $\phi_0 := \phi(s_0)$
  for $t \in (0, 1, \dots, T)$ **do** ▷ for all time steps
    select action a_t ϵ-greedily as $\arg\max_a \hat{q}(s_t, a, \mathbf{w})$
    perform action a_t in the emulator, obtain reward r_{t+1} and image x_{t+1}
    $s_{t+1} := (s_t, a_t, x_{t+1})$
    preprocess $\phi_{t+1} := \phi(s_{t+1})$
    store transition $(\phi_t, a_t, r_{t+1}, \phi_{t+1})$ in D
    sample transitions $(\phi_j, a_j, r_{j+1}, \phi_{j+1})$ from D
    set

$$y_j := \begin{cases} r_j, & \text{if episode terminates at step } j+1, \\ r_j + \gamma \max_{a'} \hat{q}(\phi_{j+1}, a', \mathbf{w}^-), & \text{otherwise} \end{cases}$$

    perform a gradient-descent step on $\sum_j \big(y_j - \hat{q}(s, a, \mathbf{w})\big)^2$ w.r.t. $\mathbf{w}$
    every C steps: $\mathbf{w}^- := \mathbf{w}$
  end for
end for

We denote the approximation, a neural network, of the action-value function by $\hat{q}(s, a, \mathbf{w})$ as usual. The Q-learning update uses the quadratic loss function

$$L_i(\mathbf{w}_i) := \mathbb{E}_{(s,a,r,s') \sim U(D_i)}\big[\big(r + \gamma \max_{a'} \hat{q}(s', a', \mathbf{w}_i^-) - \hat{q}(s, a, \mathbf{w}_i)\big)^2\big],$$

which is the mean-squared error of the Bellman equation. Here the agent's experiences

$$(s_t, a_t, r_{t+1}, s_{t+1})$$

in each time step t are stored in data sets $D_t := \{e_1, \dots, e_t\}$, and the expected value is approximated by drawing samples, or minibatches in the language of neural networks, (s, a, r, s') uniformly from the data set D_i in training iteration i.

The parameters $\mathbf{w}_i^-$ are parameters from an iteration before the i-th one, in contrast to the parameters $\mathbf{w}_i$ in the i-th iteration. The gradient of the loss function is given by

$$\nabla_{\mathbf{w}_i} L_i(\mathbf{w}_i) = -2\mathbb{E}_{(s,a,r,s')\sim U(D_i)}\big[(r+\gamma \max_{a'} \hat{q}(s', a', \mathbf{w}_i^-) - \hat{q}(s, a, \mathbf{w}_i))\nabla_{\mathbf{w}_i} \hat{q}(s, a, \mathbf{w}_i)\big].$$

Q-learning is recovered as the special case where $\mathbf{w}_i^- := \mathbf{w}_{i-1}$ and the expectation is replaced by just using the current sample.

The neural network that serves as the approximation of the action-value function takes a preprocessed $84 \times 84 \times 4$ image as its input. Three convolutional layers are followed by two fully connected layers, and the activation functions in each layer are the rectifiers $x \mapsto \max(0, x)$. The output layer has a single output for each valid action. There are eighteen valid actions: nine directions (including no input corresponding to a centered joystick) and these nine directions with the fire button pressed simultaneously.

The RMSProp algorithm with samples or minibatches of size 32 was used to train the neural network in the DQN algorithm. The particular choice of training algorithm is not a defining feature of the whole procedure, as different algorithms were used in later works.

For all games, the discount factor was $\gamma = 0.99$. The behavior policy was ϵ-greedy and started with $\epsilon = 1$, which was linearly reduced to $\epsilon = 0.1$ after the first million frames and fixed at this value thereafter. Fifty million frames were used for training, which corresponded to about 38 days of game experience. Frames were skipped; more precisely, in all games actions were selected only on every fourth frame, and the last action was repeated on all frames in between. This simple frame-skipping technique helped reduce computation time, since the emulator runs much faster than having the agent select an action. The size of the buffer used for experience replay was one million of the most recent frames.

The stability of the algorithm is important, especially when dealing nonlinear functions such as neural networks. Two provisions improve the stability. First, the error terms $y_j - \hat{q}(s, a, \mathbf{w})$ are clipped to always be between -1 and 1. Second, two separate networks, represented by the parameters $\mathbf{w}$ and $\mathbf{w}^-$, are used. The target values y_j in the Q-learning updates are found using the network with the older parameters $\mathbf{w}^-$, and the parameters $\mathbf{w}^-$ are updated to the current parameters $\mathbf{w}$ regularly after a certain number of time steps. This provision increases the stability of the algorithm, since an update that increases $\hat{q}(s_t, a_t)$ often also increases $\hat{q}(s_{t+1}, a)$ for all a as consecutive states are often highly correlated in such applications. Hence the target y_j is also increased, which possibly leads to oscillations or divergence of the action-value function. Generating the update targets y_j using the older parameters $\mathbf{w}^-$ delays any ef-

fects of updates to $\hat{q}$ on the targets y_j and thus makes oscillations or divergence much more unlikely.

10.3 Go and Tree Search (AlphaGo)

Go is the most challenging of the classic board games due to its huge search space, being larger than the one of chess, and the difficulty of evaluating positions and moves. In [54], a computer program named AlphaGo defeated a human professional player in the full-sized game of Go for the first time. The human player was the European Go champion and he was defeated by 5 games to 0. Playing against other programs, AlphaGo won 99.8% of the games.

MCTS (see, e.g., [23, Section 8.11]) had been known to be the best algorithm for playing Go and to achieve strong amateur play, and had previously been used with policies or value functions based on linear combinations of input features. The main innovation in [54] was to employ deep neural networks as policies and value functions and to learn in a stable manner while doing so. Unsurprisingly, the computational effort was enormous.

Training the AlphaGo program consisted of several stages using different methods. In the first step, the policy network was trained using a data set of expert human moves and supervised learning. The advantage is fast learning in the beginning based on gradients of high quality, although learning in this step maximizes predictive accuracy and not winning games.

Then, reinforcement learning was used to optimize the policy network from the first step using self-play. In this way, the policy network is adjusted towards winning games rather than accurate prediction of expert human moves. Stochastic gradient ascent was used for optimization, and the current policy network played against a randomly selected previous iteration of the policy network. Randomizing the opponents stabilized training and prevented overfitting to playing against the current policy.

In the third step, a value network was trained to predict the winners of games played by the policy from the second step playing against itself. A new data set was generated to prevent overfitting, and stochastic gradient descent was used to minimize the mean squared error.

Finally, the AlphaGo program used MCTS and combined the policy and value networks from the previous steps. The positions were evaluated using the value network, and the actions were sampled using the policy network, making the MCTS algorithm very efficient.

10.4 Learning Go Tabula Rasa (AlphaGo Zero)

The next stage in the development of the Alpha programs was to render the existence of a preexisting data set of expert moves superfluous. This was achieved in [8], where AlphaGo Zero learned tabular rasa, i.e., without any preexisting knowledge, to achieve superhuman proficiency in playing Go. It became the first program to defeat a world champion, thus achieving superhuman performance, and it won 100 to 0 against AlphaGo [54].

In AlphaGo, MCTS evaluated positions and selected moves using the value and policy deep neural networks. These neural networks in AlphaGo were initially trained using supervised learning from a data set with human expert moves. Reinforcement learning and self-play were used only later. AlphaGo Zero used solely reinforcement learning without any human expert moves, guidance, or domain knowledge beyond the rules of the game; it learned tabula rasa, i.e., it started learning from random play.

While AlphaGo used MCTS and separate policy and a value networks, AlphaGo Zero used solely a single neural network. It also employed a simpler tree search based on this single neural network to evaluate positions and sample moves. The single neural network receives a raw board representation of the position and its history, and it outputs both a vector of move probabilities and scalar value that estimates the probability of the current player winning from the current position. Thus the single network takes on the roles of both the policy and value networks. The structure of the network is quite complicated, consisting of many blocks of convolutional layers with batch normalization and rectifier nonlinearities.

10.5 Chess, Shogi, and Go through Self-Play (AlphaZero)

In [9], the approach taken in AlphaGo Zero was generalized into a single algorithm, called AlphaZero, that achieved superhuman performance in the challenging board games. Like AlphaGo Zero, AlphaZero started from random play and without any domain knowledge except the game rules. AlphaZero could defeat a world champion in chess, shogi (Japanese chess, with a more complex game tree than chess), and Go.

10.6 Video Games of the 2010s (AlphaStar)

In [14] and [55], attention returned to video games with partial observability after the board games with no hidden information.

In [14], learning in the presence of multiple agents was addressed as an extension of the work on the board games that are two-player turn based games. The video game was a three-dimensional multi-player first-person video game, namely Quake III Arena in a mode called Capture the Flag. A tournament-style evaluation was used to evaluate the performance of the agent, which could only use pixels and game points scored as its input and achieved human-level performance.

Learning proceeded by concurrently training a diverse population of agents which played against each other. This approach seemed to stabilize learning despite the partially observable environments and the multi-agent nature of the game. The learning algorithm for each player was a multi-step actor-critic policy-gradient algorithm with off-policy correction and auxiliary tasks. The policies were represented as multi-time-scale recurrent neural networks with external memory. The agents built hierarchical temporary representations and recurrent latent variable model for their sequential input data. This results in the construction of temporally hierarchical representations that favor the use of memory and of temporally coherent action sequences.

Self-play can be unstable and does not support concurrent training in its basic form. Therefore a population of different agents was trained in parallel, which stabilized training. In the episodes, each agent learned by playing with team mates and against opponents sampled from the whole population.

In [55], the AlphaStar program that plays the StarCraft II game is described. StarCraft II is considered one of the most difficult games in professional esports due to its complexity and multi-agent challenges. AlphaStar employs data from both human and agent games and strategies and counter-strategies that are continually adapted. The policies are represented by deep neural networks. AlphaStar competed in a series of online games against human players in the full game of StarCraft II. It was rated at grandmaster level for all three StarCraft races and ranked above 99.8% of officially ranked human players.

10.7 Improvements to DQN and their Combination

In [56], six independent extensions to the DQN algorithm [13] were discussed, and their combinations were studied empirically. Each of the six extensions turned out to improve performance on a selection of 57 Atari 2600 games, and

substantially so in most cases. All six improvements can also be combined into an algorithm called the rainbow algorithm in this work. Furthermore, an ablation study was performed to show the contribution of each improvement to the overall performance of the rainbow algorithm.

While many extensions to the DQN algorithm, Algorithm 19, have been proposed, the six extensions were chosen such that they address distinct limitations of the DQN algorithm. Before discussing the six extensions, we recall that DQN uses Q-learning to define the loss function that is minimized in order to determine the parameters of a (deep) neural network that is trained using stochastic gradient descent. Two important features to improve its stability in face of nonlinear function approximation are experience replay and the use of two parameters $\mathbf{w}$ (of the online network) and $\mathbf{w}^-$ (of the target network).

10.7.1 Double Q-Learning

The first extension is double Q-learning already presented as Algorithm 10 in Section 5.6.

10.7.2 Prioritized Replay

The DQN algorithm samples uniformly from the experience-replay buffer. However, using transitions from which there is much to learn more often would seem to be more efficient, and these transitions should be sampled more frequently. Prioritized replay hence assigns the probability

$$p_t \propto \left|R_{t+1} + \gamma_{t+1} \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}^-) - \hat{q}(S_t, A_t, \mathbf{w})\right|^\omega$$

to transitions based on the absolute TD error, where ω is a hyperparameter. Furthermore, new transitions are inserted in the replay buffer with higher priority.

10.7.3 Dueling Networks

Dueling networks are an architecture of neural networks specifically designed for value functions in reinforcement learning. They correspond to an action-value function of the form

$$q(s, a, \mathbf{w}) = v(f(s, \mathbf{w}_1), \mathbf{w}_2) + d(f(s, \mathbf{w}_1), a, \mathbf{w}_3) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} d(f(s, \mathbf{w}_1), a', \mathbf{w}_3),$$

where $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$ are the parameters of the shared encoder f, the value stream v, and the advantage stream d such that $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$. Thus the value and advantage streams share the convolutional encoder.

10.7.4 Multi-Step Methods

Multi-step methods are discussed in Chapter 5 and often lead to faster learning with a suitable chosen number of steps. A multi-step variant of DQN can be defined as minimizing the loss

$$\left(G_{t:t+n} + \gamma_{n,t} \max_{a' \in \mathcal{A}} \hat{q}(S_{t+n}, a', \mathbf{w}) - q(S_t, A_t, \mathbf{w})\right)^2$$

based on the n-step return $G_{t:t+n}$.

10.7.5 Distributional Reinforcement Learning

Distributional reinforcement learning is the largest extension and provides a conceptual shift. Instead of maximizing the expected return as usual in reinforcement learning, we can approximate the distribution of the returns. This can be achieved for example by discretizing the distribution of returns as discrete probability masses placed on a discrete support or (equidistant) grid $\mathbf{z}$. Then the distribution d_t at time t takes the values $p^i_\theta(s,a)$ on each grid point z_i, and d_t can be written as $d_t = (\mathbf{z}, \mathbf{p}_\theta(s,a))$. The parameter θ must be determined such that this distribution approximates the true distribution of returns.

To approximate the distribution of returns, a variant of Bellman's optimality equation for distributions is useful. Furthermore, the difference between the target distribution

$$d'_t := (R_{t+1} + \gamma_{t+1}\mathbf{z}, \mathbf{p}_{\theta'}(s, \arg\max_{a \in \mathcal{A}} \hat{q}(S_{t+1}, a, \theta'))$$

and d_t must be minimized, for example by minimizing the Kullbeck-Leibler divergence

$$D_{\mathrm{KL}}(\Phi_{\mathbf{z}}(d'_t) \,\|\, d_t).$$

Here the second argument of $\mathbf{p}_{\theta'}$ is the greedy action with respect to the mean action values

$$\hat{q}(S_{t+1}, a, \theta') = \mathbf{z} \cdot \mathbf{p}_{\theta'}(S_{t+1}, a),$$

and $\Phi_{\mathbf{z}}$ is the L^2 projection of the target distribution d'_t onto the grid $\mathbf{z}$.

10.7.6 Noisy Neural Networks

In some applications, such as in the game Montezuma's Revenge, rewards are delayed a long time from the actions and many actions must be performed to collect the first reward. In such cases, ϵ-greedy policies may provide insufficient

exploration. To overcome this limitation, noisy neural networks include a noisy linear hidden layer of the form

$$\mathbf{y} := (W\mathbf{x} + \mathbf{b}) + \left((W_{\text{noisy}} \odot \epsilon_W)\mathbf{x} + \mathbf{b}_{\text{noisy}} \odot \epsilon_{\mathbf{b}}\right)$$

which combines a deterministic, linear stream (the first term) with a noisy stream (the second term). Here ϵ_W and $\epsilon_{\mathbf{b}}$ are random variables, and $\odot$ denotes elementwise multiplication. Because the hidden layer consists of a deterministic and a noisy term, the neural network can learn to ignore the noisy stream over time even with different rates in different parts of the state space. This makes it possible to explore different parts of the state space at different speeds.

Chapter 11

Distributional Reinforcement Learning

11.1 Introduction

In this chapter, we assume that the state and action spaces are finite. Analogously to the term categorical probability distribution, this chapter is hence about categorical distributional reinforcement learning.

11.2 Markov Decision Processes and Bellman Operators

Definition 11.1 (Markov decision process)**.** A Markov decision process is described by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$.

$$P\colon \quad \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S}),$$

where $\mathcal{P}(\mathcal{E})$ is the set of probability measures on the set $\mathcal{E}$ of events. $\mathcal{P}_b(\mathcal{E})$ is the set of probability measures on $\mathcal{E}$ having bounded support.

$$p(s' \mid s, a) = P(s, a)(s')$$

Next, we recapitulate Bellman operators in the classic, non-distributional approach.

$$Q\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

T_π is a classic Bellman operator.

$$(T_\pi Q)(s,a) := \sum_{s'} p(s' \mid s,a)(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s')Q(s',a'))$$

T_π is a γ-contraction with respect to the maximum norm $\|.\|_\infty$.
Therefore there exists a unique fixed point Q_π

$$Q_\pi = T_\pi[Q_\pi]$$

T_* is the classic Bellman optimality operator

$$(T_* Q)(s,a) := \sum_{s'} p(s' \mid s,a)(r(s,a,s') + \gamma \max_{a' \in \mathcal{A}(s')} Q(s',a'))$$

Analogously to T_π, the Bellman optimality operator T_* is a γ-contraction with respect to the maximum norm. Therefore there exists a unique fixed point Q_*

$$Q_* = T_*[Q_*]$$

In order to formulate distributional versions of the Bellman operators, we need the push-forward measure.

If $\nu \in \mathcal{P}_b(\mathbb{R})$ is a probability measure and $f\colon \mathbb{R} \to \mathbb{R}$ is a measurable function, then the push-forward measure $f\#\nu$ is defined by

$$(f\#\nu)(A) := \nu(f^{-1}(A)) = \nu(\{x \in \mathbb{R} : f(x) \in A\})$$

where $A \subset \mathbb{R}$ is a Borel set.

In the following, we will need measures that are pushed by affine transformations of the form

$$l_{r,\gamma}(x) := r + \gamma x.$$

$$(l_{r,\gamma}\#\nu)(A) = \nu(l_{r,\gamma}^{-1}(A)) = \nu(\{x \in \mathbb{R} : r + \gamma x \in A\})$$

If $\gamma \neq 0$, we have $r + \gamma x = z \in A$ and therefore $x = (z - r)/\gamma$. This yields

$$(l_{r,\gamma}\#\nu)(A) = \nu\left(\left\{\frac{z-r}{\gamma} \in \mathbb{R} : z \in A\right\}\right).$$

If $\gamma = 0$, then

$$(l_{r,0}\#\nu)(A) = \nu(\{x \in \mathbb{R} : r \in A\}).$$

We have

$$\{x \in \mathbb{R} : r \in A\} = \begin{cases} \emptyset, & r \notin A, \\ \mathbb{R}, & r \in A, \end{cases}$$

and therefore

$$\nu(\{x \in \mathbb{R} : r \in A\}) = [\![r \in A]\!].$$

distributional Bellman operator

Lemma 11.2. *Suppose $r \in \mathbb{R}$, $\gamma \in \mathbb{R}^+$, and Z is a random variable. Then the CDF of the transformed random variable $r + \gamma Z$ can be written in terms of the CDF of Z as*

$$F_{r+\gamma Z}(z) = F_Z\left(\frac{z-r}{\gamma}\right) \qquad \forall z \in \mathbb{R}.$$

Proof. Using the definition of the CDF, we have

$$F_{r+\gamma Z}(z) = \mathbb{P}[r + \gamma Z \leq z] = \mathbb{P}\left[Z \leq \frac{z-r}{\gamma}\right] = F_Z\left(\frac{z-r}{\gamma}\right)$$

for any $z \in \mathbb{R}$. □

11.3 Distributional Speedy Q-Learning

[57]

11.4 Bibliographical Remarks

11.5 Exercises

Chapter 12

Large Language Models

Large Language Models are artificial neural networks that generate texts in natural languages following commands or prompts. In this chapter, the training and functioning of large language models, and ChatGPT in particular, is explained. Transformers and their attention mechanism provide the basis for text generation token by token. But generating grammatically correct and beautifully formulated text is not enough. The output of large language models should be aligned to the intentions of the user. This is achieved by various training steps, one of them being reinforcement learning from human feedback.

12.1 Introduction

The processing of natural language has been part of the goals of the field of artificial intelligence since its inception. The proposal for the Dartmouth Summer Research Project on Artificial Intelligence, dated August 31, 1955, mentions

- natural language,
- artificial neuronal networks,
- self-improvement, and
- creativity

among "some aspects of the artificial intelligence problem." (The Dartmouth Workshop took place one year later, in the summer of 1956.) These four aspects are certainly essential for large language models (LLM), showing how prescient the proposal was. The proposal says the following about the second of the seven aspects mentioned.

> "How Can a Computer be Programmed to Use a Language
>
> It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning and rules of conjecture. From this point of view, forming a generalization consists of admitting a new word and some rules whereby sentences containing it imply and are implied by others. This idea has never been precisely formulated nor have examples been worked out."

ChatGPT was launched on November 30, 2022. It revolutionized natural-language processing and has achieved at least some of the goals broadly outlined in the proposal for the Dartmouth Workshop. To which extent is ChatGPT intelligent and creative? The answer to this question is in the eye of the beholder, but there can be no doubt that ChatGPT shows intelligence and creativity.

Why did it take nearly seventy years until computers could be programmed to use natural language at a level comparable to humans or even at a superhuman level? There are two complications.

The first is that any program that can usefully deal with natural language requires a world model or common sense, thus knowing how the world works and how the subjects and objects in a text may interact. This information must be conveyed to the program somehow. Common sense is also required to be able to deal with ambiguities.

The second complication is that natural language is full of references, sometimes over large distances in a text. In order to resolve references correctly, context must be understood. A common example are personal pronouns that reference nouns.

LLM such as ChatGPT address the first complication by being training on huge, high-quality text corpora. This is an example of self-improvement: the learning algorithm extracts information (including grammar) from the text corpora and stores it in a new form, i.e., the neural network, for later use. No grammatical rules or facts are programmed into an LLM directly. Self-improvement and the use of huge text corpora necessitate a probabilistic approach to learning and to generating textual output in order to deal with contradictions that are most likely present in large text corpora. The probabilistic approach to generating output is linked to the so-called temperature parameter, which in turn is linked to creativity.

The second challenge is addressed by the attention mechanism of transformers.

12.2 Transformers

Transformers are a certain kind of deep neural network. Their defining feature is their sole use of the so-called attention mechanism, which makes it possible to learn references within a text or a time series, in a rather simple architecture. Transformers were introduced in [58] for use in machine translation in 2017. Before this publication, the dominant sequence transduction models for use in machine translation were recurrent or convolutional neural networks, but already used attention mechanisms. The advantages of transformers reported in [58] are superior quality, more parallelizable models, and shorter training times.

The following discussion of transformers is based on [58] and pertains mostly to the features that set them apart from other neural-network architectures, i.e., scaled dot-product attention, multi-head attention, and positional encoding. A simplified version of transformers for machine translation are the basis of ChatGPT, discussed later. Much of the design of neural-network architectures is empirical; after the success of transformers in machine translation, their use has expanded to many other application areas, and many variations have been proposed and implemented.

A schematic diagram of the original transformer architecture for machine translation is shown in Figure 12.1. We start with an overview, and go into the details later. The left part is the encoder stack, and the right part is the decoder stack. The input text in the source language is converted into a vector of so-called tokens by the input embedding. Positional encoding is added. The light gray box on the left side in the figure contains the encoder and consists of six identical layers. In each layer, there are two sublayers, i.e., multi-head attention and feed forward. At the end of each sublayer, the two indicated vectors are added, and their sum is normalized.

The input to the encoder is a window, i.e., a vector of tokens with fixed length, that slides along the whole input text. The length of this vector is called the model dimension and has the value $d_{\text{model}} = 512$ in [58]. To facilitate all connections, all embeddings and sublayers have the same length, i.e., the model dimension.

In the first iteration, the vector containing the previous output in the target language shown at the bottom right contains only padding. The decoder shown on the right is similar to the encoder shown on the left. Its layers are also repeated six times, but it contains three sublayers, i.e., masked multi-head attention, multi-head attention, and feed forward. The masking in the first sublayers prevents positions from attending to subsequent positions, and hence the predictions only depend on known output tokens. The output of the encoder is passed to the second sublayers, i.e., the multi-head attention layers, of the six

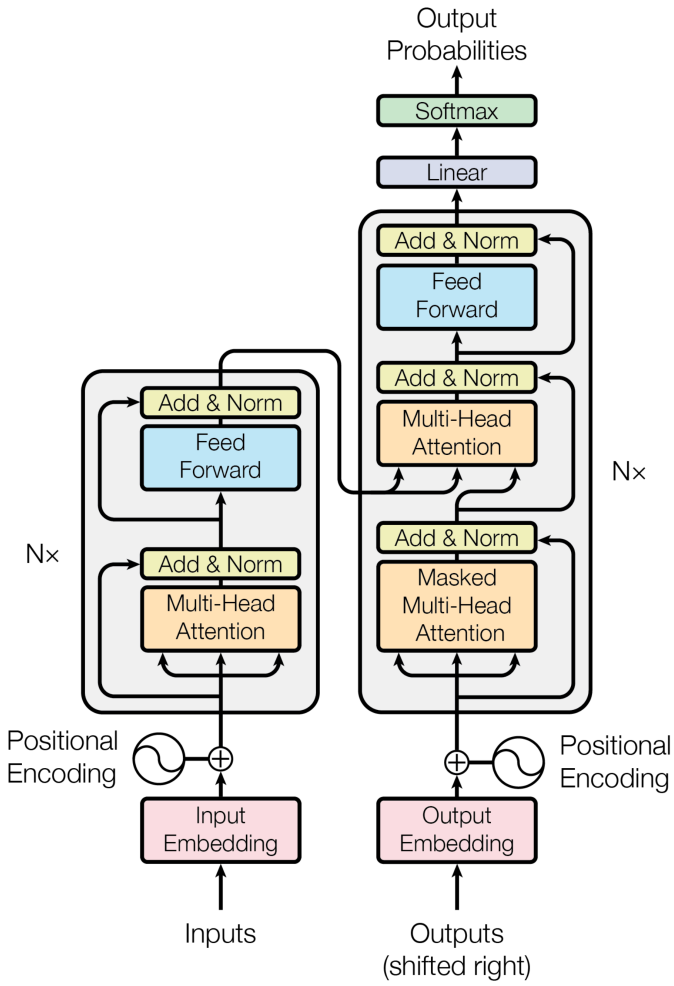


Figure 12.1: Transformer architecture for machine translation. Source: [58, Figure 1].

decoder layers, and conveys all meaning from the input to the output text.

Finally, a linear transformation is applied, and a softmax layer generates a single output token in a probabilistic manner depending on the temperature parameter T by assigning the probability

$$\text{softmax}(x_i) := \frac{\mathrm{e}^{x_i/T}}{\sum_i \mathrm{e}^{x_i/T}}$$

to the i-th element of the real-valued input vector x to the softmax layer.

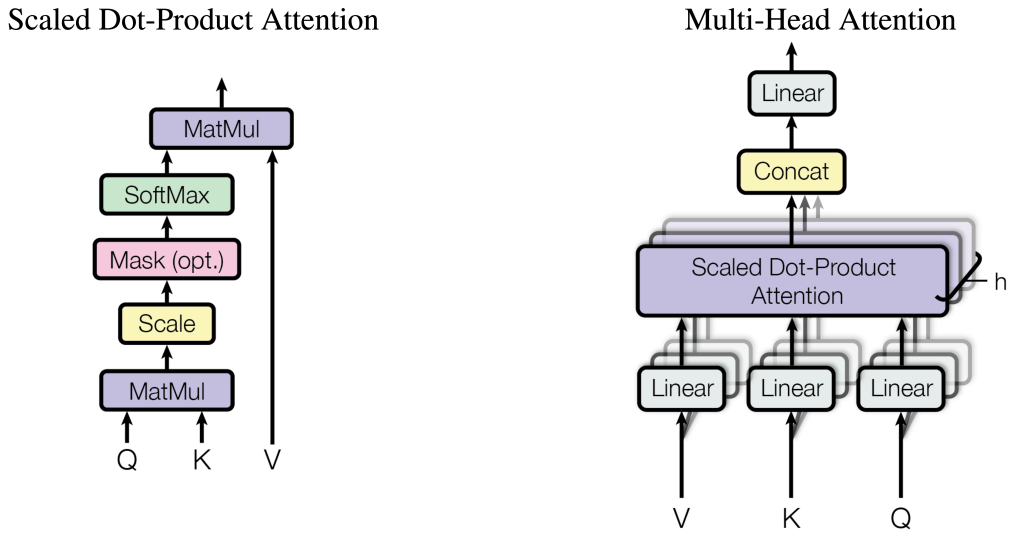


Figure 12.2: Scaled dot-product attention (left) and multi-head attention (right). Source: [58, Figure 2].

In this manner, a single token is generated in each iteration, which is an evaluation of the transformer, and appended to the output sequence. The transformer model is auto-regressive, as the previously generated tokens serve as additional input when generating the next token.

All textual inputs to a transformer are converted by embeddings to vectors of tokens, and the output of the transformer is another token. Why are tokens more useful than characters or words? Experience has shown that characters as the smallest textual units do not carry enough information. On the other hand, using words as the smallest units results in large vocabularies that contain tens of thousands of entries and is inflexible. Tokens are a useful compromise, and very good mental model of tokens are syllables.

The process of converting a text to tokens is called tokenization and is performed by a tokenizer. Each token has an ID, and the set of all tokens is called the vocabulary. A tokenizer can parse, i.e., convert a string to a list of characters; it can encode, i.e., convert a vector of tokens to their IDs; and it can decode, i.e., convert a vector of token IDs to a string.

Embeddings convert tokens to real valued vectors of dimension d_{model}. This makes it possible, for example, that tokens of similar meaning or use are converted to points that are close in the real vector space. to In [58], the embeddings are learned linear transformation, although other choices exist [59].

Next, we turn our attention to… attention. Figure 12.2 shows schematic

diagrams for scaled dot-product attention and multi-head attention.

In the mechanism of scaled dot-production attention, queries and keys of dimension d_k as well as values of dimension d_v are used. In practice, the attention function works on a set of queries simultaneously, and hence the queries, keys, and values are stored in matrices Q, K, and V. Scaled dot-product attention is defined as

$$\text{attention}(Q, K, V) := \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V.$$

Elements of the inner product $QK^\top$ are large whenever queries and keys point into the same direction and vanish whenever they are orthogonal. Thus, whenever the queries and keys are aligned, the values V are the result of the attention, otherwise the attention is (close to) zero. Therefore this mechanism enables the learning of grammar. For example, queries and keys make it possible to match nouns and pronouns.

All operations in this definition are standard functions and can be implemented using highly optimized code. The scaling factor $\sqrt{d_k}$ in the denominator is important, because the inner product in the argument to the softmax function may become large in magnitude for large values of d_k. The value of the scaling factor is motivated by the following fact. Suppose the elements of the two vectors $\mathbf{q} \in \mathbb{R}^{d_k}$ and $\mathbf{k} \in \mathbb{R}^{d_k}$ are independent random variables with expectation 0 and variance 1. Then their inner product $\mathbf{q} \cdot \mathbf{k}$ has mean 0 and variance d_k, which means that after scaling by $1/\sqrt{d_k}$ the variance of the inner product is 1.

In the mechanism of multi-head attention [58], the d_{model}-dimensional keys, values, and queries are projected h times using learned linear projections to d_k, d_k, and d_v dimensions, respectively. The attention function is applied to the projected versions in parallel, yielding d_v-dimensional vectors. These are concatenated and projected again, i.e.,

$$\begin{aligned} \text{head}_i &:= \text{attention}(QW_i^Q, KW_i^K, VW_i^V) \in \mathbb{R}^{d_v}, \\ \text{multiHead}(Q, K, V) &:= \text{concat}(\text{head}_1, \ldots, \text{head}_h) W^O \end{aligned}$$

with the parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. In [58], $h = 8$ and $d_k = d_v = d_{\text{model}}/h = 64$ was used. Since each of the multiple heads is smaller, the total computational cost is similar to that of single-head attention with full dimensionality.

The advantage of multi-head attention is that the model can pay attention to information from different representation subspaces at different positions. This is inhibited by averaging if a single attention head is used.

There are three applications of attention in [58]. Firstly, in the encoder-decoder attention layers, the queries come from the previous decoder layer, while the keys and the values come the output of the encoder. In this manner, all positions in the decoder can attend to all positions in the input sequence.

Secondly, the attention layers in the encoder are self-attention layers. In the encoder, all keys, values, and queries come from the output of the previous encoder layer. Hence, all positions in the encoder can attend to all positions in the previous layer in the encoder.

Thirdly, analogously to the attention layers in the encoder, the attention layers in the decoder are also self-attention layers, but with the slight difference that all positions in the decoder attend to all positions in the decoder up to and including that position. Otherwise, information could flow leftward and the auto-regressive property would be violated.

Finally, we discuss how positional information is encoded in transformers [58]. It is advantageous for the neural network to be able to use the order of the sequence and to know the distances between tokens. Such information about the relative and absolute positions of tokens is made available by positional encoding, which is added to the input embeddings at the bottoms of the encoder and decoder stacks. Both the embeddings and the positional embeddings have dimension d_{model} and are added.

Many approaches to positional embeddings exist. In [58], the sine and cosine functions

$$\begin{aligned} \text{PE}(p, 2i) &:= \sin(p/10\,000^{2i/d_{\text{model}}}), \\ \text{PE}(p, 2i+1) &:= \cos(p/10\,000^{2i/d_{\text{model}}}) \end{aligned}$$

of different frequencies, where p is the position and i the dimension, were used. The wavelengths are a geometric progression from 2π to $10\,000 \cdot 2\pi$.

Such a positional encoding has two advantages. Firstly, all values of PE are between -1 and $+1$, which is advantageous for training neural networks. Secondly, attention to relative positions is easily learned, since $\text{PE}(p+k, i)$ is a linear function of $\text{PE}(p, i)$ for any fixed k.

The authors compared this sinusoidal positional encoding with learned positional encodings, finding that both yielded nearly identical results, and eventually chose the sinusoidal positional encoding, because its behavior for sequence lengths longer than the ones encountered during training is more predictable.

12.3 InstructGPT and ChatGPT

G: generative, P: pretrained, T: transformer.

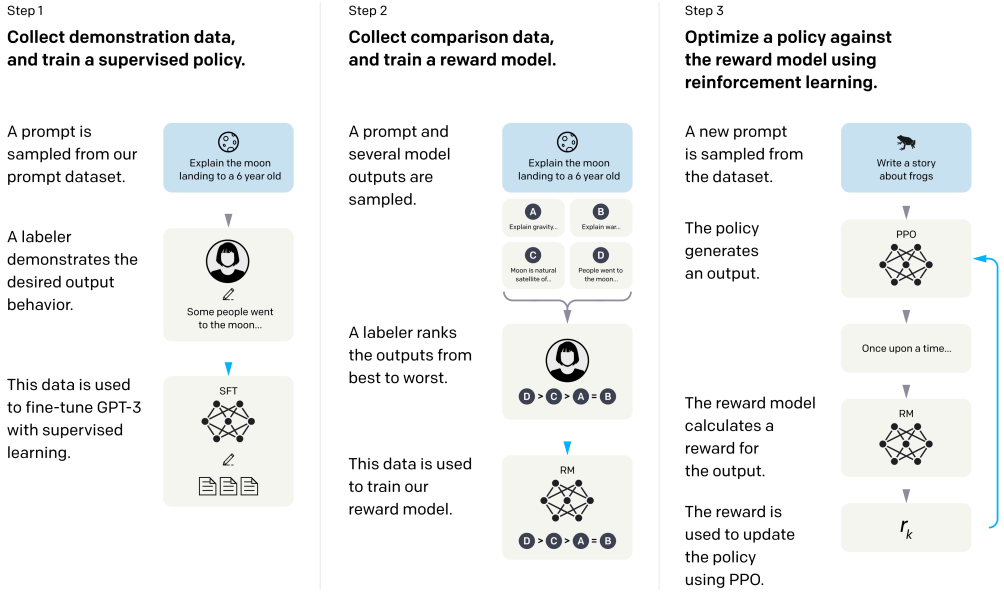


Figure 12.3: The three steps used in training InstructGPT [60, Figure 2].

The predecessor of ChatGPT is called InstructGPT [60].
A model is called aligned if it is

- helpful,
- honest, and
- harmless.

Figure 12.3 shows the three steps used in training InstructGPT.
Table 12.1 shows the criteria given to the labelers.

12.4 Proximal Policy Optimization (PPO)

The third step in training InstructGPT and ChatGPT uses reinforcement learning based on the reward model found in the second step (see Figure 12.3). More precisely, proximal policy optimization (PPO), which is a policy-gradient method, is used.

Whenever local optima of smooth functions are sought, the gradient of the objective function is instrumental in devising efficient optimization algorithms [61, Chapter 12].

Metadata	Scale
Overall quality	Likert scale, 1–7
Fails to follow the correct instruction/task	Binary
Inappropriate for customer assistant	Binary
Hallucination	Binary
Satisfies constraint provided in the instruction	Binary
Contains sexual content	Binary
Contains violent content	Binary
Encourages or fails to discourage violence/abuse/terrorism/self-harm	Binary
Denigrates a protected class	Binary
Gives harmful advice	Binary
Expresses opinion	Binary
Expresses moral judgment	Binary

Table 12.1: The criteria the human labelers used to judge model output [60, Table 3].

In reinforcement learning, gradient methods can be applied to the action-value function or to the policy directly or to both. Policy-gradient methods are based on the policy-gradient theorem (see Section 8.3).

PPO is a family of policy-gradient methods[62]. The algorithms alternate between sampling epochs of minibatches from the environment and optimizing a surrogate objective function using stochastic gradient ascent. According to [62], PPO methods share some of the benefits of trust-region policy optimization (TRPO), are simpler to implement, and empirically have better sample complexity.

Policy-gradient methods maximize the performance

$$J_{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t],$$

where $\hat{A}_t(s,a)$ is the estimate of the advantage function

$$A_t(s,a) := Q_t(s,a) - V_t(s).$$

The approximation $\hat{\mathbb{E}}$ of the expectation is the sample mean based on a finite batch of samples. This objective function, i.e., the performance, leads to the estimate

$$\hat{g}(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t],$$

of the gradient.

It is not well-justified to perform multiple optimization steps on the performance J_{PG} using the same trajectory and empirically it often leads to destructively large updates [62].

TRPO is a trust-region method [63]. It uses the surrogate objective function

$$\begin{aligned} &\text{maximize}_\theta \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \\ &\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta, \end{aligned}$$

which is maximized subject to the constraint on the size of the update, i.e., the difference between the old and new policies. Here KL denotes the Kullback-Leibler divergence

$$\text{KL}(P, Q) := \int p(x) \log \left(\frac{p(x)}{q(x)} \right) \mathrm{d}x$$

of two continuous random variables P and Q, where p and q are their densities.

Alternatively, instead of a constraint, a penalty can be used, which results in the unconstrained optimization problem

$$\text{maximize}_\theta \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t - \beta \, \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right], \tag{12.1}$$

where β is a non-negative coefficient. Unfortunately, reasonable choices of β vary over the course of learning and among different learning problems.

Next, we define the clipped surrogate objective [62]. We note the probability ratio between the old and the new policies by

$$r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}.$$

TRPO maximizes the objective

$$J_{\text{CPI}}(\theta) := \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t[r_t(\theta) \hat{A}_t].$$

The index refers to conservative policy iteration[1].

[1] [S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". In: ICML. Vol. 2. 2002, pp. 267–274]

In order to limit the size of policy updates, the objective function is now modified such that changes in $r_t(\theta)$ away from $r(\theta_{\text{old}}) = 1$ are penalized. In [62], the performance

$$J_{\text{CLIP}}(\theta) := \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

is hence proposed as the main objective function. The clip function ensures that the value $r_t(\theta)$ remains in the interval $[1-\epsilon, 1+\epsilon]$. Taking the minimum of the clipped and unclipped values ensures that the performance is a lower or pessimistic bound on the unclipped value. In other words, changes in the probability ratio $r_t(\theta)$ are included when they make the objective worse and are ignored when they would improve the objective.

Next, we discuss how the hyperparameter β in the penalized, unconstrained optimization problem (12.1) can be adapted dynamically [62]. The main idea is to adapt the penalty coefficient β so that a prescribed target value d_{targ} of the KL divergence is achieved.

We denote the expectation in (12.1) by $J_{\text{KLPEN}}(\theta)$. It is maximized using several epochs of minibatches. Then the difference

$$d := \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$$

between the old and new policies is computed, and the parameter β may be updated according to its value: if $d < d_{\text{targ}}/1.5$, then the updated penalty parameter is $\beta \leftarrow \beta/2$; if $d > d_{\text{targ}} \cdot 1.5$, then it becomes $\beta \leftarrow \beta \cdot 2$. The new value of β is used for the next policy updated, and so on. The parameters 1.5 and 2 are chosen heuristically.

It was found in [62] that J_{KLPEN} performs worse than J_{CLIP}.

Finally, we can formulate the main PPO algorithm. It combines three terms in its performance

$$J_{\text{CLIP+VS+S}}(\theta) := \hat{\mathbb{E}}_t \left[J_{\text{CLIP}} - c_1 J_{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

to be maximized, where c_1 and c_2 are non-negative coefficients. The second term is the squared-error loss

$$J_{\text{VF}} := \left(V_\theta(s_t) - V_t^{\text{target}} \right)^2,$$

and the third term S is the entropy. The entropy is added as a reward in order to ensure sufficient exploration.

The policy-gradient implementation is often based on segments of episodes [2], which is also well-suited for use with recurrent neural networks. The policy

[2] [V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: arXiv preprint arXiv:1602.01783 (2016).]

is run for T time steps, where T is much less than the episode length, and this truncated segment of an episode is used as the sample for the policy-gradient update. This approach means that the estimator of the advantage function only uses T time steps. A very general choice is

$$\hat{A}_t := \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$
$$\delta_t := r_t + \gamma V(s_{t+1}) - V(s_t).$$

The proximal policy optimization (PPO) algorithm is shown in Algorithm 20.

Algorithm 20 Proximal policy optimization (PPO) for policy optimization [62].

loop iterations
  for all actors from 1 to N **do**
    run policy $\pi_{\theta_{\text{old}}}$ for T time steps
    compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end for
  optimize surrogate performance/objective $J_{\text{CLIP+VS+S}}$ w.r.t. θ
    using K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} := \theta$
end loop

12.5 Bibliographical and Historical Remarks

Transformers were introduced in [58]. Proximal policy optimization (PPO) was introduced in [62]. InstructGPT was introduced in [60].

12.6 Problems

1. Number of letters. Ask one or preferably more LLM how many letters some words contain and compare the results.

2. Implement a character based tokenizer.

3. Implement an embedding layer.

Appendix A

Analysis

This chapter contains advanced definitions and results from analysis.

A.1 The Riemann-Stieltjes Integral

The Riemann-Stieltjes integral is a generalization of the Riemann integral and a precursor of the Lebesgue integral. It is very useful to uniformly formulate statistical formulae and theorems that apply both to discrete and continuous probabilities.

Definition A.1 (partition (of a real interval))**.** A *partition* of a set X is a set of subsets $S \subset X$ of X such that each subset $S \neq \emptyset$ is non-empty and that every element $x \in X$ is in exactly one of these subsets.

If the set X to be partitioned into a partition P is a real interval $[a, b] \subset \mathbb{R}$, a set of points $x_n \in [a, b]$, $0 \leq n \leq N(P)$, such that

$$a = x_0 < x_1 < \cdots < x_{N(P)} = b$$

gives rise to

$$P = \{[x_0, x_1), [x_1, x_2), [x_2, x_3), \ldots, [x_{N(P)-1}, x_{N(P)}]\},$$

the *partition of a real interval.* The number of points is $N(P)+1$, and the mesh $|P|$ of a partition P of this form is defined to be the length

$$|P| := \max_{n=1}^{N(P)} |x_n - x_{n-1}|$$

of its longest subinterval.

Definition A.2 (total variation)**.** Suppose $[a,b] \subset \mathbb{R}$ and $f\colon [a,b] \to \mathbb{R}$ is a real valued function. Then the quantity

$$V_a^b(f) := \sup_{P \in \mathcal{P}} \sum_{n=1}^{N(P)} |f(x_n) - f(x_{n-1})|$$

is called the *total variation* of f, where the supremum is taken over all partitions $\mathcal{P}$ of the real interval $[a,b]$.

Definition A.3 (bounded variation)**.** Suppose $[a,b] \subset \mathbb{R}$ and $f\colon [a,b] \to \mathbb{R}$ is a real valued function. The function f is said to be of *bounded variation,* i.e., $f \in \mathrm{BV}([a,b])$, if its total variation is finite, i.e., $V_a^b(f) < \infty$.

It can be shown that if a real valued function $f\colon [a,b] \to \mathbb{R}$ is differentiable and its derivative f' is Riemann-integrable, then its total variation is given by

$$V_a^b(f) = \int_a^b |f'(x)| \mathrm{d}x,$$

which is the vertical component of the arc length of its graph.

Furthermore, the Jordan decomposition of a function means that a real function $f\colon \mathbb{R} \to \mathbb{R}$ is of bounded variation in $[a,b]$ if and only if it can be written as the difference $f = f_1 - f_2$ of two non-decreasing functions real functions f_1 and f_2 on $[a,b]$.

Definition A.4 (Riemann-Stieltjes integral)**.** Suppose a partition P of the real interval $[a,b] \subset \mathbb{R}$ is given by the points $\{x_0, \dots, x_{N(P)}\}$. Then the approximating sum S of a Riemann-Stieltjes integral is defined as

$$S(P,f,g) := \sum_{n=1}^{N(P)} f(c_n)(g(x_n) - g(x_{n-1})),$$

where $c_n \in [x_{n-1}, x_n]$ for all $n \in \{1, \dots, N(P)\}$, and the limit

$$\int_{x=a}^{b} f(x) \mathrm{d}g(x) := \lim_{|P| \to 0} S(P,f,g),$$

if it exists, is called the *Riemann-Stieltjes integral* of the integrand $f\colon \mathbb{R} \to \mathbb{R}$ on the interval $[a,b]$ with respect to the integrator $g\colon \mathbb{R} \to \mathbb{R}$, which is assumed to be of bounded variation. The limit is defined to be equal to $S \in \mathbb{R}$, if for every $\epsilon \in \mathbb{R}^+$ there exists a $\delta \in \mathbb{R}^+$ such that for every partition P with mesh $|P| < \delta$ and for every choice of points $c_n \in [x_{n-1}, x_n]$, the inequality

$$|S(P,f,g) - S| < \epsilon$$

holds.

Theorem A.5 (integration by parts)**.** *Suppose the real valued functions $f\colon \mathbb{R} \to \mathbb{R}$ and $g\colon \mathbb{R} \to \mathbb{R}$ are of bounded variation. Then the existence of either Riemann-Stieltjes integral $\int_{x=a}^{b} f(x)\mathrm{d}g(x)$ or $\int_{x=a}^{b} g(x)\mathrm{d}f(x)$ implies the existence of the other. If the integrals exist, then the formula*

$$\int_{x=a}^{b} f(x)\mathrm{d}g(x) = f(b)g(b) - f(a)g(a) - \int_{x=a}^{b} g(x)\mathrm{d}f(x),$$

called integration by parts, *holds.*

If the integrator g is smooth enough, then the Riemann-Stieltjes integral reduces to a Riemann integral.

Theorem A.6 (relation to Riemann integral)**.** *Suppose $f\colon \mathbb{R} \to \mathbb{R}$ is bounded on the interval $[a, b] \in \mathbb{R}$, $g\colon \mathbb{R} \to \mathbb{R}$ increases monotonically, and g' is Riemann-integrable. Then the Riemann-Stieltjes integral of f with respect to g is given by the Riemann integral of fg', i.e.,*

$$\int_{x=a}^{b} f(x)\mathrm{d}g(x) = \int_{x=a}^{b} f(x)g'(x)\mathrm{d}x.$$

Suppose the integrator is a unit-step function. Then the Riemann-Stieltjes integral of an integrand (that is continuous at the step) with respect to the step function equals the integrand evaluated at the unit step.

Theorem A.7 (unit-step function as integrator)**.** *Suppose $f\colon \mathbb{R} \to \mathbb{R}$ is continuous at $\xi \in (a, b) \subset \mathbb{R}$ and H_ξ is the unit-step function*

$$H_\xi\colon \mathbb{R} \to \mathbb{R}, \quad H_\xi(x) := \begin{cases} 0, & x < \xi, \\ 1, & x \geq \xi. \end{cases}$$

Then the Riemann-Stieltjes integral of f with respect to H_ξ is

$$\int_{x=a}^{b} f(x)\mathrm{d}H_\xi(x) = f(\xi).$$

More generally, the following theorem holds for piecewise constant functions as integrators.

Definition A.8 (piecewise constant function)**.** A *piecewise constant function* $g\colon \mathbb{R} \to \mathbb{R}$ is a function that has finitely many points of discontinuity $x_1 < \cdots < x_N$, $N \in \mathbb{N}$, and that is constant on the intervals $(-\infty, x_1)$, (x_1, x_2), …, (x_{N-1}, x_N), (x_N, ∞).

In this definition, the values of g at the points x_n of discontinuity are arbitrary and not necessarily equal to $g(x_n-)$ or $g(x_n+)$. We write

$$g(\xi-) := \lim_{\substack{x\in(-\infty,\xi),\\ x\to\xi}} g(x),$$
$$g(\xi+) := \lim_{\substack{x\in(\xi,+\infty),\\ x\to\xi}} g(x).$$

Theorem A.9 (piecewise constant function as integrator)**.** *Suppose $g\colon \mathbb{R}\to\mathbb{R}$ is a piecewise constant function, and $f\colon \mathbb{R}\to\mathbb{R}$ is continuous at the points of discontinuity $x_1 < \cdots < x_N$, $N\in\mathbb{N}$. Then the Riemann-Stieltjes integral of f with respect to g is*

$$\int_{x=-\infty}^{\infty} f(x)\mathrm{d}g(x) = \sum_{n=1}^{N} f(x_n)(g(x_n+) - g(x_n-)).$$

Integrators that are piecewise constant functions are the link between continuous and discrete random variables and probability distributions. Here we consider the expectation as the leading example of an integral (in the case of a continuous probability distribution) or a summation (in the case of a discrete probability distribution), but the principle applies in general. The Riemann-Stieltjes integral provides a unified point of view of the integrals arising in the continuous case and the summations arising in the discrete case, where the discrete case becomes a special case of the continuous one.

Suppose F_X is the cumulative probability distribution of a random variable X. As such, F_X is of bounded variation, which means that it can always serve as the integrator of a Riemann-Stieltjes integral. By definition, the expectation of $h(X)$ is

$$\mathbb{E}[h(X)] := \int_{x=-\infty}^{\infty} h(x)\mathrm{d}F_X(x).$$

If the cumulative probability distribution F_X is continuous differentiable, the corresponding probability density function is $f_X := F_X'$, and we have

$$\mathbb{E}[h(X)] = \int_{x=-\infty}^{\infty} h(x)f_X(x)\mathrm{d}x,$$

which can be viewed as a Riemann or a Lebesgue integral and is often used as the definition of the expectation.

However, if the random variable X does not have a probability density function with respect to the Lebesgue measure, this formula does not hold. The

leading examples are discrete random variables, i.e., random variables whose probabilities are assigned to a finite number of real points.

When using Riemann-Stieltjes integrals, how do discrete random variables and distributions become special cases of continuous ones? Suppose that the sample space is the finite set $\Omega := \{x_1, \dots, x_N\}$, $N \in \mathbb{N}$, such that $x_1 < \dots < x_N$, and that the probabilities of the outcomes of the random variable X are

$$\mathbb{P}[X = x_n] := p_n \in [0,1], \qquad n \in \{1, \dots, N\},$$

such that $\sum_{n=1}^{N} p_n = 1$. The cumulative distribution function of X is

$$F_X(x) := \mathbb{P}[X \le x] = \begin{cases} 0, & x < x_1, \\ \sum_{i=1}^{n} p_i, & x_n \le x < x_{n+1}, \\ 1, & x_N \le x, \end{cases}$$

where we have only used the general definition of a cumulative distribution function. We consider the expectation of $h(X)$ as a leading example of an integral and find

$$\begin{aligned} \mathbb{E}[h(X)] &:= \int_{x=-\infty}^{\infty} h(x) \mathrm{d}F_X(x) \\ &= \sum_{n=1}^{N} h(x_n)(F_X(x_n+) - F_X(x_n-)) \\ &= \sum_{n=1}^{N} h(x_n) p_n \end{aligned}$$

by Theorem A.9. The last expression is commonly used as the special definition of the expectation in the discrete case, but using the Riemann-Stieltjes integral it immediately follows from the general definition.

A.2 The Banach Fixed-Point Theorem

Definition A.10 (contraction, Lipschitz constant)**.** Suppose (X, d) is a metric space. A map $T\colon X \to X$ is called a *contraction* on X if

$$\exists \lambda \in [0,1)\colon \quad \forall (x,y) \in X \times X\colon \quad d(T(x), T(y)) \le \lambda d(x,y).$$

Any such value of λ is called a *Lipschitz constant* for T.

Clearly, every contraction is a continuous function.

Theorem A.11 (Banach fixed-point theorem)**.** *Suppose (X, d) is a non-empty complete metric space and $T\colon X \to X$ is a contraction with constant λ. Then T has a unique fixed point x_*, i.e.,*

$$x_* = T(x_*).$$

Furthermore, define a sequence $\langle x_n \rangle_{n\in\mathbb{N}}$ by

$$x_{n+1} := T(x_n), \qquad n \in \mathbb{N},$$

and $x_0 \in X$ chosen arbitrarily. Then this sequence converges to the unique fixed point, i.e.,

$$\lim_{n\to\infty} x_n = x_*.$$

Furthermore, the inequalities

$$\begin{aligned} d(x_*, x_{n+1}) &\le \lambda d(x_*, x_n) && \forall n \in \mathbb{N}, \\ d(x_*, x_n) &\le \frac{\lambda^n}{1-\lambda} d(x_1, x_0) && \forall n \in \mathbb{N}, \\ d(x_*, x_{n+1}) &\le \frac{\lambda}{1-\lambda} d(x_{n+1}, x_n) && \forall n \in \mathbb{N} \end{aligned}$$

hold.

The following generalizations can be shown.

Theorem A.12 (generalizations)**.** *Suppose (X, d) is a non-empty complete metric space and $T\colon X \to X$ is a function.*

1. *Suppose that an iterate T^n of T is a contraction. Then T has a unique fixed point.*
2. *Suppose that for all $n \in \mathbb{N}$ there exists a $c_n \in \mathbb{R}$ such that $d(T^n(x), T^n(y)) \le c_n d(x, y)$ for all $x \in X$ and $y \in X$ and that $\sum_{n\in\mathbb{N}} c_n < \infty$. Then T has a unique fixed point.*
3. *Suppose $d(T(x), T(y)) < d(x, y)$ for all distinct $x \in X$ and $y \in X$. Then T has a unique fixed point.*

A.3 Exercises

Exercise A.1. Show Theorem A.5.

Exercise A.2. Show Theorem A.6.

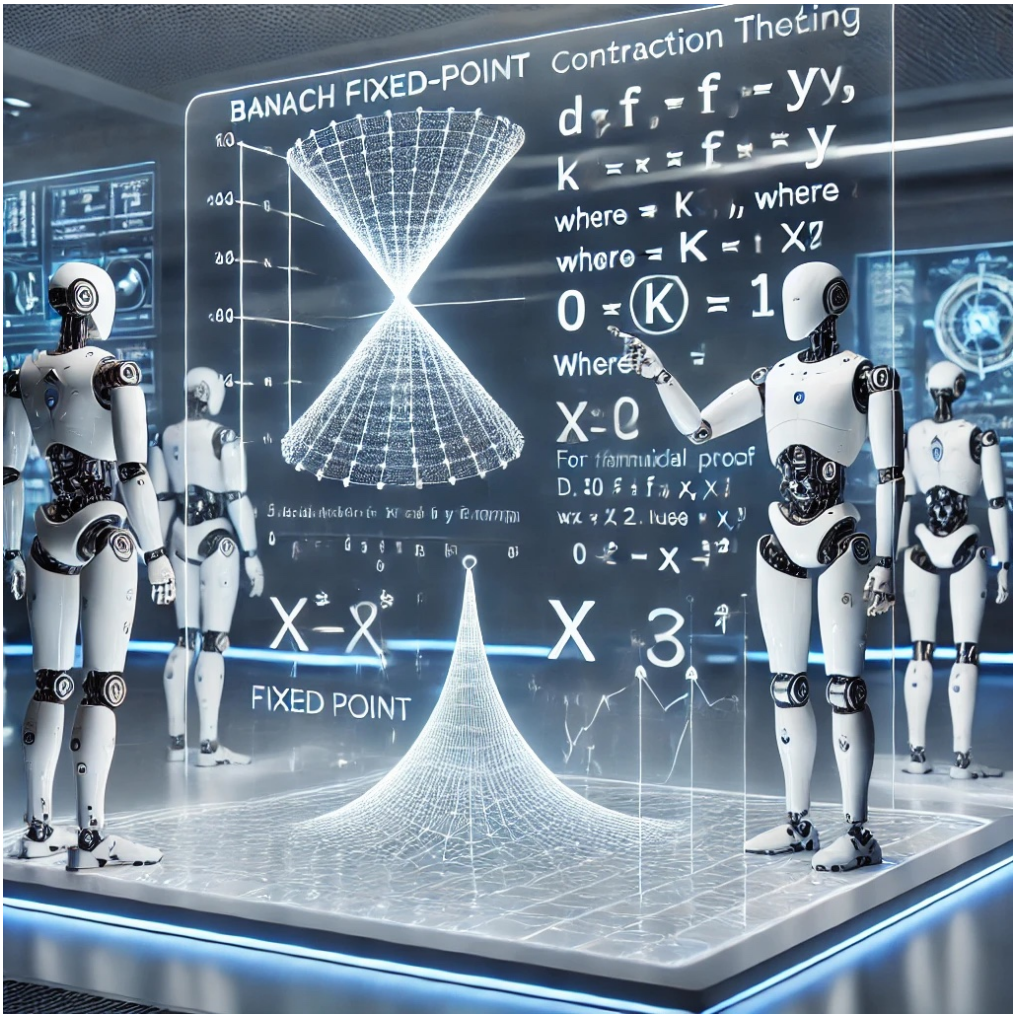
Exercise A.3. Show Theorem A.7.

Exercise A.4. Show Theorem A.9.

Exercise A.5. Define all terms used in Section A.2.

Exercise A.6. Prove Theorem A.11.

Exercise A.7 (generalizations)**.** Prove Theorem A.12.



Appendix B

Measure and Probability Theory

This chapter contains definitions and results that serve as the interface between measure and probability theory on the one hand and RL on the other hand.

B.1 Notation

The following notation is used throughout the book. The first definition concerns ranges of integers.

Definition B.1 (range)**.** The set $\{m, \dots, n\}$ of all integers between m and n is denoted by $[m{:}n]$.

The second notation assigns one and zero to true and false statements, respectively.

Definition B.2 (Iverson bracket)**.** The *Iverson bracket* of a statement is defined as

$$[\![\text{statement}]\!] := \begin{cases} 1 & \text{if statement is true,} \\ 0 & \text{if statement is false.} \end{cases}$$

B.2 Measures and Measure Spaces

In measure and probability theory, it is often convenient to augment the real numbers by $+\infty = \infty$ and $-\infty$. In order to save space, the arithmetic operations on and the algebraic properties of the extended real numbers are not discussed here.

Definition B.3 (extended real numbers)**.** The *extended real numbers* are the set $\mathbb{R} \cup \{-\infty, +\infty\} = [-\infty, +\infty]$.

The concept of a σ-algebra is fundamental for the following definitions. In the following, sets of sets and σ-algebras in particular are denoted by calligraphic letters.

Definition B.4 (σ-algebra)**.** Suppose Ω is a non-empty set. Then a subset $\mathcal{F} \subset \mathcal{P}(\Omega)$ of its power set $\mathcal{P}(\Omega)$ is called a *σ-algebra over the universal set* Ω if it satisfies the following properties:

1. The set $\mathcal{F}$ contains the universal set Ω, i.e., $\Omega \in \mathcal{F}$.

2. The set $\mathcal{F}$ is closed under complements, i.e., if $A \in \mathcal{F}$, then also $\Omega \setminus A \in \mathcal{F}$.

3. The set $\mathcal{F}$ is closed under countable unions, i.e., if $A_i \in \mathcal{F}$ for all $i \in \mathbb{N}$, then also $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

Since the σ-algebra $\mathcal{F}$ is closed under complements and countable unions, $\mathcal{F}$ is also closed under countable intersections, i.e., if $A_i \in \mathcal{F}$ for all $i \in \mathbb{N}$, then also $\bigcap_{i=1}^{\infty} A_i \in \mathcal{F}$, by De Morgan's law.

Given a universal set Ω, the set $\{\emptyset, \Omega\}$, also called the trivial σ-algebra, is the smallest possible σ-algebra. The power set $\mathcal{P}(\Omega)$ is the largest possible σ-algebra over Ω. The smallest σ-algebra that contains a subset $A \subset \Omega$ is $\{\emptyset, A, \Omega \setminus A, \Omega\}$.

The most common used σ-algebra over the real numbers is the Borel σ-algebra over the real numbers. In order to define Borel σ-algebras, we need the definitions of a topological space and a σ-operator.

Definition B.5 (topological space, topology, open set)**.** A *topological space* is a pair $(\Omega, \mathcal{O})$ where Ω is a set and the *topology* $\mathcal{O}$ is a set of subsets of Ω, called the *open sets*, that satisfy the following properties:

1. The empty set and the set Ω are elements of the topology $\mathcal{O}$, i.e., $\emptyset \in \mathcal{O}$ and $\Omega \in \mathcal{O}$.

2. Any (finite or infinite) union of elements of the topology $\mathcal{O}$ is an element of the topology $\mathcal{O}$, i.e., any (finite or infinite) union of open sets is again an open set.

3. The intersection of any finite number of elements of the topology $\mathcal{O}$ is an element of the topology $\mathcal{O}$, i.e., any intersection of a finite number of open sets is again an open set.

Definition B.6 (σ-operator, generator)**.** Suppose that Ω is a set and that the *generator* $\mathcal{M}$ is a subset of its power set $\mathcal{P}(\Omega)$. Then the *σ-operator* is defined as

$$\sigma(\mathcal{M}) := \bigcap_{\mathcal{A} \in \mathcal{F}(\mathcal{M})} \mathcal{A},$$

where

$$\mathcal{F}(\mathcal{M}) := \{\mathcal{A} \subset \mathcal{P}(\Omega) : \mathcal{M} \subset \mathcal{A} \wedge \mathcal{A} \text{ is a } \sigma\text{-algebra}\}.$$

The set $\mathcal{F}(\mathcal{M})$ contains all σ-algebras that contain $\mathcal{M}$. Since the intersection of σ-algebras is again a σ-algebra, the set $\sigma(\mathcal{M})$ of subsets of Ω is the smallest σ-algebra that contains $\mathcal{M} \subset \mathcal{P}(\Omega)$. The set $\sigma(\mathcal{M})$ is uniquely determined and it is called the σ-algebra generated by $\mathcal{M}$.

Definition B.7 (Borel σ-algebra, Borel sets)**.** Suppose $(\Omega, \mathcal{O})$ is a topological space. The σ-algebra $\mathcal{B}((\Omega, \mathcal{O})) := \sigma(\mathcal{O})$ generated by the σ-operator applied to the open sets $\mathcal{O}$ is called the *Borel σ-algebra* over Ω. If the open sets $\mathcal{O}$ are implicitly known, it is customary to write $\mathcal{B}(\Omega)$ for $\mathcal{B}((\Omega, \mathcal{O}))$. The elements of a Borel σ-algebra are called *Borel sets*.

By the definition of the σ-operator and the discussion above, the Borel σ-algebra is the smallest σ-algebra that contains all open sets $\mathcal{O}$ given a topological space $(\Omega, \mathcal{O})$.

The canonical topological space $(\mathbb{R}, \mathcal{O})$ over the real numbers is the one whose topology $\mathcal{O}$ consists of the open intervals (a, b) with rational endpoints $a, b \in \mathbb{Q}$. The Borel σ-algebra $\mathcal{B}((\mathbb{R}, \mathcal{O}))$ thus generated does not contain all subsets of $\mathbb{R}$; in fact, it can be shown that $\mathbb{R}$ and $\mathcal{B}((\mathbb{R}, \mathcal{O}))$ are equinumerous, while the power set of $\mathbb{R}$ has a larger cardinality than $\mathbb{R}$.

Since it is the most common one, the Borel σ-algebra $\mathcal{B}((\mathbb{R}, \mathcal{O}))$ is usually simply called the Borel σ-algebra over $\mathbb{R}$ and denoted by $\mathcal{B}(\mathbb{R})$.

As noted above, a Borel σ-algebra $\sigma(\mathcal{M})$ is uniquely determined by its generator $\mathcal{M}$; however, different generators may generate the same Borel σ-algebra. The Borel σ-algebra $\mathcal{B}(\mathbb{R})$ is generated by

$$\mathcal{M}_0 := \{A \subset \mathbb{R} : A \in \mathcal{O}\}, \tag{B.1a}$$
$$\mathcal{M}_1 := \{[a, b] \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a \leq b\}, \tag{B.1b}$$
$$\mathcal{M}_2 := \{[a, b] \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a \leq b\}, \tag{B.1c}$$
$$\mathcal{M}_3 := \{(a, b) \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a < b\}, \tag{B.1d}$$
$$\mathcal{M}_4 := \{(a, b) \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a < b\}, \tag{B.1e}$$
$$\mathcal{M}_5 := \{(a, b] \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a \leq b\}, \tag{B.1f}$$
$$\mathcal{M}_6 := \{(a, b] \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a \leq b\}, \tag{B.1g}$$

$$\mathcal{M}_7 := \{(-\infty, a] \subset \mathbb{R} : a \in \mathbb{R}\}, \tag{B.1h}$$
$$\mathcal{M}_8 := \{(-\infty, a] \subset \mathbb{R} : a \in \mathbb{Q}\}, \tag{B.1i}$$
$$\mathcal{M}_9 := \{(-\infty, a) \subset \mathbb{R} : a \in \mathbb{R}\}, \tag{B.1j}$$
$$\mathcal{M}_{10} := \{(-\infty, a) \subset \mathbb{R} : a \in \mathbb{Q}\}. \tag{B.1k}$$

When working with cumulative distribution functions, the generators $\mathcal{M}_7$, $\mathcal{M}_8$, $\mathcal{M}_9$, and $\mathcal{M}_{10}$ are most useful and can be used with rational endpoints together with approximation arguments.

Next, we define measures and measure spaces.

Definition B.8 (measurable space)**.** A *measurable space* is a pair $(\Omega, \mathcal{F})$ consisting of a non-empty set Ω and a σ-algebra $\mathcal{F}$ over Ω.

Definition B.9 (measurable function)**.** Suppose $(\Omega, \mathcal{F})$ and $(\Psi, \mathcal{G})$ are measurable spaces. An $(\mathcal{F}, \mathcal{G})$-*measurable function* from $(\Omega, \mathcal{F})$ to $(\Psi, \mathcal{G})$ is a function $X \colon \Omega \to \Psi$ such that $X^{-1}(G) \in \mathcal{F}$ for every $G \in \mathcal{G}$.

The set of measurable functions is closed under algebraic operations. It is also closed under the pointwise sequential limits

$$\liminf_{n\to\infty} f_n, \quad \limsup_{n\to\infty} f_n, \quad \sup_{n\to\infty} f_n,$$

i.e., these limits are measurable if the functions f_n in the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ are measurable.

Definition B.10 (measure)**.** Suppose $(\Omega, \mathcal{F})$ is a measurable space. A function $\mu \colon \mathcal{F} \to [0, \infty]$ is called a *measure* if it satisfies the following properties:

1. The measure of the empty set is zero, i.e., $\mu(\emptyset) = 0$.
2. The function μ is countably additive (or σ-additive), i.e., if $\{A_i\}_{i=1}^{\infty} \subset \mathcal{F}$ is a countable set of pairwise disjoint sets $A_i \in \mathcal{F}$, then

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i).$$

Definition B.11 (measure space)**.** A *measure space* is a triple $(\Omega, \mathcal{F}, \mu)$ such that $(\Omega, \mathcal{F})$ is a measurable space and the function μ is a measure on $(\Omega, \mathcal{F})$.

The push-forward measure transfers or pushes forward a measure from one measurable space to another by using a measurable function.

Definition B.12 (push-forward measure)**.** Suppose that $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ are measurable spaces, that $f\colon \Omega_1 \to \Omega_2$ is a measurable function, and that $\mu\colon \Omega_1 \to [0, \infty]$ is a measure. Then the *push-forward measure* of μ is defined to be the measure

$$\mu \circ f^{-1}\colon \Omega_2 \to [0, \infty],$$

also written as

$$f\#\mu := \mu \circ f^{-1}.$$

The push-forward measure has an application in the well-known change-of-variables formula

$$\int_{\Omega_1} g \circ f \mathrm{d}\mu = \int_{\Omega_2} g \mathrm{d}(f\#\mu).$$

The integrals here are Lebesgue integrals (see Section B.3).

In measure and probability theory as well as in integration, statements which are true everywhere except on sets whose measures vanishe often occur. This motivates the following definitions.

Definition B.13 (null set)**.** Suppose $(\Omega, \mathcal{F}, \mu)$ is a measure space. A set $N \in \mathcal{F}$ is called a *null set* with respect to the measure μ, if $\mu(N) = 0$.

Definition B.14 (almost all/everywhere/surely)**.** Suppose $(\Omega, \mathcal{F}, \mu)$ is a measure space. A statement is said to be true for *almost all* $x \in \Omega$ or *almost everywhere/surely in* Ω, if there exists a null set N with respect to μ such that the statement is true for all $x \in \Omega \setminus N$. In this case, we write

$$(\forall_\mu x \in \Omega\colon \text{statement}) \quad :\Longleftrightarrow \quad (\exists N \in \mathcal{F}\colon\ \mu(N) = 0\ \wedge\ (\forall x \in \Omega \setminus N\colon \text{statement})).$$

B.3 The Lebesgue Integral

The Lebesgue integral plays an essential role in measure and probability theory. In very general terms, the domain of the function to be integrated is partitioned in order to define and to compute a Riemann (or Riemann-Stieltjes) integral of the function, while the range of the function to be integrated is partitioned in the Lebesgue integral. The Lebesgue integral interacts better with taking limits of sequences of functions; results for this situation are collected in Section B.5. Furthermore, the Lebesgue integral is defined for a larger class of functions than the Riemann integral. For example, the Dirichlet function

$$\mathbb{R} \to \{0, 1\}, \quad x \mapsto [\![x \in \mathbb{Q}]\!]$$

is not Riemann integrable, but it is Lebesgue integrable (and has Lebesgue integral zero).

In the following, two ways of constructing and defining the Lebesgue integral will be discussed. The first (in Section B.3.1) is based on partitions of the ranges of the functions and reduces the Lebesgue integral to a Riemann integral. The second (in Section B.3.2) is based on so-called simple functions, which provide discretizations of the areas under the graphs of the functions.

B.3.1 Construction and Definition Using the Riemann Integral

As mentioned above, in Lebesgue integration, the classical method of exhaustion can be applied to horizontal slices that partition the range of the function.

We start with a measure space $(\Omega, \mathcal{F}, \mu)$ (see Definition B.11). We recall that by Definition B.9 measurable functions defined on this measure space are functions such that the preimages of all sets in the σ-algebra of the image (measurable) space are in the σ-algebra $\mathcal{F}$ of the preimage (measurable) space Ω. In particular, the preimages

$$\{x \in \Omega \mid f(x) \geq y\}, \quad y \in \mathbb{R},$$

of measurable real valued functions f are elements of the σ-algebra $\mathcal{F}$. In other words, the measure μ assigns the lengths

$$\mu(\{x \in \Omega \mid f(x) \geq y\})$$

to the preimages, which we will use in the following.

We consider a non-negative measurable real valued function $f \colon \mathbb{R} \to \mathbb{R}^+$ in the (x, y)-plane. At each point y in the range of f, thin horizontal slices between y and $y - \mathrm{d}y$ contribute to the integral wherever the function value $f(x)$ is greater equal y such that these slices lie between the x-axis and the graph of the function. The area that is contributed to the integral at any point y in the range of f is hence given by

$$\mu(\{x \in \mathbb{R} \mid f(x) \geq y\})\mathrm{d}y.$$

Summing these contributions as a Riemann integral yields the following definition.

Definition B.15 (Lebesgue integral (via Riemann integral))**.** Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space and that $f \colon \mathbb{R} \to \mathbb{R}^+$ is non-negative measurable function. Its *Lebesgue integral* is then defined by

$$\int_\Omega f \mathrm{d}\mu := \int_0^\infty \mu(\{x \in \mathbb{R} \mid f(x) \geq y\})\mathrm{d}y, \tag{B.2}$$

where the integral on the right-hand side is an improper Riemann integral.

In this definition, $\mu(\{x \in \mathbb{R} \mid f(x) \geq y\})$ can be replaced by $\mu(\{x \in \mathbb{R} \mid f(x) > y\})$.

The following theorem is well-known in Riemann integration.

Theorem B.16. *Suppose $f\colon [a,b] \to \mathbb{R}$ is monotone. Then f is Riemann integrable on the interval $[a,b]$.*

Since the integrand on the right-hand side in (B.2) is a non-negative and monotone decreasing function, the improper Riemann integral exists and has a value in the interval $[0,\infty)$.

In the next step, we extend these considerations from non-negative functions to signed functions. If f is a measurable function to the extended real numbers, we define

$$f^+(x) := \begin{cases} f(x), & f(x) > 0, \\ 0, & f(x) \leq 0, \end{cases}$$
$$f^-(x) := \begin{cases} -f(x), & f(x) < 0, \\ 0, & f(x) \geq 0, \end{cases}$$

both of which are non-negative and measurable. With these definitions, we have

$$f = f^+ - f^-,$$
$$|f| = f^+ + f^-,$$

which allows us to reduce the Lebesgue integrals of signed functions f to the Lebesgue integrals $\int f^+ \mathrm{d}\mu$ and $\int f^- \mathrm{d}\mu$ of the non-negative functions f^+ and f^-.

Definition B.17 (Lebesgue integral of a measurable function)**.** Suppose f is a measurable function to the extended real numbers. If the Lebesgue integral $\int f^+ \mathrm{d}\mu$ or the Lebesgue integral $\int f^- \mathrm{d}\mu$ exists and is finite, i.e.,

$$\int_\Omega f^+ \mathrm{d}\mu < \infty \quad \vee \quad \int_\Omega f^- \mathrm{d}\mu < \infty,$$

then

$$\int_\Omega f \mathrm{d}\mu := \int_\Omega f^+ \mathrm{d}\mu - \int_\Omega f^- \mathrm{d}\mu$$

is called the *Lebesgue integral* of f.

Lebesgue integrals have values in the extended real numbers. A function is called Lebesgue integrable, if the area between its graph and the x-axis is finite as expressed in the following definition.

Definition B.18 (Lebesgue integrable function)**.** Suppose f is a measurable function to the extended real numbers defined on a measure space $(\Omega, \mathcal{F}, \mu)$. It is called *Lebesgue integrable* with respect to μ if the integral

$$\int_\Omega |f| \mathrm{d}\mu < \infty$$

of its absolute value is finite.

The following theorem gives some important classes of Lebesgue measurable functions including Riemann integrable ones.

Theorem B.19 (some classes of Lebesgue measurable functions)**.** *Continuous functions, semicontinuous functions, step functions, monotone functions, Riemann integrable functions, and functions of bounded variation are Lebesgue measurable.*

Complex valued functions are integrated by considering the real and imaginary parts separately. If $f = f_1 + \mathrm{i} f_2$ for real valued integrable functions f_1 and f_2, then the Lebesgue integral of f is defined by

$$\int_\Omega f \mathrm{d}\mu := \int_\Omega f_1 \mathrm{d}\mu + \mathrm{i} \int_\Omega f_2 \mathrm{d}\mu. \tag{B.3}$$

Furthermore, the function f is Lebesgue integrable if and only if its absolute value is Lebesgue integrable.

The Riemann integral with respect to an orientation is defined as $\int_b^a f := -\int_a^b f$. In Lebesgue integration, there is no orientation, since the domains of integration are sets. Still, if the domain is an interval $I := [a, b]$, we can define

$$\int_b^a f \mathrm{d}\mu := -\int_{[a,b]} f \mathrm{d}\mu.$$

B.3.2 Construction and Definition Using Simple Functions

In the previous construction using the Riemann integral, we have used the method of exhaustion in Definition B.15 twice: first for horizontal slices, i.e., a partition of the range of the function, and then for the vertical slices in the Riemann integral. The second construction and definition in this section applies the method of exhaustion to both directions directly via so-called simple functions, which are essentially a discretization of the area between the graph of the function and the x-axis.

Again, we start with a measure space $(\Omega, \mathcal{F}, \mu)$ (see Definition B.11). In order to discretize the area between the graph of the function and the x-axis, we consider the indicator functions

$$\chi_S\colon \quad \Omega \to \{0,1\}, \quad x \mapsto [\![x \in S]\!]$$

of measurable sets S. The only possible value of an integral that is consistent with the measure μ is

$$\int_\Omega \chi_S \mathrm{d}\mu := \mu(S),$$

which may be equal to ∞.

Using this first definition and the notion that the integral should be a linear operator, we extend the definition of the Lebesgue integral to linear combinations of indicator functions, the so-called simple functions.

Definition B.20 (simple function)**.** A *simple function* is a (finite) linear combination

$$\sum_{n=1}^{N} a_n \chi_{S_n},$$

where $a_n \in \mathbb{R}$ for all $n \in [1{:}N]$ and S_n, $n \in [1{:}N]$, are disjoint measurable sets.

Simple functions are measurable.

For non-negative simple functions s, i.e., when the coefficients a_n are non-negative, we define

$$\int_\Omega s \mathrm{d}\mu = \int_\Omega \left(\sum_{n=1}^{N} a_n \chi_{S_n}\right) \mathrm{d}\mu := \sum_{n=1}^{N} a_n \int_\Omega \chi_{S_n} \mathrm{d}\mu = \sum_{n=1}^{N} a_n \mu(S_n)$$

by linearity, which may be equal to ∞. Here the definition $0 \cdot \infty := 0$ is used for the products. By the σ-additivity of the measure μ (see Definition B.10), the integral does not depend on the particular linear combination used to represent the simple function.

If Ψ is a measurable subset of Ω, then we define the integral of a non-negative simple function s on Ψ by

$$\int_\Psi s \mathrm{d}\mu := \int_\Omega \chi_\Psi s \mathrm{d}\mu := \sum_{n=1}^{N} a_n \mu(S_n \cap \Psi).$$

In the next step, we extend the construction of the Lebesgue integral from non-negative simple functions to non-negative measurable functions f which take values in the extended real numbers.

Definition B.21 (Lebesgue integral (via simple functions))**.** Suppose f is a non-negative measurable function on a measurable subset Ψ of a measure space $(\Omega, \mathcal{F}, \mu)$. Then its *Lebesgue integral on* Ψ is defined by

$$\int_\Psi f \mathrm{d}\mu := \sup \left\{ \int_\Psi s \mathrm{d}\mu \;\middle|\; s \text{ is simple } \wedge\ 0 \leq s \leq f \right\}.$$

The value of this integral may be equal to ∞. This definition and the preceding one for non-negative simple functions coincide when non-negative simple functions are integrated. Furthermore, it can be shown that Definition B.15 (via Riemann integrals) and Definition B.21 (via simple functions) coincide.

The Lebesgue integral of signed functions is constructed as in Definition B.17 via the positive and negative parts f^+ and f^-, which are non-negative. Furthermore, the Lebesgue integral of complex valued functions is defined as in (B.3) via their real and imaginary parts.

B.3.3 Properties

We again suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space. For the purposes of measure and probability theory, it is useful to define the equality of two functions as equality almost everywhere/surely, i.e., they are equal if they coincide outside a subset of measure zero.

Definition B.22 (equality of functions)**.** Two functions f and g defined on a measure space $(\Omega, \mathcal{F}, \mu)$ are called *equal* if they are equal almost everywhere/surely, i.e.,

$$f \overset{\mu}{=} g \quad :\Longleftrightarrow \quad \forall_\mu x \in \Omega \colon f(x) = g(x).$$

This equality relation is an equivalence relation. Analogously, we also define $f < g$, $f \leq g$, $f > g$, and $f \geq g$.

With this definition of equality, equal functions have equal integrals if the integrals exist.

Theorem B.23 (integrals of equal functions)**.** *Suppose two functions f and g defined on a measure space $(\Omega, \mathcal{F}, \mu)$ are equal. Then f is Lebesgue integrable if and only if g is Lebesgue integrable, and if their integrals exists, then*

$$f \overset{\mu}{=} g \implies \int_\Omega f \mathrm{d}\mu = \int_\Omega g \mathrm{d}\mu$$

holds.

Theorem B.24 (linearity of the Lebesgue integral)**.** *Suppose that f and g are two Lebesgue integrable functions defined on a measure space $(\Omega, \mathcal{F}, \mu)$ and that a and b are two real numbers. Then the function $af + bg$ is Lebesgue integrable, and the equality*

$$\int_\Omega (af + bg)\mathrm{d}\mu = a\int_\Omega f\mathrm{d}\mu + b\int_\Omega g\mathrm{d}\mu$$

holds.

Theorem B.25 (monotonicity of the Lebesgue integral)**.** *Suppose that f and g are two Lebesgue integrable functions defined on a measure space $(\Omega, \mathcal{F}, \mu)$. If $f \leq g$ almost everywhere/surely, then the inequality*

$$\int_\Omega f\mathrm{d}\mu \leq \int_\Omega g\mathrm{d}\mu$$

holds.

B.4 The Radon-Nikodym Derivative

Before we can state the Radon-Nikodym theorem and define the Radon-Nikodym derivative, two definitions are needed.

Definition B.26 (σ-finite measure)**.** Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space. Then the measure μ is called a *σ-finite measure* if the set Ω can be covered by at most countably many measurable sets with finite measure, i.e., there exist sets A_n with $\mu(A_n) < \infty$ for all $n \in \mathbb{N}$ such that $\Omega = \bigcup_{n\in\mathbb{N}} A_n$.

Definition B.27 (absolutely continuous)**.** Suppose that $(\Omega, \mathcal{F})$ is a measurable space on which the measures μ and ν are defined. Then the measure μ is called *absolutely continuous with respect to ν* and we write $\mu \ll \nu$ if $\mu(A) = 0$ for every set $A \in \mathcal{F}$ for which $\nu(A) = 0$, i.e.,

$$\mu \ll \nu \quad :\Longleftrightarrow \quad (\forall A \in \mathcal{F} : \quad \nu(A) = 0 \implies \mu(A) = 0).$$

The absolute continuity of measures is reflexive and transitive, but not antisymmetric, and hence it is a preorder and not a partial order.

Using σ-finiteness and absolute continuity, we can state the following theorem.

Theorem B.28 (Radon-Nikodym)**.** *Suppose that $(\Omega, \mathcal{F})$ is a measurable space on which the σ-finite measures μ and ν are defined. If $\mu \ll \nu$, then there exists a $\mathcal{F}$-measurable function $f : \Omega \to [0, \infty)$ such that*

$$\forall A \in \mathcal{F} : \quad \mu(A) = \int_A f\mathrm{d}\nu$$

holds. The function f is unique up to a null set with respect to ν.

Since the equation in the theorem must hold for every measurable set A, we can intuitively understand that the equation must hold for infinitesimal elements $\mathrm{d}\mu$ and $\mathrm{d}\nu$ and we can informally write

$$\mathrm{d}\mu = f\mathrm{d}\nu. \tag{B.4}$$

From this point of view, the function value f is the factor of proportionality between $\mu(A)$ and $\nu(A)$. Hence the condition that μ must be absolutely continuous with respect to ν becomes obvious. Suppose $\nu(A) = 0$. If $\mu(A) \neq 0$, then certainly no value f that satisfies (B.4) can exist. Therefore $\nu(A) = 0$ must imply $\mu(A) = 0$.

An extension to finite valued signed measures ν holds. This theorem makes the following definition possible.

Definition B.29 (Radon-Nikodym derivative)**.** The ν-almost unique function f in Theorem B.28 is called the *Radon-Nikodym derivative of μ with respect to ν* and written

$$f = \frac{\mathrm{d}\mu}{\mathrm{d}\nu}.$$

Important properties of the Radon-Nikodym derivative are collected in the following.

Theorem B.30 (properties of the Radon-Nikodym derivative)**.** *Suppose λ, μ, and ν are σ-finite measures on the measurable space $(\Omega, \mathcal{F})$.*

1. *Linearity: If $\mu \ll \lambda$ and $\nu \ll \lambda$, then*

$$\frac{\mathrm{d}(\mu+\nu)}{\mathrm{d}\lambda} \overset{\lambda}{=} \frac{\mathrm{d}\mu}{\mathrm{d}\lambda} + \frac{\mathrm{d}\nu}{\mathrm{d}\lambda}.$$

2. *Chain rule: If $\nu \ll \mu \ll \lambda$, then*

$$\frac{\mathrm{d}\nu}{\mathrm{d}\lambda} \overset{\lambda}{=} \frac{\mathrm{d}\nu}{\mathrm{d}\lambda}\frac{\mathrm{d}\mu}{\mathrm{d}\lambda}.$$

3. *In particular (choosing $\nu = \lambda$), if $\mu \ll \nu$ and $\nu \ll \mu$, then*

$$\frac{\mathrm{d}\mu}{\mathrm{d}\nu} \overset{\nu}{=} \left(\frac{\mathrm{d}\nu}{\mathrm{d}\mu}\right)^{-1}.$$

4. *If $\mu \ll \lambda$ and f is a Lebesgue integrable function with respect to μ, then*

$$\int_\Omega f\mathrm{d}\mu = \int_\Omega f\frac{\mathrm{d}\mu}{\mathrm{d}\lambda}\mathrm{d}\lambda.$$

5. *If ν is a finite signed or complex measure, then*

$$\frac{\mathrm{d}|\nu|}{\mathrm{d}\mu} = \left|\frac{\mathrm{d}\nu}{\mathrm{d}\mu}\right|.$$

B.5 Lebesgue Convergence Theorems

Fatou's lemma, the Lebesgue monotone convergence theorem, and the Lebesgue dominated convergence theorem are important results in the theory of Lebesgue integration. Given a sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ of functions, they answer the question when the limit $\lim_{n\to\infty}$ and Lebesgue integration commute. The results are also important in probability theory, since they provide sufficient conditions for the convergence of expected values of random variables.

The first result in this section is Fatou's lemma, which shows that an inequality holds when the limit $\liminf_{n\to\infty}$ and Lebesgue integration are interchanged. The sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ does not have to converge pointwise, but the functions are supposed to be non-negative. (For definitions of lim inf and lim sup, see Definition B.58).

Lemma B.31 (Fatou's lemma)**.** *Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space, that $X \in \mathcal{F}$, and that $\langle f_n \rangle_{n\in\mathbb{N}}$ is a sequence of $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable non-negative functions $f_n \colon X \to [0,\infty]$. Suppose further that $f \colon X \to [0,\infty]$ is defined as $f(x) := \liminf_{n\to\infty} f_n(x)$ for μ-almost all $x \in X$. Then the function f is $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable, and the inequality*

$$\int_X f \mathrm{d}\mu = \int_X \liminf_{n\to\infty} f_n \mathrm{d}\mu \leq \liminf_{n\to\infty} \int_X f_n \mathrm{d}\mu$$

holds.

An example for the strict inequality is the measure space $\Omega := [0,1]$ with the Borel σ-algebra and the Lebesgue measure. The functions are defined by

$$f_n(x) := \begin{cases} n, & x \in [0, 1/n), \\ 0, & \text{otherwise.} \end{cases}$$

Then the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ converges pointwise to the zero function, but each function f_n has integral one, leading to the strict inequality in Lemma B.31.

The reverse Fatou's lemma shows that an inequality holds when the limit $\limsup_{n\to\infty}$ and Lebesgue integration are interchanged. Again the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ does not have to converge pointwise, but now the functions are supposed to be dominated by a majorant g.

Lemma B.32 (reverse Fatou's lemma)**.** *Suppose that* $(\Omega, \mathcal{F}, \mu)$ *is a measure space, that* $X \in \mathcal{F}$*, and that* $\langle f_n \rangle_{n\in\mathbb{N}}$ *is a sequence of* $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$*-measurable functions* $f_n \colon X \to [-\infty, \infty]$*. Suppose further that* $g \colon X \to [0, \infty]$ *is a non-negative,* $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$*-measurable, and integrable function on* X *such that*

$$\forall_\mu x \in X \colon \quad \forall n \in \mathbb{N} \colon \quad f_n(x) \leq g(x).$$

Then the inequality

$$\limsup_{n\to\infty} \int_X f_n \mathrm{d}\mu \leq \int_X \limsup_{n\to\infty} f_n \mathrm{d}\mu$$

holds.

Proof. We consider the sequence $g - f_n$. Since $\int_X g \mathrm{d}\mu = \int_X |g| \mathrm{d}\mu < \infty$ by assumption, this sequence is defined μ-almost everywhere. It is also non-negative by the assumption that g dominates the f_n. Therefore we can apply Fatou's lemma, Lemma B.31, to this sequence and use the linearity of Lebesgue integration to find the inequality. □

The next result in this section is the Fatou-Lebesgue theorem. It collects both inequalities of Fatou's lemma and reverse Fatou's lemma. Noting that the middle inequality is trivially true makes remembering the directions of the first inequality (Fatou's lemma) and the third inequality (reverse Fatou's lemma) easier.

Theorem B.33 (Fatou-Lebesgue theorem)**.** *Suppose that* $(\Omega, \mathcal{F}, \mu)$ *is a measure space, that* $X \in \mathcal{F}$*, and that* $\langle f_n \rangle_{n\in\mathbb{N}}$ *is a sequence of* $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$*-measurable functions* $f_n \colon X \to [-\infty, \infty]$ *that is* μ*-almost everywhere dominated by an integrable function* $g \colon X \to [0, \infty]$*, i.e.,*

$$\forall_\mu x \in X \colon \quad \forall n \in \mathbb{N} \colon \quad |f_n(x)| \leq g(x).$$

Then all functions f_n *are integrable as well as the pointwise defined functions* $\liminf_{n\to\infty} f_n$ *and* $\limsup_{n\to\infty} f_n$*, and the inequalities*

$$\int_X \liminf_{n\to\infty} f_n \mathrm{d}\mu \leq \liminf_{n\to\infty} \int_X f_n \mathrm{d}\mu \leq \limsup_{n\to\infty} \int_X f_n \mathrm{d}\mu \leq \int_X \limsup_{n\to\infty} f_n \mathrm{d}\mu$$

hold.

Proof. The absolute values of all functions f_n well as the pointwise defined functions $\liminf_{n\to\infty} f_n$ and $\limsup_{n\to\infty} f_n$ are dominated by the majorant g and are hence integrable, since g is integrable by assumption.

Fatou's lemma, Lemma B.31, can be applied to the functions $f_n + g$, which yields the first inequality. The third inequality is reverse Fatou's lemma, Lemma B.32. □

If we assume that the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ is a pointwise non-decreasing sequence of non-negative functions that converges pointwise to a function f, then the limit $\lim_{n\to\infty}$ and Lebesgue integration indeed commute. This is the Lebesgue monotone convergence theorem.

Theorem B.34 (Lebesgue monotone convergence theorem)**.** *Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space, that $X \in \mathcal{F}$, and that $\langle f_n \rangle_{n\in\mathbb{N}}$ is a μ-almost everywhere pointwise non-decreasing sequence of $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable non-negative functions $f_n \colon X \to [0, \infty]$, i.e.,*

$$\forall_\mu x \in X \colon \quad \forall n \in \mathbb{N} \colon \quad 0 \leq f_n(x) \leq f_{n+1}(x) \leq \infty.$$

Suppose further that the pointwise limits $\lim_{n\to\infty} f_n(x)$ exist for μ-almost all $x \in X$ and that the function f is μ-almost everywhere equal to this μ-almost everywhere pointwise limit of the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$, i.e., $f(x) = \lim_{n\to\infty} f_n(x)$ for μ-almost all $x \in X$. Then the function f is $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable, and the equality

$$\int_X f \mathrm{d}\mu = \int_X \lim_{n\to\infty} f_n \mathrm{d}\mu = \lim_{n\to\infty} \int_X f_n \mathrm{d}\mu$$

holds.

If we assume that the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$ converges pointwise to a function and that is dominated by a majorant g, then the limit $\lim_{n\to\infty}$ and Lebesgue integration indeed commute. This is the Lebesgue dominated convergence theorem.

Theorem B.35 (Lebesgue dominated convergence theorem)**.** *Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space, that $X \in \mathcal{F}$, and that $\langle f_n \rangle_{n\in\mathbb{N}}$ is a sequence of $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable functions $f_n \colon X \to [-\infty, \infty]$ that is μ-almost everywhere dominated by an integrable function $g \colon X \to [0, \infty]$, i.e.,*

$$\forall_\mu x \in X \colon \quad \forall n \in \mathbb{N} \colon \quad |f_n(x)| \leq g(x).$$

Suppose further that the pointwise limits $\lim_{n\to\infty} f_n(x)$ exist for μ-almost all $x \in X$ and that the function f is μ-almost everywhere equal to this μ-almost everywhere pointwise limit of the sequence $\langle f_n \rangle_{n\in\mathbb{N}}$, i.e., $f(x) = \lim_{n\to\infty} f_n(x)$ for μ-almost all $x \in X$. Then the function f is integrable, and the equality

$$\lim_{n\to\infty} \int_X |f_n - f| \mathrm{d}\mu = 0$$

holds, which also implies

$$\int_X f \mathrm{d}\mu = \int_X \lim_{n\to\infty} f_n \mathrm{d}\mu = \lim_{n\to\infty} \int_X f_n \mathrm{d}\mu.$$

B.6 Probability Spaces and Random Variables

A probability space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$ consisting of a sample space Ω, an event space $\mathcal{F}$, and a probability function $\mathbb{P}$. The sample space Ω is the set of all possible outcomes, where an outcome is the result of a single execution of the model. The event space $\mathcal{F}$ is the set of all events, where an event is a set of zero or more outcomes, i.e., a subset of the sample space Ω. The probability function $\mathbb{P} \colon \mathcal{F} \to [0, 1]$ returns the probability of each event; the probability of the whole sample space Ω must be equal to one.

In the following, formal definitions of a probability space and a random variable are given.

Definition B.36 (probability measure)**.** A *probability measure* $\mathbb{P}$ is a measure that assigns value one to the entire (sample) space Ω, i.e., $\mathbb{P}(\Omega) = 1$.

Definition B.37 (probability space)**.** A *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ is a triple consisting of a sample space Ω, an event space $\mathcal{F}$, and a probability function $\mathbb{P}$ that satisfy the following properties:

1. The sample space Ω is an arbitrary non-empty set.
2. The event space $\mathcal{F}$ is a set of subsets (events) of the sample space Ω and a σ-algebra.
3. The probability function $\mathbb{P} \colon \mathcal{F} \to [0, 1]$ is a probability measure.

If $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space, then it is also a measure space and $(\Omega, \mathcal{F})$ is a measurable space by the definition of a probability space.

Random variables are functions on probability spaces which are compatible with measuring probabilities.

Definition B.38 (random variable)**.** Suppose $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space and $(\Psi, \mathcal{G})$ is a measurable space. Then a $(\Psi, \mathcal{G})$*-valued random variable* X is a measurable function $X \colon \Omega \to \Psi$.

The only difference between a measurable function (with domain $(\Omega, \mathcal{F})$) and a random variable (with domain $(\Omega, \mathcal{F}, \mathbb{P})$) is that a random variable comes with a probability measure.

While the domain of a random variable X is the sample space, its codomain Ψ is called the observation space. The definition of a random variable X means that the probability measure $\mathbb{P}$ yields the probabilities of all preimages $X^{-1}(G)$ (as long as G is in the σ-algebra $\mathcal{G}$ of the measurable space that is the codomain or observation space of the random variable X). In other words, a random

variable X maps any outcome $\omega \in \Omega$ to an observed quantity $\psi \in \Psi$ such that the outcomes that lead to an observation $G \in \mathcal{G}$ in the observation space have a probability given by the probability measure $\mathbb{P}$.

Real valued random variables $X : \Omega \to \mathbb{R}$ are the important special case where the observation space are the real numbers. We can use the generator (B.1h) of the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ (see Section B.2). Since it suffices to check measurability on any generator of the Borel σ-algebra and the preimages in Definition B.9 are $X^{-1}((-\infty, a]) = \{\omega \in \Omega : X(\omega) \leq a\}$, a function $X : \Omega \to \mathbb{R}$ is a (real valued) random variable if $\{\omega \in \Omega : X(\omega) \leq a\} \in \mathcal{F}$ holds for all $a \in \mathbb{R}$.

Integrable random variables are Lebesgue integrable functions (see Definition B.18).

Definition B.39 (integrable random variable)**.** A random variable X is called *integrable* if it is a Lebesgue integrable function, i.e., if

$$\mathbb{E}[|X|] = \int_\Omega |X| \mathrm{d}\mathbb{P} < \infty.$$

The definition of an integrable random variable implies that $\mathbb{E}[X]$ exists and has a finite value as well.

Intuitively, in the one-dimensional case, a real valued random variable $X : \Omega \to \mathbb{R}$ has the probability density function f_X if

$$\mathbb{P}[a < X \leq b] = \int_a^b f_X(x) \mathrm{d}x$$

holds for all intervals $[a, b] \subset \mathbb{R}$, where f_X is a non-negative Lebesgue integrable function. The cumulative distribution function of X is the function

$$F_X(x) := \mathbb{P}[X \leq x] = \int_{-\infty}^x f_X(t) \mathrm{d}t.$$

Using the cumulative distribution function, we can also write

$$\mathbb{P}[a < X \leq b] = F_X(b) - F_X(a) = \int_a^b f_X(x) \mathrm{d}x.$$

If the probability density function f_X is continuous at x, then both functions are related by

$$f_X(x) = F_X'(x).$$

If the cumulative distribution function F_X is left-continuous at x, the value of

$$\mathbb{P}[X = b] = F_X(b) - \lim_{x \to b-} F_X(x)$$

vanishes, which means that there is no discrete component at b (continuous random variable). If it is not left-continuous, this value is called the discrete component of the probability distribution at b (discrete random variable).

Formally, the probability density function is defined within the measure theoretic framework as follows.

Definition B.40 (probability density function (PDF), cumulative distribution function (CDF))**.** Suppose that $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space, that $(\Psi, \mathcal{G}, \nu)$ is a measure space, and that $X \colon \Omega \to \Psi$ is a $(\Psi, \mathcal{G}, \nu)$-valued random variable. The so-called reference measure ν is the counting measure in the case that Ψ is finite (discrete random variable) and the Lebesgue measure in the case that $\Psi = \mathbb{R}^d$, $d \in \mathbb{N}$ ((multidimensional) continuous random variable). Then any measurable function $f_X \colon \Psi \to \mathbb{R}_0^+$ that satisfies

$$\forall G \in \mathcal{G} \colon \quad \mathbb{P}[X \in G] := \mathbb{P}(G) := \int_{X^{-1}(G)} \mathrm{d}\mathbb{P} = \int_G f_X \mathrm{d}\nu$$

is called a *probability density function* of the random variable X. Furthermore, its *cumulative distribution function* is the function

$$F_X \colon \Psi \to \mathbb{R}_0^+, \quad F_X(x) := \mathbb{P}[X \leq x] := \int_\Psi [\![t_1 \leq x_1]\!] \cdots [\![t_n \leq x_n]\!] f_X(t) \mathrm{d}\nu(t),$$

where $X \leq x$ is understood elementwise whenever the random variable X is vector valued and the ordering $\leq$ in the integrand is a total ordering on Ψ.

In other words, the probability density function f_X is the Radon-Nikodym derivative

$$f_X = \frac{\mathrm{d}(X\#\mathbb{P})}{\mathrm{d}\nu} = \frac{\mathrm{d}(\mathbb{P} \circ X^{-1})}{\mathrm{d}\nu}$$

and as such it is almost unique. Here $X\#\mathbb{P} = \mathbb{P} \circ X^{-1}$ is the push-forward measure of $\mathbb{P}$.

Next, we define commonly used operators on random variables. If the random variable is continuous, then the reference measure ν is the Lebesgue measure and $\mathrm{d}\nu(x)$ is commonly replaced by $\mathrm{d}x$ to denote the Lebesgue measure.

Definition B.41 (moment)**.** Suppose X is a random variable with probability density function f_X as in Definition B.40. Then

$$M_k(X, c) := \int_{X^{-1}(\Psi)} (x - c)^k \mathrm{d}\mathbb{P}(x) = \int_\Psi (x - c)^k f_X(x) \mathrm{d}\nu(x)$$

is called the *k-th moment* of the random variable X about the center c.

Definition B.42 (mean / expected value / expectance)**.** The *expected value* or *expectance* of a random variable X is its first moment about the center zero, i.e.,

$$\mathbb{E}[X] := M_1(X, 0) := \int_{X^{-1}(\Psi)} x \mathrm{d}\mathbb{P}(x) = \int_{\Psi} x f_X(x) \mathrm{d}\nu(x).$$

Definition B.43 (variance, standard deviation)**.** The *variance* of a random variable X is its second moment about the expected value, i.e.,

$$\mathbb{V}[X] := M_2(X, \mathbb{E}[X]) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \int_{\Psi} (x - \mathbb{E}[X])^2 f_X(x) \mathrm{d}\nu(x).$$

The *standard deviation* is the square root of the variance, i.e.,

$$\sigma[X] := \sqrt{\mathbb{V}[X]}.$$

The formulae above can also expediently be written using Riemann-Stieltjes integrals in terms of the probability density function and the cumulative distribution function. The equality

$$\int g(x) \mathrm{d}F_X(x) = \int g(x) f_X(x) \mathrm{d}x$$

holds for Riemann-Stieltjes integrals, if the involved functions are smooth enough such that all integrals exist. For example, we hence have

$$\begin{aligned} M_k(X, c) &= \int (x - c)^k \mathrm{d}F_X(x), \\ \mathbb{E}[X] &= \int x \mathrm{d}F_X(x), \\ \mathbb{V}[X] &= \int (x - \mathbb{E}[X])^2 \mathrm{d}F_X(x). \end{aligned}$$

B.7 Inequalities

In this section, important inequalities connected to measure and probability theory are collected.

B.7.1 Basic Inequalities

Jensen's inequality is an important inequality, whose measure-theoretic form is stated in the following theorem.

Definition B.44 (convex set)**.** A set C is called *convex* if

$$\forall (x,y) \in C^2 \colon \quad \forall y \in C \colon \quad \forall \alpha \in [0,1] \colon \quad \alpha x + (1-\alpha)y \in C.$$

Definition B.45 ((strictly) convex and concave functions)**.** Suppose C is a convex set. A function $f \colon C \to \mathbb{R}$ is called *convex* if

$$\forall (x,y) \in C^2 \colon \quad \forall \alpha \in [0,1] \colon \quad f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

It is called *strictly convex* if it satisfies this property with $\leq$ replaced by $<$. It is called *(strictly) concave* if $-f$ is (strictly) convex.

Definition B.46 (subderivative)**.** A *subderivative* of a convex function $f \colon I \to \mathbb{R}$ at a point $x_0 \in I$ in the open interval I is a real number $c \in \mathbb{R}$ such that

$$\forall x \in I \colon \quad f(x) - f(x_0) \geq c(x - x_0).$$

Lemma B.47 (subderivatives)**.** *Suppose $f \colon I \to \mathbb{R}$ is a convex function on an open interval I. Then the set of subderivatives at a point $x_0 \in I$ is the non-empty closed interval*

$$\left[\lim_{x \to x_0 -} \frac{f(x) - f(x_0)}{x - x_0}, \lim_{x \to x_0 +} \frac{f(x) - f(x_0)}{x - x_0} \right].$$

Theorem B.48 (Jensen's inequality)**.** *Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space, that $I \subset \mathbb{R}$ is an interval, that the function $g \colon \Omega \to I$ is Lebesgue integrable with respect to μ, and that the function $\phi \colon I \to \mathbb{R}$ is convex on the interval I. Then the inequality*

$$\phi\left(\frac{1}{\mu(\Omega)} \int_\Omega g \mathrm{d}\mu \right) \leq \frac{1}{\mu(\Omega)} \int_\Omega \phi \circ g \mathrm{d}\mu$$

holds. If the function ϕ is concave, the inequality is reversed.

If the space is a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, the inequality is commonly written as

$$\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)].$$

Proof. We start by defining the point

$$x_0 := \frac{1}{\mu(\Omega)} \int_\Omega g \mathrm{d}\mu \in I.$$

Next, by the convexity of ϕ and by Lemma B.47, there exists a $c \in \mathbb{R}$ for the point $x_0 \in I$ such that

$$\forall x \in I \colon \quad \phi(x) - \phi(x_0) \geq c(x - x_0).$$

Since this inequality holds for all $x \in I$, it also holds for all $x = g(\omega) \in I$, i.e.,

$$\forall \omega \in \Omega\colon \quad \phi(g(\omega)) \geq \phi(x_0) + c(g(\omega) - x_0).$$

Using the monotony of the integral, integration of both sides yields

$$\int_\Omega \phi \circ g \mathrm{d}\mu \geq \mu(\Omega)\phi(x_0) + c\left(\int_\Omega g \mathrm{d}\mu - \mu(\Omega)x_0\right) = \mu(\Omega)\phi(x_0).$$

If the function ϕ is concave, the analogous argument shows the reversed inequality, which concludes the proof. □

B.7.2 Concentration Inequalities

Concentration inequalities provide probability bounds on how much a random variable deviates from its expected value.

The Markov and Chebyshev Inequalities

Theorem B.49 (Markov inequality)**.** *Suppose X is a non-negative random variable and $a \in \mathbb{R}^+$. Then the inequality*

$$\mathbb{P}[X \geq a] \leq \frac{\mathbb{E}[X]}{a} \tag{B.5}$$

holds.

Proof. Since X is non-negative random variable, its expected value can be written as

$$\mathbb{E}[X] = \int_0^\infty x f_X(x) \mathrm{d}x.$$

By splitting the interval at $a \in \mathbb{R}^+$, we can estimate

$$\begin{aligned}\mathbb{E}[X] = \int_0^a x f_X(x)\mathrm{d}x + \int_a^\infty x f_X(x)\mathrm{d}x \\ \geq \int_a^\infty x f_X(x)\mathrm{d}x \geq a\int_a^\infty f_X(x)\mathrm{d}x = a\mathbb{P}[X \geq a],\end{aligned}$$

which concludes the proof. □

Corollary B.50. *Inequality* (B.5) *is equivalent to*

$$\mathbb{P}\left[X \leq \frac{\mathbb{E}[X]}{\delta}\right] \geq 1 - \delta.$$

Proof. Inequality (B.5) is equivalent to $\mathbb{P}[X \leq a] = 1 - \mathbb{E}[X \geq a] \geq 1 - \mathbb{E}[X]/a = 1 - \delta$, where we have defined $\delta := \mathbb{E}[X]/a$. □

Theorem B.51 (Chebyshev inequality)**.** *Suppose X is an integrable random variable with expected value $\mu := \mathbb{E}[x]$ and finite, non-zero variance $\sigma^2 := \mathbb{V}[X] \in (0, \infty)$. Then the inequality*

$$\forall k \in \mathbb{R}^+ : \quad \mathbb{P}[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

holds.

Proof. The inequality can be shown by applying Markov's inequality to the random variable $(X-\mu)^2$ and the constant $a := (k\sigma)^2$ in Markov's inequality. □

Hoeffding's Inequality

In [64], variants of Hoeffding's inequality were shown. We start by proving Hoeffding's lemma before stating Hoeffding's inequalities and discussing how they can be applied.

Lemma B.52 (Hoeffding's lemma)**.** *Suppose that X is a real valued random variable such that*

$$\exists (a, b) \in \mathbb{R}^2 : \quad \mathbb{P}[a \leq X \leq b] = 1.$$

Then the inequality

$$\forall \lambda \in \mathbb{R} : \quad \mathbb{E}\big[\mathrm{e}^{\lambda(X - \mathbb{E}[X])}\big] \leq \mathrm{e}^{\lambda^2 (b-a)^2/8}$$

holds.

Proof. If $a = b$, then the inequality simply becomes $\mathbb{E}[1] \leq \mathbb{E}[1]$.

Otherwise, if $a < b$, since the function $x \mapsto \mathrm{e}^{\lambda(x - \mathbb{E}[X])}$ is convex, the inequality

$$\forall x \in [a, b] : \quad \mathrm{e}^{\lambda(x - \mathbb{E}[X])} \leq \frac{b - x}{b - a} \mathrm{e}^{\lambda(a - \mathbb{E}[X])} + \frac{x - a}{b - a} \mathrm{e}^{\lambda(b - \mathbb{E}[X])}$$

holds. Applying the expected value to both sides yields

$$\begin{aligned} \mathbb{E}\big[\mathrm{e}^{\lambda(X - \mathbb{E}[X])}\big] &\leq \frac{b - \mathbb{E}[X]}{b - a} \mathrm{e}^{\lambda(a - \mathbb{E}[X])} + \frac{\mathbb{E}[X] - a}{b - a} \mathrm{e}^{\lambda(b - \mathbb{E}[X])} \\ &= (1 - \alpha) \mathrm{e}^{\lambda(a - \mathbb{E}[X])} + \alpha \mathrm{e}^{\lambda(b - \mathbb{E}[X])}, \end{aligned}$$

where

$$\alpha := \frac{\mathbb{E}[X] - a}{b - a}.$$

Using $y := \lambda(b-a)$, we have

$$\begin{aligned}\mathbb{E}[\mathrm{e}^{\lambda(X-\mathbb{E}[X])}] &\leq (1-\alpha+\alpha\mathrm{e}^{\lambda(b-a)})\mathrm{e}^{\lambda(a-\mathbb{E}[X])} \\ &= (1-\alpha+\alpha\mathrm{e}^{\lambda(b-a)})\mathrm{e}^{-\alpha\lambda(b-a)} \\ &= (1-\alpha+\alpha\mathrm{e}^{y})\mathrm{e}^{-\alpha y}.\end{aligned}$$

Defining the function

$$f\colon \quad \mathbb{R}\to\mathbb{R}, \quad y\mapsto \ln(1-\alpha+\alpha\mathrm{e}^{y})-\alpha y,$$

the inequality becomes

$$\mathbb{E}[\mathrm{e}^{\lambda(X-\mathbb{E}[X])}] \leq \mathrm{e}^{f(y)}.$$

The function f is well-defined, since

$$1-\alpha+\alpha\mathrm{e}^{y} = \alpha\left(\frac{1}{\alpha}-1+\mathrm{e}^{y}\right) = \alpha\left(\frac{b-\mathbb{E}[X]}{\mathbb{E}[X]-a}+\mathrm{e}^{y}\right) > 0$$

because of $a \leq \mathbb{E}[X] \leq b$ by the assumption $\mathbb{P}[a \leq X \leq b] = 1$.

It remains to find an upper bound for f. Since f is sufficiently smooth, Taylor's theorem yields

$$\forall y\in\mathbb{R}\colon \quad \exists z\in[0,y]\colon \quad f(y) = f(0)+f'(0)y+\frac{f''(z)}{2}y^2.$$

Because of

$$\begin{aligned}f'(y) &= \frac{\alpha\mathrm{e}^{y}}{1-\alpha+\alpha\mathrm{e}^{y}}-\alpha, \\ f''(y) &= \frac{\alpha\mathrm{e}^{y}}{1-\alpha+\alpha\mathrm{e}^{y}}\left(1-\frac{\alpha\mathrm{e}^{y}}{1-\alpha+\alpha\mathrm{e}^{y}}\right) = u(1-u), \quad u := \frac{\alpha\mathrm{e}^{y}}{1-\alpha+\alpha\mathrm{e}^{y}},\end{aligned}$$

we have $f(0) = 0$, $f'(0) = 0$, and $f''(z) \leq 1/4$ due to $u(1-u) \leq 1/4$ for all $u\in\mathbb{R}$. Therefore the estimate

$$f(y) \leq \frac{y^2}{8} = \frac{\lambda^2(b-a)^2}{8}$$

holds, which completes the proof. □

Theorem B.53 (Hoeffding's inequality)**.** *Suppose that $\{X_i\}_{i=1}^n$ are independent real valued random variables, each being bounded such that*

$$\forall i\in[1{:}n]\colon \quad \exists(a_i,b_i)\in\mathbb{R}^2\colon \quad \mathbb{P}[a_i\leq X_i\leq b_i] = 1.$$

Then the inequalities

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right),$$
$$\mathbb{P}[|\overline{X} - \mathbb{E}[\overline{X}]| \geq t] \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right),$$
$$\overline{X} := \frac{1}{n}\sum_{i=1}^n X_i$$

hold for all $t \in \mathbb{R}^+$.

These inequalities also hold when the random variables $\{X_i\}_{i=1}^n$ are obtained by sampling without replacement [64]. Better bounds for this case can be found in [65].

Proof. We first define

$$S := \sum_{i=1}^n X_i.$$

Using an arbitrary parameter $\lambda \in \mathbb{R}^+$ and Markov's inequality, Theorem B.49, we find

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] = \mathbb{P}[S - \mathbb{E}[S] \geq nt] = \mathbb{P}[\mathrm{e}^{\lambda(S-\mathbb{E}[S])} \geq \mathrm{e}^{\lambda nt}] \leq \frac{\mathbb{E}[\mathrm{e}^{\lambda(S-\mathbb{E}[S])}]}{\mathrm{e}^{\lambda nt}}.$$

Since the random variables X_i are independent, the inequality

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \mathrm{e}^{-\lambda nt} \prod_{i=1}^n \mathbb{E}[\mathrm{e}^{\lambda(X_i - \mathbb{E}[X_i])}] \tag{B.6}$$

holds. Using Hoeffding's lemma, Lemma B.52, we can estimate the right-hand side to obtain

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \mathrm{e}^{-\lambda nt} \prod_{i=1}^n \mathrm{e}^{\lambda^2(b_i-a_i)^2/8} = \exp\left(-\lambda nt + \frac{\lambda^2}{8}\sum_{i=1}^n (b_i - a_i)^2\right).$$

We can now use the parameter $\lambda \in \mathbb{R}^+$ to find the best possible upper bound. Because the right-hand side is a quadratic function of λ, it is straightforward to calculate that it achieves its global minimum at

$$\lambda := \frac{4nt}{\sum_{i=1}^n (b_i - a_i)^2} > 0,$$

which is positive since $t > 0$. This value for λ yields

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right),$$

which proves the first inequality.

The see the second one, we calculate

$$\begin{aligned}\mathbb{P}[|\overline{X} - \mathbb{E}[\overline{X}]| \geq t] &= \mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] + \mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \leq -t] \\ &= \mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] + \mathbb{P}[-\overline{X} + \mathbb{E}[\overline{X}] \geq t].\end{aligned}$$

The first term is bounded by the first inequality. The second term is also bounded by the first inequality, but now applied to the random variables $Y_i := -X_i$ and noting that the assumption $\mathbb{P}[-b_i \leq Y_i \leq -a_i] = 1$ also results in the sum $\sum_{i=1}^n (b_i - a_i)^2$, which concludes the proof. □

Corollary B.54 (Hoeffding's inequality for sums)**.** *Suppose the assumptions of Theorem B.53 hold. Then the inequalities*

$$\begin{aligned}\mathbb{P}[S - \mathbb{E}[S] \geq t] &\leq \exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right), \\ \mathbb{P}[|S - \mathbb{E}[S]| \geq t] &\leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right), \\ S &:= \sum_{i=1}^n X_i,\end{aligned}$$

hold for all $t \in \mathbb{R}^+$.

Proof. Theorem B.53 can be applied to

$$\mathbb{P}[S - \mathbb{E}[S] \geq t] = \mathbb{P}[n\overline{X} - n\mathbb{E}[\overline{X}] \geq t] = \mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t/n]$$

since $t/n \in \mathbb{R}^+$. □

Corollary B.55 (Hoeffding's inequality for confidence intervals)**.** *Suppose the assumptions of Theorem B.53 hold. Then the inequalities*

$$\mathbb{P}\left[\mathbb{E}[\overline{X}] > \overline{X} - \sqrt{\frac{-\ln(\delta)\sum_{i=1}^n (b_i - a_i)^2}{2n^2}}\right] \geq 1 - \delta, \tag{B.7a}$$

$$\mathbb{P}\left[|\mathbb{E}[\overline{X}] - \overline{X}| < \sqrt{\frac{-\ln(\delta)\sum_{i=1}^{n}(b_i - a_i)^2}{2n^2}}\right] \geq 1 - 2\delta \tag{B.7b}$$

hold for all $\delta \in (0, 1)$.

The first inequality is often formulated as follows: with probability at least $1 - \delta$, the inequality

$$\mathbb{E}[\overline{X}] > \overline{X} - \sqrt{\frac{-\ln(\delta)\sum_{i=1}^{n}(b_i - a_i)^2}{2n^2}}$$

holds. The second inequality is equivalent to saying that the inequality

$$|\mathbb{E}[\overline{X}] - \overline{X}| < \sqrt{\frac{-\ln(\delta)\sum_{i=1}^{n}(b_i - a_i)^2}{2n^2}}$$

holds with probability at least $1 - 2\delta$.

Proof. We start from the function

$$\delta\colon \quad \mathbb{R}^+ \to (0, 1), \quad \delta(t) := \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{B.8}$$

and its inverse

$$t\colon \quad (0, 1) \to \mathbb{R}^+, \quad t(\delta) := \sqrt{\frac{-\ln(\delta)\sum_{i=1}^{n}(b_i - a_i)^2}{2n^2}} \tag{B.9}$$

and note that δ is a (monotone decreasing) bijection between $t \in \mathbb{R}^+$ and $\delta \in (0, 1)$.

Using these definitions, the first inequality in Theorem B.53, where $t \in \mathbb{R}^+$ is assumed, becomes

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \delta(t),$$

which is equivalent to

$$\mathbb{P}[\mathbb{E}[\overline{X}] > \overline{X} - t] \geq 1 - \delta(t)$$

after negation. This is the first inequality.

The second inequality in Theorem B.53 becomes

$$\mathbb{P}[|\overline{X} - \mathbb{E}[\overline{X}]| \geq t] \leq 2\delta(t),$$

whose negation leads to

$$\mathbb{P}[|\mathbb{E}[\overline{X}] - \overline{X}| < t] \geq 1 - 2\delta(t).$$

This is the second inequality, which completes the proof. □

This corollary shows how Hoeffding's inequality is used to calculate one-sided and two-sided confidence intervals. We are interested in the (true) expected value $\mathbb{E}[\overline{X}]$, but we can only empirically calculate the sample mean $\overline{X}$. In accordance with the assumptions of Theorem B.53, we assume that the random variables are independent or that sampling is performed without replacement (by the comment below the theorem). Then (B.7a) yields the one-sided confidence interval

$$(\overline{X} - t(\delta), \infty)$$

at confidence level $1 - \delta$ (usually close to one) for the true value $\mathbb{E}[\overline{X}]$. Similarly, (B.7b) yields the (symmetric) two-sided confidence interval

$$(\overline{X} - t(\delta), \overline{X} + t(\delta))$$

at the confidence level $1 - 2\delta$ for the true value $\mathbb{E}[\overline{X}]$.

So far we have considered the number n of random variables (and their bounds a_i and b_i) to be fixed. If we view them as variable, another way to interpret the inequalities in Corollary B.55 is to ask the question how many samples should be obtained in order to acquire a confidence interval of given size t_0 (smaller t_0 is better) and of given confidence level $1 - \delta_0$ (larger $1 - \delta_0$ is better, i.e., smaller δ_0 is better).

To shorten the notation, we define

$$m := \frac{2n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}$$

and note that in the important special case that all a_i are equal to a constant $a \in \mathbb{R}$ and all b_i are equal to a constant $b \in \mathbb{R}$, we have

$$m = \frac{2n^2}{n(b-a)^2} = \frac{2n}{(b-a)^2},$$

meaning that m grows just as the number n of sampled random variables increases.

To acquire a given one-sided confidence interval of given size $t_0 \in \mathbb{R}^+$ and of given confidence level $1 - \delta_0$ with $\delta_0 \in (0, 1)$, we first write $\delta(t, m)$ and $t(\delta, m)$ for the two functions defined in (B.8) and (B.9) to underline their dependence

on m. In the first case, we are given the confidence level $1 - \delta_0$ and would like to find values of m that also satisfy a given size t_0, i.e., we seek m such that

$$t(\delta_0, m) \leq t_0,$$

which is equivalent to

$$m \geq \frac{-\ln \delta_0}{t_0^2} > 0.$$

In the second case, we are given a confidence interval size t_0 and would like to find values of m that also satisfy a given confidence level $1 - \delta_0$, i.e., we seek m such that

$$\delta(t_0, m) \leq \delta_0,$$

which is equivalent to

$$m \geq \frac{-\ln \delta_0}{t_0^2} > 0.$$

In both cases we arrive at the same condition for m, and thus define

$$m\colon \quad (0,1) \times \mathbb{R}^+ \to \mathbb{R}^+, \quad m(\delta, t) := \frac{-\ln \delta}{t^2}.$$

This condition for m can interpreted in terms of the number n of samples needed. If the size t_0 of the confidence interval is to be reduced by a factor λ (while the confidence level $1 - \delta_0$ is kept constant), m and hence the number n of samples scales quadratically since

$$\frac{m(\delta_0, \lambda t_0)}{m(\delta_0, t_0)} = \frac{1}{\lambda^2}.$$

The quadratic scaling is consistent with the law of large numbers and the central limit theorem (see Section B.11).

Similarly, if δ_0 is to be reduced by a factor λ (while the size t_0 is kept constant), m and hence the number n of samples scales as

$$\frac{m(\lambda \delta_0, t_0)}{m(\delta_0, t_0)} = 1 + \frac{\ln \lambda}{\ln \delta_0}.$$

Bernstein's Inequality

Bernstein-type inequalities date back to the 1920s and 1930s and are the oldest inequalities that give bounds on the probability how much a sum of random variables deviates from its mean. While Hoeffding's inequality, which serves the same purpose, only supposes that the random variables are bounded, Bernstein inequalities also use the variance of the distribution to get tighter bounds. A typical inequality of the Bernstein type is the following one.

Theorem B.56 (Bernstein's inequality)**.** *Suppose* $\{X_i\}_{i=1}^n$ *are independent random variables such that*

$$\forall i \in [1{:}n]\colon \quad \mathbb{E}[X_i] = 0$$

and

$$\exists M \in \mathbb{R}^+\colon \quad \forall i \in [1{:}n]\colon \quad \mathbb{P}[|X_i| \leq M] = 1.$$

Then the inequality

$$\begin{aligned}
\mathbb{P}[\overline{X} \geq t] &\leq \exp\left(-\frac{nt^2}{2(Mt/3 + n\sigma^2)}\right), \\
\overline{X} &:= \frac{1}{n}\sum_{i=1}^n X_i, \\
\sigma^2 &:= \mathbb{V}[\overline{X}] = \frac{1}{n^2}\sum_{i=1}^n \mathbb{V}[X_i] = \frac{1}{n^2}\sum_{i=1}^n \mathbb{E}[X_i^2],
\end{aligned}$$

holds for all $t \in \mathbb{R}^+$.

Proof. Since the assumptions of Hoeffding's inequality, Theorem B.53, are satisfied, we will be able to use (B.6), which contains the terms $\mathbb{E}[\mathrm{e}^{\lambda X_i}]$, to show the proposed inequality.

Before we do so, we estimate the terms $\mathbb{E}[\mathrm{e}^{\lambda X_i}]$. We start by using the Taylor expansion of the exponential function and the assumption that $\mathbb{E}[X_i] = 0$ for all $i \in [1{:}n]$ to find

$$\forall \lambda \in \mathbb{R}\colon \quad \mathbb{E}[\mathrm{e}^{\lambda X_i}] = 1 + \sum_{k=2}^{\infty} \frac{\lambda^k \mathbb{E}[X_i^k]}{k!}$$

for all random variables indexed by $i \in [1{:}n]$. We define the infinite sum

$$f\colon \quad \mathbb{R} \to \mathbb{R}, \quad \lambda \mapsto \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\mathbb{E}[X_i^k]}{\sigma_i^2 k!}, \quad \sigma_i^2 := \mathbb{V}[X_i] = \mathbb{E}[X_i^2]$$

and note that it converges for all $\lambda \in \mathbb{R}$, which can easily be seen by the ratio test.

Using the infinite sum f, we can write

$$\forall \lambda \in \mathbb{R}\colon \quad \mathbb{E}[\mathrm{e}^{\lambda X_i}] = 1 + \lambda^2 \sigma_i^2 f(\lambda).$$

We now use the inequality $1 + x \leq \mathrm{e}^x$ for all $x \in \mathbb{R}$, which can be proved by using the starting point $x = 0$ (for both intervals $[0, \infty)$ and $(-\infty, 0]$) and showing

the differentiated inequality. By integration, the inequality follows from the differentiated one. This inequality yields

$$\forall \lambda \in \mathbb{R}\colon \quad \mathbb{E}[\mathrm{e}^{\lambda X_i}] \leq \mathrm{e}^{\lambda^2 \sigma_i^2 f(\lambda)}. \tag{B.10}$$

Next, we consider the terms $\mathbb{E}[X_i^k]$ in the infinite sum f. The Cauchy-Schwarz inequality yields

$$\begin{aligned}\mathbb{E}[X_i^k] = \int x_i x_i^{k-1} \mathrm{d}\mathbb{P}(x_i) &\leq \left(\int |x_i|^2 \mathrm{d}\mathbb{P}(x_i)\right)^{1/2} \left(\int |x_i^{k-1}|^2 \mathrm{d}\mathbb{P}(x_i)\right)^{1/2} \\ &= \sigma_i \left(\int |x_i|^{2k-2} \mathrm{d}\mathbb{P}(x_i)\right)^{1/2}.\end{aligned}$$

We continue to apply the Cauchy-Schwarz inequality to the last factor recursively. Each application of the Cauchy-Schwarz inequality has the form

$$\begin{aligned}\int x_i^\alpha \mathrm{d}\mathbb{P}(x_i) = \int x_i x_i^{\alpha-1} \mathrm{d}\mathbb{P}(x_i) &\leq \left(\int |x_i|^2 \mathrm{d}\mathbb{P}(x_i)\right)^{1/2} \left(\int |x_i^{\alpha-1}|^2 \mathrm{d}\mathbb{P}(x_i)\right)^{1/2} \\ &= \sigma_i \left(\int |x_i|^{2(\alpha-1)} \mathrm{d}\mathbb{P}(x_i)\right)^{1/2}.\end{aligned}$$

Therefore the exponents of $|x_i|$ satisfy the recursion

$$a_1 := 2k-2, \qquad a_{m+1} = 2(a_m - 1).$$

It is straightforward to show by induction that

$$a_m = 2^m k - 2^{m+1} + 2.$$

In summary, continuing to apply the Cauchy-Schwarz inequality recursively m times in total, each time splitting off a term $|x_i|$ in the last factor, results in

$$\begin{aligned}\forall m \in \mathbb{N}\colon \quad \mathbb{E}[X_i^k] &\leq \sigma_i^{1+1/2+\cdots+(1/2)^{m-1}} \left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}\mathbb{P}(x_i)\right)^{1/2^m} \\ &= \sigma_i^{2(1-1/2^m)} \left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}\mathbb{P}(x_i)\right)^{1/2^m}.\end{aligned}$$

By assumption, the absolute values of the random variables X_i are bounded by the constant M with probability one. Therefore the last factor can be bounded by

$$\left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}\mathbb{P}(x_i)\right)^{1/2^m} \leq (M^{2^m k - 2^{m+1}+2})^{1/2^m},$$

which leads to

$$\mathbb{E}[X_i^k] \leq \sigma_i^{2(1-1/2^m)} M^{k-2+1/2^{m-1}}.$$

Taking the limit $m \to \infty$ yields

$$\mathbb{E}[X_i^k] \leq \lim_{m\to\infty} \sigma_i^{2(1-1/2^m)} (M^{k-2+1/2^{m-1}}) = \sigma_i^2 M^{k-2}.$$

Therefore, the infinite sum can be bounded above by

$$\begin{aligned}\forall \lambda \in \mathbb{R}_0^+ \colon \quad f(\lambda) = \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\mathbb{E}[X_i^k]}{\sigma_i^2 k!} &\leq \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\sigma_i^2 M^{k-2}}{\sigma_i^2 k!} = \frac{1}{\lambda^2 M^2} \sum_{k=2}^{\infty} \frac{\lambda^k M^k}{k!} \\ &= \frac{1}{\lambda^2 M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\end{aligned}$$

if the parameter λ is non-negative. Applying this estimate to (B.10) yields

$$\forall \lambda \in \mathbb{R}_0^+ \colon \quad \mathbb{E}[\mathrm{e}^{\lambda X_i}] \leq \exp\left(\lambda^2 \sigma_i^2 \frac{1}{\lambda^2 M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right).$$

Next, we use inequality (B.6) as alluded to in the beginning and the assumptions on the random variables X_i (that imply $\sum_{i=1}^n \sigma_i^2 = n^2\sigma^2$) to find

$$\begin{aligned}\forall \lambda \in \mathbb{R}_0^+ \colon \quad \forall t \in \mathbb{R}^+ \colon \quad \mathbb{P}[\overline{X} \geq t] &\leq \mathrm{e}^{-\lambda n t} \prod_{i=1}^n \exp\left(\lambda^2 \sigma_i^2 \frac{1}{\lambda^2 M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right) \\ &= \exp\left(-\lambda n t + \frac{n^2\sigma^2}{M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right).\end{aligned}$$

As in the proof of Hoeffding's inequality, we now minimize the right-hand side with respect to the unknown parameter $\lambda \in \mathbb{R}_0^+$ to find the best (i.e., smallest) upper bound. The first derivative of the right-hand side r is

$$r'(\lambda) = \left(-nt + \frac{n^2\sigma^2}{M}\mathrm{e}^{\lambda M} - \frac{n^2\sigma^2}{M}\right) \exp\left(-\lambda n t + \frac{n^2\sigma^2}{M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right),$$

which vanishes only for

$$\lambda_{\min} := \frac{1}{M} \ln\left(\frac{tM}{n\sigma^2} + 1\right) \in \mathbb{R}^+.$$

The second derivative $r''(\lambda_{\min}) > 0$ is positive at this point. Furthermore, $r'(0) < 0$ and $\lim_{\lambda\to\infty} r(\lambda) = \infty$. Therefore $\lambda_{\min}$ is the global minimum.

With the abbreviation

$$g \colon \quad \mathbb{R}^+ \to \mathbb{R}, \quad g(x) := (1+x)\ln(1+x) - x,$$

we have

$$\forall t \in \mathbb{R}^+ : \quad \mathbb{P}[\overline{X} \geq t] \leq \exp\left(-\frac{n^2\sigma^2}{M^2} g\left(\frac{tM}{n\sigma^2}\right)\right), \tag{B.11}$$

which is also called Bennett's inequality.

In the next step, g is bounded below by

$$h : \quad \mathbb{R}^+ \to \mathbb{R}, \quad h(x) := \frac{3}{2}\frac{x^2}{x+3},$$

i.e., we have

$$\forall x \in \mathbb{R}^+ : \quad h(x) \leq g(x).$$

This inequality is shown by differentiating both sides twice and checking the starting point $x = 0$. More precisely, in other words, we have $g(0) = 0 = h(0)$, $g'(0) = 0 = h'(0)$, and

$$\forall x \in \mathbb{R}^+ : \quad h''(x) = \frac{27}{(x+3)^3} \leq \frac{1}{x+1} = g''(x),$$

which implies the inequality by integrating twice.

Applying this last inequality to (B.11) yields

$$\forall t \in \mathbb{R}^+ : \quad \mathbb{P}[\overline{X} \geq t] \leq \exp\left(-\frac{n^2\sigma^2}{M^2} h\left(\frac{tM}{n\sigma^2}\right)\right) = \exp\left(\frac{-nt^2}{2(Mt/3 + n\sigma^2)}\right),$$

which concludes the proof. □

The last part of the proof, going from (B.11) to the final inequality, serves two purposes. First, it serves a cosmetic purpose, as the structure of the final inequality is much simpler than the one of (B.11). Second, and closely related to the cosmetic appeal, the scaling as the number n of random variables is increased and the influences of the bound M and the variance σ^2 can be discussed more easily in the final inequality.

On the other hand, the final inequality is less strict than (B.11). Therefore is better suited to obtain numerical bounds of the mean value $\overline{X}$.

Empirical Bernstein's Inequality

Anderson Inequality

B.8 Characteristic Functions

Definition B.57 (characteristic function)**.** The *characteristic function* ϕ_X of a random variable X is the expected value of e^{itX}, i.e.,

$$\phi_X : \quad \mathbb{R} \to \mathbb{C}, \quad \phi_X(t) := \mathbb{E}[e^{itX}] = \int_{\mathbb{R}} e^{itx} \mathrm{d}F_X(x) = \int_{\mathbb{R}} e^{itx} f_X(x) \mathrm{d}x,$$

where f_X is the probability density function of X and F_X its cumulative distribution function.

If the random variable has a probability density function f_X, then the characteristic function is its (inverse) Fourier transform up to a constant in the complex exponential.

If $\{X_i\}_{i\in\mathbb{N}}$ is a set of independent random variables and $a_i \in \mathbb{R}$ are constants, then the characteristic function ϕ_{S_n} of the linear combination

$$S_n := \sum_{i=1}^{n} a_i X_i$$

is given by

$$\phi_{S_n}(t) = \prod_{i=1}^{n} \phi_{X_i}(a_i t).$$

The characteristic function of the Delta distribution δ_a is

$$\phi_{\delta_a}(t) = \mathrm{e}^{\mathrm{i}ta}.$$

The characteristic function of the normal distribution $N(\mu, \sigma^2)$ is

$$\phi_{N(\mu,\sigma^2)}(t) = \exp\left(\mathrm{i}t\mu - \frac{\sigma^2 t^2}{2}\right).$$

The equality

$$\phi_X^{(k)}(0) = \mathrm{i}^k \mathbb{E}[X^k] \tag{B.12}$$

is useful since it relates the derivatives of the characteristic function at zero to the moments.

B.9 Types of Convergence

In order to define almost sure convergence, the limit supremum of a sequence of sets is needed.

Definition B.58 (limit infimum and limit supremum of a sequence of real numbers)**.** The *limit infimum* and the *limit supremum* of a sequence $\langle x_n \rangle_{n\in\mathbb{N}}$ of real numbers is defined as

$$\liminf_{n\to\infty} := \lim_{n\to\infty} \inf_{i\geq n} x_i,$$
$$\limsup_{n\to\infty} := \lim_{n\to\infty} \sup_{i\geq n} x_i.$$

Lemma B.59. *Suppose $\langle x_n \rangle_{n\in\mathbb{N}}$ is a sequence of real numbers. Then the equalities*

$$\begin{aligned}\liminf_{n\to\infty} x_n &= \sup_{n\in\mathbb{N}} \inf_{i\geq n} x_i,\\ \limsup_{n\to\infty} x_n &= \inf_{n\in\mathbb{N}} \sup_{i\geq n} x_i\end{aligned}$$

hold.

Definition B.60 (limit infimum, limit supremum, and limit of a sequence of sets)**.** Suppose that Ω is a set and that $\langle A_n \rangle_{n\in\mathbb{N}}$ is a sequence of subsets $A_n \subset \Omega$. Then the *limit infimum* and the *limit supremum* of the sequence $\langle A_n \rangle_{n\in\mathbb{N}}$ are defined as

$$\begin{aligned}\liminf_{n\to\infty} A_n &:= \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i,\\ \limsup_{n\to\infty} A_n &:= \bigcap_{n\in\mathbb{N}} \bigcup_{i\geq n} A_i.\end{aligned}$$

If the limit infimum and the limit supremum two sets are equal, the *limit* of the sequence $\langle A_n \rangle_{n\in\mathbb{N}}$ exists and is written as

$$\lim_{n\to\infty} A_n := \liminf_{n\to\infty} A_n = \limsup_{n\to\infty} A_n.$$

The following lemma shows how the limit infimum and the limit supremum of a sequence of sets can be written in terms of the limit infimum and the limit supremum of a sequence of real numbers and the indicator function $x \mapsto [\![x \in A_n]\!]$ (see Definition B.2) that indicates whether x is an element of A_n.

Lemma B.61. *Suppose that Ω is a set and that $\langle A_n \rangle_{n\in\mathbb{N}}$ is a sequence of subsets $A_n \subset \Omega$. Then the limit infimum and the limit supremum of the sequence $\langle A_n \rangle_{n\in\mathbb{N}}$ are equal to*

$$\begin{aligned}\liminf_{n\to\infty} A_n &= \{x \in \Omega : \liminf_{n\to\infty} [\![x \in A_n]\!] = 1\},\\ \limsup_{n\to\infty} A_n &= \{x \in \Omega : \limsup_{n\to\infty} [\![x \in A_n]\!] = 1\}.\end{aligned}$$

In other words, $x \in \liminf_{n\to\infty} A_n$ if and only if x is an element of all but finitely many sets A_n. Analogously, $x \in \limsup_{n\to\infty} A_n$ if and only if x is an element of infinitely many sets A_n.

Definition B.62 (almost sure convergence, convergence with probability one)**.** A sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ of random variables *converges almost surely* (or *converges with probability one*) to a random variable X if

$$\forall \epsilon \in \mathbb{R}^+ : \quad \mathbb{P}[\limsup_{n\to\infty} \{\omega \in \Omega : |X_n(\omega) - X(\omega)| > \epsilon\}] = 0$$

holds. We then write

$$X_n \xrightarrow[n\to\infty]{\text{a. s.}} X.$$

Definition B.63 (convergence in probability)**.** A sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ of random variables *converges in probability* to a random variable X if

$$\forall \epsilon \in \mathbb{R}^+ : \quad \lim_{n\to\infty} \mathbb{P}[|X_n - X| > \epsilon] = 0$$

holds. We then write

$$X_n \xrightarrow[n\to\infty]{\mathbb{P}} X.$$

Almost sure convergence implies convergence in probability.

Definition B.64 (convergence in distribution, weak convergence, convergence in law)**.** A sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ of real valued random variables with the cumulative distribution functions F_n *converges in distribution* (or *converges weakly* or *converges in law*) to a random variable X with the cumulative distribution function F if

$$\forall x \in \{x \in \mathbb{R} : F \text{ is continuous at } x\} : \quad \lim_{n\to\infty} F_n(x) = F(x)$$

holds. We then write

$$X_n \xrightarrow[n\to\infty]{\text{d}} X.$$

Convergence in probability implies convergence in distribution.

B.10 Lévy's Continuity Theorem

Theorem B.65 (Lévy's continuity theorem)**.** *Suppose that $\langle X_n \rangle_{n\in\mathbb{N}}$ is a sequence of random variables, not necessarily sharing a common probability space. If the sequence $\langle \phi_n \rangle_{n\in\mathbb{N}}$ of their characteristic functions converges pointwise to a function $\phi : \mathbb{R} \to \mathbb{C}$, i.e.,*

$$\lim_{n\to\infty} \phi_n(t) = \phi(t) \quad \forall t \in \mathbb{R},$$

then the following statements are equivalent:

1. *the X_n converge in distribution to a random variable X, i.e.,*

$$X_n \xrightarrow[n\to\infty]{\mathrm{d}} X;$$

2. *ϕ is the characteristic function of a random variable X;*

3. *ϕ is a continuous function;*

4. *ϕ is continuous at zero;*

5. *the sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ is tight, i.e.,*

$$\lim_{x\to\infty} \left(\sup_{n\in\mathbb{N}} \mathbb{P}[|X_n| > x]\right) = 0.$$

Proofs can be found in [66, Section 18.1] and in [67, Theorems 14.15 and 18.21].

B.11 The Laws of Large Numbers and the Central Limit Theorem

A natural question to ask how the mean value

$$\overline{X}_n := \frac{S_n}{n},$$

where

$$S_n := \sum_{i=1}^{n} X_i,$$

of n independent and identically distributed random variables X_i behaves as $n \to \infty$ and additionally how fast it converges if this is the case.

The answers are provided by the law of large numbers and the central limit theorem via an expansion. Informally speaking, the law of large numbers

$$\overline{X}_n = \frac{S_n}{n} \to \mu,$$

which holds if each X_i has finite mean μ, yields the first term, and the central limit theorem

$$\sqrt{n}(\overline{X}_n - \mu) = \frac{S_n - n\mu}{\sqrt{n}} \to \xi \sim N(0, \sigma^2),$$

which holds if additionally each X_i has finite variance σ^2, yields the second term in the informal expansion

$$\overline{X}_n \approx \mu + \frac{\xi}{\sqrt{n}}$$

or

$$S_n \approx \mu n + \xi\sqrt{n}.$$

In the following, the law of large numbers and the central limit theorem are stated and proven.

Theorem B.66 (weak law of large numbers)**.** *Suppose $\langle X_i \rangle_{i\in\mathbb{N}}$ is a sequence of independent and identically distributed integrable random variables with expected value $\mu := \mathbb{E}[X_i]$. Then*

$$\overline{X}_n \xrightarrow[n\to\infty]{\mathbb{P}} \mu.$$

Proof using Chebyshev's inequality and assuming finite variance. Under the additional assumption that all random variables X_i have finite variance, i.e., $\mathbb{V}[X_i] < \infty$ for all $i \in \mathbb{N}$, Chebyshev's inequality, Theorem B.51, can be used to show the weak law of large numbers.

Since the random variables are independent, we have

$$\mathbb{V}[\overline{X}_n] = \frac{1}{n^2}\mathbb{V}\left[\sum_{i=1}^{n} X_i\right] = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}.$$

Therefore applying Chebyshev's inequality, Theorem B.51, to $\overline{X}_n$ yields

$$\forall k \in \mathbb{R}^+ : \quad \mathbb{P}[|\overline{X}_n - \mu| \geq k] \leq \frac{\sigma^2}{k^2 n},$$

which implies

$$\forall k \in \mathbb{R}^+ : \quad \lim_{n\to\infty} \mathbb{P}[|\overline{X}_n - \mu| \geq k] = 0,$$

i.e.,

$$\overline{X}_n \xrightarrow[n\to\infty]{\mathbb{P}} \mu,$$

which concludes the proof. □

Proof using characteristic functions. The (complex) Taylor expansion around zero of the characteristic function ϕ_{X_1} of random variable X_1 with finite mean μ can be written as

$$\phi_{X_1}(t) = 1 + \mathrm{i}\mu t + o(t),$$

where $o(t)$ denotes a function that goes to zero more rapidly than t. Here we have used (B.12); for $k = 0$ we have $\phi_{X_1}(0) = 1$, and for $k = 1$ we find $\phi'_{X_1}(0) = \mathrm{i}\mu$. The same expansion holds for the other random variables, since they are all identically distributed.

Therefore the characteristic function $\phi_{\overline{X}_n}$ of the mean $\overline{X}_n$ is

$$\phi_{\overline{X}_n}(t) = \left(\phi_{X_1}\left(\frac{t}{n}\right)\right)^n = \left(1 + \mathrm{i}\mu\frac{t}{n} + o\left(\frac{t}{n}\right)\right)^n$$

Its limit is

$$\forall t \in \mathbb{R}\colon \quad \lim_{n\to\infty} \phi_{\overline{X}_n}(t) = \mathrm{e}^{\mathrm{i}\mu t}$$

due to the well-known limit $\lim_{n\to\infty}(1 + x/n)^n = \mathrm{e}^x$. The limit $\mathrm{e}^{\mathrm{i}\mu t}$ is the characteristic function of the constant random variable μ. Therefore, by Lévy's continuity theorem, Theorem B.65, the $\overline{X}_n$ converge in distribution to μ as $n \to \infty$, i.e.,

$$\overline{X}_n \xrightarrow[n\to\infty]{\mathrm{d}} \mu.$$

Finally, since μ is a constant, convergence in distribution to μ and convergence in probability to μ are equivalent. Therefore we even have

$$\overline{X}_n \xrightarrow[n\to\infty]{\mathbb{P}} \mu,$$

which concludes the proof. □

Theorem B.67 (strong law of large numbers)**.** *Suppose $\langle X_i\rangle_{i\in\mathbb{N}}$ is a sequence of independent and identically distributed integrable random variables with expected value $\mu := \mathbb{E}[X_i]$. Then*

$$\overline{X}_n \xrightarrow[n\to\infty]{\text{a. s.}} \mu.$$

Theorem B.68 (central limit theorem)**.** *Suppose $\langle X_i\rangle_{i\in\mathbb{N}}$ is a sequence of independent and identically distributed random variables X_i with expected value $\mu := \mathbb{E}[X_i]$ and finite variance $\sigma^2 := \mathbb{V}[X_i] < \infty$. Then*

$$\sqrt{n}(\overline{X}_n - \mu) \xrightarrow[n\to\infty]{\mathrm{d}} N(0, \sigma^2).$$

In other words, the pointwise convergence to the cumulative distribution function of the normal distribution $N(0, \sigma^2)$ means that

$$\forall x \in \mathbb{R}\colon \quad \lim_{n\to\infty} \mathbb{P}[\sqrt{n}(\overline{X}_n - \mu) \le x] = \Phi\left(\frac{x}{\sigma}\right),$$

where Φ is the cumulative distribution function of the standard normal distribution.

Proof. The classical proof uses characteristic functions. Since the random variables X_i are independent and identically distributed by assumption, their sum $\sum_{i=1}^{n} X_i$ has mean $n\mu$ and variance $n\sigma^2$. We define the random variables

$$Y_i := \frac{X_i - \mu}{\sigma}$$

that have zero mean and unit variance. Just as the X_i, they are also independent and identically distributed. Using the Y_i, we have

$$Z_n := \frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n\sigma^2}} = \sum_{i=1}^{n} \frac{Y_i}{\sqrt{n}}.$$

The characteristic function ϕ_{Z_n} of Z_n is the product

$$\phi_{Z_n}(t) = \phi_{\sum_{i=1}^{n} \frac{Y_i}{\sqrt{n}}}(t) = \prod_{i=1}^{n} \phi_{Y_i}\left(\frac{t}{\sqrt{n}}\right) = \left(\phi_{Y_1}\left(\frac{t}{\sqrt{n}}\right)\right)^n,$$

where the last equality holds since all the Y_i are identically distributed.

The Taylor expansion of the characteristic function ϕ_{Y_1} around zero starts with the terms

$$\phi_{Y_1}\left(\frac{t}{\sqrt{n}}\right) = 1 - \frac{t^2}{2n} + o\left(\frac{t^2}{2n}\right),$$

where $o(t^2/n)$ denotes a function that goes to zero more rapidly than t^2/n. To obtain this Taylor expansion, we have used (B.12); for $k = 0$ we have $\phi_{Y_1}(0) = 1$, for $k = 1$ we find $\phi'_{Y_1}(0) = 0$ since $\mathbb{E}[Y_1] = 0$, and for $k = 2$ we have $\phi''_{Y_1}(0) = -\mathbb{E}[Y_1^2] = -\mathbb{V}[Y_1] = -1$.

Using this Taylor expansion, we find the characteristic function of Z_n as

$$\phi_{Z_n}(t) = \left(1 - \frac{t^2}{2n} + o\left(\frac{t^2}{2n}\right)\right)^n,$$

which has the limit

$$\lim_{n\to\infty} \phi_{Z_n}(t) = \mathrm{e}^{-t^2/2}$$

due to the well-known limit $\lim_{n\to\infty}(1 + x/n)^n = \mathrm{e}^x$.

The limit $\mathrm{e}^{-t^2/2}$ is the characteristic function of the standard normal distribution $N(0, 1)$. Therefore, by Lévy's continuity theorem, Theorem B.65, the Z_n converge in distribution to $N(0, 1)$ as $n \to \infty$, i.e.,

$$Z_n \xrightarrow[n\to\infty]{\mathrm{d}} N(0, 1),$$

which implies

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n}} \xrightarrow[n\to\infty]{\mathrm{d}} N(0,\sigma^2).$$

Since

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n}} = \frac{n\overline{X}_n - n\mu}{\sqrt{n}} = \sqrt{n}(\overline{X}_n - \mu),$$

we find

$$\sqrt{n}(\overline{X}_n - \mu) \xrightarrow[n\to\infty]{\mathrm{d}} N(0,\sigma^2),$$

which concludes the proof. □

B.12 Wald's Equation

In its basic form, Wald's equation makes it possible to simplify a sum of random variables, whose number of terms is itself a random variable. More precisely, suppose that the number of terms in the sum is an integer valued random variable N such that $N \geq 1$ and that it is independent of the sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ of real valued, independent, and identically distributed random variables X_n to be summed. Then the expected value of the sum of N terms of X_n is given by

$$\mathbb{E}\left[\sum_{n=1}^{N} X_n\right] = \mathbb{E}[N]\mathbb{E}[X_1],$$

i.e., it is equal to the expected number of terms times the expected value of a single term, as can be expected since all random variables are independent.

More generally, the following theorem holds.

Theorem B.69 (Wald's equation)**.** *Suppose*

1. *that $\langle X_n \rangle_{n\in\mathbb{N}}$ is a sequence of real valued integrable random variables,*
2. *that N is an integer valued random variable such that $N(\Omega) \subset \mathbb{N} \setminus \{0\}$,*
3. *that $\mathbb{E}[X_n [\![n \leq N]\!]] = \mathbb{E}[X_n]\mathbb{P}[n \leq N]$ for all $n \in \mathbb{N}$, and*
4. *that the infinite sum*

$$\sum_{n=1}^{\infty} \mathbb{E}[|X_n| [\![n \leq N]\!]] < \infty$$

converges.

Then the random variables

$$S_N := \sum_{n=1}^{N} X_n,$$
$$T_N := \sum_{n=1}^{N} \mathbb{E}[X_n]$$

are integrable and have the same expected value, i.e.,

$$\mathbb{E}[S_N] = \mathbb{E}[T_N].$$

If additionally

5. *all random variables X_n, $n \in \mathbb{N}$, have the same expected value and*
6. *the random variable N is integrable,*

then Wald's equation

$$\mathbb{E}[S_N] = \mathbb{E}[N]\mathbb{E}[X_1]$$

holds.

Proof. In the first step, we show that the random variable S_N is integrable, i.e., $\mathbb{E}[|S_N|] < \infty$. Using the partial sums

$$S_i := \sum_{n=1}^{i} X_n, \quad i \in \mathbb{N},$$

we have

$$|S_N| = \sum_{i=1}^{\infty} |S_i| [\![i = N]\!].$$

The Lebesgue monotone convergence theorem, Theorem B.34, applied to the partial sums $k \mapsto \sum_{i=1}^{k} |S_i| [\![i = N]\!]$ means that integration and summation can be interchanged, yielding

$$\mathbb{E}[|S_N|] = \mathbb{E}\left[\sum_{i=1}^{\infty} |S_i| [\![i = N]\!]\right] = \sum_{i=1}^{\infty} \mathbb{E}[|S_i| [\![i = N]\!]].$$

The triangle inequality gives

$$\forall i \in \mathbb{N}\colon \quad |S_i| \leq \sum_{n=1}^{i} |X_n|$$

which implies

$$\mathbb{E}[|S_N|] = \sum_{i=1}^{\infty} \mathbb{E}[|S_i| [\![i = N]\!]] \leq \sum_{i=1}^{\infty} \sum_{n=1}^{i} \mathbb{E}[|X_n| [\![i = N]\!]].$$

Here the order of summation can be changed, since all terms are non-negative. Therefore we have the estimate

$$\mathbb{E}[|S_N|] \leq \sum_{n=1}^{\infty} \sum_{i=n}^{\infty} \mathbb{E}[|X_n| [\![i = N]\!]] = \sum_{n=1}^{\infty} \mathbb{E}[|X_n| [\![n \leq N]\!]].$$

Assumption 4 now yields $\mathbb{E}[|S_N|] < \infty$, i.e., the random variable S_N is integrable.

In the second step, we show that the random variable T_N is integrable, i.e., $\mathbb{E}[|T_N|] < \infty$. Using the partial sums

$$T_i := \sum_{n=1}^{i} \mathbb{E}[X_n], \quad i \in \mathbb{N},$$

we have

$$|T_N| = \sum_{i=1}^{\infty} |T_i| [\![i = N]\!].$$

Analogously to the first step, the Lebesgue monotone convergence theorem, Theorem B.34, yields

$$\mathbb{E}[|T_N|] = \mathbb{E}\left[\sum_{i=1}^{\infty} |T_i| [\![i = N]\!]\right] = \sum_{i=1}^{\infty} \mathbb{E}[|T_i| [\![i = N]\!]] = \sum_{i=1}^{\infty} |T_i| \mathbb{P}[i = N].$$

The triangle inequality gives

$$\forall i \in \mathbb{N}\colon \quad |T_i| \leq \sum_{n=1}^{i} |\mathbb{E}[X_n]|,$$

which implies

$$\mathbb{E}[|T_N|] \leq \sum_{i=1}^{\infty} \sum_{n=1}^{i} |\mathbb{E}[X_n]| \mathbb{P}[i = N].$$

Here the order of summation can be changed, since all terms are non-negative. Therefore we have the estimate

$$\mathbb{E}[|T_N|] \leq \sum_{n=1}^{\infty} \sum_{i=n}^{\infty} |\mathbb{E}[X_n]| \mathbb{P}[i = N] = \sum_{n=1}^{\infty} |\mathbb{E}[X_n]| \sum_{i=n}^{\infty} \mathbb{P}[i = N]$$

$$= \sum_{n=1}^{\infty} |\mathbb{E}[X_n]| \mathbb{P}[n \leq N].$$

Due to Assumption 3 and Jensen's inequality, Theorem B.48, we find the estimates

$$\forall n \in \mathbb{N}: \quad |\mathbb{E}[X_n]| \mathbb{P}[n \leq N] = |\mathbb{E}[X_n [\![n \leq N]\!]]| \leq \mathbb{E}[|X_n| [\![n \leq N]\!]]$$

for the single terms of the sum, which result in

$$\mathbb{E}[|T_N|] \leq \sum_{n=1}^{\infty} \mathbb{E}[|X_n| [\![n \leq N]\!]].$$

Assumption 4 now yields $\mathbb{E}[|T_N|] < \infty$, i.e., the random variable T_N is integrable.

In the third step, the expected value $\mathbb{E}[S_N]$ is calculated. The Lebesgue dominated convergence theorem, Theorem B.35, with the functions S_i and with the majorant $|S_N|$ (since $N \geq 1$) yields

$$\mathbb{E}[S_N] = \mathbb{E}\left[\sum_{i=1}^{\infty} S_i [\![i = N]\!]\right] = \sum_{i=1}^{\infty} \mathbb{E}[S_i [\![i = N]\!]],$$

which can also be written as

$$\mathbb{E}[S_N] = \sum_{i=1}^{\infty} \sum_{n=1}^{i} \mathbb{E}[X_n [\![i = N]\!]]$$

by substituting the definition of S_i. Because of the absolute convergence of this sum shown in the first step, the order of summation can be changed such that we arrive at

$$\mathbb{E}[S_N] = \sum_{n=1}^{\infty} \sum_{i=n}^{\infty} \mathbb{E}[X_n [\![i = N]\!]].$$

We again use the Lebesgue dominated convergence theorem, Theorem B.35, with the majorant $|X_N|$ to change the order of the expectation operator and the inner summation to find

$$\mathbb{E}[S_N] = \sum_{n=1}^{\infty} \sum_{i=n}^{\infty} \mathbb{E}[X_n [\![i = N]\!]] = \sum_{n=1}^{\infty} \mathbb{E}\left[\sum_{i=n}^{\infty} X_n [\![i = N]\!]\right] = \sum_{n=1}^{\infty} \mathbb{E}[X_n [\![n \leq N]\!]].$$

Due to Assumption 3 and the σ-additivity of the probability measure, the terms in the last sum are equal to

$$\mathbb{E}[X_n [\![n \leq N]\!]] = \mathbb{E}[X_n] \mathbb{P}[n \leq N] = \mathbb{E}[X_n] \sum_{i=n}^{\infty} \mathbb{P}[i = N],$$

which can be rewritten as

$$\mathbb{E}[X_n[\![n \leq N]\!]] = \sum_{i=n}^{\infty} \mathbb{E}[\mathbb{E}[X_n][\![i = N]\!]]$$

using the properties of the expected value. With these terms, the sum becomes

$$\mathbb{E}[S_N] = \sum_{n=1}^{\infty}\sum_{i=n}^{\infty} \mathbb{E}[\mathbb{E}[X_n][\![i = N]\!]] = \sum_{i=1}^{\infty}\sum_{n=1}^{i} \mathbb{E}[\mathbb{E}[X_n][\![i = N]\!]],$$

where we could change the order of summation due to the absolute convergence shown in the first step, i.e., $\mathbb{E}[|S_N|] < \infty$. By the definition of T_i, this is equal to

$$\mathbb{E}[S_N] = \sum_{i=1}^{\infty} \mathbb{E}[T_i[\![i = N]\!]] = \sum_{i=1}^{\infty} \mathbb{E}[T_N[\![i = N]\!]].$$

The Lebesgue dominated convergence theorem, Theorem B.35, with the majorant $|T_N|$ makes it possible to change the order of the expectation operator and the summation to find

$$\mathbb{E}[S_N] = \sum_{i=1}^{\infty} \mathbb{E}[T_N[\![i = N]\!]] = \mathbb{E}\left[\sum_{i=1}^{\infty} T_N[\![i = N]\!]\right].$$

Furthermore, we can calculate

$$\mathbb{E}[S_N] = \mathbb{E}\left[T_N \sum_{i=1}^{\infty}[\![i = N]\!]\right] = \mathbb{E}[T_N[\![1 \leq N]\!]] = \mathbb{E}[T_N],$$

since the codomain of the random variable N is $\mathbb{N} \setminus \{0\}$. This proves the third statement of the theorem.

If Assumptions 5 and 6 are additionally satisfied, then $\mathbb{E}[T_N]$ can be simplified to

$$\mathbb{E}[T_N] = \mathbb{E}\left[\sum_{n=1}^{N} \mathbb{E}[X_n]\right] = \mathbb{E}[X_1]\mathbb{E}\left[\sum_{n=1}^{N} 1\right] = \mathbb{E}[N]\mathbb{E}[X_1],$$

which completes the proof. □

In the last line of the proof, it becomes clear that it is only required that the expected values $\mathbb{E}[X_n]$ of the random variables are identical; this is sufficient to lift them out of the sum and the outer expected value. In particular, the random variables are *not* required to be independent.

The last line of the proof also explains why all the sums in the statement of the theorem start at one and why zero is excluded from the codomain of the random variable N in Assumption 2. These two facts match such that $\mathbb{E}[\sum_{n=1}^{N} 1] = \mathbb{E}[N]$.

Appendix C

Stochastic Approximation

Stochastic approximation is concerned with iterative methods for finding roots or solving optimization problems. The functions whose roots or extreme values are approximated cannot be computed directly, but only estimated via random samples. Stochastic approximation is of great importance in machine learning and artificial intelligence for two reasons. First, optimization problems are fundamental in machine learning. Second, in machine learning, the samples are drawn from a data distribution, meaning that only random samples of the objective functions are available.

C.1 Dvoretzky's Approximation Theorem

The following two theorems are Dvoretzky's approximation theorem and Dvoretzky's extended approximation theorem [68], cited according to [69, Theorems 4 and 5].

Theorem C.1 (Dvoretzky's approximation theorem)**.** *Suppose that*

1. $(\Omega, \mathcal{F}, \mathbb{P})$ *is a probability space,*
2. $\langle \mathcal{F}_n \rangle_{n \in \mathbb{N}}$ *is an increasing sequence of sub-σ-fields of $\mathcal{F}$,*
3. $\langle X_n \rangle_{n \in \mathbb{N}}$ *are $\mathcal{F}_n$-measurable random real-valued variables,*
4. $\langle T_n \colon \mathbb{R}^n \to \mathbb{R} \rangle_{n \in \mathbb{N}}$ *is a sequence of measurable functions,*
5. $\langle W_n \rangle_{n \in \mathbb{N}}$ *are $\mathcal{F}_{n+1}$-measurable real-valued random variables such that*

$$X_{n+1} = T_n(x_1, \dots, x_n) + W_n,$$

6. $\mathbb{E}[W_n|\mathcal{F}_n] = 0$,
7. $\sum_{n\in\mathbb{N}} \mathbb{E}[W_n^2] < \infty$,
8. $\langle a_n\rangle_{n\in\mathbb{N}}$, $\langle b_n\rangle_{n\in\mathbb{N}}$, *and* $\langle c_n\rangle_{n\in\mathbb{N}}$ *are sequences of non-negative real numbers,*
9. $\lim_{n\to\infty} a_n = 0$,
10. $\sum_{n\in\mathbb{N}} b_n < \infty$,
11. $\sum_{n\in\mathbb{N}} c_n = \infty$, *and*
12. *there exists a point* $x_* \in \mathbb{R}$ *such that*

$$|T_n(x_1, \dots, x_n) - x_*| \leq \max(a_n, (1 + b_n)|x_n - x_*| - c_n) \quad \forall n \in \mathbb{N}.$$

Then the sequence $\langle X_n\rangle_{n\in\mathbb{N}}$ *of random variables converges to* x_* *with probability one, i.e.,*

$$\mathbb{P}\Big[\lim_{n\to\infty} X_n = x_*\Big] = 1.$$

In the extended version of this theorem, the three sequences $\langle a_n\rangle_{n\in\mathbb{N}}$, $\langle b_n\rangle_{n\in\mathbb{N}}$, and $\langle c_n\rangle_{n\in\mathbb{N}}$ are promoted to real-valued functions, and the T_n are promoted to $\mathcal{F}_n$-measurable random variables.

Theorem C.2 (Dvoretzky's extended approximation theorem)**.** *Suppose that*

1. $(\Omega, \mathcal{F}, \mathbb{P})$ *is a probability space,*
2. $\langle\mathcal{F}_n\rangle_{n\in\mathbb{N}}$ *is an increasing sequence of sub-*σ*-fields of* $\mathcal{F}$*,*
3. $\langle X_n\rangle_{n\in\mathbb{N}}$ *are* $\mathcal{F}_n$*-measurable random real-valued variables,*
4. $\langle T_n \colon \mathbb{R}^n \to \mathbb{R}\rangle_{n\in\mathbb{N}}$ *is a sequence of* $\mathcal{F}_n$*-measurable real-valued random variables,*
5. $\langle W_n\rangle_{n\in\mathbb{N}}$ *are* $\mathcal{F}_{n+1}$*-measurable real-valued random variables such that*

$$X_{n+1} = T_n(x_1, \dots, x_n) + W_n,$$

6. $\mathbb{E}[W_n|\mathcal{F}_n] = 0$,
7. $\sum_{n\in\mathbb{N}} \mathbb{E}[W_n^2] < \infty$,
8. $\langle a_n \colon \Omega \to \mathbb{R}_0^+\rangle_{n\in\mathbb{N}}$, $\langle b_n \colon \Omega \to \mathbb{R}_0^+\rangle_{n\in\mathbb{N}}$, *and* $\langle c_n \colon \Omega \to \mathbb{R}_0^+\rangle_{n\in\mathbb{N}}$ *are sequences of real-valued non-negative functions,*

9. $\mathbb{P}\left[\lim_{n\to\infty} a_n = 0\right] = 1$,
10. $\mathbb{P}\left[\sum_{n\in\mathbb{N}} b_n < \infty\right] = 1$,
11. $\mathbb{P}\left[\sum_{n\in\mathbb{N}} c_n = \infty\right] = 1$, *and*
12. *there exists a point* $x_* \in \mathbb{R}$ *such that*

$$|T_n(x_1, \dots, x_n) - x_*| \leq \max(a_n, (1 + b_n)|x_n - x_*| - c_n) \quad \forall n \in \mathbb{N}.$$

Then the sequence $\langle X_n \rangle_{n\in\mathbb{N}}$ *of random variables converges to* x_* *with probability one, i.e.,*

$$\mathbb{P}\left[\lim_{n\to\infty} X_n = x_*\right] = 1.$$

C.2 Venter's Generalization

In Venter's generalization, the T_n are transformations of $H^n \times \Omega$ into itself, where H is a Hilbert space. The following is [70, Theorem 1].

Theorem C.3 (Venter's generalization: almost sure convergence)**.** *Suppose that* $(\Omega, \mathcal{F}, \mathbb{P})$ *is a probability space and that* $(H, \| \, \|)$ *is a real separable Hilbert space. For all* $n \in \mathbb{N}$*, let* $T_n \colon H^n \times \Omega \to H^n \times \Omega$ *be a transformation of* $H^n \times \Omega$ *into itself. Let* $\theta \in H$*. Let* N *be a (finite) integer-valued random variable on* Ω *and suppose that for each sequence* $\langle x_n \rangle_{n\in\mathbb{N}}$ *in* H *and for* $\omega \in \Omega_0 \in \mathcal{F}$ *with* $\mathbb{P}[\Omega_0] = 1$*, we have, for* $n > N(\omega)$*,*

$$\|T_n(x_1, \dots, x_n, \omega) - \theta\|^2 \leq \max(a, (1 + b_n)\|x_n - \theta\|^2 - c_n),$$

where

1. a *is a positive constant,*
2. b_n *is a non-negative real-valued function on* $H^n \times \Omega$ *such that*

$$b_n(x_1, \dots, x_n, \omega) \leq K_1 \quad \text{and} \quad \sum_{n\in\mathbb{N}} b_n(x_1, \dots, x_n, \omega) < \infty$$

   for all sequences $\langle x_n \rangle$ *in* H *and for all* $\omega \in \Omega_0$*,*

3. c_n *is a real-valued function on* $H^n \times \Omega$ *such that for all* $\langle x_n \rangle$ *in* H*,*

$$c_n(x_1, \dots, x_n, \omega) \geq 0 \quad \text{if } n > N(\omega)$$

and if $\omega \in \Omega_0$, *while, if*

$$\sup_{n\in\mathbb{N}} \|x_n\| < \infty,$$

then

$$\sum_{n\in\mathbb{N}} c_n(x_1, \dots, x_n, \omega) = \infty.$$

Let X_1 *be an arbitrary random element in* H *and let the sequence* $\langle X_n\rangle_{n\in\mathbb{N}}$ *satisfy*

$$X_{n+1}(\omega) = T_n(X_1(\omega), \dots, X_n(\omega), \omega) + U_n(\omega),$$

where $\langle U_n\rangle_{n\in\mathbb{N}}$ *is a sequence of random elements satisfying the conditions*

$$\sum_{n\in\mathbb{N}} \mathbb{E}[\|U_n\|^2] < \infty \tag{C.1}$$

and

$$\mathbb{P}\left[\sum_{n\in\mathbb{N}} \|\mathbb{E}[U_n|\mathcal{F}_n]\| < \infty\right] = 1, \tag{C.2}$$

where $\langle \mathcal{F}_n\rangle_{n\in\mathbb{N}}$ *is an increasing sequence of sub-*σ*-fields of* $\mathcal{F}$ *having the properties that the random elements* $\{X_1, \dots, X_n, T_1(X_1), \dots, T_n(X_1, \dots, X_n)\}$ *are measurable with respect to* $\mathcal{F}_n$ *for* $n \in \{2, 3, \dots\}$ *and that*

$$\{\omega \in \Omega : n > N(\omega)\} \in \mathcal{F}_n. \tag{C.3}$$

Then

$$\mathbb{P}\left[\limsup_{n\to\infty} \|X_n - \theta\|^2 \le a\right] = 1.$$

Under the conditions of this theorem, the sequence $\langle X_n\rangle$ is stochastically attracted towards the sphere with center θ and radius $a^{1/2}$. The elements X_n will eventually almost surely be within or arbitrarily close to this sphere.

The following is [70, Theorem 2]. It states that under somewhat stronger conditions, $\langle X_n\rangle$ will eventually be within or close to this sphere in the mean square (or quadratic mean) sense.

Theorem C.4 (Venter's generalization: convergence in mean square)**.** *Consider the set-up in Theorem C.3 and let the conditions be strengthened as follows:* N *is a fixed finite integer and* $\langle b_n\rangle_{n\in\mathbb{N}}$ *a fixed sequence of non-negative numbers such that*

$$\sum_{n\in\mathbb{N}} b_n < \infty.$$

$\langle U_n \rangle_{n\in\mathbb{N}}$ satisfies (C.1) *and instead of* (C.2),

$$\sum_{n\in\mathbb{N}} \mathbb{E}\big[\|\mathbb{E}[U_n|\mathcal{F}_n]\|^2\big]^{1/2} < \infty.$$

Other conditions remain unchanged except in so far as they are changed by the new conditions introduced. Thus e.g. (C.3) *becomes redundant. If*

$$\mathbb{E}[\|X_N\|^2] < \infty$$

then

$$\limsup_{n\to\infty} \mathbb{E}[\|X_n - \theta\|^2] \le a.$$

Note that if the positive constant $a \in \mathbb{R}$ can be chosen arbitrarily small, then Theorem C.3 gives conditions for almost sure convergence of $\langle X_n \rangle$ to θ, and Theorem C.4 gives conditions for convergence in mean square (or in quadratic mean) of $\langle X_n \rangle$ to θ.

C.3 Example: Perturbed Fixed-Point Iteration

A well-known example of a fixed-point iteration is the approximation of π using the contraction

$$K\colon \quad I \to I, \quad x \mapsto x + \sin x$$

on the interval $I := [3\pi/4, 5\pi/4]$. The function K maps I to I since $K'(x) = 1 + \cos x \ge 0$ for all $x \in I$, $3\pi/4 \approx 2.356 < 3.063 \approx K(3\pi/4)$, and $K(5\pi/4) \approx 3.220 < 3.927 \approx 5\pi/4$.

Furthermore, the function K is indeed a contraction since it is continuously differentiable with $0 \le K'(x) \le 1 - 1/\sqrt{2}$ for all $x \in I$ (note $\cos(5\pi/4) = -1/\sqrt{2}$). Therefore the Lipschitz constant

$$q := \max_{x\in I} K'(x) = 1 - \frac{1}{\sqrt{2}} < 1$$

serves as the contraction factor.

Therefore, the assumptions of the Banach fixed-point theorem are satisfied. The resulting fixed-point iteration is

$$x_{n+1} := K(x_n) = x_n + \sin x_n, \qquad x_0 := 3 \in I,$$

and we know

$$\exists! x_* \in I\colon \quad K(x_*) = x_*,$$

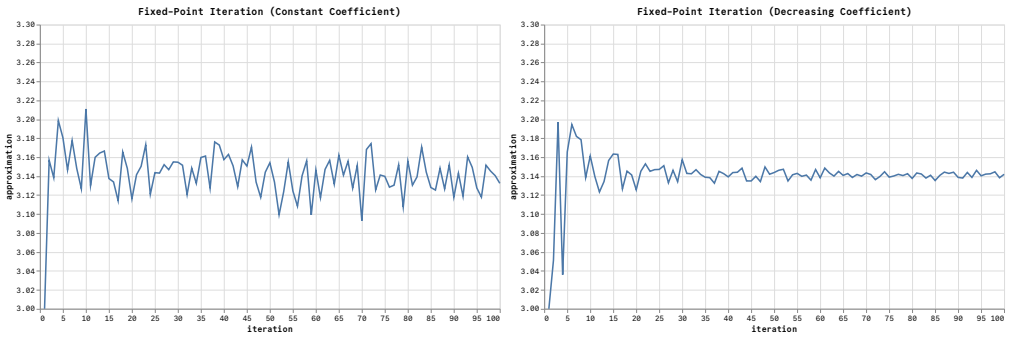


Figure C.1: Perturbed fixed-point iteration (C.4) with constant perturbation $0.02 \cdot N(0,1)$ (left) and decreasing perturbation $(0.2/n) \cdot N(0,1)$ (right), where n is the iteration number and $N(0,1)$ means drawing a random number according to the standard normal distribution. The initial value is 3 in all cases.

$$\lim_{n \to \infty} x_n = x_*.$$

Letting $n \to \infty$ in the iteration yields $\sin(x_*) = 0$, showing that $x_* \in I$ is indeed $x_* = \pi$.

Unfortunately, numerical calculations involving real numbers are not exact, but include errors. This is true in our example as well, also because the sine must be approximated. This fact strongly motivates the consideration of the perturbed fixed-point iteration

$$X_{n+1} := K(X_n) + W_n, \tag{C.4}$$

a stochastic process, where the W_n are random variables representing the error in each iteration. Hence the iterates X_n are random variables as well, denoted by capital letters. We assume that

$$\mathbb{E}[W_n] = 0,$$

since a biased error should not occur or – if it does – should become part of the fixed-point operator.

Figure C.1 shows numerical calculations for the perturbed fixed-point iteration.

Does the perturbed fixed-point iteration still converge? In which sense? To which limit? We use the theorems in Section C.1 and Section C.2 to answer these fundamental questions.

Theorem C.5 (convergence of perturbed fixed-point iteration)**.** *Suppose that* $(H, \|\,\|)$ *is a Hilbert space and that the function* $K \colon H \to H$ *is a contraction with*

respect to the norm $\|\,\|$ *of the Hilbert space. Denote the unique fixed point of the contraction* K *by* x_**. Suppose that* $(\Omega, \mathcal{F}, \mathbb{P})$ *is a probability space and that* W *is a random variable with zero mean, i.e.,* $\mathbb{E}[W] = 0$*, and finite variance, i.e.,* $\mathbb{V}[W] < \infty$*. Suppose further that* $\langle \lambda_n \rangle_{n\in\mathbb{N}}$ *is a sequence with* $\lambda_n \geq 0$ *for all* $n \in \mathbb{N}$ *and* $\sum_{n\in\mathbb{N}} \lambda_n^2 < \infty$*. Then the perturbed fixed-point iteration*

$$X_{n+1} := K[X_n] + \lambda_n W$$

converges to x_* *with probability one, i.e.,*

$$\mathbb{P}\Big[\lim_{n\to\infty} X_n = x_*\Big] = 1.$$

Proof. The aim is to apply Theorem C.1, which generalizes to Hilbert spaces because of Theorem C.3. We try to find sequences $\langle a_n \rangle_{n\in\mathbb{N}}$, $\langle b_n \rangle_{n\in\mathbb{N}}$, and $\langle c_n \rangle_{n\in\mathbb{N}}$ satisfying the condition

$$\|K(x_n) - x_*\| \leq \max(a_n, (1 + b_n)\|x_n - x_*\| - c_n) \quad \forall n \in \mathbb{N}.$$

Because of

$$\|K(x_n) - x_*\| = \|K(x_n) - K(x_*)\| \leq q\|x_n - x_*\| \quad \forall n \in \mathbb{N},$$

we try to find sequences such that

$$q\|x_n - x_*\| \leq \max(a_n, (1 + b_n)\|x_n - x_*\| - c_n) \quad \forall n \in \mathbb{N}.$$

Suppose that the second argument of the maximum is relevant as $n \to \infty$. Then the estimate

$$c_n \leq (b_n + 1 - q)\|x_n - x_*\| \quad \forall n \in \mathbb{N}$$

would hold. However, $\lim_{n\to\infty} b_n = 0$ necessarily holds, and all contractions satisfy the inequality

$$\|x_n - x_*\| \leq \frac{q^n}{1-q}\|x_1 - x_0\| \quad \forall n \in \mathbb{N}.$$

Therefore there exists a constant $C \in \mathbb{R}^+$ such that $c_n \leq Cq^n\|x_1 - x_0\|$ for sufficiently large n, which contradicts $\sum_{n\in\mathbb{N}} c_n = \infty$.

Next, suppose that the first argument of the maximum is relevant as $n \to \infty$, i.e.,

$$q\|x_n - x_*\| \leq a_n \quad \forall n \in \mathbb{N}.$$

Setting

$$a_n := \frac{q^{n+1}}{1-q}\|x_1 - x_0\|$$

satisfies this inequality and the condition $\lim_{n\to\infty} a_n = 0$. To ensure that the first argument of the maximum is relevant, we must finally ensure that

$$(1 + b_n)\|x_n - x_*\| - c_n \leq a_n \quad \forall n \in \mathbb{N},$$

which is equivalent to

$$(1 + b_n)\|x_n - x_*\| - \frac{q^{n+1}}{1-q}\|x_1 - x_0\| \leq c_n \quad \forall n \in \mathbb{N}.$$

The left side can be estimated by

$$(1 + b_n)\|x_n - x_*\| - \frac{q^{n+1}}{1-q}\|x_1 - x_0\| \leq (1 + b_n - q)\frac{q^n}{1-q}\|x_1 - x_0\| \leq c_n \quad \forall n \in \mathbb{N}.$$

Setting

$$\begin{aligned} b_n &:= 0, \\ c_n &:= \|x_1 - x_0\| \end{aligned}$$

satisfies all conditions.

In summary, the assumptions of Theorem C.1, which also holds for sequences $\langle X_n \rangle$ in the Hilbert space H because of Theorem C.3, are satisfied. □

C.4 Polyak and Tsypkin's Theorem

In [71], a general theorem regarding the convergence of various stochastic algorithms for minimizing functionals, based on the notion of the pseudogradient, is shown. The following is [71, Theorem 1].

Theorem C.6 (Polyak and Tsypkin's theorem)**.** *Suppose that*

1. $(H, \|\,\|)$ *is a Hilbert space,*
2. *the function* $J\colon H \to \mathbb{R}$ *is differentiable and bounded from below, i.e.,* $J(\mathbf{c}) \geq J_* > -\infty$ *for all* $\mathbf{c} \in H$*,*
3. *the gradient of* J *is Lipschitz continuous, i.e.,*

$$\exists L \in \mathbb{R}\colon \forall \mathbf{x}, \mathbf{y} \in H\colon \quad \|\nabla J(\mathbf{x}) - \nabla J(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \tag{C.5}$$

4. *the sequence* $\langle \mathbf{c}_n \rangle_{n\in\mathbb{N}} \subset H$ *is generated by* $\mathbf{c}_0 \in H$ *being arbitrary and*

$$\mathbf{c}_n := \mathbf{c}_{n-1} - \gamma_n \mathbf{s}_n,$$

where $\langle \gamma_n \rangle_{n\in\mathbb{N}} \subset \mathbb{R}_0^+$,

5. *the sequence* $\langle \mathbf{s}_n \rangle_{n\in\mathbb{N}} \subset H$ *consists of pseudogradients of* J, *i.e.,*

$$\forall n \in \mathbb{N}\colon \quad \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle \geq 0, \tag{C.6}$$

6. *the step lengths are constrained by*

$$\exists \langle \lambda_n \rangle_{n\in\mathbb{N}} \subset \mathbb{R}_0^+\colon \quad \exists\exists K_1, K_2 \in \mathbb{R}_0^+\colon \\ \mathbb{E}[\|\mathbf{s}_n\|^2] \leq \lambda_n + K_1 J(\mathbf{c}_{n-1}) + K_2 \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle, \tag{C.7}$$

7. *the sequence* $\langle \gamma_n \rangle_{n\in\mathbb{N}} \subset \mathbb{R}_0^+$ *satisfies*

$$\sum_{n\in\mathbb{N}} \gamma_n = \infty, \tag{C.8}$$

8. *the sequences* $\langle \gamma_n \rangle_{n\in\mathbb{N}}$ *and* $\langle \lambda_n \rangle_{n\in\mathbb{N}}$ *satisfy*

$$\sum_{n\in\mathbb{N}} \gamma_n^2 \lambda_n < \infty, \tag{C.9}$$

9. *either*

$$\sum_{n\in\mathbb{N}} \gamma_n^2 < \infty \tag{C.10}$$

or

$$\forall n \in \mathbb{N}\colon \lambda_n = 0 \quad \wedge \quad K_1 = 0 \quad \wedge \quad \limsup_{n\to\infty} \gamma_n < \frac{2}{LK_2}. \tag{C.11}$$

Then the limit $\lim_{n\to\infty} J(\mathbf{c}_n)$ *almost surely exists, and*

$$\liminf_{n\to\infty} \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle = 0 \quad \textit{almost surely.}$$

Based on this theorem, the following three corollaries are stated in [71].

Corollary C.7. *In addition to the assumptions in Theorem C.6, suppose that for all* $\epsilon \in \mathbb{R}^+$ *the inequality* $J(\mathbf{c}_{n-1}) \geq J_* + \epsilon$ *implies*

$$\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle \geq \delta(\epsilon) > 0,$$

where $\delta\colon \mathbb{R}^+ \to \mathbb{R}^+$ *is a monotonically increasing function. Then*

$$\lim_{n\to\infty} J(\mathbf{c}_n) = J_* \quad \textit{almost surely.}$$

Corollary C.8. *In addition to the assumptions in Theorem C.6, suppose that H is a finite-dimensional space ($H = \mathbb{R}^d$), that sets of the form $\{\mathbf{c} \in H : J(\mathbf{c}) \geq \text{const.}\}$ are bounded, and that for all $\epsilon \in \mathbb{R}^+$ the inequality $\|\nabla J(\mathbf{c}_{n-1})\| \geq \epsilon$ implies*

$$\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle \geq \delta(\epsilon) > 0,$$

where $\delta \colon \mathbb{R}^+ \to \mathbb{R}^+$ is a monotonically increasing function. Then there almost surely exist a subsequence $\langle n_i \rangle \subset \mathbb{N}$ and points $\mathbf{c}_*$ such that

$$\begin{aligned} \nabla J(\mathbf{c}_*) &= 0, \\ \lim_{i \to \infty} \mathbf{c}_{n_i} &= \mathbf{c}_* \quad \textit{almost surely}, \\ \lim_{i \to \infty} J(\mathbf{c}_{n_i}) &= J(\mathbf{c}_*) \quad \textit{almost surely}. \end{aligned}$$

For any subset $C \subset H$ and any point $c \in H$, the distance from $\mathbf{c}$ to C is denoted by

$$\rho(\mathbf{c}, C) := \inf_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{c}\|$$

in the following.

Corollary C.9. *In addition to the assumptions in Theorem C.6, suppose that the set C_* of minimum points of J is not empty, that for all $\epsilon \in \mathbb{R}^+$ the inequality $\rho(\mathbf{c}, C_*) \geq \epsilon$ implies*

$$\inf_{\mathbf{c} \in H} J(\mathbf{c}) > J_*,$$

and that for all $\epsilon \in \mathbb{R}^+$ the inequality $\rho(\mathbf{c}, C_*) \geq \epsilon$ implies

$$\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n] \rangle \geq \delta(\epsilon) > 0,$$

where $\delta \colon \mathbb{R}^+ \to \mathbb{R}^+$ is a monotonically increasing function. Then

$$\begin{aligned} \lim_{n \to \infty} \rho(\mathbf{c}_n, C_*) &= 0 \quad \textit{almost surely}, \\ \lim_{n \to \infty} J(\mathbf{c}_n) &= J_* \quad \textit{almost surely}. \end{aligned}$$

In particular, if C_* consists of the single point $\mathbf{c}_*$, then

$$\lim_{n \to \infty} \mathbf{c}_n = \mathbf{c}_* \quad \textit{almost surely}.$$

Proof of the theorem. The proof follows [71, Theorem 1].

By (C.5), Taylor expansion of J around $\mathbf{c}$ yields

$$\forall \forall \mathbf{c}, \mathbf{z} \in H \colon \quad |J(\mathbf{c} + \mathbf{z}) - J(\mathbf{c}) - \langle \nabla J(\mathbf{c}), \mathbf{z} \rangle| \leq \frac{L}{2} \|\mathbf{z}\|_2^2.$$

By setting $\mathbf{c} := \mathbf{c}_n$ and $\mathbf{z} := \gamma_n \mathbf{s}_n$, we have $\mathbf{c} + \mathbf{z} = \mathbf{c}_{n-1}$ and find

$$J(\mathbf{c}_n) \leq J(\mathbf{c}_{n-1}) - \gamma_n \langle \nabla J(\mathbf{c}_{n-1}), \mathbf{s}_n \rangle + \frac{L}{2} \gamma_n^2 \|\mathbf{s}_n\|_2^2.$$

We define $\mathcal{F}_{n-1}$ to be the minimum σ-algebra generated by the random variables $\mathbf{s}_1, \dots, \mathbf{s}_{n-1}$, and we define

$$\mu_n := J(\mathbf{c}_n) - J_*.$$

Subtracting J_* and taking the conditional expectation of both sides of the last inequality yields

$$\begin{aligned} \mathbb{E}[\mu_n | \mathcal{F}_{n-1}] &\leq \mu_{n-1} - \gamma_n \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n | \mathcal{F}_{n-1}] \rangle + \frac{L}{2} \gamma_n^2 \mathbb{E}[\|\mathbf{s}_n\|_2^2 | \mathcal{F}_{n-1}] \\ &\leq \mu_{n-1} \left(1 + \frac{K_1 L}{2} \gamma_n^2\right) - \gamma_n \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n | \mathcal{F}_{n-1}] \rangle \left(1 - \frac{K_2 L}{2} \gamma_n\right) \\ &\quad + \frac{L}{2} \gamma_n^2 \lambda_n + \frac{K_1 L}{2} \gamma_n^2 J_*, \end{aligned} \tag{C.12}$$

where we have used assumption (C.7) and the equation $J(\mathbf{c}_{n-1}) = \mu_{n-1} + J_*$. Because of assumption (C.10) or (C.11), the inequality

$$1 - \frac{K_2 L}{2} \gamma_n \geq 0$$

holds for sufficiently large n. Therefore and because of assumption (C.6), the second term on the right side of (C.12) is non-negative (before it is subtracted). Therefore inequality (C.12) simplifies to

$$\mathbb{E}[\mu_n | \mathcal{F}_{n-1}] \leq \mu_{n-1} \left(1 + \frac{K_1 L}{2} \gamma_n^2\right) + \frac{L}{2} \gamma_n^2 \lambda_n + \frac{K_1 L}{2} \gamma_n^2 J_*.$$

Next, we show that $\langle \mu_n \rangle_{n \in \mathbb{N}}$ is a semimartingale, i.e., $\mathbb{E}[\mu_n | \mathcal{F}_{n-1}] \leq \mu_{n-1}$ for all $n \in \mathbb{N}$, under assumption (C.10) or (C.11).

Under assumption (C.10), …

Under assumption (C.11), we have $K_1 = 0$ and $\lambda_n = 0$, showing that $\langle \mu_n \rangle$ is a semimartingale.

In both cases, $\langle \mu_n \rangle$ is semimartingale, and hence $\lim_{n \to \infty} \mu_n$ almost surely exists, and the unconditional expectations $\mathbb{E}[\mu_n]$ are uniformly bounded, i.e.,

$$\exists a \in \mathbb{R}_0^+ \colon \quad \forall n \in \mathbb{N} \colon \quad \mathbb{E}[\mu_n] \leq a. \tag{C.13}$$

We note that all conclusions until now remain valid if assumption (C.10) is replaced by the weaker assumption $\lim_{n \to \infty} \gamma_n = 0$.

Therefore we can replace the conditional expectations in (C.12) by unconditional expectations, yielding

$$\mathbb{E}[\mu_n] \leq \left(1 + \frac{K_1 L}{2}\gamma_n^2\right)\mathbb{E}[\mu_{n-1}] - \gamma_n\left(1 - \frac{K_2 L}{2}\gamma_n\right)\mathbb{E}[\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n|\mathcal{F}_{n-1}]\rangle] + \frac{L}{2}\gamma_n^2\lambda_n + \frac{K_1 L}{2}\gamma_n^2 J_*.$$

Next, we sum the last inequality over n from 1 to ∞. Using (C.13) as well as assumption (C.9), we obtain

$$\frac{K_1 L}{2}\sum_{n\in\mathbb{N}}\gamma_n^2\mathbb{E}[\mu_{n-1}] < \infty,$$
$$\frac{L}{2}\sum_{n\in\mathbb{N}}\gamma_n^2\lambda_n < \infty,$$
$$\frac{K_1 L}{2}|J_*|\sum_{n\in\mathbb{N}}\gamma_n^2 < \infty$$

for both assumptions (C.10) and (C.11). Hence, summing the inequality implies

$$\sum_{n\in\mathbb{N}}\gamma_n\left(1 - \frac{K_2 L}{2}\gamma_n\right)\mathbb{E}[\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n|\mathcal{F}_{n-1}]\rangle] < \infty.$$

For both assumptions (C.10) and (C.11) and for sufficiently large n, the estimate

$$1 - \frac{K_2 L}{2}\gamma_n \geq \epsilon > 0$$

holds, which implies

$$\sum_{n\in\mathbb{N}}\gamma_n\mathbb{E}[\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n|\mathcal{F}_{n-1}]\rangle] < \infty.$$

Assumption (C.6) means $\langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n|\mathcal{F}_{n-1}]\rangle \geq 0$, and assumption (C.8) means $\sum_{n\in\mathbb{N}}\gamma_n = \infty$. Therefore, the last inequality implies that there exists a subsequence $\langle n_i\rangle_{i\in\mathbb{N}}$ such that

$$\lim_{i\to\infty}\mathbb{E}[\langle \nabla J(\mathbf{c}_{n_i-1}), \mathbb{E}[\mathbf{s}_{n_i}|\mathcal{F}_{n_i-1}]\rangle] = 0.$$

The last equation implies that there exists another subsequence $\langle n_{i_j}\rangle_{j\in\mathbb{N}}$ such that

$$\lim_{j\to\infty}\langle \nabla J(\mathbf{c}_{n_{i_j}-1}), \mathbb{E}[\mathbf{s}_{n_{i_j}}|\mathcal{F}_{n_{i_j}-1}]\rangle = 0 \quad \text{almost surely.}$$

Because of assumption (C.6), the last equation implies

$$\liminf_{n\to\infty} \langle \nabla J(\mathbf{c}_{n-1}), \mathbb{E}[\mathbf{s}_n|\mathcal{F}_{n-1}]\rangle = 0,$$

which concludes the proof.

□

The following theorem is useful for checking the pseudogradient condition (C.6) in certain situations. There are three points: x_* (the limit), x_1 (the current iterate), and x_2. We assume that the point x_2 is closer (with respect to the L^2-norm) to the limit x_* than the current iterate x_1. Then the angle between $x_* - x_1$ and $x_2 - x_1$ is acute.

Theorem C.10 (acute angle)**.** *Suppose that H is a Hilbert space, and that the three arbitrary points $x_* \in H$, $x_1 \in H$, and $x_2 \in H$ satisfy the inequality*

$$\exists \gamma \in [0,1]: \quad \|x_2 - x_*\|_2 \leq \gamma \|x_1 - x_*\|_2.$$

Then the inequality

$$\langle x_* - x_1, x_2 - x_1\rangle \geq (1-\gamma)\|x_* - x_1\|_2^2 \geq 0$$

holds.

Proof. We start from the equation

$$\langle x_* - x_1, x_* - x_2\rangle + \langle x_* - x_1, x_2 - x_1\rangle = \langle x_* - x_1, x_* - x_1\rangle = \|x_* - x_1\|_2^2,$$

which yields

$$\frac{\langle x_* - x_1, x_* - x_2\rangle}{\|x_* - x_1\|_2} + \frac{\langle x_* - x_1, x_2 - x_1\rangle}{\|x_* - x_1\|_2} = \|x_* - x_1\|_2. \tag{C.14}$$

In order to estimate the first term, we use the Cauchy-Schwarz inequality and the assumption to find

$$\langle x_* - x_1, x_* - x_2\rangle \leq \|x_* - x_1\|_2 \|x_* - x_2\|_2 \leq \gamma \|x_* - x_1\|_2^2,$$

which implies

$$\frac{\langle x_* - x_1, x_* - x_2\rangle}{\|x_* - x_1\|_2} \leq \gamma \|x_* - x_1\|_2.$$

Equation (C.14) and the last inequality imply

$$\frac{\langle x_* - x_1, x_2 - x_1\rangle}{\|x_* - x_1\|_2} \geq (1-\gamma)\|x_* - x_1\|_2 \geq 0,$$

which concludes the proof. □

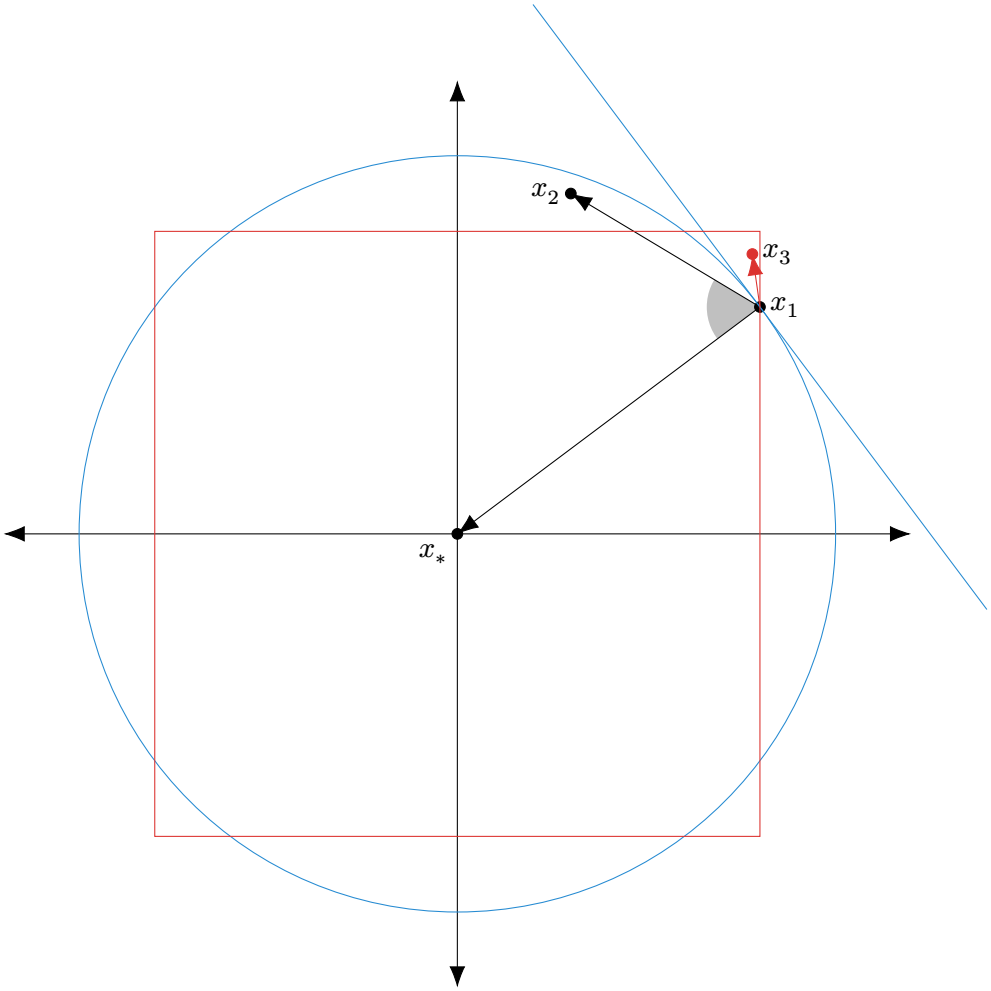


Figure C.2: Illustration of the geometry of the three points x_*, x_1, and x_2 in Theorem C.10.

Equation (C.14) has a geometric interpretation. We project the point x_2 onto $x_* - x_1$. The first term is the length of the line segment from x_* to the projection of x_2, and the second term is the length of the line segment from the projection of x_2 to x_1; of course, the sum of the two lengths is the length of the line segment $x_* - x_1$.

Unfortunately, Theorem C.10 does not hold for the maximum norm, as illustrated in Figure C.2. In an application of Theorem C.6, x_* would be the limit, x_1 the current iterate, and $x_2 - x_1$ the search direction. Then Theorem C.10

can be used to show the pseudogradient condition, as the angle

$$\phi = \arccos \frac{\langle x_* - x_1, x_2 - x_1 \rangle}{\|x_* - x_1\|_2 \|x_2 - x_1\|_2}$$

indicated in the figure is acute. The blue line is normal to $x_* - x_1$, which is the direction of the gradient, and therefore all points that lie in the same half-space as x_* with respect to the blue line satisfy the pseudogradient condition. In particular, all points x_2 that are closer to x_* than x_1 with respect to the L^2-norm – this is the assumption in Theorem C.10 and indicated by the blue circle – satisfy the pseudogradient condition.

However, this is not true for points x_3 that are closer to x_* than x_1 with respect to the maximum norm, i.e., if $\|x_3 - x_*\|_\infty \leq \|x_1 - x_*\|_\infty$ holds. All points with the same maximum norm $\|x_1 - x_*\|_\infty$ as the vector $x_1 - x_*$ are indicated by the red square. Points such as x_3 satisfy this condition on the distance, but lie in the opposite half-space with respect to the blue line and hence their angle is not acute.

This effect is known as the maximization bias of Q-learning. Maximization bias is a known problem of Q-learning and means that the action-value function is overestimated, as taking the maximum of random approximations may lead to some state-action pairs to appear more valuable than they are. In the present context, we can interpret maximization bias as moving away (with respect to the Euclidean norm) from the true value.

C.5 Bibliographical and Historical Remarks

The Robbins-Monro algorithms was introduced in 1951 by Herbert Robbins and Sutton Monro in a paper entitled “a Stochastic Approximation Method” [72]. Dvoretzky’s theorem was published in 1956 [68]. A generalization to variables that take values in generic Hilbert spaces was shown in [70]. A survey of stochastic approximation can be found in [73]. A formalization of Dvoretzky’s theorem was given in [69].

C.6 Exercises

Exercise C.1 (implementation of sine)**.** Find out how sine is implemented in your computer.

Exercise C.2 (perturbed approximation of π)**.** Implement the perturbed approximation of π shown in Figure C.1. Plot histograms for both cases and discuss them.

Exercise C.3 (an inequality for squares)**.** Show that the inequality

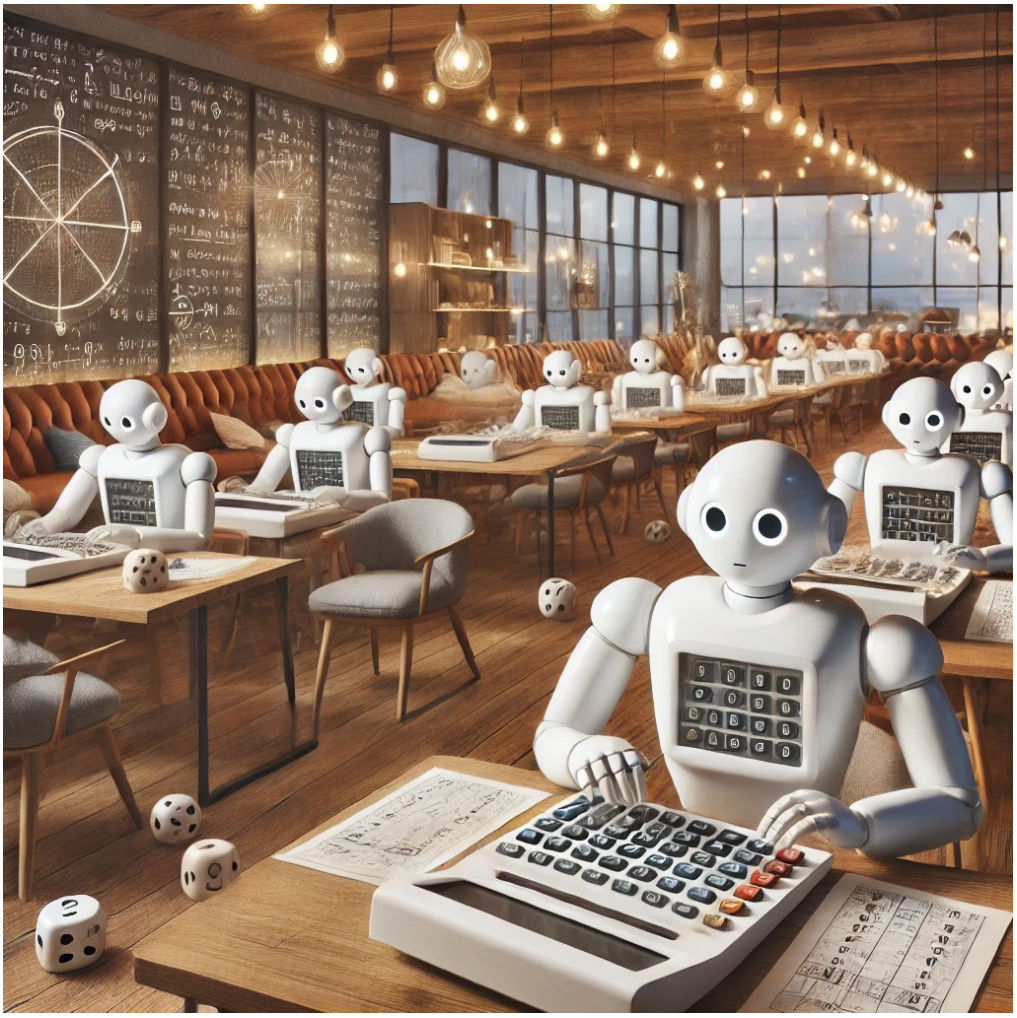
$$\forall a \in \mathbb{R} \colon \forall b \in \mathbb{R} \colon \quad (a+b)^2 \leq 2(a^2+b^2) \tag{C.15}$$

holds.

Exercise C.4 (equivalence of L^2- and L^∞-norms)**.** Show that the inequalities

$$\forall x \in \mathbb{R}^d \colon \quad \|x\|_\infty \leq \|x\|_2 \leq \sqrt{d}\|x\|_\infty \tag{C.16}$$

hold.



Appendix D

Software Libraries

This chapter contains an overview of software libraries that may be useful in implementing RL systems. The overview is in alphabetical order and probably not exhaustive.

D.1 Reinforcement Learning

D.1.1 Environments and Applications

1. BTGym is an event driven RL financial backtesting library.

   https://github.com/Kismuz/btgym

2. Gymnasium is a fork of OpenAI Gym. It is an open-source Python library for developing and comparing RL algorithms by providing a standard API to communicate between learning algorithms and environments. It also provides a standard set of environments.

   https://github.com/Farama-Foundation/Gymnasium

3. MuJoCo is a library for advanced physics simulation.

   https://mujoco.org
   https://github.com/google-deepmind/mujoco

4. OpenAI Gym provides a standard API to communicate between learning algorithms and environments as well as a standard set of environments. It has been superseded by Gymnasium.

   https://github.com/openai/gym

5. Personae provides RL algorithms and environments for quantitative trading.

   https://github.com/Ceruleanacg/Personae

6. RecSim is a platform for authoring simulation environments for recommender systems [20].

   https://github.com/google-research/recsim

7. TensorTrade is an RL library for trading.

   https://github.com/tensortrade-org/tensortrade

D.1.2 Learning

1. Awesome Deep RL is a list of deep RL libraries.

   https://github.com/kengz/awesome-deep-rl

2. CleanRL is a deep RL library that provides single-file implementations.

   https://github.com/vwxyzjn/cleanrl

3. DeepMind Acme is research framework for RL.

   https://github.com/google-deepmind/acme

4. DeepMind OpenSpiel is a framework for RL in games.

   https://github.com/google-deepmind/open_spiel

5. Google Dopamine is a research framework for prototyping RL algorithms.

   https://github.com/google/dopamine

6. Keras RL provides deep RL algorithms and integration with the deep-learning library Keras.

   https://github.com/keras-rl/keras-rl

7. Meta Pearl calls itself a production ready RL library.

   https://pearlagent.github.io
   https://github.com/facebookresearch/pearl/

8. Mushroom RL is Python RL library providing classical and deep RL algorithms.

   https://github.com/MushroomRL/mushroom-rl

9. OpenAI Baselines is in maintenance mode.

   https://github.com/openai/baselines

10. PFRL is a deep RL library based on PyTorch.

    https://github.com/pfnet/pfrl

11. Ray RLlib is an open-source library with support for a variety of industry applications. RLlib is part of the Ray AI libraries.

    https://github.com/ray-project/ray
    https://docs.ray.io/en/master/rllib

12. Stable Baselines is a fork of the OpenAI Baselines library.

    https://github.com/hill-a/stable-baselines

13. TensorFlow Agents is a RL library for TensorFlow.

    https://www.tensorflow.org/agents/overview

14. Tensorforce is a TensorFlow library for deep RL.

    https://github.com/tensorforce/tensorforce

15. Tianshou is an RL library based on PyTorch and Gymnasium.

    https://github.com/thu-ml/tianshou/

D.1.3 Policy Evaluation

1. DICE, the Distribution Correction Estimation library.

   https://github.com/google-research/dice_rl

D.2 Artificial Neural Networks / Deep Learning

1. Flax can be used to use JAX with artificial neural networks.

   https://flax.readthedocs.io/en/latest

2. JAX is library for high-performance numerical computing and large-scale machine learning.

   https://jax.readthedocs.io/en/latest

3. Keras is a deep-learning API written in Python that runs on top of either JAX, TensorFlow, or PyTorch.

   https://keras.io

4. PyTorch is more research focused.

   https://pytorch.org

5. TensorFlow is more industry focused.

   https://www.tensorflow.org

D.3 Large Language Models

1. LLMs from Scratch contains code for coding, pre-training, and fine-tuning LLM.

   https://github.com/rasbt/LLMs-from-scratch

2. minGPT is a reimplementation of GPT training and inference using PyTorch and well-suited for educational purposes.

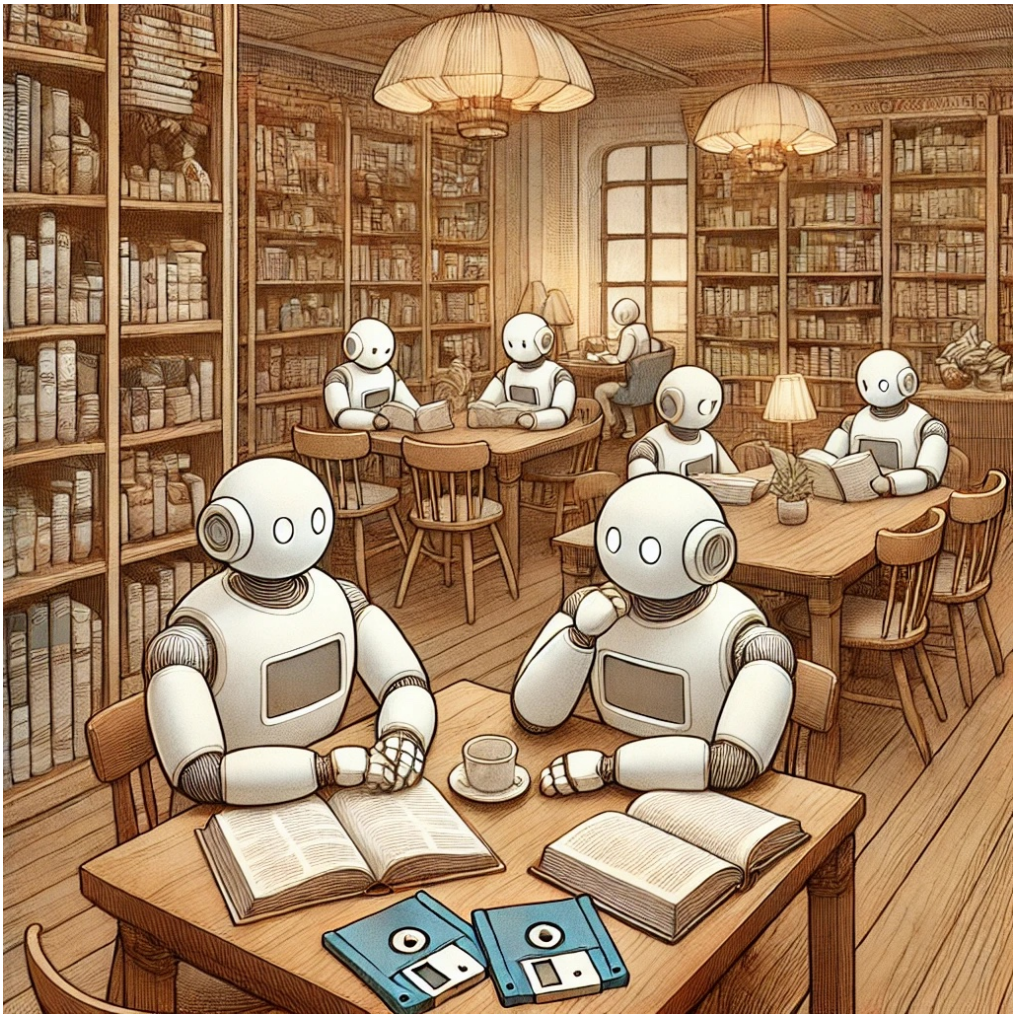
   https://github.com/karpathy/minGPT

3. nanoGPT is a rewrite of minGPT by the same author.

   https://github.com/karpathy/nanoGPT

4. spaCy is a Python library for natural language processing and contains support for embeddings.

   https://spacy.io
   https://pypi.org/project/spacy/



Bibliography

[1] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: an open urban driving simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[2] Peter R. Wurman et al. "Outracing champion Gran Turismo drivers with deep reinforcement learning". In: *Nature* 602 (2022), pp. 223–228. DOI: 10.1038/s41586-021-04357-7.

[3] Helmut Horvath. "Deep reinforcement learning with applications to autonomous driving". MA thesis. Vienna, Austria: TU Wien, 2024. URL: http://Clemens.Heitzinger.name.

[4] Tobias Kietreiber. "Combining maximum entropy reinforcement learning with distributional Q-value approximation methods". MA thesis. Vienna, Austria: TU Wien, 2023. URL: http://Clemens.Heitzinger.name.

[5] Pierrick Lorang, Horvath Helmut, Tobias Kietreiber, Patrik Zips, Clemens Heitzinger, and Matthias Scheutz. "Adapting to the "open world": the utility of hybrid hierarchical reinforcement learning and symbolic planning". In: *Proc. 2024 IEEE International Conference on Robotics and Automation (ICRA 2024). At press.* May 2024. DOI: TBD. URL: TBD.

[6] Gerald Tesauro. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68. DOI: 10.1145/203330.203343.

[7] "TD-Gammon". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Springer US, 2010, pp. 955–956. DOI: 10.1007/978-0-387-30164-8_813.

[8] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359. DOI: 10.1038/nature24270.

[9] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362 (2018), pp. 1140–1144.

[10] Noam Brown and Thomas Sandholm. “Superhuman AI for multiplayer poker”. In: *Science* 365.6456 (2019), pp. 885–890. DOI: 10.1126/science.aay2400.

[11] Tobias Salzer. “Reinforcement learning for games with imperfect information – teaching an agent the game of Schnapsen”. MA thesis. Vienna, Austria: TU Wien, 2023. URL: http://Clemens.Heitzinger.name.

[12] Stephen W. Falken. “Computers and Theorem Proofs: Toward an Artificial Intelligence”. Cited in *War Games* (1983). PhD thesis. Cambridge, MA, USA: MIT, 1960.

[13] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533.

[14] Max Jaderberg et al. “Human-level performance in 3D multiplayer games with population-based reinforcement learning”. In: *Science* 364 (2019), pp. 859–865.

[15] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. *Deep reinforcement learning for sepsis treatment*. 2017. arXiv: 1711.09602.

[16] Markus Böck, Julien Malle, Daniel Pasterk, Hrvoje Kukina, Ramin Hasani, and Clemens Heitzinger. “Superhuman performance on sepsis MIMIC-III data by distributional reinforcement learning”. In: *PLOS ONE* 17.11 (2022). Impact factor of *PLOS ONE:* 3.752., e0275358/1–18. DOI: 10.1371/journal.pone.0275358. URL: https://doi.org/10.1371/journal.pone.0275358.

[17] Razvan Bologheanu, Lorenz Kapral, Daniel Laxar, Mathias Maleczek, Christoph Dibiasi, Sebastian Zeiner, Asan Agibetov, Ari Ercole, Patrick Thoral, Paul Elbers, Clemens Heitzinger, and Oliver Kimberger. “Development of a reinforcement learning algorithm to optimize corticosteroid therapy in critically ill patients with sepsis”. In: *Journal of Clinical Medicine* 12.4 (2023). Impact factor of *Journal of Clinicial Medicine:* 4.964., pp. 1513/1–13. DOI: 10.3390/jcm12041513. URL: https://doi.org/10.3390/jcm12041513.

[18] Jongchan Baek, Changhyeon Lee, Young Sam Lee, Soo Jeon, and Soohee Han. “Reinforcement learning to achieve real-time control of triple inverted pendulum”. In: *Engineering Applications of Artificial Intelligence* 128 (2024), p. 107518. DOI: 10.1016/j.engappai.2023.107518.

[19] Carlotta Tubeuf, Jakob aus der Schmitten, René Hofmann, Clemens Heitzinger, and Felix Birkelbach. "Improving control of energy systems with reinforcement learning: application to a reversible pump turbine". In: *Proc. 18th ASME International Conference on Energy Sustainability (ES 2024). At press.* Anaheim, CA, USA, July 2024, TBD. DOI: TBD. URL: TBD.

[20] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. *RecSim: a configurable simulation platform for recommender systems.* 2019. arXiv: 1909.04847.

[21] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. *Reinforcement learning based recommender systems: a survey.* 2021. arXiv: 2101.06286.

[22] Federico Tomasi, Joseph Cauteruccio, Surya Kanoria, Kamil Ciosek, Matteo Rinaldi, and Zhenwen Dai. "Automatic music playlist generation via simulation-based reinforcement learning". In: *Proc. 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2023).* 2023, pp. 4948–4957. DOI: 10.1145/3580305.3599777.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an Introduction.* 2nd edition. The MIT Press, 2018.

[24] Dimitri P. Bertsekas. *Reinforcement learning and optimal control.* Athena Scientific, 2019.

[25] Dimitri P. Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning.* Athena Scientific, 2020.

[26] Dimitri P. Bertsekas. *Abstract dynamic programming.* Athena Scientific, 2022.

[27] Csaba Szepesvári. *Algorithms for Reinforcement Learning.* Morgan and Claypool Publishers, 2010.

[28] Marc G. Bellemare, Will Dabney, and Mark Rowland. *Distributional Reinforcement Learning.* MIT Press, 2023.

[29] Warren B. Powell. *Reinforcement Learning and Stochastic Optimization: a Unified Framework for Sequential Decisions.* John Wiley & Sons, Inc., 2022.

[30] Laura Graesser and Wah Loon Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python.* Addison-Wesley, 2020.

[31] Maxim Lapan. *Deep Reinforcement Learning Hands-On.* 2nd edition. Packt Publishing, 2020.

[32] Miguel Morales. *Grokking Deep Reinforcement Learning.* Manning Publications Co., 2020.

[33] Alexander Zai and Brandon Brown. *Deep Reinforcement Learning in Action*. Manning Publications Co., 2020.

[34] Richard Bellman. *Eye of the Hurricane – an Autobiography*. World Scientific Publishing, 1984.

[35] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[36] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[37] S.P. Singh and R.S. Sutton. "Reinforcement learning with replacing eligibility traces". In: *Machine Learning* 22.1–3 (1996), pp. 123–158.

[38] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37. DOI: 10.1126/science.153.3731.34.

[39] Christopher J.C.H. Watkins. "Learning from Delayed Rewards". PhD thesis. University of Cambridge, 1989.

[40] Christopher J.C.H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[41] Harold J. Kushner and Dean S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag New York Inc., 1978.

[42] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. "On the convergence of stochastic iterative dynamic programming algorithms". In: *Neural Computation* 6.6 (1994), pp. 1185–1201.

[43] John N. Tsitsiklis. "Asynchronous stochastic approximation and Q-learning". In: *Machine Learning* 16 (1994), pp. 185–202.

[44] Michael L. Littman and Csaba Szepesvári. "A generalized reinforcement-learning model: convergence and applications". In: *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*. Ed. by Lorenzo Saitta. Morgan Kaufmann, 1996, pp. 310–318.

[45] Csaba Szepesvári and Michael L. Littman. *Generalized Markov decision processes: dynamic-programming and reinforcement-learning algorithms*. Tech. rep. Technical Report CS-96-11. Providence, Rhode Island, USA: Department of Computer Science, Brown University, Nov. 1996.

[46] István Szita, Bálint Takács, and András Lőrincz. "ϵ-MDPs: learning in varying environments". In: *Journal of Machine Learning Research* 3 (2002), pp. 145–174.

[47] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization methods for large-scale machine learning”. In: *SIAM Review* 60.2 (2018), pp. 223–311.

[48] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in Neural Information Processing Systems 12 (NIPS 1999)*. Ed. by S.A. Solla, T.K. Leen, and K. Müller. MIT Press, 2000, pp. 1057–1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf.

[49] Wendell H. Fleming and Halil Mete Soner. *Controlled Markov Processes and Viscosity Solutions*. 2nd edition. Springer, 2006.

[50] Rémi Munos. “A study of reinforcement learning in the continuous case by the means of viscosity solutions”. In: *Machine Learning* 40.3 (2000), pp. 265–299.

[51] Michael G. Crandall and Pierre-Louis Lions. “Viscosity solutions of Hamilton-Jacobi equations”. In: *Trans. Amer. Math. Soc.* 277.1 (1983), pp. 1–42.

[52] Lawrence C. Evans. *Partial Differential Equations*. 2nd edition. American Mathematical Society, 2010.

[53] Michael G. Crandall, Hitoshi Ishii, and Pierre-Louis Lions. “User’s guide to viscosity solutions of second order partial differential equations”. In: *Bull. Amer. Math. Soc.* 27.1 (1992), pp. 1–67.

[54] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–489.

[55] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575 (2019), pp. 350–354.

[56] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. “Rainbow: combining improvements in deep reinforcement learning”. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-2018)*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 3215–3222. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204.

[57] Markus Böck and Clemens Heitzinger. “Speedy categorical distributional reinforcement learning and complexity analysis”. In: *SIAM Journal on Mathematics of Data Science* 4.2 (2022), pp. 675–693. DOI: 10.1137/20M1364436. URL: https://doi.org/10.1137/20M1364436.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Ed. by I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 6000–6010. arXiv: 1706.03762. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[59] O. Press and L. Wolf. *Using the output embedding to improve language models*. 2016. arXiv: 1608.05859.

[60] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155.

[61] Clemens Heitzinger. *Algorithms with Julia – Optimization, Machine Learning, and Differential Equations using the Julia Language*. ISBN 978-3-031-16559-7, ISBN 978-3-031-16560-3 (eBook), DOI: 10.1007/978-3-031-16560-3, https://doi.org/10.1007/978-3-031-16560-3. Springer Nature, 2022.

[62] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal policy optimization algorithms*. 2017. arXiv: 1707.06347.

[63] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. *Trust region policy optimization*. 2015. arXiv: 1502.05477.

[64] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[65] Robert J. Serfling. "Probability inequalities for the sum in sampling without replacement". In: *Annals of Statistics* 2.1 (1974), pp. 39–48.

[66] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[67] B.E. Fristedt and L.F. Gray. *A Modern Approach to Probability Theory*. Birkhäuser, 1996.

[68] Aryeh Dvoretzky. "On stochastic approximation". In: *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability (1954–1955)*. University of California Press, 1956, pp. 39–55.

[69] Koundinya Vajjha, Barry Trager, Avraham Shinnar, and Vasily Pestun. "Formalization of a stochastic approximation theorem". In: *13th International Conference on Interactive Theorem Proving (ITP 2022)*. Ed. by June Andronick and Leonardo de Moura. Vol. 237. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl

– Leibniz-Zentrum für Informatik, 2022, 31:1–31:18. ISBN: 978-3-95977-252-5. DOI: 10.4230/LIPIcs.ITP.2022.31. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2022.31.

[70] J.H. Venter. "On Dvoretzky stochastic approximation theorems". In: *Annals of Mathematical Statistics* 37.6 (1966), pp. 1534–1544. DOI: 10.1214/aoms/1177699145. URL: https://doi.org/10.1214/aoms/1177699145.

[71] B.T. Polyak and Ya.Z. Tsypkin. "Pseudogradient adaptation and training algorithms". In: *Automation and Remote Control* 34.3 (1973). Translation of Russian original, pp. 377–397.

[72] Herbert Robbins and Sutton Monro. "A stochastic approximation model". In: *Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586.

[73] Tze Leung Lai. "Stochastic approximation (invited paper)". In: *Annals of Statistics* 31.2 (2003), pp. 391–406. DOI: 10.1214/aos/1051027873. URL: https://doi.org/10.1214/aos/1051027873.

List of Algorithms

Index