
The Mathematics of Reinforcement Learning

LEIF, LEO, SARA, ALMUT, SIMON, ANDRÉ

UNIVERSITY OF MANNHEIM

Contents

I	Multiarmed Bandits	2
1	Stochastic bandits	3
1.1	A quick dive into two-stage stochastic experiments	3
1.2	Model, examples, and the goals	3
1.3	Algorithms: the exploration-exploitation trade-off	11
1.3.1	Basic committ-then-exploit algorithm	11
1.3.2	From greedy to UCB	16
1.3.3	Boltzmann exploration	25
1.3.4	Simple policy gradient for stochastic bandits	27
1.4	Minimax lower bounds for stochastic bandits	31
2	More bandits	32
2.1	Adversarial bandits	32
2.2	Contextual bandits	32
II	Tabular Reinforcement Learning	33
3	Basics: Finite MDPs and Dynamic Programming Methods	34
3.1	Markov decision processes	34
3.1.1	A quick dive into Markov chains	34
3.1.2	Markov decision processes	35
3.1.3	Stochastic control theory	46
3.2	Basic value iteration algorithm	57
3.3	Basic policy iteration algorithm	61
3.3.1	Policy evaluation	61
3.3.2	Policy improvement	63
3.3.3	Exact and generalised policy iteration algorithms	67
3.4	Stochastic control in finite time	70
3.4.1	Setting and dynamic programming	71
3.4.2	Dynamic programming algorithms	74
4	Approximate dynamic programming methods	76
4.1	A guiding example	77
4.2	Monte Carlo policy evaluation and control	78
4.2.1	First visit Monte Carlo policy evaluation	79
4.2.2	Generalised policy iteration with first visit Monte Carlo estimation	81
4.3	Asynchronous stochastic fixed point iterations	82
4.3.1	Basic stochastic approximation theorem	82
4.3.2	Asynchronous stochastic approximation	85
4.4	One-step approximate dynamic programming TD(0)	93
4.4.1	One-step policy evaluation (approximate policy evaluation)	94
4.4.2	Q -learning and SARSA (approximate value iteration)	98
4.4.3	Double Q -learning	102

4.5	Multi-step approximate dynamic programming	107
4.5.1	n -step TD for policy evaluation and control	107
4.5.2	TD(λ) algorithms	110
III	Non-Tabular Reinforcement Learning	120
5	Policy Gradient methods	121
5.1	A quick dive into gradient descent methods	123
5.1.1	Gradient descent for L -smooth functions	124
5.1.2	Gradient descent for L -smooth, convex functions	127
5.1.3	Gradient descent for L -smooth functions with PL inequality	130
5.1.4	Stochastic gradient descent methods	132
5.1.5	Regularisation	132
5.2	Policy gradient theorems	132
5.2.1	Finite-time undiscounted MDPs	133
5.2.2	Infinite-time MDPs with discounted rewards	138
5.2.3	Convergence of REINFORCE	142
5.2.4	Variance reduction tricks	142
5.2.5	Natural policy gradient	144
5.3	A quick dive into neural networks	144
5.3.1	What are neural network functions?	144
5.3.2	Approximation properties	147
5.3.3	Differentiation properties	150
5.3.4	Using neural networks to parametrise policies	152
5.4	Reinforcement learning with (deep) function approximation	154
5.4.1	Simple actor-critic (AC) and advantage actor-critic (A2C)	154
5.4.2	Soft actor-critic (SAC)	158
5.4.3	Proximal policy optimisation (PPO)	158
6	Deep Q-learning	159
7	Monte Carlo Tree Search	160

Introduction

AlphaGo, protein folding, chatGPT. There is little doubt the reader of these lecture notes has crossed one of the most hot topics in artificial intelligence during the past years. Large companies such as Alphabet and Microsoft with their affiliated reinforcement learning teams have managed to achieve unbelievable progress in artificial intelligence that goes far beyond computer science. One piece amongst others is reinforcement learning, a mathematical concept to learn (near) optimal decision making. The aim of these notes is to introduce the most important basic concepts that should be understood before digging into the state of the art algorithms that lie behind this fascinating technology. To get a first impression of all basic ideas there is a wonderful book of Sutton and Barto, being targeted to a wide audience the book skips essentially all details. Our aim is to dig deeper into the details with the drawback of covering less topics.

- Multiarmed bandits, the simplest setting to understand some of the basic ideas like the exploration/exploitation trade-off.
- Classical Markov decision process theory, the basics of optimal decision making that must be well understood before proceeding to the advanced topics.
- Stochastic approximation and temporal difference techniques

Part I

Multiarmed Bandits

Chapter 1

Stochastic bandits

Multiarmed bandits can be considered to be the simplest situation in which optimal decision making can be learnt. Due to its simple structure many ideas get more visible that we will get to know much later for the more general setup of Markov decision processes. In fact, a vast literature of precise results exists for multiarmed bandits in contrast to many unproved methods for the general Markov decision situation. What we will try to achieve in this first chapter is to bridge the two worlds. We discuss ideas and methods that will be crucial for Markov decision processes mostly in the precise language of multiarmed bandits. The aim is to find the right compromise of the very imprecise Chapter 2 of Sutton and Barton and the very detailed book of Lattimore and Szepesvári.



The chapter focuses only on algorithmic ideas towards the trade-off between exploration and exploitation in optimal decision making. Topics like theoretical lower bounds will not be touched even though their importance cannot be underestimated at all.

The language used in this chapter will thus be a combination of bandit and reinforcement learning (RL) notation. The chapter can also be seen as a motivation for later chapters to see how little we actually understand about general reinforcement learning compared to the well-understood case of multiarmed bandits.

1.1 A quick dive into two-stage stochastic experiments

steht im WT Skript

1.2 Model, examples, and the goals

As there is no need to loose the reader in the mathematical formalisation of multiarmed bandits we will first gently introduce the ideas. For a multiarmed bandit there is a number of possible experiments among which a learner (in RL called agent) has the target to identify the best arm by observing random samples of the experiments. The goal is to learn efficiently which experiment yields the best outcome. There are different ways of defining what best means, in bandit theory best typically means the highest expectation. The simplest example to keep in mind is a daily visit to the university canteen. Let's say the canteen offers four choices: vegan, vegetarian, classic, oriental. These are the four possible experiments, the random outcomes are the quality of the dish (measured in some way, such as on a 1-10 scale). There are plenty of other situations that immediately come to mind such as

- medical treatment of patients with different doses, outcome measures the success, for instance 1 for healed and 0 otherwise,

- placing different advertisements on websites, outcome measures the click rates.

The wording multiarmed bandit is rather historic and comes from gambling. A one-armed bandit is a gambling machine in which every round of the game yields a random reward. Typically there are several one-armed bandits next to each other and a player aims to play the most favorable bandit. Since every round has a fixed cost, say one Euro, the player tries to figure out as quickly as possible which machine works best for him/her. In these notes we will only discuss so-called stochastic bandits. These are bandits where the random experiments are stationary, i.e. the distribution of the experiments do not change over time. More general bandit situations are adversarial bandits and contextual bandits which we will not touch in this section.

Before we go into more details let us discuss what we want to optimize. Suppose we have a finite set $\mathcal{A} = \{a_1, \dots, a_K\}$ of experiments (arms to play) and denote by Q_a the expectation of the random outcome whose distribution we denote by P_a . Of course there could be other goals than finding the arm with the highest average outcome, but this is what we decide to optimize. Now fix a time-horizon n (which could be infinite), the number of rounds we can use for learning. A learning strategy is a sequence $(\pi_t)_{t=1, \dots, n}$ of probability distributions on \mathcal{A} that only depend on what has been played prior to time t . Here $\pi_t(\{a\})$ is the probability that the player choses action a at time t and then receives the random outcome of the corresponding experiment.



Throughout these lecture notes we will be sloppy about measures on the power-set of discrete (finite or countably infinite) sets. Instead of writing $p(\{a\})$ we typically write $p(a)$ for probabilities of singleton sets if there is no danger of confusion.

The aim is to find a learning strategy that maximises the outcome of this procedure. To have an idea in mind think of the example of the university canteen. During your studies you might have $n = 500$ choices in total. If you are not vegan or vegetarian you probably started without preferences, i.e. $\pi_t(\text{vegan}) = \dots = \pi_t(\text{oriental}) = \frac{1}{4}$, and over time learnt from experience how to change the following distributions π_t in order to maximise the lunch experience.

Let's turn these basic ideas into mathematics.



Definition 1.2.1. Suppose \mathcal{A} is an index-set and $\nu = \{P_a\}_{a \in \mathcal{A}}$ is a family of real-valued distributions.

- The set ν is called a stochastic bandit model. In these lectures we will always assume $\mathcal{A} = \{a_1, \dots, a_K\}$ is finite, K is the number of arms.
- The action value (or Q -value) of an arm is defined by it's expectation $Q_a := \int_{\mathbb{R}} x P_a(dx)$. A best arm, usually denoted by a_* , is an arm with highest Q -value, i.e.

$$Q_{a_*} = \operatorname{argmax}_{a \in \mathcal{A}} Q_a.$$

Typically one abbreviates Q_* for the largest action value Q_a and if there are several optimal arms the argmax choses any of them.

- A learning strategy for n rounds ($n = +\infty$ is allowed) consists of
 - an initial distribution π_1 on \mathcal{A} ,
 - a sequence $(\pi_t)_{t=2, \dots, n}$ of kernels on $\Omega_{t-1} \times \mathcal{A}$,

where Ω_t denotes all trajectories $(a_1, x_1, a_2, x_2, \dots, a_t, x_t) \in (\mathcal{A} \times \mathbb{R})^t$. We will write the kernels in reverse ordering of the arguments

$$\pi_t(\cdot; a_1, x_1, a_2, x_2, \dots, a_t, x_t)$$

with the meaning that $\pi_t(\{a\}; a_1, x_1, a_2, x_2, \dots, a_t, x_t)$ is the probability arm a is chosen at time t if the past rounds resulted in actions/rewards $a_1, x_1, a_2, x_2, \dots, a_t, x_t$.

Recall that a probability distribution on a finite set is nothing but a probability vector (numbers in $[0, 1]$) that sum up to 1. An important special case occurs if the vector consists of one 1 (and 0s otherwise), i.e. the measure is a Dirac measure.



We will always assume that the action values Q_a are unknown but the bandit is a generative model, i.e. the random variables can be sampled. Everything that can be learnt about the model must be achieved by simulations (playing arms).

We will later see that learning strategies typically depend on different ingredients such as the maximal time-horizon if this is known in advance or certain quantities of the underlying bandit model. Of course it is desirable to have as little dependences as possible, but often additional information is very useful.



Definition 1.2.2. A learning strategy is called an index strategy if all appearing measures are Dirac measures, i.e. in all situations only a single arm is played with probability 1. The learning strategy is called soft if in all situations all arms have strictly positive probabilities.

Of course index strategies are just a small subset of all strategies but most algorithms we study are index strategies.

Next, we introduce stochastic bandit processes. This is a bit in analogy to Markov chains where first one introduces transition matrices and then defines a Markov chain and proves the existence. Or a random variable where first one defines the distribution function and then proves the existence of a random variable.



Definition 1.2.3. Suppose ν is a stochastic bandit model and $(\pi_t)_{t=1, \dots, n}$ a learning strategy for n rounds. Then a stochastic process $(A_t^\pi, X_t^\pi)_{t=1, \dots, n}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is called a stochastic bandit process with learning strategy π if $A_1^\pi \sim \pi_1$ and

- $\mathbb{P}(A_t^\pi = a | A_1^\pi, X_1^\pi, \dots, A_{t-1}^\pi, X_{t-1}^\pi) = \pi_t(\{a\}; A_1^\pi, X_1^\pi, \dots, A_{t-1}^\pi, X_{t-1}^\pi),$
- $\mathbb{P}(X_t^\pi \in \cdot | A_1^\pi, X_1^\pi, \dots, A_t^\pi) \sim P_{A_t^\pi},$

where $P_{A_t^\pi}$ is the (random) distribution $\sum_{a \in \mathcal{A}} P_a(\cdot) \mathbf{1}_{A_t^\pi = a}$. We will call A_t^π the action (the chosen arm) at time t and X_t^π the outcome (or reward) when playing arm A_t^π at time t . The superscripts π will always be dropped if the learning strategy is clear from the context.

In words, a stochastic bandit process is a stochastic process that works in a two-step fashion. Given a learning strategy π , in every round the strategy suggests probabilities for actions based on the past behavior. The sampled action A_t is then used to play the corresponding arm and observe the outcome. The process (A_t, X_t) thus describes the sequence of action/rewards over time.

Just as for random variables or Markov chains it is not completely trivial that there is a probability space and a stochastic process (A_t, X_t) that satisfies the defining property of a stochastic bandit process.



Theorem 1.2.4. For every stochastic bandit model and every learning strategy $(\pi_t)_{t=1, \dots, n}$ there is a corresponding stochastic bandit process.

Proof. We give a construction that is known under the name random table model as the bandit process is constructed from a table of independent random variables.



Recall that sampling from a discrete distribution π on \mathcal{A} can be performed using $\mathcal{U}([0, 1])$ random variables. Suppose $\pi(\{a_k\}) = p_k$ and $[0, 1]$ is subdivided into disjoint intervals I_k of lengths p_k , then the discrete random variable defined by $\bar{U} = a_k$ if and only if $U \in I_k$ is distributed according to π .

Now suppose $(X_t^{(a)})_{a \in \mathcal{A}, t \in \mathbb{N}}$ is a table of independent random variables such that $X_t^{(a)} \sim P_a$ for all t and suppose $(U_t)_{t \in \mathbb{N}}$ is a sequence of independent $\mathcal{U}([0, 1])$. These random variables (all defined on some joint probability space $(\Omega, \mathcal{F}, \mathbb{P})$) exist due to the Kolmogorov extension theorem.

	A_1	A_2	A_3	A_4	
	$\uparrow \pi_1$	$\uparrow \pi_2$	$\uparrow \pi_3$	$\uparrow \pi_4$	
	U_1	U_2	U_3	U_4	\dots
	$X_1^{(\mathbf{a}_1)}$	$X_2^{(a_1)}$	$X_3^{(a_1)}$	$X_4^{(\mathbf{a}_1)}$	\dots
	$X_1^{(a_2)}$	$X_2^{(a_2)}$	$X_3^{(\mathbf{a}_2)}$	$X_4^{(a_2)}$	\dots
	\vdots	\vdots	\vdots	\vdots	\vdots
	$X_1^{(a_\kappa)}$	$X_2^{(\mathbf{a}_\kappa)}$	$X_3^{(a_\kappa)}$	$X_4^{(a_\kappa)}$	\dots

Construction of bandit processes: bold entries were selected as $X_1, X_2, X_3, X_4, \dots$

If $(\pi_t)_{t \in \mathbb{N}}$ a learning strategy then the stochastic bandit process is constructed as follows:

- $t = 1$: Use U_1 to sample from the discrete measure $\pi_1(\cdot)$ an arm a , denote this arm by A_1 and set $X_1 := X_1^{(a)}$.
- $t \mapsto t + 1$: Use U_{t+1} to sample from the discrete (random) measures $\pi(\cdot; A_1, X_1, \dots, A_t, X_t)$ (relying only on the table to the left of column $t + 1$) an arm a , denote this arm by A_t and use the a th row from the rewards table to define X_t , i.e. $X_t = \sum_{a \in \mathcal{A}} X_t^{(a)} \mathbf{1}_{A_t=a}$.

To have a picture in mind think of a large table of random variables. Only using the variables to the left of column t , the uniform variable U_t is used to produce the action A_t and the X_t is produced by choosing the reward from row A_t . To see that this process (A, X) is indeed a bandit process with learning strategy π we need to check the defining properties. Let us denote by $I_a(a_1, \dots, x_{t-1})$ a partition of $[0, 1]$ into K disjoint intervals with lengths $\pi_t(\{a\}; a_1, \dots, x_{t-1})$. Then, using $h(u, a_1, \dots, x_{t-1}) := \mathbf{1}_{u \in I_a(a_1, \dots, x_{t-1})}$ and

$$\kappa(V, a_1, \dots, x_{t-1}) := \mathbb{P}(U \in V | A_1 = a_1, \dots, X_{t-1} = x_{t-1}) \stackrel{\text{ind.}}{=} \lambda(V), \quad V \in \mathcal{B}([0, 1]),$$

yields

$$\begin{aligned} \mathbb{P}(A_t = a | A_1, X_1, \dots, A_{t-1}, X_{t-1}) &= \mathbb{P}(U_t \in I_a(A_1, \dots, X_{t-1}) | A_1, X_1, \dots, A_{t-1}, X_{t-1}) \\ &= \mathbb{E}[h(U_t, A_1, \dots, X_{t-1}) | A_1, X_1, \dots, A_{t-1}, X_{t-1}] \\ &= \int h(u, A_1, \dots, X_{t-1}) \kappa(du, A_1, \dots, X_{t-1}) \\ &= \int \mathbf{1}_{u \in I_a(A_1, \dots, X_{t-1})} du \\ &= \pi_t(\{a\}; A_1, \dots, X_{t-1}). \end{aligned}$$

The second property is derived as follows:

$$\begin{aligned}
 \mathbb{P}(X_t \in \cdot | A_1, X_1, \dots, A_t) &= \sum_{a \in \mathcal{A}} \mathbb{P}(X_t \in \cdot, A_t = a | A_1, X_1, \dots, A_t) \\
 &= \sum_{a \in \mathcal{A}} \mathbb{P}(X_t^{(a)} \in \cdot, A_t = a | A_1, X_1, \dots, A_t) \\
 &\stackrel{\text{ind., meas.}}{=} \sum_{a \in \mathcal{A}} \mathbb{P}(X_t^{(a)} \in \cdot) \mathbf{1}_{A_t=a} \\
 &\stackrel{\text{Def. } P_{A_t}}{\sim} P_{A_t}.
 \end{aligned}$$

□

There is an equivalent way of constructing the process that sometimes yields a more convenient notation.



The random table model can be slightly modified to what is known as the stack of rewards model. The only difference is that all random variables appearing in the random table model are used. If the a th arm is played for the n th time then the reward variable $X_n^{(a)}$ is used instead of the reward variable corresponding to the time at which the a th arm is played for the n th time. In formulas, one sets $X_t = X_{T_a(t)}^{(a)}$ instead of $X_t = X_t^{(a)}$, where $T_a(t) = \sum_{s \leq t} \mathbf{1}_{X_s=a}$ is the number of times the a th arm was played before time t .

In mathematics it is always good to have concrete examples in mind. Here are two examples to keep in mind. These examples will always be used in the practical exercises.

Example 1.2.5. • Gaussian bandit: all arms are Gaussians $\mathcal{N}(\mu_i, \sigma_i^2)$, for instance

$$(\mu, \sigma) \in \{(0, 1), (1.1, 1), (0.9, 1), (2, 1), (-5, 1), (-3, 2)\}.$$

- Bernoulli bandit: all arms take value 1 with probability p_i and value 0 with probability $1 - p_i$, for instance

$$p \in \{0.9, 0.85, 0.8, 0.5, 0.1, 0.88, 0.7, 0.3, 0.88, 0.75\}.$$

Now that bandit models are defined the next step is to discuss the questions of interest. In fact, with the targets of this lecture course in mind this is not completely clear as the goals of stochastic bandit theory and reinforcement learning are different. The reason is that the research communities of statistics and AI traditionally have different examples in mind. While the stochastic bandit community originates from statistical question of optimal experimental design in medicine the AI community is more focused on artificial decision making (of computers). While both aim at developing and analysing algorithms that find the optimal arm, the different goals yield in different optimization goal. As an guiding example we go back to the two examples of medical treatment and advertisement. While in medical treatment every single round of learning refers to the medical treatment of an individual (which has the highest priority and the number of rounds is clearly limited say by $n = 200$) in online advertisement it might be much less problematic to play many rounds (say $n = 100k$) in the training procedure and to waste possible income. Here are three typical goals that we will formalise in due course:

- For fixed $n \in \mathbb{N}$ find an algorithm that produces a learning strategy $(\pi_t)_{t=1, \dots, n}$ such that the expected reward $\mathbb{E}[\sum_{k=1}^n X_k]$ is maximised.
- Find an algorithm that produces a learning strategy $(\pi_t)_{t \in \mathbb{N}}$ that converges (in some sense) quickly towards a best arm, i.e. to a Dirac measure on a best arm or a mixture of Dirac measures on best arms (if there are several). Try to minimize the computational complexity of the algorithm so that extensions to much more complicated situations are feasible.

- (C) For fixed $n \in \mathbb{N}$ find an algorithm that produces a learning strategy $(\pi_t)_{t=1, \dots, n}$ such that the probability of choosing wrong arms is minimized.

Checking papers and text books you will realise that the first goal is typical in the stochastic bandit community (statistics), the second typical in AI, and the third somewhat in between. The aim of these lecture notes is to introduce reinforcement learning, so why bother on discussing questions from stochastic bandit theory? The reason is that a much better mathematical understanding is available from the stochastic bandit community, optimal choices of parameters have been derived theoretically for many bandit algorithms. In contrast, the AI community tends to deal with more realistic (more complicated) models in which choices of parameters are found by comparing simulations. It is the goal of these lecture to find the right compromise. To understand well enough the important mechanisms in simple models to improve the educated guesses in realistic models that might be untractable for a rigorous mathematical analysis.

Let us first discuss the classical stochastic bandit approach (A). We already defined an optimal arm, an arm with maximal action value Q_a . Of course there might be several best arms, then a_* is chosen as any of them. Since the index set is not ordered there is no preference in which best arm to denote a_* . The goal is to maximise over all learning strategies the expectation $\mathbb{E}[\sum_{t=1}^n X_t] = \sum_{t=1}^n \mathbb{E}[X_t]$ for a fixed time-horizon n . There is a simple upper bound for the reward until time n , which is nQ_* . Hence, if all Q_a would be known in advance then the stochastic bandit optimization problem would be trivial, just chose the best arm a_* in all rounds. Hence, we always assume the expectations are unknown but the outcomes of the arms (random variables) can be played (simulated). Since the expected rewards are upper bounded by nQ_* it is common practice not to maximise the expected reward but instead the difference to the best case as this gives an objective criterion that does not depend on the bandit model itself.



Definition 1.2.6. Suppose ν is a bandit model and $(\pi_t)_{t=1, \dots, n}$ a learning strategy. Then the (cumulated) regret is defined by

$$R_n(\pi) := nQ_* - \mathbb{E}\left[\sum_{t=1}^n X_t\right].$$

The stochastic bandit problem consists in finding learning strategies that minimise the regret. If π is clear from the context then we will shorten to R_n .

The regret is called regret because (in expectation) this is how much is lost by not playing the best arm from the beginning. To get acquainted with the definition please check the following facts:



- Suppose a two-armed bandit with $Q_1 = 1$ and $Q_2 = -1$ and a learning strategy π given by

$$\pi_t = \begin{cases} \delta_1 & : t \text{ even,} \\ \delta_2 & : t \text{ odd.} \end{cases}$$

Calculate the regret $R_n(\pi)$.

- Define a stochastic bandit and a learning strategy such that the regret is 5 for all $n \geq 5$.
- Show for all learning strategies π that $R_n(\pi) \geq 0$ and $\limsup_{n \rightarrow \infty} \frac{R_n(\pi)}{n} < \infty$.
- Let $R_n(\pi) = 0$. Prove that π is deterministic, i.e. all π_t are almost surely constant and only chose the best arm.

What is considered to be a good learning strategy? Linear regret (as a function in n) is always

possible by just uniformly choosing arms as this (stupid) learning strategy yields

$$R_n(\pi) = nQ_* - n\mathbb{E}[X_1] = n\left(Q_* - \sum_{k=1}^K \frac{1}{K} Q_a\right).$$

Thus, a linearly increasing regret can always be achieved when learning nothing. As a consequence only learning strategies with sublinearly increasing regret are considered reasonable.



In stochastic bandit theory any algorithm that produces a learning strategies with linearly growing regret is considered useless. The aim is to significantly improve on linear regret.

There are different kind of bounds that one can aim for. First of all, one can aim for upper bounds and lower bounds for regret. In these lectures notes we mainly focus on upper bounds. Nonetheless, there are celebrated lower bounds due to Lai and Robbins that are not too hard to prove. These theoretical lower bounds are important as they tell us if there is any hope to search for better algorithms as the one we discuss (the actual answer is that one cannot do much better than the UCB algorithm presented below). Furthermore, the kind of estimates differ:

- bounds that depend on the bandit model ν are called model-based, such bounds typically involve the differences between the action values Q_a ,
- bounds that only depend on n are called model independent, they are typically proved for entire classes of bandit models for which certain moment conditions are assumed.

For the committ-then-explore and UCB algorithms below model-based upper bounds will be derived from which also model independent upper bounds can be deduced. We will see that it is not too hard to obtain algorithms that achieve model-based upper bounds that are logarithmic in n regret bounds that also involve differences of action values Q_a that can make the estimates as terrible as possible by choosing models where the best and second best arms are hard to distinguish. In fact, the model independent Lai-Robbins lower bounds shows that the best algorithm on all subgaussian bandits can only have a regret as good as $C\sqrt{Kn}$ for some constant C .



From the practical perspective one might wonder why to deal with regret upper bounds. If the bounds are reasonably good, then they can be used in order to tune appearing parameters to optimize the algorithms with guaranteed performance bounds. As an example, we will use the bounds for the explore-then-commit algorithm to tune the exploration lengths. Even though the resulting parameters might involve unrealistic quantities the understanding can still help us to understand how to work with the algorithms.

In principle, algorithms could depend on a time-horizon n if n is specified in advance. In that case asymptotic regret bounds are non-sense and we aim for finite n bounds only. Sometimes algorithms also depend on the unknown expectations Q_a through the so-called reward gaps.



Definition 1.2.7. The differences $\Delta_a := Q_* - Q_a$ are called reward gaps.

Of course it is not desirable to have algorithms that depend on the reward gaps as a priori knowledge of the expectations Q_a would turn the stochastic bandit problem into a trivial one (just chose the arm with the largest expectation). Nonetheless, the analysis of such algorithms can be of theoretial interest to better understand the mechanism of learning strategies. Also we will see some examples below, the explore-then-commit algorithm depends on the time-horizon n , so does the simple UCB algorithm, whereas the ε_n -algorithm is independent of n but mildly depends on the Δ_a through a constant. Bounds on the regret typically depend on n and the expectations μ_a often in the form $\Delta_a := Q_* - Q_a$.

In order to analyse the regret of a given algorithm in many instances (such as explore-then-commit and UCB) one always uses the regret decomposition lemma:

**Lemma 1.2.8. (Regret decomposition lemma)**

Defining $T_a(n) := \sum_{t=1}^n \mathbf{1}_{A_t=a}$ the following decomposition holds:

$$R_n(\pi) = \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}[T_a(n)].$$

Proof. If you know a bit of probability theory it is clear what we do, we insert a clever 1 that distinguishes the appearing events:

$$R_n(\pi) = nQ_* - \mathbb{E}\left[\sum_{t \leq n} X_t\right] = \sum_{t \leq n} \mathbb{E}[Q_* - X_t] = \sum_{t \leq n} \sum_{a \in \mathcal{A}} \mathbb{E}[(Q_* - X_t) \mathbf{1}_{A_t=a}].$$

To compute the right hand side note that

$$\begin{aligned} \mathbb{E}[(Q_* - X_t) \mathbf{1}_{A_t=a} \mid A_1, X_1, \dots, A_t] &= \mathbf{1}_{A_t=a} \mathbb{E}[Q_* - X_t \mid A_1, X_1, \dots, A_t] \\ &= \mathbf{1}_{A_t=a} (Q_* - Q_{A_t}) \\ &= \mathbf{1}_{A_t=a} (Q_* - Q_a) \\ &= \mathbf{1}_{A_t=a} \Delta_a. \end{aligned}$$

Using the tower property a combination of both computations yields

$$R_n = \sum_{t \leq n} \sum_{a \in \mathcal{A}} \mathbb{E}[\mathbb{E}[(Q_* - X_t) \mathbf{1}_{A_t=a} \mid A_1, X_1, \dots, A_t]] = \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}\left[\underbrace{\sum_{t \leq n} \mathbf{1}_{A_t=a}}_{T_a(n)}\right].$$

□

The statistical regret analysis for many bandit algorithm follows the same approach, using the regret decomposition lemma to reduce regret estimates to so-called concentration inequalities. Under suitable assumptions on the distributions of the arms one can plug-in different concentration inequalities from probability theory to derive regret bounds.

We continue our discussion with the second perspective on how to analyse bandit algorithms. Approach (C) is more refined than (A), as an analogue to function (C) is similar to studying the asymptotic behavior of a function through that of its derivative. To get this idea clear let us introduce a new notation:



Definition 1.2.9. Suppose ν is a bandit model and π a learning strategy. Then the probability the learner choses a suboptimal arm in round t , i.e.

$$\tau_t(\pi) := \mathbb{P}(Q_{A_t} \neq Q_*)$$

is called the failure probability in round t .

It is clearly desirable to have $\tau_t(\pi)$ decay to zero as fast as possible. Note that the failure probability is typically not the target for stochastic bandits but connects well to ideas in reinforcement learning. Since $\mathbb{E}[T_n(a)] = \sum_{t=1}^n \mathbb{E}[\mathbf{1}_{X_t=a}] = \sum_{t=1}^n \mathbb{P}(X_t = a)$ the regret decomposition lemma can be reformulated as follows:

**Lemma 1.2.10.**

$$R_n(\pi) = \sum_{t=1}^n \sum_{a \in \mathcal{A}} \Delta_a \mathbb{P}(X_t = a),$$



and, in particular,

$$R_n(\pi) \leq \max_a \Delta_a \sum_{t=1}^n \tau_t(\pi) \quad \text{and} \quad R_n(\pi) \geq \min_a \Delta_a \sum_{t=1}^n \tau_t(\pi).$$

As a consequence we see that the study of regret and failure probability is ultimately connected. If we interpret the sum as an integral, then understanding the failure probability instead of the regret is just as studying the asymptotics of a function by its derivate (which is typically harder). Here are two observations that we will use later for the examples:



- If the failure probabilities do not decay to zero (no learning of the optimal arm), then the regret grows linearly.
- If the failure probabilities behave (make this precise) like $\frac{1}{n}$, then the regret behaves like $\sum_{a \in \mathcal{A}} \Delta_a \log(n)$ with constants that depend on the concrete bandit model. Hint: Recall from basic analysis that $\int_1^t \frac{1}{x} dx = \log(t)$ and how to relate sums and integrals for monotone integrands.

The abstract discussion will become more accessible when analysing in detail specific algorithms. For explore-then-commit, using the regret-decomposition lemma, we will only estimate the regret while for the ε_n -greedy algorithm we will chose appropriate exploration rates to even upper bound the failure probabilities. The analysis is indeed crucial in order to improve the naive ε -greedy algorithm. It seems like the approach (A) is more common in the statistical bandits literature as there are not many examples for which the failure probabilities can be computed whereas in the reinforcement learning literature the approach (C) is more popular as for the most important example (ε -greedy) the failure rates are accessible.

For the reinforcement learning approach (B) there is actually not much (if at all) theory. We will discuss below the example of softmax-exploration in which the optimal arm is learned using gradient descent on a parametrised family of distributions on arms.

1.3 Algorithms: the exploration-exploitation trade-off

Lecture 2

We will now go into a few of the most popular algorithms. There is not much choice on how to design an algorithm. Essentially, all that can be done is to learn about arms that one is not too sure about (called exploration) and play arms that one expects to be good (called exploitation). We will not only present algorithms but also discuss theoretical regret bounds. Even though those won't be directly useful for our later understanding of reinforcement learning there is one important learning: theoretical results allow to understand how to chose optimally the parameters involved, in contrast to learn by experimenting which is always restricted to particular examples. In spirit we follow the exposition of Chapter 2 in Sutton and Barto, but we try to mix in more mathematics to push the understanding further than just simulations.

1.3.1 Basic committ-then-exploit algorithm

Without any prior understanding of stochastic bandits here is a simple algorithm that everyone would come up with himself/herself. Recalling the law of large numbers $\frac{1}{n} \sum_{t=1}^n Y_t \rightarrow \mathbb{E}[Y_1]$ we first produce estimates \hat{Q}_a for the expectations Q_a and then play the arm with the largest estimated expectation. How do we use the law of large numbers? By just playing every arm m times and for the remaining $n - mk$ rounds play the best estimated arm. That's it, that is the commit-then-exploit algorithm. Before turning the basic idea into an algorithm let us fix a notation that will occur again and again.



Definition 1.3.1. If (A_t, X_t) is a stochastic bandit process for some learning strategy π , then we define

$$\hat{Q}_a(t) := \frac{\sum_{k=1}^t X_k \mathbf{1}_{A_k=a}}{T_a(t)}, \quad a \in \mathcal{A},$$

and call \hat{Q} an estimated action value. $\hat{Q}_a(t)$ is the average return from playing arm a up to time t .

Here is a technical note on why estimated action values converge to the true action values if the number of rounds is infinite and arms are played infinitely often. Using the stack of rewards construction shows that the limit $\lim_{t \rightarrow \infty} \hat{Q}_a(t)$ is nothing but $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t X_k^{(a)}$, an average limit of an iid sequence which converges almost surely by the law of large numbers. The algorithm can be written in different ways. Either to try all arms m -times in a row or to alternate between the arms, for the pseudocode of Algorithm 1 we chose the latter. As a first example on how to

Algorithm 1: Basic m -rounds explore-then-commit algorithm

Data: m, n , bandit model ν, n

Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n

set $\hat{Q}(0) \equiv 0$;

while $t \leq n$ **do**

$$A_t := \begin{cases} at \bmod K + 1 & : t \leq mK \\ \operatorname{argmax}_a \hat{Q}_a(mK) & : t > mK \end{cases};$$

Obtain reward X_t by playing arm A_t ;

end

estimate the regret of bandit algorithms we prove the following simple upper bound. The notion 1-subgaussian will be introduced in the course of the proof, keep in mind Bernoulli bandits or Gaussian bandits with variance at most σ^2 .



Theorem 1.3.2. (Regret bound for simple explore-then-commit)

Suppose ν is a σ -subgaussian bandit model, i.e. all P_a are σ -subgaussian (see below), with K arms and $Km \leq n$ for some $n \in \mathbb{N}$, then

$$R_n(\pi) \leq \underbrace{m \sum_{a \in \mathcal{A}} \Delta_a}_{\text{exploration}} + \underbrace{(n - mK) \sum_{a \in \mathcal{A}} \Delta_a \exp\left(-\frac{m\Delta_a^2}{4\sigma^2}\right)}_{\text{exploitation}}.$$

Since the regret bound looks a bit frightening on first view let us discuss the ingredients first. What do we believe a bound should depend on? Certainly on the total number of rounds n , the number of exploration rounds m , and the number of arms. Probably also on the distributions of the arms. Why is this? If all arms have the same law, i.e. $\Delta_a = 0$ for all arms, then the regret would be 0 as we always play an optimal arm. If the best arm is much better than other arms, then the exploration phase forces a larger regret as we decided to also play the worst arm m times. Looking into the regret bound, the summand $m \sum_{a \in \mathcal{A}} \Delta_a$ is clearly the regret from the exploration phase.



Summands of the form $\sum_{a \in \mathcal{A}} \Delta_a$ must appear in all reasonable regret bounds as every reasonable algorithm must try every arm at least once.

The interesting question is the regret obtained during the exploitation phase if a suboptimal arm is played. It seems clear that the best arm is the most likely to be exploited as $\hat{Q}_a(n) \approx Q_a(n)$

for large n by the law of large numbers. The regret thus comes from deviations of this large n principle, if the rewards of an arm exceed what they would yield on average. Since the simple explore-then-commit algorithm involves a lot of independence probabilities overestimation of arms can easily be estimated by inequalities which are known as concentration inequalities in probability theory.

Proof, decomposing exploration and exploitation: Without loss of generality (the order of the arms does not matter) we may assume that $Q_* = Q_{a_1}$. Using the regret decomposition and the algorithm yields the following decomposition into exploitation and exploration:

$$R_n(\pi) = \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}[T_a(n)] = m \sum_{a \in \mathcal{A}} \Delta_a + \sum_{a \in \mathcal{A}} \Delta_a (n - mK) \mathbb{P}(\hat{Q}_a(mK) \geq \max_{b \in \mathcal{A}} \hat{Q}_b(mK)).$$

Where does this come from? Each arm is explored m times and, if the arm was the best, then another $n - mK$ times. Hence, $T_a(n) = m + (n - mK) \mathbf{1}_{\{a \text{ was best up to } mK\}}$. Computing the expectation the probability appears due to the construction of the learning strategy π . The probability can be estimated from above by replacing the maximal arm by some other arm (we chose the first). This leads to

$$\begin{aligned} R_n(\pi) &\leq m \sum_{a \in \mathcal{A}} \Delta_a + (n - mK) \sum_{a \in \mathcal{A}} \Delta_a \mathbb{P}(\hat{Q}_a(mK) \geq \hat{Q}_{a_1}(mK)) \\ &= m \sum_{a \in \mathcal{A}} \Delta_a + (n - mK) \sum_{a \in \mathcal{A}} \Delta_a \mathbb{P}((\hat{Q}_a(mK) - \hat{Q}_{a_1}(mK)) - (Q_a - Q_{a_1}) \geq \Delta_a). \end{aligned}$$

The appearing probability has the particularly nice form

$$\mathbb{P}(Z_a - \mathbb{E}[Z_a] \geq \Delta_a), \quad \text{with} \quad Z_a = \frac{1}{m} \sum_{j \leq m} (X_j^{(a)} - X_j^{(1)}),$$

where $X_1^{(a)}, \dots, X_m^{(a)}$ are distributed according to arm a and all of them are independent. If we can estimate the probabilities by $\exp(-m\Delta_a^2/4)$ the proof is complete. In order to do so we first need an excursion to probability theory. \square

Let's have a short excursion into the topic of concentration. A concentration inequality is a bound on the deviation of a random variable from its mean, either in a two- or one-sided fashion:

$$c(a) \leq \mathbb{P}(|X - \mathbb{E}[X]| > a) \leq C(a) \quad \text{or} \quad c(a) \leq \mathbb{P}(X - \mathbb{E}[X] > a) \leq C(a).$$

The faster the function $C(a)$ decreases in a the more the random variable is concentrated (takes values close to its expectation with larger probability, the randomness is less significant). The idea is that a random variable is stronger concentrated (has less mass away from its expectation) if larger moments are finite. Where this idea comes from is easily seen from the expectation formula

$$\mathbb{E}[g(X)] = \begin{cases} \int_{\mathbb{R}} g(y) f_X(y) dy & : X \text{ has the probability density function } f_X \\ \sum_{k=1}^N g(a_k) p_k & : X \text{ takes the values } a_k \text{ with probabilities } p_k \end{cases},$$

so that finite expectation for a (strongly) increasing function g such as an exponential function forces the density (or probability weights) to decrease (strongly) at infinity and thus to be more concentrated. Here is an example: Markov's inequality states that

$$\mathbb{P}(|X - \mathbb{E}[X]| > a) \leq \frac{\mathbb{V}[X]}{a^2}$$

for every random variable for which $\mathbb{E}[X^2] < \infty$. Markov's (concentration) inequality is useful as it holds for many random variables but the concentration inequality is very bad, the upper bound only decreases like a polynomial as we move away from the mean. A more useful inequality holds for so-called subgaussian random variables.



Definition 1.3.3. A random variable X on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ is called σ -subgaussian for $\sigma > 0$, if $\mathcal{M}_X(\lambda) = \mathbb{E}[e^{\lambda(X - \mathbb{E}[X])}] \leq e^{\frac{\lambda^2 \sigma^2}{2}}$ for all $\lambda \in \mathbb{R}$.

The wording subgaussian of course comes from the fact that $\mathbb{E}[e^{\lambda X}] = e^{\lambda^2 \sigma^2 / 2}$ if $X \sim \mathcal{N}(0, \sigma^2)$. Here is a little exercise to get acquainted to the definition:



- Show that every σ -subgaussian random variable satisfies $\mathbb{V}[X] \leq \sigma^2$.
- If X is σ -subgaussian, then cX is $|c|\sigma$ -subgaussian.
- Show that $X_1 + X_2$ is $\sqrt{\sigma_1^2 + \sigma_2^2}$ -subgaussian if X_1 and X_2 are independent σ_1 -subgaussian and σ_2 -subgaussian random variables.
- Show that a Bernoulli-variable is $\frac{1}{4}$ -subgaussian by explicitly computing $\log \mathcal{M}_{X-p}(\lambda)$, checking for which p the formula for $\log \mathcal{M}_{X-p}(\lambda)$ is maximal and then estimating the remaining function by $\frac{\lambda^2}{8}$.
- Every centered bounded random variable, say bounded below by a and above by b is $\frac{(b-a)}{2}$ -subgaussian (this is called Hoeffding's lemma).

It is important to note that every σ -subgaussian random variable is also σ' -subgaussian for every $\sigma' > \sigma$ but this is not interesting as we will use σ to bound the variability (σ somehow measures the variance) as good as possible. This becomes clear in the next proposition, using σ larger than necessary only weakens the concentration inequality:



Proposition 1.3.4. If X is σ -subgaussian, then

$$\mathbb{P}(X \geq a) \leq e^{-\frac{a^2}{2\sigma^2}} \quad \text{and} \quad \mathbb{P}(|X| \geq a) \leq 2e^{-\frac{a^2}{2\sigma^2}}, \quad \forall a > 0.$$

Proof. The proof is based on a trick called Cramér-Chernoff method. The trick uses the Markov inequality for a parametrized family of functions and then optimising over the parameter to find the best estimate:

$$\mathbb{P}(X \geq a) = \mathbb{P}(e^{\lambda X} \geq e^{\lambda a}) \leq \frac{\mathbb{E}[e^{\lambda X}]}{e^{\lambda a}} \leq \frac{e^{\frac{\lambda^2 \sigma^2}{2}}}{e^{\lambda a}} = e^{\frac{\lambda^2 \sigma^2}{2} - \lambda a}.$$

Minimizing the right hand side for λ (differentiation!) shows that $\lambda = \frac{a}{\sigma^2}$ yields the smallest bound and this is the first claim of the proposition. Since the same holds for $-X$ we obtain the second claim by writing $\mathbb{P}(|X| \geq a) = \mathbb{P}(X \geq a \text{ or } X \leq -a) \leq \mathbb{P}(X \geq a) + \mathbb{P}(-X \geq a)$. \square

As an application of the above we get a first simple concentration inequality for sums of random variables:



Corollary 1.3.5. (Hoeffding's inequality)

Suppose X_1, \dots, X_n are iid random variables on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ with expectation $\mu = \mathbb{E}[X_1]$ such that X_1 is σ -subgaussian. Then

$$\mathbb{P}\left(\frac{1}{n} \sum_{k=1}^n X_k - \mu \geq a\right) \leq e^{-\frac{na^2}{2\sigma^2}} \quad \text{and} \quad \mathbb{P}\left(\left|\frac{1}{n} \sum_{k=1}^n X_k - \mu\right| \geq a\right) \leq 2e^{-\frac{na^2}{2\sigma^2}}, \quad \forall a > 0.$$

Proof. This follows from the exercise and the proposition above because $\frac{1}{n} \sum_{k=1}^n X_k - \mu = \frac{1}{n} \sum_{k=1}^n (X_k - \mathbb{E}[X_k])$ is a centered $\frac{\sigma}{\sqrt{n}}$ -subgaussian random variable. \square

We can now combine the exploration decomposition with the concentration inequality to derive the upper bound of the regret in the explore-then-commit algorithm:

Completing the proof of Theorem 1.3.2. Since we assumed that the exploitation phase consists of independent runs of the same arm we are exactly in the situation of Corollary 1.3.5. Hence, with the notation from the first part of the proof we get the concentration bound

$$\mathbb{P}(Z_a - \mathbb{E}[Z_a] \geq \Delta_a) \leq \exp\left(-\frac{\Delta_a^2}{2\frac{\sigma^2}{m}}\right) = \exp\left(-\frac{m\Delta_a^2}{4\sigma^2}\right).$$

Plugging-in yields the upper bound from the theorem. \square

Also without the estimates the following logic is clear: If m is large (a lot of exploration) then the exploration regret is large (this is the first summand) and the exploitation regret is small (second summand). In the exercises you will explore numerically how to properly choose m in different examples. Let's explore the regret upper bound to find a reasonable choice of m . Of course, this approach is only reasonable if the upper bound is reasonably good. Indeed, the first summand is an equality, the second summand only uses Hoeffding's inequality which is a reasonably good estimate. To show how to find a good exploration length let us consider the simple case $K = 2$ of two arms (again, the first arm is assumed to be the optimal one). Since $\Delta_{a_1} = 0$, we abbreviate $\Delta = \Delta_{a_2}$, the estimate simplifies to

$$R_n(\pi) \leq m\Delta + (n - 2m)\Delta \exp\left(-\frac{m\Delta^2}{4\sigma^2}\right) \leq \Delta\left(m + n \exp\left(-\frac{m\Delta^2}{4\sigma^2}\right)\right).$$

Pretending that m is a continuous parameter the righthand side can be minimized (in m) by differentiation. Do this to solve the following exercise:



The regret upper bound is minimized by

$$m = \max\left\{1, \left\lceil \frac{4\sigma^2}{\Delta^2} \log\left(\frac{n\Delta^2}{4\sigma^2}\right) \right\rceil\right\}. \quad (1.1)$$

Thus, in terms of regret optimisation, our best guess is the explore-then-commit algorithm with this particular m . For this choice of m the regret is upper bounded by

$$R_n(\pi) \leq \min\left\{n\Delta, \Delta + \frac{4\sigma^2}{\Delta} \left(1 + \max\left\{0, \log\left(\frac{n\Delta^2}{4\sigma^2}\right)\right\}\right)\right\}, \quad (1.2)$$

which for $n \geq \frac{4}{\Delta^2}$ gives a model-dependent logarithmic regret bound $R_n(\pi) \leq C_\Delta + \frac{\log(n)}{\Delta}$.

In the programming exercises you will be asked to compare this theoretical m with the best m that can be „seen“ from simulations in a special example. No doubt, tuning parameters by simulating examples is not very appealing as the parameter might be useless for other examples.



Do you think the explore-then-commit algorithm with m from (1.1) is reasonable? Well, in most situations it is not. The algorithm relies on n and Δ through the choice of m . While the number of rounds n might be fixed in advance for some examples it might be not for other examples. The dependence on Δ is much more severe as the action values are never known in advance (otherwise we could just choose a best arm to obtain zero regret). Hence, the only non-trivial situation is that of unknown action values but known difference $\Delta = Q_{a_1} - Q_{a_2}$, but this is extremely special.

Throughout these lectures notes we will again and again discuss dependences of algorithms and model parameters. What does the algorithm depend on and sometimes even more important

what does the algorithm not depend on? Sometimes it turns out that understanding proofs is crucial in order to figure out what algorithms do not depend on, hence, understanding how algorithms can be modified without destroying major properties such as convergence.

1.3.2 From greedy to UCB

A greedy learning strategy is a strategy that always choses the arm that has proven to be best in the past. Since such concepts will reappear in reinforcement learning in policy iteration and generalised policy iteration algorithms we will spend some time on this topic. It will turn out that pure greedy algorithms do not work at all, but modifications that force additional exploitation work very well.

Purely greedy bandit algorithm

The pure greedy algorithm is complete non-sense if we think about it for a second or two. Nonetheless, it is a good start into the discussion to get acquainted with the notation and simple modifications give useful algorithms. The plain idea is as follows: Take the past observations of each arm to define estimates $\hat{Q}_a(t-1)$ of the action values Q_a at time t and then chose the maximal estimated arm, i.e. $A_t := \operatorname{argmax}_a \hat{Q}_a(t-1)$. As always, since there is no preference order for the arms, if several estimated action values are equal we take any of them. The pure

Algorithm 2: Purely greedy bandit algorithm

Data: bandit model ν , initialisation $\hat{Q}(0)$, n
Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n
while $t \leq n$ **do**
 Set $A_t = \operatorname{argmax}_a \hat{Q}_a(t-1)$;
 Obtain reward X_t by playing arm A_t ;
 Update the estimated action value function $\hat{Q}(t)$;
end

greedy algorithm depends extremely on the initiation of $\hat{Q}(0)$ and the distribution of the arms. As an extreme szenario, suppose $\hat{Q}(0) \equiv 0$. Suppose one arm only returns positive values and this arm is chosen at the beginning. Then the estimated action value is increased to a positive number and no other arm will be played in the future. Similarly, if an arm takes positive values with high probability then future exploration is very unlikely to occur. Also for a Gaussian bandit model a similar phenomenon arises. Suppose (at least) one arm has positive expectation and suppose $3\sigma < \mu$. If initially that arm is played then with probability at least 0.997 (three- σ -rule) the result will be positive so that the arm will be played again. Continuing like that it will take a long time until eventually a different arm will be explored. Since that phenomenal will occur again we give it a name:



Definition 1.3.6. The phenomenon that a bandit (later: reinforcement learning) algorithm focuses too early on a suboptimal decision is called committal behavior.

From a practical point of view there are workarounds for instance using different initialisations $\hat{Q}(0)$. As an example one might start with extremely large $\hat{Q}(0)$. In that case the algorithms would start with many rounds of exploitation before starting the greedy update. A problem remains: How large would be good without knowing details of the bandit model? If $\hat{Q}(0)$ is chosen too large there would be too much exploitation before playing greedy the arms with the largest estimated action values. If $\hat{Q}(0)$ would not be large enough then there would be too little exploitation. A second workaround would be trying to center the distributions of all arms by subtracting the same constant from all arms to reduce the committal behavior effect described above. Again, it is unclear what constant should be subtracted without a priori knowledge on

the action values. We will get back to this idea when we discuss the policy gradient method where this idea will return as the base-line trick.

ε -greedy bandit algorithms

The simplest variant to make the pure greedy algorithm more reasonable is to force completely random additional exploitation. The idea originates from the reinforcement learning community as from the point of view of minimizing regret the algorithm is completely useless. To get a feeling

Algorithm 3: ε -greedy bandit algorithm

Data: bandit model ν , exploration rate $\varepsilon \in (0, 1)$, initialisation $\hat{Q}(0)$, n
Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n

```

while  $t \leq n$  do
  Chose  $U \sim \mathcal{U}([0, 1])$ ;
  if  $U < \varepsilon$  then
    [exploration part];
    Uniformly chose an arm  $A_t$ ;
    Obtain reward  $X_t$  by playing arm  $A_t$ ;
    Update the estimated action value function  $\hat{Q}(t)$ ;
  end
  if  $U \geq \varepsilon$  then
    [greedy part];
    Set  $A_t = \operatorname{argmax}_a \hat{Q}_a(t-1)$ ;
    Obtain reward  $X_t$  by playing arm  $A_t$ ;
    Update the estimated action value function  $\hat{Q}(t)$ ;
  end
end
end

```

Lecture 3

for the algorithm you will be asked to run different simulations in the exercises. Simulations of this kind can be very useful to understand basic mechanisms, but not much more. Of course, for this particular example we could repeat the simulations again and again to fine-tune the choice of ε . But this does not result in further understanding of the basic mechanisms for an unknown problem. So let us have a quick look at some theory. The following simple estimate shows that ε -greedy is not interesting for regret minimisation. The exploitation phase only results in a lower bound

$$\tau_t(\pi) \geq \varepsilon \left(1 - \frac{K_*}{K}\right)$$

for the failure probability, where K_* denotes the number of arms with optimal action value Q_* . Hence, from the point of view of stochastic bandit learning the ε -greedy strategy is useless, the regret growth linearly (compare Lemma 1.2.10).

There are many versions of ε -greedy with reasonable regret bounds. We will only touch upon an example (introduced and studied in Auer et al.¹) that one could call „explore-then- ε -greedy with decreasing exploitation rate“. The algorithm replaces in the simple ε -greedy algorithm ε by the time-dependent exploration rate $\varepsilon_t = \min \left\{1, \frac{CK}{d^2 t}\right\}$. To justify the name note that for the first $\frac{CK}{d^2}$ rounds the exploration rate is 1. Thus, the algorithm first explores randomly and then plays ε -greedy with decreasing exploration rate ε .



Theorem 1.3.7. (Explore-then- ε -greedy with decreasing ε)

Suppose that all arms take values in $[0, 1]$, $d < \min_{a: Q_a \neq Q_*} \Delta_a$, and $C >$

¹P. Auer, N. Cesa-Bianchi, P. Fischer: „Finite-time Analysis of the Multiarmed Bandit Problem“, Machine Learning, 47:235-256, (2002)



$\max\{5d^2, 2\}$. Then the ε -greedy algorithm with decreasing exploration rate $\varepsilon_t = \min\{1, \frac{CK}{d^2 t}\}$ satisfies

$$\limsup_{n \rightarrow \infty} \frac{\tau_n(\pi)}{n} \leq \frac{(K-1)C}{d^2}.$$

Using Lemma 1.2.10 (note that $\Delta_a \leq 1$ if the arms only take values in $[0, 1]$) a consequence of the theorem is logarithmic upper bound

$$\limsup_{n \rightarrow \infty} \frac{R_n(\pi)}{\log(n)} \leq \frac{(K-1)C}{d^2}. \quad (1.3)$$

While logarithmically growing regret is very good (compare the UCB algorithm in Theorem 1.3.8 below) there are two disadvantages. First, the constants are pretty big (the possibly very small reward gap even appears twice in the numerator) and will dominate the logarithm for reasonably sized n ($\log(100.000) \approx 11.5$ so that even an additional factor 5 matters quite a lot). Secondly, with the appearing constant d the algorithm assumes prior knowledge on the bandit model reward gaps, a feature that is highly unwanted. Compared to other algorithms there is also a clear advantage. The number of rounds n does not enter the algorithm.

Proof (not part of the course). Auer et al. proved a much more precise estimate. Suppose j is a suboptimal arm and $n > \frac{CK}{d^2}$. Then we prove for all $C > 0$ that

$$\begin{aligned} \mathbb{P}(A_t = j) &\leq \frac{C}{d^2 n} + 2 \left(\frac{C}{d^2} \log \left(\frac{(n-1)d^2 e^{1/2}}{CK} \right) \right) \left(\frac{CK}{(n-1)d^2 e^{1/2}} \right)^{C/(5d^2)} \\ &\quad + \frac{4e}{d^2} \left(\frac{CK}{(n-1)d^2 e^{1/2}} \right)^{C/2}. \end{aligned}$$

Since there are at most $K-1$ suboptimal arms an upper bound for $\tau_t(\pi)$ is obtained by multiplying by $(K-1)$. The choice $C > 5$ ($C > 5d$ actually suffices) implies that the first summand dominates in the limit.



The failure probability for the first $\frac{CK}{d^2}$ rounds (uniform exploration) is easily seen to be $\tau_t(\pi) = 1 - \frac{K^*}{K}$, the probability to chose one of the suboptimal arms.

The proof is mostly a brute force estimation of the probabilities plus Bernstein's inequality a concentration inequality that is a bit more general than Hoeffding's inequality:



Suppose X_1, \dots, X_n are independent (not necessarily identically distributed!) random variables with variances σ_k^2 and expectations $\mu_k = \mathbb{E}[X_k]$. If all X_i are bounded by M , i.e. $|X_i| \leq M$ for all i , $\sigma^2 := \sum_{k=1}^n \sigma_k^2$, then

$$\mathbb{P}\left(\frac{1}{n} \sum_{k=1}^n X_k - \frac{1}{n} \sum_{k=1}^n \mu_k \geq a\right) \leq e^{-\frac{n a^2}{2(\sigma^2 + \frac{1}{3} n a M)}}.$$

In particular, if X_1, \dots, X_n are iid with variance σ^2 and expectation μ then Bernstein's inequality becomes

$$\mathbb{P}\left(\frac{1}{n} \sum_{k=1}^n X_k - \mu \geq a\right) \leq e^{-\frac{n a^2}{2(\sigma^2 + \frac{1}{3} a M)}}.$$

Let $x_0 := \frac{1}{2K} \sum_{s=1}^t \varepsilon_s$ with $\varepsilon_n = \frac{CK}{d^2 n}$ the exploration rate from the algorithm. Suppose j is a

suboptimal arm, we are going to estimate $\mathbb{P}(A_t = j)$. From the algorithm we obtain

$$\begin{aligned}\mathbb{P}(A_t = j) &= \frac{\varepsilon_n}{K} + \left(1 - \frac{\varepsilon_n}{K}\right) \mathbb{P}(\hat{Q}_j(t-1) \geq \max_a \hat{Q}_a(t-1)) \\ &\leq \frac{\varepsilon_n}{K} + \left(1 - \frac{\varepsilon_n}{K}\right) \mathbb{P}(\hat{Q}_j(t-1) \geq \hat{Q}_*(t-1)) \\ &\leq \frac{\varepsilon_n}{K} + \left(1 - \frac{\varepsilon_n}{K}\right) \left(\mathbb{P}(\hat{Q}_j(t-1) \geq Q_j + \Delta_j/2) + \mathbb{P}(\hat{Q}_*(t-1) < Q_{a_*} - \Delta_j/2)\right),\end{aligned}$$

where Q_* denotes the estimated action value for a fixed optimal arm, say the first. Here we used that, by definition of Δ_j , $Q_* - \Delta_j/2 = Q_j + \Delta_j/2$ and the elementary estimate

$$\mathbb{P}(X \geq Y) = \mathbb{P}(X \geq Y, Y \geq a) + \mathbb{P}(X \geq Y, Y < a) \leq \mathbb{P}(X \geq a) + \mathbb{P}(Y < a).$$

From the above we estimate both probabilities separately (the argument is the same). Denote by $T_j^R(t)$ the numbers of random explorations of the arm j before time t . Then

$$\begin{aligned}\mathbb{P}(\hat{Q}_j(t) \geq Q_j + \Delta_j/2) &= \sum_{s=1}^t \mathbb{P}(\hat{Q}_j(t) \geq Q_j + \Delta_j/2, T_j(t) = s) \\ &= \sum_{s=1}^t \mathbb{P}(T_j(t) = s \mid \hat{Q}_j(t) \geq Q_j + \Delta_j/2) \mathbb{P}(\hat{Q}_j(t) \geq Q_j + \Delta_j/2) \\ &\stackrel{1.3.5}{\leq} \sum_{s=1}^t \mathbb{P}(T_j(t) = s \mid \hat{Q}_j(t) \geq Q_j + \Delta_j/2) e^{-\Delta_j^2 t/2},\end{aligned}$$

using that random variables with values in $[0, 1]$ are $\frac{1}{2}$ -subgaussian. Splitting the sum into the sum up to $\lfloor x_0 \rfloor$ and the rest, using the estimate $\sum_{t=x+1}^{\infty} e^{-\kappa t} \leq \frac{1}{\kappa} e^{-\kappa x}$ (think of the integral!) yields the upper bounded

$$\begin{aligned}&\sum_{s=1}^{\lfloor x_0 \rfloor} \mathbb{P}(T_j(t) = s \mid \hat{Q}_j(t) \geq Q_j + \Delta_j/2) + \frac{2}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor / 2} \\ &\leq \sum_{s=1}^{\lfloor x_0 \rfloor} \mathbb{P}(T_j^R(t) \leq \lfloor x_0 \rfloor \mid \hat{Q}_j(T) \geq Q_j + \Delta_j/2) + \frac{2}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor / 2} \\ &= \lfloor x_0 \rfloor \mathbb{P}(T_j^R(n) \leq \lfloor x_0 \rfloor) + \frac{2}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor / 2},\end{aligned}$$

where the conditioning could be dropped because the exploration is independent. Using Benaymé and mean, variance of Bernoulli random variables yields

$$\mathbb{E}[T_j^R(t)] = \frac{1}{K} \sum_{s=1}^t \varepsilon_s = 2x_0 \quad \text{and} \quad \mathbb{V}[T_j^R(t)] = \sum_{s=1}^t \frac{\varepsilon_s}{K} \left(1 - \frac{\varepsilon_s}{K}\right) \leq \frac{1}{K} \sum_{s=1}^t \varepsilon_s$$

so that Bernstein's inequality gives $\mathbb{P}(T_j^R(t) \leq \lfloor x_0 \rfloor) \leq e^{-x_0/5}$. In total we derived the bound

$$\mathbb{P}(\hat{Q}_j(t) \geq Q_j + \Delta_j/2) \leq \lfloor x_0 \rfloor e^{-x_0/5} + \frac{2}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor / 2}$$

From the probabilistic point the proof is complete but we have not taking into account the choice of ε . It only remains to play around with $x_0 = \frac{1}{2K} \sum_{t=1}^n \varepsilon_t$ to simplify the presentation. Recall the choice of the exploitation rate ε_t and set $n' = \frac{CK}{d^2}$. The exploitation rate is constant 1 up to n' and then equal to $\varepsilon_t = \frac{CK}{d^2 t}$, hence,

$$x_0 = \frac{1}{2K} \sum_{t=1}^{n'} \varepsilon_t + \frac{1}{2K} \sum_{t=n'+1}^n \varepsilon_t \geq \frac{n'}{2K} + \frac{C}{d^2} \log\left(\frac{n}{n'}\right) \geq \frac{C}{d^2} \log\left(\frac{nd^2 e^{1/2}}{CK}\right).$$

Putting everything together yields, $x \mapsto xe^{-x/5}$ is decreasing for $x > 5$, for a suboptimal arm j and $n \geq n'$,

$$\begin{aligned} \mathbb{P}(A_t = j) &\leq \frac{\varepsilon_n}{K} + 2\lfloor x_0 \rfloor e^{-\lfloor x_0 \rfloor / 5} + \frac{4}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor / 2} \\ &\leq \frac{C}{d^2 n} + 2 \left(\frac{C}{d^2} \log \left(\frac{(n-1)d^2 e^{1/2}}{CK} \right) \right) \left(\frac{CK}{(n-1)d^2 e^{1/2}} \right)^{C/(5d^2)} \\ &\quad + \frac{4e}{d^2} \left(\frac{CK}{(n-1)d^2 e^{1/2}} \right)^{C/2}. \end{aligned}$$

□

We are not going to discuss further the ε -greedy algorithm. It turns out that that the dependence on the parameters K, Δ_a, C is horribly bad compared to the UCB algorithm that is discussed next. Still, it is important to keep the previous algorithm in mind as ε -greedy algorithms are used frequently in reinforcement learning.

UCB algorithm

The UCB algorithm (upper confidence bound algorithm) is usually not considered as a version of the greedy algorithm but more a further development of explore-then-commit. UCB follows similar ideas that involve concentration inequalities but with more thoughts. Still, since UCB can also be seen as greedy with an additional exploration bonus we prefer to interpret UCB as an extension of greedy. For that sake define the UCB function:

$$\text{UCB}_a(t, \delta) := \begin{cases} \infty & : T_a(t) = 0 \\ \underbrace{\hat{Q}_a(t)}_{\text{greedy}} + \underbrace{\sqrt{\frac{2 \log(1/\delta)}{T_a(t)}}}_{\text{exploration bonus}} & : T_a(t) \neq 0. \end{cases}$$

Algorithm 4: UCB algorithms

Data: bandit model $\nu, \delta \in (0, 1), n$
Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n
while $t \leq n$ **do**
 $A_t = \operatorname{argmax}_a \text{UCB}_a(t-1, \delta);$
 Obtain reward X_t by playing arm A_t ;
 Update the estimated action value function $\hat{Q}(t);$
end

Note that the initialisation $\text{UCB}(0, \delta) \equiv +\infty$ forces the algorithm to explore every arm at least once, a condition that reasonable algorithms should fulfill. A part from this is important to note that the algorithm does not use any model information, in contrast to the optimal explore-then-commit and the explore-then- ε greedy with decreasing exploration rate.

The choice of δ is extremely crucial for the performance of the algorithm as can be seen very well from simulations (do it!). When choosing δ to depend on n we can derive the following regret upper bound.



Theorem 1.3.8. Suppose ν is a bandit model with 1-subgaussian arms, $n \in \mathbb{N}$, and $\delta = \frac{1}{n^2}$. Then the UCB algorithms has the following regret upper bound:

$$R_n(\pi) \leq 3 \sum_{a \in \mathcal{A}} \Delta_a + 16 \log(n) \sum_{a: Q_a \neq Q_*} \frac{1}{\Delta_a}.$$

The regret upper bound is better than the logarithmic regret bounds that we have seen for the algorithms before. In fact, the simulation exercises show that the simple UCB algorithms performs very well on most examples. The only drawback is the appearing $\delta = \frac{1}{n^2}$ in the algorithm to obtain the regret upper bound. If the time-horizon is not fixed in advance this algorithmic choice is of course problematic.

Proof. Before we go into the estimates let's have a quick look at the concentration inequalities again. Chosing a suitably we can make the exponential dissappear:

$$\mathbb{P}(X \geq \mathbb{E}[X] + \sqrt{2 \log(1/\delta)}) \leq \delta, \quad \text{and} \quad \mathbb{P}(|X - \mathbb{E}[X]| \geq \sqrt{2 \log(1/\delta)}) \leq 2\delta, \quad (1.4)$$

if X is σ -subgaussian. In other words, if δ is small, then a σ -subgaussian random variable takes with values in the confidence interval $(-\sqrt{2\sigma^2 \log(1/\delta)}, \sqrt{2\sigma^2 \log(1/\delta)})$ with high probability. In essence, this inequality is the reason for the logarithmic regret boundes that will appear all the time, most importantly in the UCB algorithm below. The concentration inequality for sums of σ -subgaussian iid random variables with mean μ (Hoeffdings inequality 1.3.5) gives

$$\mathbb{P}\left(\sum_{k=1}^n X_k \geq \mu + \sqrt{\frac{2 \log(1/\delta)}{n}}\right) \leq \exp\left(-\frac{n\left(\sqrt{\frac{2 \log(1/\delta)}{n}}\right)^2}{2}\right) = \delta, \quad (1.5)$$

which will explain the choice of the exploration bonus.

We will assume again without loss of generality that $a_* = a_1$ and estimate the expected times a suboptimal arm a will be played. Combined with the regret decomposition Lemma 1.2.8 this will give the upper bound on the regret. For the proof we will use the random table construction of the bandit process (A, X) . For that sake recall the array $\{X_t^{(a)}\}_{t \leq n, a \in \mathcal{A}}$ of independent random variables with $X_t^{(a)} \sim P_a$. One way of expressing the bandit process X_t is as $X_{T_{A_t}(t)}^{(A_t)}$. Now define

$$\bar{Q}_s^{(a)} = \frac{1}{s} \sum_{k \leq s} X_k^{(a)}.$$

We will study the event $G_a = G_1 \cap G_{a,2}$ with

$$G_1 = \{\omega : Q_{a_1} < \min_{t \leq n} \text{UCB}_{a_1}(t, \delta)(\omega)\},$$

$$G_{a,2} = \left\{ \omega : \bar{Q}_{u_a}^{(a)}(\omega) + \sqrt{\frac{2 \log(1/\delta)}{u_a}} < Q_{a_1} \right\},$$

with a natural number $u_a \leq n$ to be specified later. The idea of the proof lies in the following observation:

$$\text{If } \omega \in G_a, \text{ then } T_a(n)(\omega) \leq u_a. \quad (1.6)$$

Let's first check why (1.6) holds by assuming the contrary. Suppose $\omega \in G_a$ but $T_a(n)(\omega) > u_a$ holds. Then there must be some time index $t \leq n$ with $T_a(t-1)(\omega) = u_a$ and $A_t(\omega) = a$. Using the definitions of $G_1, G_{a,2}$, and UCB yields

$$\begin{aligned} \text{UCB}_a(t-1, \delta)(\omega) &\stackrel{\text{Def.}}{=} \hat{Q}_a(t-1)(\omega) + \sqrt{\frac{2 \log(1/\delta)}{T_a(t-1)(\omega)}} \\ &\stackrel{(d)}{=} \bar{Q}_{u_a}^{(a)}(\omega) + \sqrt{\frac{2 \log(1/\delta)}{u_a}} \\ &\stackrel{G_{a,2}}{<} Q_{a_1} \\ &\stackrel{G_1}{<} \text{UCB}_{a_1}(t-1, \delta)(\omega), \end{aligned}$$

a contradiction to $a = A_t(\omega) = \operatorname{argmax}_b \operatorname{UCB}_b(t-1, \delta)$ which is correct by the definition of the algorithm. Hence, (1.6) holds. Since $T_a(n)$ is trivially bounded by n the following key estimate holds:

$$\mathbb{E}[T_a(n)] = \mathbb{E}[T_a(n)\mathbf{1}_{G_a}] + \mathbb{E}[T_a(n)\mathbf{1}_{G_a^c}] \leq u_a + n(\mathbb{P}(G_1^c) + \mathbb{P}(G_{a,2}^c)). \quad (1.7)$$

Thus, in order to estimate the expected number of playing arms it is enough to estimate both probabilities. It now suffices to estimate separately both summands on the right hand side of (1.7). First, it holds that

$$\begin{aligned} \mathbb{P}(G_1^c) &= \mathbb{P}(Q_{a_1} \geq \operatorname{UCB}_{a_1}(t, \delta) \text{ for some } t \leq n) \\ &\leq \sum_{t \leq n} \mathbb{P}\left(Q_{a_1} - \sqrt{\frac{2 \log(1/\delta)}{t}} \geq \hat{Q}_{a_1}(t)\right) \\ &\stackrel{(1.5)}{\leq} \sum_{t \leq n} \delta = n\delta. \end{aligned}$$

Next, we chose u_a which so far was left unspecified. Let us chose u_a large enough so that

$$\Delta_a - \sqrt{\frac{2 \log(1/\delta)}{u_a}} \geq \frac{1}{2}\Delta_a \quad (1.8)$$

holds, for instance $u_a = \frac{2 \log(1/\delta)}{\frac{1}{4}\Delta_a^2} + 1$. Then

$$\begin{aligned} \mathbb{P}(G_{a,2}^c) &= \mathbb{P}\left(\bar{Q}_{u_a}^{(a)} + \sqrt{\frac{2 \log(1/\delta)}{u_a}} \geq Q_{a_1}\right) \\ &= \mathbb{P}\left(\bar{Q}_{u_a}^{(a)} - Q_a \geq \Delta_a - \sqrt{\frac{2 \log(1/\delta)}{u_a}}\right) \\ &\stackrel{(1.8)}{\leq} \mathbb{P}\left(\bar{Q}_{u_a}^{(a)} \geq Q_a + \frac{1}{2}\Delta_a\right) \\ &\stackrel{\text{Hoeffding}}{\leq} \exp\left(-\frac{u_a \Delta_a^2}{8}\right). \end{aligned}$$

Combining the above yields

$$\mathbb{E}[T_a(n)] \leq u_a + n\left(n\delta + \exp\left(-\frac{u_a \Delta_a^2}{8}\right)\right). \quad (1.9)$$

It remains to plug-in. Using $\delta = \frac{1}{n^2}$ yields

$$u_a = 1 + \frac{2 \log(n^2)}{\frac{1}{4}\Delta_a^2} = 1 + \frac{16 \log(n)}{\Delta_a^2}$$

and plugging-in yields

$$\mathbb{E}[T_a(n)] \leq u_a + 1 + n n^{-2} \leq u_a + 2 \leq 3 + \frac{16 \log(n)}{\Delta_a^2}.$$

Combined with the regret decomposition the claim follows. \square

Lecture 4

The previous bound is logarithmic in n with inverse reward gaps. While the dependency in n is very good the inverse reward gaps can be arbitrarily large if the second best arm is close to the best arm. Estimating slightly differently in the final step of the proof we can derive a different upper bound which is less favorable in the dependency of n but more favorable in term of the reward gap (which of course is a priori unknown).



Theorem 1.3.9. Suppose ν is a bandit model with 1-subgaussian arms, $n \in \mathbb{N}$, and $\delta = \frac{1}{n^2}$. Then the UCB algorithms also has the alternative regret upper bound

$$R_n(\pi) \leq 8\sqrt{Kn \log(n)} + 3 \sum_{a \in \mathcal{A}} \Delta_a.$$

The term $\sum_{a \in \mathcal{A}} \Delta_a$ appearing in both bounds is very natural as every reasonable algorithm plays each arm at least once and thus, by the regret decomposition lemma, gives $\sum_{a \in \mathcal{A}} \Delta_a$. In many examples the sum of the reward gaps can be bounded. If for instance all arms are Bernoulli distributed, then the sum can be replaced by k and be neglected as it will be dominated by the first summand. More interesting is the first summand for which one can decide to emphasize more the dependence on n or the regret gap.

Proof. The proof of the regret bound given above revealed that $\mathbb{E}[T_a(n)] \leq 3 + \frac{16 \log(n)}{\Delta_a^2} s$. The idea of the proof is as follows. From the regret decomposition we know that small reward gaps should not pose problems as they appear multiplicatively. Separating the arms by a threshold into those with small and those with large reward gaps we only use the estimate for the ones with large reward gaps and then minimise over the threshold. Pursuing this way leads to

$$\begin{aligned} R_n(\pi) &= \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}[T_a(n)] \\ &= \sum_{a \in \mathcal{A}: \Delta_i < \Delta} \Delta_a \mathbb{E}[T_a(n)] + \sum_{a \in \mathcal{A}: \Delta_i \geq \Delta} \Delta_a \mathbb{E}[T_a(n)] \\ &\leq n\Delta + \sum_{a \in \mathcal{A}: \Delta_i \geq \Delta} \left(\Delta_a 3 + \frac{16 \log(n)}{\Delta_a} \right) \\ &\leq n\Delta + \frac{16K \log(n)}{\Delta} + 3 \sum_{a \in \mathcal{A}} \Delta_a. \end{aligned}$$

Since Δ can be chosen arbitrarily it suffices to minimise the righthand side as a function in Δ . The minimum can be found easily by differentiation in Δ to be located at $\Delta = \sqrt{16K \log(n)/n}$. Plugging-in yields

$$R_n(\pi) \leq 8\sqrt{Kn \log(n)} + 3 \sum_{a \in \mathcal{A}} \Delta_a.$$

□



For σ -subgaussian bandit models the UCB exploration bonus is modified as

$$\text{UCB}_a(t, \delta) := \begin{cases} \infty & : T_a(t) = 0 \\ \underbrace{\hat{Q}_a(t)}_{\text{greedy}} + \underbrace{\sqrt{\frac{2\sigma^2 \log(1/\delta)}{T_a(t)}}}_{\text{exploration bonus}} & : T_a(t) \neq 0. \end{cases}$$

Check that the regret bound in Theorem 1.3.8 with the choice $\delta = \frac{1}{n^2}$ changes to

$$R_n(\pi) \leq 3 \sum_{a \in \mathcal{A}} \Delta_a + 16\sigma^2 \log(n) \sum_{a: Q_a \neq Q_*} \frac{1}{\Delta_a},$$

and this leads to

$$R_n(\pi) \leq 8\sigma\sqrt{Kn \log(n)} + 3 \sum_{a \in \mathcal{A}} \Delta_a$$



in Theorem 1.3.9. Thus, for Bernoulli bandits the exploration bonus should be $\sqrt{\frac{\frac{1}{4} \log(n)}{T_a(t)}}$ and, as you should check in simulations (!) the constant $\frac{1}{4}$ is crucial for a good performance.

The idea of the last proof is very important. The model based regret bounds are often completely useless as they emphasise too strongly small reward gaps which by means of the regret decomposition should not be important at all. To get a better feeling please think a moment about the following exercise:



Recall (1.2), the upper bound for ETC in the case of two arms. Use the idea from the proof of Theorem 1.3.9 to derive the following upper bound of the ETC regret:

$$R_n(\pi) \leq \Delta + C\sqrt{n},$$

for some model-free constant C (for instance $C = 8 + \frac{2}{\epsilon}$) so that, in particular, $R_n(\pi) \leq 1 + C\sqrt{n}$ for all bandit models with regret bound $\Delta \leq 1$ (for instance for Bernoulli bandits).

It is very crucial to compare the constants appearing in the regret bounds. As mathematicians we tend to overestimate the importance of n and always think of $n \rightarrow \infty$ (as we did in the formulation of Theorem 1.3.7). Keeping in mind logarithmic growth one quickly realises the importance of constants. As an example $\log(1.000.000) \approx 13$ so that a constant \sqrt{K} or even worse $1/\Delta$ can be much more interesting. As an example the constants (5 and $1/\Delta$ appears even squared) in the regret bound of explore-then- ϵ greedy are extremely bad even though the logarithm in n is right order.



UCB is typically used as benchmark algorithm to compare other bandit algorithms. The reason is that UCB is simple to state, simple to analyse, and also pretty close to optimal both in the model-based and model-independent regret bounds. Authors typically compare their bounds with the growth in n (typically logarithmic model-based and square-root model-independent), the appearing generic constants (such as 8 in UCB), and how the reward bounds enter the regret upper bounds.

Comparing with the Lai-Robbins lower bounds for subgaussian bandit algorithms shows that the UCB algorithm is pretty close to optimal. There are more refined versions of the simple UCB algorithm presented above, such as the MOOS algorithm. What changes are different choices for additional exploitation. For further reading we refer to the wonderful book „Bandit Algorithms“ of Tor Lattimore and Csaba Szepesvári which is available online for free.

Memory optimised variants of the algorithms

As mentioned earlier it will be of fundamental importance to efficiently implement reinforcement learning algorithms. All simple learning strategies involving \hat{Q} are useful for a first discussion. How do we most efficiently handle the approximate action values $\hat{Q}_a(t)$? More generally, how do we most efficiently handle sums of the form $Q_n = \frac{1}{n} \sum_{k=1}^n R_k$ occurring in algorithms? On first sight it may look like it is necessary to store the entire sequence R_1, \dots to compute the values

Q_n . In fact, there is a simple patch. Rewriting

$$\begin{aligned}
Q_n &= \frac{1}{n} \sum_{k=1}^n R_k \\
&= \frac{1}{n} \left(R_n + \sum_{k=1}^{n-1} R_k \right) \\
&= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{k=1}^{n-1} R_k \right) \\
&= \frac{1}{n} \left(R_n + (n-1) Q_{n-1} \right) \\
&= Q_{n-1} + \frac{1}{n} \left(R_n - Q_{n-1} \right) \\
&= \frac{n-1}{n} Q_{n-1} + \frac{1}{n} R_n,
\end{aligned} \tag{1.10}$$

we observe that in order to compute Q_n only the values Q_n and R_n are needed. Hence, if estimated action values need to be computed in loops, it is only necessary to store the current estimated action value/reward in the memory. As an example here is a version of *UCB* that only needs to store the current reward but not the past rewards.

Algorithm 5: memory optimized UCB algorithms

Data: bandit model $\nu, \delta \in (0, 1), n$

Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n

Initialise $\hat{Q}(0) \equiv 0, N \equiv 0$;

while $t \leq n$ **do**

$A_t = \operatorname{argmax}_a \text{UCB}_a(t-1, \delta)$;

Obtain reward X_t by playing arm A_t ;

Set $N_{A_t} = N_{A_t} + 1$;

Update the action value function using $\hat{Q}_{A_t}(t) = \hat{Q}_{A_t}(t-1) + \frac{1}{N_{A_t}}(X_t - \hat{Q}_{A_t}(t-1))$;

end

1.3.3 Boltzmann exploration

In this section we present a method that connects greedy exploration and the UCB algorithm in a surprising way. Before explaining the algorithm the concept of softmax distributions is needed.



Definition 1.3.10. Suppose $x, \theta \in \mathbb{R}^d$, then x defines a discrete distribution $\text{SM}(\theta, x)$ on $\{1, \dots, d\}$ in the following way:

$$p_i := \frac{e^{\theta_i x_i}}{\sum_{k=1}^d e^{\theta_k x_k}}, \quad i = 1, \dots, d.$$

The weights are called Boltzmann weights of the vector x .

The wording comes from physics, where θ is called inverse temperature. The softmax distribution is a so-called categorical distribution, written $\text{Categorical}(p_1, \dots, p_d)$ which is also called a multinoulli or generalised Bernoulli distribution with probabilities p_1, \dots, p_d . For us, the following idea is much more important. Sampling from $\text{SM}(\theta, x)$ is somewhat similar to the deterministic distribution M_x that only charges mass on the index $\operatorname{argmax}_k x_k$ because $\text{SM}(\theta, x)$ assigns the highest probabilities to the coordinate with largest value x_k . But unlike the deterministic maximum distribution the softmax distribution only weights stronger the maximal element than the smaller elements. The

role of θ is important: for large θ the softmax resembles the argmax distribution while for small θ the distribution resembles the uniform distribution on $\{1, \dots, d\}$.

Replacing sampling from the (deterministic) argmax distribution in the greedy algorithm yields the Boltzmann exploration algorithm. In contrast to the greedy algorithm there is continuous

Algorithm 6: Simple Boltzmann exploration

Data: bandit model ν , initialisation $\hat{Q}(0)$, parameter θ

Result: actions A_1, \dots, A_n and rewards X_1, \dots, X_n

while $t \leq n$ **do**

 Sample A_t from $\text{SM}(\theta, \hat{Q}(t-1))$;

 Obtain reward X_t by playing arm A_t ;

 Update the estimated action value function $\hat{Q}(t)$;

end

exploration as all arms have positive probabilities. It is quite obvious that the choice of θ is crucial as the algorithm resembles two unfavorable algorithms with linear regret, both for small and large θ (uniform and greedy exploration). In fact², both constant and decreasing choices of θ are unfavorable. To guess a better choice for θ we need the following surprising lemma:



Lemma 1.3.11. Let $x, \theta \in \mathbb{R}^d$, then

$$\text{SM}(\theta, x) \stackrel{(d)}{=} \text{argmax}_k \{\theta_k x_k + g_k\},$$

where g_1, \dots, g_k are iid Gumbel random variables with scale 1 and mode 0, i.e. have probability density function $f(x) = e^{-(x+e^{-x})}$ or CDF $F(t) = e^{-e^{-t}}$.

Proof. Reformulated right the proof is quite simple. We first prove very well-known statement from elementary probability theory on exponential random variables. Suppose E_1, E_2 are independent exponential random variables with parameters λ_1, λ_2 . Then $\mathbb{P}(E_1 \leq E_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}$. This is a simple integration exercise: with $A = \{0 \leq x_1 \leq x_2\} \in \mathcal{B}(\mathbb{R}^2)$ the computation rules for vectors of independent random variables yields

$$\begin{aligned} \mathbb{P}(E_1 \leq E_2) &= \mathbb{E}[\mathbf{1}_A(E_1, E_2)] \\ &= \int_A f_{(E_1, E_2)}(x_1, x_2) \, d(x_1, x_2) \\ &= \int_0^\infty \int_0^{x_2} \lambda_1 e^{-\lambda_1 x_1} \lambda_2 e^{-\lambda_2 x_2} \, dx_1 \, dx_2 \\ &= \lambda_1 \lambda_2 \int_0^\infty e^{-\lambda_2 x_2} \frac{1}{\lambda_1} (1 - e^{-\lambda_1 x_2}) \, dx_2 \\ &= \lambda_2 \int_0^\infty (e^{-\lambda_2 x_2} - e^{-(\lambda_1 + \lambda_2)x_2}) \, dx_2 \\ &= 1 - \frac{\lambda_2}{\lambda_1 + \lambda_2} = \frac{\lambda_1}{\lambda_1 + \lambda_2}. \end{aligned}$$

Next, we can compute the argmin distribution of independent exponential variables. Suppose E_1, \dots, E_n are independent exponential random variables with parameters $\lambda_1, \dots, \lambda_n$, then $\mathbb{P}(\text{argmin}_{i \leq n} E_i = i) = \frac{\lambda_i}{\lambda_i + \sum_{k \neq i} \lambda_k}$. The identity is a direct consequence from the above:

$$\mathbb{P}(\text{argmin}_{i \leq n} E_i = i) = \mathbb{P}(E_i \leq E_k, \forall k \neq i) = \mathbb{P}(E_i \leq E) = \frac{\lambda_i}{\lambda_i + \sum_{k \neq i} \lambda_k},$$

²N. Cesa-Bianchi, C. Gentile, G. Lugosi, G. Neu: „Boltzmann exploration done right“, NeuRIPS, (2017)

where $E := \min_{k \neq i} E_k$ is independent of E_i and exponentially distributed with parameter $\sum_{k \neq i} \lambda_k$. Here recall from elementary probability that the minimum of independent exponentials is again exponential and the parameter is the sum of the parameters. The second ingredient is a connection between exponential and Gumbel variables. If $X \sim \text{Exp}(1)$, then $Z := -\log(X)$ is Gumbel with scale parameter $\beta = 1$ and mode $\mu = 0$. The claim is checked by computing the cumulative distribution function:

$$\mathbb{P}(Z \leq t) = \mathbb{P}(X \geq e^{-t}) = e^{-e^{-t}}, \quad t \in \mathbb{R}.$$

As a consequence, with g Gumbel with scale $\beta = 1$ and mode $\mu = 0$ and $E_\lambda \sim \text{Exp}(\lambda)$, we obtain the identity in law

$$c + g \sim -\log(e^{-c}) - \log(E_1) = -\log(e^{-c}E_1) \sim -\log(E_{e^c}).$$

For the last equality in law we have also used the scaling property of the exponential distribution. If g_1, \dots, g_k are iid Gumbel with scale $\beta = 1$ and mode $\mu = 0$ we finally obtain

$$\begin{aligned} \mathbb{P}(\operatorname{argmax}_{k \leq n} (\theta_k x_k + g_k) = i) &= \mathbb{P}(\operatorname{argmax}_{k \leq n} -\log(E_{e^{\theta_k x_k}}) = i) \\ &= \mathbb{P}(\operatorname{argmin}_{k \leq n} E_{e^{\theta_k x_k}} = i) \\ &= \frac{e^{\theta_i x_i}}{\sum_k e^{\theta_k x_k}}. \end{aligned}$$

In other words, the Gumbel argmax is distributed according to $\text{SM}(\theta, x)$. \square

Using the lemma the Boltzmann algorithm can now be interpreted differently: arms are chosen according to the distribution

$$A_t \sim \operatorname{argmax}_a \{ \theta \hat{Q}_a + g_a \} = \operatorname{argmax}_a \{ \hat{Q}_a + \theta^{-1} g_a \},$$

where the g_a are iid Gumbel and independent of \hat{Q} . Does this look familiar? Of course, this is the greedy strategy with an additional random exploration bonus as we have seen in the UCB algorithm. Motivated by the UCB algorithm a first idea should immediately come to mind: θ should be arm-dependent and $\sqrt{\frac{C}{T_a}}$ might be a good idea. In fact, that's true as was shown in the article of Cesa-Bianchi et al. We will not go into detail here. The aim of this section was to link the Boltzmann exploration with greedy-type algorithms. Here is an interesting research question to think about. Let us forget that UCB-type exploration with Gumbel random exploration bonus is linked to the rather natural exploitation strategy given by Boltzmann softmax weights. It is then very natural to replace the Gumbel distribution by some other distribution. It seems most plausible that non-negative distributions should be more reasonable as a negative value turns the exploration bonus into an exploration malus which was never intended.



Use the simulation code provided for the exercises to play with different distributions and see if replacing the Gumbel distribution can be favorable. Choosing distributions that concentrate around 1 (Dirac measure at 1 gives UCB) might be reasonable, such as a Gamma distribution with shaper parameter larger than 1 and scale parameter that forces expectation 1.

1.3.4 Simple policy gradient for stochastic bandits

We now come to a completely different approach, the so-called policy gradient approach. The approach presented here is not really part of the research field of multiarmed bandits but is a very simple special case of which will later be called policy gradient method for non-tabular reinforcement learning. We use the softmax policy gradient method to connect the topic of multiarmed bandits with reinforcement learning that will be started in the next lecture. For that sake, here is a new way of thinking of bandits. The bandit game is a one-step game, the game of choosing one of the K arms and obtaining the corresponding outcome.



Definition 1.3.12. A probability distribution π on \mathcal{A} is called a policy for the bandit game, the expected reward $V(\pi) := Q_\pi := \sum_{a \in \mathcal{A}} Q_a \pi(a)$ when playing the policy is called the value of the policy.

Keep in mind that since \mathcal{A} is assumed to be finite a distribution on \mathcal{A} is nothing but a probability vector, a vector of non-negative numbers that sums up to 1. A vector $(0, \dots, 1, \dots, 0)$ with 1 for the a th arm (position) corresponds to playing only arm a and in this case $Q_\pi = Q_a$. A policy is optimal if the expected outcome Q_π is maximal, i.e. equals Q_* . If there are several optimal arms, i.e. arms satisfying $Q_a = Q_*$, then any policy is optimal that has mass only on optimal arms. Still, choosing any of the optimal arms with probability 1 is always optimal.



The goal in reinforcement learning is similar to that of multiarmed bandits, but different. In reinforcement learning we are typically concerned with identifying the best policy (here: in the bandit game the best arm) as quickly as possible, not in minimising the regret of the learning strategy for fixed n . Essentially, we do not care if very bad arms are played as long as the best arm is found quickly.

One of the reasons for this different point of view is the field of applications. While one of the main motivations for the multiarmed bandit stems from medicine where every single life counts a major field of applications that pushed the development of reinforcement is automated learning of optimal strategies in gaming or online advertisement where obviously the impact of suboptimal attempts is much less severe.

In the much more general context of Markov decision processes we will get interested in so-called policy gradient methods.



Definition 1.3.13. If $\Theta \subseteq \mathbb{R}^d$, then a set $\{\pi_\theta : \theta \in \Theta\}$ of probability distributions on \mathcal{A} is called a parametrised family of policies.

The policy gradient idea is as follows: given a parametrised policy over a continuous index set we aim to maximise the value function J over the parameter set:

$$\theta \mapsto J(\theta) := Q_{\pi_\theta} = \sum_{a \in \mathcal{A}} \pi_\theta(a) Q_a.$$

The value function J is nothing but a multidimensional function, hence, maximisation algorithms can be used. If the parametric family can approximate Dirac-measures δ_a then one could hope to identify the optimal arm using an optimisation procedure. So how can we find the optimal arm with policy gradient? By typical optimization methods from lectures on numerical analysis. One example is the classical gradient ascent method:

$$\theta_{n+1} := \theta_n + \alpha \nabla J(\theta_n), \quad n \in \mathbb{N}.$$

Under suitable assumptions on J (such as convexity) that algorithm converges to a maximum θ_* of J . Unfortunately, that approach has diverse difficulties which are topics of ongoing research:

1. Given a bandit model, what is a good parametric family of policies?
2. If J is unknown (because the expectations Q_a are unknown) how can gradient descent be carried out (most efficiently)?
3. Even if the function J would be known explicitly, will the gradient ascent algorithm converge to an optimal policy? How fast?

In this first section on the policy gradient method we will address the first two issues by a discussion of one particular parametrised family and a sketch of what later will be discussed in details in the chapter on the policy gradient method. The third question is widely open with

some recent progress³. We will start with the second issue on how to deal with the gradient of J if J is unknown but samples can be generated. The trick that will occur a few times in this lecture course is the so-called log-trick, here in its simplest form:



The following short computation is typically called log-trick (or score-function trick):

$$\begin{aligned}\nabla J(\theta) &= \nabla \sum_{a \in \mathcal{A}} Q_a \pi_\theta(a) \\ &= \sum_{a \in \mathcal{A}} Q_a \nabla \pi_\theta(a) \frac{\pi_\theta(a)}{\pi_\theta(a)} \\ &= \sum_{a \in \mathcal{A}} Q_a \nabla \log(\pi_\theta(a)) \pi_\theta(a) \\ &= \mathbb{E}_{\pi_\theta} [Q_A \nabla \log(\pi_\theta(A))] \\ &= \mathbb{E}_{\pi_\theta} [X_A \nabla \log(\pi_\theta(A))],\end{aligned}$$

where the last equality follows from the tower-property and $\mathbb{E}[X_A|A] = Q_A$ for instance by using $X_t = \sum_{a \in \mathcal{A}} X_t^{(a)} \mathbf{1}_{A_t=a}$ the random table model from the proof of Theorem 1.2.4. Note that whenever we take the expectation of a vector of random variables (the gradient is a vector) the expectation is taken coordinate wise.

Now that the gradient is expressed as the expectation of a random vector the idea is to replace in the gradient ascent algorithm the true gradient by samples of the underlying random variable. Either by one sample

$$\tilde{\theta}_{n+1} := \tilde{\theta}_n + \alpha X_{A_n} \nabla \log(\pi_{\tilde{\theta}_n}(A_n)), \quad n \in \mathbb{N}, \quad (1.11)$$

or by a so-called batch of independent samples:

$$\tilde{\theta}_{n+1} := \tilde{\theta}_n + \alpha \frac{1}{N} \sum_{i=1}^N X_{A_n^i}^i \nabla \log(\pi_{\tilde{\theta}_n}(A_n^i)), \quad n \in \mathbb{N}, \quad (1.12)$$

where the A_n^i are independently sampled according to $\tilde{\pi}_{\tilde{\theta}_n}$ and the $X_{A_n^i}^i$ are independent samples from the arms A_n^i .



Definition 1.3.14. The appearing function $a \mapsto \nabla \log \pi_\theta(a)$ is called the score-function of π_θ .

Of course the algorithm is most useful if the score-function is nice so that the gradient disappears. Here is the most prominent example:

Example 1.3.15. Suppose $d = |\mathcal{A}|$, so that there is a parameter θ_a for each arm a . Furthermore, define

$$\pi_\theta(a) = \frac{e^{\tau \theta_a}}{\sum_{k \in \mathcal{A}} e^{\tau \theta_k}}$$

for some fixed parameter τ . Then the policies $\{\pi_\theta : \theta \in \mathbb{R}^d\}$ are called the softmax family. If the sequence of policies converges to a limiting policy that only plays one arm, say arm a , the corresponding parameter θ_a must converge to infinity. This already indicates that the convergence might be slow (indeed, the convergence is slow!). The score-function is easily computed to be

$$\left(\nabla \log \pi_\theta(a)\right)_i = \tau \mathbf{1}_{a=i} - \left(\nabla \log \left(\sum_{a \in \mathcal{A}} e^{\tau \theta_a}\right)\right)_i = \tau \mathbf{1}_{a=i} - \frac{\tau e^{\tau \theta_i}}{\sum_{a \in \mathcal{A}} e^{\tau \theta_a}} = \tau (\mathbf{1}_{a=i} - \pi_\theta(i)).$$

³see for instance A. Agarwal, S. Kakade, J. Lee, G. Mahajan: „On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift“, JMLR, 1-76, (2021)

Plugging-into (1.11) yields the updates (now written component wise)

$$\begin{aligned}\theta_{1,n+1} &= \theta_{1,n} - \alpha\tau X_i\pi_\theta(1), \\ &\dots = \dots \\ \theta_{i,n+1} &= \theta_{i,n} + \alpha\tau X_i(1 - \pi_\theta(i)), \\ &\dots = \dots \\ \theta_{d,n+1} &= \theta_{d,n} - \alpha\tau X_i\pi_\theta(d)\end{aligned}$$

if $N = 1$, $A_n = i$, and X_i is a sample of arm i . Here is the first explanation why reinforcement learning is called reinforcement learning. Remember that the θ_a are indirect parametrisations of the probabilities to play arm a . If at time n the policy current policy decides to play arm i then the probabilities are reinforced as follows. If the reward obtained by playing arm i is positive, let's interpret the positive reward as positive feedback, then the probability to play arm i is increased and all other probabilities are decreased. Similarly, if the feedback from playing arm i is negative, then the probability to do so in the next round is decreased and all other probabilities are increased.

Here is a big problem that becomes immediately visible in the softmax policy gradient algorithm. Suppose for instance that the starting vector is $\theta_0 \equiv 0$ and all arms mostly return positive values (for instance Bernoulli bandits). Then the first arm is chosen uniformly as π_{θ_0} is uniform. If the first chosen arm, say a , yields a positive return then the the updated parameter vector θ_1 only has one positive entry, namely the a th entry. Hence, for the second update the a th arm is most likely to be chosen again. As a consequence, the committal behavior seen for the simple greedy is expected to occur again. It does occur! Now there is a trick that can improve the situation a lot, called baseline:



Here is another representation for the gradient:

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}[(X_A - b)\nabla \log(\pi_\theta(A))],$$

where b is any constant. The reason is simply that, using the same computation as above,

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[b\nabla \log(\pi_\theta(A))] &= b \sum_{a \in \mathcal{A}} \nabla \log(\pi_\theta(a))\pi_\theta(a) \\ &= b \sum_{a \in \mathcal{A}} \nabla \pi_\theta(a) \frac{\pi_\theta(a)}{\pi_\theta(a)} \\ &= b\nabla \sum_{a \in \mathcal{A}} \pi_\theta(a) \\ &= b\nabla 1 = 0.\end{aligned}$$

It is important to realise that b can be chosen arbitrary without changing the gradient and as such not changing the gradient ascent algorithm

$$\theta_{n+1} := \theta_n + \alpha\nabla J(\theta), \quad n \in \mathbb{N}.$$

Still, it drastically changes the stochastic version (here for the softmax parametrisation)

$$\tilde{\theta}_{i,n+1} := \tilde{\theta}_{i,n} + \alpha(X_{A_n} - b)\tau(\mathbf{1}_{A_n=i} - \pi_{\tilde{\theta}_n}(i)), \quad n \in \mathbb{N},$$

as now the effect of committal behavior can be reduced drastically. An important question is what baseline to chose so that the algorithm is speed up the most. In fact, that question is widely open. Usually people chose b to minimise the variance of the estimator of ∇J , but on the pathwise level this choice is not optimal.

Even though not optimal on a pathwise level it can be instructive to compute the minimum variance baseline for every step of the policy gradient scheme with baseline:



The variance of a random vector X is defined by to be $\mathbb{V}[X] = \mathbb{E}[\|X^2\|] - \mathbb{E}[X]^T \mathbb{E}[X]$. Show by differentiation that

$$b_* = \frac{\mathbb{E}_{\pi_\theta}[X_A \|\nabla \log \pi_\theta(A)\|_2^2]}{\mathbb{E}_{\pi_\theta}[\|\nabla \log \pi_\theta(A)\|_2^2]}$$

is the baseline that minimises the variance of the unbiased estimators

$$(X_A - b) \nabla \log(\pi_\theta(A)), \quad A \sim \pi_\theta,$$

of $J(\theta)$.

From the point of view of producing good estimates of the gradient this baseline is to be preferred. Unfortunately, from the point of view of fast convergence the choice is suboptimal⁴.

To finish the discussion we might wish to estimate the regret $R_n(\pi)$ of a learning strategy $(\pi_{\theta_n})_{n \geq 1}$. In fact, almost nothing is known, only some rough estimates for the softmax parametrisation can be found in the literature. Give it a try!

1.4 Minimax lower bounds for stochastic bandits

kommt noch

⁴W. Chung, V. Thomas, M. Machado, N. Le Roux: „Beyond Variance Reduction: Understanding the True Impact of Baselines on Policy Optimization“, ICML, (2021)

Chapter 2

More bandits

The assumption of stationarity (same bandits at every time) and independence can be relaxed significantly but still allows to write down (and analyse) bandit algorithms. For completeness two more sections on adversarial and contextual bandits will be added but won't be part of this lecture course for time constraints.

2.1 Adversarial bandits

eine Vorlesung, naechstes Jahr

2.2 Contextual bandits

eine Vorlesung, naechstes Jahr

Part II

Tabular Reinforcement Learning

Chapter 3

Basics: Finite MDPs and Dynamic Programming Methods

Lecture 6

3.1 Markov decision processes

After the first introductory chapter on bandits we will now turn towards the main topic of this course: optimal decision making in which decisions also influence the system (in contrast to stochastic bandits). This basic chapter covers the following topics:

- Introduce the basic setup for decision making in complex systems under uncertainty, we will use so-called Markov decision processes (MDP).
- Understand optimal decision policies and their relations to Bellman optimality and expectation equations.
- Understand how to turn the theoretical results into algorithms, so-called value and policy iteration algorithms.

All topics covered here are old and standard, they can be found in the comprehensive overview of Putterman¹. One should only keep in mind that the appearing Q -functions are not popular in stochastic optimal control, Q -functions are much more relevant to the reinforcement learning approaches developed in the next chapters.

3.1.1 A quick dive into Markov chains

Before starting with Markov decision processes let us briefly recall some facts on Markov chains. A finite-state Markov chain is a discrete-time stochastic process $(S_t)_{t \in \mathbb{N}_0}$ with values in some finite set \mathcal{S} on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ that satisfies the Markov property:

$$\mathbb{P}(S_{t+1} = s_{t+1} | S_0 = s_0, \dots, S_t = s_t) = \mathbb{P}(S_{t+1} = s_{t+1} | S_t = s_t)$$

for all $t \geq 0$ and $s_0, \dots, s_{t+1} \in \mathcal{S}$. In words, transitions from a state to another do not depend on the past. The most important special case is that of a time-homogeneous Markov chain for which transition probabilities do not change over time. Time-homogeneous Markov chains are closely related to stochastic matrices (non-negative elements, all rows sum to 1). Given an initial distribution μ on \mathcal{S} and a stochastic $|\mathcal{S}| \times |\mathcal{S}|$ -matrix, a Markov chain on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with initial distribution μ and transition matrix P is uniquely determined through the basic path probabilities

$$\mathbb{P}(X_0 = s_0, \dots, X_n = s_n) = \mu(\{s_0\}) p_{s_0, s_1} \cdot \dots \cdot p_{s_{n-1}, s_n}. \quad (3.1)$$

¹M. Putterman: „Markov decision processes: discrete stochastic dynamic programming“, Wiley

Alternatively, and this is how Markov chains are generalised to more than finitely many states, the transition matrix can be interpreted as a Markov kernel on $\mathcal{S} \times \mathcal{S}$, that is, a family of probability measures $P(\cdot, s)$ for all $s \in \mathcal{S}$. Then the path probabilities can be written as

$$\mathbb{P}(X_0 = s_0, \dots, X_n = s_n) = \mu(\{s_0\})P(\{s_1\}, s_0) \cdot \dots \cdot P(\{s_n\}, s_{n-1}).$$

Many probabilities can be computed from the basic path formula, for instance

$$\mathbb{P}(S_{t+1} = s_{t+1}, \dots, S_{t+k} = s_{t+k} \mid S_t = s_t) = p_{s_t, s_{t+1}} \cdot \dots \cdot p_{s_{t+k-1}, s_{t+k}}.$$

The computation tricks are always the same. Spell out the conditional probability, write the events of interest as disjoint union of simple paths, plug-in the path probability formula, and finally cancel from the appearing products. We always imagine a Markov chain to jump on a graph of states connected by arrows carrying the transition probabilities from states s to s' .² Another way of expressing the Markov property is as follows: If $\mathbb{P}(S_t = k) > 0$ and $\tilde{\mathbb{P}} := \mathbb{P}(\cdot \mid S_t = k)$, then on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ the shifted process $(\tilde{S}_t)_{t \geq 0} := (S_{t+k})_{t \geq 0}$ is again a Markov chain with transition matrix P but started from k . To compute with the path probabilities please check the previous claim:



Check that \tilde{S} indeed is a Markov chain on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ with transition matrix P .

A Markov process can be seen as a process that we can observe from outside, for instance a game that changes states over time. The concept of a Markov reward process is less well-known. A Markov reward process is a Markov chain with an addition coordinate $(R_t)_{t \in \mathbb{N}_0}$ such that $R_0 = 0$ almost surely and $R_{t+1} \sim h(\cdot; S_t)$, where h is a Markov kernel on $\mathbb{R} \times \mathcal{S}$. This means that the reward random variable R_{t+1} is a random variable with dependence on the last state S_t of the Markov chain. If we think of a Markov process as describing a game than the rewards might be direct consequences of the rules such as $R_{t+1} = 1$ if the player that we observe has scored a goal.

3.1.2 Markov decision processes

The situation of Markov decision processes (MDPs) is slightly more complicated. For MDPS we do not observe a game but take the role of a participant. We observe the Markov chain describing the match but can influence the transitions by taking actions. As an example, by substituting as a coach an additional attack player we increase the probability of players scoring goals (positive reward) but decrease the probabilities of players winning duels which leads to larger probabilities to receive a goal (negative reward). The target will be to find a strategy (called policy) that maximises the expected reward of the game. Finding such a strategy, this is what reinforcement learning is all about!

Unfortunately, the formal notion in Markov decision processes is more sophisticated. The definition is kept quite general to emphasise that everything we will later prove in the discrete setting could be kept much more general replacing vectors and matrices by Markov kernels. For this course the intention is to keep the level of sophistication high enough to observe most difficulties but stay simple enough to not disturb too much the understanding of concepts. We will write down definitions and the existence theorem in a more general setting but then work only with finite Markov decision processes which makes our lives much easier without losing most features of interest.



Definition 3.1.1. (Markov decision model)

A Markov decision model is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ consisting of the following ingredients:

- (i) $(\mathcal{S}, \bar{\mathcal{S}})$ is a measurable space, called the state space,
- (ii) For every $s \in \mathcal{S}$, $(\mathcal{A}_s, \bar{\mathcal{A}}_s)$ is a measurable space, called the action space of

²drawing!



state s . The entire action space is defined to be $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$, \mathcal{A} contains all actions and is equipped with the σ -algebra $\bar{\mathcal{A}} = \sigma(\bigcup_{s \in \mathcal{S}} \bar{\mathcal{A}}_s)$.

(iii) A measurable set $\mathcal{R} \subseteq \mathbb{R}$ of rewards with $0 \in \mathcal{R}$, it's restricted Borel- σ -algebra denoted by $\bar{\mathcal{R}}$.

(iv) A function

$$p : \bar{\mathcal{S}} \otimes \bar{\mathcal{R}} \times (\mathcal{S} \times \mathcal{A}) \rightarrow [0, 1], (B, (s, a)) \mapsto p(B; s, a)$$

is called transition/reward-function if p is a Markov kernel on $\bar{\mathcal{S}} \otimes \bar{\mathcal{R}} \times (\mathcal{S} \times \mathcal{A})$, i.e.

- $(s, a) \mapsto p(B; a, s)$ is $(\bar{\mathcal{A}} \otimes \bar{\mathcal{S}})$ - $\bar{\mathcal{R}}$ -measurable for all B ,
- $B \mapsto p(B; a, s)$ is a probability measure on $\bar{\mathcal{S}} \otimes \bar{\mathcal{R}}$ for all s, a .

A Markov decision model is called discrete if $\mathcal{S}, \mathcal{A}, \mathcal{R}$ are finite or countably infinite and the σ -algebras are chosen to be the corresponding power sets.

No worries if measure theory is something that makes you want to cry. Once we go into the algorithmic theory of reinforcement learning we will mostly assume the model to be discrete so that all appearing functions are measurable and measures are nothing but vectors of non-negative numbers that sum to 1.



The key ingredient of the definition is the kernel p with the following interpretation. If the system is currently in state s and action a is played, then $p(\cdot; s, a)$ is the joint distribution of the next state s' and the reward r obtained. If all sets are discrete then $p(\{(s', r)\}; s, a)$ is the probability to obtain reward r and go to state s' if action a was taken in state s . According to the definition the reward can depend on the current state/action pair and the next state, but in most examples the reward and the next state will be independent (see Example 3.1.5 below).

In most books you will find the notion $p(\cdot | s, a)$. In this course the notion „|“ will be used exclusively for conditional probabilities. The function p defines the dynamics of the Markov reward model. In order to arrive at a stochastic process we additionally have to define a of protocol of how actions are chosen. These protocols are formalized by so called policies.



Definition 3.1.2. (Policy)

For a Markov decision model $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ a policy is

- an initial Markov kernel π_0 on $\bar{\mathcal{A}} \times \mathcal{S}$,
- a sequence of probability kernels $\pi = (\pi_t)_{t \in \mathbb{N}}$ on $\bar{\mathcal{A}} \times ((\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S})$ such that

$$\pi_t(\mathcal{A}_s; s_0, a_0, \dots, s_{t-1}, a_{t-1}, s) = 1 \tag{3.2}$$

for all $(s_0, a_0, \dots, s_{t-1}, a_{t-1}, s) \in (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S}$.

The set of all policies is denoted by Π .

A policy governs the choices of actions given the current state and all previous states-action pairs (not the rewards) seen before. Condition (3.2) means that only allowed actions from \mathcal{A}_s can be played in state s . Sometimes all \mathcal{A}_s are identical but in most examples they are not. As an example, playing a game like chess the allowed actions clearly depend strongly on the state of the game.



From now on we assume the Markov decision models are discrete and all σ -algebras are powersets. In particular, all measures are discrete and need to be defined only for singleton sets. The brackets for singleton sets will often be omitted.

With the definition of a policy we can now define a stochastic process on $\mathcal{S} \times \mathcal{R} \times \mathcal{A}$ whose dynamics are entirely defined by the function p and policy π (plus an initial probability distribution over which state the stochastic process will start in). Let us formalize this stochastic process and prove its existence.



Theorem 3.1.3. (Existence of (discrete) MDPs)

Let $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ be a (discrete) Markov decision model, π a policy, μ a probability measure on \mathcal{S} . Then there exists a probability space $(\Omega, \mathcal{F}, \mathbb{P}_\mu^\pi)$ carrying a stochastic process $(S_t, A_t, R_t)_{t \in \mathbb{N}_0}$ with values in $\mathcal{S} \times \mathcal{R} \times \mathcal{A}$ on the probability space $(\Omega, \mathcal{F}, \mathbb{P}_\mu^\pi)$ such that, for all $t \in \mathbb{N}_0$,

$$\begin{aligned} \mathbb{P}_\mu^\pi(S_0 \in B) &= \mu(B) \quad \text{and} \quad \mathbb{P}_\mu^\pi(R_0 = 0) = 1, \\ \mathbb{P}_\mu^\pi(A_t \in C \mid S_0 = s_0, A_0 = a_0, \dots, S_t = s_t) &= \pi_t(C; s_0, a_0, \dots, s_t), \\ \mathbb{P}_\mu^\pi(S_{t+1} \in B, R_{t+1} \in D \mid S_t = s, A_t = a) &= p(B \times D; s, a), \end{aligned}$$

for $B \subseteq \mathcal{S}$, $C \subseteq \mathcal{A}$, and $D \subseteq \mathcal{R}$.

In words the mechanism goes as follows. In a state s an action is sampled according to the policy which is allowed to refer to the entire past of states and actions (not to the rewards!). The current and past state-action pairs (s, a) are used to sample the next state s' and the reward. Note: The reward is allowed to depend both on the current state-action pair (s, a) and the future state s' ! Typically, the reward will only depend on (s, a) , not on s' , but some examples require this greater generality.

Proof. We only give the proof in the discrete setting to save quite a bit of time. The proof is essentially identical to the existence proof for Markov chains.



Measure for finite time-horizon $T < \infty$.

The probability space is written down explicitly as the set of trajectories, the σ -algebra is chosen to be the power set, a canonical measure is defined for every element of the probability space by an explicit definition, and the process using the identity mapping. Define the measurable space $(\Omega_T, \mathcal{F}_T)$ by $\Omega := (\mathcal{S} \times \mathcal{R} \times \mathcal{A})^T$, the trajectories of length T , i.e.

$$\omega = (s_0, r_0, a_0, \dots, s_T, r_T, a_T),$$

and \mathcal{F}_T as the powerset. As for Markov chains the probabilities for paths are written down explicitly:

$$\begin{aligned} &\mathbb{P}_T(\{(s_0, r_0, a_0, \dots, s_T, r_T, a_T)\}) \\ &:= \mu(\{s_0\}) \cdot \delta_0(r_0) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i). \end{aligned}$$

In words: An initial state is chosen by μ , the first action is sampled using π_0 , and the first reward is set to 0. For every further time-step the new state s_i and the reward r_i are determined by p , an action a_i is chosen according to π_i .



Show that defining \mathbb{P}_T on the singletons as above yields a probability measure.

The exercise is important to get a feeling for the path probabilities. You will have to sum over all possible paths, i.e. over $\sum_{a_0, r_0, s_0} \dots \sum_{a_T, r_T, s_T}$, then pull out the factors that are independent of

the summands to simplify backwards using the kernel property to obtain (here for the summands corresponding to T)

$$\begin{aligned} & \sum_{a_T, r_T, s_T} p(\{(s_T, r_T)\}; s_{T-1}, a_{T-1}) \cdot \pi_T(\{a_T\}; s_0, a_0, \dots, a_{T-1}, s_T) \\ &= \sum_{r_T, s_T} p(\{(s_T, r_T)\}; s_{T-1}, a_{T-1}) \sum_{a_T} \pi_T(\{a_T\}; s_0, a_0, \dots, a_{T-1}, s_T) = 1. \end{aligned}$$

Summing over all trajectories shows easily that \mathbb{P}_T is a probability measure on the paths.



Measure for infinite time-horizon.

The same construction cannot work for $T = \infty$ as the infinite products would be zero. Instead, Kolmogorov's extension theorem is employed. Define the measurable space (Ω, \mathcal{F}) by $\Omega := (\mathcal{S} \times \mathcal{R} \times \mathcal{A})^\infty$, the trajectories of infinite length, and the corresponding σ -algebra $\mathcal{F} := (\bar{\mathcal{S}} \otimes \bar{\mathcal{R}} \otimes \bar{\mathcal{A}})^{\otimes \infty}$, the cylinder σ -algebra. The elements of Ω can be written as infinite sequences of the form

$$\omega = (s_0, r_0, a_0, \dots).$$

Consider the projections $\pi_T^{T+1} : \Omega^{T+1} \rightarrow \Omega^T, \omega \mapsto \omega|_T$ and $\pi_T : \Omega \rightarrow \Omega^T, \omega \mapsto \omega|_T$. The projections simply remove the triple corresponding to time $T + 1$ from the sequence. We now show the consistency property $\mathbb{P}_T = \mathbb{P}_{T+1} \circ (\pi_T^{T+1})^{-1}$ for any $T \in \mathbb{N}$. If the consistency can be proved, then Kolmogorov's extension theorem gives a unique probability measure \mathbb{P} on (Ω, \mathcal{F}) such that $\mathbb{P}_T = \mathbb{P} \circ (\pi_T)^{-1}$. The consistency property simply follows from the product structure in the measures \mathbb{P}_T defined above. Since the measures are discrete the equality can be checked on all elements separately:

$$\begin{aligned} & \mathbb{P}_{T+1}((\pi_T^{T+1})^{-1}(\{s_0, r_0, a_0, \dots, s_T, r_T, a_T\})) \\ &= \mathbb{P}_{T+1}(\cup_{s \in \mathcal{S}} \cup_{a \in \mathcal{A}} \cup_{r \in \mathcal{R}} \{(s_0, r_0, a_0, \dots, s_T, r_T, a_T, s, a, r)\}) \\ &= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}} \mathbb{P}_{T+1}(\{(s_0, r_0, a_0, \dots, s_T, r_T, a_T, s, a, r)\}) \\ &\stackrel{\text{linearity}}{=} \mu(\{s_0\}) \cdot \delta_0(r_0) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i) \\ &\quad \times \underbrace{\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(\{s\}, \{r\}; s_T, a_T)}_{=1} \cdot \underbrace{\sum_{a \in \mathcal{A}} \pi_{T+1}(\{a\}; s_0, a_0, \dots, a_T, s_{T+1})}_{=1} \\ &= \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_{T+1}(\{a_i\}; s_0, a_0, \dots, a_T, s_{T+1}) \\ &= \mathbb{P}_T(\{(s_0, r_0, a_0, \dots, s_T, r_T, a_T\})). \end{aligned}$$

Kolmogorov's extension theorem now yields a unique measure \mathbb{P} extending all \mathbb{P}_T . We set $\mathbb{P}_\mu^\pi := \mathbb{P}$ to indicate the dependency of μ and π .



Canonical construction $(\Omega, \mathcal{F}, \mathbb{P}_\mu^\pi, (S, A, R)_{t \in \mathbb{N}_0})$.

The measurable space (Ω, \mathcal{F}) has been defined above. On Ω we define $(S_t, A_t, R_t)(\omega) = (s_t, a_t, r_t)$ if ω takes the form (s_0, a_0, r_0, \dots) .



The properties claimed in the theorem hold.

The first two claims follow directly from the definition of the measure \mathbb{P}_μ^π and (S_0, A_0, R_0) . We check the claim for A and leave the claims for A and R as an exercise. Using the extension

property of \mathbb{P}_μ^π (there is no dependence later than t) yields

$$\begin{aligned} & \mathbb{P}_\mu^\pi(A_t \in C, S_0 = s_0, A_0 = a_0, \dots, A_{t-1} = a_{t-1}, S_t = s_t) \\ &= \mathbb{P}_t(S_0 = s_0, A_0 = a_0, \dots, A_{t-1} = a_{t-1}, S_t = s_t, A_t \in C, R_t \in \mathcal{R}) \\ &= \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^{t-1} p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i) \\ & \quad \times \sum_{r \in \mathcal{R}} p(\{(s_t, r)\}; s_{t-1}, a_{t-1}) \cdot \sum_{a \in C} \pi_t(\{a\}; s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t) \end{aligned}$$

and

$$\begin{aligned} & \mathbb{P}_\mu^\pi(S_0 = s_0, A_0 = a_0, \dots, A_{t-1} = a_{t-1}, S_t = s_t) \\ &= \mathbb{P}_t(S_0 = s_0, A_0 = a_0, \dots, A_{t-1} = a_{t-1}, S_t = s_t, A_t \in \mathcal{A}, R_t \in \mathcal{R}) \\ &= \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^{t-1} p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i) \\ & \quad \times \sum_{r \in \mathcal{R}} p(\{(s_t, r)\}; s_{t-1}, a_{t-1}) \cdot \underbrace{\sum_{a \in \mathcal{A}} \pi_t(\{a\}; s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)}_{=1}. \end{aligned}$$

Hence, using the definition of conditional probability the fraction cancels to give

$$\mathbb{P}_\mu^\pi(A_t \in C | S_0 = s_0, A_0 = a_0, \dots, A_{t-1} = a_{t-1}, S_t = s_t) = \sum_{a \in C} \pi_t(\{a\}; s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t).$$



Check the other two claimed conditional probability identities.

□



Definition 3.1.4. (Markov decision process (MDP))

Given a Markov decision model $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, a policy π , and an initial distribution μ on \mathcal{S} , the stochastic process $(S_t, A_t)_{t \in \mathbb{N}_0}$ on $(\Omega, \mathcal{F}, \mathbb{P}_\mu^\pi)$ is called discrete-time Markov decision process (MDP), $(R_t)_{t \in \mathbb{N}_0}$ the corresponding reward process. To abbreviate we will write \mathbb{P}_s^π instead of $\mathbb{P}_{\delta_s}^\pi$. The super- and subscript are sometimes removed from \mathbb{E}_μ^π and \mathbb{P}_μ^π if there is no possible confusion about the initial distribution of \mathcal{S} and the policy.

In the literature, a Markov decision model is mostly called an MDP directly and the terms model and process are not distinguished. We keep in mind that the Markov decision model just provides the prevailing instructions and together with any policy and initial distribution induces an MDP. However, for reasons of convenience, we will mostly use the term MDP interchangeable, if we do not want to emphasize the differences.



It is very important to remember the formula

$$\begin{aligned} & \mathbb{P}(S_0 = s_0, R_0 = r_0, A_0 = a_0, \dots, S_T = s_T, R_T = r_T, A_T = a_T) \\ &:= \mu(\{s_0\}) \cdot \delta_0(r_0) \cdot \pi_0(\{a_0\}; s_0) \\ & \quad \times \prod_{i=1}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i), \end{aligned} \tag{3.3}$$

which gives the probabilities an MDP follows a given path $(s_0, a_0, r_0, \dots, s_T, a_T, r_T)$ of states, actions, and rewards. The formula will later explain the practical impor-



tance of the policy gradient theorem.

It is always very instructive to compare with the situation of an ordinary Markov chain that appears as a special case if $\mathcal{A} = \{a\}$ and $\mathcal{R} = \{r\}$. In that case then the process (S_t) is a Markov chain with transition matrix $p_{s_i, s_j} = p(\{s_j, r\}; s_i, a)$ and the path probabilities immediately simplify to the Markov chain formula (3.1). No doubt, for MDPs the formula is more complicated but many probabilities can be computed similarly to the situation of a Markov chain. Playing with conditional probabilities it is instructive to check the following formulas (using a cancellation of the form $\sum_i (a \cdot b_i) / \sum_i b_i = a$):

$$\begin{aligned} & \mathbb{P}(S_t = s_t, R_t = r_t, A_t = a_t, \dots, S_T = s_T, R_T = r_T, A_T = a_T \mid S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}) \\ &= \prod_{i=t}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i) \end{aligned}$$

and

$$\begin{aligned} & \mathbb{P}(S_t = s_t, R_t = r_t, A_t = a_t, \dots, S_T = s_T, R_T = r_T, A_T = a_T \mid S_{t-1} = s) \\ &= \sum_{a_{t-1} \in \mathcal{A}_{s-1}} \pi_i(\{a_{t-1}\}; s_0, a_0, \dots, a_{i-1}, s) \prod_{i=t-1}^T p(\{(s_i, r_i)\}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i). \end{aligned}$$

which are the analogues to the Markov chain formulas recalled above.

Here are two more special situations in which more simple Markov decision models can be abstracted in our general definition:

Example 3.1.5. There are many situations in which the reward and the transition both only depend on (s, a) but do not depend on each other. Suppose there are

- a kernel h on $\bar{\mathcal{S}} \times (\mathcal{S} \times \mathcal{A})$ for the transitions from (s, a) to s' ,
- a kernel q on $\bar{\mathcal{R}} \times (\mathcal{S} \times \mathcal{A})$ for the rewards r obtained from (s, a) .

Note that in the discrete setting measurability is always satisfied, kernel only means that $h(\cdot; s, a)$ and $q(\cdot; s, a)$ are discrete measures for all state-action pairs (s, a) . If both transitions are assumed to be independent then we can define the product kernel

$$p(\{(s', r)\}; a, s) := h(\{s'\}; a, s) \cdot q(\{r\}; a, s) \quad (3.4)$$

on $\bar{\mathcal{S}} \otimes \bar{\mathcal{R}} \times (\mathcal{S} \times \mathcal{A})$ and the corresponding Markov decision process samples next states/rewards independently. Now the Markov decision model in the sense of Definition 3.1.1 with Markov decision process from Theorem 3.1.3 has exactly the claimed transitions. Plugging-in p immediately gives

$$\begin{aligned} \mathbb{P}(R_{t+1} \in D \mid S_t = s_t, A_t = a_t, S_{t+1} \in B) &\stackrel{\text{cond. prob.}}{=} \frac{\mathbb{P}(S_{t+1} \in B, R_{t+1} \in D \mid S_t = s_t, A_t = a_t)}{\mathbb{P}(S_{t+1} \in B \mid S_t = s_t, A_t = a_t)} \\ &\stackrel{3.1.3}{=} \frac{p(B \times D; s_t, a_t)}{p(B \times \mathcal{R}; s_t, a_t)} \\ &\stackrel{(3.4)}{=} p(\mathcal{S} \times D; s_t, a_t) \\ &\stackrel{3.1.3}{=} \mathbb{P}(R_{t+1} \in D \mid S_t = s_t, A_t = a_t), \end{aligned}$$

so that the proclaimed independence of the reward from the future state indeed holds.

Example 3.1.6. There are many situations in which the rewards are deterministic functions of state, actions, and next state but themselves do not influence the next state (say $R_{t+1} =$

$R(s_t, a_t, s_{t+1})$). An example appears in the ice vendor example below. In that situation the kernel p can be written as

$$p(\{s'\} \times B; s, a) = h(\{s'\}; s, a) \cdot \delta_B(R(s, a, s')),$$

where h is the transition function from (s, a) to the next state s' . In that case one obtains immediately

$$\begin{aligned} \mathbb{P}(S_{t+1} \in D | S_0 = s_0, A_0 = a_0, \dots, A_t = a_t) &= h(D; s_t, a_t), \\ \mathbb{P}(R_{t+1} \in B | S_0 = s_0, A_0 = a_0, \dots, A_t = a_t) &= \delta_B(R(s, a, s')). \end{aligned}$$

An even simpler situation that is very common is a relation $R_{t+1} = R(S_t, A_t)$ where the rewards are determined deterministically by the prior state-action pair.

The discussion is sometimes reversed as follows, with slight abuse of notation:



Definition 3.1.7. For a Markov decision model we define state-action-state probabilities and state-action expected rewards as follows:

$$\begin{aligned} p(s'; s, a) &:= p(\{s'\} \times \mathcal{R}; s, a), \\ r(s, a) &:= \sum_{r \in \mathcal{R}} r p(\mathcal{S} \times \{r\}; s, a), \\ r(s, a, s') &:= \sum_{r \in \mathcal{R}} r \frac{p(\{(s', r)\}; s, a)}{p(s'; s, a)} \end{aligned}$$

for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ if the denominator is non-zero. Note that all p and r are defined through the Markov decision model only they can always be assumed to be known explicitly from the model only.

The notation will be used frequently, in reinforcement algorithms of the upcoming chapters.



Use the formal definition of the stochastic process (S, A, R) to check that

$$\begin{aligned} p(s'; s, a) &= \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a), \\ r(s, a) &= \mathbb{E}[R_{t+1} | S_t = s, A_t = a], \\ r(s, a, s') &= \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] \end{aligned}$$

holds if the events in the condition have positive probability.

To get an idea about Markov decision processes let us discuss two examples of simple Markov decision models, an example with terminating states and an example without terminating states.³

Example 3.1.8. (Grid world)

Suppose you have a robot that you would like to teach to walk through your garden. The garden is set up as a small grid and at each time the robot can go up, down, left or right, however the robot cannot move through a wall (or additional obstacles). The aim is to move from the starting position S to the target position G . To make the problem a bit more difficult, there is a pit B just before the goal G . If the robot lands in the pit or in the target, the game is over. In more brutal versions of the game, the pit is also interpreted as a bomb, which is why the abbreviation B is used. In the following we will formulate this example as a Markov Decision Model. For this purpose, the individual objects from Definition 3.1.1 must be determined.

The first step is to determine the state space. The state space should contain all relevant information about the game. The garden can be represented by a two dimensional grid similar

³Leif: definiere terminating MDPs schon hier

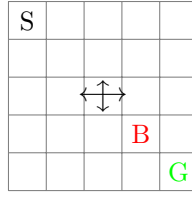


Figure 3.1: Illustration of Example 3.1.8 for a grid of length 5. The start of the robot is marked by S , the pit by B , and the goal by G .

0,0	1,0	2,0	3,0	4,0
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

Figure 3.2: Representation of the state space of Example 3.1.8 for a grid of length 5.

to a chess board. Thus, for a grid with side length $Z \in \mathbb{N}, Z > 2$, the state space is given by $S := \{(i, j), i, j \in \{0, 1, \dots, Z - 1\}\}$. Since the state space S is finite, we will use as σ -algebra the power set, i.e. $\mathcal{S} := \mathcal{P}(S)$. At this point we want to note that we use Python for the implementation of the games, which is why we start counting from zero.

The second component of the Markov decision model is the entire action space \mathcal{A} and the action space \mathcal{A}_s for a state $s \in S$. In many cases it is easier to first define the set of all possible actions \mathcal{A} and then exclude actions that are not possible for a certain state $s \in S$ to determine \mathcal{A}_s . In this example we want to use this method to define the action spaces of the individual states as well as the whole action space.

An action describes the movement of the robot down, up, right or left. Often actions in a Markov decision problem are simply ‘counted through’. For our example, the action space could be given by the set $\{0, 1, 2, 3\}$, where action 0 describes the movement of the robot to the right, action 1 describes the movement of the robot downwards and so on. To describe the actions in a more mathematically meaningful way and to relate them to the state space we describe the action space by the following object:

$$\{\text{right, up, left, down}\} \equiv \mathcal{A} := \{(1, 0), (0, 1), (-1, 0), (0, -1)\} \subseteq \mathbb{R}^2$$

The advantage of this definition is that by simply adding states and actions we get the new state of the robot. In the following we want to determine the action space for a state $s \in \mathcal{S}$ using the approach described above. The approach is also called masking. Let $s = (i, j) \in S$ then the valid actions for the state are given by

$$\mathcal{A}_s = \mathcal{A}_{(i,j)} = \mathcal{A} \setminus \begin{cases} \{2\} & : \text{if } i = 0 \\ \{1\} & : \text{if } i = Z - 1 \\ \emptyset & : \text{otherwise} \end{cases} \setminus \begin{cases} \{0\} & : \text{if } j = 0 \\ \{3\} & : \text{if } j = Z - 1 \\ \emptyset & : \text{otherwise} \end{cases}$$

The idea here is relatively simple. The agent may play all actions, except the agent is on the edge of the playing field. Then certain actions must be excluded.

The third components of the Markov decision model are the rewards. We introduce the rewards and motivate them afterwards. Let \mathcal{R} be given by $\{-10, -1, 0, 10\}$ and $\overline{\mathcal{R}} := \mathcal{P}(\mathcal{R})$. If the agent lands in the target, the reward is 10. If the agent falls into the pit, the reward is -10 . We want to determine the rewards in such a way that desirable scenarios are rewarded and undesirable scenarios are punished. If the agent lands on another field, the agent should receive the reward -1 . In this way we want to avoid that the agent is not penalized for not choosing the fastest

possible way to the destination. This will become clear when we deal with the optimization problem in Markov decision problems in the following chapters. The 0 reward is used to construct the stochastic process.

Next, we define the fourth and probably most complicated component of the Markov decision process, the transition/reward-function

$$p : \bar{\mathcal{S}} \otimes \bar{\mathcal{R}} \times (\mathcal{S} \times \mathcal{A}) \rightarrow [0, 1], (B, (s, a)) \mapsto p(B; s, a).$$

To define this function, we will first define a helper function $g : \mathcal{S} \rightarrow \mathcal{R}$ which should return the associated reward for a state $s \in \mathcal{S}$. The function is given by

$$g : \mathcal{S} \rightarrow \mathcal{R}, s \mapsto \begin{cases} 10 & : s = (Z - 1, Z - 1) \\ -10 & : s = (Z - 2, Z - 2) \\ -1 & : \text{otherwise} \end{cases}.$$

In addition, we still need to define the set of terminal states T . In a terminal state $s \in \mathcal{S}$ the agent is stuck, this means that playing any action $a \in \mathcal{A}_s$ does not lead to any change of the state. In our example, the set of terminal states is given by $T = \{(Z - 2, Z - 2), (Z - 1, Z - 1)\}$, so the goal position G and the pit position B . Let $B \in \bar{\mathcal{S}}$ and $C \in \bar{\mathcal{R}}$, then we define the transition function by

$$p(B \times C; (s, a)) = \mathbf{1}_T(\{s\}) \cdot \mathbf{1}_C(g(s)) + \mathbf{1}_{T^c}(\{s\}) \cdot (\mathbf{1}_B(s + a) \cdot \mathbf{1}_C(g(s + a)))$$

for $(s, a) \in \mathcal{S} \times \mathcal{A}_s$. As a small task, it can be recalculated that the transition/reward function is a Markov kernel. The transition/reward function looks a bit confusing and complicated at first sight. If we break the function down into its individual cases, we quickly realize that the function has the desired properties. Thus all components of a Markov Decision Model are defined. Since those will be used in algorithms discussed in future chapters let us compute the functions from Definition 3.1.7. For $(s, a) \in \mathcal{S} \times \mathcal{A}_s$ the transition probability to arrive in state s' when playing action $a \in \mathcal{A}_s$ in state $s \in \mathcal{S}$ is

$$p(s'; s, a) = \sum_{r \in \mathcal{R}} p(\{s'\}, \{r\}; s, a) = \mathbf{1}_T(\{s\}) + \mathbf{1}_{T^c}(\{s\}) \cdot (\mathbf{1}_B(s + a)).$$

Furthermore, the expected reward when playing action $a \in \mathcal{A}_s$ in state $s \in \mathcal{S}$ is

$$r(s, a) = \mathbf{1}_T(\{s\}) \cdot g(s) + \mathbf{1}_{T^c}(\{s\}) \cdot g(s + a).$$

Grid world is a standard example on which tabular reinforcement algorithms of the following chapters will be tested.

We next formulate a simple supply optimisation problem in the setting of Markov decision processes, an ice vendor who has to decide every day about the amount of produced/bought ice cream. In the course of these lecture notes it will be discussed how to optimise the production decision in order to maximise the profit.

Example 3.1.9. (Ice vendor)

The ice vendor owns an ice cream truck that can store up to $M \in \mathbb{N}$ ice creams. Every morning the ice vendor can produce/buy a discrete amount $0, 1, 2, \dots, M$ of ice cream. Throughout the day, a random amount of ice cream is consumed, and the demand can be higher than the stored amount of ice cream. In this simplified model, seasonality in demand is not considered. If less ice cream is consumed than the ice vendor has in stock, the ice cream must be stored overnight and can then be sold the next day. There are costs for storing ice cream overnight, thus, the ice vendor should carefully decide about the amount of ice cream produced/bought in the morning. In addition, there are costs for the production/purchase of ice cream. For the sale of ice cream the vendor receives a fixed price per ice cream scoop. For simplicity, we also assume that revenues and costs do not change over time (which is realistic for ice cream, the price is

typically adjusted once per year). The trade-off in this problem is simple. How much ice should be ordered in the morning so that the vendor can maximize the revenue (serve demand) but keep the costs low.

The ice vendor situation can be formulated as Markov decision process. The state space is given by the stored amount of ice: $\mathcal{S} = \{0, 1, \dots, M\}$. An action models the amount of ice cream produced/ordered in a morning. If there are already $s \in \mathcal{S}$ scoops in stock, the vendor can produce/order at most $M - s$ many scoops. Thus, $\mathcal{A}_s = \{0, 1, \dots, M - s\}$ and $\mathcal{A} = \{0, \dots, M\}$. For demand, we assume that it is independently identically distributed regardless of timing. Thus $\mathbb{P}(D_t = d) = p_d$ for $d \in \mathbb{N}$ holds for all time $t \in \mathbb{N}$. For simplicity, let us assume the demand can only take values in $\{0, 1, \dots, M\}$. In order to define the transition reward function and the reward, auxiliary functions are used to determine the revenues and costs. The selling price of a scoop of ice cream is given by a function $f : \mathcal{S} \rightarrow \mathbb{R}_+$, for instance $f(s) = c \cdot s$, where $c > 0$ is the selling price for a scoop of ice cream. Similarly, we define a mapping $o : \mathcal{A} \rightarrow \mathbb{R}_+$ for the production/purchase cost of the products and $h : \mathcal{S} \rightarrow \mathbb{R}_+$ for the storage cost. Thus, for a pair $(s, a) \in \mathcal{S} \times \mathcal{A}_s$ and another state $s' \in \mathcal{S}$, the gain is given by the mapping $R : (\mathcal{S} \times \mathcal{A}_s) \times \mathcal{S} \rightarrow \mathbb{R}$ with

$$R(s, a, s') := \underbrace{f(s + a - s')}_{\text{sold ice cream}} - \underbrace{o(a)}_{\text{production costs}} - \underbrace{h(s + a)}_{\text{storage costs}}.$$

Moreover, let us define a mapping $h : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$

$$h(s'; s, a) := \begin{cases} p_{s+a-s'} & : 1 \leq s' \leq s + a \\ \sum_{i \geq 0} p_{s+a+i} & : s' = 0 \\ 0 & : \text{otherwise} \end{cases}$$

and from this the transition probabilities

$$p(\{(s', r)\}; s, a) := h(s'; s, a) \cdot \mathbf{1}_{\{r\}}(R(s, a, s')).$$

As for grid world we can compute the state-action-state transition probabilities as

$$p(s'; s, a) = p(\{s'\} \times \mathcal{R}; s, a) = \sum_{r \in \mathcal{R}} h(s'; s, a) \cdot \mathbf{1}_{\{r\}}(R(s, a, s')) = h(s'; s, a)$$

and the reward expectations as

$$\begin{aligned} r(s, a) &= \sum_{r \in \mathcal{R}} r p(\mathcal{S} \times \{r\}; s, a) \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r p(\{(s', r)\}; s, a) \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r \cdot h(s'; s, a) \cdot \mathbf{1}_{\{r\}}(R(s, a, s')) \\ &= \sum_{s' \in \mathcal{S}} R(s, a, s') \cdot h(s'; s, a). \end{aligned}$$

So far the definition of Markov decision models and policies is very general. It will later turn out that much smaller classes of policies are necessary to solve optimal control problems in the MDP setting.



Definition 3.1.10. (Markov and stationary policies)

A policy $\pi = (\pi_t)_{t \in \mathbb{N}_0} \in \Pi$ is called

- (i) a Markov policy if there exists a sequence of kernels $(\varphi_t)_{t \in \mathbb{N}_0}$ on $\bar{\mathcal{A}} \times \mathcal{S}$ such that

$$\pi_t(\cdot; s_0, a_0, \dots, s_t) = \varphi_t(\cdot; s_t), \quad \forall (s_0, a_0, \dots, s_t) \in (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S}.$$



The set of all Markov policies is denoted by Π_M .

(ii) a stationary policy if there exists a kernel φ on $\bar{\mathcal{A}} \times \mathcal{S}$ such that

$$\pi_t(\cdot; s_0, a_0, \dots, s_t) = \varphi(\cdot; s_t), \quad \forall (s_0, a_0, \dots, s_t) \in (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S}.$$

The set of all stationary policies is denoted by Π_S .

(iii) a deterministic stationary policy if there exists a kernel φ on $\bar{\mathcal{A}} \times \mathcal{S}$ taking only values in $\{0, 1\}$ such that

$$\pi_t(\cdot; s_0, a_0, \dots, s_t) = \varphi(\cdot; s_t), \quad \forall (s_0, a_0, \dots, s_t) \in (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S}.$$

The set of all deterministic stationary policies is denoted by Π_S^D .

In the stationary cases we typically write just π instead of φ .

From the definition it holds that

$$\Pi_S^D \subseteq \Pi_S \subseteq \Pi_M.$$

In words: A Markov policy only uses the actual state (not the past) to chose the action, a stationary policy does not depend on time (and the past), a deterministic stationary policy only choses one action (like an index strategy for bandits). In fact, it will turn out that only deterministic stationary policies are needed to solve the central optimisation problem that will be defined below.

A special type of policies will be used later, those policies always first chose a fixed action and then proceed according to some policy.



Definition 3.1.11. We denote by Π^a the set of all policies with $\pi_0(a; s) = 1$ for all $s \in \mathcal{S}$. For a given policy $(\pi_t)_{t \in \mathbb{N}_0}$ we denote by π^a it's modification in which π_0 is replaced by the deterministic action distribution only charging a .

We will call such policies one-step deterministic.

Lecture 7

Example 3.1.12. (Multiarmed bandits)

A very trivial situation is a multiarmed bandits. There are actually two ways of interpreting bandits as MDPs.

- If $|\mathcal{S}| = 1$, then a one-step Markov decision model is nothing but a bandit model that is played once. There are only actions (arms) that are played once according to a one-step policy, R_1 is the reward obtained by playing the arm.
- If $|\mathcal{S}| = 1$ and $T = n$, then a Markov decision model is a bandit model and a policy is a learning strategy. The rewards R_1, \dots, R_n are the outcomes of playing the arms chosen according to π . The way a learning strategy was defined for bandits it is neither Markovian nor stationary policy.

We now can prove that for a Markov policy, and hence also for (deterministic) stationary policies, a Markov Decision Processes has the Markov Property.



Proposition 3.1.13. (Markov Property)

Let (S, A) be a Markov decision process on $(\Omega, \mathcal{F}, \mathbb{P})$ with Markov policy (π_t) and initial distribution μ . Then $(S_t, A_t)_{t \in \mathbb{N}_0}$ satisfies the Markov Property, i.e.

$$\begin{aligned} & \mathbb{P}(S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1} | S_0 = s_0, A_0 = a_0, \dots, S_t = s_t, A_t = a_t) \\ &= \mathbb{P}(S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1} | S_t = s_t, A_t = a_t). \end{aligned}$$

If $\pi \in \Pi_S$, then (S, A) is a time-homogeneous Markov chain on $\mathcal{S} \times \mathcal{A}$ with transition



matrix

$$p_{(a,s),(a',s')} = p(\{s'\} \times \mathcal{R}; s, a) \cdot \pi(a'; s').$$

Proof. The main point of the proof (similarly to the above proofs) is to realise that

$$\begin{aligned} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} p(\{s\} \times \mathcal{R}; s', a') \cdot \varphi_t(a; s) &= \sum_{s \in \mathcal{S}} p(\{s\} \times \mathcal{R}; s', a') \sum_{a \in \mathcal{A}} \varphi_t(a; s) \\ &= \sum_{s \in \mathcal{S}} p(\{s\} \times \mathcal{R}; s', a') \cdot 1 = 1. \end{aligned}$$

Then the proof is of the cancellation form $\sum_i (a \cdot b_i) / \sum_i b_i = a$. Combined with the path probabilities we obtain

$$\begin{aligned} &\mathbb{P}(S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1} | S_t = s_t, A_t = a_t) \\ &= \frac{\mathbb{P}(S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1}, S_t = s_t, A_t = a_t)}{\mathbb{P}(S_t = s_t, A_t = a_t)} \\ &= \frac{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mathbb{P}(S_0 = s_0, A_0 = a_0, \dots, S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1})}{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mathbb{P}(S_0 = s_0, A_0 = a_0, \dots, S_t = s_t, A_t = a_t)} \\ &= \frac{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \pi_0(\{a_0\}; s_0)}{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \pi_0(\{a_0\}; s_0)} \\ &\quad \times \frac{\prod_{i=1}^{t+1} p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i)}{\prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \pi_i(\{a_i\}; s_0, a_0, \dots, a_{i-1}, s_i)} \\ &\stackrel{\pi \in \Pi_M}{=} \frac{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \pi_0(\{a_0\}; s_0) \prod_{i=1}^{t+1} p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)}{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \pi_0(\{a_0\}; s_0) \prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)} \\ &= \frac{\sum_{s_0, a_0} \dots \sum_{s_{t-2}, a_{t-2}} \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^{t+1} p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)}{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)} \\ &= p(\{s_{t+1}\} \times \mathcal{R}; s_t, a_t) \cdot \varphi_{t+1}(a_{t+1}; s_{t+1}) \\ &\quad \times \frac{\sum_{s_0, a_0} \dots \sum_{s_{t-2}, a_{t-2}} \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)}{\sum_{s_0, a_0} \dots \sum_{s_{t-1}, a_{t-1}} \mu(\{s_0\}) \cdot \pi_0(\{a_0\}; s_0) \cdot \prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \cdot \varphi_i(a_i; s_i)} \\ &= p(\{s_{t+1}\} \times \mathcal{R}; s_t, a_t) \cdot \varphi_{t+1}(a_{t+1}; s_{t+1}) \\ &= \frac{\mathbb{P}(S_0 = s_0, A_0 = a_0, \dots, S_t = s_t, A_t = a_t, S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1})}{\mathbb{P}(S_0 = s_0, A_0 = a_0, \dots, S_t = s_t, A_t = a_t)} \\ &= \mathbb{P}(S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1} | S_0 = s_0, A_0 = a_0, \dots, S_t = s_t, A_t = a_t). \end{aligned}$$

Finally, if $\varphi = \varphi$ the computation (compare the third line from below) shows that $\mathbb{P}(S_{t+1} = s', A_{t+1} = s' | S_t = s, A_t = a)$ is independent of t . \square

3.1.3 Stochastic control theory

So far we have introduced the concept of a Markov decision processes for a given transition function p and a policy π . But what is the actual question that we are after? The question is to find a policy that maximise what we will call the expected discounted reward. Since a policy can be used to control a system this problem is also known under stochastic optimal control. In order to do so there will be two main steps. How to compute the expected discounted reward (this will be called prediction) and then how to find the optimal policy (this will be called control). Here is the optimization target that stochastic optimal control is about and reinforcement learning tries to solve:



Given a Markov reward model find a policy π that maximizes the expected sum of discounted future rewards for all initial conditions s :

$$\mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right].$$

The parameter γ is a fixed positive number strictly smaller than 1 that ensures convergence of the sum (as we will assume R to be bounded). As its choice does not play a fundamental role the dependence on γ is typically ignored in all objects.

A little note on notation. We will frequently misuse notation and write $\mathbb{P}(\cdot \mid S_0 = s)$ instead of $\mathbb{P}_{\delta_s}(\cdot)$ or $\mathbb{P}_s(\cdot)$. This is justified for MDPs by the fact that $\mathbb{P}_\mu(\cdot \mid S_0 = s) = \mathbb{P}_s(\cdot)$ whenever $\mu(\{s\}) > 0$ as can be seen from cancellations in the path probability formula (3.3).

There are other possible optimisation targets. If the time-horizon is finite then one might also be interested in optimising $\mathbb{E}^\pi [\sum_{k=1}^T R_k]$. For infinite time-horizon alternatively one might aim to optimise $\lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}^\pi [\sum_{k=1}^T R_k]$. In these lecture notes we mainly focus on discounted rewards.

Before going into the details let us fix some conditions that we want to assume, mostly to make our lives easier:

Assumption 1: The state space \mathcal{S} and the action space \mathcal{A} are finite, the set of rewards \mathcal{R} is discrete and bounded. All appearing σ -algebras are chosen to be the power sets.

The assumption of bounded rewards is not necessary but makes our lives much easier for the presentation as everything will be finite and exchanging expectations and sums will always follow from dominated convergence.

The assumption is not needed but makes the notation simpler. In reinforcement it is actually a non-trivial but very important problem how to deal carefully with the sets of allowed actions.

Assumption 2: Given a state-action pair, the reward is independent of the next state, i.e.

$$\mathbb{P}(R_{t+1} \in D \mid S_t = s_t, A_t = a_t, S_{t+1} \in B) = \mathbb{P}(R_{t+1} \in D \mid S_t = s_t, A_t = a_t).$$

This is the special case discussed in Example 3.1.5. Formulated in terms of the model transition function this means

$$p(\{s, r\}; s, a) = p(\mathcal{S} \times \{r\}; s, a) \cdot p(\{s\} \times \mathcal{R}; s, a).$$



Definition 3.1.14. (State value function)

For $\pi \in \Pi$ and $\gamma \in (0, 1)$, the function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ defined by

$$V^\pi(s) := \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]$$

is called state-value function. Occasionally we will also use

$$V^\pi(\mu) = \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]$$

for the expected discounted reward for the MDP started in the distribution μ .

Recalling the notation π^a for the policy π modified to first chose action a we also define the state-action function (Q -function):


Definition 3.1.15. (State-action value function - Q-function)

For $\pi \in \Pi$ and $\gamma \in (0, 1)$, the function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defined by

$$Q^\pi(s, a) = V^{\pi^a}(s)$$

is called state-action value function; or Q-function.

In words: The state value functions is the expected discounted total reward when started in s , the state-action function the expected total reward if started in s and a is the first action played. It is important to note that the state and state-action value functions are typically defined differently:



The typical definitions of V^π and Q^π are as follows:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right] \quad \text{and} \quad Q^\pi = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right]$$

In fact, if μ is an initial condition with $\mu(s) > 0$ and π is a policy with $\pi(a; s) > 0$ then both definitions are equal and do not depend on μ !

The claim follows by a quick computation with (3.3). If $\mu(s) = 0$ and/or $\pi(a; s) = 0$ the typical definition is not even defined! For V the problem can be solved easily by choosing μ to be uniform on \mathcal{S} . For Q we could choose μ uniform (or $\mu = \delta_s$) and modify π_0 to only choose a . This is exactly what the definition is about. A quick computation shows that our definition and the classical definition is equal if $\mu(s) > 0$ and $\pi_0(a; s) > 0$.

The general aim in the following is two-fold:

- How to compute the value and state-value function for a given policy?
- How to find an optimal policy, i.e. a policy that maximises (in the right sense) the value function?

The Q -function is not the main object of interest but very useful to simplify appearing equations. In the next chapter it will turn out to be more useful to numerically find the optimal Q -function. In order to derive a set of linear equations for the value function (the so-called Bellman expectation equation) we will start with the following lemma:



Lemma 3.1.16. Suppose $s \in \mathcal{S}, a \in \mathcal{A}$ and $s' \in \mathcal{S}$ is such that $p(s'; s, a) > 0$. Then

$$\sup_{\pi \in \Pi} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] = \sup_{\pi \in \Pi} V^\pi(s')$$

and, if $\pi \in \Pi$ is stationary also

$$\mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] = V^\pi(s').$$

Proof. The basic argument is the same for both claims. Define

$$\tilde{R}_t := R_{t+1}, \quad \tilde{S}_t := S_{t+1}, \quad \tilde{A}_t := A_{t+1}, \quad t \geq 0,$$

the shifted policy

$$\tilde{\pi}_t(\tilde{a}; \tilde{s}_0, \tilde{a}_0, \dots, \tilde{s}_t) = \pi_{t+1}(\tilde{a}; s, a, \tilde{s}_0, \tilde{a}_0, \dots, \tilde{s}_t),$$

and a new probability measure $\tilde{\mathbb{P}}^\pi := \mathbb{P}^\pi(\cdot | S_0 = s, A_0 = a, S_1 = s')$. Then on the probability space $(\Omega, \mathcal{F}, \tilde{\mathbb{P}}^\pi)$, the process $(\tilde{S}_t, \tilde{A}_t)_{t \geq 0}$ is an MDP started in $\mu = \delta_{s'}$, transition function p , and policy $(\tilde{\pi}_t)_{t \in \mathbb{N}_0}$. To see why, one only needs to compute the path probabilities:

$$\begin{aligned}
& \tilde{\mathbb{P}}^\pi(\tilde{S}_0 = s_0, \tilde{A}_0 = a_0, \dots, \tilde{S}_t = s_t, \tilde{A}_t = a_t) \\
&= \frac{\mathbb{P}^\pi(S_1 = s_0, A_1 = a_0, \dots, S_{t+1} = s_t, A_{t+1} = a_t, S_0 = s, A_0 = a, S_1 = s')}{\mathbb{P}^\pi(S_0 = s, A_0 = a, S_1 = s')} \\
&= \frac{\delta_{s'}(s_0) \cdot \mu(s) \cdot \pi_0(a_0; s_0) \cdot \prod_{i=1}^{t+1} p(\{s_{i+1}\} \times \mathcal{R}; s_i, a_i) \pi_i(\{a_i\}; s, a, s_0, a_0, \dots, s_{i-1}, a_{i-1}, s_i)}{\mu(s) \cdot \pi_0(a; s) \cdot p(\{s'\} \times \mathcal{R}; s, a)} \\
&= \delta_{s'}(s_0) \cdot \pi_1(a_1; s, a, s_1) \cdot \prod_{i=2}^{t+1} p(\{s_{i+1}\} \times \mathcal{R}; s_i, a_i) \pi_i(\{a_i\}; s, a, s_0, a_0, \dots, s_{i-1}, a_{i-1}, s_i) \\
&= \delta_{s'}(s_0) \cdot \tilde{\pi}_0(a_1; s_0) \cdot \prod_{i=1}^t p(\{s_i\} \times \mathcal{R}; s_{i-1}, a_{i-1}) \tilde{\pi}_i(\{a_i\}; s_0, a_0, \dots, s_{i-1}, a_{i-1}, s_i)
\end{aligned}$$

If π is stationary, then $\pi = \tilde{\pi}$ so that the shifted MDP (\tilde{S}, \tilde{A}) under $\mathbb{P}^\pi(\cdot | S_0 = s, A_0 = a, S_1 = s')$ has the same law as (S, A) . If the law of R_{t+1} is determined by (S_t, A_t) only, then the law of \tilde{R}_{t+1} is determined by \tilde{S}_t, \tilde{A}_t only. Hence, the second claim follows.

The first claim is a bit more delicate as the shifted process is an MDP with same transition function p but different policy $\tilde{\pi}$. If we denote by $\tilde{\Pi}$ all policies obtained from Π^a (the policies that always start with action a) by shifting by 1 (which is actually identical to Π), then

$$\begin{aligned}
\sup_{\pi \in \Pi} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] &= \sup_{\pi \in \Pi^a} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] \\
&= \sup_{\pi \in \Pi^a} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_0 = s, A_0 = a, S_1 = s' \right] \\
&= \sup_{\pi \in \Pi^a} \tilde{\mathbb{E}}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t \tilde{R}_{t+1} \right] \\
&= \sup_{\tilde{\pi} \in \tilde{\Pi}^a} \mathbb{E}_s^{\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t \tilde{R}_{t+1} \right] \\
&= \sup_{\pi \in \Pi} V^\pi(s).
\end{aligned}$$

□

The upcoming proofs will always use conditioning on $A_0 = a$ which is problematic if $\pi(a; s) = 0$. To avoid trouble we will denote $\mathcal{A}_s^a = \{a \in \mathcal{A} : \pi(a; s) > 0\}$. The notation will only appear in the proofs and usually drop out in the theorems. The following trick will be used a few times with $A_a = \{A_0 = a\}$:



If $\cup_a A_a$ is a partition of Ω with $\mathbb{P}(A_a) > 0$ for all a , then expanding the definition of conditional probability yields

$$\mathbb{P}(B|C) = \sum_a \mathbb{P}(A_a|C) \mathbb{P}(B|C \cap A_a). \quad (3.5)$$

Similarly, writing $\mathbb{E}[X|C] = \int_\Omega X(\omega) \mathbb{P}(d\omega|C)$ also yields

$$\mathbb{E}[X|C] = \sum_a \mathbb{P}(A_a|C) \mathbb{E}[X|C \cap A_a]. \quad (3.6)$$

The next simple lemma is important, it shows how to relate value and state-value functions.



Lemma 3.1.17. Given a stationary policy $\pi \in \Pi_S$, the following relations between the state- and action-value functions hold:

$$V^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^\pi(s, a), \quad s \in \mathcal{S},$$

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^\pi(s'), \quad s \in \mathcal{S}, a \in \mathcal{A}.$$

On first view the lemma shows that thinking of V^π or Q^π is equivalent. There is an important detail that will only become relevant in the next sections. To compute Q^π from V^π the transition function must be known. In so-called model free RL we will later try to remove that assumption but then cannot simply compute Q^π from V^π .

Proof. Writing out conditional probabilities (and discrete expectation as sum over conditional probabilities) yields the first claim:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right] \\ &= \sum_{a \in \mathcal{A}_s} \mathbb{P}^\pi(A_0 = a \mid S_0 = s) \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right] \\ &= \sum_{a \in \mathcal{A}_s} \pi_0(a; s) Q^\pi(s, a) \\ &\stackrel{\text{stationary}}{=} \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^\pi(s, a). \end{aligned}$$

For the second equation we first do the same and use the definition of r :

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\ &= \mathbb{E}_s^{\pi^a} [R_1] + \gamma \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_0 = s, A_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right]. \end{aligned}$$

Using Lemma 3.1.16 yields the claim. \square

The classical theory of optimising Markov decision processes is very much about functional analysis. For that sake we will always use the Banach space $(U, \|\cdot\|_\infty)$ consisting of $U := \{u : \mathcal{S} \rightarrow \mathbb{R}\}$ equipped with the maximum-norm. Since we assume that \mathcal{S} is finite, U is nothing but $\mathbb{R}^{|\mathcal{S}|}$ where a vector is written as a function (mapping indices to coordinate values).



Definition 3.1.18. (Bellman expectation operator)

Given a Markov decision model and a stationary policy π the operator

$$(T^\pi u)(s) := \sum_{a \in \mathcal{A}_s} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) u(s') \right),$$

mapping U into U is called the Bellman expectation operator.

The Bellman expectation operator can also be written in a more compact form if we are willing to use probabilities and assume that $\mu(s) > 0$, e.g. μ uniform:

$$(T^\pi u)(s) = \sum_{a \in \mathcal{A}_s} r(s, a) \pi(a; s) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}^\pi(S_1 = s' \mid S_0 = s) u(s').$$

To see this we only need to write the discrete conditional expectations:

$$\begin{aligned}\mathbb{P}^\pi(S_1 = s' | S_0 = s) &= \sum_{a \in \mathcal{A}_s} \frac{\mathbb{P}^\pi(S_1 = s', S_0 = s, A_0 = a)}{\mathbb{P}^\pi(S_0 = s)} \\ &= \sum_{a \in \mathcal{A}_s} \frac{\mathbb{P}^\pi(S_1 = s' | S_0 = s, A_0 = a) \mathbb{P}^\pi(S_0 = s, A_0 = a)}{\mathbb{P}^\pi(S_0 = s)} \\ &= \sum_{a \in \mathcal{A}_s} \frac{p(s'; s, a) \mu(s) \pi(a; s)}{\mu(s)} = \sum_{a \in \mathcal{A}_s} p(s'; s, a) \pi(a; s).\end{aligned}$$

One of the most fundamental results we need are the Bellman expectation equations. The value function solves a linear system of equations, alternatively, is a fixed point of the Bellman expectation operator:



Theorem 3.1.19. (Bellman expectation equation)

For every stationary policy π the value function satisfies the linear system of equations

$$V^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^\pi(s') \right), \quad s \in \mathcal{S}.$$

In words, V^π is a fixed point of the Bellman expectation operator T^π , i.e. $T^\pi V^\pi = V^\pi$.

Proof. Easy: just apply Lemma 3.1.17 twice:

$$V^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^\pi(s, a) = \sum_{a \in \mathcal{A}_s} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^\pi(s') \right)$$

□

Note that in principle Bellman's expectation equation can be solved to compute the value function V^π for a given stationary policy π . This is a linear equation for which linear algebra provides plenty of explicit and approximate solution methods.



Derive the expectation equation for the Q -function of a stationary policy π :

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_s} p(s'; s, a) \pi(a'; s) Q^\pi(s', a')$$

With the value functions we can define the concept of an optimal policies. The search for optimal policies in MDPs will be the main content of the subsequent chapters.



Definition 3.1.20. (Optimal policy & optimal value function)

For a given Markov decision model the following quantities will be of central importance:

- (i) The function $V^* : \mathcal{S} \rightarrow \mathbb{R}$ that takes values

$$V^*(s) := \sup_{\pi \in \Pi} V^\pi(s), \quad s \in \mathcal{S},$$

is called optimal (state) value function.

- (ii) A policy $\pi^* \in \Pi$ that satisfies

$$V^{\pi^*}(s) = V^*(s), \quad s \in \mathcal{S},$$

is called optimal policy.



(iii) The function $Q^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that takes values

$$Q^*(s, a) := \sup_{\pi \in \Pi} Q^\pi(s, a), \quad s \in \mathcal{S}, a \in \mathcal{A},$$

is called optimal state-action value function.

It should be emphasised that we make our lives much simpler by assuming a finite Markov decision model. In that case the maximum is always attained and plenty of technical problems do not appear. In fact, the results that we are going to prove next do not necessarily hold for infinite Markov decision models.



It is important to keep in mind that a priori V^* and Q^* are not the value functions for the best policy but instead pointwise the best possible expected rewards. In fact, it will turn out that there is a policy π^* (even stationary and deterministic!) with value functions achieving the best possible, i.e. $V^{\pi^*} = V^*$ and $Q^{\pi^*} = Q^*$. Even more, the policy can be obtained by solving a system of equations, Bellman's optimality equation.

The main result that is going to be proved in this section is that there is an optimal policy π , a policy that beats every other policy for every starting value. Even more, the policy is stationary and deterministic and can be obtained by solving a non-linear system of equations (Bellman's optimality equation).

Lecture 8



Lemma 3.1.21. The following holds for the optimal state-value function and the optimal state-action-value function:

- (i) $V^*(s) = \max_{a \in \mathcal{A}_s} Q^*(s, a)$ for $s \in \mathcal{S}$
- (ii) $Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^*(s')$ for $s \in \mathcal{S}, a \in \mathcal{A}$

The lemma even holds for non-discrete setting by replacing the max operator by the sup operator. Please keep already in mind the operation

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V(s'), \quad s \in \mathcal{S}, a \in \mathcal{A},$$

that turns the vector V into a matrix Q . The operation will occur frequently in what follows.

Proof. We start with the simpler task, obtaining V^* from Q^* :

$$\begin{aligned} \max_{a \in \mathcal{A}_s} Q^*(s, a) &= \max_{a \in \mathcal{A}_s} \sup_{\pi \in \Pi} Q^\pi(s, a) \\ &= \max_{a \in \mathcal{A}_s} \sup_{\pi \in \Pi} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\ &= \sup_{\pi \in \Pi} \max_{a \in \mathcal{A}_s} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\ &\stackrel{(*)}{=} \sup_{(\pi_t)_{t \in \mathbb{N}}} \sup_{\pi_0} \mathbb{E}_s^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\ &= \sup_{(\pi_t)_{t \in \mathbb{N}_0}} \mathbb{E}_s^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\ &= V^*(s). \end{aligned}$$

Checking (*) is not hard: „ \leq “ is trivial, since the deterministic policy only choosing a gives the lower bound so that the supremum over all policies can only be larger. „ \geq “ follows from the

inequality

$$\begin{aligned} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] &= \sum_{a \in \mathcal{A}_s} \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right] \pi_0(a; s) \\ &\leq \max_{a \in \mathcal{A}_s} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \underbrace{\sum_{a \in \mathcal{A}_s} \pi_0(a; s)}_{\leq 1} \\ &= \max_{a \in \mathcal{A}_s} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]. \end{aligned}$$

This proves (i). To prove (ii) we proceed similarly to the proof of Bellman's expectation equation (see the proof of Lemma 3.1.17) and recall that

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \mathbb{E}_{s'}^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \right]$$

holds for any fixed policy π . This implies

$$\begin{aligned} Q^*(s, a) &= \sup_{\pi \in \Pi} Q^\pi(s, a) \\ &= r(s, a) + \sup_{\pi \in \Pi} \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \sup_{\pi \in \Pi} \mathbb{E}_s^{\pi^a} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right]. \end{aligned}$$

Applying Lemma 3.1.16 the proof is complete. There is only one last bit to justify. Why could we interchange the final sum and supremum (generally this only gives an upper bound). To justify let us check that

$$\sup_{\pi: \mathcal{S} \rightarrow \mathbb{R}} \sum_{s \in \mathcal{S}} h(s, \pi(s)) = \sum_{s \in \mathcal{S}} \sup_{\pi: \mathcal{S} \rightarrow \mathbb{R}} h(s, \pi(s))$$

holds. Since „ \leq “ always holds we show the contrary by contradiction. Let us suppose that

$$\sup_{\pi} \sum_s h(s, \pi(s)) < \sum_s \sup_{\pi} h(s, \pi(s)),$$

⁴ and chose $\delta > 0$ such that $\sum_s \sup_{\pi} h(s, \pi(s)) - \delta > \sup_{\pi} \sum_s h(s, \pi(s))$. Next chose some $\pi^*(s)$ and some $\varepsilon > 0$ such that $h(s, \pi^*(s)) > \sup_{\pi} h(s, \pi(s)) - \frac{\varepsilon}{|\mathcal{S}|}$ holds for all $s \in \mathcal{S}$. But then

$$\begin{aligned} \sum_s \sup_{\pi} h(s, \pi(s)) &\leq \sum_s h(s, \pi^*(s)) + \varepsilon \\ &< \sup_{\pi} \sum_s h(s, \pi(s)) + \varepsilon \\ &\leq \sum_s \sup_{\pi} h(s, \pi(s)) + \varepsilon - \delta. \end{aligned}$$

Chosing $\varepsilon < \delta$ this gives a contradiction. □

We now turn towards the second operator of Bellman, the optimality operator. As for the expectation operator we work on $U = \{u : \mathcal{S} \rightarrow \mathbb{R}\}$ but also on the set of all state-action pairs $V := \{v : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}\}$. As long as we assume the Markov decision model to be finite U is nothing but $\mathbb{R}^{|\mathcal{S}|}$ and V is $\mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$. To make U and V Banach spaces we will fix the maximum-norm.

⁴warum nur $\pi : \mathcal{S} \rightarrow \mathbb{R}$? sonst nicht vertauschen von nur endlichen


Definition 3.1.22. (Bellman optimality operators)

For a given Markov decision model we define the following operators:

- (i) The non-linear system of equations

$$v(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) v(s') \right\}, \quad s \in \mathcal{S},$$

is called Bellman optimality equation. The operator $T^* : U \rightarrow U$ defined by

$$(T^*v)(s) := \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) v(s') \right\}, \quad s \in \mathcal{S},$$

is called the Bellman optimality operator (for state-value functions).

- (ii) The non-linear system of equations

$$q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_s} q(s', a'), \quad (s, a) \in \mathcal{S} \times \mathcal{A},$$

is called Bellman state-action optimality equation. The state-action Bellman optimality operator $T^* : V \rightarrow V$ is defined as

$$(T^*q)(s, a) := r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_s} q(s', a'), \quad (s, a) \in \mathcal{S} \times \mathcal{A}.$$

Warning: both optimality operators are denoted by T but are safely distinguished by the number of arguments.

It will turn out that the Bellman optimality operators are directly linked to optimal value functions and solving the corresponding fixed point equations (i.e. the Bellman optimality equations) is equivalent to finding optimal policies. Unfortunately, the equations are non-linear and as such not easy to solve.



Corollary 3.1.23. The optimal value functions (resp. state-action value functions) are fixed points the Bellman optimality operators:

$$T^*V^* = V^* \quad \text{and} \quad T^*Q^* = Q^*.$$

Proof. Both claims follow directly from Lemma 3.1.21 by applying (i) and (ii) in reversed order respectively:

$$V^*(s) = \max_{a \in \mathcal{A}_s} Q^*(s, a) = \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^*(s) \right\} = T^*V^*(s)$$

and

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^*(s) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_s} Q^*(s, a').$$

□

From basic analysis courses you might remember one of the most useful, yet simple, theorems from functional analysis.


Theorem 3.1.24. (Banach fixed-point theorem)

Let $(U, \|\cdot\|)$ be a Banach space, i.e. a complete normed vector space, and $T : U \rightarrow U$ a contraction, i.e. there exists $\lambda \in [0, 1)$ such that

$$\|Tv - Tu\| \leq \lambda \|v - u\|$$

for all $u, v \in U$. Then

- (i) there exists a unique fixed point, i.e. $v^* \in U$ such that $Tv^* = v^*$; and
- (ii) for arbitrary $v_0 \in U$, the sequence $(v_n)_{n \in \mathbb{N}}$ defined by

$$v_{n+1} = Tv_n = T_{n+1}v_0$$

converges to v^* .

The Banach fixed-point is useful because the condition can be checked in many cases and also the algorithm yields a fast converging algorithm. One of the major insights of optimal control is that the Bellman operators are contractions so that the optimal value functions are uniquely determined by the Bellman optimality equations. As a direct consequence there is also a numerical scheme to find the optimal value functions, just iterate the Bellman operator again and again.



Theorem 3.1.25. The optimal value function V^* is the unique solution to the Bellman optimality equation, i.e.

$$V^*(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^*(s') \right\}, \quad s \in \mathcal{S}.$$

Similarly, the optimal state-action value function Q^* is the unique solution to the other Bellman state-action optimality equation, i.e.

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a \in \mathcal{A}_s} Q^*(s', a), \quad (s, a) \in \mathcal{S} \times \mathcal{A}.$$

Proof. Recalling Corollary 3.1.23 it suffices to prove that the Bellman optimality equations have unique fixed points. Using Banach's fixed point theorem this boils down to proving the contraction properties. To deal with the operators a simple inequality from analysis will be used:

$$\left| \max_{a \in \mathcal{A}} f(a) - \max_{a \in \mathcal{A}} g(a) \right| \leq \max_{a \in \mathcal{A}} |f(a) - g(a)|.$$

If the number in the absolute value of the left hand side is positive, then

$$\left| \max_{a \in \mathcal{A}} f(a) - \max_{a \in \mathcal{A}} g(a) \right| \leq \max_{a \in \mathcal{A}} (f(a) - g(a)) \leq \max_{a \in \mathcal{A}} |f(a) - g(a)|.$$

Otherwise, the role of f and g is reversed. With $v, u \in U$, the Bellman optimality operator yields

$$\begin{aligned} \|T^*v - T^*u\|_\infty &= \max_{s \in \mathcal{S}} |T^*v(s) - T^*u(s)| \\ &\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) |v(s') - u(s')| \\ &\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \|v - u\|_\infty \\ &= \gamma \|v - u\|_\infty. \end{aligned}$$

Hence, T^* is a contraction on $U = \{u : \mathcal{S} \rightarrow \mathbb{R}\}$ equipped with the maximum norm and thus has a unique fixed point. Next, we show the same for T^* on $V = \{u : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}\}$: With $v, u \in V$, the second Bellman operator yields

$$\begin{aligned} \|T^*v - T^*u\|_\infty &= \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}_s} |T^*v(s, a) - T^*u(s, a)| \\ &\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} |\gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) (\max_{a' \in \mathcal{A}_s} v(s', a') - \max_{a' \in \mathcal{A}_s} u(s', a'))| \\ &\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_s} |v(s', a') - u(s', a')| \\ &\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{s' \in \mathcal{S}} \max_{a' \in \mathcal{A}_s} |v(s', a') - u(s', a')| \\ &= \gamma \|v - u\|_\infty. \end{aligned}$$

Note that we used the general fact that $|\max a_n - \max b_n| \leq \max_n |a_n - b_n|$. \square



To train the argument please proof that Bellmann's expectation operators are also $\|\cdot\|_\infty$ -contractions.

Uniqueness for Bellmann's optimality equations has important consequences such as the following corollary:



Corollary 3.1.26. A policy $\pi^* \in \Pi$ is optimal if and only if V^{π^*} and/or Q^{π^*} satisfy the Bellman optimality equations from 3.1.22.

This section concludes with the most important piece of understanding of optimal state- and action-value functions and their relation to optimal policies: Given an optimal state- or action-value function, how would one get the good stuff, that is, the optimal policy?



Definition 3.1.27. (Greedy policy)

Given a function $q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the deterministic stationary policy defined by

$$\pi_q(a; s) := \begin{cases} 1 & : a = a^*(s) \\ 0 & : \text{otherwise} \end{cases} \quad \text{with} \quad a^*(s) \in \arg \max_{a \in \mathcal{A}_s} q(s, a)$$

is called a greedy policy with respect to q . Sometimes we also write $\text{greedy}(q)$ instead of π_q .

Keep in mind that the greedy policy is very special, only one action is proposed that maximises the given q -function. In case several actions yield the same state-action value a fixed one of them is chosen. As a consequence we learn how solving the Bellman state-action optimality equation yields an optimal policy.



Theorem 3.1.28. An optimal policy π^* always exist and can always be chosen to be stationary and deterministic! Such a policy is given by solving the Bellman state-action optimality equation and using the greedy policy with respect to the solution.

Alternatively, a solution v to the Bellman optimality equation plugged-into the „ V - Q -transfer operator“

$$q_v(s, a) := r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a)v(s'), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$


yields Q^* and, hence, the greedy optimal policy (compare Lemma 3.1.21).

Proof. Suppose we have a solution q of Bellman's state-action optimality equation. By uniqueness q equals Q^* . If π is the greedy policy obtained from q (thus of Q^*), then the definition of greedy policies yields


$$V^*(s) \stackrel{3.1.21}{=} \max_{s \in \mathcal{A}_s} Q^*(s, a) = V^\pi(s).^5$$

Thus, π is optimal by definition of optimality. \square

Since this is the basic result on which many ideas of reinforcement learning are based let us summarise the findings of this chapter:

 Solving Bellman's optimality equation (or state-value optimality equation) yields a stationary deterministic policy π^* as the greedy policy obtained from the solution. Thus, all algorithms that approximatively solve the Bellman optimality equation give rise to approximation algorithms that find an optimal policy of the stochastic optimal control problem $\max_{\pi \in \Pi} V^\pi$.

As the remark indicates we will be interested in learning the optimal policies by approximation. Recalling the setting of stochastic bandits (interpreted as one-step MDP) this sounds familiar. In that setting an optimal policy is nothing but a dirac measure on optimal arms, a learning strategy a sequence of policies that learns (approximates) the optimal policy. To make the connection to stochastic bandits let us define the notation of a learning strategy:


 **Definition 3.1.29.** A sequence $(\pi^n)_{n \in \mathbb{N}}$ of policies for a Markov decision model is called a learning strategy, if π^{n+1} only depends on everything seen for the first n rounds of learning.

We keep the definition extremely vague as it will not play any further role in what follows. The aim is to find algorithms that produce learning strategies that converge quickly and efficiently to the optimal strategy π^* . What we mean by convergence will depend on the context, the minimal requirement is convergence of the value function to the optimal value function, i.e. $\|V^{\pi^n} - V^*\|_\infty \rightarrow 0$. In contrast to stochastic bandit theory the regret plays no role in reinforcement learning, the situation is too complex.

There are two typical approaches that we will encounter in different setups:

- value function based learning,
- policy based learning.

For value function the idea is to learn the optimal value function V^* (or optimal state-value function Q^*) and then infer the optimal policy π^* by taking argmax (the greedy policy from Theorem 3.1.28). In contrast, policy learning tries to approximate directly the optimal policy π^* .

 The algorithms presented below are called **dynamic programming algorithms**, they belong to a large class of algorithms that break down a problem into (hopefully simpler) subproblems. Here, this means for instance to compute $V^\pi(s)$ from all other $V^\pi(s')$. Since for discounted MDP problems the subproblems do not immediately simplify we do not go further into the ideas of dynamic programming but still refer to all algorithms based on Bellman's equations as dynamic programming algorithms.

Lecture 9

3.2 Basic value iteration algorithm

We have understood the problem and its ingredients and have learned that in order to act optimally, that is to know the optimal policy, one only needs to know the optimal value function (or the optimal state-action value function). The goal is now to develop methods to derive the

optimal value function. As seen in the previous section, the Banach fixed point theorem gives not only the existence of a unique fixed point of a contraction mapping T but also the convergence of the sequence $(T^{n+1}v_0)_{n \in \mathbb{N}_0}$ to that fixed point for arbitrary v_0 . We learned that for T^* the unique fixed point corresponded to V^* . The idea of value iteration is to use this convergence property and turn the Bellman optimality equation into an update procedure.

Algorithm 7: Value iteration

Data: $\varepsilon > 0$

Result: Approximation $V \approx V^*$

Initialize $V \equiv 0, V_{\text{new}} \equiv 0$

$\Delta := \varepsilon \frac{(1-\gamma)}{\gamma}$

while $\Delta > \varepsilon \frac{(1-\gamma)}{2\gamma}$ **do**

for $s \in \mathcal{S}$ **do**

$V(s) = V_{\text{new}}(s)$

end

for $s \in \mathcal{S}$ **do**

$$V_{\text{new}}(s) = \max_{a \in \mathcal{A}} \left\{ r(s, a) + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V(s')}_{(T^*V)(s)} \right\}$$

end

$\Delta := \max_{s \in \mathcal{S}} (|V_{\text{new}}(s) - V(s)|)$

end

return V_{new}



Theorem 3.2.1. The value iteration Algorithm 7 terminates and $\|V - V^*\|_\infty \leq \frac{\varepsilon}{2}$.

Proof. Suppose the sequence of vectors $v_{n+1} = T^*v_n$ is obtained by iteratively applying Bellman's optimality operator and v^* is the limit. The finite-time termination of the algorithm follows immediately from Banach's fixed point theorem as

$$\|v_n - v_{n+1}\|_\infty \stackrel{\Delta}{\leq} \|v_n - v^*\|_\infty + \|v^* - v_{n+1}\|_\infty \rightarrow 0$$

for $n \rightarrow \infty$. Now suppose the algorithm terminated after n steps, i.e. $V = v_n$ and $\|v_n - v_{n-1}\|_\infty < \varepsilon \frac{(1-\gamma)}{2\gamma}$. Using the contraction property of T^* yields, for $m \in \mathbb{N}$,

$$\begin{aligned} \|v_n - v_{n+m}\|_\infty &\stackrel{\Delta}{\leq} \sum_{k=0}^{m-1} \|v_{n+k} - v_{n+k+1}\|_\infty \\ &= \sum_{k=0}^{m-1} \|(T^*)^k v_n - (T^*)^k v_{n+1}\|_\infty \\ &\leq \sum_{k=0}^{m-1} \gamma^k \|v_n - v_{n+1}\|_\infty. \end{aligned}$$

Using continuity of the norm yields

$$\begin{aligned} \|V - V^*\|_\infty &= \lim_{m \rightarrow \infty} \|v_n - v_{n+m}\|_\infty \\ &\leq \lim_{m \rightarrow \infty} \sum_{k=0}^{m-1} \gamma^k \|v_{n+1} - v_n\|_\infty \\ &= \frac{\gamma}{1-\gamma} \|v_{n+1} - v_n\|_\infty \leq \frac{\gamma}{1-\gamma} \|v_n - v_{n-1}\|_\infty. \end{aligned}$$

Now the termination condition implies the claim. \square

The approximate value function from Algorithm 7 is clearly not equal to the optimal value function V^* . Hence, the effect on transferring to a policy needs to be analysed.



Definition 3.2.2. For $\varepsilon > 0$ a policy $\pi \in \Pi$ is called ε -optimal if

$$V^*(s) \leq V^\pi(s) + \varepsilon$$

for all $s \in \mathcal{S}$.

Now recall from Lemmas 3.1.21 and 3.1.17 how transfer value-functions into state-value functions. For both the optimal value functions and the value functions for a given policy the vector V is transferred into the Q -matrix as

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a)V(s'). \quad (3.7)$$

We now do the same and use (3.7) to define from the approximation V of V^* an approximate Q -function. Then the corresponding greedy policy is ε -optimal:



Theorem 3.2.3. The greedy policy π obtained from Q is ε -optimal.

Proof. The main point of the argument is a relation of optimality and expectation operator in the case of greedy policies. To emphasise the importance, let us highlight the trick once more:



Bellman's optimality and expectation operators are closely connected for greedy policies!

The crucial computation links the optimality operator with the expectation operator of the greedy policy:

$$\begin{aligned} T^*V(s) &= \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a)V(s') \right\} \\ &\stackrel{V=v_n}{=} \max_{a \in \mathcal{A}_s} \{q_V(s, a)\} \\ &= \sum_{a \in \mathcal{A}_s} \pi(a; s) \mathbf{1}_{a=\arg \max q_V(a')} (q_V(a, s)) \\ &= \sum_{a \in \mathcal{A}_s} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a)V(s') \right) \\ &= T^\pi V(s) \end{aligned}$$

The rest of the proof is straight forward using fixed points and the contraction property. Suppose the algorithm terminated after n steps, i.e. $V = v_n$ and $\|v_n - v_{n-1}\| < \varepsilon \frac{(1-\gamma)}{2\gamma}$. The identity above yields

$$\begin{aligned} \|V^\pi - V\|_\infty &= \|T^\pi V^\pi - V\|_\infty \\ &\leq \|T^\pi V^\pi - T^*V\|_\infty + \|T^*V - V\|_\infty \\ &= \|T^\pi V^\pi - T^\pi V\|_\infty + \|T^*V - T^*v_{n-1}\|_\infty \\ &\leq \gamma \|V^\pi - V\|_\infty + \gamma \|V - v_{n-1}\|_\infty. \end{aligned}$$

Rearranging this equation yields

$$\|V^\pi - V\|_\infty \leq \frac{\gamma}{1-\gamma} \|v_n - v_{n-1}\|_\infty.$$

In the proof of the previous theorem we have already shown that

$$\|V - V^*\|_\infty \leq \frac{\gamma}{1-\gamma} \|v_n - v_{n+1}\|_\infty \leq \frac{\gamma}{1-\gamma} \|v_n - v_{n-1}\|_\infty.$$

Finally, using the terminal condition of Algorithm 7 yields

$$\|V^\pi - V^*\|_\infty \leq \|V^\pi - V\|_\infty + \|V - V^*\|_\infty \leq 2\frac{\gamma}{1-\gamma} \|V - v_n\|_\infty \leq \varepsilon.$$

This proves the claim. \square

We next analyse the convergenc rates of the value iteration algorithm.



Definition 3.2.4. Suppose $(V, \|\cdot\|)$ is a Banach space. For a sequence (y_n) in V with limit y^* we say the convergence is of order $\alpha > 0$ if there exists a constant $K < 1$ for which

$$\|y_{n+1} - y^*\| \leq K \|y_n - y^*\|^\alpha, \quad n \in \mathbb{N}. \quad (3.8)$$

Linear convergence corresponds to $\alpha = 1$.

Since the algorithm is based on Banach's fixed point theorem it is clear that the convergence is at least linear. A simple initialisation shows that the convergence generally cannot be improved.



Theorem 3.2.5. For all initialisations the convergence in Algorithm 7 (without termination) is linear with constant $K = \gamma$. There is an initialisation for which the speed of convergence is exactly linear.

Proof. Let us denote again v_n for the n th update of the iteration $v_n = T^*v_{n-1}$. The linear convergence rate follows directly from the fixed point property of T^* :

$$\|v_{n+1} - V^*\|_\infty = \|T^*v_n - T^*V^*\|_\infty \leq \gamma \|v_n - V^*\|_\infty, \quad n \in \mathbb{N}. \quad (3.9)$$

In fact, the rate cannot be better as the following example shows. If Algorithm 7 is initialised with $v_0 := V^* + k\mathbf{1}$, it holds that

$$\begin{aligned} \|v_1 - V^*\|_\infty &= \|T^*v_0 - T^*V^*\|_\infty = \|T^*(V^* + k\mathbf{1}) - T^*V^*\|_\infty \\ &\stackrel{\text{Def. } T^*}{=} \|V^* + \gamma k\mathbf{1} - T^*V^*\|_\infty \\ &= \|V^* + \gamma k\mathbf{1} - V^*\|_\infty \\ &= \gamma \|(V^* + k\mathbf{1}) - V^*\|_\infty = \gamma \|v_0 - V^*\|_\infty. \end{aligned}$$

An induction shows that for this example $\|v_{n+1} - V^*\|_\infty = \gamma \|v_n - V^*\|_\infty$ for all $n \in \mathbb{N}$. \square



Of course we could equally use iterations obtained from Banach's fixed point theorem to directly approximate Q^* instead of approximating V^* and then transferring to Q^* . This will be done later for approximate dynamic programming (Q -learning and SARSA) in which we do not assume explicit knowledge on the transition function, thus, cannot transfer from V to Q . In the explicit case it is more reasonable to work with vector iterations than matrix iterations and only transfer from V to Q once.

3.3 Basic policy iteration algorithm

In this section we want to explore another method of reaching an optimal value function and hence an optimal policy. The method is not part of the classical methodology of stochastic control but much more common in the reinforcement learning community as it motivates some of the most powerful approximation algorithms. The idea goes as follows. Iterate between the following two steps:

- Policy evaluation: Compute or estimate Q (or V) for the currently best known policy π .
- Policy improvement: Improve the best known policy by taking $\pi' = \text{greedy}(Q)$.

In the following both steps will be discussed separately and then alternated for the policy iteration algorithm.

3.3.1 Policy evaluation

We are now going to address the question on how to compute $V^\pi(s)$ for a given policy π . There are essentially three direct ideas that one might come up with:

- approximate the expectation by Monte Carlo,
- solve the (linear) Bellman expectation equation by matrix using linear algebra (e.g. matrix inversion),
- find the fixed point of the Bellman expectation operator using Banach's fixed point iteration.

The first idea and subtle variants will be topic of the next chapter, the latter two approaches will be address below.

Recall the Bellman expectation operator for a stationary policy $\pi \in \Pi_s$:

$$(T^\pi u)(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) u(s') \right), \quad s \in \mathcal{S}.$$

We know from Lemma 3.1.19 that the value function V^π is a fixed point of the Bellman operator T^π .



The one-step reward $r(s, a)$ is a shortcut for $r(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r p(s', r; s, a)$, sometimes we prefer to write the detailed form in order to emphasise more explicitly the dependence on the Markov decision model. The Bellman operator then takes the form

$$(T^\pi u)(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r; s, a) [r + \gamma u(s')], \quad s \in \mathcal{S}.$$

The equation looks complicated but (\mathcal{S} is assumed finite) is just a system of linear equations that can be solved by means of linear algebra techniques. This becomes directly evident when we rewrite the fixed point equation in vector notation

$$V^\pi = r^\pi + \gamma P^\pi V^\pi,$$

where

$$P^\pi = \left(\sum_{a \in \mathcal{A}_s} \pi(a; s) p(s'; s, a) \right)_{(s, s') \in \mathcal{S} \times \mathcal{S}}$$

$$r^\pi = \left(\sum_{a \in \mathcal{A}_s} \pi(a; s) r(s, a) \right)_{s \in \mathcal{S}}$$

with $P_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ and $r^\pi, V^\pi \in \mathbb{R}^{|\mathcal{S}|}$.



Please check that indeed $T^\pi = r^\pi + \gamma P^\pi$.

Given different use cases, each of these notations may be favorable. The reader may forgive our shifty notation in favor of uncluttered statements.



Proposition 3.3.1. The linear equation $V = r^\pi + \gamma P^\pi V$ has a unique solution. Hence, V^π can be computed explicitly as

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi. \quad (3.10)$$

Proof. This follows immediately as the Bellman expectation operator is a contraction (compare the exercise after Theorem 3.1.25). To compute the solution is straightforward linear algebra operation:

$$\begin{aligned} V^\pi = r^\pi + \gamma P^\pi V^\pi &\Leftrightarrow (I - \gamma P^\pi) V^\pi = r^\pi \\ &\Leftrightarrow V^\pi = (I - \gamma P^\pi)^{-1} r^\pi \end{aligned}$$

The existence of the inverse is guaranteed as the linear equation has a unique solution. \square

As a consequence we see that the evaluation of a policy simply corresponds to the inversion of a matrix. However, although this computation is straightforward, it is a tedious and expensive computation. The complexity of matrix inversion (using Gauss-Jordan elimination) is $\mathcal{O}(n^3)$ and $n = |\mathcal{S}|$ can be huge. Thus we will also explore iterative solution methods.

As seen with value iteration we can use the convergence properties of the iterates of a contraction mapping. We want to do the same for the operator T^π for a stationary policy. Using Banach's fixed point theorem convergence $(T^\pi)^n v_0 \rightarrow V^\pi$ holds for any initialisation v_0 . Implemented as an algorithm we obtain Algorithm 8.

Algorithm 8: Iterative policy evaluation (Naive)

Data: Policy $\pi \in \Pi_{\mathcal{S}}$, $\varepsilon > 0$

Result: Approximation $V \approx V^\pi$

Initialize $V \equiv 0, V_{\text{new}} \equiv 0$

$\Delta := 2\varepsilon \frac{(1-\gamma)}{\gamma}$

while $\Delta > \varepsilon \frac{(1-\gamma)}{\gamma}$ **do**

for $s \in \mathcal{S}$ **do**

$V(s) = V_{\text{new}}(s)$

end

for $s \in \mathcal{S}$ **do**

$V_{\text{new}}(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r; s, a) [r + \gamma V(s')]$

end

$\Delta := \max_{s \in \mathcal{S}} (|V_{\text{new}}(s) - V(s)|)$

end

$V := V_{\text{new}}$



Theorem 3.3.2. Algorithm 8 terminates and $\|V - V^\pi\|_\infty < \varepsilon$.

Proof. The proof is identical to the proof of Theorem 3.2.1 which is only based on the fact that T^* is a contraction with constant γ . Since T^π is equally a contraction with constant γ the same result holds. \square

Algorithm 8 can be improved in terms of memory. The simple fixed point iteration algorithm requires to occupy memory for $2|\mathcal{S}|$ values since V has to be fully stored in order to compute every value $V_{new}(s)$. This can be done more efficient by directly using available data, see Algorithm 9. The algorithm does not perform the matrix computation with T^π directly but row by row, it updates coordinate by coordinate instead of all coordinates at once.

Algorithm 9: Iterative policy evaluation (totally asynchronous updates)

Data: Policy $\pi \in \Pi_S$, $\varepsilon > 0$

Result: Approximation $V \approx V^\pi$

Initialize $V(s) = 0$ for all $s \in \mathcal{S}$

$\Delta := 2\varepsilon$

while $\Delta > \varepsilon$ **do**

$\Delta := 0$

for $s \in \mathcal{S}$ **do**

$v := V(s)$

$V(s) := \sum_{a \in \mathcal{A}_s} \pi(a; s) \underbrace{\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r; s, a) [r + \gamma V(s')]}_{T^\pi V(s) = (r_\pi + P_\pi)V(s)}$

$\Delta := \max(\Delta, |v - V(s)|)$

end

end



Try to prove convergence of the totally asynchronous policy evaluation algorithm (without termination) to V^π . To do label \mathcal{S} by s_1, \dots, s_K and define

$$T_s^\pi V(s') = \begin{cases} T^\pi V(s) & : s = s' \\ V(s) & : s \neq s' \end{cases}$$

i.e. T^π is only applied to coordinate s while leaving all other coordinates unchanged. Then the inner loop of the algorithm performs nothing but the composition $\bar{T}^\pi := (T_{s_K}^\pi \circ \dots \circ T_{s_1}^\pi)(V)$, which is not the same as applying T^π ! Show that V^π is a fixed point of the composition and the composition is a contraction on $(U, \|\cdot\|_\infty)$, proceed step by step using the estimates to show that Bellman operators are contractions. Without the termination the outer loop is nothing but an iteration of \bar{T}^π , hence, without termination the algorithm converges to V^π .

Algorithms in which coordinates s are treated differently are called asynchronous algorithms. In fact, there are other versions of the algorithm where coordinates are not swept in order but randomly. We will come back to such asynchronous algorithms in the next chapter (e.g. Q -learning is of exactly that kind replacing Bellman's expectation operator by the state-action optimality operator).

Lecture 10

3.3.2 Policy improvement

We now aim to improve a given policy, that is to slightly change it such that its value function takes larger values. Mathematically speaking, for a policy $\pi \in \Pi$ and value function V^π we aim to find a policy $\pi' \in \Pi$ such that

$$V^\pi(s) \leq V^{\pi'}(s), \quad \forall s \in \mathcal{S},$$

and

$$V^\pi(s) < V^{\pi'}(s) \quad \text{for at least one } s \in \mathcal{S}.$$

If π is not optimal, there always exists such a policy, e.g. the optimal policy. We now want to define a procedure to update a non-optimal policy to an improved policy. The key idea is to change the policy at a single state $s \in \mathcal{S}$ to a particular action. For this we look at the action-value function of a stationary policy $\pi \in \Pi_{\mathcal{S}}$. Recalling from Lemma 3.1.17

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^{\pi}(s, a)$$

it becomes apparent that

$$\max_{a \in \mathcal{A}_s} Q^{\pi}(s, a) \geq V^{\pi}(s). \quad (3.11)$$

In other words, the inequality above implies that choosing an action $a \in \mathcal{A}$ that maximizes the expected reward in state s and the expected future value of following a policy π is at least as good as following the policy π . Recalling Definition 3.1.27 this suggests to use the greedy policy $\pi_{Q^{\pi}}$ induced by the Q -function of the current policy π . This then leads to the simple policy improvement algorithm. The improvement of the greedy policy improvement is a special case of

Algorithm 10: Greedy policy improvement

Data: policy $\pi \in \Pi_{\mathcal{S}}$, Q -function Q^{π}
Result: improved policy $\pi' = \text{greedy}(Q^{\pi})$
 $\pi' := \pi$
for $s \in \mathcal{S}$ **do**
 Choose $a^*(s) \in \arg \max_{a \in \mathcal{A}_s} Q^{\pi}(s, a)$
 $\pi'(a^*(s); s) := 1$
 for $a \in \mathcal{A} \setminus \{a^*(s)\}$ **do**
 $\pi'(a; s) = 0$
 end
end

the policy improvement theorem.



Theorem 3.3.3. (Policy improvement theorem)

Let $\pi, \pi' \in \Pi_{\mathcal{S}}$ be two stationary policies and define

$$Q^{\pi}(s, \pi'(s)) := \sum_{a \in \mathcal{A}_s} \pi'(a; s) Q^{\pi}(s, a).$$

(i) If

$$V^{\pi}(s) \leq Q^{\pi}(s, \pi'(s)), \quad \forall s \in \mathcal{S}, \quad (3.12)$$

then π' is an improvement of π , i.e.

$$V^{\pi}(s) \leq V^{\pi'}(s), \quad \forall s \in \mathcal{S}. \quad (3.13)$$

(ii) If there is a strict inequality in (3.12) for some state s then there must be a strict inequality in (3.13) at that same state s .

In particular, for every policy $\pi \in \Pi_{\mathcal{S}}$ the greedy policy obtained from Q^{π} improves π .

For a better understanding let's spell out the condition of the policy improvement lemma. The condition means that changing the policy π only in the very first step to the action probabilities of π' then the value function is improved by changing from π to π' . Hence, on a heuristic level the theorem is trivial: if a stationary policy is better for one step (in all states) then (by the

Markov property) it will also lead to a larger reward in all future time-steps. But then the value function is clearly larger.

Proof. For the proof we use 3.1.17 to push the assumed one-step improvement successively further:

$$\begin{aligned}
V^\pi(s) &= \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^\pi(s, a) \\
&\leq \sum_{a \in \mathcal{A}_s} \pi'(a; s) Q^\pi(s, a) \\
&\stackrel{\text{Lemma}}{=} \sum_{a \in \mathcal{A}_s} \pi'(a; s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^\pi(s') \right] \\
&= \mathbb{E}_s^{\pi'} [R_1] + \gamma \mathbb{E}_s^{\pi'} [V^\pi(S_1)],
\end{aligned}$$

where the final equality stems from the definition of the (discrete) expectation and the MDP path probability $\mathbb{P}_s^{\pi'}(S_1 = s') = \sum_{a \in \mathcal{A}_s} \pi'(a; s) p(s'; s, a)$. Proceeding as above yields

$$\begin{aligned}
\mathbb{E}_s^{\pi'} [V^\pi(S_1)] &= \sum_{s' \in \mathcal{S}} \mathbb{P}_s^{\pi'}(S_1 = s') V^\pi(s') \\
&\leq \sum_{s' \in \mathcal{S}} \mathbb{P}_s^{\pi'}(S_1 = s') (\mathbb{E}_{s'}^{\pi'} [R_1] + \gamma \mathbb{E}_{s'}^{\pi'} [V^\pi(S_1)]) \\
&= \mathbb{E}_s^{\pi'} [R_2] + \gamma \mathbb{E}_s^{\pi'} [V^\pi(S_2)].
\end{aligned}$$

Let us check the first summand, the computation for the second is similar:

$$\begin{aligned}
\mathbb{E}_s^{\pi'} [R_2] &= \sum_{r \in \mathcal{R}} r \mathbb{P}_s^{\pi'}(R_2 = r) \\
&= \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} r \mathbb{P}_s^{\pi'}(R_2 = r | S_1 = s') \mathbb{P}_s^{\pi'}(S_1 = s') \\
&\stackrel{(*)}{=} \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} r \mathbb{P}_s^{\pi'}(R_1 = r) \mathbb{P}_s^{\pi'}(S_1 = s') \\
&= \sum_{s' \in \mathcal{S}} \mathbb{E}_{s'}^{\pi'} [R_1] \mathbb{P}_s^{\pi'}(S_1 = s').
\end{aligned}$$

To be very careful note that the sum is only over those s' for which the probability is positive so that there is no problem in the conditioning. The justification of (*) is the usual computation with path probabilities (showing by cancellations that both probabilities equal $\sum_{a \in \mathcal{A}_s} p(\mathcal{S} \times \{r\}; s', a) \pi(a; s')$). Continuing inductively (the notation becomes increasingly unpleasant) yields

$$V^\pi(s) \leq \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi'} [R_{t+1}] \stackrel{\text{DCT}}{=} V^{\pi'}(s).$$

For strict inequalities the above steps imply strict inequalities in (3.13).

Checking the greedy policy update satisfies the general condition is easy. As defined in Algorithm 10, with $a^*(s) \in \arg \max_{a \in \mathcal{A}_s} Q^\pi(s, a)$, the greedy policy π' satisfies the improvement condition:

$$\sum_{a \in \mathcal{A}_s} \pi'(a; s) Q^\pi(s, a) = Q^\pi(s, a^*(s)) = \max_{a \in \mathcal{A}_s} Q^\pi(s, a) \geq \sum_{a \in \mathcal{A}_s} \pi(a; s) Q^\pi(s, a) = V^\pi(s)$$

□

For the following section we also want to prove the following lemma that links greedy policy improvement to optimal policies. After all, this is still what we are on the look for.



Lemma 3.3.4. Let $\pi \in \Pi_S$ and π' the greedy policy obtained from Q^π , then

$$V^\pi = V^{\pi'} \text{ (or } Q^\pi = Q^{\pi'}) \implies \pi \text{ and } \pi' \text{ are optimal.}$$

Proof. It follows from Lemma 3.1.17 that equality of the value functions implies equality of the state-action value functions, hence, we work with Q . As in the previous proof we compute, using $a^*(s) \in \arg \max_{a \in \mathcal{A}_s} Q^\pi(s, a)$,

$$\begin{aligned} Q^{\pi'}(s, a) &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi'(a'; s) Q^{\pi'}(s', a') \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) Q^{\pi'}(s', a^*(s)). \end{aligned}$$

Now suppose that $Q^\pi = Q^{\pi'}$, then the equation becomes

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q^\pi(s', a').$$

Hence, Q^π and $Q^{\pi'}$ both solve Belman's state-action optimality equation. Since this has a unique solution we deduce $Q^* = Q^\pi$ (and both are optimal). \square



Corollary 3.3.5. For a non-optimal policy $\pi \in \Pi_S$ and the greedy policy $\pi' = \text{greedy}(Q^\pi)$ obtained from Q^π it holds that there exists some $s \in \mathcal{S}$ such that

$$V^\pi(s) < V^{\pi'}(s).$$

Proof. The claim follows directly by Lemma 3.3.4. \square

Apart from the greedy policies there is a second interesting application of the policy improvement theorem. For that sake we need a bit of extra notation.



Definition 3.3.6. A policy $\pi \in \Pi_S$ is called

- soft, if it fulfils

$$\pi(a; s) > 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s,$$

- ε -soft for some $1 \geq \varepsilon > 0$, if it fulfils

$$\pi(a; s) > \frac{\varepsilon}{|\mathcal{A}_s|} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s,$$

- ε -greedy with regard to Q , if it selects the greedy action with respect to Q with probability $(1 - \varepsilon)$ and a (uniform) random action with probability ε , i.e.

$$\pi(a; s) = \begin{cases} (1 - \varepsilon) + \frac{\varepsilon}{|\mathcal{A}_s|} & : a = a^*(s) \\ \frac{\varepsilon}{|\mathcal{A}_s|} & : a \text{ is not greedy} \end{cases},$$

where $a^*(s) \in \arg \max_a Q(s, a)$.

Let us recall the discussion of the ε -greedy learning strategies for stochastic bandits. Such policies are considered suboptimal as they lead to linear regret, they play suboptimal arms with probability ε . Similarly, ε -soft policies cannot be optimal as suboptimal actions must be played in contrast to greedy policies that only play optimal actions. Hence, ε -greedy policies will typically not improve policies, but they do improve all other ε -soft policies:



Proposition 3.3.7. If $\pi \in \Pi_S$ is ε -soft, then the ε -greedy policy π' obtained from Q^π improves π .

Proof. This follows by checking the condition from the policy improvement theorem:

$$\begin{aligned}
 V^\pi(s) &= \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{a \in \mathcal{A}_s} Q^\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{a \in \mathcal{A}_s} Q^\pi(s, a) + \sum_a \pi(a; s) Q^\pi(s, a) \\
 &= \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{a \in \mathcal{A}_s} Q^\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} Q^\pi(s, a) \\
 &\leq \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{a \in \mathcal{A}_s} Q^\pi(s, a) + (1 - \varepsilon) \max_a Q^\pi(s, a) \\
 &= \sum_{a \in \mathcal{A}_s} \pi'(a; s) Q^\pi(s, a).
 \end{aligned}$$

□

3.3.3 Exact and generalised policy iteration algorithms

The ingredients developed above can now be combined to obtain the policy iteration algorithm. The idea is simple: alternate policy evaluation and improvement, compute V^π and then improve to π' using the greedy strategy obtained from Q^π . The above results show that every improvement step improves the policy and the limit of the procedure is π^* . Here is an illustration of the procedure:

$$\pi_0 \nearrow V^{\pi_0} \searrow \pi_1 \nearrow V^{\pi_1} \searrow \pi_2 \nearrow \dots \searrow \pi^*$$

The algorithm obtained this way is called policy iteration algorithm. Since there are many variations how to perform the policy evaluation (exact or approximatedly) we will meet several variants in the chapters below. We will first restrict ourselves to exact evaluations of V^π using

Algorithm 11: Greedy exact policy iteration

Data: initial policy $\pi \in \Pi_S$, initial value function V

Result: optimal policy $\pi^* \in \Pi_S^D$

initialise arbitrarily $V_{\text{new}}, \pi_{\text{new}}$

stop = *False*

while stop = *False* **do**

 Policy evaluation: Obtain V^π by computing (3.10).

 set $Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r; s, a) [r + \gamma V^\pi(s')]$ for all a, s

 Policy improvement: Obtain the improved greedy policy π_{new} from Q^π

if $Q^{\pi_{\text{new}}} = Q^\pi$ **then**

 | stop = *True*

end

end

return $\pi^* = \pi$

the matrix inversion (3.10).



Theorem 3.3.8. (Greedy exact policy iteration)

Started in any policy the policy iteration Algorithm 17 with exact policy evaluation and greedy policy update for finite MDPs terminates in a finite number of iterations (at most $|\mathcal{A}| \cdot |\mathcal{S}|$) with a solution of the optimality equation and an optimal policy π^* .

Proof. By Corollary 3.3.5 in each iteration there is a strict improvement of the next policy π' (i.e. there exists at least one $s \in \mathcal{S}$ such that $V^\pi(s) < V^{\pi'}(s)$) and the set of deterministic stationary strategies is finite ($|\Pi_{\mathcal{S}}^D| \leq |\mathcal{S}|^{|\mathcal{A}|} < \infty$) the algorithm has to terminate in finitely many steps. By Lemma 3.3.4 the termination of the algorithm, thus $V^\pi = V^{\pi'}$, implies that π' is optimal. \square

Also for infinite state and action space the exact policy iteration algorithm converges monotonically and in norm to the optimal policy. Since the exact policy evaluation is hardly possible if \mathcal{S} and/or \mathcal{A} are huge we do not go further into the analysis



Typically the value functions V^{π_n} will not be computed explicitly but the policy iteration algorithm will be used in an approximate manner where V^π is estimated. One example is to replace the explicit evaluation of V^π by a few steps of the Banach fixed point iteration corresponding to T^π , compare Algorithms 8 or 9. Other examples will be discussed in Chapter 4. All algorithms alternating between value function estimation a policy improvement (not necessarily greedy)

$$\pi_0 \nearrow \underbrace{\hat{V}^{\pi_0}}_{\approx V^{\pi_0}} \searrow \pi_1 \nearrow \underbrace{\hat{V}^{\pi_1}}_{\approx V^{\pi_1}} \searrow \pi_2 \nearrow \dots \searrow \pi$$

will be called **generalised policy iteration algorithm**. The aim will be to generate algorithms that converge as quickly as possible to a policy π which is as close to π^* as possible.

Algorithm 12: Generalised policy iteration paradigm

Data: initial policy π

Result: optimal policy π^* (or approximation of π^*)

while *not converged* **do**

Policy evaluation: Obtain estimates for \hat{V}^π and/or \hat{Q}^π from **some** algorithm.

Policy improvement: Obtain (hopefully improved) policy π_{new} by **some** algorithm.

Set $\pi = \pi_{\text{new}}$.

end

return π

It is not clear at all if a generalised policy iteration algorithm converges to an optimal policy and when to stop the algorithm! The errors made by policy evaluation and improvement might easily build up. It is a very non-trivial task to find and tune approximate algorithms that converge. In Section 4.2.1 will come back to generalised policy iteration with ε -greedy policies.



The notion "while not converged" in algorithm pseudo code will always refer to a non-specified termination condition. In many algorithms we will specify a concrete termination condition.

An interesting version of generalised policy iteration comes in combination with ε -greedy policies that will be useful later to introduce exploitation into some algorithms. Let us extend the notion of optimality to the class of soft policies.



Definition 3.3.9. An ε -soft policy π^* is called ε -soft optimal if

$$V^{\pi^*}(s) = \sup_{\pi \text{ } \varepsilon\text{-soft}} V^\pi(s) =: \tilde{V}^*(s), \quad \forall s \in \mathcal{S}.$$

There is a nice trick how to show convergence of ε -greedy policy iteration. Since ε -greedy policies play some action with probability ε they will never be optimal (except in trivial cases) so they won't converge to an optimal policy. Nonetheless, they do converge to a policy which is optimal among the ε -soft policies.


Theorem 3.3.10. (ε -greedy exact policy iteration)

Started in any ε -soft policy the generalised policy iteration algorithm with exact policy evaluation and ε -greedy policy update converges to an ε -soft optimal policy.

Proof. To prove convergence of exact policy iteration with greedy policy update (Theorem 3.3.8) used the Bellman equation to guarantee that no further improvement means that the current policy is already optimal. This does not work for the ε -greedy policy iteration algorithm. To circumvent this problem we use a trick and “move the ε -softness into the environment”. For that sake let us define a new MDP with transition function

$$\tilde{p}(s', r; s, a) := (1 - \varepsilon)p(s', r; s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} p(s', r; s, b).$$

This means, that with probability ε , the transition kernel will ignore the selected action and behave as if a uniformly random action was chosen. We can transform stationary ε -soft policies π from the old MDP to stationary policies $\tilde{\pi}$ of the new MDP via

$$\tilde{\pi}(a; s) := \frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} \geq 0.$$

Let us denote the mapping by $f : \pi \mapsto \tilde{\pi}$. Conversely, for every stationary policy $\tilde{\pi}$ of the transformed MDP we can define the ε -soft policy π by

$$\pi(a; s) := (1 - \varepsilon)\tilde{\pi}(a; s) + \frac{\varepsilon}{|\mathcal{A}_s|},$$

which is the inverse mapping. Therefore f is a bijection between the ε -soft policies in the old MDP and all stationary policies in the new MDP. We now show, that the value functions V^π stay invariant with regard to this mapping. For that sake note that

$$\tilde{p}(s'; s, a) = (1 - \varepsilon)p(s'; s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} p(s'; s, b)$$

and

$$\tilde{r}(s, a) = (1 - \varepsilon)r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} r(s, b),$$

hence,

$$\begin{aligned} \sum_{a \in \mathcal{A}_s} \tilde{\pi}(a; s) \tilde{r}(s, a) &= \sum_{a \in \mathcal{A}_s} \left(\frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} \right) \left((1 - \varepsilon)r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} r(s, b) \right) \\ &= \sum_{a \in \mathcal{A}_s} \left(\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|} \right) r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} r(s, b) \underbrace{\sum_{a \in \mathcal{A}_s} \frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon}}_{=1} \\ &= \sum_{a \in \mathcal{A}_s} \pi(a; s) r(s, a). \end{aligned}$$

Similarly:

$$\begin{aligned} \sum_{a \in \mathcal{A}_s} \tilde{\pi}(a; s) \tilde{p}(y; s, a) &= \sum_{a \in \mathcal{A}_s} \left(\frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} \right) \left((1 - \varepsilon)p(y; s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} p(y; s, b) \right) \\ &= \sum_{a \in \mathcal{A}_s} \pi(a; s) p(y; s, a) \end{aligned}$$

Combining the above yields

$$\begin{aligned}\tilde{T}^{\tilde{\pi}}V^\pi(s) &= \sum_{a \in \mathcal{A}_s} \tilde{\pi}(a; s) \left[\tilde{r}(s, a) + \gamma \sum_{y \in \mathcal{S}} \tilde{p}(y; s, a) V^\pi(y) \right] \\ &= \sum_{a \in \mathcal{A}_s} \pi(a; s) \left[r(s, a) + \gamma \sum_{y \in \mathcal{S}} p(y; s, a) V^\pi(y) \right] \\ &= T^\pi V^\pi(s) = V^\pi(s).\end{aligned}$$

Since the fixed point is unique it follows that $\tilde{V}^{\tilde{\pi}} = V^\pi$ and, as f is bijective,

$$\sup_{\tilde{\pi} \in \tilde{\Pi}_S} \tilde{V}^{\tilde{\pi}}(s) = \sup_{\pi \in \Pi_S} V^\pi(s), \quad (3.14)$$

For the Q -functions we obtain

$$\begin{aligned}\tilde{Q}^{\tilde{\pi}}(s, a) &= \tilde{r}(s, a) + \gamma \sum_{y \in \mathcal{S}} \tilde{p}(y; s, a) V^\pi(y) \\ &= (1 - \varepsilon) \left(r(s, a) + \gamma \sum_{y \in \mathcal{S}} p(y; s, a) V^\pi(y) \right) \\ &\quad + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} \left(r(s, b) + \gamma \sum_{y \in \mathcal{S}} p(y; s, b) V^\pi(y) \right) \\ &= (1 - \varepsilon) Q^\pi(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} \left(r(s, b) + \gamma \sum_{y \in \mathcal{S}} p(y; s, b) V^\pi(y) \right),\end{aligned}$$

which implies that

$$\arg \max_{a \in \mathcal{A}_s} \tilde{Q}^{\tilde{\pi}}(s, a) = \arg \max_{a \in \mathcal{A}_s} Q^\pi(s, a).$$

Therefore greedy with respect to Q^π and $\tilde{Q}^{\tilde{\pi}}$ is the same. Let π_n be an ε -soft policy, and let π_{n+1} be ε -greedy with regard to Q^{π_n} . Then $\tilde{\pi}_{n+1} := f(\pi_{n+1})$ is greedy with respect to $\tilde{Q}^{\tilde{\pi}_n}$:

$$\tilde{\pi}_{n+1}(a; s) = \frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = \begin{cases} \frac{\left((1 - \varepsilon) + \frac{\varepsilon}{|\mathcal{A}_s|} \right) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = 1 & : a = a^*(s) \\ \frac{\frac{\varepsilon}{|\mathcal{A}_s|} - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = 0 & : a \text{ otherwise} \end{cases}.$$

Since we proved in Theorem 3.3.8 convergence of exact policy iteration with greedy policy updates the proof can be finished:

$$\sup_{\pi \in \Pi_S} V^\pi(s) \stackrel{(3.14)}{=} \sup_{\tilde{\pi} \in \tilde{\Pi}_S} \tilde{V}^{\tilde{\pi}}(s) = \tilde{V}^{\tilde{\pi}^*}(s) = \lim_{n \rightarrow \infty} \tilde{V}^{\tilde{\pi}_n}(s) = \lim_{n \rightarrow \infty} V^{\pi_n}(s)$$

□

On first view it is completely unclear why there might be any interest in ε -greedy policy iteration if we already have greedy policy iteration that converges to an optimal policy. The reason, as we will discuss in the next chapter, comes from the policy evaluation step. Greedy policies can be very unfavorable to estimate Q -values or the value function if those cannot be computed explicitly. In contrast, ε -greedy policies have pleasant exploration properties, they force the algorithm to look at all actions and not only the ones that are already known to be good. The reader might compare with the exploration-exploitation trade-off for stochastic bandits in Chapter 1.

3.4 Stochastic control in finite time

⁶ So far we discussed the setting of infinite Markov decision problems with discounted total rewards. There are certainly many problems for which the time-horizon is finite. Such as stochastic bandits, where the time-horizon is 1. In this section we quickly go through the theory of non-discounted finite-time Markov decision problems.

⁶discuss stationary policies and examples (that will help for policy gradient theorems)

3.4.1 Setting and dynamic programming

Suppose T is a fixed finite time-horizon and $D = \{0, \dots, T\}$. As before \mathcal{S} denotes the state-space and \mathcal{A} the action space, both will be assumed finite again. Actions may depend on the state, we write \mathcal{A}_s for the allowed actions in state s , and p governs the transitions to a new state and the distribution of the reward. For a given policy $\pi = (\pi_t : t \in D)$ we consider the finite-time MDPs $(S_t, A_t, R_t)_{t \in D}$ and call them T -step MDPs. For simplicity we again assume that rewards R_t only depend on S_{t-1} and A_{t-1} but not on S_t . The aim for finite-time MDPs is different, non-discounted rewards $\mathbb{E}_s^\pi[\sum_{t=1}^T R_t]$, i.e. $\gamma = 1$, are typically maximised over all policies because finite time-horizon does not require discounting to have a finite total reward.

Example 3.4.1. An example to keep in mind is the ice vendor who has only three month summer season (Germans also love ice cream during winter, most others don't). Thus, the production strategy of the vendor will depend upon the distance to the terminal day, there is no need in having ice cream left after the last day of the season. An optimal policy of time-dependent problems will henceforth never be stationary!

Compared to infinite time-horizon MDPs crucial differences appear. Most importantly, the idea of dynamic programming (reduce the problem to simpler sub-problems) becomes much clearer. The general idea of dynamic programming is to reduce a problem to a smaller problem and then build a solution to a larger problem from solutions of smaller problems. For infinite horizon control the idea did not become very explicit as the reduced problem (starting one time-step later) is identical to the original problem (multiplied by γ). Thus, the reduction attempt only yields a set of equations. Here, the finite time horizon forces the reduced problem (starting one time-step later) to be simpler. The resulting set of equations will turn out to be a backwards recursion instead of a closed system of equations.

To set up the

Let us write $\pi \in \Pi_t^T$ to denote that $\pi = (\pi_i)_{i=t}^{T-1}$ is a policy that runs from time t to T , i.e. π consists of $T - t$ kernels. The definition of the state value function for T -step MDPs is as follows.



Definition 3.4.2. For any policy $\pi \in \Pi_t^T$ the functions (vectors) defined by $V_{t,T}^\pi \equiv 0$ and

$$V_{t,T}^\pi : \mathcal{S} \rightarrow \mathbb{R}, \quad s \mapsto \mathbb{E}_s^{\tilde{\pi}} \left[\sum_{t'=1}^{T-t} R_{t'} \right] =: \mathbb{E}_{S_t=s}^\pi \left[\sum_{t'=t+1}^T R_{t'} \right],$$

for $t < T$ are called time-state value functions, where $\tilde{\pi}$ is the policy π shifted by t , i.e. $\tilde{\pi}_i = \pi_{t+i}$ for $i = 0, \dots, T - t - 1$. The function $V_{0,T}^\pi$ is called state value function.

Typically the time-state value function is defined as $V_{t,T}^\pi(s) = \mathbb{E}[\sum_{i=t}^{T-1} R_{i+1} | S_t = s]$. To avoid discussions of conditioning on zero-sets we shift the starting time to 0 and force the start in s . This is rigorous (not pretty) and we keep in mind that $V_{t,T}^\pi$ is the total reward gained after time t . We also have to redefine the state-action value function



Definition 3.4.3. For any policy $\pi \in \Pi_t^T$ the functions (matrices) defined by $Q_{t,T}^\pi \equiv 0$ and

$$Q_{t,T}^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \quad (s, a) \mapsto V_{t,T}^{\pi^a}(s),$$

for $t < T$ are called time-state-action value functions. Here the policy π^a plays action a in the t th step and then continues according to π . The function $Q_{0,T}^\pi(s, a)$ is called state-action value function.

From now on we will drop the subscript T and simply write Q_t^π and V_t^π . We will just write V_t^π

and Q_t^π , except in situation in which the emphasise lies. As for discounted MDPs the difference between V and Q is that the first action is fixed to be a for Q . Similarly to Lemma 3.1.17, we also get a lemma that describes the relation between the state value function and the state-action value function for a fixed policy:



Proposition 3.4.4. Given a Markovian policy $\pi = (\pi_t)_{t \in \mathcal{D}}$ and a T -step Markov decision problem. Then the following relation between the state and state-action value function hold

$$V_t^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi_t(a; s) Q_t^\pi(s, a),$$

$$Q_t^\pi(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^\pi(s')$$

for all $t < T$. In particular, the Bellman expectation equations (backwards recursions)

$$V_t^\pi(s) = \sum_{a \in \mathcal{A}} \pi_t(a; s) \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^\pi(s') \right],$$

$$Q_t^\pi(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_s} p(s'; s, a) \pi_t(a; s') Q_{t+1}^\pi(s', a')$$

hold for $t < T$.

*Proof.*⁷ Prove the proposition by comparing with the discounted counterpart. Once the first part is proved the second follows by plugging-in. \square

It is important to note that the system of equations is different from discounted MDP problems. First, the discounting factor disappears as $\gamma = 1$ and, secondly, the linear systems are actually recursions that gradually simplify the system towards the terminal conditions $Q_T^\pi \equiv 0$, $V_T^\pi \equiv 0$. Similarly to the discounted setting the optimal value functions are defined, now depending on the remaining time-horizon:



Definition 3.4.5. For any $t \leq T$ and a given Markov decision problem

- the function $V_t^* : \mathcal{S} \rightarrow \mathbb{R}$ that takes values

$$V_t^*(s) := \sup_{\pi \in \Pi_t^T} V_t^\pi(s), \quad s \in \mathcal{S},$$

is called optimal time-state value function.

- a policy $\pi^* \in \Pi_t^T$ that satisfies

$$V_t^{\pi^*}(s) = V_t^*(s), \quad s \in \mathcal{S},$$

is called optimal policy for the time-state value function.

- the function $Q_t^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that takes values

$$Q_t^*(s, a) = \sup_{\pi \in \Pi_t^T} Q_t^\pi(s, a), \quad s \in \mathcal{S}, a \in \mathcal{A},$$

is called optimal time-state-action value function.

⁷Sara, irgendwann reinschreiben

As for discounted MDPs the optimal value functions are the best value functions that are theoretically achievable by a policy. It is far from obvious that there is an optimal policy. As in the discounted setting we get the following relations



Lemma 3.4.6. The following holds for the optimal time-state value function and the optimal time-state-action value function for any $s \in \mathcal{S}$:

- (i) $V_t^*(s) = \max_{a \in \mathcal{A}_s} Q_t^*(s, a)$ for all $t < T$
- (ii) $Q_t^*(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^*(s')$ for all $t < T$

In particular, V^* and Q^* satisfy the following Bellman optimality equations (backwards recursions):

$$V_t^*(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^*(s') \right\}, \quad s \in \mathcal{S},$$

and

$$Q_t^*(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q_{t+1}^*(s', a'), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

for all $t < T$.

*Proof.*⁸ The proof of (i) and (ii) is similar to the discounted setting and left as an exercise. The Bellman optimality equations are then a direct consequence of (i) and (ii) by plugging (i) into (ii) and vice-versa. \square

For discounted infinite time horizon problems we could now show that it is sufficient to consider stationary greedy policies. **The stationarity is in general not true for finite time horizon MDPs.** Consider for example the ice-vendor MDP and assume that we close our store in the winter. Then the amount of ice cream we want to order depends strongly on the time horizon up to the closing date. It is clear that we would like to order with a different strategy if we can sell the ice cream for 6 weeks rather than just for 1 week. Given this observation it is clear that we can no longer restrict the set of policies to stationary policies and it follows also that we have no fixed point relation in the value function. We can formulate the following theorem:



Theorem 3.4.7. Suppose $v_t : \mathcal{S} \rightarrow \mathbb{R}$, $t = 0, \dots, T$, is a sequence of vectors that fulfill the Bellman optimality equations (backwards recursions)

$$v_t(s) = \begin{cases} 0 & : t = T \\ \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) v_{t+1}(s') \right\} & : t < T \end{cases},$$

then $v_t = V_t^*$ for all $t = 0, \dots, T$. Similarly, if $q_t : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, $t = 0, \dots, T$, is a sequence of vectors that fulfill the Bellman state-action optimality equations (backwards recursions)

$$q_t(s, a) = \begin{cases} 0 & : t = T \\ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} q_{t+1}(s', a') & : t < T \end{cases},$$

then $q_t = Q_t^*$ for all $t = 0, \dots, T$. Most importantly, an optimal (non-stationary) policy is given by the greedy policy

$$\pi_t^*(a; s) = \begin{cases} 1 & : a = a_t^*(s) \\ 0 & : \text{otherwise} \end{cases}, \quad \text{where } a_t^*(s) = \arg \max_{a \in \mathcal{A}_s} q_t(s, a).$$

⁸Sara, irgendwann reinschreiben

Proof. Not much needs to be proved. In contrast to the discounted infinite-time horizon case the Bellman equations are no closed equations, they are recursions which uniquely determine the solutions through the terminal conditions. Solutions are given by V^* and Q^* due to the previous lemma (the only non-trivial point of the theory). If π is the greedy policy obtained from q (thus of Q^*), then the definition of greedy policies yields

$$V_t^*(s) = \max_{a \in \mathcal{A}_s} Q_t^*(s, a) = V_t^{\pi^*}(s)$$

because $q_t = Q_t^*$. □

The key tool to solve finite time horizon MDPs follows from Theorem 3.4.7 and is called backward induction. If the state-action space is not too large and we have access to the transition probabilities, then the optimal control problem can be solved backwards iteration. Let us recall the ice vendor example on finite time-horizon to get an idea why this is intuitive. In the last timestep, there is no opportunity to sell any more ice cream, the summer season is over. In the optimal situation we obviously have no more stock which corresponds to $V_T^* \equiv 0$. Then we go backwards in time step-by-step and consider what is optimal in every possible state. Suppose one day is left. The recursion gives

$$Q_{T-1}^*(s, a) = \underbrace{r(s, a)}_{\text{last days profit}} + \underbrace{\sum_{s' \in \mathcal{S}} p(s'; s, a) V_T^*(s')}_{=0, \text{ no future to be considered}} = r(s, a)$$

and the optimal policy becomes

$$\pi_{T-1}^*(s) = \arg \max_{a \in \mathcal{A}_s} r(s, a).$$

What does this mean? For the last step we need to chose the action that maximises the expected reward without any thought about the future. No surprise! If the expectations are know, nothing needs to be done. If not, this is nothing but the bandit problem! For the next time-step $T - 2$ the recursion gives

$$Q_{T-2}^*(s, a) = \underbrace{r(s, a)}_{\text{today's profit}} + \underbrace{\sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q_{T-1}^*(s', a')}_{\text{next days profit}}$$

and optimally the ice vendor orders/produces ice cream according to

$$\pi_{T-2}^*(s) = \arg \max_{a \in \mathcal{A}_s} Q_{T-2}^*(s, a).$$

If one could carry out explicitly the recursion till $t = 0$ the sequence $(\pi_t^*)_{t=1, \dots, T}$ is an optimal time-dependent policy.

3.4.2 Dynamic programming algorithms

In the context of fully available dynamics the dynamic programming algorithms are very simple for finite MDPs. No equations must be solved, only recursion followed backwards from the terminal conditions.

Theorem 3.4.7 immediately translates into a simple optimal control algorithm.

Algorithm 13: Policy evaluation for finite-time MDPs

Data: MDP with finite time-horizon T , policy $\pi \in \Pi_0^T$ **Result:** V_t^π and Q_t^π for all $t = 0, \dots, T-1$ Set backward induction start $V_T^\pi \equiv 0$ **for** $t = T-1, \dots, 0$ **do** **for** $s \in S$ **do** **for** $a \in \mathcal{A}_s$ **do**

$$Q_t^\pi(s, a) := r(s, a) + \sum_{s' \in S} p(s'; s, a) V_{t+1}^\pi(s')$$

end

$$V_t^\pi(s) := \sum_{a \in \mathcal{A}_s} \pi_t(a; s) Q_t^\pi(s, a)$$

end**end**

Algorithm 14: Optimal control for finite-time MDPs

Data: MDP with finite time-horizon T **Result:** V_t^* and Q_t^* for all $t = 0, \dots, T$ and optimal policy π^* Set induction beginning $V_T^* \equiv 0$ **for** $t = T-1, \dots, 0$ **do** **for** $s \in S$ **do** **for** $a \in \mathcal{A}_s$ *afely* **do**

$$Q_t^*(s, a) := r(s, a) + \sum_{s' \in S} p(s'; s, a) V_{t+1}^*(s')$$

end

$$V_t^*(s) := \sum_{a \in \mathcal{A}} \pi_t^*(a; s) Q_t^*(s, a)$$

$$\pi_t^* := \text{greedy}(Q_t^*)$$

end**end**

Chapter 4

Approximate dynamic programming methods



In this section we assume the rewards R_{t+1} only depend on (S_t, A_t) and write $R(s, a)$ for the reward distribution given $(S_t, A_t) = (s, a)$ and $r(s, a)$ for its expectation.

The aim of this chapter is to turn the theoretical solution methods from the previous chapter into more useful algorithms that suit better to real-world situations. To understand what this means let us discuss major drawbacks of using Bellman equations that we try to overcome in this chapter:

- Since the dynamic programming algorithms are based on iterating Bellman operators the operators must be known and accessible. Most importantly, the transition function p (the environment of the problem) must be known explicitly. In many situations this is not the case. This sounds weird, on first glance one might think that there is no way to formulate Markov decision problems without knowing the decision problem. Indeed, this is not the case. There might be a physical model underlying a decision problem but the learning agent has no access to the physical model. We can play a computer game without knowing the transitions, we can drive a car without knowing the outcome of a steering decision. What can be done in both examples (and this is what we do in real life!) is to learn the transition probabilities by sampling (playing).
- There is theoretical access to all ingredients of the decision problem but state (and/or action) space might be too big to be repeatedly usable, or even worse, too big to be stored at all. The state space might also be too big to learn about all possible decisions, but perhaps most states and actions are irrelevant.

In both cases there is no way to use the standard dynamic programming algorithms as they need explicit knowledge of the transitions and treat all states/actions simultaneously. In this chapter we discuss approximation ideas that mostly deal with the first problem, the second problem is attacked later in non-tabular RL by approximating the decision problem by smaller decision problems.



Definition 4.0.1. A solution algorithm is called **model-based** if the algorithm requires explicit knowledge on the Markov decision model. A solution algorithm is called **model-free** if the algorithm only requires the ability to simulate all appearing random variables.

Comparing the situation to stochastic bandits all algorithms were model-free. In fact, knowing all probabilities explicitly allows to compute the action values Q_a which then allows to use the trivial algorithm that compares the action values and only plays the optimal arm. The

situation might be different in practice if all probabilities are known but the expectations cannot be computed. Then one would also use a model-free algorithm even if the model is known.

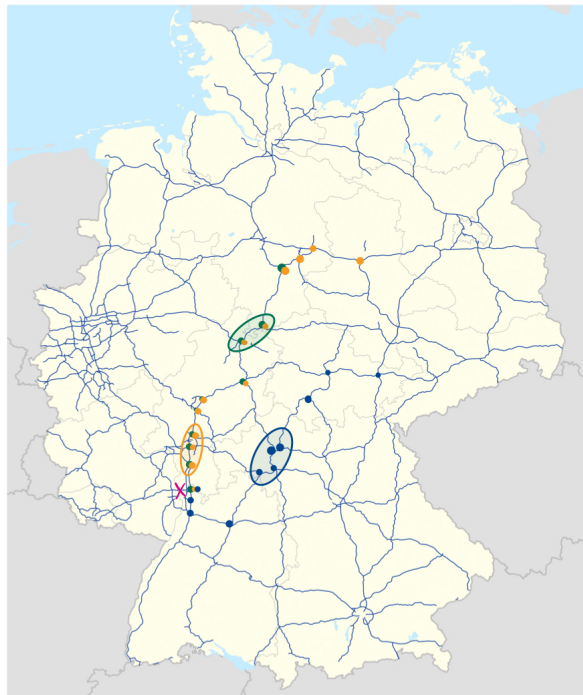
In what follows we develop approximate dynamic programming algorithms for the tasks we addressed with the dynamic programming algorithms based on Bellman's equations:

- policy evaluation, i.e. estimate $V^\pi(s)$ for a given policy π ,
- optimal control, solve the stochastic control problem (i.e. find the optimal V^* , Q^* , π^*).

The chapter is organised as follows. We start with a quick naive discussion on how to evaluate policies using Monte Carlo. The procedure leads to an obvious disadvantage (roll-outs of the MDP need to be sampled till the end) which is tackled in the rest of the chapter. For this sake a stochastic version of Banach's fixed point theorem is proved from which a couple of different approximation algorithms is derived.

4.1 A guiding example

We start the discussion with a guiding example that should provide some intuition why the mathematical concepts introduced below are completely intuitive¹. Suppose we have two high-way navigation systems that are supposed to bring us back to Mannheim. Which one is better?



Temporal differences of 1, 2, and 3 steps

To formulate the problems the state space consists for instance of autobahn junctions or drive-ups or cities. The actions are the possible decisions of the navigation systems. The MDP has a terminating state, Mannheim. The rewards are the times it takes to reach the next state (city, junction, etc.). Since the terminal state is reached sufficiently fast, a discounting factor 0.99 has

¹For a real-world example with Taxis on the streets of Pittsburgh (in a different context) see Ziebart, Maas, Bagnell, Dey: "Maximum Entropy Inverse Reinforcement Learning", AAAI Conference on Artificial Intelligence, 2008

not much effect ($.99^{20} \approx .82$) so the total discounted reward is a good approximation of the time it takes from city s (e.g. Berlin) to Mannheim. To compare two navigation systems π and π' we should compute $V^\pi(\text{Berlin})$ and $V^{\pi'}(\text{Berlin})$ for all cities. There are two approaches that are natural.

- Drive the car repeatedly from every city to Mannheim and take the average time of arrival as estimator for $V(s)$. Well, only a Mathematician would suggest such a stupid approach. The simple Monte Carlo approach does not reuse estimates (this is called bootstrapping). Looking at the visualisation this is clearly not clever, many trajectories have common sections. The simplest improvement is to use the strong Markov property of an MDP and also reuse the remaining time from all cities s' on the way to estimate $V(s')$.
- Intuitively, as a human being we would proceed much smarter. Typically we have a good feeling of the time it takes between two cities because we bootstrap a lot more information. Whenever we drive any segment we remember times and intuitively use new information to update all other information. Imagine we take the autobahn from Wolfsburg to Frankfurt and observe a new construction site then we will automatically use the estimate for the time from Wolfsburg to Frankfurt (this is called a temporal difference) to update the current estimate of the time it takes from Berlin to Mannheim. In the Mathematics below we will use temporal differences of different length, 1, n , but also infinitely many. Now think about the idea, that's what we do! Thinking practical, it's easy to imagine how much information can be extract from all trucks that constantly move around on the autobahn and yield estimates for all kind of temporal differences. Using

This chapter consists of different approaches to turn such ideas into algorithms. Monte Carlo is rather obvious, while temporal difference methods require a bit of Mathematics. In fact, it turns out that temporal difference methods are random versions of the Bellman equation, justifying their name approximate dynamic programming methods. There are always two questions that are optimally solved. Policy evaluation and optimal control.



With the bootstrapping ideas sketched above try to improve decision making (i.e. improve the navigation systems) of the current system using only temporal difference observations. It's not completely unlikely that you will develop an idea of what later will be called Q -learning.

To be honest, this navigation system example is not very realistic. A navigation system developed this way won't be competitive as it does not incorporate live-data which we all know is crucial and why google maps is as strong as it is. Also to be honest, the Mathematics will be more restricted than the example above. The use of new estimates for temporal differences (driving times between cities) will not be as generic as some truck drivers performing their duty. Still, the example captures the main ideas of this chapter.

4.2 Monte Carlo policy evaluation and control

The aim of this short section is to get a glimpse on what it means to use Monte Carlo for policy evaluation without any further thoughts. Recall that policy evaluation is important to know the value of a policy but also to perform generalised policy iteration (alternate between policy evaluation and policy improvement). In the chapter on dynamic programming we did not care much about feasibility of computational issues, we pretended that everything is available and can be computed. As discussed above this is not always possible. Since the value function by definition is an expectation the most obvious model-free variant for policy evaluation is Monte Carlo estimate with independent roll-outs of the MDP:

$$\hat{V}^\pi(s) = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \gamma^t R(S_t^i, A_t^i) \quad (4.1)$$

for some large T and N . This naive approach is extremely inefficient as roll-outs (S^i, A^i) of the MDP are needed for all starting point s and initial action a . Nonetheless, the usual advantage of Monte Carlo methods still holds, Monte Carlo methods are very robust. In the following we discuss more clever ways to improve the sample efficiency of the Monte Carlo method.



Sample efficiency means that algorithms are supposed to use as few random samples as possible. In the context of Monte Carlo this means to extract as much information as possible from any roll-out (S, A) of the Markov decision process.

4.2.1 First visit Monte Carlo policy evaluation

Before we proceed note that the truncation in the estimate of V^π by (4.1) automatically induces a bias (i.e. the expectation of the estimator $\hat{V}^\pi(s)$ is different from $V^\pi(s)$). While there might be no way around such a truncation in practice (we cannot simulate an infinite length MDP) the problem does not occur in many situations. For examples most games do not run forever but stop in a state in which the game is over.



Definition 4.2.1. A state $s' \in \mathcal{S}$ is called terminating if

$$p(s'; s', a) = 1 \quad \text{and} \quad R(s', a) = 0, \quad \forall a \in A_{s'}.$$

An MDP is called terminating if there is a terminating state (possibly several) which is hit in finite time almost surely. The (random) time of termination is typically denoted by T .

A terminating MDP hits a terminating state almost surely in finite time and then stays in the state without generating any further reward. In case of a game that might be any state in which the play won/lost. In contrast to general MDPs the discounted total reward of a terminating MDP can be simulated (in finite time). For the theory that follows we will either assume we could simulate an MDP of infinite length or the MDP is terminating even though in practice it might be necessary to stop the roll-outs after a finite number of steps.

Here is the clever idea, so-called first visit Monte Carlo policy evaluation:



The strong Markov property of an MDP (every Markov chain is strong Markov!) implies that restarted when first hitting a state s' the process $(S'_t, A'_t) := (S_{T_{s'}+t}, A_{T_{s'}+t})$ is an MDP with the same transitions but initial condition $S'_0 = s'$. Hence, we can extract from one roll-out (correlated) estimates of V^π for many different starting points. Similarly, restarting first visit of a given state-action pair it is possible to extract from one roll-out (correlated) estimates of Q^π for several different state-action pairs.

The idea is easy to implement. Run a number of roll-out and for every roll-out store the future discounted reward if a state is visited for the first time. Next, for every state take the average future discounted reward. For the latter recall from (1.10) a memory efficient way to avoid storing all past rewards.

Thinking for a moment it becomes clear how to modify the algorithm to obtain a first visit algorithm that estimates Q^π . Since (S, A) is Markov for stationary policies the algorithm uses all future discounted rewards when first hitting a state-action pair (s, a) . First visit Monte Carlo converges to the true value function (state-action value function) if infinitely many updates can be guaranteed. For every visit of a state s (resp. state-action pair (s, a)) the algorithm produces a sample of the discounted total reward, thus, the law of large number implies convergence.



Theorem 4.2.2. The first visit Monte Carlo algorithms satisfy the following convergence properties for $s \in \mathcal{S}$ and $a \in \mathcal{A}$:

Algorithm 15: First visit Monte Carlo policy evaluation of V^π for terminating MDPs

Data: Policy $\pi \in \Pi_S$, initial condition μ
Result: Approximation $V \approx V^\pi$
 Initialize $V_0 \equiv 0$ and $N \equiv 0$
 $n = 0$
while *not converged* **do**
 Sample s_0 from μ .
 Generate trajectory (s_0, a_0, s_1, \dots) until termination point T using policy π .
 for $t = 0, 1, 2, \dots, T$ **do**
 if $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$ **then**
 $v = R(s_t, a_t) + \sum_{k=t+1}^{\infty} \gamma^{k-t} R(s_k, a_k)$
 $V_{n+1}(s_t) = \frac{1}{N(s_t)+1} v + \frac{N(s_t)}{N(s_t)+1} V_n(s_t)$
 $N(s_t) = N(s_t) + 1$
 end
 end
 $n = n + 1$
end

Algorithm 16: First visit Monte Carlo policy evaluation of Q^π

Data: Policy $\pi \in \Pi_S$, initial condition μ
Result: Approximation $Q \approx Q^\pi$
 Initialize $Q_0 \equiv 0$ and $M \equiv 0$
 $n = 0$
while *not converged* **do**
 Sample s_0 from μ
 Generate trajectory (s_0, a_0, s_1, \dots) until termination point T using policy π .
 for $t = 0, 1, 2, \dots, T$ **do**
 if $(s_t, a_t) \notin \{(s_0, a_0), (s_1, a_1), \dots, (s_{t-1}, a_{t-1})\}$ **then**
 $q = R(s_t, a_t) + \sum_{k=t+1}^{\infty} \gamma^{k-t} R(s_k, a_k)$
 $Q_{n+1}(s_t, a_t) = \frac{1}{M(s_t, a_t)+1} q + \frac{M(s_t, a_t)}{M(s_t, a_t)+1} Q_n(s_t, a_t)$
 $M(s_t, a_t) = M(s_t, a_t) + 1$
 end
 end
 $n = n + 1$
end



- If $N(s) \rightarrow \infty$ almost surely, then $V_n(s) \rightarrow V^\pi(s)$ almost surely.
- If $M(s, a) \rightarrow \infty$ almost surely, then $Q_n(s, a) \rightarrow Q^\pi(s, a)$ almost surely.

Proof. Strong Markov property, law of large numbers □

How can we achieve the condition of the theorem? There is not much we can manipulate, only the initial condition μ and the policy π :

- Chose an initial distribution μ (such as uniform) that puts mass on all states, this is called exploring start.
- Chose a policy π' that distributes mass more evenly on all possible actions than the target policy π , this is called policy exploration.

Comparing with stochastic bandits there is a similar exploration vs. exploitation trade-off for first visit Monte Carlo, playing with μ and π has different advantages and disadvantages: Forcing

more exploration through μ and/or π improves the Monte Carlo convergence for states that are less favorable under μ and/or π but downgrades convergence for states that are more favorable under μ and π . Additionally, an error occurs as the Monte Carlo procedure estimates $V^{\pi'}$ (resp. $Q^{\pi'}$) instead of V^π (resp. Q^π). Nonetheless, if the evaluation is used for a generalised policy iteration scheme it might be favorable to estimate (explore) more carefully actions that are less likely for the policy π that is currently believed to be best. As for bandits, in practice one will first force more exploration by π and during the learning process decrease the exploration bonus.



There is another version of Monte Carlo that is used in practice but theoretically much more problematic. Instead of updating at first visits of states, the discounted future reward is taken as a sample of the discounted total reward for every visit of every state. For samples obtained from the same roll-out the sampled discounted rewards are clearly strongly dependent, thus, not allowing the use of the law of large numbers. The corresponding (biased) algorithm is called **every visit Monte Carlo**.

The Monte Carlo policy evaluation algorithm is only written for episodic (terminating) MDPs. For infinite horizon one would have to truncate the update at some finite time. An alternative using stopping at independent geometric times is discussed below in Section 4.5.2.

4.2.2 Generalised policy iteration with first visit Monte Carlo estimation

We have introduced an algorithm that can estimate the Q -values of a given policy only by simulation. Thus, there is a natural model-free procedure to approximate an optimal policy π^* by interacting with the environment (i.e. running the MDP for a given policy). The algorithm would start with a policy $\pi \in \Pi_S$, estimate Q -values \hat{Q}^π by performing the first visit Monte Carlo procedure up to some termination condition (or till the end of the MDP terminates) and then improve the policy. According to the discussion above the procedure has an exploitation-exploration trade-off. Unfortunately, there is an exploration problem. Suppose we start with a greedy policy with weights on suboptimal actions only. Then the first visit algorithm will continue to update the suboptimal Q -values and the greedy policy update will continue playing those actions. In order to force more exploration we will exchange the greedy policy update from policy iteration by ε -greedy policy update. Similarly to stochastic bandits, ε -greedy puts mass ε to randomly chosen actions which defavours the policy. Still, comparing with the decreased cost of Monte Carlo policy evaluation it can still be suitable to force exploration using soft-policies. Since

Algorithm 17: Monte Carlo generalised ε -greedy policy iteration

Data: initial policy π

Result: approximation of π^*

while *not converged* **do**

 Policy evaluation: Obtain estimates \hat{Q}^π from first visit Monte Carlo.

 Policy improvement: Obtain the ε -greedy policy π_{new} from \hat{Q}^π

 Set $\pi = \pi_{new}$.

end

return π

the algorithms is not very practical we leave open the question of convergence (in some sense) and cost analysis (in some sense) for generalised policy iteration with ε -greedy policy update and first visit Monte Carlos policy evaluation. Such a discussion should combine the advantage in effort/precision of first visit Monte Carlo policy evaluation with exploitation with the error committed by choosing ε -greedy policy update. As for bandits one should expect decreasing ε to be advantageous.



Simulate Monte Carlo generalised ε -greedy policy iteration for some MDP example with different choices of ε fixed or $\varepsilon_n \downarrow 0$.

No doubt, other exploration strategies such as Boltzman exploration can be equally reasonable. But to the best of our knowledge not much is known for theoretical results in generalised policy iteration with with different exploration schemes².

4.3 Asynchronous stochastic fixed point iterations

In order to justify the convergence of approximate dynamic programming algorithms we will use arguments that are mostly due to John Tsitsiklis and coauthors. The idea of the following is simple. Suppose we aim at solving the fixed point equation $F(x) = x$ using Banachs fixed point iteration but cannot compute $F(x)$ exactly but only have stochastic approximations $\widehat{F}(x)$ (such as Monte Carlo estimates of an expectation). Can we still perform Banachs fixed point iteration as $x(n+1) = \widehat{F}(x(n))$ and converge (almost surely) to the unique fixed point of F ? The answer is no but the update scheme $x(n+1) = (1 - \alpha(n))x(n) + \alpha(n)\widehat{F}(s(n))$ converges to x^* if the estimats are unbiased with well-behaved second moments and the so-called step-sizes α decay with a suitable rate. Writing Bellman operators with expectations this will readily lead us to different numerical schemes for policy evaluation and optimal control.

We will first discuss in the most simple settings the ideas of Robbins and Monro³ to find roots of functions that can only be approximated and then turn towards Tsitsiklis' stochastic approximation theorem that serves as the main tool to prove convergence for all approximate dynamic programming algorithms.

4.3.1 Basic stochastic approximation theorem

The simple Robbins Monro algorithm addresses the task to find zeros $G(x) = 0$ for very particular functions G . From lectures on numerical analysis different algorithms (such as Newton's method $x_{n+1} = x_n + \frac{G(x_n)}{G'(x_n)}$) might be known to the reader. The situation that Robbins and Monro addressed is different in the sense that they considered functions G for which there is no access to the true values $G(x)$ but only to unbiased approximations $\tilde{G}(x) = G(x) + \varepsilon$, where ε is a mean-zero error. The situation to keep in mind is that of a function G that is defined as an expectation (such as $V^\pi(s)$ or $Q^\pi(s, a)$) that can be approximated for instance by Monte Carlo. The most generic Robbins/Monro approximation scheme is

$$x_{n+1} = x_n - \alpha_n y_n, \quad (4.2)$$

where α_n are step-sizes specified below and y_n are stochastic approximations of $G(x_n)$. Here is a first simple convergence theorem, introducing the first stochastic approximation (SA) algorithm to approximate roots of nice enough functions:



Theorem 4.3.1. (Simplest Robbins-Monro algorithm)

Suppose there is some $\kappa > 0$ such that $G(x) - G(y) \geq \kappa(x - y)$ for all $x, y \in \mathbb{R}$ and the stochastic process (x_n) is defined by (4.2), where y_n are stochastic approximations of $G(x_n)$:

$$y_n = G(x_n) + \varepsilon_n$$

and the processes are defined on a filtered probability space $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$ carrying all appearing random variables. Assume that there are deterministic step-sizes (α_n)

²Check here for some results: <https://arxiv.org/pdf/1903.05926.pdf>

³H. Robbins and S. Monro: A stochastic approximation method. The Annals of Mathematical Statistics, 22:400–407, 1951.



satisfying the so-called Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty$$

and \mathcal{F}_{n+1} -measurable errors satisfying $\mathbb{E}[\varepsilon_n | \mathcal{F}_n] = 0$. Suppose furthermore that $\sum_{n=0}^{\infty} \mathbb{E}[y_n^2] \alpha_n^2 < \infty$, then $x_n \xrightarrow{L^2} x_*$ for $n \rightarrow \infty$, where $G(x_*) = 0$.

The condition $\sum_{n=0}^{\infty} \mathbb{E}[y_n^2] \alpha_n^2 < \infty$ is satisfied for instance if G is bounded and the errors have bounded conditional second moments, a condition that will occur in all theorems below.



The most important sequence satisfying the Robbins-Monro conditions is $\alpha_n = \frac{1}{n}$. Other obvious choices are $\alpha_n = \frac{1}{n^p}$ for all $\frac{1}{2} < p \leq 1$.

The assumption on G is way too strong and the convergence mode weak (only L^2 , thus, in probability), we discuss this simple version as it allows us to give a simple proof. We will later give a much more general version of the Robbins-Monro theorem but need to involve stability theory of ordinary differential equations. Here are two (very similar) classical situations:

- If $G = F'$, then the assumption on G is what is typically called strongly convex of F and the algorithm converges (in L^2) to a critical point of F . If G is differentiable then the condition implies $F'' \geq \kappa$, a condition that is stronger than convex.
- If G is defined on $[a, b]$ then the condition is for instance satisfied if $G(a) < 0 < G(b)$ and has a unique root x_* where $G' \geq \kappa$.

In fact, the second example shows that Robbins-Monro is not really made for strongly convex functions. The most important situation to keep in mind is the one of functions defined through (unknown) expectations of random variables that can be sampled:

Example 4.3.2. Suppose $G(x) = \mathbb{E}[f(x, Y)]$ for some random variable Y (and all expectations are finite). If we have access to f and samples \tilde{Y} of Y then we have access to stochastic approximations $f(x, \tilde{Y}) = G(x) + (f(x, \tilde{Y}) - G(x)) =: G(x) + \varepsilon$ of $f(x)$. Keeping in mind that reinforcement learning is about reward functions (expectations) it might be little surprising that stochastic approximation is related to reinforcement learning.

Proof of simple Robbins Monro. The proof mostly relies on a simple recursion result on positive sequences:



Suppose that z_n is a positive sequence such that $z_{n+1} \leq (1 - a_n)z_n + c_n$, where a_n, c_n are positive sequences satisfying

$$\sum_{n=1}^{\infty} a_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} c_n < \infty.$$

Then $\lim_{n \rightarrow \infty} z_n = 0$.

Without loss of generality we can assume equality in the assumption as otherwise we could decrease c_n or increase a_n which does not change the summation tests. The positivity assumptions lead to the lower bound

$$-z_0 \leq z_n - z_0 = \sum_{k=0}^{n-1} (z_{k+1} - z_k) = \sum_{k=0}^{n-1} c_k - \sum_{k=0}^{n-1} a_k z_k.$$

Since the left hand side is finite and $\sum_{k=0}^{\infty} c_k < \infty$ by assumption it follows that $\sum_{k=0}^{\infty} a_k z_k < \infty$. Finally, since $\sum_{k=0}^{\infty} a_k = \infty$ it follows that $\lim_{n \rightarrow \infty} z_n = 0$.



A positive recursion for the L^2 -error.

Define $z_n = \mathbb{E}[(x_n - x_*)^2]$, $e_n = \mathbb{E}[y_n^2]$, and $d_n = \mathbb{E}[(x_n - x_*)(G(x_n) - G(x_*))]$. Then simple algebra leads to

$$\begin{aligned}
 z_{n+1} &= \mathbb{E}[(x_{n+1} - x_n + x_n - x_*)^2] \\
 &= \mathbb{E}[(x_{n+1} - x_n)^2] + 2\mathbb{E}[(x_{n+1} - x_n)(x_n - x_*)] + \mathbb{E}[(x_n - x_*)^2] \\
 &= \mathbb{E}[(\alpha_n y_n)^2] + 2\mathbb{E}[(-\alpha_n y_n)(x_n - x_*)] + \mathbb{E}[(x_n - x_*)^2] \\
 &= \alpha_n^2 \mathbb{E}[y_n^2] - 2\alpha_n (\mathbb{E}[(G(x_n))(x_n - x_*)] + \mathbb{E}[\varepsilon_n(x_n - x_*)]) + \mathbb{E}[(x_n - x_*)^2] \\
 &= \alpha_n^2 \mathbb{E}[y_n^2] - 2\alpha_n (\mathbb{E}[(G(x_n) - G(x_*))(x_n - x_*)] + \mathbb{E}[\mathbb{E}[\varepsilon_n | \mathcal{F}_n](x_n - x_*)]) + \mathbb{E}[(x_n - x_*)^2] \\
 &= \alpha_n^2 e_n - 2\alpha_n d_n + z_n.
 \end{aligned}$$

Now the assumption on G implies that

$$d_n \geq \kappa \mathbb{E}[(x_n - x_*)(x_n - x_*)] = \kappa z_n,$$

so that in total we derived the positive recursion

$$z_{n+1} \leq z_n(1 - 2\alpha_n \kappa) + \alpha_n^2 e_n.$$

Hence, the claim follows from the first step and shows very clearly the origin of the summability condition on the step sizes. \square

From the probabilistic point of view it is important to note that almost nothing (like independence, adaptivity, etc.) was assumed on the stochastic approximations y_n , the catch is the weak L^2 -convergence mode. We will later see a version with almost sure convergence that rely on the almost sure (super)martingale convergence theorem. In that case much more needs to be assumed.



Robbins-Monro type algorithms can equally be used to find roots of $G(x) = a$, the corresponding algorithm becomes

$$x_{n+1} = x_n - \alpha_n(y_n - a), \quad (4.3)$$

where y are again approximations of $G(x_n)$.

We finish the first discussion of the stochastic approximation algorithms with an important example that shows that stochastic approximation can not be expected to have good convergence rate:

Example 4.3.3. Suppose Z_1, Z_2, \dots is an iid sequence with finite mean μ . Then

$$\theta_{n+1} = \theta_n + \frac{1}{n+1}(Z_n - \theta_n)$$

can be solved explicitly as

$$\theta_n = \frac{1}{n}\theta_0 + \frac{1}{n} \sum_{k=0}^{n-1} Z_k$$

and the righthand side converges to μ by the law of large numbers. In fact, the convergence is even almost surely and in L^2 if Z_i have finite variance. This way of writing the law of large numbers can be considered a stochastic approximation iteration to find a zero for $G(\theta) = \mu - \theta$ with the Robbins-Monro iteration with step-sizes $\alpha_n = \frac{1}{n+1}$ and unbiased approximations $Z_n - \theta_n$ of $G(\theta_n)$. Keeping in mind that the convergence in the law of large numbers is very slow (order \sqrt{n}) the example gives a first hint that stochastic approximation is a robust but slow algorithm.

4.3.2 Asynchronous stochastic approximation

We will now develop the main tools to prove convergence of Q -learning by proving a much more interesting version of the Robbins-Monro algorithm that results in almost sure convergence to fixedpoints. The proof is very interesting as it makes very transparent how the learning algorithms can be modified and still converge to the optimal state-action function.

The asynchronous stochastic approximation theorem that is needed to prove convergence of Q -learning must be an almost sure convergence theorem to give a useful convergence statement of Q -learning. Comparing with the proof of the simple Robbins-Monro theorem a stochastic version (with almost sure convergence) of the first step of the proof is needed. Formulated in the right way this is a consequence of the almost sure (super)martingale convergence theorem: ⁴



Theorem 4.3.4. (Robbins-Siegmund Theorem)

Suppose $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space carrying a non-negative adapted stochastic process (Z_n) . If

$$\mathbb{E}[Z_{n+1} | \mathcal{F}_n] \leq (1 - a_n + b_n)Z_n + c_n \quad (4.4)$$

for some non-negative (\mathcal{F}_n) -adaptive random variables a_n, b_n, c_n such that almost surely

$$\sum_{n=1}^{\infty} a_n = \infty, \quad \sum_{n=1}^{\infty} b_n < \infty, \quad \text{and} \quad \sum_{n=1}^{\infty} c_n < \infty,$$

then $\lim_{n \rightarrow \infty} Z_n = 0$ almost surely.

Proof. The proof⁵ is a nice application of Doob's almost sure martingale convergence theorem.



Without loss of generality b can be assumed to be equal to 0.

If we divide both sides of the assumption by $\prod_{m=0}^n (1 + b_m)$, then we get

$$\mathbb{E}[Z'_{n+1} | \mathcal{F}_n] \leq (1 - a'_n)Z'_n + c'_n,$$

where $a'_n = \frac{a_n}{1+b_n}$, $c'_n = \frac{c_n}{\prod_{m=0}^n (1+b_m)}$ and $Z'_{n+1} = \frac{Z_n}{\prod_{m=0}^{n-1} (1+b_m)}$. Note that since $\sum b_n$ converges then so does $\prod (1 + b_m)$:

$$\log \left(\prod_{m=0}^{n-1} (1 + b_m) \right) = \sum_{m=0}^{n-1} (\log(1 + b_m)) \leq \sum_{m=0}^{n-1} b_m \leq \sum_{m=0}^{\infty} b_m < \infty.$$

Thus, a'_n , c'_n and Z'_n have the same convergence properties as those required for a_n , c_n , and Z_n .



Auxiliary limit through (super)martingale convergence theorem.

From now on we assume $b \equiv 0$. The crucial observation of the proof is that

$$Y_n := Z_n + \sum_{k=0}^{n-1} a_k Z_k - \sum_{k=0}^{n-1} c_k$$

⁴put Simons version, diese Version aus Korollar

⁵H. Robbins and D. Siegmund. „A convergence theorem for non-negative almost supermartingales and some applications“, Optimizing methods in statistics, pages 233-257, Elsevier, 1971.

is a supermartingale because

$$\begin{aligned}
\mathbb{E}[Y_{n+1} | \mathcal{F}_n] &= \mathbb{E}[Z_{n+1} | \mathcal{F}_n] + \mathbb{E}\left[\sum_{k=0}^n a_k Z_k \middle| \mathcal{F}_n\right] - \sum_{k=0}^n c_k \\
&= \mathbb{E}[Z_{n+1} | \mathcal{F}_n] + \sum_{k=0}^n a_k Z_k - \sum_{k=0}^n c_k \\
&\leq (1 - a_n)Z_n + c_n + \sum_{k=0}^n a_k Z_k - \sum_{k=0}^n c_k \\
&= Z_n + \sum_{k=0}^{n-1} a_k Z_k - \sum_{k=0}^{n-1} c_k = Y_n.
\end{aligned}$$

In order to apply the supermartingale convergence theorem little thought is needed. The supermartingale convergence theorem applies if $\sup_n \mathbb{E}[Y_n^-] < \infty$, in particular, if Y would be bounded below. Since both assumptions are not necessarily satisfied we proceed by localising. The assumed adaptivity implies that $\tau_m := \inf\{n : \sum_{k=0}^n c_k > m\}$ is a stopping-time for every $m \in \mathbb{N}$. Hence, the stopped process $Y_n^{\tau_m} := Y_{\tau_m \wedge n}$ is a supermartingale which is bounded below by $-m$ because

$$Y_{n \wedge \tau_m} \geq - \sum_{k=0}^{n \wedge \tau_m - 1} c_k \geq -m.$$

Thus, for all $m \in \mathbb{N}$ the limits $\lim_{n \rightarrow \infty} Y_n^{\tau_m}$ exist. Define $\Omega_m := \{\omega \in \Omega : \lim_{n \rightarrow \infty} Y_n^{\tau_m} \text{ exists}\}$ with $\mathbb{P}(\Omega_m) = 1$. Let

$$\tilde{\Omega} := \bigcap_{m \in \mathbb{N}} \Omega_m$$

and $\Omega_c := \{\omega \in \Omega : \sum_{k=0}^{\infty} c_k(\omega) < \infty\} \cap \tilde{\Omega}$. It holds $\mathbb{P}(\Omega_c) = 1$ by assumption. Let $\omega \in \Omega_c$, then there exists $m \in \mathbb{N}$ with $\sum_{k=0}^{\infty} c_k(\omega) \leq m$ and therefore, $Y_n(\omega) = Y_n^{\tau_m}(\omega)$ for every $n \in \mathbb{N}$. Hence,

$$\lim_{n \rightarrow \infty} Y_n(\omega) = \lim_{n \rightarrow \infty} Y_n^{\tau_m}(\omega) < \infty$$

because $\omega \in \tilde{\Omega} \subseteq \Omega_m$. Therefore, $\lim_{n \rightarrow \infty} Y_n$ exists for every $\omega \in \Omega_c$ and hence, almost surely.



Limit of Z equals 0 almost surely.

Rewriting gives

$$\sum_{k=1}^n c_k - \sum_{k=1}^n a_k Z_k = Z_{n+1} - Y_{n+1} \geq -Y_{n+1}.$$

Since the right hand side converges almost surely and the series $\sum_{k=1}^n c_k$ is assumed to be convergent we can deduce that $\sum_{k=1}^{\infty} a_k Z_k$ converges. Since $\sum_{k=1}^{\infty} a_k = \infty$ almost surely by assumption we can deduce that $\lim_{n \rightarrow \infty} Z_n = 0$ almost surely. \square



Lemma 4.3.5. Suppose $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_t))$ is a filtered probability space carrying all appearing random variables. Suppose

- $\varepsilon(t)$ are \mathcal{F}_{t+1} -measurable with $\mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0$ and $\mathbb{E}[\varepsilon^2(t) | \mathcal{F}_t] \leq C$ for all $t \in \mathbb{N}$,



- the step-sizes $\alpha(t)$ are adapted with

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty$$

almost surely.

Then the stochastic process W defined by some \mathcal{F}_0 -measurable initial condition $W(0)$ and the recursion

$$W(t+1) = (1 - \alpha(t))W(t) + \alpha(t)\varepsilon(t), \quad t \in \mathbb{N},$$

converges to 0 almost surely.

Proof. We are going to apply the Robbins-Siegmund theorem to the sequence W^2 :

$$\begin{aligned} \mathbb{E}[W(t+1)^2 \mid \mathcal{F}_t] &= \mathbb{E}[(1 - \alpha(t))^2 W^2(t) + \alpha^2(t)\varepsilon^2(t) + 2\alpha(t)(1 - \alpha(t))W(t)\varepsilon(t) \mid \mathcal{F}_t] \\ &= (1 - 2\alpha(t) + \alpha^2(t))W^2(t) + \mathbb{E}[\alpha^2(t)\varepsilon^2(t) + 2\alpha(t)(1 - \alpha(t))W(t)\varepsilon(t) \mid \mathcal{F}_t] \\ &\leq (1 - a_t + b_t)W^2(t) + c_t, \end{aligned}$$

with $a_t = -2\alpha(t)$, $b_t = \alpha^2(t)$, and $c_t = \alpha^2(t)C^2$. Now the claim follows from Robbins-Siegmund. \square

Note that assuming bounded conditional second moments makes our lives much easier, the theorem also holds under much less restricted assumption on the error. The condition is very strong and typically not satisfied for applications of stochastic approximation. Nonetheless, for convergence of Q-learning and TD-algorithms the assumption will turn out to be enough.



Lemma 4.3.6. Suppose $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_t))$ is a filtered probability space carrying all appearing random variables. Suppose the step-sizes $\alpha(t)$ are adapted with $\sum_{t=1}^{\infty} \alpha(t) = \infty$ almost surely. If A is a non-negative constant, then the non-negative stochastic process Y defined by some \mathcal{F}_0 -measurable initial condition $Y(0) \geq 0$ and the recursion

$$Y(t+1) = (1 - \alpha(t))Y(t) + \alpha(t)A, \quad t \in \mathbb{N},$$

converges to A almost surely as $t \rightarrow \infty$.

Proof. The non-negativity of Y follows from the recursive definition. Subtracting and iterating yields

$$Y(t+1) - A = (1 - \alpha(t))(Y(t) - A) = \dots = \prod_{s=1}^t (1 - \alpha_s)(Y(0) - A).$$

The right hand side vanishes almost surely in the limit because $\sum_{t=1}^{\infty} \alpha_t = \infty$ almost surely:

$$\log \left(\prod_{s=1}^t (1 - \alpha(s)) \right) = \sum_{s=1}^t \log(1 - \alpha(s)) \leq \sum_{s=1}^t -\alpha(s) \rightarrow -\infty, \quad t \rightarrow \infty.$$

\square

Before proceeding with asynchronous stochastic approximation let us think for a moment about the previous two lemmas. The formulation is extremely robust, all random variables are only assumed to be measurable, there is no more precise assumption. Also it is important to note

that the recursive algorithms are very explicit. For example, there is no problem looking at the processes started at times (random or deterministic) different from 0, the convergence will still hold. In the proofs we will for instance use the notation $(W(t : t_0))_{t \geq t_0}$ for the stochastic process with a given initial condition $W(t_0 : t_0)$, similarly for Y , at time t_0 and recursive update

$$W(t + 1 : t_0) = (1 - \alpha(t))W(t : t_0) + \alpha(t)\varepsilon(t), \quad t \geq t_0. \quad (4.5)$$

It obviously holds that $W = W(\cdot : 0)$. The convergence property of $W(\cdot : t_0)$ is identical to that of $W(\cdot : 0)$. Since different $W(\cdot : t_0)$ are defined through the same error variables these processes can be compared on a pathwise level („coupled“), a feature that will be crucial to prove asynchronous stochastic approximation below. Here is a first useful property:



Lemma 4.3.7. In the setting of Lemma 4.3.5 we consider for all $t_0 \in \mathbb{N}$ the stochastic processes $(W(t : t_0))_{t \geq t_0}$ from (4.5) with initial conditions $W(t_0 : t_0) = 0$. For every $\delta > 0$ there exists some random variable T with values in \mathbb{N} such that $|W(t : t_0)| < \delta$ for all $T \leq t_0 \leq t$ almost surely.

Proof. From Lemma 4.3.5 we know that $\lim_{t \rightarrow \infty} W(t : 0) = 0$ almost surely. Next, we note that

$$\begin{aligned} W(t : 0) &= (1 - \alpha(t - 1))W(t - 1 : 0) + \alpha(t - 1)\varepsilon'(t - 1) \\ &= (1 - \alpha(t - 1))(1 - \alpha(t - 2))W(t - 2 : 0) + \alpha(t - 2)\varepsilon(t - 2) + \alpha(t - 1)\varepsilon(t - 1) \\ &= \dots \\ &= \prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : 0) + \sum_{s=t_0}^{t-1} \alpha(s) \prod_{k=s}^{t-2} (1 - \alpha(k))\varepsilon(s) \\ &= \prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : 0) + \underbrace{\prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : t_0)}_{=0} + \sum_{s=t_0}^{t-1} \alpha(s) \prod_{k=s}^{t-2} (1 - \alpha(k))\varepsilon(s) \\ &= \underbrace{\prod_{s=t_0}^{t-1} (1 - \alpha(s))}_{\leq 1} W(t_0 : 0) + W(t : t_0) \end{aligned}$$

holds for all $t > t_0$. Hence, $|W(t : t_0)| \leq |W(t : 0)| + |W(t_0 : 0)|$ almost surely, i.e. $W(t : t_0)$ can be controlled by $W(t : 0)$. Since for every $\omega \in \Omega$, $W(t : 0)(\omega)$ vanishes in the limit we can chose $T(\omega)$ large enough so that $|W(t : 0)(\omega)| < \frac{\delta}{2}$ for all $t \geq T(\omega)$. This implies the claim. \square

For didactic reasons we next prove a theorem on fixed points of real-valued contractions. In a way the theorem is an approximate version of Banachs fixedpoint iteration set up for situations in which the contraction operator cannot be computed explicitly but only with an error. The theorem is not particularly interesting itself (do you know any interesting contraction on \mathbb{R} ?), but it will become much more interesting when extended to $(\mathbb{R}^d, \|\cdot\|_\infty)$ to prove convergence of temporal difference schemes.



Theorem 4.3.8. (Stochastic fixed point iteration for contractions on \mathbb{R}) Suppose $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Suppose that

- $F : \mathbb{R} \rightarrow \mathbb{R}$ is a contraction, i.e. there is some $\beta < 1$ such that $|F(x) - F(y)| \leq \beta|x - y|$ for all $x, y \in \mathbb{R}$,
- $\varepsilon(t)$ are \mathcal{F}_{t+1} -measurable with $\mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0$ and $\mathbb{E}[\varepsilon^2(t) | \mathcal{F}_t] \leq C$ for all $t \in \mathbb{N}$,

♥

- the step-sizes $\alpha(t)$ are only (!) adapted with

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty.$$

Then the stochastic process defined through some \mathcal{F}_0 -measurable initial condition $x(0)$ and the recursion

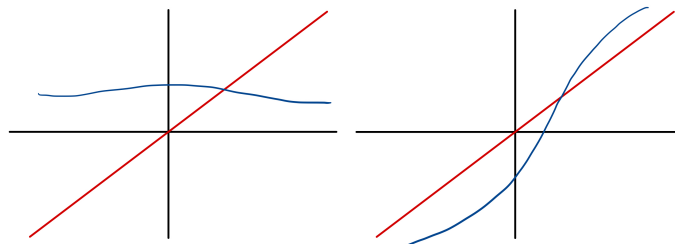
$$x(t+1) = x(t) + \alpha(t)(F(x(t)) + \varepsilon(t) - x(t)), \quad t \in \mathbb{N},$$

converges almost surely to the unique fixed point x^* of F .

Before going into the proof let us quickly motivate why we call the iterative scheme a stochastic fixed point iteration. For known F a Banach's fixed point iteration with approximated F looks like

$$x(t+1) = F(x(t)) = x(t) + (F(x(t)) - x(t)) \approx x(t) + (F(x(t)) + \varepsilon(t) - x(t)).$$

Little surprisingly the scheme would not converge as the errors are not assumed to diminish and the scheme would fluctuate around x^* without converging. This is circumvented by decreasing the update size using $\alpha(t)$. Knowing other approximation schemes that find zeros or fixed points of real-valued functions one might ask why the scheme is so simple. For instance, there are no derivatives in the update scheme. The reason is the particularly simple class of functions. Real-valued contractions are simple functions, the bisecting line can never be crossed bottom-up, it must be crossed downwards (see the drawing).



contraction vs. non-contraction

Hence, if $x > x^*$, then $F(x)$ is below the dissecting line implying $F(x) - x < 0$. Similarly, $F(x) - x > 0$ if $x < x^*$. Thus, the scheme is set up such that $x(t)$ is pushed left if $x(t) > x^*$ and pushed right if $x(t) < x^*$. This little graphic discussion shows why fixed points of contractions (even in the approximate case) can be obtained using very simple approximation schemes that do not involve anything but (approximate) evaluations of the function.

Proof. The main idea of the proof is to compare the stochastic processes x to the stochastic processes W and Y from the previous lemmas. These processes are much simpler and we already know their almost sure limits.

🔗

Without loss of generality we can assume that $F(x^*) = x^* = 0$.

⁶ This follows by defining the auxiliary function $\tilde{F}(x) = F(x + x^*) - x^*$ which has fixed point 0. Suppose the statement holds for \tilde{F} . Then, subtracting x_* on both sides of the recursion yields

$$x(t+1) - x^* = x_i(t) - x^* + \alpha(t)((\tilde{F}(x(t) - x^*) - x^*)) - (x(t) + \varepsilon(t) - x^*), \quad t \in \mathbb{N},$$

⁶nochmal checken

so that $\tilde{x}(t) := x(t) - x^*$ satisfies the original recursion with $x^* = 0$. Since convergence of the shifted recursion to 0 is equivalent to convergence of the original recursion to x^* we may assume $x^* = 0$. Assuming $x^* = 0$ is useful as we can use the estimate

$$|F(x)| = |F(x) - 0| = |F(x) - F(x^*)| \leq \beta|x - x^*| = \beta|x|$$

below.



$\sup_{t \geq 0} |x(t)|$ is finite almost surely, say by the (random) constants D_0 .

⁷ First chose $\eta > 0$ such that $\beta(1 + \eta) = 1$ and define $M(t) := \max\{1, \max_{s \leq t} |x(s)|\}$ as well as $\bar{\varepsilon}(t) = \frac{\varepsilon(t)}{M(t)}$. Then $\bar{\varepsilon}(t)$ is again \mathcal{F}_{t+1} -measurable with

$$\mathbb{E}[\bar{\varepsilon}(t) | \mathcal{F}_t] = \frac{1}{M_t} \mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0 \quad \text{and} \quad \mathbb{E}[\bar{\varepsilon}^2(t) | \mathcal{F}_t] \leq \frac{C}{M_t} < C.$$

Hence, $\bar{\varepsilon}$ is again an error sequence to which the above lemmas apply. Now let $\bar{W}(\cdot : t_0)$ the recursively defined process from Lemma 4.3.7 started in $\bar{W}(t_0 : t_0) = 0$. Since $\lim_{t \rightarrow \infty} \bar{W}(t : t_0) = 0$ almost surely there is some t_0 such that $|\bar{W}(t : t_0)| < \eta$ for all $t \geq t_0$. We are next going to show that

$$|x(t)| \leq M(t_0) + \bar{W}(t : t_0)M(t_0) \leq (1 + \eta)M(t_0), \quad t \geq t_0, \quad (4.6)$$

holds almost surely. By the choice of t_0 we only need to prove the first inequality, which we will achieve by induction in $t \geq t_0$.

Induction start: The inequality clearly holds for $t = t_0$ as $\bar{W}(t_0 : t_0) = 0$ by definition and $x(t) \leq M(t)$ holds for every $t \in \mathbb{N}$.

Induction step: For the induction step we will use the recursion to transfer the inequality from t to $t + 1$:

$$\begin{aligned} x(t+1) &= (1 - \alpha(t))x(t) + \alpha(t)F(x(t)) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{contraction}}{\leq} (1 - \alpha(t))x(t) + \alpha(t)\beta|x(t)| + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{induction}}{\leq} (1 - \alpha(t))(M(t_0) + \bar{W}(t : t_0)M(t_0)) + \alpha(t)\beta(1 + \eta)M(t_0) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{rearranging}}{=} ((1 - \alpha(t))\bar{W}(t : t_0) + \alpha(t)\bar{\varepsilon}(t))M(t_0) + (1 - \alpha(t))M(t_0) + \alpha(t)\underbrace{\beta(1 + \eta)}_{=1}M(t_0) \\ &= \bar{W}(t+1 : t_0)M(t_0) + M(t_0) \leq (1 + \eta)M(t_0). \end{aligned}$$

That's it. Since $(x_t)_{t \leq t_0}$ is clearly bounded the upper bound for $t \geq t_0$ from (4.6) shows that x is bounded.

For the next step fix some $\varepsilon > 0$ such that $\beta(1 + 2\varepsilon) < 1$ and define recursively $D_{k+1} = \beta(1 + 2\varepsilon)D_k$ so that $\lim_{k \rightarrow \infty} D_k = 0$.



There is a (random) sequence $t_k \rightarrow \infty$ such that $\sup_{t \geq t_k} |x(t)| \leq D_k$ almost surely.

Note that $\lim_{k \rightarrow \infty} D_k = 0$, hence, the proof of the theorem is complete once the claim is proved. The proof of this claim is carried out by induction.

Induction start: Set $t_0 = 0$, then the claim holds as D_0 is the global upper bound of x .

Induction step: Suppose t_k is given. We now use the auxiliary processes from the lemmas above, but started at different times than time 0. The lemmas did not use anything but measurability and equally apply to changed starting times. They also apply to random starting times using

⁷da passt ein t_0 noch nicht

the same error functions in the recursions. Let $W(\cdot : s)$ the process from Lemma 4.3.5 started at time s in 0 and

$$\tau := \min \{s \geq t_k : W(t : s) < \beta \varepsilon D_k \forall t \geq s\}.$$

Note that $\tau < \infty$ as $\lim_{t \rightarrow \infty} W(t : s) = 0$ almost surely. Furthermore, denote by $Y(\cdot : \tau)$ the process from Lemma 4.3.6 with $A = \beta D_k$ but started at time τ at D_k . The crucial inequality of the proof is the following:

$$|x(t) - W(t : \tau)| \leq Y(t : \tau), \quad \forall t \geq \tau, \quad (4.7)$$

where $Y(\cdot : \tau)$ runs for $t \geq \tau$ and is started in $Y(\tau : \tau) = D_k$. This again is proved by induction over $t \geq \tau$. The induction start $t = \tau$ holds by definition of τ because $W(\tau, \tau) = 0$, $Y(\tau : \tau) = D_k$, and $|x(t)| \leq D_k$ for all $t \geq t_k$. Next, suppose the estimate holds for some $t \geq \tau$. We use the defining recursions and the contraction property of F to prove the estimate for $t + 1$:

$$\begin{aligned} x(t+1) &= (1 - \alpha(t))x(t) + \alpha(t)F(x(t)) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{Ind., contraction}}{\leq} (1 - \alpha(t))(Y(t : \tau) + W(t : \tau)) + \alpha(t)\beta|x(t)| + \alpha(t)\varepsilon(t) \\ &\stackrel{t \geq t_k}{\leq} (1 - \alpha(t))(Y(t : \tau) + W(t : \tau)) + \alpha(t)\beta D_k + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{rearranging}}{=} (1 - \alpha(t))Y(t : \tau) + \alpha(t)\beta D_k + (1 - \alpha(t))W(t : \tau) + \alpha(t)\varepsilon(t) \\ &= Y(t+1 : \tau) + W(t+1 : \tau) \end{aligned}$$

In the same way we can show $-Y(t+1 : \tau) + W(t+1 : \tau) \leq x(t+1)$. This proves (4.7).

The reverse triangle inequality ($|a| - |b| \leq |a - b|$) applied to (4.7) yields

$$|x(t)| \leq Y(t : \tau) + |W(t : \tau)|, \quad \forall t \geq \tau.$$

Since $W(t : \tau) \leq \beta \varepsilon D_k$ for all $t \geq \tau$ and $Y(\cdot : \tau)$ converges to βD_k there is some index $t_{k+1} > \tau \geq t_k$ for which $|x(t)| < \beta \varepsilon D_k + (1 + \varepsilon)\beta D_k = D_{k+1}$ for all $t \geq t_{k+1}$. This proves the claim and the proof is complete. \square

We continue with a proof of the main result of interest, the asynchronous stochastic approximation theorem from which convergences of temporal difference schemes follows readily. The wording asynchronous comes from the fact that the step-sizes α_i do not need to be the same, the coupling between the equations only comes through F .



Theorem 4.3.9. (Asynchronous stochastic fixed point iteration for $\|\cdot\|_\infty$ -contractions on \mathbb{R}^d)

Suppose $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Suppose that

- $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a contraction with respect to $\|\cdot\|_\infty$, i.e.

$$\|F(x) - F(y)\|_\infty \leq \beta \|x - y\|_\infty, \quad \forall x, y \in \mathbb{R}^d,$$

for some $\beta \in (0, 1)$,

- $\varepsilon_i(t)$ are \mathcal{F}_{t+1} -measurable with $\mathbb{E}[\varepsilon_i(t) | \mathcal{F}_t] = 0$ and there is some $C > 0$ with $\sup_{i,t} \mathbb{E}[\varepsilon_i^2(t) | \mathcal{F}_t] < C$,
- the step-sizes $\alpha_i(t)$ are adapted with

$$\sum_{t=1}^{\infty} \alpha_i(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_i^2(t) < \infty.$$



Then for any \mathcal{F}_0 -measurable initial condition $x(0)$ the (asynchronous) stochastic approximation scheme

$$x_i(t+1) = x_i(t) + \alpha_i(t)(F_i(x(t)) - x_i(t) + \varepsilon_i(t)), \quad t \in \mathbb{N},$$

converges almost surely to the unique fixed point $x^* \in \mathbb{R}^d$ of F .

Proof. The main point is to realise that the proof of the real-valued case can be extended analogously to the multi-dimensional situation. One only needs to be careful where the fixed point property of F (the coordinates are not necessarily contractions) is used and where the coupling of the equations through F appears. The first is surprisingly simple, the fixed point property was hardly used in the previous proof. It is only used to shift x^* to 0 and to ensure F growth at most linearly, thus, the recursion is bounded.



Without loss of generality we can assume that $F(x^*) = x^* = 0$.

The argument is exactly the same as in the previous proof.



$\sup_{t \geq 0} |x(t)|$ is finite almost surely.

The argument is exactly the same using the same η, β , and M . Then one defines the modified errors $\bar{\varepsilon}_i = \frac{\varepsilon_i}{M}$ and from this the recursions W_i . The contraction with respect to the supremum norm is used to estimate

$$|F_i(x(t))| = |F_i(x(t)) - F_i(x^*)| \leq \beta \|x(t) - x^*\|_\infty = \beta \|x(t)\|_\infty \leq \beta M t,$$

allowing us to estimate in the same way

$$x_i(t+1) \leq W_i(t+1 : t_0)M(t_0) + M(t_0) \leq (1 + \eta)M(t_0).$$

Hence, every coordinate process x_i is bounded, thus, also x is bounded.

Comparing with the rest of the previous proof F can be removed in the same way, only the linear growth is used to reduce to coordinate wise auxiliary processes W_i and Y_i and the argument works equally.



Go through the one-dimensional proof to check that there is a sequence $t_k \rightarrow \infty$ such that $\sup_{t \geq t_k} |x(t)| \leq D_k$ almost surely and $\lim_{k \rightarrow \infty} D_k = 0$.

Again, the argument is line by line the same for each coordinate x_i . Only τ needs to be defined as

$$\tau := \min \{s : W_i(t : s) < \beta \varepsilon D_k \forall i = 1, \dots, d, t \geq s\}$$

to allow the same estimate for each x_i . □

The theorem is called asynchronous because the step-sizes $\alpha_i(t)$ can be different for different coordinates in contrast to the synchronous case where they are equal. Most importantly, if only one coordinate of $\alpha(t)$ is non-zero than only that coordinate is updated in step t . Such extreme case will be called totally asynchronous and we have already crossed such an update scheme in Algorithm 9 in which coordinates were updated by passing through one coordinate after the other.



We have only proved convergence of stochastic fixed point iterations. The proofs do not contain any hint on the quality of convergence. Indeed, stochastic approximation schemes converge terribly slow which is not surprising given the law of large number



is a special case (Example (4.3.3)). We only keep in mind that errors with small variance will typically lead to faster (but still slow) convergence.

For reinforcement learning the setting of Example 4.3.2 will become relevant because Bellman operators can be written as

$$F_i(x) = \mathbb{E}_i[f(x, Z)] \quad (4.8)$$

so that approximations of the expectations using samples \tilde{Z}_i yield model-free learning (approximation) algorithms of the form

$$x_i(t+1) = x_i(t) + \alpha_i(t)(f(x(t), \tilde{Z}_i) - x_i(t)) \quad (4.9)$$

that converge almost surely because they can be rewritten as

$$x_i(t+1) = x_i(t) + \alpha_i(t)(F_i(x(t)) - x_i(t) + \varepsilon_i(t))$$

with the unbiased error terms $\varepsilon_i(t) := f(x(t), \tilde{Z}_i) - F_i(x(t))$. The algorithms will be asynchronous through the choice of α that determines what coordinate to update and how strong the update effect should be. Using different fixed point equations (for V^π , Q^π , V^* , Q^*) and different representations of F as expectation yields different algorithms with different advantages/disadvantages.



Many reinforcement learning algorithms are random versions of Banach's fixed point iteration to solve Bellman equations. The algorithms are asynchronous in the sense that the update of coordinates is not synchronous (i.e. all step-sizes α_i are equal). Typically one is interested in the situation in which the vector α_t is non-zero for exactly one entry. In this situation the vector $x(t+1)$ gets an update in one coordinate only. We refer to this situation as **totally asynchronous** and every asynchronous algorithm needs to clarify where to update and what step-sizes to choose. The asynchronous choice of α allows to balance the exploration of new states or state-action pairs. The second Robbins-Monro condition implies that such a totally asynchronous algorithm can only work if every state is visited infinitely often.

4.4 One-step approximate dynamic programming TD(0)

The first class of algorithms is extracted readily from writing Bellman operators as one-step expectation:

$$\begin{aligned} T^\pi V(s) &= \sum_{a \in \mathcal{A}} \pi(a; s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V(s') \right) = \mathbb{E}_s^\pi [R(s, A_0) + \gamma V(S_1)] \\ T^\pi Q(s, a) &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi(a'; s) Q(s', a') = \mathbb{E}_s^\pi [R(s, a) + \gamma Q(S_1, A_1)] \\ T^* Q(s, a) &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q(s', a') = \mathbb{E}_s^\pi [R(s, a) + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')]. \end{aligned}$$

These expressions are exactly of the form (4.8), thus, approximate fixed point iterations that only use one-step MDP samples can be used that converge to the unique fixed points. In the following we present following variants:

- SARS policy estimation of V^π ,
- SARSA policy estimation of Q^π based on,

- Q-learning and modifications such as SARSA and double-Q-learning.

Convergence proofs are similar, they follow immediately from Theorem 4.3.9 applied to the Bellman operators. More interesting is the practical implementation, i.e. the choice of asynchronous updates through α . Finally, we prove convergence of the generalised policy iteration scheme resulting from SARSA updates of Q^π combined with policy improvement using policies that sufficiently explore but become greedy in the limit.



Compared to the direct Monte Carlo methods the methods presented below are much simpler to use. While Monte Carlo required an entire roll-out, here every update step only requires one step forwards of the MDP. This is very much like dynamic programming but in a random fashion since not every state (or state-action pair) is used for the update. Such methods are called temporal difference methods, but this will become clearer in the sections below where we discuss algorithms that use several steps.

4.4.1 One-step policy evaluation (approximate policy evaluation)

We start with two simple model-free algorithms for evaluation of V^π and Q^π . In order to understand the tremendous freedom that stochastic approximation allows let us first formulate a general theorem and then transform them into useful algorithms.



Theorem 4.4.1. (Convergence of one-step policy evaluation for V^π)
Suppose $\pi \in \Pi_S$ and $V_0 \in \mathbb{R}^{|S|}$ is arbitrary and the asynchronous update rule is

$$V_{n+1}(s) = V_n(s) + \alpha_n(s)((R(s, a) + \gamma V_n(s')) - V_n(s)), \quad s \in S,$$

with $a \sim \pi(\cdot; s)$, $s' \sim p(\cdot; s, a)$ and step-sizes α_n that only depend on the past steps and satisfy the Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s) < \infty \quad \text{a.s.}$$

for every $s \in S$. Then $\lim_{n \rightarrow \infty} V_n(s) = V^\pi(s)$ almost surely, for every state $s \in S$.

It is important to note that the stochastic approximation algorithm does not impose any independence assumptions, α_n only needs to be adapted to the algorithm and satisfy the decay properties. All possibilities from synchronous to totally asynchronous are perfectly fine as long as the Robbins-Monro condition are satisfied.

Proof of Theorem 4.4.1. The convergence follows immediately from Theorem 4.3.9 once the algorithm is rewritten in the right way:

$$V_{n+1}(s) = V_n(s) + \alpha_n(s)(F(V_n)(s) - V_n(s) + \varepsilon_s(n))$$

with

$$F(V)(s) := \mathbb{E}_s^\pi[R(s, A_0) + \gamma V(S_1)]$$

and

$$\varepsilon_s(n) := (R(s, a) + \gamma V_n(s')) - \mathbb{E}_s^\pi[R(s, A_0) + \gamma V_n(S_1)].$$

We need to be a bit careful with the filtration. The σ -algebras \mathcal{F}_n is generated by all random variables used to determine $s, a, R(s, a)$ that appear in the definition of $V(0), \dots, V(n)$.

- In the proof of Theorem 3.1.28 it was shown that the Bellman expectation operator $F = T^\pi$ is a $\|\cdot\|_\infty$ -contraction (the proof was for the expectation operator for Q but exactly the same proof works for V).
- The errors $\varepsilon_s(n)$ are \mathcal{F}_{n+1} -measurable (they involve the next state-action pair (s, a)) with $\mathbb{E}[\varepsilon_s(n) | \mathcal{F}_n] = 0$ by definition. The assumed boundedness of the rewards also implies $\sup_{n,s} \mathbb{E}[\varepsilon_s^2(n) | \mathcal{F}_n] \leq C$.
- The step-sizes are adapted and satisfy $\sum_{n=1}^{\infty} \alpha_s(n) = \infty$ a.s. and $\sum_{n=1}^{\infty} \alpha_s^2(n) < \infty$ a.s. by assumption.

Hence, Theorem 4.3.9 can be applied and we get almost sure convergence to the unique fixpoint of F which is V^π . \square

Let us turn the special situation of totally asynchronous updates into a concrete policy evaluation algorithm. There is a lot of freedom how to chose the step-size (it may for instance depend on the number of past updates of the coordinate as in the UCB bandit algorithm) and the coordinates. But the principle coordinate update rule must be

$$V_{\text{new}}(S_t) = V_{\text{old}}(S_t) + \alpha((R(S_t, A_t) + \gamma V_{\text{old}}(S_{t+1})) - V_{\text{old}}(S_t))$$

for a and s' chosen by the algorithm. Here is the simplest version of one-step value estimation in which we decide to chose a and s' from the value update as next coordinate to be updated. So,

Algorithm 18: SARS policy evaluation

Data: Policy $\pi \in \Pi_S$

Result: Approximation $V \approx V^\pi$

Initialize $V \equiv 0$ and s arbitrary (for instance uniformly).

while *not converged* **do**

$a \sim \pi(\cdot; s)$

Sample reward $R(s, a)$.

Sample next state $s' \sim p(\cdot; s, a)$.

Determine stepsize $\alpha = \alpha(s)$.

Update $V(s) = V(s) + \alpha(R(s, a) + \gamma V(s')) - V(s)$

Set $s = s'$.

end

if we start in an initial state s , we sample $a \sim \pi(\cdot; s)$ and $s' \sim p(\cdot; s, a)$. Then, we update the value as given above. Then, to move on and to find a new suitable initial state, we take s' as new initial state and repeat the same update scheme. Since the scheme uses S - A - R - S' to update the value of s we call the algorithm SARS policy evaluation.

There is a situation in which the above algorithm seems unsuitable. If the MDP terminates in finite times then states are not visited infinitely often so that the second Robbins-Monro condition is violated. Since the order of choosing states is completely irrelevant (this is governed by α that only needs to be measurable) for convergence of stochastic approximation we can just repeatedly take roll-out of finite length. As usually, a similar estimation procedure works for Q . The disadvantage is that the algorithm requires a larger table to store all the values, the advantage is to get hold of Q -values that can be used for policy improvement. The analogous $TD(0)$ -algorithm is based on

$$T^\pi Q(s, a) = \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma Q(S_1, A_1)]$$

It is not hard to guess that updates of the form

$$Q_{\text{new}}(S_t, A_t) = Q_{\text{old}}(S_t, A_t) + \alpha((R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})) - Q_{\text{old}}(S_t, A_t)),$$

where now every updated state-action pair requires a sample of the next state-action pair.

Algorithm 19: SARS policy evaluation with termination

Data: Policy $\pi \in \Pi_S$
Result: Approximation $V \approx V^\pi$
Initialize $V \equiv 0$
while *not converged* **do**
 Initialize s arbitrarily (for instance uniformly).
 while *s not terminal* **do**
 $a \sim \pi(\cdot; s)$
 Sample reward $R(s, a)$.
 Sample next state $s' \sim p(\cdot; s, a)$.
 Determine stepsize $\alpha = \alpha(s)$.
 Update $V(s) = V(s) + \alpha((R(s, a) + \gamma V(s')) - V(s))$.
 Set $s = s'$.
 end
end

**Theorem 4.4.2. (Convergence of one-step policy evaluation for Q^π)**

Suppose $Q_0 \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ is arbitrary and, for $n \in \mathbb{N}$, the asynchronous update rule is

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a)((R(s, a) + \gamma Q_n(s', a')) - Q_n(s', a)).$$

We assume for every n that $s'(s, a) \sim p(\cdot; s, a)$ and $a'(s, a) \sim \pi(\cdot; s'(s, a))$, as well as α_n depend only on the past steps and satisfy the Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_n(s, a) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s, a) < \infty \quad \text{a.s.}$$

for every $(s, a) \in \mathcal{S} \times \mathcal{A}$. Then $\lim_{n \rightarrow \infty} Q_n(s) = Q^\pi(s)$ almost surely, for every state-action pair (s, a) .

Again, note that there are no assumption whatsoever on the dependencies of α_n , they only need to depend on the past only (be adapted). If the matrix α_n only has one non-zero entry then the procedure is again called totally asynchronous.

Proof. Let us denote by $(\tilde{S}_0, \tilde{A}_0, \dots)$ the state-action pairs defined by the algorithm. Furthermore, rewrite the updates as

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a)(F(Q_n)(s, a) - Q_n(s, a) + \varepsilon_n(s, a)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

with

$$F(Q)(s, a) := \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma Q(S_1, A_1)]$$

and

$$\varepsilon_n(\tilde{S}_n, \tilde{A}_n) := (R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1})) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(S_1, A_1)].$$

and $\varepsilon_n(s, a) = 0$ for $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$.

- In the proof of Theorem 3.1.28 it was shown that the Bellman expectation operator $F = T^\pi$ is a $\|\cdot\|_\infty$ -contraction.
- The errors $\varepsilon_n(s)$ are \mathcal{F}_{n+1} -measurable with $\mathbb{E}[\varepsilon_n(s) | \mathcal{F}_n] = 0$ by definition. The assumed boundedness of the rewards also implies $\sup_{n,s} \mathbb{E}[\varepsilon_n^2(s) | \mathcal{F}_n] \leq C$.

- The step-sizes are adapted and satisfy $\sum_{n=1}^{\infty} \alpha_n(s) = \infty$ a.s. and $\sum_{n=1}^{\infty} \alpha_n^2(s) < \infty$ a.s. by assumption.

Hence, Theorem 4.3.9 can be applied and we get almost sure convergence to the unique fixpoint of F which is Q^π . \square

Similarly to the state value function a totally asynchronous algorithm is derived from the theorem. Now, we have to initialize a state-action pair, sample the reward and next state s' . The following action is then chosen according to the given policy and inserted into the estimate of the state-value function at the matrix position (s', a') . The state-action pair (s', a') is used for the next update except s' is a terminal state. Since the update uses S - A - R - S' - A' the policy algorithm is called SARSA. Here we only give the version with termination. In all algorithms

Algorithm 20: SARSA policy evaluation for Q^π

Data: Policy $\pi \in \Pi_S$
Result: Approximation $Q \approx Q^\pi$
Initialize $Q(s, a) = 0$ for all $s \in S$ $a \in A$
while not converged do
 Initialize s .
 Sample $a \sim \pi(\cdot; s)$.
 while s not terminal do
 Sample reward $R(s, a)$.
 Sample next state $s' \sim p(\cdot; s, a)$.
 Sample next action $a' \sim \pi(\cdot; s')$.
 Determine step-size α .
 Update $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma Q(s', a')) - Q(s, a))$
 Set $s = s', a = a'$.
 end
end

above (and below) that depend on stochastic fixed point iterations the step-size α needs to be set. For totally asynchronous learning there are two things that need to be specified for the algorithms:

- Which coordinates should be updated, i.e. which state or state-action pair is the only coordinate for which $\alpha(n)$ is non-zero.
- How strongly should the coordinate be updated, i.e. what is the value $\alpha_i(n)$ for the non-zero coordinate?

The coordinate to be updated was already specified in the algorithms by s' for SARS (resp. (s', a') for SARSA). Due to the Robbins-Monro condition every state (resp. state-action pair) must be visited infinitely often to ensure the values of $\alpha_i(n)$ might sum-up to $+\infty$. But what is good choice for the non-zero value of $\alpha(n)$?



Even though invalid for the theory a popular choice for the step-size is a constant value α . The most obvious choice to guarantee convergence of the algorithms is, for some $\frac{1}{2} < p \leq 1$,

$$\alpha_s(n) = \frac{1}{(T_s(n) + 1)^p} \quad \text{resp.} \quad \alpha_{(s,a)}(n) = \frac{1}{(T_{(s,a)}(n) + 1)^p}$$

if $T_s(n)$ denotes the number of times the state s was updated during the first n updates (resp. $T_{(s,a)}(n)$ the number of times the state-action pair (s, a) was updated during the first n updates). The choice is reasonable because if states (or



state-action pairs) are visited infinitely often, then

$$\sum_{n=1}^{\infty} \alpha_s(n) = \sum_{n=1}^{\infty} \frac{1}{n^p} = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_s^2(n) = \sum_{n=1}^{\infty} \frac{1}{n^{2p}} < \infty.$$

4.4.2 Q-learning and SARSA (approximate value iteration)

We continue with the most famous tabular control algorithms. Approximate value iteration to solve $T^*Q = Q$ by iterating in an approximate way Bellman's state-action optimality operator T^* . Such algorithms directly learn the optimal state-action function Q^* without repeated policy improvement steps, they are an approximate form of value iteration for Q . The main advantage (but also disadvantage) of the Q -learning algorithm (and its modifications) is the flexibility, there are plenty of choices that can be made to explore the state-action space.



Definition 4.4.3. A learning algorithm is called **off-policy** if the choice of actions is not governed by the currently estimated policy but the exploration of actions can be driven by outside (random) sources. In contrast, an algorithm is called **on-policy** if the exploration only uses the currently estimated policy to explore actions. In many off-policy examples the actions are explored using a fixed policy that is typically called a **behavior policy**.

The term off-policy is not always used in the same way, also our definition is relatively vague. The most extreme scenario is that of a behavior policy that does not take into account the algorithmic learning at all. To have a clear idea in mind consider the following example. There are two ways of learning chess. Either observing repeated matches of other players or by repeatedly playing by oneself. The first is a typical off-policy situation, learning is accomplished by looking over someone's shoulders. Same with learning how to use a tool. Either looking over someone's shoulders or trying the tool oneself. This illustration already shows the advantages and disadvantages. A disadvantage of off-policy learning is that it might be hard to learn from a bad player and more efficient to learn from a player that plays good and bad moves. In contrast, on-policy learning might lead to little exploration of actions and the algorithm focuses too much on what the algorithm believes to be good, similarly to the exploration-exploitation trade-off for stochastic bandits.



From the mathematics of convergence proofs the situation is relatively simple. In totally asynchronous approximate fixed point iterations for Q the vector α governs the choice of state-action pairs. Since the sequence (α_n) only needs to be adapted to the algorithm (its filtration) there is a lot of freedom on how to explore the actions. Essentially, we can explore in all ways that do not use future knowledge of the algorithm (how should we?) and explores all state-action pairs infinitely often.

No doubt, the remark only concerns convergence in the limit but not the speed of convergence. The mathematical theory of approximate learning of state-action value functions is not particularly well developed. In practise, algorithms work differently well on different examples. We start with the most direct algorithm, the approximate fixed point iteration with samples of Bellman's state-action optimality operator: Q -learning - Algorithm 22. There is a lot that can be changed about Q -learning. To understand why let us check a convergence proof:



Theorem 4.4.4. (Convergence of Q-learning)

Suppose that a behavior policy π is such that all state-action pairs are visited infinitely often and the step-sizes are adapted and satisfy the Robbins-Monro conditions. Then $\lim_{t \rightarrow \infty} Q_t(s, a) = Q^*(s, a)$ holds almost surely for all state-action pairs (s, a) .

Algorithm 21: Q-learning (with behavior policy)

Data: Behavior policy $\pi \in \Pi_S$
Result: Approximations $Q \approx Q^*$, $\pi = \text{greedy}(Q) \approx \pi^*$
Initialize Q (e.g. $Q \equiv 0$).
while *not converged* **do**
 Initialize s .
 while s *not terminal* **do**
 Sample $a \sim \pi(\cdot; s)$.
 Sample reward $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 Determine stepsize α .
 Update $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma \max_{a' \in \mathcal{A}_{s'}} Q(s', a')) - Q(s, a))$.
 Set $s = s'$.
 end
end

Proof. The proof is exactly the same that we have seen above, now using the optimal Bellman operator

$$T^*Q(s, a) = \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')]$$

on $\mathbb{R}^{|\mathcal{S}| \cdot |\mathcal{A}|}$. Please finish the proof yourself as an exercise:



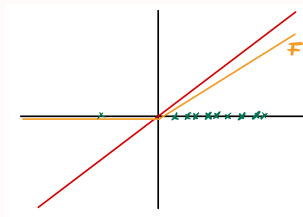
Rewrite the Q-learning algorithm as approximate fixed point iteration and check the conditions of Theorem 4.3.9 to prove the almost sure convergence.

□

Before discussing versions of Q-learning let us think for a moment what Q-learning really does. Q-learning challenges current estimates $Q(s, a)$ of $Q^*(s, a)$ by comparing them with new one-step estimates. If the Q-values are stored in a table, then the algorithm successively explores table entries and updates their values. In every step one entry of the table is updated as follows. The current estimate $Q(s, a)$ is compared to a new estimate $Q(s, a)$, namely, by resampling only the first step (this is $R(s, a)$) and then assuming the estimated future discounted reward from the new state s' is perfect (this is $\max_{a'} Q(s', a')$). The table entry $Q_{\text{new}}(s, a)$ is increased/decreased according to the new estimate $R(s, a) + \max_{a'} Q(s', a')$ for $Q^*(s, a)$ being larger/smaller than the old entry. Unfortunately, there is a huge drawback of Q-learning: Q-learning converges in the limit but can overestimate the Q-values by a lot. Here is an example that shows what can go wrong.



Stochastic approximation algorithms are not unbiased, i.e. the estimates x_n of x^* typically satisfy $\mathbb{E}[x_n] \neq x^*$. Here is a one-dimensional illustration that roughly captures the overestimation effect of Q-learning. Take $F(x) = \gamma \max\{0, x\}$ for some $\gamma < 1$, this is a contraction.



An overestimating approximate fixedpoint iteration



Without error the algorithm quickly converges from the left to $x^* = 0$ (even in one step if $\alpha = 1$) because the steps are $\alpha_n(F(x_n) - x_n) = -\alpha_n x_n$ while the convergence from the right takes some time as $\alpha_n(F(x_n) - x_n) \approx 0$. With errors the situation becomes worse even with $x_0 = x^*$. A positive error sends x_n to the positives from where the algorithm slowly comes back to 0 while after sending x_n to the negatives with a negative error the algorithm comes back to 0 much faster. The same effect happens for Q -learning. Estimated Q -values are typically too large. Variants of Q -learning that deal with the overestimation effect are discussed in the next section.

For convergence, the way the state-action pairs are chosen can be even more arbitrary as long as all state-action pairs are visited infinitely often and the step-sizes decrease according to the Robbins-Monro conditions. For instance, state-action pairs could also be chosen randomly with equal step-size $\frac{1}{n}$. We know from stochastic bandits that ignoring all learned information is not going to be beneficial. If we are interested in the performance of the algorithm during training, e.g. the total rewards during training (like the regret for bandit algorithms), we want to choose π such that *good* actions are taken more often but still enough exploration of new state-action pairs takes place. Thus, the exploration of table entries is often organised by ε -greedy or Boltzman exploration.

A variant of Q -learning is SARSA which has the following update rule:

$$Q_{\text{new}}(S_t, A_t) = Q_{\text{old}}(S_t, A_t) + \alpha((R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})) - Q_{\text{old}}(S_t, A_t)),$$

where the choice of the next action must be on-policy (determined by Q , such as ε -greedy). The algorithm is not always going to converge, the right trade-off between exploration and exploitation of the best known estimate of Q^* (the current Q) is needed. For ε too large the algorithm will play unfavorable state-action pairs too often, for ε too small the algorithm might miss to learn good actions. Since the updates use S - A - R - S' - A' this on-policy version of Q -learning is called SARSA control.

Algorithm 22: SARSA

Result: Approximations $Q \approx Q^*$, $\pi = \text{greedy}(Q) \approx \pi^*$

Initialize Q , e.g. $Q \equiv 0$.

while *not converged* **do**

 Initialise s, a , e.g. uniform.

while *s not terminal* **do**

 Sample reward $R(s, a)$.

 Chose new policy π from Q (e.g. ε -greedy).

 Sample next state $s' \sim p(\cdot; s, a)$.

 Sample next action $a' \sim \pi(\cdot; s')$.

 Determine stepsize α .

 Update $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma Q(s', a')) - Q(s, a))$.

 Set $s = s', a = a'$.

end

end



SARSA simplifies if the policy update is greedy with respect to Q . In that case $a' = \arg \max_a Q(s', a)$ so that the Q -updates are exactly the updates of Q -learning.

In contrast to Q -learning SARSA cannot be performed off policy.



Suppose for instance the actions a' are chosen uniformly then SARSA would not converge to Q^* . Think about it and try it in simulations!

Unlike Q -learning it is less obvious that SARSA converges to Q^* , in contrast to Q -learning the iteration is not an approximate version of the fixed point iteration for T^* ! The following proof is important as it is used in other settings as well, in the literature one can occasionally see reference to the SARSA convergence proof for modifications of Q -learning (for instance for double- Q and clipping). The idea is to compare to the Q -learning iteration and then show the appearing error vanishes. Unfortunately, our version of approximative fixedpoint iteration only allows unbiased errors while the proof needs unbiased errors $\mathbb{E}[\varepsilon_n | \mathcal{F}_n] \neq 0$. For that sake a generalisation of Theorem 4.3.9 is needed.



Show that the statement of Theorem 4.3.9 also holds with $\mathbb{E}[\varepsilon_i(n) | \mathcal{F}_n] \neq 0$ if

$$\sum_{n=1}^{\infty} \alpha_i(n) |\mathbb{E}[\varepsilon_i(n) | \mathcal{F}_n]| < \infty \quad (4.10)$$

holds almost surely. Going through the proof it will become clear that only Lemma 4.3.5 needs to be improved to this situation. Luckily the proof is short enough to see quickly (estimating $W \leq 1 + W^2$) that (4.10) is enough to reduce the lemma to the Robbins-Siegmund theorem.

In fact, there are even more general versions that allow to suppress the summation condition of the next theorem. Since the most influential paper ⁸ seems to have an error (there is a circular argument in the last paragraph of the paper) we only use what has been developed in these lecture notes for the next convergence proof.



Theorem 4.4.5. (Convergence of SARSA control for terminating MDPs)

Suppose the step-sizes satisfy the Robbins-Monro conditions and the probabilities $p_n(s, a)$ that the policy π_{n+1} is greedy satisfies are such that

$$\sum_{n=1}^{\infty} \alpha_n(s, a) p_n < \infty \quad \text{a.s.}$$

for all (a, s) . Then the SARSA algorithm converges to Q^* almost surely.

The most important policy to which the theorem applies is α_n -greedy because $\sum \alpha_n^2(s, a) < \infty$ holds by assumption. Choosing $\alpha_n(s, a) = \frac{1}{T_{s,a}(n)+1}$ the exploration rate in a state-action pair (s, a) decreases with the number of updates of $Q(a, s)$.

Proof. The proof is different from the ones before. The reason is that the updates are not directly estimates of a contraction operator. Nonetheless, a similar argument works. Adding a zero the algorithm can be reformulated in an approximate fixed point iteration with an error-term that is biased but with a bias decreasing to zero. We will use a variant of Theorem 4.3.9. The convergence also holds if

$$\sum_{n=1}^{\infty} \alpha_n(s, a) |\mathbb{E}[\varepsilon_n(s, a) | \mathcal{F}_n]| < \infty \quad \text{a.s.} \quad (4.11)$$

for all state-action pairs. Hence, we check the condition (4.11) instead of $\mathbb{E}[\varepsilon_n(s, a) | \mathcal{F}_n] = 0$ with an appropriately chosen error-term. Let us denote by $\tilde{S}_0, \tilde{A}_0, \dots$ the sequence of state-action pairs obtained from the algorithm. First, writing

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a) (T^* Q_n(s, a) - Q_n(s, a) + \varepsilon_n(s, a)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

⁸S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms", Machine Learning, 38(3):287–308, 2000

with

$$\varepsilon_n(\tilde{S}_n, \tilde{A}_n) = (R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1})) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [R(\tilde{S}_n, \tilde{A}_n) + \gamma \max_{a'} Q_n(S_1, a')]$$

and $\varepsilon(s, a) = 0$ for $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$. The errors $\varepsilon_n(s, a)$ are \mathcal{F}_{n+1} -measurable. Furthermore, $\mathbb{E}[\varepsilon(s, a) | \mathcal{F}_n] = 0$ for $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$ and

$$\begin{aligned} & \mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ is greedy}\}} (\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ & \quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} (\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ greedy}\}} | \mathcal{F}_n] \underbrace{\mathbb{E}[\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]]}_{=0} | \mathcal{F}_n] \\ & \quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} | \mathcal{F}_n] \mathbb{E}[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} | \mathcal{F}_n] \mathbb{E}[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n]. \end{aligned}$$

Finally, recall that we assume throughout these lecture notes that rewards are bounded. Thus, the assumed boundedness of Q_0 and the iteration scheme combined with $\sum_{k=0}^{\infty} \gamma^k < \infty$ implies that $|Q_n(s, a)| < C$ for some C and all $n \in \mathbb{N}$. Thus,

$$|\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) | \mathcal{F}_n]| \leq C \mathbb{P}(\pi_{n+1}(\cdot; s') \text{ non greedy} | \mathcal{F}_n) = Cp_n(s, a).$$

By assumption, the summation condition 4.11 is satisfied. By the boundedness of rewards, also $\mathbb{E}[\varepsilon_n^2(s, a) | \mathcal{F}_n] \leq C < \infty$. As T^* is a contraction and the Robbins-Monro conditions are satisfied, the iteration converges to Q^* almost surely. \square

To get a feeling of the SARSA algorithm think about stochastic bandits seen as one-step MDPs.



To get a feeling for Q -learning and SARSA try to relate the algorithms with $\alpha_n(s, a) = \frac{1}{T_{s,a}(n)+1}$ to the ε -greedy algorithm for stochastic bandits introduced in Chapter 1.

4.4.3 Double Q -learning

The aim of this section is to introduce double variants of Q -learning that deal with overestimation of Q -learning. For a rough understanding of what goes wrong in Q -learning recall that the $Q_n(s, a)$ are (random) estimates of the expectations $Q^*(s, a)$. In the updates of Q -learning we use the estimates $\max_{a' \in \mathcal{A}_s} Q_n(s, a)$ of $\max_{a' \in \mathcal{A}_s} Q^*(s, a)$ but those overestimate.



Suppose $\hat{\mu}_1, \dots, \hat{\mu}_K$ are estimates for expectations μ_1, \dots, μ_K . Then the pointwise maximum $\max_{i=1, \dots, K} \hat{\mu}_i$ overestimates $\hat{\mu} = \max_{i=1, \dots, K} \mu_i$ because $\mathbb{E}[\max \hat{\mu}_i] \geq \mathbb{E}[\mu_i]$ for all i so that $\mathbb{E}[\max \hat{\mu}_i] \geq \max \mathbb{E}[\mu_i]$. As a simple example suppose M_1, \dots, M_K are $\text{Ber}(p)$ -distributed. They are all unbiased estimators of the mean but

$$\begin{aligned} \mathbb{E}[\max M_i] &= \mathbb{P}(\{M_1 = 0, \dots, M_K = 0\}) \cdot 0 + \mathbb{P}(\{M_1 = 0, \dots, M_K = 0\}^c) \cdot 1 \\ &= 1 - (1 - p)^K > p. \end{aligned}$$

Van Hasselt⁹ suggested to use two-copies of Q -learning and intertwine them such that one estimates the optimal action, the other the optimal value. This leads to the idea of double Q -learning and modifications.

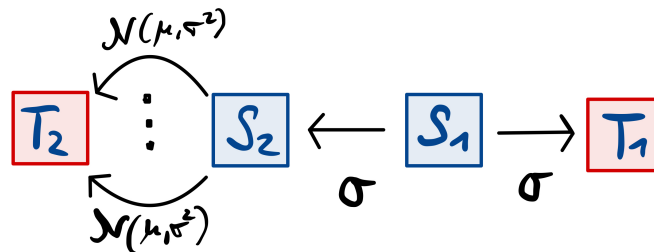
⁹H. van Hasselt, "Double Q-learning", NIPS 2010

Algorithm 23: Double Q-learning (with behavior policy)

Data: Behavior policy $\pi \in \Pi_S$
Result: Approximations $Q \approx Q^*$, $\pi = \text{greedy}(Q) \approx \pi^*$
Initialize Q^A, Q^B (e.g. 0).
while *not converged* **do**
 Initialise s .
 while s *not terminal* **do**
 Sample $a \sim \pi(\cdot; s)$.
 Sample reward $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 Determine stepsize α .
 Randomly choose $\text{update} = A$ or $\text{update} = B$
 if $\text{update} = A$ **then**
 $a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$
 Update $Q^A(s, a) = Q^A(s, a) + \alpha(R(s, a) + \gamma Q^B(s', a^*) - Q^A(s, a))$
 end
 else
 $b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$
 Update $Q^B(s, a) = Q^B(s, a) + \alpha(R(s, a) + \gamma Q^A(s', b^*) - Q^B(s, a))$
 end
 Set $s = s'$.
 end
end

Similarly to SARSA (see the proof of Theorem 4.4.7) double Q-learning can be interpreted as classical Q-learning with an additional negatively biased term $\hat{\epsilon}$. We will not prove convergence of double Q-learning. The proof requires a stronger unbiased version of approximate dynamic programming. Instead we will introduce a truncation which allows us to follow the proof of convergence for SARSA. Interestingly, our new version of double Q-learning performs better than Q-learning and double Q-learning on some of the standard examples. Before stating the truncated double Q-learning algorithm let us check an example:

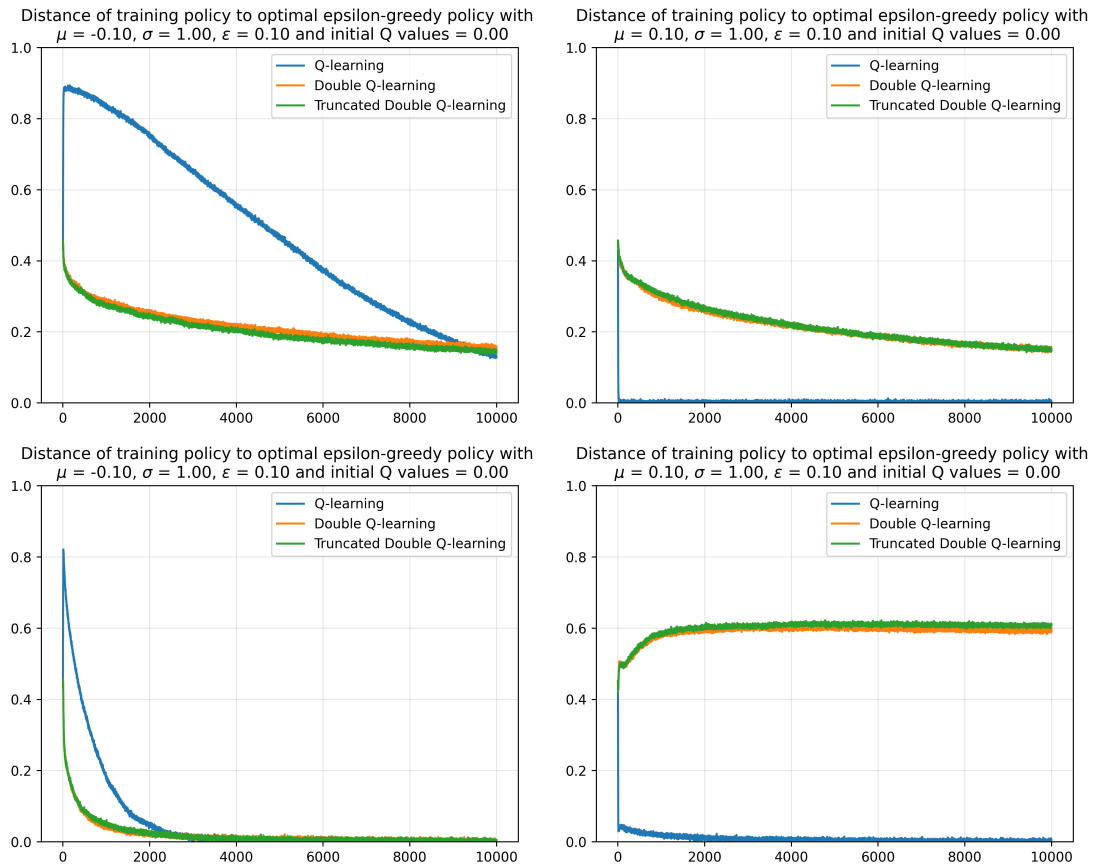
Example 4.4.6. The example MDP consists of two interesting states S_1, S_2 and two terminal states T_1, T_2 . The transition probabilities are indicated in the graphical representation. If action left/right is chosen in S_1 then the MDP certainly moves to the left/right with zero reward. In state S_2 there are several possible actions that all lead to T_2 and yield a Gaussian reward with mean μ and variance σ^2 (typically equal to 1). Started in S_1 an optimal policy clearly choses to terminate in T_1 as the expected total reward is 0, a policy that also allows to go to S_2 yields a negative expected total reward.



red states are terminating

In the classical example Q is initialised as 0 and μ is set to -0.1 . The behavior policy always choses S_1 as a starting state. Now suppose the Q-learning algorithm terminates in the first iteration in T_2 and the final reward is positive, say 1. This leads to a Q-value $Q(S_2, a) = \alpha$.

During the next iteration in which the Q -learning algorithm uses S_2 to update $Q(S_1, \text{left})$ the update-rule will overestimate $Q(S_1, \text{left})$ to some positive value. It will then take some time the discounting factor decreases $Q(S_1, \text{left})$ back to 0. The plots (running 300 episodes and averaging over 10.000 runs) show a few different situation of the simple MAP example with different μ and different initialisation for Q .



A few examples, truncation constants $C_- = 10^5, C_+ = 10^5$

In the upper plots a completely random behavior policy was used to select the actions during learning, while in the lower two plots an ϵ -greedy policy with $\epsilon = 0.1$ was used. The plots show depending on the setup/initialisation either positive or negative bias can be beneficial.

Let us now proceed with what we call the truncated double Q -learning Algorithm that contains other important double algorithms as special cases.

Algorithm 24: Truncated Double Q-learning**Data:** Behavior policy $\pi \in \Pi_S$, positive truncation $C_+ > -$, negative truncation $C_- > 0$ **Result:** Approximations $Q \approx Q^*$, $\pi = \text{greedy}(Q) \approx \pi^*$ Initialize Q^A, Q^B (e.g. 0).**while** *not converged* **do** Initialise s . **while** s *not terminal* **do** Sample $a \sim \pi(\cdot; s)$. Sample reward $R(s, a)$. Sample $s' \sim p(\cdot; s, a)$. Determine stepsize α . Randomly choose $\text{update} = A$ or $\text{update} = B$ **if** $\text{update} = A$ **then** $a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$ $\epsilon = \gamma \max(-C_- \alpha, \min(Q^B(s', a^*) - Q^A(s', a^*), C_+ \alpha))$ Update $Q^A(s, a) = Q^A(s, a) + \alpha (R(s, a) + \gamma Q^A(s', a^*) - Q^A(s, a) + \epsilon)$ Set $s = s'$. **end** **else** $b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$ $\epsilon = \gamma \max(-C_- \alpha, \min(Q^A(s', b^*) - Q^B(s', b^*), C_+ \alpha))$ Update $Q^B(s, a) = Q^B(s, a) + \alpha (R(s, a) + \gamma Q^B(s', b^*) - Q^B(s, a) + \epsilon)$ Set $s = s'$. **end** **end****end**

To understand double Q -learning and truncated double Q -learning let us proceed similarly to SARSA and rewrite the update as a Q -learning update with additional error:

$$\begin{aligned} Q_{n+1}^A(s, a) &= Q_n^A(s, a) + \alpha (R(s, a) + \gamma Q_n^B(s', a^*) - Q_n^A(s', a)) \\ &= Q_n^A(s, a) + \alpha (R(s, a) + \gamma Q_n^A(s', a^*) - Q_n^A(s, a) + \gamma (Q_n^B(s', a^*) - Q_n^A(s', a^*))) \end{aligned}$$

which can be written as

$$Q_{n+1}^A(s, a) = Q_n^A(s, a) + \alpha (T^*(Q_n^A)(s, a) + \epsilon_n(s, a) - Q_n(s, a)).$$

with errors

$$\epsilon_n(s, a) := \underbrace{[R(s, a) + \gamma Q_n^A(s', a^*) - T^*Q^A(s, a)]}_{=: \epsilon_n^Q(s, a)} + \underbrace{\gamma (Q_n^B(s', a^*) - Q_n^A(s', a^*))}_{=: \hat{\epsilon}_n(s, a)}.$$

Thus, from the point of view of approximate dynamic programming, double Q -learning is nothing but Q -learning with an additional error. Since the two equations are symmetric the error is negatively biased (the Q -function for some action should be smaller than the Q -function for the best action).



Make this intuition rigorous, show that $\mathbb{E}[\hat{\epsilon}_n | \mathcal{F}_n] < 0$ almost surely.

Furthermore, looking carefully at the algorithm, truncated double Q -learning equals double Q -learning if $C_+ = C_- = +\infty$ (or, in practice, just very large) as in that case

$$\max(-C_- \alpha, \min(Q_n^B(s', a^*) - Q_n^A(s', a^*), C_+ \alpha)) = Q_n^B(s', a^*) - Q_n^A(s', a^*).$$



Theorem 4.4.7. Consider the updating procedure of truncated double Q -learning in algorithm for Q^A and Q^B . Suppose that a behavior policy π is such that all state-action pairs are visited infinitely often and the step sizes are adapted and satisfy the Robbins-Monro conditions. Then

$$\lim_{t \rightarrow \infty} Q^A(s, a) = \lim_{t \rightarrow \infty} Q^B(s, a) = Q^*(s, a)$$

holds almost surely for all state-action pairs (s, a) .

Please note that truncated Q -learning was only introduced for these lectures notes as it allows us to prove convergence using Theorem 4.3.9. Interestingly, the algorithm performs pretty well on examples and it might be worth improving the algorithm by adaptive (depending on the past data) choice of C_+ and C_- to learn the need of over- or understimation.

Proof. Because of the symmetry of the update procedure it is sufficient to prove the convergence of Q^A . As for SARSA the point is to use the reformulation as Q -learning with additional error and show that errors are sufficiently little biased. This is why we introduced the additional truncation in order to be able to check

$$\sum_{n=0}^{\infty} \alpha_n(s, a) |\mathbb{E}[\epsilon_n(s, a) | \mathcal{F}_n]| < \infty. \quad (4.12)$$

In the following we will write $a^* = \arg \max_{a \in \mathcal{A}_{s'}} Q^A(s', a)$ and $b^* = \arg \max_{a \in \mathcal{A}_{s'}} Q^B(s', a)$ for given s' and write

$$Q_{n+1}^A(s, a) = Q_n^A(s, a) + \alpha_n(s, a) (T^*(Q_n^A)(s, a) + \epsilon_n(s, a) - Q_n^A(s, a)).$$

with error

$$\begin{aligned} \epsilon_t(s, a) &:= \left[R(s, a) + \gamma Q_n^A(s', a^*) - T^* Q_n^A(s, a) \right] \\ &\quad + \gamma \max(-C_- \alpha_n(s, a), \min(Q_n^B(s', a^*) - Q_n^A(s', a^*), C_+ \alpha_n(s, a))) \\ &=: \epsilon_n^Q + \hat{\epsilon}_n(s, a). \end{aligned}$$

All that remains to show is that the error term has bounded conditional second moments and the bias satisfies (4.12). Finite second moments follow as we assume in these lectures (for simplicity) that the rewards are bounded so that $\sum_{k=0}^{\infty} \gamma^k = \frac{\gamma}{1-\gamma}$ implies boundedness. The Q -learning error is unbiased (sample minus expectation of the sample). The truncated double Q -learning error is also bounded:

$$\sum_{n=0}^{\infty} \alpha_n(s, a) |\mathbb{E}[\hat{\epsilon}_n(s, a) | \mathcal{F}_n]| \leq \max\{C_+, C_-\} \sum_{n=0}^{\infty} \alpha_n^2(s, a) < \infty. \quad (4.13)$$

Since T^* is a contraction and the learning rates satisfy the Robbins-Monro conditions the convergence follows from Theorem 4.3.9 with the modification. \square



The interesting feature of truncated double Q -learning is the interpolation effect between Q -learning and double Q -learning. Large C makes the algorithm closer to double Q -learning, small C to Q -learning. It would be interesting to see if an adaptive choice of C (depending on the algorithm) could be used to combine the overestimation of Q -learning and the understimation of double Q -learning.

We finish this section with another variant of double Q -learning, so-called clipping¹⁰:

¹⁰Fujimoto, van Hoof, Meger: "Addressing Function Approximation Error in Actor-Critic Methods", ICML 2018

Algorithm 25: Clipped double Q-learning (with behavior policy)

Data: Behavior policy $\pi \in \Pi_S$
Result: Approximations $Q \approx Q^*$, $\pi = \text{greedy}(Q) \approx \pi^*$
Initialize Q^A, Q^B (e.g. 0).
while *not converged* **do**
 Initialise s .
 while s *not terminal* **do**
 Sample $a \sim \pi(\cdot; s)$.
 Sample reward $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 Determine stepsize α .
 $a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$
 $Q^A(s, a) = Q^A(s, a) + \alpha(R(s, a) + \gamma \min\{Q^A(s', a^*), Q^B(s', a^*)\} - Q^A(s, a))$.
 $b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$
 $Q^B(s, a) = Q^B(s, a) + \alpha(R(s, a) + \gamma \min\{Q^A(s', b^*), Q^B(s', b^*)\} - Q^B(s, a))$.
 Set $s = s'$.
 end
end

The convergence proof of clipped double Q-learning again follows the SARSA approach.



Rewrite Q^A to see that Q^A is nothing but Q-learning with additional error term

$$\varepsilon^c(s, a) = \begin{cases} 0 & : Q^A(s', a^*) \leq Q^B(s', a^*) \\ Q^B(s', a^*) - Q^A(s', a^*) & : Q^A(s', a^*) > Q^B(s', a^*) \end{cases}$$

Clipped Q-learning is thus nothing but double Q-learning with clipping (truncation) of positive bias terms $Q^B(s', a^*) - Q^A(s', a^*)$.

Setting $C_+ = 0$ clipping is nothing but truncated Q-learning with very large C_- .



In the exercises we will compare the performance of the different algorithms. Unfortunately, none of them outperforms in all settings. Adding error terms that are negatively biased helps to reduce overestimation of Q-learning but clearly has other drawbacks. To our knowledge there is no deeper theoretical understanding of how to deal optimally with overestimation.

4.5 Multi-step approximate dynamic programming

The one-step approximate dynamic programming algorithms were derived rather directly from Theorem 4.3.9. We next turn towards finitely many steps forwards which is inbetween the one-step and infinitely many steps (aka Monte Carlo) approaches.

4.5.1 n -step TD for policy evaluation and control

The idea of approximate is to solve fixed point equation in the case in which operators that are given by expectations can only be approximated by sampling. We now ignore this point of view and recall what really happens in the updates. For SARSA policy evaluation those were

$$\underbrace{V_{\text{new}}(S_t)}_{\text{new estimate}} = V_{\text{old}}(S_t) + \alpha \left(\underbrace{R(S_t, A_t) + \gamma V_{\text{old}}(S_{t+1})}_{\text{reestimate of } V(S_t)} - \underbrace{V_{\text{old}}(S_t)}_{\text{old estimate}} \right)$$

and for evaluation of Q^π

$$\underbrace{Q_{\text{new}}(S_t, A_t)}_{\text{new estimate}} = Q_{\text{old}}(S_t, A_t) + \alpha \left(\underbrace{R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})}_{\text{reestimate of } Q(S_t, A_t)} - \underbrace{Q_{\text{old}}(S_t, A_t)}_{\text{old estimate}} \right).$$

In every step the algorithms reestimate the state(-action) value function by resampling the first step and then continuing according to dynamic programming with the current estimate. The difference between new estimate and old estimate is called temporal-difference (TD) error. Weighting old and new estimates then leads to an increase for positive TD error (resp. a decrease for negative TD error). A natural generalisation for this reestimation procedure uses longer temporal differences, i.e. resample the next n steps and then continue according to the old estimate, i.e. reestimating with $\sum_{i=1}^{n-1} \gamma^i R(S_{t+i}, A_{t+i}) + \gamma^n V^\pi(S_{t+n})$. The corresponding algorithms are called n -step TD algorithms. We are not going to spell-out the details, the only difference is the update which is given below and that (compared to one-step) updates are stopped n steps before the termination as n steps in the future are used for the update.



Use the update rule

$$\underbrace{V_{\text{new}}(S_t)}_{\text{new estimate}} = V_{\text{old}}(S_t) + \alpha \left(\underbrace{\sum_{i=0}^{n-1} R(S_{t+i}, A_{t+i}) + \gamma V_{\text{old}}(S_{t+n})}_{\text{reestimate of } V(S_t)} - \underbrace{V_{\text{old}}(S_t)}_{\text{old estimate}} \right)$$

to write down pseudocode for n -step TD algorithms for evaluation of V^π and Q^π and prove the convergence by checking the n -step Bellman expectation equations ^a

$$T^\pi V(s) = \mathbb{E}_s^\pi \left[R(s, A_0) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) \right] \quad (4.14)$$

and

$$T^\pi Q(s, a) = \mathbb{E}_s^{\pi^a} \left[R(s, a) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n Q(S_n, A_n) \right]$$

and the conditions of Theorem 4.3.9 on the error term.

^awrite down?

Analogously, a new n -step SARSA-type control algorithm can be written down, we will compare the performance to 1-step SARSA in the implementation exercises.



Write pseudocode for an n -step SARSA control algorithm in the non-terminating case. Try to prove convergence in the same way we did for 1-step SARSA in Theorem 4.4.5.

To understand multistep methods better let us compare the Monte Carlo estimator of V^π from Section 4.2.1 with the one-step approximate dynamic programming estimator of V^π from Section 4.4.1. Looking closely at the n -step estimator a crucial observation can be made. For large n the TD update is essentially the Monte Carlo update. For a particular problem one can thus chose n such that the algorithm is closer to Monte Carlo (no reuse of samples) or closer to one-step approximate dynamic programming.

- The Monte Carlo estimator averages independent samples $\hat{V}_k^\pi = \sum \gamma^t R(s_t, a_t)$ of the discounted total reward to get

$$\hat{V}^\pi(s) = \frac{1}{N} \sum_{k=1}^N \hat{V}_k^\pi(s).$$

The Monte Carlo estimator uses every sample $R(s, a)$ once, whereas the dynamic programming estimator reuses samples (this is called bootstrapping) because the iteration scheme

$$V_{\text{new}}(s) = V_{\text{old}}(s) + \alpha(R(s, a) + \gamma V_{\text{old}}(s') - V_{\text{old}}(s))$$

reuses all samples $R(s, a)$ that were used to estimate $V_{\text{old}}(s')$. From the practical point of view the bootstrapping is desirable if the sampling is expensive. Additionally, the reuse of estimates reduces the variance of the SARS estimator compared to the Monte Carlo estimator.

- Being unbiased Monte Carlo has a clear advantage to the unbiased algorithms obtained from stochastic approximation schemes for which we know nothing about the bias.

Let's turn these thoughts into a formal error decomposition. Suppose an estimate V of V^π is given, this is typically V_N . How close (in L^2) is the new n -step estimate from the true vector V^π ?



Proposition 4.5.1. (TD bias-variance decomposition)

Suppose V is an old estimate of V^π , then there is a constant C (depending on the assumed bound for R) such that

$$\begin{aligned} & \mathbb{E}_s^\pi \left[\overbrace{\left(\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right)^2}^{\text{new estimation error}} \right] \\ & \leq \underbrace{\gamma^{2n} \mathbb{E}_s^\pi [(V^\pi(S_n) - V(S_n))]^2}_{\text{old estimation bias}} + \underbrace{\mathbb{V}_s^\pi \left[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) \right] + \gamma^{2n} C \mathbb{V}_s^\pi [V(S_n)]}_{\text{n-step + future prediction variance}}. \end{aligned}$$

Before going through the proof let us discuss what can be learnt from the proposition. Recall that γ is fixed, V given by prior iterations of the algorithm, and n could be chosen.

- The first summand involves γ^{2n} which decreases in n and the squared error of the current estimate at time n .
- The second summand does not depend on the current estimate V , but is the variance of the n -step rewards under the target policy. This is the Monte Carlo variance for n steps.
- The last summand again involves γ^{2n} which decreases in n and the variance of the current n -step prediction.

The proof uses the classical bias-variance decomposition from statistics combined with the fact that $T^\pi V^\pi = V^\pi$ and (4.14).

Proof.

$$\begin{aligned} & \mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right)^2 \right] \\ & = \mathbb{E}_s^\pi \left[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right]^2 \\ & \quad + \mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) - \mathbb{E}_s^\pi \left[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right] \right)^2 \right] \\ & \stackrel{(4.14)}{=} \left(-\gamma^n \mathbb{E}_s^\pi [V^\pi(S_n)] + \gamma^n \mathbb{E}_s^\pi [V(S_n)] \right)^2 \end{aligned}$$

$$\begin{aligned}
& + \mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) - \mathbb{E}_s^\pi \left[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) \right] + \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) - \underbrace{(V^\pi(S_0) - \mathbb{E}_s^\pi [V^\pi(S_0)])}_{=0} \right)^2 \right] \\
& = \gamma^{2n} \left(\mathbb{E}_s^\pi [V^\pi(S_n) - V(S_n)] \right)^2 + \mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right)^2 \right] \\
& + 2\mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right] \\
& + \mathbb{E}_s^\pi \left[\left(\gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right)^2 \right].
\end{aligned}$$

If we can estimate the third summand we are done. The following estimate is pretty rough and uses the boundedness of R :

$$\begin{aligned}
& \mathbb{E}_s^\pi \left[\left(\sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right] \\
& \leq \mathbb{E}_s^\pi \left[\left| \left(\sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right| \right] \\
& \leq \frac{2C_R \gamma^n}{1 - \gamma} \mathbb{E}_s^\pi [(V(S_n) - \mathbb{E}_s^\pi [V(S_n)])^2] = \frac{2C_R \gamma^n}{1 - \gamma} \mathbb{V}_s^\pi [V(S_n)].
\end{aligned}$$

□

Now suppose a policy evaluation algorithm is run with adaptive choice of n , i.e. in every update n is adapted to the situation. Then, in theory, n would be chosen large if the current estimate has large variance or large error, otherwise small. Thus, it seems plausible that a policy evaluation algorithm based on n -step TD updates might decrease n over time.

4.5.2 TD(λ) algorithms

¹¹ There is a nice-trick in temporal different learning (learning by resampling segments). Instead of using n steps for a fixed number n one mixes different n or, alternatively, chooses n random. Since the Markov property is compatible with memoryless random variables only, it might not be surprising that geometric distributions (the only memoryless distributions) play a role. Recall that an integer valued random variables is called geometric with parameter $\lambda \in (0, 1)$ if $\mathbb{P}(X = k) = (1 - \lambda) \frac{1}{\lambda^k}$ for $k \in \mathbb{N}_0$. One interpretation is to decide successively with probability λ to first stop at 0, 1, 2, and so on. The striking fact of TD(λ) schemes is that they interpolate between the simple one-step updates (justifying the name TD(0) for one-step approximate dynamic programming) and Monte Carlo for $\lambda = 1$. In practice there will be some $\lambda \in (0, 1)$ for which the bias-variance advantages/disadvantages of TD(λ) and Monte Carlo turns out to be most effective.

In the following we present several ways of thinking that are more or less practical. The so-called forwards approach extends n -step temporal difference updates (which use paths forwards in time) to mixtures of infinitely many updates. Little surprisingly, the forwards approach is not very practical and mainly used for theoretical considerations. Interestingly, for instance used in a first visit setup the approach can be rewritten equivalently in an update scheme that can be implemented in a backwards manner (using the past values for updates). Both update schemes are different but such that the updates over an entire roll-out are equal.

Forwards TD(λ) for policy evaluation

Let's start a bit with the idea to mix temporal differences of different lengths into one update. Interestingly, there are different methods than can be worked out from the mixed temporal

¹¹mache notation Geo(λ) or Geo($1-\lambda$) kompatibel mit Stochastik 1 Skript

difference update

$$\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left(\sum_{t=0}^{n-1} \gamma^t R(S_{t+k}, A_{t+k}) + \gamma^n V_{\text{old}}(S_{t+n}) - V_{\text{old}}(S_t) \right).$$

Here is a first simple algorithm that can be seen as a rigorous version (instead of stopping at some large time) of first visit Monte Carlo estimation of V^π for MDPs with infinite horizon:

Algorithm 26: First visit Monte Carlo for non-terminating MDPs

Data: Policy $\pi \in \Pi_S$, initial condition μ , $\lambda \in (0, 1)$

Result: Approximation $V \approx V^\pi$

Initialize V_0 (e.g. $V_0 \equiv 0$).

$n = 0$

while *not converged* **do**

 Sample $T \sim \text{Geo}(\lambda)$.

 Determine stepsizes $\alpha_{n+1}(s)$ for every $s \in S$.

 Generate trajectory (s_0, a_0, s_1, \dots) until T using policy π .

for $t = 0, 1, 2, \dots, T$ **do**

if $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$ **then**

$V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t) \left(\sum_{i=0}^{T-1} \gamma^i R(s_{t+i}, a_{t+i}) + \gamma^T V_n(s_T) - V_n(s_t) \right)$

end

end

$n = n + 1$

end

Return V_n .



Theorem 4.5.2. Suppose $\pi \in \Pi_S$, $\lambda \in (0, 1)$, and α satisfies the Robbins-Monro conditions. Then $V_n(s) \rightarrow V^\pi(s)$ almost surely for all $s \in S$.

Proof. As always the convergence follows from Theorem 4.3.9 once the algorithm is reformulated with a suitable contraction and (here: unbiased) error. Let us first introduce the contraction that we will interpret in two ways:

$$F(V)(s) := \mathbb{E}_s^\pi \left[\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left(\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) \right) \right]$$

It is easy to see that $F : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$ is a contraction:

$$\begin{aligned} \|F(V) - F(W)\|_\infty &= \max_{s \in S} \left| \mathbb{E}_s^\pi \left[\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} (\gamma^n V(S_n) - \gamma^n W(S_n)) \right] \right| \\ &\leq \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \max_{s \in S} \gamma^n |\mathbb{E}_s^\pi[(V(S_n) - W(S_n))]| \\ &\leq \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \max_{s \in S} \gamma^n |V(s) - W(s)| \\ &= \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \gamma^n \|V - W\|_\infty \\ &= \mathbb{E}[\gamma^X] \|V - W\|_\infty \end{aligned}$$

for $X \sim \text{Geo}(\lambda)$. Furthermore, the unique fixed point is V^π :

$$\begin{aligned} F(V^\pi)(s) &= \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} \mathbb{E}_s^\pi \left[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V^\pi(S_n) \right] \\ &\stackrel{(4.14)}{=} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(s) \\ &= V^\pi(s) \end{aligned}$$

To prove convergence of the two algorithms we give two interpretations on how to sample from the expectation defining $F(V)$. The two ways of sampling from the expectation gives the two first-visit algorithms. By Fubini's theorem $F(V)(s)$ is the expectation of a two-stage stochastic experiment: first sample $T \sim \text{Geo}(\lambda)$ and then $\sum_{t=0}^{T-1} (\gamma^t R(S_t, A_t) + \gamma^T V(S_T))$ for a sample of the MDP started in s independently of T . This is exactly what the algorithm does so the convergence is exactly the one from one-step (or n -step) stochastic approximation writing

$$V_{n+1}(s_0) = V_n(s_0) + \alpha_{n+1}(s_0) (F(V_n)(s_0) + \varepsilon_n(s_0) - V_n(s_0))$$

with errors $\varepsilon_n(s_0) = (\sum_{t=1}^{T-1} \gamma^t R(s_t, a_t) - \gamma^T V_n(s_T)) - F(V_n)(s)$. We only need to be careful with the filtration and define \mathcal{F}_n as the σ -algebra generated by all random variables of the first n roll-outs. Since we assumed the step-sizes are fixed for each roll-out they are \mathcal{F}_n -measurable. Furthermore, ε_n is \mathcal{F}_{n+1} -measurable (only random variables from roll-out $n+1$ are used) and $\mathbb{E}[\varepsilon_n | \mathcal{F}_n] = 0$ because the errors take the form sample minus expectation of the sample. As always the errors are bounded as we assume R to be bounded in these lecture notes. \square

Next, we come to the λ -return algorithm. The algorithm is the direct adaption of n -step updates to the infinite mixture of n -steps. For every roll-out there is only one update, no further bootstrapping occurs. Once a state is hit the entire future trajectory is used to update V_n . Algorithms of this kind are typically called offline because no updates are obtained during a roll-out (in contrast for instance to the one-step algorithms).¹² The algorithm We are not going

Algorithm 27: First visit λ -return algorithm

Data: Policy $\pi \in \Pi_S$, $\lambda \in [0, 1]$

Result: Approximation $V \approx V^\pi$

Initialize V_0 (e.g. $V_0 \equiv 0$).

Set $n = 0$.

while not converged do

Determine stepsizes $\alpha_{n+1}(s)$ for every $s \in S$.

Generate trajectory (s_0, a_0, s_1, \dots) using policy π .

for $t = 0, 1, 2, \dots$ **do**

if $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$ **then**

$V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t) (\sum_{n=t}^{\infty} (1-\lambda) \lambda^{n-1} (\sum_{k=0}^{n-1} \gamma^k R(s_{t+k}, a_{t+k}) + \gamma^n V_{\text{old}}(s_{t+n}) - V_{\text{old}}(s_t)))$

end

end

$n = n + 1$

end

Return V_n .

to prove the convergence. Instead, we prove the convergence for an equivalent algorithm, the so-called first visit TD(λ) backwards algorithm.

¹²schreiben mit termination, oder diskutieren, dass nach termination alles 0 ist



There is no way of turning a forwards algorithm into an online algorithm, an algorithm where the current trajectory gradually updates the values of V because all future values are in use.

The surprising fact is that the first visit λ -return algorithm can actually be transformed into an equivalent forwards algorithm. This is the main beautiful idea of TD(λ).

TD(λ) backwards algorithms

To turn the λ -return algorithm into a backwards algorithm (e.g. only states from the past are used for every future update) a little neat lemma is needed.



Lemma 4.5.3. Suppose $s_0, a_0, s_1, a_1, \dots$ is a state-action sequence, $\lambda \in (0, 1)$, and $V : \mathcal{S} \rightarrow \mathbb{R}$, then

$$\begin{aligned} & \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left(\sum_{t=0}^{n-1} \gamma^t R(s_{t+k}, a_{t+k}) + \gamma^n V(s_{t+n}) - V(s_t) \right) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)). \end{aligned}$$

There are two interesting points of this representation of the TD(λ) update. First, the formula is more pleasant as one infinite sum disappeared. Secondly, the new formula also works for $\lambda = 0$ and $\lambda = 1$. Plugging-in yields exactly the formulas for one-step (use $0^0 = 1$) and Monte Carlo value (use a telescopic sum argument) updates.

Proof. For the proof we use that $\sum_{n=t}^{\infty} \lambda^n = \lambda^t \sum_{n=0}^{\infty} \lambda^n = \frac{\lambda^t}{1-\lambda}$ so that

$$\begin{aligned} & (1-\lambda) \left(\sum_{n=1}^{\infty} \lambda^{n-1} \left(\sum_{t=0}^{n-1} \gamma^t R(s_t, a_t) + \gamma^n V(s_n) - V(s_0) \right) \right) \\ &= (1-\lambda) \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \sum_{n=t+1}^{\infty} \lambda^{n-1} + (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \gamma^n V(s_n) - V(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + (1-\lambda)\gamma V(s_{t+1})) - V(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

The last equation can be checked by using a telescoping sum (write out the sums):

$$\begin{aligned} \sum_{t=0}^{\infty} (\gamma\lambda)^t \gamma (1-\lambda) V(s_{t+1}) - V_{\text{old}}(s_0) &= \sum_{t=1}^{\infty} (\gamma\lambda)^{t-1} \gamma V(s_t) - \sum_{t=1}^{\infty} (\gamma\lambda)^t V(s_t) - V_{\text{old}}(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (\gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

□

What do we learn from the lemma? Instead of updating once the λ -return we can equally update sequentially because a sum $\sum_{t=0}^{\infty} a_t$ can be computed sequentially by $b_{t+1} = b_t + a_t$. Turning this into an algorithm is simple. Wait for the first visit of a state s_0 and then add in every subsequent round the corresponding summand, this gives, with some inconvenient notation, Algorithm 28.

Algorithm 28: Offline TD(λ) policy evaluation with first-visit updates

Data: Policy $\pi \in \Pi_S$, $\lambda \geq 0$
Result: Approximation $V \approx V^\pi$
 Initialize V_0 (e.g. $V_0 \equiv 0$)
while *not converged* **do**
 Initialize s , $n = 1$, $N \equiv 0$, $k \equiv 0$, $t = 0$.
 Determine step-sizes $\alpha(s)$, $s \in \mathcal{S}$, for next roll-out.
 while *s not terminal* **do**
 Sample $a \sim \pi(\cdot; s)$
 Sample reward $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 $N(s) = N(s) + 1$
 if $N(s) = 1$ **then**
 $k(s) = t$
 end
 for $\tilde{s} \in S$ **do**
 if $N(\tilde{s}) \geq 1$ **then**
 $V_{n+1}(\tilde{s}) = V_n(\tilde{s}) + \alpha(\tilde{s})(\gamma\lambda)^{t-k(\tilde{s})}(R(s, a) + \gamma V_n(s') - V_n(\tilde{s}))$
 end
 end
 $s = s'$, $t = t + 1$
 end
 $n = n + 1$
end
 Return V_n .



Theorem 4.5.4. Suppose $\pi \in \Pi_S$, $\lambda \in (0, 1)$, and α satisfies the Robbins-Monro conditions. Then $V_n(s) \rightarrow V^\pi(s)$ almost surely for all $s \in \mathcal{S}$ in Algorithms 27 and 28.

Proof. By Lemma 4.5.3 the updates of V_n per roll-out are equal for both algorithms, thus, the convergence only needs to be proved for one of them. We prove convergence for the forwards algorithm. We use the same F from the proof of Theorem 4.5.2. F is a contraction with unique fixed point V^π . As always the algorithm is rewritten into a asynchronous stochastic approximation update. Without loss of generality, we assume $k(s) = 0$ if state s has been visited. Else, we can shift the sum again. From Proposition 4.5.3 we get

$$\begin{aligned}
 V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V_n(s_{t+1}) - V_n(s_t)) \\
 &= V_n(s_0) + \alpha_n(s_0) \sum_{i=1}^{\infty} (1-\lambda)\lambda^{i-1} \sum_{t=0}^{i-1} (\gamma^t R(s_t, a_t) + \gamma^i V_n(s_i)) - V_n(s_t) \\
 &= V_n(s_0) + \alpha_n(s_0) (F(V_n)(s_0) - V_n(s_0) + \varepsilon_n(s_t)),
 \end{aligned}$$

with $F(V)$ from above and error-term

$$\varepsilon_n(s_0) := \sum_{i=1}^{\infty} (1-\lambda)\lambda^{i-1} \sum_{t=0}^{i-1} (\gamma^t R(s_t, a_t) + \gamma^i V_n(s_i)) - F(V_n)(s_0)$$

for every $s \in S$. Moreover, the error-term $\varepsilon_n(s)$ fulfills

$$\mathbb{E}_s^\pi[\varepsilon_n(s) \mid \mathcal{F}_n] = (F(V_n))(s) - (F(V_n))(s) = 0.$$

Again, the errors are bounded as we assume R to be bounded. Hence, we got all assumptions for Theorem 4.3.9 and convergence towards the fixpoint V^π . ¹³ \square

We can now derive another algorithm that is much more common in practice. Essentially, we use the same algorithm as before but instead use every-visit updates instead of first visit updates. Another nice simplification turns this into the famous TD(λ) algorithm with eligibility traces.



Lemma 4.5.5. Suppose $s_0, a_0, s_1, a_1, \dots$ is a state-action sequence, $\lambda \in (0, 1)$, and $V : \mathcal{S} \rightarrow \mathbb{R}$, then

$$\begin{aligned} & \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\ &= \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \underbrace{\sum_{t=0}^k (\gamma\lambda)^{k-t} \mathbf{1}_{S_t=s_0}}_{=: e_k(s_0)}. \end{aligned}$$

Proof. The proof follows from a Fubini flip using the indicator $\mathbf{1}_{k \geq t} = \mathbf{1}_{t \leq k}$:

$$\begin{aligned} & \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\ &= \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (\gamma\lambda)^{k-t}. \end{aligned}$$

\square

Let us go back to the first visit algorithm (28) that implements first visit updates. Replacing first visit updates

$$V_{n+1}(s_0) = V_n(s_0) + \mathbf{1}_{\{\text{first visit of } s_0 \text{ at } t\}} \alpha_{n+1}(s_0) \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k))$$

by the every visit update yields

$$\begin{aligned} V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\ &\stackrel{\text{Lemma 4.5.5}}{=} V_n(s_0) + \alpha_{n+1}(s_0) \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \underbrace{\sum_{t=0}^k (\gamma\lambda)^{k-t} \mathbf{1}_{S_t=s_0}}_{=: e_k(s_0)}. \end{aligned}$$

Implemented as an Algorithm the update immediately yields Algorithm 29, backwards TD(λ) with eligibility traces.

Before proving convergence of offline TD(λ) with eligibility traces let us quickly discuss what the algorithm does. In fact, the simple mechanism is certainly a main reason for its success. The term $e_k(s)$ is called the eligibility trace at time k in state s . It determines the effect of the previous visits in state s on the current value. If s was visited at times t_1, t_2, \dots , the reward after the current visit in state s_k needs to be considered in the value function in state s . The first visit in s still has an effect on the current visit in s_k of $(\gamma\lambda)^{k-t_1}$, the second visit has a larger effect of $(\gamma\lambda)^{k-t_2}$ and so on. The first effects will vanish in the limit $k \rightarrow \infty$. In this algorithm it

¹³both proofs are incomplete. since a state might not be visited in a roll-out it might not be sampled in a run, thus, ϵ_n is not unbiased. Solution: Restart in unvisited states as long as all states have been visited. Maths ok, run-time nightmare.

Algorithm 29: Offline TD(λ) with eligibility traces

Data: Policy $\pi \in \Pi_S$, $\lambda \in [0, 1]$
Result: Approximation $V \approx V^\pi$
Initialize V_n (e.g. $V_n \equiv 0$) for all $n \in \mathbb{N}$.
 $N = 0$
while *not converged* **do**
 Initialize $e(s) = 0$ for all $s \in S$
 Initialize s .
 Determine step-sizes $\alpha(s)$, $s \in S$, for next roll-out.
 while *s not terminal* **do**
 $a \sim \pi(\cdot; s)$.
 Sample reward $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 Set $\Delta = R(s, a) + \gamma V_N(s') - V_N(s)$.
 Set $e(s) = e(s) + 1$.
 for $\tilde{s} \in S$ **do**
 Update $V_{N+1}(\tilde{s}) = V_N(\tilde{s}) + \alpha(\tilde{s})\Delta e(\tilde{s})$.
 Set $e(\tilde{s}) = \gamma\lambda e(\tilde{s})$.
 end
 $s = s'$
 end
 $N = N + 1$
end

is important to note that we still use V_N until we are in a terminal state for the update with Δ . Thus, the algorithm also does not bootstrap information of the beginning of a trajectory to later times.



There is an online version, see Algorithm 30, of TD(λ) with eligibility traces in which V is updated during the roll-out (online) via

$$V(\tilde{s}) = V(\tilde{s}) + \alpha(\tilde{s})\Delta e(\tilde{s})$$

For $\lambda = 0$ this is nothing but TD(0) whereas for $\lambda = 1$ and suitable choice of α (which?) this is the every visit Monte Carlo estimator.

This is because we can see in equation ?? that the value function is updated only when the trajectory ends and not in between. Note that for a functioning algorithm terminating MDPs are required then. For a similar convergence proof as ??, we necessarily require terminating MDPs. We need finite expected visits in each non-terminating state such that the update in ?? stays finite:

For every $s \in S$, let $0 < m(s) := \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] < \infty$ and $\tilde{\alpha}_n(s) := \alpha_n(s)m(s)$. Then we can rewrite the every visit update scheme in the mathematical form

$$\begin{aligned} & V_{N+1}(s) \\ &= V_N(s) + \tilde{\alpha}_n(s) \left[\frac{1}{m(s)} \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) - V_N(s) \right]. \end{aligned}$$

In this way backwards TD(λ) can be interpreted as stochastic approximation algorithm with step-sizes $\tilde{\alpha}_n(s)$ that satisfy the Robbins-Monro condition if and only if $\alpha_n(s)$ do.



Theorem 4.5.6. (Convergence of TD(λ) with every-visit updates)



Let for all $s \in S$ with the first visit in $k(s)$ the update of V be

$$V_{N+1}(s) = V_N(s) + \alpha_N(s) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V_N(s_{k+1}) - V_N(s_k)).$$

with $\lambda < 1$ and the trajectory (s_0, a_0, s_1, \dots) sampled according to the policy π and $m(s) < \infty$ for every $s \in S$.

Suppose that all states are visited infinitely often in the algorithm such that the step-sizes are adapted and satisfy the Robbins-Monro conditions are satisfied:

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s) < \infty \quad \text{a.s.}$$

for every $s \in S$. Moreover, let the rewards be bounded and $0 < \gamma < 1$. Then $\lim_{n \rightarrow \infty} V_N(s) = V^\pi(s)$ almost surely, for every state $s \in S$.

Proof. The filtration \mathcal{F}_N is defined to be generated by all random variables needed to for the N st roll-out. As always we rewrite the update scheme into an asynchronous stochastic approximation scheme:

$$\begin{aligned} & V_{N+1}(s_0) \\ &= V_N(s_0) + \tilde{\alpha}_n(s_0) \left(\frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) - V_N(s_0) \right) \\ &= V_N(s_0) + \tilde{\alpha}_n(s_0) (F(V_N)(s_0) - V_N(s_0) + \varepsilon_N(s_0)) \end{aligned}$$

with

$$F(V)(s) := \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V_N(S_{t+n})) \right]$$

and

$$\begin{aligned} \varepsilon_N(s_0) &:= \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) \\ &\quad - F(V_N)(s_0) + V_N(s_0) - \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} V_N(s_0) \end{aligned}$$

Similarly to the previous proof, V^π is a fixed point of F because

$$\begin{aligned} F(V^\pi)(s) &= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})) \right] \\ &= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \mathbb{E}_s^\pi \left[\sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})) \mid (S_t, A_t) \right] \right] \\ &= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(S_t) \right] \\ &= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(s) \\ &= \frac{m(s)}{m(s)} V^\pi(s) = V^\pi(s) \end{aligned}$$

for every $s \in S$. Similarly to the previous proof we also obtain that F is a contraction:

$$\begin{aligned} \|F(V) - F(W)\|_\infty &= \max_{s \in S} \left| \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} (\gamma^n V(S_{t+n}) - \gamma^n W(S_{t+n})) \right] \right| \\ &\leq \max_{s \in S} \frac{1}{m(s)} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \gamma^n \|V - W\|_\infty \right] \\ &\leq \max_{s \in S} \frac{1}{m(s)} \mathbb{E}[\gamma^X] \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] \|V - W\|_\infty = \mathbb{E}[\gamma^X] \|V - W\|_\infty \end{aligned}$$

for $X \sim \text{Geo}(\lambda)$. The error term $\varepsilon_N(s)$ fulfills

$$\mathbb{E}_s^\pi[\varepsilon_N(s) \mid \mathcal{F}_N] = (F(V_N))(s) - (F(V_N))(s) - \frac{1}{m(s)} m(s) V_N(s) + V_N(s) = 0.$$

As the rewards are bounded, we also get

$$\mathbb{E}_s^\pi[\varepsilon_N^2(s) \mid \mathcal{F}_N] \leq C \quad \forall N \in \mathbb{N}, s \in S.$$

So, we got all assumptions for Theorem 4.3.9 and convergence towards the fixpoint V^π . \square

The algorithm can also be modified by not waiting until termination before updating. Then, V_N is directly updated after each step. This is called online updating: We do not wait until the trajectory is terminated, but use a new version of V every time it has been updated. Using this update scheme, we can not use the forward and the backward view equivalently anymore: Instead to wait for the future rewards and update afterwards (forward view), we update the effect of previous states directly with a new version of the value function. The algorithm according to this scheme is now given by:

Algorithm 30: Online backwards TD(λ) with eligibility traces

Data: Policy $\pi \in \Pi_S$, $\lambda \geq 0$
Result: Approximation $V \approx V^\pi$
Initialize V (e.g. $V \equiv 0$).
while not converged do
 Initialize $e(s) = 0$ for all $s \in S$
 Initialize s .
 Determine step-sizes $\alpha(s)$, $s \in S$, for next roll-out.
 while s not terminal do
 $a \sim \pi(\cdot; s)$
 Sample $a \sim \pi(\cdot; s)$.
 Sample $R(s, a)$.
 Sample $s' \sim p(\cdot; s, a)$.
 Set $\Delta = R(s, a) + \gamma V(s') - V(s)$.
 Set $e(s) = e(s) + 1$.
 for $\tilde{s} \in S$ do
 Update $V(\tilde{s}) = V(\tilde{s}) + \alpha(\tilde{s}) \Delta e(\tilde{s})$.
 Update $e(\tilde{s}) = \gamma \lambda e(\tilde{s})$.
 end
 Set $s = s'$.
 end
end

The convergence of the both versions of TD(λ) can be proven by proving the offline version first and then showing that the online version and the offline version will converge to the same function.



To summarise the discussion of temporal difference with eligibility traces the story of the chapter kind of reversed. Earlier we suggested different contractions F with fixed point V^π (or Q^π or Q^*) and immediately got an algorithm as approximate fixed point iteration. Here the approach got reversed. A very simple algorithm is written down and then proved to converge by rewriting into a very tricky contraction operator.

TD(λ) for Control

To be written... We will only adapt the ideas of the last section to control algorithms now. No proofs - just algorithms!

SARSA(λ)

Q(λ)

Q-learning can be played offline. Therefore, we can not estimate returns along the sampled trajectory if the selection policy is not the greedy policy. But we noticed that it might be a good idea to choose actions which are close to the greedy policy, e.g. ϵ -greedy. So, if we play the greedy action in many cases, we can derive Q(λ) as well.

Part III

Non-Tabular Reinforcement Learning

Chapter 5

Policy Gradient methods

The third part of these lecture notes takes a very different view on optimal control problems. While in the previous parts we developed a complete theory of exact and approximate control in the tabular setting the third part of these lecture notes will be less complete. The theory is more complex and much less understood. In the case of state-action spaces that are too large to deal with all Q -values $Q(s, a)$ different further approximations will be discussed. We start with powerful policy gradient method, where the optimal control problem will be attacked with numerical maximisation of the value function. While (approximate) dynamical programming are methods to find policies that are optimal for all states s (by definition of an optimal policy) we now fix a state s and try to numerically optimise $V(s)$ only. Without much thought an immediate idea is the following:



Fix a subclass $\Pi^\Theta \subseteq \Pi_S$ of parametrised stationary policies π^θ that hopefully contains the optimal policy. Maximising the mapping

$$\theta \mapsto J_\mu(\theta) = V^{\pi^\theta}(\mu) = \sum_{s \in \mathcal{S}} \mu(s) V^{\pi^\theta}(s)$$

using a numerical method based on gradient ascent is called a policy gradient method to optimise the value function started in μ . The best policy is denoted by π^{θ^*} and we hope that $\pi^{\theta^*} \approx \pi^*$. Since $\nabla J_\mu = \sum_{s \in \mathcal{S}} \mu(s) \nabla J_s$ we will state all formulas only for the case $J(\theta) = V^{\pi^\theta}(s)$.

There are many questions to be considered. (1) How to set up an algorithm, how to compute the gradients? This is answered with the policy gradient theorems that surprisingly show that gradient estimates can be obtained even in a model free way. (2) How to chose Π^Θ such that $\pi^* \in \Pi^\Theta$? This is a delicate problem, as it results in a trade-off of choosing large and small families of policies. Not much is known in practical applications, typically neural networks are involved and one hopes to get close to π^* . (3) How good is the policy obtained for different starting conditions? A priori the policy gradient approach does not give any information. Typically one will uses neural networks and try to optimise them for different states s . In order to use gradient methods it is not surprising that differentiability assumptions are needed.



Definition 5.0.1. Let $\Theta \subset \mathbb{R}^d$, then a set $\{\pi^\theta : \theta \in \Theta\}$ of stationary policies such that $\theta \mapsto \pi^\theta(a, ; s)$ is differentiable in θ for every $s \in \mathcal{S}$ and $a \in \mathcal{A}$ is called differentiable parameterised family of stationary policies.

We will later discuss policies parametrised by neural networks which must be large enough to include enough (hopefully the optimal) policies but at the same time small enough to be numerically tractable.



If $\Theta = \mathbb{R}^d$ then to reduce the dimension of the (very!) large state-action spaces it is strongly desirable to have $d < |\mathcal{A}| |\mathcal{S}|$.

There is a simple example to keep in mind but massively violates the goal to reduce the dimension of the problem. The stationary softmax policies that were already discussed for bandits in Section 1.3.4 readily extend to general Markov decision problems:

Example 5.0.2. Suppose $d = |\mathcal{S}| |\mathcal{A}|$ so that every state-action pair has an own parameter. Denote by $\theta = (\theta_{s,a})_{s \in \mathcal{S}, a \in \mathcal{A}}$ the parameters for the parametrised family and define the softmax policy

$$\pi^\theta(a; s) = \frac{e^{\theta_{s,a}}}{\sum_{a' \in \mathcal{A}} e^{\theta_{s,a'}}}.$$

Instead of the (full) tabular parameterisation one can also chose any differentiable function $\theta \mapsto h_\theta(s, a)$ and define the generalised softmax policy

$$\pi^\theta(a; s) = \frac{e^{h_\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{h_\theta(s,a')}}.$$

For instance, if $h_\theta(s, a) = \theta^T \Phi(s, a)$ then the parametrisation is called linear softmax with features $\Phi(s, a)$ and needs as many dimensions as features are fixed. If there are as many features as state-action pairs and every state-action pair only has its own feature, i.e. the vector $\Phi(s, a)$ is a unit vector with 1 at the position corresponding to (s, a) , then the linear softmax equals the tabular softmax. The feature vector is supposed to reduce complexity, state-action pairs with same feature vector are treated equally. Hence, in practice the number of features should be large enough to separate state-action pairs sufficiently well but small enough to be computationally tractable.

The rest of this chapter will consist of making (practical) sense of gradient ascent algorithms to find optimal policies. Assuming perfect information, i.e. $J(\theta) = V^{\pi^\theta}(s)$ is known explicitly, then Algorithm 31 is the kind of algorithm we aim for. Unfortunately, there are plenty of problems.

Algorithm 31: Simple policy gradient algorithm

Data: Initial parameter θ_0

Result: Approximate policy $\pi^\theta \approx \pi^{\theta^*}$

Set $n = 1$.

while *not converged* **do**

 Choose step-size α .

 Calculate $K = \nabla J(\theta)|_{\theta=\theta_{n-1}}$.

 Update $\theta_n = \theta_{n-1} - \alpha K$.

 Set $n = n + 1$.

end

Most importantly the following:

- There is no reason $\theta \mapsto J(\theta)$ satisfies typical concavity conditions (it doesn't!) to apply gradient ascent methods, why should gradient ascent not get stuck in some stationary point, i.e. $\nabla J(\theta) = 0$?
- The gradient $\nabla J(\theta)$ is typically not known. But how can gradient ascent be implemented without access to the gradient?
- The first two issues can (partially) be resolved. Still, the simple policy gradient algorithm converges extremely slow. How can the algorithm be speed up?

In the next sections we will shed some light on what is going on. First, classical results for gradient descent/ascent are recalled. Here we focus on what turns out to be the right set-up for gradient ascent in the policy gradient setup. The PL inequality and L -smoothness. To get a certain feeling on how to replace $\nabla J(\theta)$ by estimates we quickly recall some results and arguments from stochastic gradient descent. Next, we will discuss the policy gradient theorems and how to combine them with neural network parametrisations. Finally, a couple of modifications are discussed that are used in practice.

5.1 A quick dive into gradient descent methods

We will focus during this section¹ on gradient descent methods for functions $f : \mathbb{R}^d \rightarrow \mathbb{R}$, i.e. we aim to find (local) minima of f . Note that all results hold similar for gradient ascent methods by considering $-f$ instead of f , then all maxima are minima.

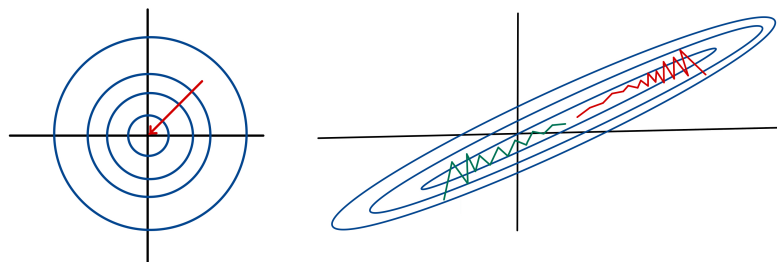
Before we get started recall from basic analysis that the negative gradient is the direction of the steepest descent, i.e.

$$\min_{\|d\|=1} \underbrace{f'(x)d}_{\text{slope in direction } d} = \nabla f(x).$$

This motivates the minimisation scheme that goes step-by-step in the direction of the gradient multiplied with the length of each step determined by a step-size:

$$x_{n+1} = x_n - \alpha \nabla f(x_n), \quad x_0 \in \mathbb{R}^d.$$

Since only derivatives of first order are involved such a numerical scheme is called a first order scheme. How to choose the step-size (length of step into gradient direction) is non-trivial. Going too far might also lead to an increase of the function, not going far enough might lead to very slow convergence. To get a first feeling have a look at the illustration of two extreme scenarios. For a quadratic function $f(x) = \|x\|^2$ gradient descent can converge in one step, for functions with narrow valleys the convergence is very slow. The problem in the second example the mixture of flat direction (in the valley) and steep direction (down the ridge) that is reflected in a large ratio of largest and smallest Eigenvalue of the Hessian $\nabla^2 f(x)$. Thus, it is not surprising that good convergence properties hold for functions not too big Eigenvalues (so-called L -smooth functions) and not too small Eigenvalues (so-called strongly convex). We will not talk about Eigenvalues but use equivalent definitions that are more straightforward to be used in the proofs.



To get a first impression in what follows we will go through the most basic theory, for L -smooth (not too steep) and strongly convex (not too flat) functions. What is more relevant for policy gradient in reinforcement learning is the continuation for functions that satisfy the so-called PL inequality (gradient dominates the function differences).⁹

Let us start with L -smoothness.

¹For much more please check the lecture notes https://www.wim.uni-mannheim.de/media/Lehrstuehle/wim/doering/OptiML/Sheets/lecture_notes_01.pdf of Simon Weißmann and, of course, the standard text book "Nonlinear programming" of Dimitri P. Bertsekas from which most proofs are taken.



Definition 5.1.1. We call a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ L -smooth if f is differentiable and the gradient ∇f is L -Lipschitz continuous, i.e.

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^d. \quad (5.1)$$

L -smoothness is a property in which gradient descent algorithms have a good chance to converge and, luckily, gradients of value functions of MDPs are L -smooth for some choices of policies. To get an idea of what L -smoothness means note that for twice-differentiable f the L -smoothness is equivalent to all eigenvalues of the Hessian $\nabla^2 f(x)$ to be in $[-L, L]$. Thus, L -smooth function do not have very strong curvature.

5.1.1 Gradient descent for L -smooth functions

In this first section we consider gradient descent with the only assumption of f being L -smooth. In that case gradient descent does not necessarily converge to a (local) minimum but we can show it converges (if it converges) to a stationary point, i.e. a point with vanishing gradient. Stationary points are not necessarily (local) extreme points, saddle points are also stationary.



Lemma 5.1.2. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth for some $L > 0$. Then it holds that

$$f(x + y) \leq \underbrace{f(x) + y^T \nabla f(x)}_{\text{tangent at } x \text{ plus quadratic}} + \frac{L}{2} \|y\|^2 \quad (5.2)$$

for all $x, z \in \mathbb{R}^d$.

Note that small L makes the deviation of ∇f smaller, thus, the function smoother. It will later turn out that convergence of gradient descent is faster for smoother functions.

Proof. We define $\phi(t) = f(x + ty)$ and apply the chain rule in order to derive

$$\phi'(t) = y^T \nabla f(x + ty), \quad t \in [0, 1].$$

By the fundamental theorem of calculus it follows

$$\begin{aligned} f(x + y) - f(x) &= \phi(1) - \phi(0) = \int_0^1 \phi'(t) dt \\ &= \int_0^1 y^T \nabla f(x + ty) dt \\ &= \int_0^1 y^T \nabla f(x) dt + \int_0^1 y^T (\nabla f(x + ty) - \nabla f(x)) dt \\ &\leq y^T \nabla f(x) + \int_0^1 \|y\| \|\nabla f(x + ty) - \nabla f(x)\| dt \\ &\leq y^T \nabla f(x) + \|y\| \int_0^1 Lt \cdot \|y\| dt \\ &\leq y^T \nabla f(x) + \frac{L}{2} \|y\|^2, \end{aligned}$$

where we have applied Cauchy-Schwarz followed by L -smoothness. \square

The aim is to show that the gradient descent algorithm converges to a (local) minimum. This is not always the case, the algorithm can for instance also converge to saddle points. What is true without any assumptions, except the minimal differentiability, is the convergence to a stationary point, i.e. a point with vanishing gradient.



Theorem 5.1.3. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth and $(x_k)_{k \in \mathbb{N}}$ be defined as

$$x_{k+1} = x_k - \alpha \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with non-negative step-size $\alpha \in [\varepsilon, \frac{2-\varepsilon}{L}]$ for some $\varepsilon < \frac{2}{L+1}$. Then every accumulation point \bar{x} of $(x_k)_{k \in \mathbb{N}}$ is a stationary point of f , i.e. $\nabla f(\bar{x}) = 0$.

Proof. Sara irgendwann, war nicht in der Vorlesung. □

The theorem also holds for diminishing step-sizes α_k . Only if those decay too strongly (being summable) the algorithm might produce a sequence that stops moving away from a stationary point.



Proposition 5.1.4. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth and $(x_k)_{k \in \mathbb{N}}$ be defined as

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with non-negative step-size sequence $(\alpha_k)_{k \in \mathbb{N}}$ that fulfills

$$\lim_{k \rightarrow \infty} \alpha_k = 0 \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k = \infty.$$

Then for the sequence $(f(x_k))_{k \in \mathbb{N}}$ it holds true that either

$$\lim_{k \rightarrow \infty} f(x_k) = -\infty \quad \text{or} \quad \lim_{k \rightarrow \infty} \nabla f(x_k) = 0.$$

Moreover, every accumulation point \bar{x} of $(x_k)_{k \in \mathbb{N}}$ is a stationary point of f , i.e. $\nabla f(\bar{x}) = 0$.

Note that the step-sizes can also be constant but the convergence also holds for decreasing step-sizes that do not decrease too fast.

Proof. First we apply the descent lemma to derive

$$\begin{aligned} f(x_{k+1}) &= f(x_k - \alpha_k \nabla f(x_k)) \\ &\leq f(x_k) - \alpha_k \nabla f(x_k)^T \nabla f(x_k) + \frac{L}{2} \|\alpha_k \nabla f(x_k)\|^2 \\ &= f(x_k) - \alpha_k \left(1 - \frac{L\alpha_k}{2}\right) \|\nabla f(x_k)\|^2. \end{aligned}$$

Since we have assumed that $\lim_{k \rightarrow \infty} \alpha_k = 0$, there exists a $k_0 \in \mathbb{N}$ such that

$$f(x_{k+1}) \leq f(x_k) - \frac{\alpha_k}{2} \|\nabla f(x_k)\|^2$$

for all $k \geq k_0$. Hence, the sequence $(f(x_k))_{k \geq k_0}$ is decreasing and it either holds that $\lim_{k \rightarrow \infty} f(x_k) = -\infty$ or $\lim_{k \rightarrow \infty} f(x_k) = M$ for some $M < \infty$. Suppose that we are in the case where $\lim_{k \rightarrow \infty} f(x_k) = M$. It follows

$$\sum_{k=k_0}^K \frac{\alpha_k}{2} \|\nabla f(x_k)\|^2 \leq \sum_{k=k_0}^K [f(x_k) - f(x_{k+1})] = f(x_{k_0}) - f(x_K)$$

for ever $K > k_0$, using that $\sum_{k=k_0}^K [f(x_k) - f(x_{k+1})]$ is a telescoping sum. For $K \rightarrow \infty$ we even imply

$$\sum_{k=k_0}^{\infty} \frac{\alpha_k}{2} \|\nabla f(x_k)\|^2 \leq f(x_{k_0}) - M < \infty. \quad (5.3)$$

Since $\sum_{k=0}^{\infty} \alpha_k = \infty$, there can not exist any $\epsilon > 0$ such that $\|\nabla f(x_k)\|^2 > \epsilon$ for all $k \geq \hat{k} \geq 0$. However, this only means that

$$\liminf_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0.$$

In order to prove $\lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$ we will use (5.3) to prove that $\limsup_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0$. Suppose that $\limsup_{k \rightarrow \infty} \|\nabla f(x_k)\| > \epsilon$ for some $\epsilon > 0$ and consider two sequences $(m_j)_{j \in \mathbb{N}}$, $(n_j)_{j \in \mathbb{N}}$ with $m_j < n_j < m_{j+1}$ such that

$$\frac{\epsilon}{3} < \|\nabla f(x_k)\|, \quad \text{for } m_j \leq k < n_j$$

and

$$\|\nabla f(x_k)\| \leq \frac{\epsilon}{3}, \quad \text{for } n_j \leq k < m_{j+1}.$$

Moreover let $\bar{j} \in \mathbb{N}$ be sufficiently large such that

$$\sum_{k=m_j}^{\infty} \alpha_k \|\nabla f(x_k)\|^2 \leq 2 \frac{\epsilon^2}{9L}.$$

Using L -smoothness for $j \geq \bar{j}$ and $m_j \leq m \leq n_j - 1$ it holds that

$$\begin{aligned} \|\nabla f(x_{n_j}) - \nabla f(x_m)\| &\leq \sum_{k=m}^{n_j-1} \|\nabla f(x_{k+1}) - \nabla f(x_k)\| \\ &\leq L \sum_{k=m}^{n_j-1} \|x_{k+1} - x_k\| \\ &= \frac{3\epsilon}{3\epsilon} L \sum_{k=m}^{n_j-1} \alpha_k \|\nabla f(x_k)\| \\ &\leq \frac{3L}{\epsilon} \sum_{k=m}^{n_j-1} \alpha_k \|\nabla f(x_k)\|^2 \\ &\leq \frac{6}{\epsilon} \frac{\epsilon^2}{9L} = \frac{2\epsilon}{3}, \end{aligned}$$

where we have used that $\frac{\epsilon}{3} < \|\nabla f(x_k)\|$ for $m_j \leq k \leq n_j - 1$. This implies that

$$\|\nabla f(x_m)\| \leq \|\nabla f(x_{n_j})\| + \|\nabla f(x_{n_j}) - \nabla f(x_m)\| \leq \|\nabla f(x_{n_j})\| + \frac{\epsilon}{3} \leq \epsilon$$

for all $m \geq m_{\bar{j}}$. This is in contradiction to $\limsup_{k \rightarrow \infty} \|\nabla f(x_k)\| \geq \epsilon$ and we have proved that

$$\limsup_{k \rightarrow \infty} \|\nabla f(x_k)\| = \liminf_{k \rightarrow \infty} \|\nabla f(x_k)\| = \lim_{k \rightarrow \infty} \|\nabla f(x_k)\| = 0.$$

Finally, let $\bar{x} \in \mathbb{R}^d$ be an accumulation point of $(x_k)_{k \in \mathbb{N}}$. Since $(f(x_k))_{k \geq k_0}$ is decreasing, it follows by continuity that

$$\lim_{k \rightarrow \infty} f(x_k) = f(\bar{x}) < \infty$$

then also

$$\nabla f(\bar{x}) = \lim_{k \rightarrow \infty} \nabla f(x_k) = 0.$$

□

5.1.2 Gradient descent for L -smooth, convex functions

We will now study the convergence of gradient descent under the additional assumption of convexity. For convex and L -smooth f convergence of the gradient descent recursion to a global minimum can be proved. Since for general convex functions there is no guarantee for the existence of a unique global minimum, we do only expect convergence to some global minimum $x_* \in \mathbb{R}^d$. In order to talk about convergence rates one typically considers an error function $e : \mathbb{R}^d \rightarrow \mathbb{R}$ with the property $e(x) \geq 0$ for all $x \in \mathbb{R}^d$ and $e(x_*) = 0$ for some $x_* \in \mathbb{R}^d$, e.g. $x_* \in \arg \min_{x \in \mathbb{R}^d} f(x)$ assuming it exists. Given an error function we define the following convergence rate



Definition 5.1.5. We say that the sequence of errors $(e(x_k))_{k \in \mathbb{N}}$ converges linearly, if there exists $c \in (0, 1)$ such that

$$e(x_{k+1}) \leq ce(x_k)$$

for all $k \in \mathbb{N}$.

As gradient descent is a first order method, we cannot expect faster convergence than linear. Convergence rates which are slower than linear convergence are called sublinear. For convex and smooth function we can prove the following sublinear convergence rate.



Theorem 5.1.6. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and L -smooth, and let $(x_k)_{k \in \mathbb{N}}$ be generated by

$$x_{k+1} = x_k - \bar{\alpha} \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with $\bar{\alpha} \leq \frac{1}{L}$. Moreover, we assume that the set of all global minimums of f is non-empty. Then the sequence $(x_k)_{k \in \mathbb{N}}$ converges in the sense that

$$e(x_k) := f(x_k) - f_* \leq \frac{c}{k+1}, \quad k \in \mathbb{N}$$

for some constant $c > 0$ and $f_* = \min_{x \in \mathbb{R}^d} f(x)$.

Note again that since there are no unique minima we do not aim to prove convergence of x_n to x_* but instead convergence of $f(x_n)$ to $f(x_*)$.

Proof. We again apply the descent lemma 5.1.2 to the recursive scheme ($y = x_{k+1} - x_k$, $x = x_k$) to obtain

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \bar{\alpha} \|\nabla f(x_k)\|^2 + \frac{L\bar{\alpha}^2}{2} \|\nabla f(x_k)\|^2 \\ &= f(x_k) + \bar{\alpha} \left(\frac{L\bar{\alpha}}{2} - 1 \right) \|\nabla f(x_k)\|^2. \end{aligned}$$

Since $\bar{\alpha} \leq \frac{1}{L}$, we have $(\frac{L\bar{\alpha}}{2} - 1) < 0$ and therefore, the sequence $(f(x_k))_{k \in \mathbb{N}}$ is decreasing. Now, let $x_* \in \mathbb{R}^d$ be a global minimum of f such that due to convexity it holds true that

$$\underbrace{f(x_k) + (x_* - x_k)^T \nabla f(x_k)}_{\text{tangent at } x_*} \leq f(x_*).$$

In the one-dimensional case this is nothing but saying that the tangents at the minimum lie

above the graph. We plug this into the above inequality to imply

$$\begin{aligned}
f(x_{k+1}) &\leq f(x_k) + \bar{\alpha} \left(\frac{L\bar{\alpha}}{2} - 1 \right) \|\nabla f(x_k)\|^2 \\
&\leq f(x_*) - \frac{\bar{\alpha}}{\alpha} (x_* - x_k)^T \nabla f(x_k) + \bar{\alpha} \left(\frac{L\bar{\alpha}}{2} - 1 \right) \|\nabla f(x_k)\|^2 \\
&= f(x_*) + \frac{1}{\bar{\alpha}} \left[\frac{1}{2} \|x_* - x_k\|^2 + \frac{\bar{\alpha}^2}{2} \|\nabla f(x_k)\|^2 - \frac{1}{2} \|(x_* - x_k) + \bar{\alpha} \nabla f(x_k)\|^2 \right] \\
&\quad + \bar{\alpha} \left(\frac{L\bar{\alpha}}{2} - 1 \right) \|\nabla f(x_k)\|^2,
\end{aligned}$$

where we have used $-a^T b = \frac{1}{2} \|a\|^2 + \frac{1}{2} \|b\|^2 - \frac{1}{2} \|a + b\|^2$ for $a, b \in \mathbb{R}^d$. Rearranging the righthand side yields

$$\begin{aligned}
f(x_{k+1}) &\leq f(x_*) + \frac{1}{2\bar{\alpha}} (\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2) + \bar{\alpha} \left(\frac{L\bar{\alpha}}{2} - \frac{1}{2} \right) \|\nabla f(x_k)\|^2 \\
&\leq f(x_*) + \frac{1}{2\bar{\alpha}} (\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2),
\end{aligned}$$

where we have used again that $\bar{\alpha} \leq \frac{1}{L}$. Taking the sum over all iterations gives

$$\begin{aligned}
\sum_{k=0}^N (f(x_{k+1}) - f(x_*)) &\leq \frac{1}{2\bar{\alpha}} \sum_{k=0}^N (\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2) \\
&\leq \frac{1}{2\bar{\alpha}} (\|x_* - x_0\|^2 - \|x_* - x_{N+1}\|^2) \\
&\leq \frac{1}{2\bar{\alpha}} \|x_* - x_0\|^2,
\end{aligned}$$

where we have applied a telescoping sum. With the decrease of $(f(x_k))_{k \in \mathbb{N}}$ it holds true that

$$\sum_{k=0}^N f(x_{k+1}) \geq (N+1) f(x_{N+1}),$$

and therefore, the assertion follows with

$$f(x_{N+1}) - f(x_*) \leq \frac{1}{N+1} \sum_{k=0}^N (f(x_{k+1}) - f(x_*)) \leq \frac{1}{N+1} \frac{1}{2\bar{\alpha}} \|x_* - x_0\|^2 =: \frac{c}{N+1}.$$

□



Definition 5.1.7. Let $C \subset \mathbb{R}^d$ be a convex set. A function $f : C \rightarrow \mathbb{R}$ is called μ -strongly convex over C , if

$$f(y) \geq \underbrace{f(x) + (y-x)^T \nabla f(x)}_{\text{tangent at } x \text{ plus quadratic}} + \frac{\mu}{2} \|y-x\|^2$$

for all $x, y \in C$.

If we tighten the assumption on convexity and assume μ -strong convexity instead, we can improve the convergence rate from sublinear to linear convergence as follows.



Theorem 5.1.8. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be μ -strongly convex for some $\mu > 0$ and



L -smooth, as let $(x_k)_{k \in \mathbb{N}}$ be generated by

$$x_{k+1} = x_k - \bar{\alpha} \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with $\bar{\alpha} < \frac{1}{L}$. the the sequence $(x_k)_{k \in \mathbb{N}}$ converges linearly in the sense that there exists $c \in (0, 1)$ such that

$$e(x_k) := \|x_k - x_*\| \leq c^k \|x_0 - x_*\|, \quad k \in \mathbb{N}$$

where $x_* \in \mathbb{R}^d$ is the unique global minimum of f^a with $f(x_*) = \min_{x \in \mathbb{R}^d} f(x)$. The constant can be chosen as $c = (1 + \frac{\mu}{L})^{-1/2}$.

^ain contrast to convex functions strongly convex functions have a unique minimum

An example for such a function is $f(x) = x^2 - \cos(x)$. Note that for strongly convex L -smooth functions the convergence also implies linear convergence of $f(x_n)$ to $f(x_*)$ because (Nesterov, page 64)

$$f(x_n) - f(x_*) \leq \frac{1}{2\mu} \|\nabla f(x_n) - \nabla f(x_*)\|^2 \leq \frac{L}{2\mu} \|x_n - x_*\|^2 \leq (c^2)^n \frac{L}{2\mu} \|x_0 - x_*\|^2.$$

so that the theorem significantly strengthens the previous theorem on only convex functions.²

Proof. Let $x_* \in \mathbb{R}^d$ be the unique global minimum of f with $\nabla f(x_*) = 0$. Since f is assumed to be μ -strongly convex, by Definition 5.1.7 it holds true that

$$\frac{\mu}{2} \|x_{k+1} - x_*\|^2 = \nabla f(x_*)^T (x_{k+1} - x_*) + \frac{\mu}{2} \|x_{k+1} - x_*\|^2 \leq f(x_{k+1}) - f(x_*).$$

Because μ -strongly convex implies convexity (exercise) we can use the calculations from the previous proof of Theorem 5.1.6, where we have derived that

$$f(x_{k+1}) - f(x_*) \leq \frac{1}{2\bar{\alpha}} (\|x_k - x_*\|^2 - \|x_{k+1} - x_*\|^2)$$

and together we obtain

$$\left(\frac{\mu}{2} + \frac{1}{2\bar{\alpha}}\right) \|x_{k+1} - x_*\|^2 \leq \frac{1}{2\bar{\alpha}} \|x_k - x_*\|^2.$$

The assertion follows via induction using the inequality with $c = \sqrt{\frac{1}{1 + \frac{\mu}{L}}} \in (0, 1)$. \square

For the choice $\bar{\alpha} = \frac{1}{L}$ we see from the proof that the upperbound of gradient descent is given by

$$\|x_k - x_*\| \leq \left(\sqrt{\frac{1}{1 + \frac{\mu}{L}}}\right)^k \|x_0 - x_*\|.$$

This means that the speed of convergence (small c is better) is determined by the ratio of smoothness L and strong convexity μ :

$$c = \sqrt{\frac{1}{1 + \frac{\mu}{L}}}$$

Thus, small L (very smooth function) and large μ (very convex function) yield fast convergence.

²aufschreiben

5.1.3 Gradient descent for L -smooth functions with PL inequality

In the applications for policy gradient methods the assumption of convexity (actually concavity as functions are maximised) is not fulfilled already for simple examples. There is one way to relax this assumption to so called Polyak-Łojasiewicz (PL) conditions and this, in fact, holds for certain situations in reinforcement learning³.



Theorem 5.1.9. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth and satisfied the PL condition holds for some $r \in (0, L)$

$$\|\nabla f(x)\|^2 \geq 2r(f(x) - f_*), \quad x \in \mathbb{R}^d, \quad (5.4)$$

with $f_* = \min_{x \in \mathbb{R}^d} f(x) > -\infty$. Then the sequence $(x_k)_{k \in \mathbb{N}}$ generated by

$$x_{k+1} = x_k + \bar{\alpha} \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with $\bar{\alpha} = \frac{1}{L}$ converges linearly in the sense that there exists $c = 1 - \frac{r}{L} \in (0, 1)$ such that

$$e(x_k) := f(x_k) - f_* \leq c^k (f(x_0) - f_*).$$

Note again that since there are no unique minima we do not aim to prove convergence of x_n to x_* but instead convergence of $f(x_n)$ to $f(x_*)$.

Proof. As usually we first apply the descent Lemma 5.1.2 to get

$$f(x_{k+1}) \leq f(x_k) + \nabla f(x_k)^T (x_{k+1} - x_k) + \frac{L}{2} \|x_{k+1} - x_k\|^2.$$

With $x_{k+1} = x_k - \bar{\alpha} \nabla f(x_k)$ we obtain

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \bar{\alpha} \|\nabla f(x_k)\|^2 + \frac{L\bar{\alpha}^2}{2} \|\nabla f(x_k)\|^2 \\ &= f(x_k) - \bar{\alpha} \left(1 - \frac{L\bar{\alpha}}{2}\right) \|\nabla f(x_k)\|^2 \\ &= f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2. \end{aligned}$$

Using the PL-inequality yields

$$f(x_{k+1}) \stackrel{(5.4)}{\leq} f(x_k) - \frac{r}{L} (f(x_k) - f_*).$$

Subtracting f_* from both sides, we get

$$f(x_{k+1}) - f_* \leq \left(1 - \frac{r}{L}\right) (f(x_k) - f_*)$$

and the claim follows by induction. \square

The idea behind Inequality (5.4), also known as PL or Polyak-Łojasiewicz inequality, is very simple. The inequality ensures that the gradient does not vanish as long as the gradient descent algorithm has not reached x^* . Since the algorithm is based on the gradient the algorithm does not stop moving as long $f(x_k)$ has not reached f_* . Here is another view. For every $x \in \mathbb{R}^d$ the PL inequality lower bounds the norm of the gradient by the difference of $f(x)$ to some global minimum of f . This implies, that a small gradient at x implies a small difference between the function value $f(x)$ and the global optimum. Given that gradient descent for L -smooth functions implies convergence to a stationary point by Theorem 5.1.4 it is not surprising that the PL condition is exactly what is needed for convergence. Estimates of the PL type are called gradient domination inequalities.

³Mei, Xiao, Szepesvári, Schuurmans: "On the Global Convergence Rates of Softmax Policy Gradient Methods", ICML 2020



Prove that μ -strong convexity and L -smoothness imply the PL condition (5.4).

Condition (5.4) can be relaxed a little bit to remain convergence of gradient descent but just with a sublinear convergence rate.



Theorem 5.1.10. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth and satisfied the 'weak' PL condition

$$\|\nabla f(x)\| \geq 2r(f(x) - f_*) \quad (5.5)$$

for some $r \in (0, L)$ and all $x \in \mathbb{R}^d$ with $f_* = \min_{x \in \mathbb{R}^d} f(x) > -\infty$. Then the sequence $(x_k)_{k \in \mathbb{N}}$ generated by

$$x_{k+1} = x_k - \bar{\alpha} \nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with $\bar{\alpha} = \frac{1}{L}$ converges linearly in the sense that there exists $c = 1 - \frac{r}{L} \in (0, 1)$ such that

$$e(x_k) := f(x_k) - f_* \leq \frac{L}{2r^2(k+1)}.$$

Proof. As in the previous proof we first start with the estimate

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2.$$

that uses the L -smoothness. Applying the weak PL-inequality gives

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L} 4r^2 (f(x_k) - f_*)^2.$$

Subtracting f_* on both sides of the inequality yields the recursion

$$e(x_{k+1}) \leq e(x_k) - \frac{2r^2}{L} e(x_k)^2. \quad (5.6)$$

We will show, that for any positive sequence $(a_n)_{n \in \mathbb{N}}$ with $a_n \in [0, \frac{1}{q}]$ for some $q > 0$, which satisfies the diminishing contraction

$$0 \leq a_{n+1} \leq (1 - qa_n)a_n \quad \forall n \geq 0,$$

converges to zero with convergence rate

$$a_n \leq \frac{1}{nq + \frac{1}{a_0}} \leq \frac{1}{(n+1)q}.$$

To see this, we divide the reordered contraction

$$a_n \geq a_{n+1} + qa_n^2$$

by $a_n a_{n+1}$ to obtain

$$\frac{1}{a_{n+1}} \geq \frac{1}{a_n} + q \underbrace{\frac{a_n}{a_{n+1}}}_{\geq 1} \geq \frac{1}{a_n} + q$$

which leads to

$$\frac{1}{a_n} - \frac{1}{a_0} = \sum_{k=0}^{n-1} \left(\frac{1}{a_{k+1}} - \frac{1}{a_k} \right) \geq nq.$$

Reordering we get our claim

$$a_n \leq \frac{1}{nq + \frac{1}{a_0}} \stackrel{a_0 \leq \frac{1}{q}}{\leq} \frac{1}{(n+1)q}.$$

If we can show that $e(x_k) \leq \frac{L}{2r^2}$ for all k , then the claim follows by applying this to (5.6). By L -smoothness we obtain from the descent lemma and the recursion with $\bar{\alpha} = \frac{1}{L}$

$$\begin{aligned} f_* \leq f(x_{k+1}) &\leq f(x_k) + \nabla f(x_k)^T(x_{k+1} - x_k) + \frac{L}{2}\|x_{k+1} - x_k\|^2 \\ &= f(x_k) - \frac{1}{L}\|\nabla f(x_k)\|^2 + \frac{1}{2L}\|\nabla f(x_k)\|^2 \\ &= f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2, \end{aligned}$$

which implies

$$f_* - f(x_k) \leq -\frac{1}{2L}\|\nabla f(x_k)\|^2.$$

Therefore by the weak PL-condition

$$e(x_k) \geq \frac{1}{2L}\|\nabla f(x_k)\|^2 \geq \frac{2r^2}{L}e(x_k)^2.$$

Dividing both sides by $e(x)$ and rearranging the terms results in $e(x_k) \leq \frac{L}{2r^2}$. \square

One can ask the justified question whether there exists functions which are L -smooth and not convex but satisfy the PL condition. Indeed there are some:



Show that $f(x) = x^2 + 3\sin^2(x)$ satisfies the PL condition (5.4) and prove that f is not convex. Plot the function to see why gradient descent converges. Hint: The plot can also help to find the parameter r of the PL condition.

5.1.4 Stochastic gradient descent methods

Aus Simons Skript, nur das erste Theorem ueber Konvergenz zum stationaeren Punkt.

5.1.5 Regularisation

dieses Jahr noch nicht. ⁴

5.2 Policy gradient theorems

As the name *policy gradient* suggests, the optimisation method we want to use is gradient ascent in order to find a maximum of the objective function J . This requires that J_s is differentiable in θ and a practical formula for the gradient to apply Algorithm 31. To start the discussion we first prove formulas for $\nabla J(\theta)$ that go back to Sutton, McAlister, Singh, and Mansour⁵. Their observation was that the gradient $\nabla J(\theta)$ can be expressed rather nicely and, most importantly, can be approximated in a model-free way as the appearing expectations do not involve the transition probabilities and can be approximated by sampling only.

For didactic reasons we first deal with finite-time MDPs with undiscounted rewards and additionally consider stationary policies only. The situation is not extremely realistic as most finite-time MDPs do not have optimal strategies that are stationary (think of the ice-vendor

⁴discuss regularisation in non-linear optimisation! take previous example as an example

⁵Sutton, McAlister, Singh, Mansour: "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NIPS, 1999

where closer towards the final selling day the ice-vendor will store less ice-cream). Nonetheless, there are examples with optimal stationary policies (think of Tic-Tac-Toe) and the computations are simpler to understand the policy gradient theorems. Once the structure of the gradients is understood we proceed with the discounted infinite-time case.

5.2.1 Finite-time undiscounted MDPs

In this didactic section we will derive three different expressions of the gradient. The first theorem will show that differentiability of $\theta \mapsto J(\theta)$ follows from the differentiability of the parameterised policy and gives a first formula for the gradient as an expectation. In everything that follows let us write

$$R_t^T = \sum_{k=t}^{T-1} R_{k+1}.$$

for the rewards after time t (this is also called a reward to go) and R_0^T for the total (non-discounted) reward.



Theorem 5.2.1. Assume that (S, A, R) is a T -step MDP with finite state-action spaces and consider a stationary differentiable parameterized family of policies $\{\pi^\theta : \theta \in \Theta\}$. Then the gradient of the value function with respect to θ exists and is given by

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(A_t; S_t)) R_0^T \right].$$

The statement involves the expectation of a vector which is always defined to be the vector of expectations of all coordinates of the random vector involved. Please check carefully how this appears in the proof!

Proof. Consider a trajectory $\tau = (s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T)$ of the T -step MDP and recall the MDP path probability

$$\mathbb{P}_s^{\pi^\theta}((S, A, R) = \tau) = \delta_s(s_0) \delta_0(r_0) \prod_{t=0}^{T-1} \pi^\theta(a_t; s_t) p(\{s_{t+1}, r_{t+1}\}; s_t, a_t)$$

from (3.3). Define

$$\mathcal{T} = \{\tau = (r_0, s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T) : r_t \in \mathcal{R}, s_t \in \mathcal{S}, \forall t \leq T, a_t \in \mathcal{A}, \forall t < T\}$$

as the set of all trajectories. From the definition of the value function for a T -step MDP we deduce that

$$J_s(\theta) = V^{\pi^\theta}(s) = \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} R_{t+1} \right] = \sum_{\tau \in \mathcal{T}} \mathbb{P}_s^{\pi^\theta}((S, A, R) = \tau) \sum_{t=0}^{T-1} r_{t+1}.$$

The probabilities are clearly differentiable, thus, for finite state-action states $J(\theta)$ is a finite sum of differentiable functions. This proves the differentiability. Next we use the log-trick we have

already seen for policy gradient in bandits. We have

$$\begin{aligned}
& \nabla_{\theta} \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau) \\
&= \nabla_{\theta} (\mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau)) \frac{\mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau)}{\mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau)} \\
&= \nabla_{\theta} (\log(\mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau))) \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau) \\
&= \nabla_{\theta} \left(\log(\delta_s(s_0)) + \log(\delta_0(r_0)) + \sum_{t=0}^{T-1} (\log(\pi^{\theta}(a_t; s_t)) + \log(p(\{s_{t+1}, r_{t+1}\}; s_t, a_t))) \right) \\
&\quad \times \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau) \\
&= \sum_{t=0}^{T-1} \nabla_{\theta} (\log(\pi^{\theta}(a_t; s_t))) \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau).
\end{aligned}$$

Due to the log-trick the product of probabilities factors into a sum, where just the summands π^{θ} depend on θ and so all other summands have derivative 0. Finally we can use the finiteness of the state, action and reward space to see that we can interchange the derivative and the sum, and it follows that $V^{\pi^{\theta}}$ is differentiable. We conclude with

$$\begin{aligned}
\nabla_{\theta} J_s(\theta) &= \nabla_{\theta} V^{\pi^{\theta}}(s) \\
&= \sum_{\tau \in \mathcal{T}} \nabla_{\theta} \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau) \sum_{t=0}^{T-1} r_{t+1} \\
&= \sum_{\tau \in \mathcal{T}} \mathbb{P}_s^{\pi^{\theta}}((S, A, R) = \tau) \sum_{t=0}^{T-1} \nabla_{\theta} (\log(\pi^{\theta}(a_t; s_t))) \sum_{t=0}^{T-1} r_{t+1} \\
&= \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} (\log(\pi^{\theta}(A_t; S_t))) \sum_{t=0}^{T-1} R_{t+1} \right] \\
&= \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} (\log(\pi^{\theta}(A_t; S_t))) R_0^T \right].
\end{aligned}$$

□

The formula for $\nabla J_s(\theta)$ crucially involves $\nabla(\log(\pi^{\theta}(a; s)))$ that in Section 1.3.4 was already called the score-function of the policy.



Definition 5.2.2. If π is a policy, then the vector $\nabla_{\theta}(\log(\pi^{\theta}(a; s)))$ is called the score-function of π .

We have already computed the score-function of the (one-state) softmax policy for stochastic bandits. To get a feeling please redo the computation:



Show for the tabular softmax parametrisation from Example 5.0.2 that

$$\frac{\partial \log(\pi^{\theta}(a; s))}{\partial \theta_{s', a'}} = \mathbf{1}_{\{s=s'\}} (\mathbf{1}_{\{a=a'\}} - \pi^{\theta}(a'; s'))$$

and for the linear softmax with features $\Phi(s, a)$

$$\nabla \log(\pi^{\theta}(a; s)) = \Phi(s, a) - \sum_{a'} \pi^{\theta}(a'; s) \Phi(s, a').$$

Given the previous theorem, we weight the gradient of the log-probability of the chosen action with the reward of the whole trajectory R_0^T . So the gradient with respect to the t -th summand

(and thus the t -th action) is weighted with R_0^T , but due to the Markov property the action in time t is completely independent from the past time points $0, \dots, t-1$. In the next theorem we will see that we can indeed replace R_0^T by the rewards of the future time points $t' \geq t$. We replace the reward of the whole trajectory by the reward to go R_t^T .



Theorem 5.2.3. Assume that (S, A, R) is a T -step MDP with finite state-action spaces and consider a stationary differentiable parameterized family of policies $\{\pi^\theta : \theta \in \Theta\}$. Then the gradient of the value function can also be written as

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(A_t; S_t)) R_t^T \right].$$

Proof. From Theorem 5.2.1 we have

$$\begin{aligned} \nabla_\theta J_s(\theta) &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_t; S_t))) \sum_{t=0}^{T-1} R_{t+1} \right] \\ &= \sum_{t'=0}^{T-1} \sum_{t^*=0}^{T-1} \mathbb{E}_s^{\pi^\theta} \left[\nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) R_{t^*+1} \right]. \end{aligned}$$

For every $t^* < t'$ we see

$$\begin{aligned} &\mathbb{E}_s^{\pi^\theta} \left[\nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) R_{t^*+1} \right] \\ &= \sum_{\tau \in \mathcal{T}} \mathbb{P}_s^{\pi^\theta}((S, A, R) = \tau) \nabla_\theta (\log(\pi^\theta(a_{t'}; s_{t'}))) r_{t^*+1} \\ &= \sum_{a_0} \sum_{s_1} \sum_{r_1} \cdots \sum_{a_{T-1}} \sum_{s_T} \sum_{r_T} \prod_{t=0}^{T-1} \pi^\theta(a_t; s_t) p(\{s_{t+1}, r_{t+1}\}; s_t, a_t) (\nabla_\theta (\log(\pi^\theta(a_{t'}; s_{t'}))) r_{t^*+1}) \\ &= \sum_{a_0} \sum_{s_1} \sum_{r_1} \cdots \sum_{s_{t^*}} \sum_{r_{t^*}} \prod_{t=0}^{t^*} \pi^\theta(a_t; s_t) p(\{s_{t+1}, r_{t+1}\}; s_t, a_t) \sum_{a_{t^*}} \pi^\theta(a_{t^*}; s_{t^*}) \\ &\quad \times \sum_{s_{t^*+1}} \sum_{r_{t^*+1}} p(\{s_{t^*+1}, r_{t^*+1}\}; s_{t^*}, a_{t^*}) r_{t^*+1} \sum_{a_{t^*+1}} \pi^\theta(a_{t^*+1}; s_{t^*+1}) \times \cdots \\ &\quad \times \sum_{s_{t'}} \sum_{r_{t'}} p(\{s_{t'}, r_{t'}\}; s_{t'}, a_{t'}) \underbrace{\sum_{a_{t'}} \pi^\theta(a_{t'}; s_{t'}) \nabla_\theta (\log(\pi^\theta(a_{t'}; s_{t'})))}_{\sum_{a_{t'}} \nabla_\theta \pi^\theta(a_{t'}; s_{t'}) = \nabla_\theta \sum_{a_{t'}} \pi^\theta(a_{t'}; s_{t'}) = \nabla_\theta 1 = 0} \times \overbrace{\sum_{s_{t'}} p(\{s_{t'}, r_{t'}\}; s_{t'}, a_{t'})}^{\text{sum of probabilities}} \underbrace{1}_{\text{sum of probabilities}} \\ &= 0. \end{aligned}$$

It follows directly that

$$\begin{aligned} \nabla_\theta J_s(\theta) &= \sum_{t'=0}^{T-1} \sum_{t^*=0}^{T-1} \mathbb{E}_s^{\pi^\theta} \left[\nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) R_{t^*+1} \right] \\ &= \sum_{t'=0}^{T-1} \sum_{t^*=t'}^{T-1} \mathbb{E}_s^{\pi^\theta} \left[\nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) R_{t^*+1} \right] \\ &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t'=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) \sum_{t^*=t'}^{T-1} R_{t^*+1} \right] \\ &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t'=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_{t'}; S_{t'}))) R_{t'}^T \right]. \end{aligned}$$

□

In the last step we will replace the reward-to-go with the Q-function.



Theorem 5.2.4. Assume that (S, A, R) is a T -step MDP with finite state-action spaces and consider a stationary differentiable parameterized family of policies $\{\pi^\theta : \theta \in \Theta\}$. Then the gradient of the value function can also be written as

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(A_t; S_t)) Q_t^{\pi^\theta}(S_t, A_t) \right].$$

Proof. For the proof we use the property of a Markov reward process to get ⁶

$$Q_t^{\pi^\theta}(S_t, A_t) = \mathbb{E}_s^{\pi^\theta} [R_t^T | (S_t, A_t)].$$

Then the claim follows from Theorem 5.2.3:

$$\begin{aligned} \nabla_\theta J_s(\theta) &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_t; S_t))) R_t^T \right] \\ &= \mathbb{E}_s^{\pi^\theta} \left[\mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_t; S_t))) R_t^T \middle| (S_t, A_t) \right] \right] \\ &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_t; S_t))) \mathbb{E}_s^{\pi^\theta} [R_t^T | (S_t, A_t)] \right] \\ &= \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log(\pi^\theta(A_t; S_t))) Q_t^{\pi^\theta}(S_t, A_t) \right] \end{aligned}$$

□

For intuition the Q-function indicates the *expected* reward-to-go and not the reward-to-go for a specific trajectory. More precisely, we have to distinguish between the outer expectation and the expectation in the definition of the Q-function. While the reward-to-go depends on the trajectory of the outer expectation, the Q-function calculates its own expectation. For the moment one might wonder whether there is any use in this representation, Q^{π^θ} is not known. The miracle why this can be useful will be solved later when we discuss actor-critic methods for which a second parametrisation is used for Q .



Definition 5.2.5. In general optimization methods that aim to maximise f and the gradient of f takes the form of an expectation that can be sampled are typically called stochastic gradient methods. The usual form of the gradients is

$$\nabla f(\theta) = \mathbb{E}[g(X, \theta)],$$

for policy gradient the form is more delicate

$$\nabla f(\theta) = \mathbb{E}^\theta[g(X, \theta)].$$

If (unbiased) samples $\tilde{\nabla} f(\theta)$ for the gradients are available (samples of the expectation), the update scheme

$$\theta \leftarrow \theta - \alpha \tilde{\nabla} f(\theta)$$

is called stochastic gradient algorithm. If more than one sample is used to estimate the gradient as a Monte Carlo average the algorithm is called a batch stochastic gradient ascent algorithm. The number of samples averaged is called the batch size.

⁶irgendwann in super saubere theorie quetschen

Let us come back to the goal of performing a gradient ascent algorithm to maximise V^{π^θ} with initial distribution μ . Using the policy gradient representations the gradient ascent update can be written as

$$\theta_{t+1} = \theta_t - \alpha \mathbb{E}_\mu^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(A_t; S_t)) Q_t^{\pi^\theta}(S_t, A_t) \right]$$

or

$$\theta_{t+1} = \theta_t - \alpha \mathbb{E}_\mu^{\pi^\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(A_t; S_t)) \sum_{t'=t}^{T-1} R_{t'+1} \right].$$

Unfortunately, the exact expectations are typically unknown and must be estimated. Similar to the approximate dynamic programming chapter we rewrote the gradient as an expectation and thus can estimate the expectation with samples. As earlier the first obvious idea to estimate the gradients is the model-free Monte Carlo method. As the Q -function appearing inside the expectation is an unknown value (and an expectation itself) it is most common in applications to use the second policy gradient theorem instead of the third. This can be interpreted as estimating the Q -function by the unbiased estimator R_t^T , the reward-to-go. An estimator for the gradient is then given by

$$\tilde{\nabla} J_\mu(\theta) = \frac{1}{K} \sum_{i=1}^K \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^\theta(a_t^i; s_t^i)) \sum_{t'=t}^{T-1} r_{t'+1} \right], \quad (5.7)$$

with K trajectories $(s_0^i, a_0^i, s_1^i, r_1^i, \dots, a_{T-1}^i, s_T^i, r_T^i)$ sampled according to the current policy π^θ with initial distribution μ .

Algorithm 32: REINFORCE: (Batch-)Stochastic policy gradient algorithm

Data: Initial parameter θ_0 , $K \geq 1$, initial distribution μ

Result: Approximate policy $\pi^{\theta_L} \approx \pi^{\theta^*}$

$l = 1$.

while *not converged* **do**

for $i = 1, \dots, K$ **do**

 Sample initial condition s_0^i from μ .

 Sample trajectory $(s_0^i, a_0^i, s_1^i, r_1^i, \dots, a_{T-1}^i, s_T^i, r_T^i)$ using policy $\pi^{\theta_{l-1}}$.

end

 Determine step-size α .

$\tilde{\nabla} J(\theta_{l-1}) = \frac{1}{K} \sum_{i=1}^K \left[\sum_{t=0}^{T-1} \nabla_\theta (\log \pi^{\theta_{l-1}}(a_t^i; s_t^i)) \sum_{t'=t}^{T-1} r_{t'+1}^i \right]$

$\theta_l = \theta_{l-1} - \alpha \tilde{\nabla} J(\theta_{l-1})$

end

 Set $l = l + 1$.

This famous algorithm, even not in exactly that form, is due to Ronald Williams⁷. To use the algorithm in practice a stopping condition needs to be fixed. Typically, either the number of iterations is fixed or the algorithm is terminated when the norm of the gradient reaches a small threshold.



Sampling a trajectory sounds harmless for a Mathematician. But it is not at all! Imagine we learn how to steer a car or a certain robotic task and π^θ is a policy. Then sampling means running the car (or the robot) K times with the current (possibly

⁷Williams: "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", Machine Learning, 8, 229-256, 1992



very bad) policy while evaluating the rewards! It is thus easy to imagine pitfalls for practical applications. Sample efficiency (using as little samples as possible) in learning is essential and also sampling from very bad policies might be a costly endeavour.

5.2.2 Infinite-time MDPs with discounted rewards

In this section we will consider MDPs with infinite time horizon and discounted rewards. Recall that in this setting the use of stationary policies is justified. As before we aim to rewrite the gradient of the value function as an expectation, but now the infinite time horizon makes it much more complex to calculate the gradient of the value function. The objective function using a parameterised policy π^θ is given by

$$J_s(\theta) = V^{\pi^\theta}(s) = \mathbb{E}_s^{\pi^\theta} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] = \sum_{a \in \mathcal{A}_s} \pi^\theta(a; s) Q^{\pi^\theta}(s, a). \quad (5.8)$$

The infinite sum makes the computation of the gradient more challenging. The trick we will use, is to derive the rhs of (5.8) with the chain rule to get a fixed point equation of $\nabla J(\theta)$. Using this we will be able to obtain a closed form of the gradient.



Theorem 5.2.6. Under the assumption that J_s is differentiable for every start state $s \in \mathcal{S}$ we have a closed form of the gradient

$$\nabla J_s(\theta) = \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} \rho^{\pi^\theta}(s') \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a),$$

where $\rho^\pi(s') = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_s^\pi(S_t = s') = \mathbb{E}_s^\pi[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t=s'}]$.

Proof. The proof uses an ugly induction. With the notation $p(s \rightarrow s'; n, \pi) = \mathbb{P}_s^\pi(S_n = s')$ we first show the following identity by induction:



For all $n \in \mathbb{N}$ it holds that

$$\begin{aligned} \nabla J_s(\theta) &= \sum_{t=0}^n \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\ &\quad + \sum_{s' \in \mathcal{S}} \gamma^{n+1} p(s \rightarrow s'; n+1, \pi^\theta) \nabla J_{s'}(\theta). \end{aligned}$$

By the chain rule we have that

$$\begin{aligned} \nabla J_s(\theta) &= \sum_{a \in \mathcal{A}_s} \left(\nabla \pi^\theta(a; s) Q^{\pi^\theta}(s, a) + \pi^\theta(a; s) \nabla Q^{\pi^\theta}(s, a) \right) \\ &= \sum_{a \in \mathcal{A}_s} \left(\nabla \pi^\theta(a; s) Q^{\pi^\theta}(s, a) + \pi^\theta(a; s) \nabla (r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V^{\pi^\theta}(s')) \right) \\ &= \sum_{a \in \mathcal{A}_s} \left(\nabla \pi^\theta(a; s) Q^{\pi^\theta}(s, a) + \pi^\theta(a; s) \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \nabla J_{s'}(\theta) \right) \\ &= \sum_{a \in \mathcal{A}_s} \left(\nabla \pi^\theta(a; s) Q^{\pi^\theta}(s, a) + \pi^\theta(a; s) \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \left(\sum_{a' \in \mathcal{A}_{s'}} \right. \right. \\ &\quad \left. \left. \left(\nabla \pi^\theta(a'; s') Q^{\pi^\theta}(s', a') + \pi^\theta(a'; s') \gamma \sum_{s'' \in \mathcal{S}} p(s''; s', a') \nabla J_{s''}(\theta) \right) \right) \right) \end{aligned}$$

$$\begin{aligned}
&= \sum_{t=0}^1 \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&\quad + \sum_{s'' \in \mathcal{S}} \gamma^2 p(s \rightarrow s''; 2, \pi^\theta) \nabla J_{s''}(\theta).
\end{aligned}$$

This is the induction for $n = 1$. For the inductive step suppose the statement holds for some $n \in \mathbb{N}$. The same computation, plugging-in the gradient, yields

$$\begin{aligned}
\nabla J_s(\theta) &= \sum_{t=0}^n \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&\quad + \sum_{s' \in \mathcal{S}} \gamma^{n+1} p(s \rightarrow s'; n+1, \pi^\theta) \nabla J_{s'}(\theta) \\
&= \sum_{t=0}^n \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&\quad + \sum_{s' \in \mathcal{S}} \gamma^{n+1} p(s \rightarrow s'; n+1, \pi^\theta) \\
&\quad \times \left(\sum_{a' \in \mathcal{A}_{s'}} \left(\nabla \pi^\theta(a'; s') Q^{\pi^\theta}(s', a') + \pi^\theta(a'; s') \gamma \sum_{s'' \in \mathcal{S}} p(s''; s', a') \nabla J_{s''}(\theta) \right) \right) \\
&= \sum_{t=0}^{n+1} \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&\quad + \sum_{s' \in \mathcal{S}} \gamma^{n+1} p(s \rightarrow s'; n+1, \pi^\theta) \cdot \left(\sum_{a' \in \mathcal{A}_{s'}} \pi^\theta(a'; s') \gamma \sum_{s'' \in \mathcal{S}} p(s''; s', a') \nabla J_{s''}(\theta) \right) \\
&= \sum_{t=0}^{n+1} \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&\quad + \sum_{s'' \in \mathcal{S}} \gamma^{n+2} p(s \rightarrow s''; n+2, \pi^\theta) \nabla J_{s''}(\theta).
\end{aligned}$$

The remainder term vanishes in the limit:

$$\left| \sum_{s'' \in \mathcal{S}} \gamma^{n+2} p(s \rightarrow s''; n+2, \pi^\theta) \nabla J_{s''}(\theta) \right| \leq \gamma^{n+2} |\mathcal{S}| \max_{s \in \mathcal{S}} |\nabla J_s(\theta)| \rightarrow 0, \quad n \rightarrow \infty.$$

Hence, we proved that

$$\begin{aligned}
\nabla J_s(\theta) &= \sum_{t=0}^{\infty} \sum_{s' \in \mathcal{S}} \gamma^t p(s \rightarrow s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\
&= \sum_{s' \in \mathcal{S}} \rho^{\pi^\theta}(s') \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a).
\end{aligned}$$

The exchange of limits and sums is justified as we assume \mathcal{S} to be finite. \square



For episodic MDPs (the MDP terminates almost surely under all policies π_θ), we can get rid of the assumption of the existence of $\nabla J_s(\theta)$. Go through the proof of Theorem 5.2.6 and argue why it is enough to assume the existence of $\nabla \pi_\theta(\cdot; s)$ for all $s \in \mathcal{S}$.

Taking a closer look at the expression from the gradient, it is obvious that the infinite sum $\rho^\pi(s') = \mathbb{E}_s^\pi[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t=s'}]$ is unpleasant for implementations. There is an alternative way to rewrite the policy gradient theorem using the discounted state visitation measure:



Definition 5.2.7. The discounted state-visitation measure under policy π for starting state $s \in \mathcal{S}$ is given by

$$d_s^\pi(s) := \frac{\rho^\pi(s)}{\sum_{s'} \rho^\pi(s')}.$$

First note that, using Fubini's theorem,

$$\sum_{s' \in \mathcal{S}} \rho^{\pi^\theta}(s') = \sum_{s' \in \mathcal{S}} \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t=s'} \right] = \frac{1}{1-\gamma}.$$

Hence, the normalised state-visitation measure is actually much simpler: $d^\pi(s) = (1-\gamma)\rho^\pi(s)$. With this definition the gradient can also be written as follows:



Theorem 5.2.8. Under the assumption that $J_s(\theta)$ is differentiable for every start state $s \in \mathcal{S}$ the gradient can be expressed as

$$\begin{aligned} \nabla J_s(\theta) &= \frac{1}{1-\gamma} \mathbb{E}_{S \sim d_s^{\pi^\theta}, A \sim \pi^\theta(\cdot; S)} \left[\nabla \log(\pi^\theta(A; S)) Q^{\pi^\theta}(S, A) \right] \\ &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d_s^{\pi^\theta}(s') \pi^\theta(a; s') \nabla \log(\pi^\theta(a; s')) Q^{\pi^\theta}(s', a). \end{aligned}$$

In essence the theorem says the following. Draw a state-action pair (s, a) according to their relevance measured in terms of expected discounted frequency, compute the direction using the score function, and follow that direction according to the expected reward.

Proof. By Theorem 5.2.6 we have

$$\begin{aligned} \nabla J_s(\theta) &= \sum_{s' \in \mathcal{S}} \rho^{\pi^\theta}(s') \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\ &= \sum_{\hat{s} \in \mathcal{S}} \rho^{\pi^\theta}(\hat{s}) \sum_{s' \in \mathcal{S}} d_s^{\pi^\theta}(s') \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\ &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} d_s^{\pi^\theta}(s') \sum_{a \in \mathcal{A}_{s'}} \pi^\theta(a; s') \nabla \log(\pi^\theta(a; s')) Q^{\pi^\theta}(s', a) \\ &= \frac{1}{1-\gamma} \mathbb{E}_{S \sim d_s^{\pi^\theta}, A \sim \pi^\theta(\cdot; S)} \left[\nabla \log(\pi^\theta(A; S)) Q^{\pi^\theta}(S, A) \right], \end{aligned}$$


where we added a 1 and used the lock-trick. The second equality gives the first claim, the final equality the second claim. \square

For practical use it will be important to estimate the gradient (an expectation) using Monte Carlo. Thus, it will be necessary to sample from the discounted occupation measures and then compute score-function and Q -value. On first sight sampling from the d^π looks infeasible as an infinite sum is involved. In fact, a sample can be obtained by following a rollout up to an independent geometric random variable, counting the number of visitations and then drawing from this empirical distribution. This is justified by a quick computation:




If $T \sim \text{Geo}(1-\gamma)$ is independent from the MDP, then $d^\pi(s) = \mathbb{E}_s^\pi \left[\sum_{t=0}^{T-1} \mathbf{1}_{S_t=s} \right]$. This is easily seen as

$$\mathbb{E}_s^\pi \left[\sum_{t=0}^{T-1} \mathbf{1}_{S_t=s} \right] = \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathbf{1}_{t \leq T-1} \mathbf{1}_{S_t=s} \right]$$



$$\begin{aligned}
&\stackrel{\text{Fubini}}{=} \sum_{t=0}^{\infty} \mathbb{E}_s^{\pi} [\mathbf{1}_{t \leq T} \mathbf{1}_{S_t=s}] \\
&\stackrel{\text{ind.}}{=} \sum_{t=0}^{\infty} \mathbb{P}(t \leq T) \mathbb{E}_s^{\pi} [\mathbf{1}_{S_t=s}] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi} [\mathbf{1}_{S_t=s}] \\
&= \mathbb{E}_s^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t=s} \right]
\end{aligned}$$

Here is another representation in the spirit of the finite-time policy gradient theorem. The representation avoids the sampling from the discounted occupation measure but instead uses trajectories.



Theorem 5.2.9. Suppose that $(s, a) \mapsto \nabla_{\theta}(\log \pi^{\theta}(a; s))Q^{\pi^{\theta}}(s, a)$ is bounded. Then

$$\nabla_{\theta} J_s(\theta) = \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta}(\log \pi^{\theta}(A_t; S_t))Q^{\pi^{\theta}}(S_t, A_t) \right].$$

The assumption does not hold for arbitrary parametrisation and might also be hard to check. For softmax policies the score-function can be computed and is bounded for instance for bounded feature vector. If the rewards are additionally bounded (which implies the Q -function is bounded) the assumption of the theorem holds.

Proof.

$$\begin{aligned}
\nabla_{\theta} J_s(\theta) &= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in \mathcal{S}} \mathbb{P}_s^{\pi^{\theta}}(S_t = s') \sum_{a \in \mathcal{A}} \nabla \pi^{\theta}(a; s') Q^{\pi^{\theta}}(s', a) \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{a \in \mathcal{A}} \nabla \pi^{\theta}(a; S_t) Q^{\pi^{\theta}}(S_t, a) \right] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{a \in \mathcal{A}} \pi^{\theta}(a; S_t) \nabla \log \pi^{\theta}(a; S_t) Q^{\pi^{\theta}}(S_t, a) \right] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^{\theta}} \left[\nabla \log \pi^{\theta}(A_t; S_t) Q^{\pi^{\theta}}(S_t, A_t) \right] \\
&= \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla \log \pi^{\theta}(A_t; S_t) Q^{\pi^{\theta}}(S_t, A_t) \right]
\end{aligned}$$

The final exchange of sum and expectation is justified by dominated convergence and the assumption of the theorem. \square

A dirty version of the REINFORCE algorithm is then obtained by running trajectories up to some large T and to use the truncated series as an estimator for the gradient. Since the truncated estimator is not unbiased smarter approaches have been proposed. As for the discounted occupation measure one might ask if geometric times can be used to truncate the infinite time-horizon. Almost the same computation gives the following theorem⁸.

⁸Zhang, Koppel, Zhu, Basar: "Global Convergence of Policy Gradient Methods to (Almost) Locally Optimal Policies", SIAM Journal on Control and Optimization, 2020



Proposition 5.2.10. Suppose that $(s, a) \mapsto \nabla_{\theta}(\log \pi^{\theta}(a; s))Q^{\pi^{\theta}}(s, a)$ is bounded and $T, T' \sim \text{Geo}(1 - \gamma^{1/2})$ are independent of each other and the MDP, then

$$\nabla J(\theta) = \frac{1}{1 - \gamma} \mathbb{E}_s^{\pi^{\theta}} \left[\nabla \log(\pi^{\theta}(S_T; A_T)) \sum_{t=T}^{T+T'} \gamma^{(t-T)/2} R(S_t, A_t) \right].$$

The advantage of this policy gradient theorem is clear, unbiased estimates can be obtained by running the MDP for a finite (random) number of steps.

Proof. spaeter ausfuehren, war nicht in vorlesung ⁹ □

The representation is not only useful from a practical point of view. The random variables can be shown to have bounded variance, an ingredient that was crucial for the proof of convergence to a stationary point for stochastic gradient schemes. Exactly the same proof can be used to proof convergence of REINFORCE to a stationary point of J . Combined with an additional PI inequality (only known for tabular softmax parametrisation) then also yields convergence to an optimal policy.

5.2.3 Convergence of REINFORCE

.....

5.2.4 Variance reduction tricks

Traditionally it is believed that reducing the variance in estimating the gradient $\nabla J(\theta)$ is crucial. Recalling the discussion from Section 1.3.4 there are also other effects when using the variance reduction by baselines for the softmax policy gradient approach to stochastic bandits. Nonetheless, reducing variances is certainly a good idea.

Baseline

Let us generalise the baseline idea from stochastic bandits to general MDP policy gradient, formulated in the finite MDP setting.



Theorem 5.2.11. Assume that (S, A, R) is a T -step MDP with finite state-action spaces and consider a stationary differentiable parameterized family of policies $\{\pi^{\theta} : \theta \in \Theta\}$. For any $b \in \mathbb{R}$ the gradient of $J(\theta)$ can also be written as

$$\nabla_{\theta} J(\theta) = \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta}(\log \pi^{\theta}(A_t; S_t)) \underbrace{(Q_t^{\pi^{\theta}}(S_t, A_t) - b)}_{\text{or } (R_t^T - b) \text{ or } (R_0^T - b)} \right].$$

Proof. The computation is very similar to the bandit case, we redo the log-trick and see that for every $t \leq T - 1$

$$\begin{aligned} \mathbb{E}_s^{\pi^{\theta}} [\nabla_{\theta}(\log \pi^{\theta}(A_t; S_t)) b] &= b \sum_{s_t \in \mathcal{S}} \sum_{a_t \in \mathcal{A}_s} \mathbb{P}_s^{\pi^{\theta}}(S_t = s_t) \pi^{\theta}(a_t; s_t) \nabla_{\theta}(\log \pi^{\theta}(a_t; s_t)) \\ &= b \sum_{s_t \in \mathcal{S}} \mathbb{P}_s^{\pi^{\theta}}(S_t = s_t) \sum_{a_t \in \mathcal{A}_s} \nabla_{\theta} \pi^{\theta}(a_t; s_t) \\ &= b \sum_{s_t \in \mathcal{S}} \mathbb{P}_s^{\pi^{\theta}}(S_t = s_t) \nabla_{\theta} \underbrace{\sum_{a_t \in \mathcal{A}} \pi^{\theta}(a_t; s_t)}_{=1} = 0. \end{aligned}$$

⁹ausfuehren

Thus, the additional term vanishes in all three representations of the gradient so that $-b$ can be added. \square



Show that the constant baseline b can be replaced by any deterministic state-dependent baseline $b : \mathcal{S} \rightarrow \mathbb{R}$, i.e.

$$\nabla_{\theta} J(\theta) = \mathbb{E}_s^{\pi^{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(A_t; S_t)) (Q_t^{\pi^{\theta}}(S_t, A_t) - b(S_t)) \right].$$

We will come back to the most important state-dependent baseline $b(s) = V^{\pi^{\theta}}(s)$ when we discuss actor-critic methods.



Write down and prove the baseline gradient representation for infinite discounted MDPs.

In the bandit case we have already seen that the baseline trick is a variance reduction trick. The same is true in the MDP setting.

Importance sampling trick/likelihood ratio method

There is a nice trick from stochastic numerics that is useful in many situations for reinforcement learning, the so-called importance sampling trick (or likelihood ratio method). Suppose $m = \mathbb{E}[g(X)]$ is to be approximated using a Monte Carlo method. The likelihood ratio trick is used in two situations:

- If it is hard to sample X but easy to simulate a random variable Y with similar density.
- The variance $g(X)$ should be reduced.

The first advantage of the method is what is closer to the interest of reinforcement learning. Before showing why let us recall the trick. Suppose X has density f_X and Y is another random variable with (positive) density f_Y . Then

$$\mathbb{E}[g(X)] = \int g(x) f_X(x) dx = \int g(y) \frac{f_X(y)}{f_Y(y)} f_Y(y) dy = \mathbb{E}[\tilde{g}(Y)]$$

with $\tilde{g}(y) = g(y) \frac{f_X(y)}{f_Y(y)}$. Suppose that f_X is close to a known density f_Y but somehow wiggles around. It might then be much easier to simulate Y instead and correct the Monte Carlo estimator with the likelihood ratios. Alternatively, if $\tilde{g}(Y)$ has smaller variance than $g(X)$, then the Monte Carlo convergence using Y would converge faster. This is typically the case if Y has more mass on the important values (justifying the name importance sampling) giving the most contribution to the expectation.

We now turn towards reinforcement learning and, quite surprisingly, find the trick useful to construct off-policy estimators for the policy gradients! These are estimators that do not use samples produced from π^{θ} . Here is the main trick:



Suppose $\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$ is a policy and π, π^b are two policies. Using the path probabilities gives

$$\frac{\mathbb{P}_{\mu}^{\pi}(\tau)}{\mathbb{P}_{\mu}^{\pi^b}(\tau)} = \frac{\mu(s_0) \delta_0(r_0) \prod_{i=0}^{T-1} \pi(a_i; s_i) p(s_{i+1}; s_i, a_i)}{\mu(s_0) \delta_0(r_0) \prod_{i=0}^{T-1} \pi^b(a_i; s_i) p(s_{i+1}; s_i, a_i)} = \prod_{i=0}^{T-1} \frac{\pi(a_i; s_i)}{\pi^b(a_i; s_i)}.$$

What is crucial here is the cancellation of the transition p , the change from π to π^b is model-free!

The likelihood ratio trick now does the following. Write down the policy gradient estimator, replace π^{θ} by some behavior policy π^b and compensate by inserting the likelihood ratio. Here is the version of the finite-time non-discounted case:


Proposition 5.2.12. (Likelihood ratio trick for policy gradients)

Suppose π^b is a behavior policy (with strictly positive weights), then

$$\nabla_{\theta} J_{\mu}(\theta) = \mathbb{E}_{\mu}^{\pi^b} \left[\prod_{i=0}^{T-1} \frac{\pi^{\theta}(A_i; S_i)}{\pi^b(A_i; S_i)} \sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(A_t; S_t)) Q_t^{\pi^{\theta}}(S_t, A_t) \right]$$

Proof. All we need to do is to use the definition of (discrete) expectations and then apply the likelihood trick:

$$\begin{aligned} & \mathbb{E}_{\mu}^{\pi^b} \left[\sum_{t=0}^{T-1} \prod_{i=0}^{T-1} \frac{\pi^{\theta}(A_i; S_i)}{\pi^b(A_i; S_i)} \nabla_{\theta} (\log \pi^{\theta}(A_t; S_t)) Q_t^{\pi^{\theta}}(S_t, A_t) \right] \\ &= \sum_{\tau: \text{trajectory}} \mathbb{P}_{\mu}^{\pi^b}(\tau) \prod_{i=0}^{T-1} \frac{\pi^{\theta}(a_i; s_i)}{\pi^b(a_i; s_i)} \sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(a_t; s_t)) Q_t^{\pi^{\theta}}(s_t, a_t) \\ &= \sum_{\tau: \text{trajectory}} \mathbb{P}_{\mu}^{\pi^b}(\tau) \frac{\mathbb{P}_{\mu}^{\pi^{\theta}}(\tau)}{\mathbb{P}_{\mu}^{\pi^b}(\tau)} \sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(a_t; s_t)) Q_t^{\pi^{\theta}}(s_t, a_t) \\ &= \sum_{\tau: \text{trajectory}} \mathbb{P}_{\mu}^{\pi^{\theta}}(\tau) \sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(a_t; s_t)) Q_t^{\pi^{\theta}}(s_t, a_t) \\ &= \mathbb{E}_{\mu}^{\pi^{\theta}} \left[\prod_{i=0}^{T-1} \sum_{t=0}^{T-1} \nabla_{\theta} (\log \pi^{\theta}(A_t; S_t)) Q_t^{\pi^{\theta}}(S_t, A_t) \right] \\ &= \nabla J(\theta). \end{aligned}$$

□

There are several reasons why the trick can be useful. It can be used to reduce the variance of policy gradients (the original idea of the trick). Somehow this seems not to be very successful. Alternatively, the trick can be used to simulate from only one policy to approximate all policy gradients $\nabla_{\theta} J_{\mu}(\theta)$, i.e. run an off-policy policy gradient ascent, see for instance¹⁰. This approach turns out to be delicate as small values of π^b can strongly increase the variance. From the theoretical point of view the likelihood trick is very useful as the REINFORCE algorithm turns into a true stochastic gradient descent algorithm of a function $f(\theta) = \mathbb{E}[h(X, \theta)]$, the dependence of θ is removed from the sampled law.

5.2.5 Natural policy gradient

5.3 A quick dive into neural networks

5.3.1 What are neural network functions?

To get the discussion started let us look at the simplest example, that of a single neuron. A neuron in our brain is an electrically excitable cell that fires electric signals called action potentials. In mathematical term, a neuron is a function that takes a vector (incoming information) and returns a number (signal, e.g. 0 or 1). A network of billions of communicating neurons passes incoming signals to outgoing action signals. Inspired from neuroscience a century of research in computer science has led to incredible results in artificial intelligence. Let us first describe

¹⁰Metelli, Papini, Faccio, Restelli: "Policy Optimization via Importance Sampling", NIPS 2018

the simplest model of a neuron, and then continue with artificial networks of neurons (so-called neural networks or perceptrons). The simplest model¹¹ is

$$f(x) = \mathbf{1}_{[0, \infty)} \left(\sum_{i=1}^d w_i x_i + b \right).$$

If the combined strength of the signals (weighted according to neurons judgement of importance for the signals) exceeds a certain level the neural fires, otherwise not. More generally, for the definition of an artificial neuron the indicator is typically replaced by a generic function σ , the so-called activation function.



Definition 5.3.1. For $\sigma : \mathbb{R} \rightarrow \mathbb{R}$, $w \in \mathbb{R}^d$, and $b \in \mathbb{R}$, the function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ defined by

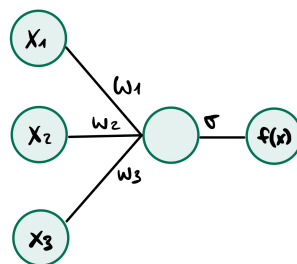
$$f(x) = \sigma \left(\sum_{i=1}^d w_i x_i + b \right)$$

is called an artificial neuron with activation function σ , weights w_1, \dots, w_d for the signals x_1, \dots, x_d and bias b .

The weights can be interpreted as importance of the signal for the neuron, the bias as susceptibility of the neuron.

Example 5.3.2. Here are some typical activation functions:

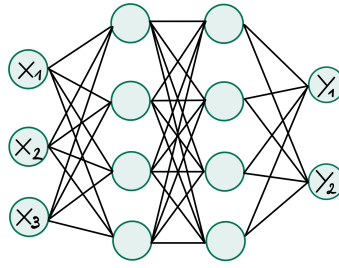
- $\sigma(s) = \mathbf{1}_{[0, \infty)}(s)$, the Heavyside function, yields a neuron that either fires or not,
- $\sigma(s) = s$ yields a linear neuron which is not really used,
- $\sigma(s) = \max\{0, s\}$ is called rectified linear unit (ReLU), a very typical practical choice,
- $\sigma(s) = \frac{1}{1+e^{-s}} = \frac{e^s}{1+e^s}$ is called logistic activation function. The logistic activation function is nothing but the softmax probability for one of two possible actions.



Graphical representation of an artificial neuron

As mentioned the brain consists of billions of neurons that form a network in some very complicated manner. We will do the same, transfer the outputs of neurons into new neurons.

¹¹McCulloch, Pitts: "A logical calculus of ideas immanent in nervous activity", Bull. Math. Biophys., 5:115-133, 1943



Graphical representation of a neural network with 2 hidden layers, 3 input dimensions, 2 output dimensions

Here is a mathematical formulation of a simple network of neurons using the same activation function, such networks of artificial neurons are also called multilayer perceptrons¹²:



Definition 5.3.3. (Feedforward neural networks)

Let $d, L \in \mathbb{N}$, $L \geq 2$, and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. Then a multilayer perceptron (or fully connected feedforward neural network) with d -dimensional input, L layers, and activation function σ is a function $F : \mathbb{R}^d \rightarrow \mathbb{R}^k$ that can be written as

$$F(x) = T_L(\sigma(T_{L-1}(\dots \sigma(T_1(x))))),$$

where $L - 1$ is the number of hidden layers with N_l units (number of neurons per layer), $N_0 = d$ and $N_L = k$, and $T_l(x) = W_l x + b_l$ are affine functions with

- weight matrices $W_l \in \mathbb{R}^{N_l \times N_{l-1}}$,
- biases $b_l \in \mathbb{R}^{N_l}$.

Note that the function σ is applied coordinate-wise to the vectors. A neural network is called shallow if $L = 1$, i.e. there is only one hidden layer and deep if the number of layers is large. Finally, if the output is supposed to be a probability vector one typically applies a final normalisation with the softmax function $\Phi(y)_i = \frac{e^{y_i}}{\sum_{j=1}^k e^{y_j}}$

so the output reads as

$$F(x) = \Phi(T_L(\sigma(T_{L-1}(\dots \sigma(T_1(x)))))).$$

¹³ The multilayer perceptron is quite special in its structure. All neurons of a fixed layer receive the same input vector (the outputs of the neurons before) which is then multiplied with the weight-vector of the neuron, shifted, and plugged-into the activation function to yield the signal. To visualise the mapping one can for instance write the real-valued weights on the arrow pointing into a neuron. If a weight is forced to be zero the arrow is usually removed from the graphical representation. This neural network was made up by thinking, not by brute-force training on mass data. The network produces a vector of features of an image, such as "number of circles", "number of edges", ...



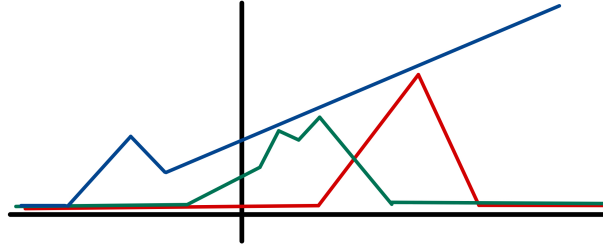
Definition 5.3.4. The architecture of a feedforwards neural network is the chosen number of layers, units, and fixed zero-weights.

The class of neural network functions has been investigated for decades. While the first approach was to understand what particular architecture leads to desired behavior the modern approach is

¹²F. Rosenblatt: "The perceptron: a probabilistic model for information storage and organization in the brain" Psychological review, 65(6):386, 1958

¹³include drawings

to use mass data to learn the weights and biases by optimising a carefully chosen loss-function. Keeping in mind that the graphical representation of the architecture is only a visualisation of the weights it can be instructive to look at plots of very small-dimensional neural network functions with ReLU activation function:



Three functions obtained from ReLU neural networks

To get an idea why neural networks with the simple ReLU activation and linear output function can create such functions two observations are useful:

- One can construct addition, shifting, and linear transformations of functions obtained from a neural network through a neural network with same activation function. To do so write the neural networks on top of each other without connections (i.e. set the linking weights to 0), and use the output weights to obtain the desired transformation.
- The sum $\text{ReLU}(-1) - 2\text{ReLU}(0) + \text{ReLU}(1)$ gives a spike at 0, with $\text{ReLU}(b) = \max\{0, b\}$.

There are other classes of artificial neural networks that are much more relevant in applications. Neural networks that do not only pass signals forwards but also have arrows pointing to previously used neurons are called recurrent neural networks. Those are particularly useful to model time-dependent behavior. More recently, the so-called transformer network architecture has received enormous attention and allowed massive progress in large language models. Since this is an introductory course to reinforcement learning we will only discuss two important features of neural networks, function approximation and differentiation by backwards propagation through the network.

5.3.2 Approximation properties

We will show that already the simplest neural network functions are useful enough to approximate important classes of functions, so-called Boolean functions and continuous functions. The proofs are not complicated but only lead to rather uninteresting existence results¹⁴.



Theorem 5.3.5. (Approximation of Boolean functions)

Every Boolean function $f : \{0, 1\}^d \rightarrow \{0, 1\}$ can be represented by a neural network with activation function $\sigma(x) = \mathbf{1}_{[0, \infty)}$.

Proof. First note that for all Boolean variables $a, b \in \{0, 1\}$ it holds that $2ab - a - b \leq 0$ with equality if and only if $a = b$ (check the cases). Thus, for $u \in \{0, 1\}^d$,

$$\mathbf{1}_{x=u} = \mathbf{1}_{[0, \infty)} \left(\sum_{i=1}^d (2x_i u_i - x_i - u_i) \right) = \sigma \left(\sum_{i=1}^d (2x_i u_i - x_i - u_i) \right)$$

¹⁴Check for instance these lecture notes of Philipp Christian Petersen for more.

Now denoting by $A := \{u \in \mathbb{R}^d : f(u) = 1\}$ it follows that

$$f(x) = \sigma\left(-1 + \sum_{u \in A} \mathbf{1}_{x=u}\right) = \sigma\left(-1 + \sum_{u \in A} \sigma\left(\sum_{i=1}^d (2x_i u_i - x_i - u_i)\right)\right).$$

This, in fact, this is nothing but a neural network with two hidden layers. The first with $|A|$ units, the second with one unit. \square



Definition 5.3.6. A subset \mathcal{F} of neural networks has the universal approximation property if for any $K \subset \mathbb{R}^d$, any $\varepsilon > 0$, and any continuous function $g : K \rightarrow \mathbb{R}$ there exists a neural network $f \in \mathcal{F}$ with $\|f - g\|_\infty < \varepsilon$.

In fact, for many activation functions the universal approximation property holds. An activation function is called sigmoid (*S*-formed) if it is bounded with limits $\lim_{x \rightarrow +\infty} \sigma(x) = 1$ and $\lim_{x \rightarrow -\infty} \sigma(x) = 0$. The terminology comes from the logistic activation function that really looks like an *S*. An important result on neural networks is that shallow (one hidden layer) feedforward neural networks with sigmoid activation function have the universal approximation property.



Theorem 5.3.7. (Universal approximation theorem)

If the activation function σ is sigmoid then the set of all shallow feedforward neural networks has the universal approximation property.

Proof. Let us denote by \mathcal{F}_p the subset of shallow neural networks with p neurons. We first consider $d = 1$ and show that Lipschitz functions on $[0, 1]$ can be approximated. To do so we first approximate h with piecewise constant functions and then approximate the piecewise constant function with a shallow neural network function. For that sake let $x_i = \frac{i}{p}$ and set $h_p(x) = \sum_{i=1}^p h(x_i) \mathbf{1}_{[x_{i-1}, x_i)}(x)$.



There is $f \in \mathcal{F}_p$ with $\|f - h\|_\infty \leq C \omega(h, 1/p)$ with C independent of h .

In the estimate the so-called modulus of continuity appears:

$$\omega(h, \delta) = \sup_{x, y \in \mathbb{R} : |x - y| \leq \delta} |h(x) - h(y)|$$

For a function $h : \mathbb{R} \rightarrow \mathbb{R}$ the modulus of continuity is a very rough measure for the flatness of h . The smaller ω the flatter the function. For Lipschitz continuous functions with Lipschitz constant L it follows immediately that $\omega(h, \delta) \leq L\delta$, the fluctuation of f over intervals of lengths δ are at most δL . The estimate shows that Lipschitz functions with smaller constant L can be well-approximated with $1/L$ many units, thus, rougher functions need more units to be approximated. We will only use the statement for the exponential function which is clearly Lipschitz continuous on compact sets (bounded derivative on compact sets). To prove the first claim note that

$$\sup_{x \in [0, 1]} |h(x) - h_p(x)| = \sup_{x \in [0, 1]} \left| h(x) - \sum_{i=1}^p h(x_i) \mathbf{1}_{[x_{i-1}, x_i)}(x) \right| \leq \omega(h, 1/p)$$

by the definition of h_p and ω . With a telescopic sum we rewrite

$$h_p(x) = h(x_1) + \sum_{i=1}^{\lfloor px \rfloor} (h(x_{i+1}) - h(x_i)).$$

Next, we define a the shallow neural network function with p units as

$$f_p(x) = h(x_1)\sigma(\alpha) + \sum_{i=1}^{p-1} (h(x_{i+1}) - h(x_i))\sigma(\alpha(px - i)) \in \mathcal{F}_p$$

for some $\alpha \in \mathbb{R}$ that is specified next. Fix $\varepsilon > 0$ and choose α large enough such that $|\sigma(z) - \mathbf{1}_{[0,\infty)}(z)| \leq \frac{\varepsilon}{p}$ whenever $|z| \geq \alpha$. Since σ is a sigmoid function this is possible. The choice of α shows that

$$|\sigma(\alpha(px - i)) - \mathbf{1}_{i \leq \lfloor px \rfloor}| \leq \frac{\varepsilon}{p}$$

holds for all $i \notin \{\lfloor px \rfloor, \lfloor px \rfloor + 1\}$ because then $|\alpha(px - i)| > \alpha$. It then follows that

$$\begin{aligned} & |f_p(x) - h_p(x)| \\ &= \left| h(x_1)(\sigma(\alpha) - 1) + \sum_{i=1}^{p-1} (h(x_{i+1}) - h(x_i))(\sigma(\alpha(px - i)) - \mathbf{1}_{i \leq \lfloor px \rfloor}) \right| \\ &\leq \frac{\varepsilon}{p} (|h(x_1)| + (p-2)\omega(h, 1/p)) + |h(x_{\lfloor px \rfloor+1}) - h(x_{\lfloor px \rfloor})| |\sigma(\alpha(px - \lfloor px \rfloor)) - 1| \\ &\quad + |h(x_{\lfloor px \rfloor+2}) - h(x_{\lfloor px \rfloor+1})| |\sigma(\alpha(px - \lfloor px \rfloor - 1)) - 1|. \end{aligned}$$

The first summand can be made arbitrarily small, the other two can both be estimated by $\omega(h, 1/p)(1 + \|\sigma\|_\infty)$. Combining the approximation of h by h_p and the approximation of h_p by f_p it follows that for all $\delta > 0$ there is some p such that

$$\begin{aligned} \sup_{x \in [0,1]} |h(x) - f_p(x)| &\stackrel{\Delta}{\leq} \sup_{x \in [0,1]} |h(x) - h_p(x)| + \sup_{x \in [0,1]} |h_p(x) - f_p(x)| \\ &\leq \omega(h, 1/p) + \delta + 2(\|\sigma\|_\infty + 1)\omega(h, 1/p). \end{aligned}$$

Now the claim follows. Covering a compact set $K \subseteq \mathbb{R}$ by finitely many intervals it then follows that for all $h : K \rightarrow \mathbb{R}$ and all $\varepsilon > 0$ there is a neural network function f such that $\sup_{x \in K} |h(x) - f(x)| < \varepsilon$.

It remains to extend the arguments to the d -dimensional case. To reduce to the one-dimensional case a simple dense family is used:



The set

$$\mathcal{E} := \left\{ g : K \rightarrow \mathbb{R} \mid g(x) = \sum_{i=1}^N s_i e^{\langle v^i, x \rangle}, x \in \mathbb{R}^d, N \in \mathbb{N}, v^i \in \mathbb{R}^d, s_i \in \{-1, 1\} \right\}$$

is dense in $(C(K), \|\cdot\|_\infty)$.

Using Stone-Weierstrass from functional analysis one only needs to check that \mathcal{E} is an algebra that separates points. Easy.

Next, we approximate \mathcal{E} by neural network functions with one hidden layer:



For every $g(x) = \sum_{i=1}^N s_i e^{\langle v^i, x \rangle} \in \mathcal{E}$ and every $\varepsilon > 0$ there is a neural network function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ with one hidden layer such that $\sup_{x \in K} |g(x) - f(x)| < \varepsilon$.

Define $g_{v^i}(x) = e^{\langle v^i, x \rangle}$. The trick is simple. Let \bar{K}_i the linear transformation of K_i under $x \mapsto \langle v^i, x \rangle$. Then $\bar{K}_i \subseteq \mathbb{R}$ is compact again. Next, apply the first step to chose from \mathcal{F}_p an

$$f_i(x) = \sum_{l=1}^p w_l \sigma(x - b)$$

with $\sup_{x \in \bar{K}_i} |f_i(x) - e^x| < \frac{\varepsilon}{N}$. Then define the neural network function $\bar{f}_i : \mathbb{R}^d \rightarrow \mathbb{R}$ as

$$\bar{f}_i(x) = s_i f_i \left(\sum_{k=1}^d v_k^i x_k \right)$$

so that

$$\begin{aligned} \sup_{x \in K} |\bar{f}_i(x) - s_i e^{\langle v^i, x \rangle}| &= \sup_{x \in K} |s_i f_i(\langle v^i, x \rangle) - s_i e^{\langle v^i, x \rangle}| \\ &= \sup_{x \in \bar{K}_i} |s_i f_i(x) - s_i e^x| < \frac{\varepsilon}{N}. \end{aligned}$$

To see that \bar{f}_i is a neural network function with one hidden layer note that

$$\bar{f}_i(x) = \sum_{l=1}^p s_i w_l \sigma \left(\sum_{k=1}^d v_k^i x_k - b_l \right).$$

Recalling that sums of neural network functions are neural network functions with the same number of hidden layers (stacking neural networks without connections) a neural network approximation of g is given by $f = \sum_{i=1}^N \bar{f}_i$:

$$\sup_{x \in K} |f(x) - g(x)| \leq \sum_{i=1}^N \sup_{x \in K} |\bar{f}_i(x) - s_i e^{\langle v^i, x \rangle}| < \varepsilon.$$



The universal approximation property is satisfied for \mathcal{F}_p .

Suppose $h \in (C(K), \|\cdot\|_\infty)$ and $\varepsilon > 0$. Using the two steps before we can choose $g \in \mathcal{E}$ with $\sup_{x \in K} |g(x) - h(x)| < \frac{\varepsilon}{2}$ and a neural network function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ with $\sup_{x \in K} |f(x) - h(x)| < \frac{\varepsilon}{2}$. Then $\sup_{x \in K} |h(x) - f(x)| \leq \sup_{x \in K} |h(x) - g(x)| + \sup_{x \in K} |g(x) - f(x)| < \varepsilon$. This proves the theorem. \square

5.3.3 Differentiation properties

There are a couple of reasons why the use of deep neural networks is extremely successful in reinforcement learning. We will highlight here a differentiation property that led to the breakthroughs in image classification. Suppose a set of labeled images is given: $\{(x_i, y_i) : i \leq N\}$. We assume the images are already transformed into a pixel vector, say \mathbb{R}^d , and they are labeled as a k -dimensional feature vector. A feature vector is an abstraction of an image used to characterize and numerically quantify the contents of an image. Simply put, a feature vector is a list of numbers used to represent an image. The pixel vector itself is a possible feature vector that fully describes an image but the aim is extract as little abstract information as possible that describes a images as good as possible. This could be counting geometric forms or more specifically the number of persons on the images.



A main goal of image analysis is to define feature vectors that encode the images well and then find mappings that are simple to evaluate and map an image to its feature vector.

The typical approach consists learning both tasks on a large training set $\{(x_i, y_i)\}$ of images x_i for which the labels y_i were coded manually by humans. There are plenty of such examples, for instance imageNet¹⁵ with more than fourteen million labeled images. There are two amazing features why neural networks are used to map images to feature vectors:

¹⁵imageNet website

- Differentiation: neural network structures allow to differentiate well for the weights, thus, allowing gradient descent methods to minimise the error between outputs of the neural network and the labels. Since neural networks used have dozens of millions of parameters this is crucial. Imagine for every gradient step to compute a gradient of fifty million partial derivatives. This is only possible if many of the computations can be reused, a property that holds for neural network functions due to the so-called back propagation algorithm.
- Generalisation: for some magic reasons that are not fully understood, yet, neural networks generalise extremely well. This means that the mapping from image to labels obtained by gradient descent on a finite labeled set generalises well to further images. Overfitting is a smaller problem than one might imagine (at least there are efficient tricks such as dropout) if for instance sixty million parameters are used to classify one million images. For imageNet one typically uses a training set of 1.2 million images and then compares the generalisation property for the other images.

Since this is not a course on deep learning we only discuss a very nice differentiation trick. Algorithm 33 summarises the most famous one, the delta-method that can be used to train a neural network on given input/output pairs or to just compute the gradient $\nabla_w F(x)$.



Theorem 5.3.8. The δ -method from Algorithm 33 is nothing but stochastic gradient descent for $F_w := \sum_{i=1}^N \frac{1}{2} \|F_w(x)_i - y_i\|^2$ with (x_i, y_i) chosen uniformly from the training set.

In the algorithm and the proof we allowed a generic function Φ to normalise the outputs so the chain of concatenations is

$$F(x) = T_L(\sigma(T_{L-1}(\dots \sigma(T_1(x))))),$$

This is needed if the network should output a probability vector.

Proof. In what follows we use the definitions of h , V appearing in the algorithm. Please check Figure ??¹⁶ for a graphical representation to ease the understanding.¹⁷ First of all note that the optimisation is equivalent to optimising $\frac{1}{N} F_w$, thus, F_w can be written as expectation $F_w = \mathbb{E}[\|F_w(x) - y\|^2]$, where the randomness comes by uniformly choosing the pair (x, y) from the training set. Hence, a stochastic gradient algorithm first choses a pair (x, y) uniformly and then computes the gradient $\nabla_w \|F_w(x) - y\|^2$. In what follows we show all partial derivatives $\frac{\partial}{\partial w_{i,j}^L}$ of the gradient are indeed computed by the δ -rule.

Output layer: To compute the partial derivative with respect to the final weights $w_{i,j}^L$ we proceed as follows. Writing the norm and the definition of the neural network yields

$$\begin{aligned} \frac{\partial}{\partial w_{i,j}^L} \frac{1}{2} \|F_w(x) - y\|^2 &= \frac{\partial}{\partial w_{i,j}^L} \sum_{s=1}^{N_L} \frac{1}{2} \left(\Phi \left(\sum_{t=1}^{N_{L-1}} w_{t,s}^L V_t^{L-1} \right) - y_s \right)^2 \\ &= \frac{\partial}{\partial w_{i,j}^L} \frac{1}{2} \left(\Phi \left(\sum_{t=1}^{N_{L-1}} w_{t,j}^L V_t^{L-1} \right) - y_j \right)^2 \\ &= \left(\Phi \left(\sum_{t=1}^{N_{L-1}} w_{t,j}^L V_t^{L-1} \right) - y_j \right) \Phi'(h_j^L) \frac{\partial}{\partial w_{i,j}^L} \sum_{t=1}^{N_{L-1}} w_{t,j}^L V_t^{L-1} \\ &= (V_j^L - y_j) \Phi'(h_j^L) V_i^{L-1} \\ &=: \delta_j^L V_i^{L-1}. \end{aligned}$$

There are two crucial points involved in the second and fourth equality. For the second equality we use that only the j th summand depends on $w_{i,j}^L$ (compare the graphical representation of

¹⁶bildchen noch malen

¹⁷put drawing

the network) so that all other derivatives equal zero. Similarly, for the fourth equality only the summand $t = i$ depends on $w_{i,j}^L$ so that all other derivatives vanish and the t th summand simply has the derivative V_i^{L-1} .

Last hidden layer: To compute the partial derivative with respect to the final weights $w_{i,j}^{L-1}$ we proceed similarly. Writing the norm and the definition of the neural network yields

$$\begin{aligned} & \frac{\partial}{\partial w_{i,j}^{L-1}} \frac{1}{2} \|F_w(x) - y\|^2 \\ &= \frac{\partial}{\partial w_{i,j}^{L-1}} \sum_{s=1}^{N_L} \frac{1}{2} \left(\Phi \left(\sum_{t=1}^{N_{L-1}} w_{t,s}^L \sigma \left(\sum_{r=1}^{N_{L-2}} w_{r,t}^{L-1} V_r^{L-2} \right) \right) - y_s \right)^2 \\ &= \sum_{s=1}^{N_L} (V_s^L - y_s) \Phi'(h_s^L) \frac{\partial}{\partial w_{i,j}^{L-1}} \sum_{t=1}^{N_{L-1}} w_{t,s}^L \sigma \left(\sum_{r=1}^{N_{L-2}} w_{r,t}^{L-1} V_r^{L-2} \right) \\ &= \sum_{s=1}^{N_L} (V_s^L - y_s) \Phi'(h_s^L) \sum_{t=1}^{N_{L-1}} w_{t,s}^L \sigma' \left(\sum_{r=1}^{N_{L-2}} w_{r,t}^{L-1} V_r^{L-2} \right) \frac{\partial}{\partial w_{i,j}^{L-1}} \sum_{r=1}^{N_{L-2}} w_{r,t}^{L-1} V_r^{L-2} \\ &= \sum_{s=1}^{N_L} (V_s^L - y_s) \Phi'(h_s^L) \sum_{t=1}^{N_{L-1}} w_{t,s}^L \sigma'(h_j^{L-2}) \frac{\partial}{\partial w_{i,j}^{L-1}} \sum_{r=1}^{N_{L-2}} w_{r,t}^{L-1} V_r^{L-2}. \end{aligned}$$

All derivatives of the last summand disappear, except for $r = i$ and $t = j$. The remaining derivative equals 1, thus, the expression simplifies to

$$\begin{aligned} \frac{\partial}{\partial w_{i,j}^{L-1}} \frac{1}{2} \|F_w(x) - y\|^2 &= V_i^{L-2} \sigma'(h_j^{L-2}) \sum_{s=1}^{N_L} (V_s^L - y_s) \Phi'(h_s^L) w_{j,s}^L \\ &= V_i^{L-2} \sigma'(h_j^{L-2}) \sum_{s=1}^{N_L} \delta_j^L w_{j,s}^L \\ &=: V_i^{L-2} \delta_j^{L-1}. \end{aligned}$$

Well, anyone who likes playing with indices is warmly invited to turn this into a clean induction :).¹⁸ □

Since the derivatives of σ repeatedly appear it is clearly desirable to choose activation functions that are easy to differentiate. The logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ is a typical example with a reasonable derivative $\sigma'(x) = \frac{e^x}{(1+e^x)^2}$.



Go through the previous arguments to check that the gradient $\nabla_w F_w(x)$ is computed analogously with $\delta_j^L = \Phi'(h_j^L)$ and $\delta_i^l = \sigma'(h_i^{l-1}) \sum_{j=1}^{N_l} w_{i,j}^l \delta_j^l$.

5.3.4 Using neural networks to parametrise policies

The success of neural networks in supervised learning was not overseen in the reinforcement learning community where neural networks are used in all branches with increasing success. We give a very brief overview of the use of neural networks for policy gradient schemes but also Q -learning. In later sections we will see how to combine both (actor-critic) and how to proceed much further in deep Q -learning.

¹⁸write induction

Algorithm 33: δ -method to train a neural network

Data: labeled training set (x_i, y_i)
Result: weights w to minimise training error $\sum_i \|F_w(x_i) - y_i\|^2$
Set $n = 0$.
Initialise all weights (for instance iid Gaussian).
while *not converged* **do**
 Chose (x, y) uniformly from the labeled training set.
 Forwards propagation: Starting from the input x compute forwards through the network $V_i^0 := x_i$ and

- $h_i^l := \sum_{k=1}^{N_{l-1}} w_{k,l}^l V_k^{l-1}$, $i = 1, \dots, N_l$,
- $V_i^l := \sigma(h_i^l)$, $i = 1, \dots, N_l$,

 for $l < L$ and

- $h_i^L := \sum_{k=1}^{N_L} w_{k,l}^L V_k^{L-1}$
- $V_i^L := \Phi(h_i^L)$

Back propagation:

- compute $\delta_j^L = (V_j^L - y_j)\Phi'(h_j^L)$.
- compute $\delta_i^l = \sigma'(h_j^{l-1}) \sum_{j=1}^{N_{l+1}} w_{i,j}^{l+1} \delta_j^{l+1}$

 Chose a step-size α .
 Update all weights as $w_{i,j}^l = w_{i,j}^l + \alpha \delta_j^l V_i^l$.
end

Linear softmax with neural networks representations:

First recall the simplest parametrisation in the tabular setting. The tabular softmax policy was defined as $\pi^\theta(a; s) = \frac{e^{\theta_{s,a}}}{\sum_{a'} e^{\theta_{s,a'}}$ with a single parameter $\theta_{s,a}$ tuning the probability of action a in state s . Since this is only reasonable for small state-action spaces the policy has no practical relevance. More relevant are linear softmax policies of the form

$$\pi^\theta(a; s) = \frac{e^{\theta^T \cdot \Phi(s,a)}}{\sum_{a'} e^{\theta^T \Phi(s,a')}} , \quad \theta \in \mathbb{R}^d ,$$

or

$$\pi^\theta(a; s) = \frac{e^{\theta_a^T \cdot \Phi(s)}}{\sum_{a'} e^{\theta_{a'}^T \cdot \Phi(s)}} , \quad \theta = (\theta_{a_1}, \dots, \theta_{a_k}) \in \mathbb{R}^{ad} ,$$

where Φ is a representation (feature vector) of the state (resp. state-action pair). In order to use such policies a reasonable representation Φ must be obtained. One way is to work with representations Φ obtained in supervised learning. For a concrete example let us think of learning to play Atari games. The state-space is large, it consists of all possible Atari frames of 210x160 pixel. Without any prior knowledge the state-space is 210×160 dimensional. Using neural networks trained on large libraries of images (such as imageNet) there are large neural networks that give a reasonably large representation of the states, say 1000-dimensional. The function Φ takes an input Atari frame and returns the last hidden layer of the neural network. Essentially, the precise structure of the state s is replaced by the essential information $\Phi(s)$. Using the linear softmax policy we could now run a policy gradient algorithm with the linear softmax policies, which is a 1000-dimensional stochastic gradient algorithm.

Direct use of neural network

The linear softmax has the advantage to provide a the simple and explicit score function $\Phi(s, a) = \sum_{a'} \pi^\theta(a'; s) \Phi(s, a')$ which is useful to compute the policy gradient. A neural network can also be used directly to write down a parametrised policy $\pi^\theta(s, a)$, where θ is the vector of all weights. There are two approaches with the same disadvantage: the parameter vector θ is the entire weight vector of the neural network, which, for many applications, will be huge.

State as input, action probabilities as output: Thinking of Atari games (many states, few actions) once more here is a visualisation of the idea. An image is fed into a neural network, the output vector are the probabilities $\pi(a_1; s), \dots, \pi(a_k; s)$. In that case a softmax function transforms the final hidden layer into probabilities.

State-action as input, action probability as output: Alternatively, one might feed a state-action pair into a neural network and obtain a one-dimensional output $\pi(a; s)$.

Combining neural networks

Suppose a state-representation function Φ is given (often by a neural network). Then concatenating Φ with an additional neural leads to a new directly parametrised policy $\bar{\pi}^\theta(a; s) = \pi^\theta \circ \Phi(s)$. The advantage can be that the neural network to be optimised in the policy gradient has much less parameters as the input dimension is only the dimension of the feature vector.

5.4 Reinforcement learning with (deep) function approximation

Rethinking the plain vanilla REINFORCE algorithm it comes as no surprise that more advanced algorithms have been developed. REINFORCE is nothing but stochastic gradient descent with non-iid data using the policy gradient theorem to evaluate the expectations. Since stochastic gradient descent is known to be slow and for reinforcement learning we are interested in very large problems there is generally not much hope that REINFORCE works in practice. Indeed, it doesn't but combined with further idea it does. In this section we will discuss some of the most important ideas for practical use of REINFORCE.

5.4.1 Simple actor-critic (AC) and advantage actor-critic (A2C)

So far we have discussed two classes of approaches to solve optimal control problems:

- Policy-based algorithms that learn the optimal policy using gradient descent. A possible drawback of such methods is that the gradient estimators may have a large variance, no bootstrapping of old values is used in the gradient updates.
- Value-based algorithms that evaluate the value-function (or Q -function). Such methods are indirect, they do not try to optimize directly over a set of policies.

Algorithms that directly deal with policies are called actor-only methods while algorithms that improve by criticising the outcome (value-function) of a policy that someone else obtained are called critic-only algorithms. Mixing both approaches is called actor-critic. The typical approach is as follows. Using the policy gradient theorems to perform the REINFORCE algorithm incorporates the Q -functions $Q^{\pi^{\theta_n}}$ or, since the Q -function is typically unknown, estimates of the Q -functions (for instance rewards to go) that introduce variance the the gradient update. An alternative approach is to fix a (simpler) parametrised family ($Q_w^{\pi^{\theta_n}}$) to approximate $Q^{\pi^{\theta_n}}$. Usually the critic's family ($Q_w^{\pi^{\theta_n}}$) is parametrised by a neural network and w is the vector of weights. Actor-critic than works by alternating actor and critic ^{19,20}:

¹⁹R. Sutton, D. McAllester, S. Singh, Y. Mansour: "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NIPS 1999

²⁰V. Konda, J. Tsitsiklis: "Actor-Critic Algorithms", NIPS 1999

- critic step: approximate $Q^{\pi^{\theta_n}}$ by some $Q_w^{\pi^{\theta_n}}$,
- actor step: perform gradient update using the policy gradient theorem with $Q_w^{\pi^{\theta_n}}$ instead of $Q^{\pi^{\theta_n}}$.

Compared to the direct REINFORCE algorithm the actor-critic approach has advantages and disadvantages. On the plus side actor-critic algorithms can reduce the variance of the policy gradient by using the critic's value function as a baseline. Less variance means less steps for convergence. Next, actor-critic algorithms can incorporate ideas from temporal-difference learning to force bootstrapping of samples. Actor-critic algorithms also have some disadvantages compared to plain vanilla policy gradient algorithms. For instance, they introduce a trade-off between bias and variance, as the approximation in the critic step introduces a bias. Additionally, the approximation increases complexity and computational cost for the algorithm as they require the actor steps and typically the training of a neural network for the parametrisation of the critic's value function. We will next show a result that shows that actor-critic with linear function approximations theoretically converges but using non-linear function approximations (such as neural network functions) there is no theoretical justification for convergence and in practice it is a delicate matter to make such an algorithm converge. Let us start with the plain vanilla actor-critic in a very theoretical setup of linear function approximation. ²¹



Proposition 5.4.1. If \mathbf{Q}^{π^θ} satisfies the compatibility with the policy (score function)

$$\nabla_w \mathbf{Q}_w^{\pi^\theta}(s, a) = \nabla_\theta \log(\pi^\theta(s, a))$$

and the approximation property

$$\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}_s} d^{\pi^\theta}(s) \pi^\theta(a; s) (Q^{\pi^\theta}(s, a) - \mathbf{Q}_w^{\pi^\theta}(s, a)) \nabla_w \mathbf{Q}_w^{\pi^\theta}(s, a) = 0,$$

then

$$\nabla J_s(\theta) = \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s') \nabla \log(\pi^\theta(a; s')) \mathbf{Q}_w^{\pi^\theta}(s', a).$$

²² The exact same formulas also hold with the other policy gradient representations from Section 5.2.2.

Proof. By the chain rule the first condition is equivalent to $\nabla_w \mathbf{Q}_w^{\pi^\theta}(s, a) \pi^\theta(s, a) = \nabla_\theta \pi^\theta(s, a)$. Plugging-into the second yields

$$\sum_{s' \in \mathcal{S}} d^{\pi^\theta}(s') \sum_{a \in \mathcal{A}_{s'}} \nabla_\theta \pi^\theta(a; s') (Q^{\pi^\theta}(s', a) - \mathbf{Q}_w^{\pi^\theta}(s', a)) = 0$$

Using the policy gradient theorem (version of Theorem 5.2.6) and the above yields the claim:

$$\begin{aligned} \nabla J_s(\theta) &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s') \nabla \log(\pi^\theta(a; s')) Q^{\pi^\theta}(s', a) \\ &\quad - \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s') \nabla_\theta \log(\pi^\theta(s, a)) (Q^{\pi^\theta}(s', a) - \mathbf{Q}_w^{\pi^\theta}(s', a)) \\ &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s') \nabla \log(\pi^\theta(a; s')) \mathbf{Q}_w^{\pi^\theta}(s', a) \end{aligned}$$

□

²¹ ueberall μ in gradient einarbeiten, auch fuer d^π

²² Bedeutung beider Eigenschaften ausfuehren

On first sight the theorem does not look useful at all as the conditions on \mathbf{Q} are very special. Here are two examples that show that the assumptions are not completely artificial. They are satisfied for linear function approximations.

Example 5.4.2. Recall the linear softmax policy

$$\pi^\theta(a; s) = \frac{e^{\theta^T \cdot \Phi(s, a)}}{\sum_{a' \in \mathcal{A}} e^{\theta^T \cdot \Phi(s, a')}}.$$

with feature vectors $\Phi(s, a)$ for the state-action pairs. The linear softmax has score function

$$\Phi(s, a) - \sum_{a'} \pi^\theta(a'; s) \Phi(s, a').$$

thus, requiring linear function approximation

$$\mathbf{Q}_w^{\pi^\theta}(s, a) = w^T \cdot \left(\Phi(s, a) - \sum_{a'} \pi^\theta(a'; s) \Phi(s, a') \right).$$

What does it mean? \mathbf{Q} must be a linear function with the same features as the policy, except normalized to be mean zero for each state.

Example 5.4.3. A similar example can be obtained for a continuous action space. Suppose π^θ is a linear Gaussian policy, i.e.

$$\pi^\theta(a; s) \sim \mathcal{N}(\theta^T \cdot \Phi(s), \sigma^2)$$

for feature vectors $\Phi(s)$ and a constant σ . For the compatibility we first need to compute the score function:

$$\nabla \log \pi^\theta(s, a) = \nabla_\theta \frac{-(\theta^T \cdot \Phi(s))^2}{2\sigma^2} = \frac{\theta^T \cdot \Phi(s)}{\sigma^2} \Phi(s)$$

For f to satisfy the compatibility, as in the previous example \mathbf{Q} must be linear:

$$\mathbf{Q}_w^{\pi^\theta}(s, a) = w^T \cdot \frac{\theta^T \cdot \Phi(s)}{\sigma^2} \Phi(s).$$

Taking account to the new representation of the policy gradient here is a first version of an actor-critic algorithm with provable convergence. We call the algorithm theoretical actor critic

Algorithm 34: Theoretical simple actor-critic algorithm

Data: Initial parameter θ_0 and approximation class $\mathbf{Q}_w^{\pi^\theta}$.

Result: Approximate policy $\pi^\theta \approx \pi^{\theta^*}$

Set $n = 0$.

while not converged do

Policy evaluation of critic: find a critical point w of the weighted approximation error, i.e.

$$\nabla_w \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}_s} d^{\pi^{\theta_n}}(s) \pi^{\theta_n}(a; s) (Q^{\pi^{\theta_n}}(s, a) - \mathbf{Q}_w^{\pi^\theta})^2 = 0.$$

Policy improvement of actor: chose step-size α and set

$$\theta_{n+1} = \theta_n + \alpha \frac{1}{1-\gamma} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}_s} d^{\pi^{\theta_n}}(s) \pi^{\theta_n} \nabla \log(\pi^{\theta_n}(a; s')) \mathbf{Q}_w^{\pi^\theta}(s, a).$$

end

as the connection of critic computation and actor update is nothing else then a regular update of the policy gradient algorithm (Proposition 5.4.1). Thus, whenever the exact gradient ascent algorithm converges to a maximum (or a stationary point) the simple actor-critic will do the same. If implemented there are a couple of approximations entering the scene:

- functions Q_w will be used that do not satisfy the needed conditions,
- the approximation of Q^{π^θ} will produce an approximation error in every round,
- the gradient will be approximated as in the REINFORCE algorithm replacing the Q -function by the estimated Q -function $Q_w^{\pi^\theta}$.

Each approximation will generate an error and a priori it seems difficult to make the algorithm converge.

In our previous discussions of baselines for policy gradients a good guess for the baseline seemed to be the value function as it reinforces actions that are better than the average and negatively reinforces actions that are worse than the mean. So far we did not follow upon the idea as the value function V^π is unknown, so how could it serve as a baseline of an implementable algorithm? Let's ignore this fundamental issue for a moment.



Definition 5.4.4. The advantage function of a policy is defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

The advantage function measures the advantage (or disadvantage) of first playing action a . Recalling that baselines can be state-dependent the policy gradient can be written as

$$\begin{aligned} \nabla J_s(\theta) &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s) \nabla \log(\pi^\theta(a; s')) (Q^{\pi^\theta}(s', a) - V^{\pi^\theta}(s')) \\ &= \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} d^{\pi^\theta}(s') \pi^\theta(a; s) \nabla \log(\pi^\theta(a; s')) A^{\pi^\theta}(s', a). \end{aligned}$$

In what follows we will discuss what is known as A2C (advantage-actor-critic) algorithms in which similarly to the approximation of Q by linear functions the advantage function is approximated. All A2C algorithms consist of two alternating steps:

- Use the current policy π^θ to produce samples from which an approximation \hat{A}^{π^θ} of the unknown A^{π^θ} is derived.
- Plug \hat{A}^{π^θ} into the policy gradient formula to obtain an estimate of the gradient which is used to update θ .

There are plenty of approaches to turn this into an algorithm. All of them are use ideas from temporal difference learning. The simplest approach is as follows. As for 1-step temporal difference we write the advantage function as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) = r(s, a) + V^\pi(s') - V^\pi(s), \quad s' \sim p(\cdot; s, a).$$

This shows that only the value function needs to be estimated. Now suppose there is a parametric family $V_w : \mathcal{S} \rightarrow \mathbb{R}$ that we intend to use to approximate all value functions V^{π^θ} . Typically, V_w will be given by a neural network with weight vector w . So how do we fit V_w to V^{π^θ} for some policy π^θ ? We approximate $V^{\pi^\theta}(s)$ using samples (Monte Carlo) and then minimise the error. More precisely, suppose y_1, \dots, y_n are the discounted rewards of n rollouts under policy π^θ , then we solve the regression problem

$$\min_w \sum_{i=1}^n \|V_w(s_0^i) - y_i\|^2.$$

If V_w is given by neural networks the regression problem is solved (approximatively) by back propagation. The solution V_w is then used to give an estimator \hat{A}^{π^θ} of A^{π^θ} from the gradient step is computed.

More advanced versions use n -step or TD(λ) approximations for V^{π^θ} to bootstrap samples. It is far from obvious if any of these versions converges to an optimal policy. ²³

²³bespreche sampling aus d^π durch stoppen nach geometrischer zeit. und bedeutung von $d^\pi \pi$

Algorithm 35: A2C with Monte Carlo approximation

Data: Initial parameter θ and approximation functions V_w **Result:** Approximate policy $\pi^\theta \approx \pi^{\theta^*}$ **while** *not converged* **do** Sample N starting points s_0^i from d^{π^θ} and using policy π^θ rollouts (s_0^i, a_0^i, \dots) . minimise $\sum_i \|V_w(s_0^i) - \sum_{t=0}^{\infty} \gamma^t R(s_{t+1}^i, a_{t+1}^i)\|^2$ over w . set $\hat{A}(s_0^i, a_0^i) = R(s_0^i, a_0^i) + \hat{V}_w^\pi(s_1^i) - \hat{V}_w^\theta(s_0^i)$. set $\hat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \frac{1}{1-\gamma} \nabla \log(\pi^\theta(a_0^i; s_0^i)) \hat{A}(s_0^i, a_0^i)$. update $\theta = \theta + \alpha \nabla \pi^\theta(a; s')$.**end**

5.4.2 Soft actor-critic (SAC)**5.4.3 Proximal policy optimisation (PPO)**

Chapter 6

Deep Q -learning

Chapter 7

Monte Carlo Tree Search