REINFORCEMENT LEARNING: ALGORITHMS & CONVERGENCE

CLEMENS HEITZINGER
Clemens.Heitzinger@TUWien.ac.at
http://Clemens.Heitzinger.name

March 8, 2021

Contents

Preface

Reinforcement learning is a general concept that encompasses many real-world applications of machine learning. This book collects the mathematical foundations of reinforcement learning and describes its most powerful and useful algorithms.

The mathematical theory of reinforcement learning mainly comprises results on the convergence of methods and the analysis of algorithms. The methods treated in this book concern predication and control and include n-step methods, actor-critic methods, etc.

In order to make these concepts concrete and useful in applications, the book also includes a number of examples that exhibit both the advantages and disadvantages of the methods as well as the challenges faced when developing algorithms for reinforcement learning. The examples and the source code accompanying the book are an invitation to the reader to further explore this fascinating subject.

As reinforcement learning has developed into a sizable research area, it was necessary to focus on the main algorithms and methods of proof, although many variants have been developed. In any case, a complete and self-contained treatment of these is given. The more advanced theoretic sections are marked with a star ($*$) and can be skipped on first reading.

This book can be used in various ways. It can be used for self-study, but it can also be used as the basis of courses. In courses, the emphasis can be on the mathematical theory, on the algorithms, or on the applications.

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

Vienna, December 2019

Part I

Algorithms

Chapter 1

Introduction

We start by introducing the basic concept of reinforcement learning and the notions used in problem formulations. Reinforcement learning is a very general concept and applies to many time-dependent problems whenever an agent interacts with its environment. The main goal in reinforcement learning is to find optimal policies for the agent to follow.

1.1 The Concept of Reinforcement Learning

One of the major appeals of reinforcement learning 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. First, the environment and the agent are (possibly randomly) initialized. In each new time step t (top right), the environment goes into the next state S_t and rewards the agent with the reward R_t. Then the agent chooses the next action A_t according to its policy and interacts with the environment. In the next iteration, the environment goes into a new state depending on the action chosen and rewards the agent, and so forth.

In this manner, sequences of states S_t, actions A_t, and rewards R_t are generated. The sequences may be infinite (at least theoretically speaking) or they may end in a terminal state (which always happens in practice), corresponding to a finite number of time steps. Each such sequence of states S_t, actions A_t, and 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 framework, which can be done under reasonable assumptions.

The main goal in reinforcement learning is to find optimal policies for the agent to follow. The input to a policy is the current state the agent finds itself in, and the output of a policy is the action to take. Policies may be stochastic, just as environments may be stochastic. The policies should be optimal in the sense that the agent maximizes the expected value of the sum of all (discounted)

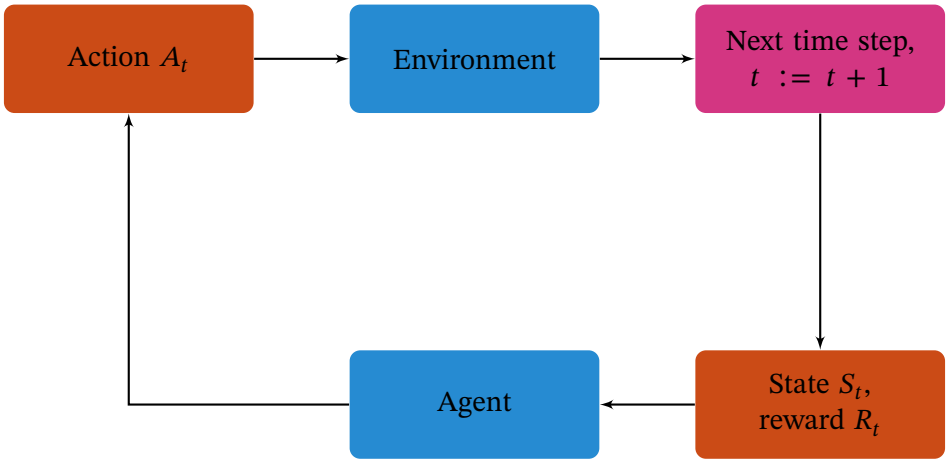

Figure 1.1: Environments, agents, and their interactions.

rewards it will receive.

The main purpose of this book is to present methods and algorithms that calculate such optimal policies.

1.2 Examples of Problem Formulations

The concept of reinforcement learning clearly encompasses many realistic problems. Any situation where an agent interacts with an environment and tries to optimize the sum of its rewards is a problem that is part of the realm of reinforcement learning, irrespective of the cardinality of the sets of spaces and actions and irrespective whether the environment is deterministic or stochastic.

Clearly, the more complicated the environment is, the more challenging the reinforcement-learning problem is. Thanks to sophisticated algorithms and, in some cases, to huge amounts of computational resources, it has become possible to solve realistic problems at the level of human intelligence or above.

Some examples are the following.

Robotics: In robotics, the roboter interacts with its environment in order to solve the task it has been assigned. In the ideal case, one defines the task for the roboter by specifying the rewards and/or penalties, which are usually straightforward to specify, and it learns to solve the problem without any further help or interaction.

Supply chains and inventories: In supply-chain and inventory management, goods have to moved around such that they arrive at their points of destination in sufficient quantity. Moving and storing goods is costly, however.

Rewards are given out whenever the goods arrive in the specified quantity at their points of destination, while penalties are given out when there are too few or when they are delayed.

Optimal control: Examples for the optimal control of industrial systems are chemical and biological reactors. The output of the product is to be maximized, while the reactions must occur within safe conditions.

Medicine: Similarly to the optimal control of systems, optimal policies for treating patients can be calculated based on measurements of their physical condition [1].

Computer games: When playing computer games, the computer game is the environment and the agent has to learn the optimal strategy. The same algorithm can learn to play many (but not all) Atari 2600 games at the human level or above [2]. A few years later, a reinforcement-learning algorithm learned to play *Quake III Arena* in *Capture the Flag* mode at the human level [3].

Go: After chess, Go (Weiqi) was the only remaining classical board game where humans could play better than computers. Go games lead to much larger search spaces than chess, which is the reason why Go was the last unsolved board game. In [4], a reinforcement-learning 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.

Chess, shogi, and Go: In the next step, the more general reinforcement-learning algorithm called *AlphaZero* in [5] learned to play the three games of chess, shogi (Japanese chess), and Go again *tabula rasa* and using only self-play. It defeats world-champion programs in each of the three games.

Autonomous driving: In autonomous driving, the agent has to traverse a stochastic environment quickly and safely. Sophisticated simulation environments including sensor output already exist for autonomous driving [6].

1.3 Overview of Methods

Here a short overview of the methods presented in this book are given. When reading the book first, many of the terms and notions will be unknown to the reader and this overview will only serve as a rough guideline to the variety of methods in reinforcement learning. The overview is meant to be useful on the second reading or when looking for a method particularly amenable to a given problem, since each method comes with a table that indicates its most important features and for which kind of problems it can be used.

1.3.1 Markov Decision Processes

Markov decision processes are, strictly speaking, not a solution method in reinforcement learning, but they serve as the starting point and the theoretical framework for describing transitions between states. The notations for states, actions, rewards, returns, etc. are fixed in Chapter 2 and used throughout the book.

1.3.2 Dynamic Programming

Dynamic programming is, of course, a whole field by itself. In dynamic programming, knowledge of the complete probability distributions of all possible transitions is required; in other words, the environment must be completely known. It can therefore be viewed as a special case of reinforcement learning, as the environment may be completely unknown in reinforcement learning.

Because dynamic programming is an important special case, Chapter 2 provides a summary of the most important techniques of dynamic programming at the beginning of the book. The following chapter, Chapter 3, reuses some ideas, but relaxes the assumption on the what must be known about the environment. For the types of problems indicated in Table 1.1, dynamic programming is the state of the art.

Cardinality of the set of states:	finite.
Cardinality of the set of actions:	finite.
Must the environment be known?	Yes, completely.
Function approximation for the policy?	No.

Table 1.1: When dynamic programming can be used.

1.3.3 Monte-Carlo Methods

Cardinality of the set of states:	finite.
Cardinality of the set of actions:	finite.
Must the environment be known?	No. A model to generate sample transitions is useful.
Function approximation for the policy?	No.

Table 1.2: When Monte-Carlo methods can be used.

1.4 Bibliographical and Historical Remarks

The most influential introductory text book on reinforcement learning is [7]. An excellent summary of the theory is [8].

Chapter 2

Markov Decision Processes and Dynamic Programming

2.1 Introduction

2.2 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

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

(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 $Q_t : \mathcal{A} \to \mathbb{R}$ in each time step; it should be as close to q_* as possible. A reasonable approximation is the expected value 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

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

which serves as a substitute for the ideal choice

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

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

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

(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.

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

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

loop
select an action a ϵ-greedily using 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

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.

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

$$\alpha_t(a) := \frac{1}{n},$$

but they are not satisfied for a constant learning rate $\alpha_t(a) := \alpha$. 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

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

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.

2.3 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 $t \in \{0, 1, 2, ...\}$. The state 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. The action performed by the agent in time step t is denoted by $A_t \in \mathcal{A}(s)$. In general, the set $\mathcal{A}(s)$ of all actions available in state s 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

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

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).

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

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

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

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

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

holds.

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 $r(s,a,s')$.

2.4 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 (expected discounted return)**.** The *expected 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.

2.5 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

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

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

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

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 formulas 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

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

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

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

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

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

and the optimal action-value function is given by

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

These two functions are related by the equation

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

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

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

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

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

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 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 $v_0 : \mathcal{S} \to \mathbb{R}$ (whose terminal state must have the value 0), we use (2.5) to define the iteration

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

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

for $v_{k+1} : \mathcal{S} \to \mathbb{R}$. 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 $v \approx v_\pi$ given $\pi \in \mathcal{P}$.

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

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

return v

2.7 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

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a] \tag{2.10a}$$
$$= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\big(r + \gamma v_\pi(s')\big) \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}(s). \tag{2.10b}$$

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

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

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

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

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

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

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

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 stochastic 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.

2.8 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.

2.9 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

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

initialization:
choose the accuracy threshold $\delta \in \mathbb{R}^+$
initialize the vector 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
        $\Delta := 0$
        for all $s \in \mathcal{S}$ **do**
            $w := v[s]$ ▷ save the old value
            $v[s] := \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, \pi(s))\big(r + \gamma v[s']\big)$
            ▷ note the change: $\pi[s]$ is an optimal action
            $\Delta := \max(\Delta, |v[s] - w|)$
        end for
    until $\Delta < \delta$

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

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

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

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.

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

initialization:
choose the accuracy threshold $\delta \in \mathbb{R}^+$
initialize the vector 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
    $\Delta := 0$
    for all $s \in \mathcal{S}$ **do**
        $w := v[s]$ ▷ save the old value
        $v[s] := \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\big(r + \gamma v[s']\big)$
        ▷ note the change
        $\Delta := \max(\Delta, |v[s] - w|)$
    end for
until $\Delta < \delta$

calculate deterministic policy:
for all $s \in \mathcal{S}$ **do**
    $\pi[s] := \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\big(r + \gamma v[s']\big)$
end for

return $v \approx v_*$ and $\pi \approx \pi_*$

2.10 Bibliographical and Historical Remarks

The most influential introductory text book on reinforcement learning is [7]. An excellent summary of the theory is [8].

Problems

Chapter 3

Monte-Carlo Methods

3.1 Introduction

3.2 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.

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.

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

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.

3.3 On-Policy Monte-Carlo Control

Control means to approximate optimal policies. The idea is the same as in Section 2.8, 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 3.4).

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

cies as shown in Algorithm 6.

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))
      $a_* := \arg\max_{a \in \mathcal{A}(s)} q(S_t, a)$ ▷ 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

3.4 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.

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-sample ratio. Fortunately, the transition probabilities of the MDP cancel.

Now the importance-sample 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.

3.5 Bibliographical and Historical Remarks

Problems

Chapter 4

Temporal-Difference Learning

4.1 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 3). 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.

4.2 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

$$V(S_t) := V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{4.1}$$

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.

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

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && \text{(4.2a)} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{(4.2b)} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. && \text{(4.2c)} \end{aligned}$$

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

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 (4.2c) 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 (4.2a) 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 (4.2c) is replaced by the previous approximation $V(S_{t+1})$ and the expected value in (4.2c) 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.

4.3 On-Policy Temporal-Difference Control: SARSA

If we can approximation 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 (4.1) by the approximation Q of the action-value function q_π to find the update

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)). \tag{4.3}$$

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.

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

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 $\epsilon_t \to 0$.

4.4 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

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

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.

4.5 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 [9]. 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.

Variants of Q-learning are presented in Chapter 5, and convergence results for Q-learning are given in Chapter 12.

4.6 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.

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

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

Definition 4.1 (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 formulas.

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 (4.1).

The corresponding algorithm is shown in Algorithm 10. 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 10 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**
            $G := \sum_{k=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
            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

4.7 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 4.3. 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 4.2 (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

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

for all $0 \leq t < T$. In the case $n = 1$, we recover the one-step update (4.3).

The corresponding algorithm is shown in Algorithm 11.

4.8 Bibliographical and Historical Remarks

Problems

Algorithm 11 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**
            $G := \sum_{k=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
            if $\tau + n < T$ **then**
                $G := G + \gamma^n v(S_{\tau+n})$
            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

Chapter 5

Q-Learning

5.1 Introduction

In this chapter, variants of Q-learning are presented.

5.2 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

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

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 12.

Algorithm 12 double Q-learning for calculating $Q \approx q_*$ and $\pi \approx \pi_*$.

initialization:
choose learning rate $\alpha \in (0, 1]$
choose $\epsilon > 0$
initialize $Q1[s, a] \in \mathbb{R}$ and $Q2[s, a]$ 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
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**
$Q1[s, a] := Q1[s, a] + \alpha\big(r + \gamma \max_{a \in \mathcal{A}(s')} Q2[s', \arg\max_{a \in \mathcal{A}(s')} Q1[s', a)] - Q1[s, a]\big)$
else
$Q2[s, a] := Q2[s, a] + \alpha\big(r + \gamma \max_{a \in \mathcal{A}(s')} Q1[s', \arg\max_{a \in \mathcal{A}(s')} Q2[s', a)] - Q2[s, a]\big)$
end if
$s := s'$
until s is the terminal state and the episode is finished
end loop

5.3 Deep Q-Learning

In deep Q-learning, the optimal action-state function is represented by an artificial neural network. For example, in [2], a deep convolutional neural network was used. Convolutional neural networks are especially well suited for inputs that are images, which is the case in [2]. In [2], 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 7.4 and Remark 12.8).

In [2], 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 [2], two consecutive states S_t and S_{t+1} are highly correlated hence oscillations are likely.

The algorithm in [2] is summarized in Algorithm 13.

Algorithm 13 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

5.4 Bibliographical and Historical Remarks

Problems

Chapter 6

On-Policy Prediction with Approximation

6.1 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.

6.2 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*

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

where μ_π is the discounted state distribution (see Definition 7.1). 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 $\overline{\mathrm{VE}}(w_*) \leq \overline{\mathrm{VE}}(w)$ 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 $\overline{\mathrm{VE}}(w_*) \leq \overline{\mathrm{VE}}(w)$ 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 $\overline{\mathrm{VE}}$ 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

$$w_{t+1} := w_t - \frac{1}{2}\alpha_t \nabla_w (v_\pi(S_t) - \hat{v}_{w_t}(S_t))^2 \tag{6.1a}$$

$$= w_t + \alpha_t (v_\pi(S_t) - \hat{v}_{w_t}(S_t)) \nabla_w \hat{v}_{w_t}(S_t), \tag{6.1b}$$

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

$$w_{t+1} := w_t + \alpha_t(U_t - \hat{v}_{w_t}(S_t))\nabla_w \hat{v}_{w_t}(S_t). \tag{6.2}$$

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.

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

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 (6.1). 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.

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.

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

6.3 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

$$v_\pi(s) \approx \hat{v}_w(s) := w^\top x(s) = \sum_{k=1}^{d} w_i x_i(s). \tag{6.3}$$

The vector valued function

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

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

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 (6.2) 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

$$w_{\text{TD}} := A^{-1}b, \tag{6.4}$$

which is called the TD fixed point.

Theorem 6.1. *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 (6.3) *converges to the* TD *fixed point w_{TD} defined in* (6.4) *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

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

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

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

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

We can write the matrix A as

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

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 6.2. *The* TD *fixed point* w_{TD} *in* (6.4) *satisfies the inequality*

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

6.4 Features for Linear Methods

6.4.1 Polynomials

But consider Weierstrass approximation theorem.

6.4.2 Fourier Basis

ith feature:

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

where c^i is a constant vector.

6.4.3 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).

6.4.4 Tile Coding

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

6.4.5 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.

6.5 Nonlinear Function Approximation

6.5.1 Artificial Neural Networks

See bonus chapter.

6.5.2 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.

6.5.3 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 $k : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a kernel function that assigns a weight to the known data about state s' when asked about state s.

6.6 Bibliographical and Historical Remarks

[10]

Problems

Chapter 7

Policy-Gradient Methods

7.1 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

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

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

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

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.

7.2 Finite and Infinite Action Sets

7.2.1 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.

7.2.2 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

$$\begin{aligned}\mu(s, \theta) &:= \theta_\mu^\top x_\mu(s),\\ \sigma(s, \theta) &:= \exp(\theta_\sigma^\top x_\sigma(s)),\end{aligned}$$

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.

7.3 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 7.1 (discounted state distribution)**.** The *discounted state distribution*

$$\begin{gathered}\mu_\pi\colon \mathcal{S} \to [0, 1],\\ \mu_\pi(s) := \mathbb{E}_{\text{all episodes}}[\lim_{t\to\infty} \mathbb{P}\{S_t = s \mid S_0 \sim \iota,\ A_0, \dots, A_{t-1} \sim \pi\}]\end{gathered}$$

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 7.3 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 7.2 (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

$$L := \sum_{s \in \mathcal{S}} \eta(s) \tag{7.1}$$

of all episodes. □

The following theorem [11] 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 7.3 (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

$$\begin{aligned}\nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(&\nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \\ &+ \pi(a \mid s,\theta)\nabla_\theta \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r \mid s,a)\big(r + \gamma v_{\pi_\theta}(s')\big)\Big) \qquad \forall s \in \mathcal{S},\end{aligned}$$

which simplifies to

$$\begin{aligned}\nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(&\nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \\ &+ \gamma\pi(a \mid s,\theta) \sum_{s'\in\mathcal{S}} p(s',r \mid s,a)\nabla_\theta v_{\pi_\theta}(s')\Big) \qquad \forall s \in \mathcal{S}.\end{aligned}$$

Performing a second time step by applying the recursive formula we just found to $\nabla_\theta v_{\pi_\theta}(s')$, we find

$$\begin{aligned}\nabla_\theta v_{\pi_\theta}(s) = \sum_{a\in\mathcal{A}(s)} \Big(&\nabla_\theta \pi(a \mid s,\theta) q_{\pi_\theta}(s,a) \\ &+ \gamma\pi(a \mid s,\theta) \sum_{s'\in\mathcal{S}} p(s',r \mid s,a) \sum_{a'\in\mathcal{A}(s)} \Big(\nabla_\theta \pi(a' \mid s',\theta) q_{\pi_\theta}(s',a') \\ &\quad + \gamma\pi(a' \mid s',\theta) \sum_{s''\in\mathcal{S}} p(s'',r \mid s',a')\nabla_\theta v_{\pi_\theta}(s'')\Big)\Big) \qquad \forall s \in \mathcal{S}.\end{aligned}$$

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 7.2 and (7.1).

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 7.4 (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 12.8.

7.4 Monte-Carlo Policy-Gradient Method: REINFORCE

Having the gradient of the performance measure available from Theorem 7.3, 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 7.3 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

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\Big[\gamma^t G_t\frac{\nabla_\theta \pi_\theta(A_t \mid S_t,\theta)}{\pi_\theta(A_t \mid S_t,\theta)}\Big].$$

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.

REINFORCE is a stochastic gradient ascent method. By the construction based on Theorem 7.3, 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.

7.5 Monte-Carlo Policy-Gradient Method: REINFORCE with Baseline

The right side in Theorem 7.3 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.,

$$\nabla_\theta J(\theta) = L\sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} q_{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a \mid s,\theta)$$

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 $(S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T)$ following π_θ
  for $t \in (0, 1, \dots, T-1)$ **do** ▷ for all time steps
    $G := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta := \theta + \alpha\gamma^t G \nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta)$
  end for
end loop

return θ

$$= L \sum_{s \in \mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a \in \mathcal{A}(s)} (q_{\pi_\theta}(s,a) - b(s)) \nabla_\theta \pi_\theta(a \mid s, \theta).$$

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

$$\alpha_w := \frac{0.1}{\mathbb{E}[\|\nabla_w \hat{v}_w(S_t)\|_\mu^2]},$$

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 $(S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T)$ following π_θ
  for $t \in (0, 1, \ldots, T-1)$ **do** ▷ for all time steps
    $G := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta := G - \hat{v}_w(S_t)$
    $w := w + \alpha_w \delta \nabla_w \hat{v}_w(S_t)$
    $\theta := \theta + \alpha_\theta \gamma^t \delta \nabla_\theta \ln \pi_\theta(A_t \mid S_t, \theta)$
  end for
end loop

return θ and w

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.

7.6 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.

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

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.

7.7 Bibliographical and Historical Remarks

Problems

Chapter 8

Hamilton-Jacobi-Bellman Equations

8.1 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

$$\dot{s}(t) = f(s(t), \pi(t)) \qquad \forall t \in \mathbb{R}_0^+, \tag{8.1a}$$

$$s(0) = s_0 \in \mathcal{S}, \tag{8.1b}$$

where the function $s : \mathbb{R}_0^+ \to \mathcal{S}$, 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 $u : \mathbb{R}_0^+ \to \mathcal{A}(t)$, 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 $f : \mathbb{R}^{n_s} \times \mathbb{R}^{n_a} \to \mathbb{R}^{n_s}$ describes the dynamics of the system as a deterministic or stochastic ordinary differential equation.

The control $u : \mathbb{R}_0^+ \to \mathcal{A}$ and the corresponding policy $\pi : \mathcal{S} \to \mathcal{A}$ 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 (8.1) 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

$$G : \mathcal{S} \times (\mathbb{R}_0^+ \to \mathcal{A}(t)) \to \mathbb{R},$$
$$G(s_0, u) := \int_0^\tau \mathrm{e}^{-\gamma t} r(s(t), u(t)) \mathrm{d}t + \mathrm{e}^{-\gamma \tau} R(s(\tau))$$

on the state space $\mathcal{S}$ and the set of all actions. The function $r : \mathcal{S} \times \mathcal{A}(t) \to \mathbb{R}$ is called the current reward, and the function $R : \partial\mathcal{S} \to \mathbb{R}$ 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.

8.2 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 8.1 (optimal value function)**.** The *optimal value function* is defined as

$$v_* : \mathcal{S} \to \mathbb{R}, \qquad v_*(s_0) := \sup_{u \in \mathcal{P}} G(s_0, u),$$

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 $u : \mathbb{R}_0^+ \to \mathcal{A}(t)$.

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 [12, Lemma I.7.1]. This is analogous to the Bellman optimality equation (2.7).

Lemma 8.2 (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 8.2 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 (8.1). 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 8.3 (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*

$$\gamma v_*(s_0) = \sup_{a\in\mathcal{A}(0)} (r(s_0,a) + \nabla v_*(s_0)\cdot f(s_0,a)) \qquad \forall s_0 \in \mathcal{S} \tag{8.2}$$

with the boundary condition

$$v_*(s) \geq R(s) \qquad \forall s \in \partial\mathcal{S}. \tag{8.3}$$

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 8.3 is a necessary condition for its solution being the optimal value function. The following theorem states that it is also a sufficient condition [12, Theorem I.7.1].

Theorem 8.4 (sufficient condition)**.** *Suppose that $w \in C^1(\overline{S})$ satisfies* (8.2) *and* (8.3)*. 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 $s \in \overline{S}$.*

Furthermore, suppose that there exists a control u_* such that

$$u_*(t) \in \operatorname*{arg\,max}_{a\in\mathcal{A}(t)} \{r(s_*(t), a) + \nabla w(s_*(t)) \cdot f(s_*(t), a)\} \tag{8.4}$$

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 (8.1) *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 $w \in C^1(\overline{S})$, we can start from the equality

$$\begin{aligned} \mathrm{e}^{-\gamma t} w(s(t)) &= w(s) + \int_0^{t'} \frac{\mathrm{d}}{\mathrm{d}t'} \big(\mathrm{e}^{-\gamma t'} w(s(t'))\big) \mathrm{d}t' \\ &= w(s) + \int_0^{t'} \mathrm{e}^{-\gamma t'} \big(-\gamma w(s(t')) + \dot{s}(t') \cdot \nabla w(s(t'))\big) \mathrm{d}t'. \end{aligned}$$

Using the state equation (8.1), we find

$$\mathrm{e}^{-\gamma t} w(s(t)) = w(s) + \int_0^{t'} \mathrm{e}^{-\gamma t'} \big(-\gamma w(s(t')) + f(s(t'), u(s)) \cdot \nabla w(s(t'))\big) \mathrm{d}t',$$

and using (8.2), 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 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 (8.2) in Theorem 8.3 for the optimal value function v_*. Knowing v_*, we can then use (8.4) in Theorem 8.4 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 $u : \mathbb{R}_0^+ \to \mathcal{A}$ and a policy $\pi : \mathcal{S} \to \mathcal{A}$ are related simply by

$$u(t) = \pi(s(t)).$$

Therefore, we use (8.4) to find an optimal policy by choosing

$$\pi_*(s) \in \arg\max_{a \in \mathcal{A}(t)} \{r(s, a) + \nabla v_*(s) \cdot f(s, a)\}.$$

8.3 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 (8.2). 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 [13]. 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 $s : \mathbb{R}_0^+ \to [0, 1]$ and $u : \mathbb{R}_0^+ \to \{\pm 1\}$. 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 $e^{-\gamma\tau}$. 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

$$v_*(s) := \max(R_0 e^{-\gamma s}, R_1 e^{-\gamma(1-s)}). \tag{8.5}$$

The HJB equation (8.2) 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 (8.2). 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 (8.2) 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 $v_* = R_1 e^{s-1}$. 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.

8.4 Viscosity Solutions

It turns out that the correct solution type for the HJB equation (8.2) 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 (8.4).

The name of viscosity solutions refers to the vanishing-viscosity method used to show their existence [14].

To state some properties of viscosity solutions, we write the first-order equation under consideration as the boundary-value problem

$$H(s, v, \nabla v) = 0 \qquad \forall s \in \overline{\mathcal{S}}, \tag{8.6a}$$

$$v(s) = w(s) \qquad \forall s \in \partial\mathcal{S}, \tag{8.6b}$$

where $\mathcal{S}$ is an open domain, $w : \partial\mathcal{S} \to \mathbb{R}$ is the boundary condition, and the given function H is called the Hamiltonian of the system. The HJB equation

(8.2) corresponds to

$$H(s, v, p) := \gamma v - \sup_{a \in \mathcal{A}(s)} (r(s, a) + p \cdot f(s, a))$$

Definition 8.5 (viscosity solution)**.** Suppose $v : \overline{\mathcal{S}} \to \mathbb{R}$ is continuous and that $v = w$ on $\partial\mathcal{S}$.

The function v is called a *viscosity subsolution* of (8.6) 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 (8.6) 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 (8.6) if it is both a viscosity subsolution and a viscosity supersolution.

Continuing the example in Section 8.3 for the case $R_0 := 1$, $R_1 := 1$, and $\gamma := 1$, it can be checked by elementary calculations that (8.5) 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 8.6. 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 8.6. *Suppose $v \in C^1(\mathcal{S})$ be a classical solution of* (8.6)*. Then it is also a viscosity solution.*

Lemma 8.7. *Suppose $v \in C(\overline{\mathcal{S}})$ is a viscosity solution of* (8.6) *and suppose v is differentiable at $s_0 \in \mathcal{S}$. Then $H(s_0, v(s_0), \nabla v(s_0)) = 0$.*

Proof. [15, Section 10.1.2]. □

Theorem 8.8. *Under certain assumptions on the Hamiltonian H, the boundary-value problem* (8.6) *has at most one bounded viscosity solution.*

Proof. [14, Theorem III.1]. □

So far, we have seen that the solution of the boundary-value problem (8.6) is unique among the viscosity solutions under some assumptions. However, the boundary condition in Theorem 8.3 is the inequality (8.3) and not an equality such as the boundary condition in (8.6). 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 8.9 (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 8.10. 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 8.11. *Suppose that Assumption 8.10 holds. Then the equation* (8.2) *with the boundary condition* (8.3) *has a unique viscosity solution with an inequality boundary condition.*

Proof. [13, Theorem 4], [12, Section II.11 and II.13]. □

In summary, viscosity solutions are the right class of solutions for the HJB equation (8.2) in the sense that there always exists a unique solution for the purposes of Theorem 8.3 and solving the optimal-control problem.

8.5 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

$$\mathrm{d}s = f(s(t)), u(t))\mathrm{d}t + \sigma(s(t), u(t))\mathrm{d}\omega \qquad \forall t \in \mathbb{R}_0^+, \tag{8.7a}$$

$$s(0) = s_0 \in \mathcal{S} \tag{8.7b}$$

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 8.12 (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 8.13 (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

$$G : \mathcal{S} \times (\mathbb{R}_0^+ \to \mathcal{A}(t)) \to \mathbb{R},$$
$$G(s_0, u) := \mathbb{E}\left[\int_0^\tau \mathrm{e}^{-\gamma t} r(s(t), u(t))\mathrm{d}t + \mathrm{e}^{-\gamma\tau} R(s(\tau))\right].$$

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

$$\gamma v_*(s) = \sup_{a \in \mathcal{A}(0)} \left(r(s, a) + \nabla v_*(s) \cdot f(s, a) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}(s, a) \frac{\partial v_*}{\partial s_i \partial s_j}(s) \right) \qquad \forall s \in \mathcal{S}, \tag{8.8a}$$

$$v_*(s) = R(s) \qquad \forall s \in \partial\mathcal{S}, \tag{8.8b}$$

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 (8.8) has a unique classical solution. Otherwise, the concept of viscosity solutions extended to second-order equations [16] can be used.

8.6 Bibliographical and Historical Remarks

Problems

Chapter 9

Deep Reinforcement Learning

Advanced algorithms 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.

9.1 Introduction

Series of papers coming out of Google DeepMind: [2], [17], [4], [5], [3], [18].

9.2 Atari 2600 Games

In [2], 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] representing a diverse collection of tasks.

The algorithm is shown in Algorithm 19.

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

$$(s_t, a_t, r_{t+1}, s_{t+1})$$

[1] DQN plays Breakout: `https://www.youtube.com/watch?v=TmPfTpjtdgg`. DQN plays Space Invaders: `https://www.youtube.com/watch?v=W2CAghUiofY`.

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

in each time step t are stored in datasets $D_t := \{e_1, \ldots, e_t\}$, and the expected value is approximated by drawing samples, or minibatches in the language of neural networks, (s, a, r, s') uniformly from the dataset 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

$$\nabla_{\mathbf{w}_i} L_i(\mathbf{w}_i) = -2\mathbb{E}_{(s,a,r,s')\sim U(D_i)}\big[\big(r+\gamma \max_{a'} \hat{q}(s', a', \mathbf{w}_i^-) - \hat{q}(s, a, \mathbf{w}_i)\big)\nabla_{\mathbf{w}_i} \hat{q}(s, a, \mathbf{w}_i)\big].$$

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.

9.3 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 [17], 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., [7, 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 [17] 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 dataset 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 optimized 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 minimized 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.

9.4 Learning Go Tabula Rasa (AlphaGo Zero)

In the next stage in the development of the Alpha programs was to render the existence of a preexisting dataset of expert moves superfluous. This was achieved in [4], 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 [17].

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 dataset 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.

9.5 Chess, Shogi, and Go through Self-Play (AlphaZero)

In [5], 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 defeated a world champion in chess, shogi (Japanese chess, with a more complex game tree than chess), and Go.

9.6 Video Games of the 2010s (AlphaStar)

In [3] and [18], attention returned to video games with partial observability after the board games with no hidden information.

In [3], learning 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 evaluated 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 teammates and against opponents sampled from the whole population.

In [18], 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.

9.7 Improvements to DQN and their Combination

In [19], six independent extensions to the DQN algorithm [2] 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).

9.7.1 Double Q-Learning

The first extension is double Q-learning already presented as Algorithm 12 in Chapter 5.

9.7.2 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.

9.7.3 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.

9.7.4 Multi-Step Methods

Multi-step methods are discussed in Chapter 4 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 $G_{t:t+n}$.

9.7.5 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

$$d'_t := \left(R_{t+1} + \gamma_{t+1}\mathbf{z}, \mathbf{p}_{\theta'}(s, \arg\max_{a\in\mathcal{A}} \hat{q}(S_{t+1}, a, \theta')\right)$$

and d_t must be minimized, for example by minimizing the Kullbeck-Leibler divergence

$$D_{\mathrm{KL}}\left(\Phi_{\mathbf{z}}(d'_t) \,\|\, d_t\right).$$

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 $\mathbf{z}$.

9.7.6 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.

Part II

Convergence

Chapter 11

Monte-Carlo Theory

11.1 Introduction

In Chapter 3, first-visit and every-visit Monte-Carlo prediction methods are presented. 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. These facts are the subjects of the following sections.

11.2 Convergence of First-Visit Monte-Carlo Prediction

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

Theorem 11.1 (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,*

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

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 A.44, 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 A together with the law of large numbers.

11.3 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 11.2 (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,*

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

The proof follows [21, 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 A.43, 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 A.45, 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. □

11.4 Bibliographical and Historical Remarks

Further investigations into the convergence properties of first- and every-visit Monte Carlo can be found in [22].

Chapter 12

Convergence of Q-Learning

12.1 Introduction

Q-learning is an off-policy temporal-difference control method that directly approximates the optimal action-value function $q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote the approximation in time step t of an episode by $Q_t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. 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

$$Q_{t+1}(s,a) := \begin{cases} (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a)), & (s,a) = (s_t,a_t), \\ Q_t(s,a), & (s,a) \neq (s_t,a_t). \end{cases} \tag{12.1}$$

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.

12.2 Convergence of the Discrete Method Proved Using Action Replay

Q-learning was introduced in [9, page 95], where an outline [9, 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 [23]. 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 12.1 (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*

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

then

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

(as defined by (12.1)*) 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 12.2. *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 (12.1). Therefore we have

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

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 [23].

Lemma 12.3. *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 12.4. *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 12.5. *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 [23, Lemma B.3] relies on a standard theorem in stochastic convergence (see, e.g., [24, 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 12.6. *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 12.2, 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 12.5, 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 12.4, 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 12.6, 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 12.4 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 12.6 stemming from imprecise transition probabilities and expected rewards.

By Lemma 12.3, 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 12.7 (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 [23, 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 12.3 holds.

Remark 12.8 (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 [2]. Cf. Remark 7.4.

12.3 Convergence of the Discrete Method Proved Using Fixed Points

In [25], the convergence of Q-learning and TD(λ) was shown by viewing the algorithms as certain stochastic processes and applying techniques of stochastic approximation and a fixed-point theorem. The same line of reasoning can be found for Q-learning in [26] and [21, Section 5.6].

The first theorem below is the convergence result for the stochastic process. The following two theorems are convergence results for Q-learning and TD(λ) that use the first theorem.

Theorem 12.9 (convergence of a stochastic process [25])**.** *The stochastic process*

$$\Delta_{t+1}(s) := (1 - \alpha_t(s))\Delta_t(s) + \beta_t(s)F_t(s) \tag{12.2}$$

converges to zero almost surely if the following assumptions hold.

1. *The state space is finite.*

2. *The equalities and inequalities*

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

   are satisfied uniformly and almost surely.

3. *The inequality*

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

holds.

4. *The inequality*

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

holds.

Here $P_t := \{\Delta_t, \Delta_{t-1}, \ldots, F_{t-1}, F_{t-2}, \ldots, \alpha_{t-1}, \alpha_{t-2}, \ldots, \beta_{t-1}, \beta_{t-2}, \ldots\}$ *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 three lemmata.

Lemma 12.10. *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.*

Classic (Dvoretzky 1956).

Lemma 12.11. *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 12.12. *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.

Based on these three lemmata, we continue with the proof of the theorem. The stochastic process $\Delta_t(s)$ can be decomposed into two processes

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

by defining

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

Applying the lemmata to these two processes finishes the proof.

□

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

Theorem 12.13 (convergence of Q-learning)**.** *The Q-learning iterates*

$$Q_{t+1}(s,a) := (1 - \alpha_t(s,a))Q_t(s_t, a_t) + \alpha_t(s,a)(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)), \quad (12.3)$$

where $\alpha_t : \mathcal{S} \times \mathcal{A} \to [0,1)$, 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 12.14. The difference between the two Q-learning iterations (12.1) and (12.3) is in the step sizes α_t. In the first form (12.1), 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 (12.3), 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 12.1).

Proof. The basic idea in applying Theorem 12.9, 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 $K : (\mathcal{S} \times \mathcal{A} \to \mathbb{R}) \to \mathbb{R}$ as

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

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

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

In summary,

$$\|Kq_1 - Kq_2\|_\infty \le \gamma \|q_1 - q_2\|_\infty. \tag{12.4}$$

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

$$Kq^* = q^*. \tag{12.5}$$

In order to relate the iteration (12.3) to the stochastic process in Theorem 12.9, we define

$$\begin{aligned}
\beta_t &:= \alpha_t, \\
\Delta_t(s, a) &:= Q_t(s, a) - q^*(s, a),
\end{aligned}$$

$$F_t(s,a) := r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - q^*(s,a).$$

With these definitions, the Q-learning iteration (12.3) and the stochastic process (12.2) are identical.

The expected value of $F_t(s,a)$ given the past P_t as it appears in Theorem 12.9 is

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

where the last equation follows from (12.5).

Using (12.4), 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 12.9 is satisfied if $\gamma < 1$.

In order to check the fourth assumption Theorem 12.9, 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 12.9 are satisfied. □

The second application of Theorem 12.9 is the convergence of TD(λ).

Theorem 12.15 (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^**.*

By extending the approach in [27, 28], convergence theorems for Q-learning for environments that change over time, but whose accumulated changes remain bounded, were shown in [29].

12.4 Bibliographical and Historical Remarks

Q-learning was introduced in [9, 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 [9, Appendix 1]. An extended, more detailed version of this proof was given in [23].

Problems

Appendix A

Measure and Probability Theory

A.1 Notation

Definition A.1 (range)**.** The set $\{m, \dots, n\}$ of all integers between m and n is denoted by [m:n].

Definition A.2 (Iverson bracket)**.** The *Iverson bracket* of a statement is defined as

$$[\![\text{statement}]\!] := \begin{cases} 1 & \text{if statement is true,} \\ 0 & \text{if statement is false.} \end{cases}$$

A.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 A.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 A.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 $\Omega \backslash A \in \mathcal{F}$.

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 $\{\emptyset, A, \Omega \backslash A, \Omega\}$.

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 A.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 A.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 A.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

$$\mathcal{M}_0 := \{A \subset \mathbb{R} : A \in \mathcal{O}\}, \tag{A.1a}$$
$$\mathcal{M}_1 := \{[a, b] \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a \leq b\}, \tag{A.1b}$$
$$\mathcal{M}_2 := \{[a, b] \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a \leq b\}, \tag{A.1c}$$
$$\mathcal{M}_3 := \{(a, b) \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a < b\}, \tag{A.1d}$$
$$\mathcal{M}_4 := \{(a, b) \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a < b\}, \tag{A.1e}$$
$$\mathcal{M}_5 := \{(a, b] \subset \mathbb{R} : a, b \in \mathbb{R} \wedge a \leq b\}, \tag{A.1f}$$
$$\mathcal{M}_6 := \{(a, b] \subset \mathbb{R} : a, b \in \mathbb{Q} \wedge a \leq b\}, \tag{A.1g}$$
$$\mathcal{M}_7 := \{(-\infty, a] \subset \mathbb{R} : a \in \mathbb{R}\}, \tag{A.1h}$$
$$\mathcal{M}_8 := \{(-\infty, a] \subset \mathbb{R} : a \in \mathbb{Q}\}, \tag{A.1i}$$
$$\mathcal{M}_9 := \{(-\infty, a) \subset \mathbb{R} : a \in \mathbb{R}\}, \tag{A.1j}$$
$$\mathcal{M}_{10} := \{(-\infty, a) \subset \mathbb{R} : a \in \mathbb{Q}\}. \tag{A.1k}$$

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 A.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 A.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 $X : \Omega \to \Psi$ such that $X^{-1}(G) \in \mathcal{F}$ for every $G \in \mathcal{G}$.

Definition A.10 (measure)**.** Suppose $(\Omega, \mathcal{F})$ is a measurable space. A function $\mu : \mathcal{F} \to [0, \infty]$ 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

$$\mu\Big(\bigcup_{i=1}^{\infty} A_i\Big) = \sum_{i=1}^{\infty} \mu(A_i).$$

Definition A.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 $(\Omega, \mathcal{F})$.

A.3 Lebesgue Integrals

A.4 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 A.34).

Lemma A.12 (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 $f_n : X \to [0, \infty]$. Suppose further that $f : X \to [0, \infty]$ 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 A.12.

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 A.13 (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 $f_n : X \to [-\infty, \infty]$. Suppose further that $g : X \to [0, \infty]$ is a non-negative, $(\mathcal{F}, \mathcal{B}(\mathbb{R}_0^+))$-measurable, and integrable function on X such that*

$$\forall_\mu x \in X : \quad \forall n \in \mathbb{N} : \quad f_n(x) \leq g(x).$$

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 A.12, 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 A.14 (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 $f_n : X \to [-\infty, \infty]$ that is μ-almost everywhere dominated by an integrable function $g : X \to [0, \infty]$, i.e.,*

$$\forall_\mu x \in X : \quad \forall n \in \mathbb{N} : \quad |f_n(x)| \leq g(x).$$

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 A.12, can be applied to the functions $f_n + g$, which yields the first inequality. The third inequality is reverse Fatou's lemma, Lemma A.13. □

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 A.15 (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 $f_n : X \to [0, \infty]$, i.e.,*

$$\forall_\mu x \in X : \quad \forall n \in \mathbb{N} : \quad 0 \leq f_n(x) \leq f_{n+1}(x) \leq \infty.$$

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 A.16 (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 $f_n : X \to [-\infty, \infty]$ that is μ-almost everywhere dominated by an integrable function $g : X \to [0, \infty]$, i.e.,*

$$\forall_\mu x \in X : \quad \forall n \in \mathbb{N} : \quad |f_n(x)| \leq g(x).$$

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

$$\int_X f \mathrm{d}\mu = \int_X \lim_{n\to\infty} f_n \mathrm{d}\mu = \lim_{n\to\infty} \int_X f_n \mathrm{d}\mu.$$

A.5 Probability Spaces

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 $\mathbb{P} : \mathcal{F} \to [0, 1]$ 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 A.17 (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 A.18 (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 $\mathbb{P} : \mathcal{F} \to [0, 1]$ is a probability measure.

If $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space, then $(\Omega, \mathcal{F})$ 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 A.19 (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* is a measurable function $X : \Omega \to \Psi$.

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 (A.1h) of the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ (see Section A.2). Since it suffices to check measurability on any generator of the Borel σ-algebra and the preimages in Definition A.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 $a \in \mathbb{R}$.

Finally, we define integrable random variables.

Definition A.20 (integrable random variable)**.** A random variable X is called *integrable* if the expected value $\mathbb{E}[|X|]$ is integrable as a Lebesgue integral (see Definition **??**).

The definition of an integrable random variable implies that $\mathbb{E}[X]$ exists and has a finite value as well.

A.6 Probability Distribution Functions

A.7 Inequalities

In this section, important inequalities connected to measure and probability theory are collected.

A.7.1 Basic Inequalities

Jensen's inequality is an important inequality, whose measure-theoretic form is stated in the following theorem.

Definition A.21 (convex set)**.** A set C is called *convex* if

$$\forall (x, y) \in C^2 : \quad \forall y \in C : \quad \forall \alpha \in [0, 1] : \quad \alpha x + (1 - \alpha) y \in C.$$

Definition A.22 ((strictly) convex and concave functions)**.** Suppose C is a convex set. A function $f : C \to \mathbb{R}$ is called *convex* if

$$\forall (x, y) \in C^2 : \quad \forall \alpha \in [0, 1] : \quad f(\alpha x + (1 - \alpha) y) \leq \alpha f(x) + (1 - \alpha) f(y).$$

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 A.23 (subderivative)**.** A *subderivative* of a convex function $f : I \to \mathbb{R}$ at a point $x_0 \in I$ in the open interval I is a real number $c \in \mathbb{R}$ such that

$$\forall x \in I : \quad f(x) - f(x_0) \geq c(x - x_0).$$

Lemma A.24 (subderivatives). *Suppose $f : I \to \mathbb{R}$ 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 A.25 (Jensen's inequality). *Suppose that $(\Omega, \mathcal{F}, \mu)$ is a measure space, that $I \subset \mathbb{R}$ is an interval, that the function $g : \Omega \to I$ is μ-integrable, and that the function $\phi : I \to \mathbb{R}$ is convex on the interval I. Then the inequality*

$$\phi\Big(\frac{1}{\mu(\Omega)}\int_\Omega g\mathrm{d}\mu\Big) \le \frac{1}{\mu(\Omega)}\int_\Omega \phi\circ g\mathrm{d}\mu$$

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

$$\phi(\mathbb{E}[X]) \le \mathbb{E}[\phi(X)].$$

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 A.24, there exists a $c \in \mathbb{R}$ for the point $x_0 \in I$ such that

$$\forall x \in I : \quad \phi(x) - \phi(x_0) \ge c(x - x_0).$$

Since this inequality holds for all $x \in I$, it also holds for all $x = g(\omega) \in I$, i.e.,

$$\forall \omega \in \Omega : \quad \phi(g(\omega)) \ge \phi(x_0) + c(g(\omega) - x_0).$$

Using the monotony of the integral, integration of both sides yields

$$\int_\Omega \phi\circ g\mathrm{d}\mu \ge \mu(\Omega)\phi(x_0) + c\Big(\int_\Omega g\mathrm{d}\mu - \mu(\Omega)x_0\Big) = \mu(\Omega)\phi(x_0).$$

If the function ϕ is concave, the analogous argument shows the reversed inequality, which concludes the proof. □

A.7.2 Concentration Inequalities

Concentration inequalities provide probability bounds on how much a random variable deviates from its expected value.

Markov's and Chebyshev's Inequalities

Theorem A.26 (Markov's inequality)**.** *Suppose X is a non-negative random variable and $a \in \mathbb{R}^+$. Then the inequality*

$$\mathbb{P}[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$$

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. □

Theorem A.27 (Chebyshev's 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 [30], 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 A.28 (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

$$\forall \lambda \in \mathbb{R} : \quad \mathbb{E}\left[\mathrm{e}^{\lambda(X - \mathbb{E}[X])}\right] \leq \mathrm{e}^{\lambda^2 (b-a)^2/8}$$

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

$$\begin{aligned}\mathbb{E}\left[\mathrm{e}^{\lambda(X-\mathbb{E}[X])}\right] &\leq \frac{b-\mathbb{E}[X]}{b-a}\mathrm{e}^{\lambda(a-\mathbb{E}[X])} + \frac{\mathbb{E}[X]-a}{b-a}\mathrm{e}^{\lambda(b-\mathbb{E}[X])} \\ &= (1-\alpha)\mathrm{e}^{\lambda(a-\mathbb{E}[X])} + \alpha\mathrm{e}^{\lambda(b-\mathbb{E}[X])},\end{aligned}$$

where

$$\alpha := \frac{\mathbb{E}[X]-a}{b-a}.$$

Using $y := \lambda(b-a)$, we have

$$\begin{aligned}\mathbb{E}\left[\mathrm{e}^{\lambda(X-\mathbb{E}[X])}\right] &\leq (1-\alpha+\alpha\mathrm{e}^{\lambda(b-a)})\mathrm{e}^{\lambda(a-\mathbb{E}[X])} \\ &= (1-\alpha+\alpha\mathrm{e}^{\lambda(b-a)})\mathrm{e}^{-\alpha\lambda(b-a)} \\ &= (1-\alpha+\alpha\mathrm{e}^{y})\mathrm{e}^{-\alpha y}.\end{aligned}$$

Defining the function

$$f: \quad \mathbb{R} \to \mathbb{R}, \quad y \mapsto \ln(1-\alpha+\alpha\mathrm{e}^{y}) - \alpha y,$$

the inequality becomes

$$\mathbb{E}\left[\mathrm{e}^{\lambda(X-\mathbb{E}[X])}\right] \leq \mathrm{e}^{f(y)}.$$

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

$$\forall y \in \mathbb{R}: \quad \exists z \in [0,y]: \quad f(y) = f(0) + f'(0)y + \frac{f''(z)}{2}y^2.$$

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 A.29 (Hoeffding's inequality)**.** *Suppose that $\{X_i\}_{i=1}^n$ are independent real-valued random variables, each being bounded such that*

$$\forall i \in [1:n]: \quad \exists (a_i, b_i) \in \mathbb{R}^2: \quad \mathbb{P}[a_i \leq X_i \leq b_i] = 1.$$

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 [30]. Better bounds for this case can be found in [31].

Proof. We first define

$$S := \sum_{i=1}^n X_i.$$

Using an arbitrary parameter $\lambda \in \mathbb{R}^+$ and Markov's inequality, Theorem A.26, we find

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] = \mathbb{P}[S - \mathbb{E}[S] \geq nt] = \mathbb{P}[\mathrm{e}^{\lambda(S-\mathbb{E}[S])} \geq \mathrm{e}^{\lambda nt}] \leq \frac{\mathbb{E}\left[\mathrm{e}^{\lambda(S-\mathbb{E}[S])}\right]}{\mathrm{e}^{\lambda nt}}.$$

Since the random variables X_i are independent, the inequality

$$\mathbb{P}[\overline{X} - \mathbb{E}[\overline{X}] \geq t] \leq \mathrm{e}^{-\lambda nt} \prod_{i=1}^n \mathbb{E}\left[\mathrm{e}^{\lambda(X_i - \mathbb{E}[X_i])}\right] \tag{A.2}$$

holds. Using Hoeffding's lemma, Lemma A.28, 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 A.30 (Hoeffding's inequality for sums)**.** *Suppose the assumptions of Theorem A.29 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 A.29 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 A.31 (Hoeffding's inequality for confidence intervals)**.** *Suppose the assumptions of Theorem A.29 hold. Then the inequalities*

$$\mathbb{P}\left[\mathbb{E}[\overline{X}] > \overline{X} - \sqrt{\frac{-\ln(\delta)\sum_{i=1}^n (b_i - a_i)^2}{2n^2}}\right] \geq 1 - \delta, \tag{A.3a}$$

$$\mathbb{P}\left[|\mathbb{E}[\overline{X}] - \overline{X}| < \sqrt{\frac{-\ln(\delta)\sum_{i=1}^n (b_i - a_i)^2}{2n^2}}\right] \geq 1 - 2\delta \tag{A.3b}$$

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

$$\delta: \quad \mathbb{R}^+ \to (0,1), \quad \delta(t) := \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i-a_i)^2}\right) \tag{A.4}$$

and its inverse

$$t: \quad (0,1) \to \mathbb{R}^+, \quad t(\delta) := \sqrt{\frac{-\ln(\delta)\sum_{i=1}^{n}(b_i-a_i)^2}{2n^2}} \tag{A.5}$$

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 A.29, 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 A.29 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 A.29, we assume that the random variables are independent or that sampling is performed without replacement (by the comment below the theorem). Then (A.3a) 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, (A.3b) 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 A.31 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 (A.4) and (A.5) 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

$$m: \quad (0,1) \times \mathbb{R}^+ \to \mathbb{R}^+, \quad m(\delta, t) := \frac{-\ln \delta}{t^2}.$$

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 A.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 A.32 (Bernstein's inequality)**.** *Suppose $\{X_i\}_{i=1}^n$ are independent random variables such that*

$$\forall i \in [1\!:\!n]: \quad \mathbb{E}[X_i] = 0$$

and

$$\exists M \in \mathbb{R}^+: \quad \forall i \in [1\!:\!n]: \quad \mathbb{P}[|X_i| \leq M] = 1.$$

Then the inequality

$$\mathbb{P}[\overline{X} \geq t] \leq \exp\left(-\frac{nt^2}{2(Mt/3 + n\sigma^2)}\right),$$
$$\overline{X} := \frac{1}{n}\sum_{i=1}^{n} X_i,$$
$$\sigma^2 := \mathbb{V}[\overline{X}] = \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{V}[X_i] = \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{E}[X_i^2],$$

holds for all $t \in \mathbb{R}^+$.

Proof. Since the assumptions of Hoeffding's inequality, Theorem A.29, are satisfied, we will be able to use (A.2), which contains the terms $\mathbb{E}[e^{\lambda X_i}]$, to show the proposed inequality.

Before we do so, we estimate the terms $\mathbb{E}[e^{\lambda X_i}]$. 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

$$\forall \lambda \in \mathbb{R}: \quad \mathbb{E}[e^{\lambda X_i}] = 1 + \sum_{k=2}^{\infty} \frac{\lambda^k \mathbb{E}[X_i^k]}{k!}$$

for all random variables indexed by $i \in [1:n]$. We define the infinite sum

$$f: \quad \mathbb{R} \to \mathbb{R}, \quad \lambda \mapsto \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\mathbb{E}[X_i^k]}{\sigma_i^2 k!}, \quad \sigma_i^2 := \mathbb{V}[X_i] = \mathbb{E}[X_i^2]$$

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

$$\forall \lambda \in \mathbb{R}: \quad \mathbb{E}[e^{\lambda X_i}] = 1 + \lambda^2 \sigma_i^2 f(\lambda).$$

We now use the inequality $1 + x \leq e^x$ 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

$$\forall \lambda \in \mathbb{R}: \quad \mathbb{E}[e^{\lambda X_i}] \leq e^{\lambda^2 \sigma_i^2 f(\lambda)}. \tag{A.6}$$

Next, we consider the terms $\mathbb{E}[X_i^k]$ in the infinite sum f. The Cauchy-Schwarz inequality yields

$$\mathbb{E}[X_i^k] = \int x_i x_i^{k-1} \mathrm{d}\mathbb{P}(X_i) \leq \left(\int |x_i|^2 \mathrm{d}P(X_i)\right)^{1/2} \left(\int |x_i^{k-1}|^2 \mathrm{d}P(X_i)\right)^{1/2}$$

$$= \sigma_i \left(\int |x_i|^{2k-2} \mathrm{d}P(X_i) \right)^{1/2}.$$

We continue to apply the Cauchy-Schwarz inequality to the last factor recursively. Each application of the Cauchy-Schwarz inequality has the form

$$\begin{aligned} \int x_i^\alpha \mathrm{d}\mathbb{P}(X_i) = \int x_i x_i^{\alpha-1} \mathrm{d}\mathbb{P}(X_i) &\leq \left(\int |x_i|^2 \mathrm{d}P(X_i) \right)^{1/2} \left(\int |x_i^{\alpha-1}|^2 \mathrm{d}P(X_i) \right)^{1/2} \\ &= \sigma_i \left(\int |x_i|^{2(\alpha-1)} \mathrm{d}P(X_i) \right)^{1/2}. \end{aligned}$$

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

$$\begin{aligned} \forall m \in \mathbb{N}: \quad \mathbb{E}[X_i^k] &\leq \sigma_i^{1+1/2+\cdots+(1/2)^{m-1}} \left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}P(X_i) \right)^{1/2^m} \\ &= \sigma_i^{2(1-1/2^m)} \left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}P(X_i) \right)^{1/2^m}. \end{aligned}$$

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

$$\left(\int |x_i|^{2^m k - 2^{m+1}+2} \mathrm{d}P(X_i) \right)^{1/2^m} \leq (M^{2^m k - 2^{m+1}+2})^{1/2^m},$$

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

$$\begin{aligned} \forall \lambda \in \mathbb{R}_0^+: \quad f(\lambda) = \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\mathbb{E}[X_i^k]}{\sigma_i^2 k!} &\leq \sum_{k=2}^{\infty} \frac{\lambda^{k-2}\sigma_i^2 M^{k-2}}{\sigma_i^2 k!} = \frac{1}{\lambda^2 M^2} \sum_{k=2}^{\infty} \frac{\lambda^k M^k}{k!} \\ &= \frac{1}{\lambda^2 M^2} (\mathrm{e}^{\lambda M} - 1 - \lambda M) \end{aligned}$$

if the parameter λ is non-negative. Applying this estimate to (A.6) yields

$$\forall \lambda \in \mathbb{R}_0^+ : \quad \mathbb{E}[\mathrm{e}^{\lambda X_i}] \leq \exp\left(\lambda^2 \sigma_i^2 \frac{1}{\lambda^2 M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right).$$

Next, we use inequality (A.2) 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

$$\begin{aligned}\forall \lambda \in \mathbb{R}_0^+ : \quad \forall t \in \mathbb{R}^+ : \quad \mathbb{P}[\overline{X} \geq t] &\leq \mathrm{e}^{-\lambda n t} \prod_{i=1}^n \exp\left(\lambda^2 \sigma_i^2 \frac{1}{\lambda^2 M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right) \\ &= \exp\left(-\lambda n t + \frac{n^2\sigma^2}{M^2}(\mathrm{e}^{\lambda M} - 1 - \lambda M)\right).\end{aligned}$$

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

$$g : \quad \mathbb{R}^+ \to \mathbb{R}, \quad g(x) := (1+x)\ln(1+x) - x,$$

we have

$$\forall t \in \mathbb{R}^+ : \quad \mathbb{P}[\overline{X} \geq t] \leq \exp\left(-\frac{n^2\sigma^2}{M^2} g\left(\frac{tM}{n\sigma^2}\right)\right), \tag{A.7}$$

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 (A.7) 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 (A.7) 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 (A.7). 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 (A.7). Therefore is better suited to obtain numerical bounds of the mean value $\overline{X}$.

Empirical Bernstein's Inequality

Anderson Inequality

A.8 Characteristic Functions

Definition A.33 (characteristic function)**.** The *characteristic function* ϕ_X of a random variable X is the expected value of e^{itX}, i.e.,

$$\phi_X : \quad \mathbb{R} \to \mathbb{C}, \quad \phi_X(t) := \mathbb{E}[e^{itX}] = \int_{\mathbb{R}} e^{itx} dF_X(x) = \int_{\mathbb{R}} e^{itx} f_X(x) dx,$$

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

$$\phi_X^{(k)}(0) = \mathrm{i}^k \mathbb{E}[X^k] \tag{A.8}$$

is useful since it relates the derivatives of the characteristic function at zero to the moments.

A.9 Types of Convergence

In order to define almost sure convergence, the limit supremum of a sequence of sets is needed.

Definition A.34 (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

$$\begin{aligned}\liminf_{n\to\infty} &:= \lim_{n\to\infty} \inf_{i\geq n} x_i,\\ \limsup_{n\to\infty} &:= \lim_{n\to\infty} \sup_{i\geq n} x_i.\end{aligned}$$

Lemma A.35. *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 A.36 (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 A.2) that indicates whether x is an element of A_n.

Lemma A.37. *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*

$$\liminf_{n\to\infty} A_n = \{x \in \Omega : \liminf_{n\to\infty} [\![x \in A_n]\!] = 1\},$$
$$\limsup_{n\to\infty} A_n = \{x \in \Omega : \limsup_{n\to\infty} [\![x \in A_n]\!] = 1\}.$$

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 A.38 (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 A.39 (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 A.40 (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

$$X_n \xrightarrow[n\to\infty]{\mathrm{d}} X.$$

Convergence in probability implies convergence in distribution.

A.10 Lévy's Continuity Theorem

Theorem A.41 (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 [32, Section 18.1] and in [33, Theorems 14.15 and 18.21].

A.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 A.42 (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 A.27, 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 A.27, 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 (A.8); 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

$$\forall t \in \mathbb{R} : \quad \lim_{n\to\infty} \phi_{\overline{X}_n}(t) = \mathrm{e}^{\mathrm{i}\mu t}$$

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 A.41, 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 A.43 (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 A.44 (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

$$\forall x \in \mathbb{R} : \quad \lim_{n \to \infty} \mathbb{P}[\sqrt{n}(\overline{X}_n - \mu) \leq x] = \Phi\left(\frac{x}{\sigma}\right),$$

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 (A.8); 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 A.41, 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. □

A.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 A.45 (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 $N(\Omega) \subset \mathbb{N}\backslash\{0\}$,*

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 A.15, 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

$$\forall i \in \mathbb{N}: \quad |S_i| \leq \sum_{n=1}^{i} |X_n|$$

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 A.15, 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

$$\forall i \in \mathbb{N}: \quad |T_i| \leq \sum_{n=1}^{i} |\mathbb{E}[X_n]|,$$

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 A.25, 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 A.16, 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 A.16, 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 A.16, 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 $\mathbb{N}\backslash\{0\}$. 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 $\mathbb{E}[\sum_{n=1}^{N} 1] = \mathbb{E}[N]$.

A.13 Bibliographical and Historical Remarks

Bibliography

[1] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. arXiv:1711.09602, 11 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[3] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364:859–865, 2019.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362:1140–1144, 2018.

[6] 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*, pages 1–16, 2017.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, 2nd edition, 2018.

[8] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[9] Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[10] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223311, 2018.

[11] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S.A. Solla, T.K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1057–1063. MIT Press, 2000.

[12] Wendell H. Fleming and Halil Mete Soner. *Controlled Markov Processes and Viscosity Solutions*. Springer, 2nd edition, 2006.

[13] Rémi Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40(3):265299, 2000.

[14] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi equations. *Trans. Amer. Math. Soc.*, 277(1):1–42, 1983.

[15] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 2nd edition, 2010.

[16] Michael G. Crandall, Hitoshi Ishii, and Pierre-Louis Lions. User's guide to viscosity solutions of second order partial differential equations. *Bull. Amer. Math. Soc.*, 27(1):167, 1992.

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[18] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P.

Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

[19] 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 Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-2018)*, pages 3215–3222. AAAI Press, 2018.

[20] Mark Rowland, Marc G. Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*, pages 29–37, 2018.

[21] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[22] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158, 1996.

[23] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[24] Harold J. Kushner and Dean S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag New York Inc., 1978.

[25] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):11851201, 1994.

[26] John N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.

[27] Michael L. Littman and Csaba Szepesvári. A generalized reinforcement-learning model: convergence and applications. In Lorenzo Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, pages 310–318. Morgan Kaufmann, 1996.

[28] Csaba Szepesvári and Michael L. Littman. Generalized Markov decision processes: dynamic-programming and reinforcement-learning algo-

rithms. Technical Report Technical Report CS-96-11, Department of Computer Science, Brown University, Providence, Rhode Island, USA, November 1996.

[29] István Szita, Bálint Takács, and András Lőrincz. ϵ-MDPs: learning in varying environments. *Journal of Machine Learning Research*, 3:145–174, 2002.

[30] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[31] Robert J. Serfling. Probability inequalities for the sum in sampling without replacement. *Annals of Statistics*, 2(1):39–48, 1974.

[32] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[33] B.E. Fristedt and L.F. Gray. *A Modern Approach to Probability Theory*. Birkhäuser, 1996.

Index