The Mathematics of Reinforcement Learning

Leif Döring

University of Mannheim

Contents

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 on algorithmic ideas towards the trade-off between exploration and exploitation in optimal decision making.

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

From a probabilistic point of view reinforcement learning will always be about stochastic two-stage experiments. First choose an action (first experiment) and given that action observe a reward (second experiment). It's a bit of a mathematical overkill but let us quickly discuss the basics from two-stage experiments. If all experiments involved are discrete (i.e. take finitely or countably infinite values) then not much is needed and all computations work with the usual conditional probabilities defined by $\mathbb{P}(X = x|Y = y) = \frac{\mathbb{P}(X=x,Y=y)}{\mathbb{P}(Y=y)}$. The choice of actions indeed is discrete in most applications, unfortunately, the rewards often are not. Going towards non-discrete probability the concept of regular conditional expectation is needed. We are not going into detail and only summarize the most important facts. First, suppose (X, Y) is a pair of discrete random variables (or random vectors) on some probability space $(\Omega, \mathcal{A}, \mathbb{P})$, then rewriting the definition of conditional probabilities gives

$$\mathbb{P}(X = x, Y = y) = \underbrace{\mathbb{P}(X = x|Y = y)}_{\text{second step, given result of first step}} \cdot \underbrace{\mathbb{P}(Y = y)}_{\text{first step}}$$

so that all quantities jointly involving X and Y can be computed by knowning the first step and the second step given the first step. For instance,

$$\begin{aligned}\mathbb{E}[h(X,Y)] &= \sum_{x,y} h(x,y)\mathbb{P}(X=x|Y=y)\mathbb{P}(Y=y),\\ \mathbb{E}[h(X,Y)|Y=y] &= \sum_{x} h(x,y)\mathbb{P}(X=x|Y=y),\\ \mathbb{E}[h(X,Y)|Y] &= \sum_{x,y} h(x,y)\mathbb{P}(X=x|Y=y)\mathbf{1}_{Y=y},\\ \mathbb{P}(X\in\cdot|Y) &= \sum_{y} \mathbb{P}(X\in\cdot|Y=y)\mathbf{1}_{Y=y}.\end{aligned}$$

The situation is more delicate if Y is not discrete because $\mathbb{P}(X=x|Y=y)$ cannot be defined directly as the definition of conditional probabilities would require devision by 0 if $\mathbb{P}(Y=y)=0$. Nonetheless, there is a way to define a unique Markov kernel $k(\cdot,\cdot)$ - a Markov kernel is a measure in the second coordinate and a measurable mapping in the first - such that the rules

$$\mathbb{E}[h(X,Y)|Y=y] = \int h(x,y)k(y,\,\mathrm{d}x) \quad \text{and} \quad \mathbb{E}[h(X,Y)|Y] = \int h(x,Y)k(Y,\,\mathrm{d}x)$$

hold. The kernel κ is called a conditional regular expectation of X given Y. In the discrete case it holds that $k(y,A)=\mathbb{P}(X\in A|Y=y)$ according to the usual definition. For absolutely continuous pairs (X,Y) the measure $k(y,\cdot)$ has density $k(y,x)=f_{(X,Y)}(x,y)/f_X(x)f_Y(y)$ but in general k is abstract. Nonetheless, to keep the analogy to the discrete setting one typically writes $\mathbb{P}(X\in A|Y=y)$ instead of $k(y,A)$ even if the conditional probability cannot be definded in the usual way. We also use the notation $\mathbb{P}(X\in A|Y)=\kappa(Y,A)$ from which it follows that $\mathbb{E}[h(X,Y)|Y]=\int h(x,Y)\mathbb{P}(X\in \mathrm{d}x|Y)$. For later purposes it is crucial to know that for independent X and Y it holds that $k(y,A)=\mathbb{P}(X\in A)$ so that $\mathbb{E}[h(X,Y)|Y]=\int h(x,Y)\mathbb{P}(X\in \mathrm{d}x)$.

1.2 Introduction to stochastic bandits

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 cantine. Let's say the cantine 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 experiements 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.

The basic mathematical model behind a stochastic bandits is as follows. There are distributions $P_{a_1}, ..., P_{a_K}$ (not necessarily discrete) from which an outcome is sampled if an actor decides to play a so-called arm a. The outcome is called X. If the choice of the actor is modeled using a random variable A taking values $a_1, ..., a_K$ then the two-stage experiment yields

$$\mathbb{P}(X \in \cdot | A = a) = P_a(\cdot) \quad \text{and} \quad \mathbb{P}(X \in \cdot | A) = \sum_{a \in \mathcal{A}} P_a(\cdot) \mathbf{1}_{Y=a} =: P_A(\cdot)$$

so that

$$\mathbb{P}(X \in \cdot, A = a) = \mathbb{P}(X \in \cdot | A = a)\mathbb{P}(A = a) = P_a(\cdot)\mathbb{P}(A = a).$$

In fact, a bandit model will depend not only on one action but on reward/action pairs of the past. This makes the mathematical framing a bit more tedious as rewards must not be discrete.

Before we go into more details let us discuss the optimization goal. Suppose we have a finite set $\mathcal{A} = \{a_1, ..., a_K\}$ 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 $(\pi_t)_{t=1,...,n}$ 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. $\pi_t(\text{vegan}) = \cdots = \pi_t(\text{oriental}) = \frac{1}{4}$, 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 with finite expectations, called the reward distributions.

- The set ν is called a stochastic bandit model. In these lectures we will always assume $\mathcal{A} = \{a_1, ..., a_K\}$ is finite, K is the number of arms. Often it will be useful to denote the arms by $1, ..., K$ to simplify formulas.
- The action value (or Q-value) of an arm is defined by the expectation $Q_a := \int_{\mathbb{R}} x \, dP_a(x)$. A best arm, usually denoted by a_*, is an arm with highest Q-value, i.e.
$$a^* = \text{argmax}_{a \in \mathcal{A}} Q_a.$$
Typically one abbreviates Q_* for the largest action value Q_{a^*} and if there are several optimal arms the argmax chooses 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 $(\pi_t)_{t=2,...,n}$ of kernels on $\Omega_{t-1} \times \mathcal{A}$,

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

$$\pi_t\big(\cdot\,;\, a_1, x_1, a_2, x_2, ..., a_{t-1}, x_{t-1}\big)$$

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

Recall that a 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). In principle the learning strategy should be written down, most of the time we will construct the kernels π_t round by round using algorithms.

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 $(\pi_t)_{t=1,...,n}$ a learning strategy for n rounds. Then a stochastic process $(A_t^\pi, X_t^\pi)_{t=1,...,n}$ 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

- $\mathbb{P}\big(A_t^\pi = a | A_1^\pi, X_1^\pi, ..., A_{t-1}^\pi, X_{t-1}^\pi\big) = \pi_t\big(a\,;\, A_1^\pi, X_1^\pi, ..., A_{t-1}^\pi, X_{t-1}\big)$,
- $\mathbb{P}\big(X_t^\pi \in B | A_1^\pi, X_1^\pi, ..., A_t^\pi\big) = P_{A_t^\pi}(B) := \sum_{a \in \mathcal{A}} P_a(B) \mathbf{1}_{A_t^\pi = a}$

for all $t = 1, ..., n$. 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. For notational convenience the superscripts π will typically 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 properties 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 (A^π, X^π).

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	$\cdots$
$\mathbf{X_1^{(a_1)}}$	$X_2^{(a_1)}$	$X_3^{(a_1)}$	$\mathbf{X_4^{(a_1)}}$	$\cdots$
$X_1^{(a_2)}$	$X_2^{(a_2)}$	$\mathbf{X_3^{(a_2)}}$	$X_4^{(a_2)}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$X_1^{(a_K)}$	$\mathbf{X_2^{(a_K)}}$	$X_3^{(a_K)}$	$X_4^{(a_K)}$	$\cdots$

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

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 $X_1 := X_1^{(A_1)}$.
- $t \mapsto t+1$: Use U_{t+1} to sample from the discrete measure $\pi(\cdot\,;\, A_1, X_1, ..., A_t, X_t)$ (relying only on the table to the left of column $t+1$) an arm a, denote this arm by A_t and set $X_{t+1} = X_{t+1}^{(A_{t+1})}$.

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 chosing 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 $I_t^a(a_1, ..., x_{t-1})$ a partition of $[0,1]$ into K disjoint intervals with lengths $\pi_t(a\,;\, a_1, ..., x_{t-1})$. Then, using $h(u, a_1, ..., x_{t-1}) := \mathbf{1}_{u\in I_t^a(a_1,\dots,x_{t-1})}$ and

$$\kappa(a_1, ..., x_{t-1}, \cdot) = \mathbb{P}(U_t \in \cdot | A_1 = a_1, ..., X_{t-1} = x_{t-1}) \overset{\text{ind.}}{=} \mathbb{P}(U_t \in \cdot) \overset{\mathcal{U}([0,1])}{=} \lambda(\cdot)$$

yields

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

The second property is derived as follows:

$$\begin{aligned}\mathbb{P}\big(X_t \in B | A_1, X_1, ..., A_t\big) &= \sum_{a \in \mathcal{A}} \mathbb{P}\big(X_t \in B, A_t = a | A_1, X_1, ..., A_t\big) \\ &= \sum_{a \in \mathcal{A}} \mathbb{P}\big(X_t^{(a)} \in B, A_t = a | A_1, X_1, ..., A_t\big) \\ &= \sum_{a \in \mathcal{A}} \mathbb{E}\big[\mathbf{1}_{X_t^{(a)} \in B} \mathbf{1}_{A_t = a} | A_1, X_1, ..., A_t\big] \\ &\overset{\text{meas.}}{=} \sum_{a \in \mathcal{A}} \mathbf{1}_{A_t = a} \mathbb{E}\big[\mathbf{1}_{X_t^{(a)} \in B} | A_1, X_1, ..., A_t\big] \\ &\overset{\text{ind.}}{=} \sum_{a \in \mathcal{A}} \mathbf{1}_{A_t = a} \mathbb{E}\big[\mathbf{1}_{X_t^{(a)} \in B}\big] \\ &= \sum_{a \in \mathcal{A}} \mathbb{P}\big(X_t^{(a)} \in B\big) \mathbf{1}_{A_t = a} \\ &\overset{\text{Def.}}{=} P_{A_t}(B).\end{aligned}$$

□

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 ath arm is played for the nth time then the reward variable $X_n^{(a)}$ is used instead of the reward variable corresponding to the time at which the ath arm is played for the nth 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 ath 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

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

- 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 experimantal design in medicin the AI community is more focused on artificial decision making (of computers). While both aim at developping 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 two typical goals that we will formalise in due course:

(A) 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 $\mathbb{E}[\sum_{k=1}^n X_k^\pi]$ is maximised.

(B) 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 chosing 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 more typical in AI. The aim of these lecture notes is to introduce reinforcement learning, so why bother with questions from stochatic 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 appraoch (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 Q_*. Hence, if all Q_a would be known in advance then the stochastic bandit optimization problem would be trivial, just choose 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 (cummulated) regret is defined by

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

The stochastic bandit problem consists in finding learning strategies that minimise the regret. Algorithms are only allowed to use samples of the reward distribution. 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 aquainted 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 $\limsup_{n\to\infty} \frac{R_n(\pi)}{n} < \infty$.
- Let $R_n(\pi) = 0$. Prove that π only chooses best arms. If there is only one best arm, then π is deterministic, i.e. $\Pi_t(a^*) = 1$ for all t.

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

$$R_n(\pi) = nQ_* - n\mathbb{E}[X_1^\pi] = n\Big(\underbrace{Q_* - \sum_{k=1}^{K} \frac{1}{K} Q_a}_{\geq 0}\Big).$$

Thus, a linearly incraesing 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, see Section 1.4. 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 chosing 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 perspect 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, wheras 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 expecations 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:

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

To compute the right hand side note that

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

Here we used the general fact $\mathbb{E}[X|Y] = \int x\mathbb{P}(X \in \mathrm{d}x|Y)$ and that $\mathbb{P}(X_t \in \cdot|A_1, X_1, ..., A_t) \sim P_{A_t} = \sum_a P_a \mathbf{1}_{A_t=a}$. Using the tower property a combination of both computations yields

$$R_n = \sum_{t\le n}\sum_{a\in\mathcal{A}} \mathbb{E}\big[\mathbb{E}[(Q_* - X_t)\mathbf{1}_{A_t=a}|A_1, X_1, ..., A_t]\big] = \sum_{a\in\mathcal{A}} \Delta_a \mathbb{E}\Big[\underbrace{\sum_{t\le n} \mathbf{1}_{A_t=a}}_{T_a(n)}\Big].$$

□

The statistical regret analysis for many bandit algorithm follows the same appraoch, 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 analogie 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 $\mathbb{E}[T_n(a)] = \sum_{t=1}^n \mathbb{E}[\mathbf{1}_{A_t=a}] = \sum_{t=1}^n \mathbb{P}(A_t = a)$ the recret decomposition lemma can be reformulated as follows:

Lemma 1.2.10.

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

and, in particular,

$$R_n(\pi) \leq \max_{a \in \mathcal{A}} \Delta_a \sum_{t=1}^n \tau_t(\pi) \quad \text{and} \quad R_n(\pi) \geq \min_{a \neq a^*} \Delta_a \sum_{t=1}^n \tau_t(\pi).$$

As a consequence we see that the study of regret and failure probability is ultimately connected. If we interprete 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 probabilies 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 exloitation). 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 $\frac{1}{n}\sum_{t=1}^{n} Y_t \to \mathbb{E}[Y_1]$ 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

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

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 as long as all arms are played infinitely often the limit $\lim_{t\to\infty} \hat{Q}_a(t)$ is nothing but $\lim_{t\to\infty} \frac{1}{t}\sum_{k=1}^{t} X_k^{(a)}$, 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. The learning strategy π_t is clearly an index strategy and can be written as

$$\pi_t(a; a_1, x_1, ..., x_t) = \begin{cases} 1 & : a = a_{t \bmod K+1} \text{ and } t \le mK \\ 1 & : \operatorname{argmax}_a \hat{Q}_a(mK) \text{ and } t > mK \\ 0 & : \text{otherwise} \end{cases}$$

for arms denoted by $1, ..., K$. As a first example on how to estimate the regret of bandit algorithms

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

Data: m, n, bandit model ν
Result: actions $A_1, ..., A_n$ and rewards $X_1, ..., X_n$
set $\hat{Q}(0) \equiv 0$;
while $t \le n$ **do**
  $A_t := \begin{cases} a_{t \bmod K+1} & : t \le mK \\ \operatorname{argmax}_a \hat{Q}_a(mK) & : t > mK \end{cases}$;
  Obtain reward X_t by playing arm A_t;
end

we prove the following 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-commmit) 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

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

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 indepence probabilitites 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 or 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:

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

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

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

The appearing probability has the particularly nice form

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

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 $\exp(-m\Delta_a^2/(4\sigma^2))$ the proof is complete. In order to do so we first need an excursion to probability theory. □

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 $\mathcal{M}_{X-\mathbb{E}[X]}(\lambda) = \mathbb{E}[e^{\lambda(X-\mathbb{E}[X])}] \leq e^{\frac{\lambda^2\sigma^2}{2}}$ 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 aquainted 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

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

for all $a > 0$.

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:

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

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 $\mathbb{P}(|X| \geq a) = \mathbb{P}(X \geq a \text{ or } X \leq -a) \leq \mathbb{P}(X \geq a) + \mathbb{P}(-X \geq a)$. □

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

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

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. □

We can now combine the exploration 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

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

Plugging-in yields the upper bound from the theorem. □

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 chose 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

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

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

The regret upper bound is minimized by

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

Thus, in terms of regret optimisation, our best guess is the explore-then-committ

algorithm with this particular m. For this choice of m the regret is upper bounded by

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

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-committ strategy with m from (1.1) is reasonable? No, it's cheating. The algorithm relies on n, σ, and Δ through the choice of m.

- The number of rounds n might be fixed in advance for some examples it might be not for other examples. For infinite time-horizon there is a trick called the "doubling-trick" that allows to combine fixed-time algorithms into an infinite time-horizon algorithm so we do accept dependence on n.
- The variance parameter σ might be known in some situations (for instance Gaussian rewards with fixed variance but unknown mean) but will typically be unknown as well. There are algorithms that estimate σ on the run.
- The dependence on $\Delta = Q_{a_1} - Q_{a_2}$ is much more severe as the action values are never known in advance (otherwise we could just chose 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.

It will turn out below in the Lai-Robbins Theorem 1.4.8 that the regret upper bound from (1.2) is close to optimal for large n because of the learning strategy independent lower bound $\liminf_{n\to\infty} \frac{R_n(\pi)}{\log(n)} \geq \frac{\sigma^2}{\Delta}$ at least for Gaussian bandit models. In the next section we will show how to construct a similarly good algorithm without cheating in the choice of the algorithm parameters.

1.3.2 From greedy to UCB

A greedy learning strategy is a strategy that always choses the option the algorithm currently believes to be the best. For bandits this is the arm with the highest believed reward, typically measured in terms of the empirical mean, that is the mean of the already observed rewards $\hat{Q}_a(t-1) = \frac{1}{T_a(t-1)} \sum_{k=1}^{t-1} X_k \mathbf{1}_{A_t=a}$. As similar concepts will reappear in reinforcement learning we will spend some time on this topic. It will turn out that pure greedy algorithms do not work at all, but simple modifications that force additional exploitation work very well.

Purely greedy bandit algorithm

The pure greedy algorithm turns out to be complete non-sense. Nonetheless, it is a good start into the discussion to get aquainted 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 randomly chose one of them. In the algorithm we only use one vector $\hat{Q}$ to store the estimates $\hat{Q}(t)$ as we only need the current estimated action values. The computation of $\hat{Q}$ looks a bit strange and relies on a simple memory trick that allows to

Algorithm 2: Purely greedy bandit algorithm

Data: bandit model ν, vector $\hat{Q}$, n
Result: actions $A_1, ..., A_n$ and rewards $X_1, ..., X_n$
Initialise $T_a = 0$ for all a;
while $t \leq n$ **do**
  Set $A_t = \text{argmax}_a \hat{Q}_a$;
  Obtain reward X_t by playing arm A_t;
  Set $T_{A_t} = T_{A_t} + 1$;
  Set $\hat{Q}_{A_t} = \hat{Q}_{A_t} + \frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t})$;
end

compute successive averages without storing all numbers. Suppose $R_1, ...$ are real numbers and all averages $Q_n = \frac{1}{n}\sum_{k=1}^{n} R_k$ should be computed without storing the rewards R_n forever. To compute Q_n it is actually only needed to know Q_{n-1} and R_n:

$$\begin{aligned} Q_n &= \frac{1}{n}\sum_{k=1}^{n} R_k \\ &= \frac{1}{n}\Big(R_n + \sum_{k=1}^{n-1} R_k\Big) \\ &= \frac{1}{n}\Big(R_n + (n-1)\frac{1}{n-1}\sum_{k=1}^{n-1} R_k\Big) \\ &= \frac{1}{n}\Big(R_n + (n-1)Q_{n-1}\Big) \\ &= Q_{n-1} + \frac{1}{n}\Big(R_n - Q_{n-1}\Big) \end{aligned} \tag{1.3}$$

Under the initialisation $\hat{Q} \equiv 0$ the $\hat{Q}$ are nothing but the sample means and the greedy algorithm plays greedily the arms with the largest empirical mean. We can also think differently about the algorithm. The actor suggests a vector with some initial estimates $\hat{Q}_a$ of the action values. If nothing is known the actor will always choose $\hat{Q} \equiv 0$. Then the actor plays greedily by always playing the arm that is believed to be best. After playing the Q-values are updated by adding $\frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t})$. They are increased (resp. decreased) if the reward was higher (resp. lower) than what was believed before.

An algorithm is called a tabular algorithm if there is a table of real numbers (here a vector) that is constantly updated and used to make the decisions. We will get to know a similar approach as Q-learning in Chapter 3.

The pure greedy algorithm depends extremely on the initialisation of the vector $\hat{Q}$ and the distribution of the arms. Suppose $\hat{Q} \equiv 0$ and 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 $\hat{Q}$. As an example one might start with large $\hat{Q}$. 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 $\hat{Q}$ is chosen too large there would be too much exploitation before playing greedy the arms with the largest estimated action values. If $\hat{Q}$ 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 substracting the same constant from all arms to reduce the committal behavior effect described above. Again, it is unclear what constant should be substracted 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 useles. To get a

Algorithm 3: ε-greedy bandit algorithm

Data: bandit model ν, exploration rate $\varepsilon \in (0,1)$, vector $\hat{Q}$, n
Result: actions $A_1, ..., A_n$ and rewards $X_1, ..., X_n$
while $t \leq n$ **do**
  Initialise $T_a = 0$ for all a;
  Sample $U \sim \mathcal{U}([0,1])$;
  if $U < \varepsilon$ **then**
    [exploration part];
    Uniformly chose an arm A_t;
    Obtain reward X_t by playing arm A_t;
    Set $T_{A_t} = T_{A_t} + 1$;
    Set $\hat{Q}_{A_t} = \hat{Q}_{A_t} + \frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t})$;
  end
  if $U \geq \varepsilon$ **then**
    [greedy part];
    Set $A_t = \text{argmax}_a \hat{Q}_a$;
    Obtain reward X_t by playing arm A_t;
    Set $T_{A_t} = T_{A_t} + 1$;
    Set $\hat{Q}_{A_t} = \hat{Q}_{A_t} + \frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t})$;
  end
end

Lecture 3

feeling for the algorithm you can 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. The following exercise shows that there is no use of the simple greedy learning strategy, it has linear regret even if each arm is explored once.

Let π the learning strategy that first explores each arm once and then continus according to ε-greedy for some $\varepsilon \in (0,1)$ fixed. Show that the regret grows linearly:

$$\lim_{n\to\infty} \frac{R_n(\pi)}{n} = \frac{\varepsilon}{K} \sum_{a \in \mathcal{A}} \Delta_a.$$

Hint: Lower bound $\mathbb{E}[T_a(n)]$ by the random exploration and upper bound $\mathbb{E}[T_a(n)]$ using arguments inspired by the proof of Theorem 1.3.7 below.

There are many versions of ε-greedy with reasonable regret bounds. We will only touch upon an example (introduced an 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 $[0,1]$, $d < \min_{a:Q_a \neq Q_{a_*}} \Delta_a$, and $C > \max\{5d^2, 2\}$. Then the ε-greedy algorithm with decreasing exploration rate $\varepsilon_t = \min\left\{1, \frac{CK}{d^2 t}\right\}$ satisfies

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

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

$$\limsup_{n\to\infty} \frac{R_n(\pi)}{\log(n)} \leq \frac{(K-1)C}{d^2}. \tag{1.4}$$

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 appears squared 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, the algorithm cheats!

Set $K = 2$ and Δ small and then compare the algorithm with the explore-then-committ algorith for two arms.

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

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

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 domminates 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:

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

Suppose $X_1, ..., X_n$ 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, and $\sigma^2 := \sum_{k=1}^n \sigma_k^2$, then

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

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

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

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

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

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

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

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

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

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

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

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

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

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 simpilfy 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,

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

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

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

□

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 - optimism in the face of uncertainty

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 interprete UCB as an extension of greedy. Here is the main idea, optimism in the face of uncertainty. The principle suggests to be more optimistic, more curious, in situations that are less certain. For instance, trying more new dishes when travelling unknown regions of the world. In the context of bandits the principle states to add an exploration bonus for less good estimated action values. In the context of estimations involving the approximate action values $\hat{Q}$ one should thus add an exploration bonus that decreases with T_a because the uncertainty in the estimate $\hat{Q}_a \approx Q_a$ decreases as T_a increases. In fact, the estimated empirical variance should also play a role but for the simple UCB algorithm this is ignored. Following the principle in the greedy algorithm we should add an exploration bonus to the estimated action values that decreases as T_a increases. The bonus should be something of the kind

$$\hat{Q}_a(t) + \frac{\dots}{T_a(t)}.$$

The choice of the exact exploration bonus is critical. For later purposes one typically uses the

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

The exploration bonus is motivated by concentration inequalities, this is the upper bound of confidence intervals for sums of iid random variables, justifying the name UCB. More precisely, for 1-subgaussian random variablies with mean Q, plugging-into Hoeffdings inequality 1.3.5 yields

$$\mathbb{P}\Big(\frac{1}{n}\sum_{k=1}^{n} X_k \geq Q + \sqrt{\frac{2\log(1/\delta)}{n}}\Big) \leq \exp\Big(-\frac{n\big(\sqrt{\frac{2\log(1/\delta)}{n}}\big)^2}{2}\Big) = \delta. \tag{1.5}$$

Thus, the UCB exploration gives a simple control of the overestimation probabilities and with it the exploration caused by the optimism principle. The parameter δ can be chosen and it turns out later that $\delta = \frac{1}{n^2}$ is a good choice of the time-horizon is fixed and known. Instead of writing

Algorithm 4: UCB algorithms with parameter δ

Data: bandit model ν, $\delta \in (0,1)$, n, vector $\hat{Q}$
Result: actions $A_1, ..., A_n$ and rewards $X_1, ..., X_n$
Initialise $T_a = 0$ for all a;
while $t \leq n$ **do**
  $A_t = \text{argmax}_a \text{UCB}_a(t-1, \delta)$;
  Obtain reward X_t by playing arm A_t;
  Set $T_{A_t} = T_{A_t} + 1$;
  Set $\hat{Q}_{A_t}(t) = \hat{Q}_{A_t}(t) + \frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t}(t))$;
end

down the algorithm to run the bandit we could also write down the learning strategy explicitly:

$$\pi_t(a; a_1, x_1, ..., x_t) = \text{argmax}_a \text{UCB}_a(t-1, \delta)$$

with the dependence $\hat{Q}_a(t) = \frac{1}{T_a(t)}\sum_{k=1}^{t} x_k \mathbf{1}_{a_k=a}$ and $T_a(t) = \sum_{k=1}^{t} \mathbf{1}_{a_k=a}$ of the past. 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.

Theorem 1.3.8. Suppose ν is a bandit model with 1-subgaussian arms, $n \in \mathbb{N}$, $\delta = \frac{1}{n^2}$. Then the UCB algorithms initialised with $\hat{Q} \equiv 0$ has the following regret upper bound:

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

Simulations show that the simple UCB algorithms performs very well on most examples. In contrast to the optimal version of ECT (cheating by using reward gaps) there is no need to use reward gaps.

Proof. We will assume without loss of generality that $a_* = a_1$ and estimate the expected times a suboptimal arm a will be played. Since δ is fixed as $\frac{1}{n^2}$ we skip the δ from $UCB(t,\delta)$. Combined with the regret decomposition Lemma 1.2.8 this will give the upper bound on the regret.

Here is the idea of the analysis. Using the regret decomposition it suffices to estimate $\mathbb{E}[T_a(n)]$. A bit similarly to the analysis of ECT we decompose

$$\mathbb{E}[T_a(n)] = \mathbb{E}[T_a(n)\mathbf{1}_{H_m}] + \mathbb{E}[T_a(n)\mathbf{1}_{H_m^c}],$$

where the events H_m should be {arm a is played at most m times}. In that case we can estimate

$$\mathbb{E}[T_a(n)] \leq m\mathbb{P}(H_m) + n\mathbb{P}(H_m^c) \tag{1.6}$$

since $T_a(n)$ can be at most n. If now the probability can be estimated then one can optimise over m. The proof is a bit more delicate, we cannot compute H_m. Instead we use smaller events $G_m \subseteq H_m$ for which the probabilities can be computed. The decomposition (1.6) works equally but the way the G_m are defined their probabilities can be estimated using the independence of all rewards.

The estimates are based on Hoeffding's inequality for sums of iid random variables. This is why we use the random stack construction of the bandit process (A, X). For that sake recall the table $\{X_t^{(a)}\}_{t\le n, a\in\mathcal{A}}$ of independent random variables with $X_t^{(a)} \sim P_a$. Using

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

this means that $\hat{Q}_a(t) = \bar{Q}_s^{(a)}$ if $T_a(t) = s$. From now on we fix arm a and estimate $\mathbb{E}[T_a(n)]$. Define $G_m = G_1 \cap G_{2,m}$ with

$$G_1 = \big\{\omega : Q_{a_1} < \min_{t\le n} \mathrm{UCB}_{a_1}(t)(\omega)\big\},$$
$$G_{2,m} = \left\{\omega : \bar{Q}_m^{(a)}(\omega) + \sqrt{\frac{2\log(1/\delta)}{m}} < Q_{a_1}\right\},$$

with a natural number m to be specified later. Thus, G_m is the event when Q_{a_1} is never underestimated by the upper confidence bound of the first arm, while at the same time the upper confidence bound for the mean of arm a after m observations are taken from this arm is below the action value of the optimal arm. In what follows we will estimate the probability of G_m and show that $G_m \subseteq H_m$ with H_m from above. Let us start with the latter and prove that

$$\text{if } \omega \in G_m, \text{ then } T_a(n)(\omega) \le m. \tag{1.7}$$

First in words: If $\omega \in G_{2,m}$ and $\omega \in G_1$ then the UCB value for the mth round of playing arm a is smaller than that for playing arm a_1 (because one is smaller, the other bigger than Q_{a_1}). Thus, arm a is not played more than m times. Now more formally. Suppose $\omega \in G_m$ but $T_a(n)(\omega) > m$ holds. Then there must be some time index $t \le n$ with $T_a(t-1)(\omega) = m$ and $A_t(\omega) = a$. Using the definitions of $G_1, G_{a,2}$, and UCB yields

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

a contradiction to $a = A_t(\omega) = \mathrm{argmax}_b \mathrm{UCB}_b(t-1)$. Hence, (1.7) holds.

We now follow the idea sketched above. Since $T_a(n)$ is trivially bounded by n the following key estimate holds:

$$\mathbb{E}[T_a(n)] = \mathbb{E}\big[T_a(n)\mathbf{1}_{G_m}\big] + \mathbb{E}\big[T_a(n)\mathbf{1}_{G_m^c}\big] \le m + n\big(\mathbb{P}(G_1^c) + \mathbb{P}(G_{2,m}^c)\big). \tag{1.8}$$

It now suffices to estimate separately both summands on the right hand side of (1.8). First, it

holds that

$$\begin{aligned}\mathbb{P}\big(G_1^c\big) &= \mathbb{P}\big(Q_{a_1} \geq \mathrm{UCB}_{a_1}(t) \text{ for some } t \leq n\big)\\ &\leq \sum_{s\leq n} \mathbb{P}\Big(Q_{a_1} - \sqrt{\frac{2\log(1/\delta)}{s}} \geq \bar{Q}_s(a_1)\Big)\\ &\overset{\text{Hoeffding}}{\leq} \sum_{t\leq n} \delta = n\delta.\end{aligned}$$

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

$$\Delta_a - \sqrt{\frac{2\log(1/\delta)}{m}} \geq \frac{1}{2}\Delta_a \tag{1.9}$$

holds, for instance $m = \left\lceil \frac{2\log(1/\delta)}{\frac{1}{4}\Delta_a^2} \right\rceil$. Then

$$\begin{aligned}\mathbb{P}\big(G_{2,m}^c\big) &= \mathbb{P}\Big(\bar{Q}_m^{(a)} + \sqrt{\frac{2\log(1/\delta)}{m}} \geq Q_{a_1}\Big)\\ &= \mathbb{P}\Big(\bar{Q}_m^{(a)} - Q_a \geq \Delta_a - \sqrt{\frac{2\log(1/\delta)}{m}}\Big)\\ &\overset{(1.9)}{\leq} \mathbb{P}\Big(\bar{Q}_m^{(a)} \geq Q_a + \frac{1}{2}\Delta_a\Big)\\ &\overset{\text{Hoeffding}}{\leq} \exp\Big(-\frac{m\Delta_a^2}{8}\Big).\end{aligned}$$

Combining the above yields

$$\mathbb{E}\big[T_a(n)\big] \leq m + n\Big(n\delta + \exp\Big(-\frac{m\Delta_a^2}{8}\Big)\Big). \tag{1.10}$$

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

$$m = \left\lceil \frac{2\log(n^2)}{\frac{1}{4}\Delta_a^2} \right\rceil \leq 1 + \frac{16\log(n)}{\Delta_a^2}$$

so that

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

Combined with the regret decomposition the claim follows. □

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 emphasie 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. Persuing this way leads to

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

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

$$\mathrm{UCB}_a(t) := \begin{cases} \infty & : T_a(t) = 0 \\ \underbrace{\hat{Q}_a(t)}_{\text{greedy}} + \underbrace{\sqrt{\frac{4\sigma^2\log(n)}{T_a(t)}}}_{\text{exploration bonus}} & : T_a(t) \neq 0 \end{cases} .$$

Check that the regret bound in Theorem 1.3.8 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.

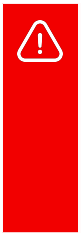
The idea of the last proof is very important. The model based regret bounds are often completely useless as they emphasise too strongly small reward gaps which by means of the regret decomposition should not be important at all. To get a better feeling please think a moment about the following exercise:

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 $C = 8 + \frac{2}{e}$) 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 $n \to \infty$ (as we did in the formulation of Theorem 1.3.7). Keeping in mind logarithmic growht 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.

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 $\mathrm{SM}(\theta, x)$ on finite sets with d elements with probability weights

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

The weights are called Boltzmann weights of the vector x.

Typically, all θ_i are equal, in which case θ is called inverse temperature. The origin of such distributions lies in statistical physics. The softmax distributions is a so-called catagorial distribution, written $\mathrm{Categorial}(p_1, ..., p_d)$ which is also called a multinoulli or generalised Bernoulli distribution with probabilities $p_1, ..., p_d$. For us, the following idea is much more important. If all θ_k are non-negative then sampling from $\mathrm{SM}(\theta, x)$ is somewhat similar to the deterministic distribution M_x that only charges mass on the index $\mathrm{argmax}_k x_k$ because $\mathrm{SM}(\theta, x)$ 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 $\{1, ..., d\}$.

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 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 fact[2], both constant and decreasing choices of

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

Algorithm 5: Simple Boltzmann exploration

Data: bandit model ν, vector $\hat{Q}$, parameter θ
Result: actions $A_1, ..., A_n$ and rewards $X_1, ..., X_n$
Initialise $T_a = 0$ for all a;
while $t \leq n$ **do**
  Sample A_t from $\mathrm{SM}(\theta, \hat{Q})$;
  Obtain reward X_t by playing arm A_t;
  Set $T_{A_t} = T_{A_t} + 1$;
  Set $\hat{Q}_{A_t} = \hat{Q}_{A_t} + \frac{1}{T_{A_t}}(X_t - \hat{Q}_{A_t})$;
end

θ are unfavorible. 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

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

where $g_1, ..., g_d$ 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

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

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

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

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 $X \sim \mathrm{Exp}(1)$, then $Z := -\log(X)$ is Gumbel with scale parameter $\beta = 1$ and mode $\mu = 0$. The claim is checked by computing the

cummulative 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 $g_1, ..., g_k$ are iid Gumbel with scale $\beta = 1$ and mode $\mu = 0$ we finally obtain

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

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

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

$$A_t \sim \text{argmax}_a\big\{\theta\hat{Q}_a(t-1) + g_a\big\} = \text{argmax}_a\big\{\hat{Q}_a(t-1) + \theta^{-1}g_a\big\},$$

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. Since typical values of g are around 1, motivated by the UCB algorithm a first idea should immediately come to mind: θ^{-1} 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-Bianci et al. We will not go into detail here. The aim of this section was to link the Boltzmann explortation 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. Chosing distributions that concentrate around 1 (Dirac measure at 1 gives UCB) might be reasonsble, such as a Gamma distribution with shaper parameter larger than 1 and scale parameter that forces expectation 1.

Lecture 5

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 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 chosing 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 $(0, ..., 1, ...0)$ with 1 for the ath 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.

The goal in reinforcement learning is similar to that of multiarmed bandits, but different. In reinforcement learning we are typically concerned with finding the best (at least very good) policy (here: in the bandit game the best arm) as quickly as possible. In reinforcement learing we are not aiming to minimise regret. Essentially, we do not care if very bad arms are played as long as the best arm is found quickly. A learning strategy is used to find the optimal policy.

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 reinforcment is automated learning of optimal strategies in gaming or online advertisement where obviously the impact of suboptimal attempts is much less severe.

In this section we start with a simple approach to policy search. A set of possible policies is defined and then a learning strategy tries to find the best among these policies.

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 topcis 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 progress[3]. 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 underyling random variable. Either by one sample

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

or by a so-called batch of independent samples:

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

where the A^i_n are independently sampled according to $\tilde{\pi}_{\theta_n}$ and the $X^i_{A^i_n}$ are independent samples from the arms A^i_n. The sequence of $\pi_{\tilde{\theta}_n}$ of policies obtained from gradient is a learning strategy that hopefully approximates the optimal policy.

Definition 1.3.14. The appearing function $(a,\theta) \mapsto \nabla \log \pi_\theta(a)$ 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 dissapears. 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

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

Then the family $\{\pi_\theta : \theta \in \mathbb{R}^d\}$ is called the tabular softmax family. The softmax family is huge (way too big to be used in practice) and rich enough to approximate all optimal policies. Indeed, if an arm a is optimal, then the parameter θ_a needs to be sent to $+\infty$. The score-function is easily computed to be

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

[3]see 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)

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

if $N = 1$, $A_n = i$, and X_i is a sample of arm i. Here is the first explanation why reinforcment 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 interprete 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. But how about the situation in which all rewards are positive? How do we learn to distinguish good and bad arms? It would be much easier if bad arms have negative rewards leading to a reduction of likelihood in the softmax update. In that situation the actor only learns indirectly. The actor only learns an arm is bad by playing good arms which reduces the likelihood to play the bad arms. It would be much more effective to learn directly that bad arms are bad. One might think about the mensa food. If a dish is bad we can learn it is bad by trying it. If all dishes are good and a few are pretty good we can only learn a dish is relatively bad by eating better food.

Another way of seeing the same problem is to think about committal behavior. Suppose for instance that the starting vector is $\theta_0 \equiv 0$ and all arms return positive values (for instance Bernoulli bandits). Then the first arm is chosen uniformly as π_{θ_0} is uniform on $\mathcal{A}$. 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 ath entry. Hence, for the second update the ath arm is most likely to be chosen again. A way to reduce committal behavior is to substract a baseline that is supposed to help distinguish good and bad arms[4].

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 parametrisa-

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

tion)

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

as now the effect of committal behavior can be reduced drastically. An important question is what baseline to choose so that the algorithm is speed up the most. Usually people choose b to minimise the variance of the estimator of ∇J, but on the pathwise level this choice is not optimal. Thinking about the pathwise behavior it would be more reasonable to use $b = V(\pi_\theta)$ as it turns rewards negatives if the arm returns less then expectation. Unfortunately, this b is unknown and should be estimated again. Later we will call such algorithms actor-critic.

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. Intuitively that makes sense as variance in the gradient approximations leads to wrong update directions and thus slows down the convergence.

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 $J(\theta)$.

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 Lower bounds for stochastic bandits

In the previous sections we have discussed a number of explicit algorithms to learn the optimal arm of a bandit model. For reward distributions with very small tails (such as Gaussian or even bounded) we derived $\log(n)$ and $\sqrt{n}$ type estimates for the regret. In this section we derive lower bounds using estimates from classical statistics (essentiallly hypothesis testing).

1.4.1 A bit on relative entropy

Let us first recall some notion from probability theory. Suppose P and Q are probability measures on some probability space $(\Omega, \mathcal{A})$. One says $P \ll Q$, P is dominated by Q or P is absolutely continuous with respect to Q, if $P(A)$ implies $Q(A)$ for all $A \in \mathcal{A}$. The Radon-Nykodym theorem states that in that situation there is a non-negative measurable mapping f such that $Q(A) = \int_A f \, \mathrm{d}P$. While it is not too hard to show the domination property computing densities might be a complicated matter.

Definition 1.4.1. (Relative Entropie or Kullback-Leibler divergence) Suppose P and Q are probability measures on a probability space $(\Omega, \mathcal{F})$. Then

$$D(P,Q) = \begin{cases} \int \log(\frac{dP}{dQ}) \, \mathrm{d}P & : \text{if } P \ll Q \\ \infty & : \text{otherwise} \end{cases}$$

is called **relative entropie** (or KL-divergence) of P with respect to Q.

The expression divergence refers to a non-negative mapping on the cartesian product with properties similar (but weaker) than a metric. For instance, the KL divergence is not symmetric and does not satisfy the triangle inequality. Nonetheless, the KL-divergence is meant to measure in a way the difference of the measures. It is non-negative and zero if and only if $P = Q$. There is a lot of information theretic meaning hidden in the definition, we will use the divergence for hypotheses testing. The assumption of absolute continuity is nothing but saying the integral is well-defined. Most importantly, if both measures are absolutely continuous, then they are clearly absolutely continuous and the Radon-Nykodym deriviative is the ratio of the densities.

Lemma 1.4.2. Suppose $P \ll Q \ll v$, then $\frac{dP}{dQ}\frac{dQ}{dv} = \frac{dP}{dv}$.

Solving the equation yields $\frac{dP}{dQ} = \frac{\frac{dP}{dv}}{\frac{dQ}{dv}}$ but one should be a bit careful with division by zero. There is no big problem as $\{\frac{dQ}{dv} = 0\}$ is a Q-zero set and thus also a P zero set. Since densities are only unique up to nullsets one typically just modifies $\frac{dP}{dQ} = \mathbf{1}_{\frac{dQ}{dv}>0}\frac{\frac{dP}{dv}}{\frac{dQ}{dv}}$

Proof.

$$P(B) = \int_B \frac{dP}{dQ}\,\mathrm{d}Q = \int_B \frac{dP}{dQ}\frac{dQ}{dv}\,\mathrm{d}v$$

Note that there is no problem with the denomination as $P \ll Q$ implies that zeros of q are also zeros of p (except on a Q-zero set). □

The lemma shows how to compute densities $\frac{dP}{dQ}$ if one can compute densities with respect to the same dominating measure.

There are two important examples to which the lemma is usually applied. Two discrete (dominating measures the counting measure) or two absolutely continuous measures (dominating measure the Lebesgue measure). For two absolutely continuous measures P and Q with positive densities p and q the formula yields $\frac{dP}{dQ}(x) = \frac{p(x)}{q(x)}$. For discrete measures on $a_1, ..., a_N$ the formula yields $\frac{dP}{dQ}(a_k) = \frac{p_k}{q_k}$.

The formulas can be used to compute relative entropies for two discrete (resp. absolutely continuous) measures. Let's check two particularly important examples, discrete Bernoulli variables and Gaussian random variables. If $P \sim Ber(p)$ and $Q \sim Ber(q)$ with $p, q \in (0, 1)$, then

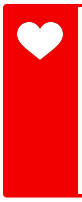
$$D(P, Q) = p \log\left(\frac{p}{q}\right) + (1-p)\log\left(\frac{1-p}{1-q}\right)$$

because both are absolutely continuous with respect to the counting measures on $\{0, 1\}$. If $P \sim \mathcal{N}(\mu_1, \sigma^2)$ and $Q \sim \mathcal{N}(\mu_2, \sigma^2)$, then, using that both are absolutely continuous with respect to the Lebesgues measure,

$$\begin{aligned}
d(P, Q) &= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}}\left(-\frac{(x-\mu_1)^2}{2\sigma^2} - \frac{(x-\mu_2)^2}{2\sigma^2}\right)e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}\mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}\frac{1}{\sigma^2}x(\mu_1-\mu_2)\mathrm{d}x + \frac{1}{\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}\ \frac{\mu_2^2-\mu_1^2}{2\sigma^2}\mathrm{d}x \\
&= \frac{1}{\sigma^2}\left(\mu_1(\mu_1-\mu_2) + \frac{\mu_2^2-\mu_1^2}{2}\right) \\
&= \frac{(\mu_2-\mu_1)^2}{2\sigma^2}.
\end{aligned}$$

There is also a sufficiently handy formula for the relative entropy of two general Gaussian vectors. We will later see how to compute relative entropies for bandit processes using absolute continuity with respect to the uniform path measure.

Here is the key application of KL-divergence needed for our purposes:



Theorem 1.4.3. (Bretagnolle-Huber)
Suppose P and Q are probability measures on a probability space $(\Omega, \mathcal{F})$. Then

$$P(A) + Q(A^C) \geq \frac{1}{2}\exp(-D(P,Q)), \qquad \forall A \in \mathcal{F}.$$

Usually the Bretagnolle-Huber inequality is stated in terms of bounding the total-variation distance of two probability measures. The version we state here is a direct consequence of the total-variation version.

Proof. Let us denote $a \wedge b = \min\{a, b\}$ and $a \vee b = \max\{a, b\}$. We set $v = P + Q$ and denote by p und q the densities of P and Q with respect to v. Those exist as clearly $P \ll v$ and $Q \ll v$. According to the remark above it follows that

$$D(P,Q) = \int_\Omega \log\left(\frac{p(\omega)}{q(\omega)}\right) P(d\omega). \tag{1.13}$$

Next, we show that

$$\int (p \wedge q) dv \geq \frac{1}{2}\exp(-D(P,Q)). \tag{1.14}$$

Since $\{\omega \colon q(\omega) = 0\}$ is a Q-zero set it follows that

$$\begin{aligned}
\exp(-D(P,Q)) &= \exp\left(-\int_{\{q>0\}} \log\left(\frac{p}{q}\right) \mathrm{d}P\right) \\
&= \exp\left(\int_{\{q>0\}} \log\left(\frac{q}{p}\right) \mathrm{d}P\right) \\
&\overset{\text{Jensen}}{\leq} \exp\left(2\log\left(\int \sqrt{\frac{q}{p}}\,\mathrm{d}P\right)\right) \\
&= \exp\left(2\log\left(\int \sqrt{\frac{q}{p}} p\,\mathrm{d}v\right)\right) \\
&= \left(\int \sqrt{pq}\,\mathrm{d}v\right)^2 \\
&= \left(\int \sqrt{(p\wedge q)(q\vee p)}\,\mathrm{d}v\right)^2 \\
&\overset{\text{Cauchy-Schwarz}}{\leq} \left(\int (p\wedge q)\,\mathrm{d}v\right)\left(\int (p\vee q)\,\mathrm{d}v\right) \\
&\leq 2\left(\int (p\wedge q)\,\mathrm{d}v\right).
\end{aligned}$$

For the final step we used that $p \wedge q + p \vee q = p + q$ so that $\int (p \vee q)\,\mathrm{d}v = 2 - \int (p \wedge q)\,\mathrm{d}v \leq 2$. This proves (1.14). Finally, using

$$\int (p\wedge q)\,\mathrm{d}v = \int_A \underbrace{(p\wedge q)}_{\leq p}\,\mathrm{d}v + \int_{A^C} \underbrace{(p\wedge q)}_{\leq q}\,\mathrm{d}v \leq P(A) + Q(A^C).$$

the claim follows. □

Here is an interesting example that explains the importance for hypothesis testing. Suppose we are given an observation and know it is Gaussian with a given variance σ^2. What is unknown is the mean, it might be 0 or Δ. Now we want to give a rule on how to determine if 0 or Δ was the mean underlying the observation. How good can such a rule be? A rule consists of a measurable set $A \subseteq \mathbb{R}$, if the observation falls in A the rule predicts $\mu = 0$, otherwise $\mu = \Delta$. Denote $P \sim \mathcal{N}(\Delta, \sigma^2)$ and $Q \sim \mathcal{N}(0, \sigma^2)$. Then the error made if the underlying distribution is P is $P(A)$, $Q(A^c)$ if the true distribution is Q. Now the Bretagnolle-Huber inequality yields

$$P(A) + Q(A^c) \geq \frac{1}{2}\exp(-D(P,Q)) = \frac{1}{2}\exp(-\frac{\Delta^2}{2\sigma^2}.)$$

If for instance $\Delta^2 < \sigma^2$ (signal (expectation) to noise (variance) ratio is small), then the righthand side is larger than $\frac{3}{10}$ implying that $\max\{P(A), Q(A^c)\} \geq \frac{3}{10}$. What does it mean? Either false-positive or false-negative must occur with at least probability $\frac{3}{10}$ if only one sample is used.

1.4.2 Mini-Max lower bounds (model-dependent)

Definition 1.4.4. If $\xi = \{v^{(i)}\}_{i \in I}$ is a family of stochastic bandit models, then

$$R_n^*(\xi) = \inf_{\pi} \sup_{v \in \xi} R_n(\pi, v)$$

is called the **minimax-regret** of ξ.

Minimising the minimax-regrets means finding a learning strategy that performs on all models, not only a fixed one. The worst possible regret should be small. In order to give a lower bound on the minimax-regret we will use the Bretagnolle-Huber inequality. To get the inequality working in the context of bandits the following lemma is neeeded:

Lemma 1.4.5. (Entropie-decomposition)
Let k the number of arms and $v = \{P_1, \ldots, P_k\}$, $v' = \{P_1', \ldots, P_k'\}$ the reward distributions of two k-armed bandits. Furthermore, let π a learning strategy and $\mathbb{P}$ and $\mathbb{P}'$ the probability distributions on $(\mathbb{R} \times \{1, \ldots, k\})^n$, that describe the n-step bandit process for the two models under the learning strategy π. In this notation $\mathbb{R}$ describes the reward and $\mathcal{A} = \{1, \ldots, k\}$ the arms. Then the following decomposition holds:

$$D(\mathbb{P}_\pi, \mathbb{P}'_\pi) = \sum_{a \in \mathcal{A}} \mathbb{E}_\pi[T_a(n)]\ D(P_a, P_a').$$

Proof. For simplicity let us first assume the bandit model is discrete, i.e. the reward distributions P_a are discrete (absolutely continuous with respect to a counting measure λ). The main observation is that the law of the bandit process for a given policy π is absolutely continuous with respect to counting measure on the finite set of action-reward paths taking values in the support of P_a. The counting measure on paths is $(\rho \otimes \lambda)^{\otimes n}$, where ρ is the counting measure on $\{1, ..., K\}$. Keep in mind, we are only talking about discrete measures which are absolutely continuous with respect to the counting measure of the underlying set and the density is giving by the singleton probabilities. Hence,

$$\mathbb{P}_\pi(A) = \int_A p(a_1, x_1, \ldots, a_n, x_n)(\rho \otimes \lambda)^{\otimes n}(\,\mathrm{d}(a_1, x_1, ..., a_n, x_n))$$

with the density being the singleton probabilities $A = \{(a_1, x_1, ..., a_n, x_n)\}$ of paths:

$$p(a_1, x_1, \ldots, a_n, x_n) = \prod_{t=1}^{n} \pi_t(\{a_t\}\,;\, a_1, x_1, \ldots a_{t-1}, x_{t-1})\ P_{a_t}(\{x_t\})$$

Similarly, $\mathbb{P}'_\pi$ has density

$$p'(a_1, x_1, \ldots, a_n, x_n) = \prod_{t=1}^{n} \pi_t(\{a_t\}\,;\, a_1, x_1, \ldots a_{t-1}, x_{t-1})\; P'_{a_t}(\{x_t\})$$

with respect to $(\rho \otimes \lambda)^{\otimes n}$. Now we are in the situation that both measures $\mathbb{P}$ and $\mathbb{P}'$ are absolutely continuous to the same domination measure. Hence, the density of $\mathbb{P}$ with respect to $\mathbb{P}'$ is

$$\frac{\mathrm{d}\mathbb{P}_\pi}{\mathrm{d}\mathbb{P}'_\pi}(a_1, x_1, \ldots, a_n, x_n) = \frac{\prod_{t=1}^n \pi_t(\{a_t\}\,;\, a_1, x_1, \ldots a_{t-1}, x_{t-1})\; P_{a_t}(x_t)}{\prod_{t=1}^n \pi_t(\{a_t\}\,;\, a_1, x_1, \ldots a_{t-1}, x_{t-1})\; P'_{a_t}(\{x_t\})} = \frac{\prod_{t=1}^n P_{a_t}(\{x_t\})}{\prod_{t=1}^n P'_{a_t}(\{x_t\})}$$

Plugging-into the definition of relative entropy yields

$$\begin{aligned}
D(\mathbb{P}_\pi, \mathbb{P}'_\pi) &= \mathbb{E}_\pi\Big[\log\Big(\prod_{t\le n} \frac{P_{A_t}(\{X_t\})}{P'_{A_t}(\{X_t\})}\Big)\Big] \\
&= \sum_{t\le n} \mathbb{E}_\pi\Big[\log\Big(\frac{P_{A_t}(\{X_t\})}{P'_{A_t}(\{X_t\})}\Big)\Big] \\
&= \sum_{t\le n}\sum_{a=1}^{k} \mathbb{E}_\pi\Big[\mathbf{1}_{\{A_t=a\}} \log\Big(\frac{P_a(\{X_t\})}{P'_a(\{X_t\})}\Big)\Big] \\
&= \sum_{a=1}^{k}\sum_{t\le n} \mathbb{P}_\pi(A_t = a) D(P_a, P'_a) \\
&= \sum_{a=1}^{k} \mathbb{E}_\pi[T_a(n)] D(P_a, P'_a),
\end{aligned}$$

using that $\mathbb{P}(X_t \in B \mid A_t = a) = P_a(B)$.

If all arms are absolutely continuous the same argument works chosing λ to be Lebesgues measure replacing P_a by the densities p_a of P_a. □

Here is the main theorem, a minimax-regret bound for Gaussian rewards.

Theorem 1.4.6. Let ξ^k the family of all k-armed bandits with Gaussian rewards unit variance and expectations $\mu_1, ..., \mu_k \in [0, 1]$. Then

$$R^*_n(\xi^k) \ge \frac{1}{27}\sqrt{(k-1)n}$$

holds for all $n \ge k$.

Proof of Theorem 1.4.6. Every model is described by an expectation vector $\mu = (\mu_1, ..., \mu_k) \in [0, 1]^k$. We show that for all $n \ge k - 1$ and every learning strategy π there is an expectation vector μ with

$$R_n(\pi, v_\mu) \ge \frac{1}{27}\sqrt{(k-1)n}.$$

Let $\Delta \in [0, \frac{1}{2}]$ arbitrary and set $\mu = \mu(\Delta) = (\Delta, 0, \ldots, 0)$. Next, let $a = \operatorname{argmin}_{i\le k} \mathbb{E}_{v_\mu,\pi}[T_i(n)]$ the arm which is played the least under learning strategy π on the bandit model v_μ. Furthermore, define

$$\mu' = (\Delta, 0, \ldots, 0, \underbrace{2\Delta}_{a\text{th position}}, 0, \ldots, 0).$$

To safe some notation we write $\mathbb{P}$ and $\mathbb{P}'$ for the laws of the bandit processes playing both policy π but on the different bandit models μ and μ'. In what follows we show that

$$\min\{R_n(\pi, v_\mu), R_n(\pi, v_{\mu'})\} \geq \frac{1}{27}\sqrt{(k-1)n}$$

for a suitably chosen Δ. Thus, there is (at least) one bandit model with regret lower bound $\frac{1}{27}\sqrt{(k-1)n}$. To do so, we start with a first computation based on the regret decomposition:

$$\begin{aligned}
R_n(\pi, v_\mu) &= \sum_i \mathbb{E}[T_i(n)]\Delta_i \\
&= \sum_{i\neq 1} \mathbb{E}[T_i(n)]\Delta \\
&= \Delta\big(n - \mathbb{E}[T_1(n)]\big) \\
&= \Delta\big(n - \mathbb{E}[T_1(n)(\mathbf{1}_{T_1(n)>n/2} + \mathbf{1}_{T_1(n)\leq n/2})]\big) \\
&\overset{T_1(n)\leq n}{\geq} \Delta\Big(n - n\mathbb{P}\Big(T_1(n) > \frac{n}{2}\Big) - \frac{n}{2}\Big(\mathbb{P}\Big(T_1(n) \leq \frac{n}{2}\Big)\Big)\Big) \\
&= \frac{n\Delta}{2}\mathbb{P}\Big(T_1(n) \leq \frac{n}{2}\Big)
\end{aligned}$$

Estimating $R_n(\pi, v_{\mu'})$ with the complementary probability is simpler as the bandit model was chosen in a way to produce regret gaps Δ when playing arm 1:

$$R_n(\pi, v_{\mu'}) = \sum_i \mathbb{E}'[T_i(n)]\Delta_i' \geq \mathbb{E}'[T_1(n)]\Delta \geq \mathbb{E}'[T_1(n)\mathbf{1}_{T_1(n)>\frac{n}{2}}]\Delta \geq \frac{n\Delta}{2}\mathbb{P}'\Big(T_1(n) > \frac{n}{2}\Big).$$

Combining both and applying Bretagnolle-Huber with $A = \{T_1(n) \leq \frac{n}{2}\}$ yields

$$\begin{aligned}
R_n(\pi, v_\mu) + R_n(\pi, v_{\mu'}) &\geq \frac{n\Delta}{2}\Big(\mathbb{P}\left(T_1(n) \leq \frac{n}{2}\right) + \mathbb{P}'\left(T_1(n) > \frac{n}{2}\right)\Big) \\
&\overset{\text{Thm 1.4.3}}{\geq} \frac{n\Delta}{4}\exp(-D(\mathbb{P}, \mathbb{P}')) \\
&\overset{\text{Thm 1.4.5}}{=} \frac{n\Delta}{4}\exp\Big(-\sum_i \mathbb{E}_\mu[T_i(n)]\underbrace{D(\mathcal{N}(\mu_i, 1), \mathcal{N}(\mu_i', 1))}_{=0, \text{ except } i=a}\Big) \\
&= \frac{n\Delta}{4}\exp\Big(-\mathbb{E}_\mu[T_a(n)]\frac{(2\Delta)^2}{2}\Big) \\
&\geq \frac{n\Delta}{4}\exp\left(-\frac{2n\Delta^2}{k-1}\right).
\end{aligned}$$

[5] The final inequality holds because a is the arm with smallest expected number of play, hence, $(k-1)\mathbb{E}[T_a(n)] \leq \sum_i \mathbb{E}[T_i(n)] = n$. Minimising the righthand side over Δ (or just chosing) $\Delta = \sqrt{\frac{k-1}{4n}}$ yields

$$R_n(\pi, v_\mu) + R_n(\pi, v_{\mu'}) \geq \sqrt{(k-1)n}\frac{1}{8}e^{-\frac{1}{2}} \geq \frac{2}{27}\sqrt{(k-1)n}.$$

□

1.4.3 Asymptotic lower bound (model-dependent)

We next turn towards model-dependent asymptotic bounds. What we try to do is to find lower bounds (asymptotically in n) for a policy for a fixed model from a certain class of models. For instance a lower bound of the UCB algorithm for a given sub-Gaussian bandit model. A general

[5] warum nicht k statt $k-1$?

solution is problematic for the following reason. If a policy only plays a fixed arm then the regret is either 0 or grows linearly depending on that arm to be optimal or not. To rule out such extremal cases we only consider policies that certainly have sublinear growth in the following sense:

Definition 1.4.7. A policy π is called consistent over a class of bandits $\mathcal{E}$ if for all $\mu \in \mathcal{E}$ and all $p > 0$ it holds that

$$\lim_{n\to\infty} \frac{R_n(\pi,\nu)}{n^p} = 0.$$

An example of a consistent policy over all sub-Gaussian bandits is the policy obtained from the UCB algorithm. Here is the famous Lai-Robbins theorem on asymptotic lower bounds[6]:

Theorem 1.4.8. (Lai-Robbins lower bound)
If π is a consistent policy for a set $\mathcal{E} = \mathcal{M}_1 \times ... \times \mathcal{M}_k$ of bandit models, then the regret for all bandits $\nu = (P_1, ..., P_k)$ from $\mathcal{E}$ satisfies

$$\liminf_{n\to\infty} \frac{R_n(\pi)}{\log(n)} \geq c^*(\nu,\mathcal{E}) := \sum_{a\in\mathcal{A}} \frac{\Delta_a}{d_a},$$

where $d_a = \inf_{P'\in\mathcal{M}_a}\{D(P_a,P') : Q_{P'} > Q_*\}$ and Q_P is the expectation of P.

Lai Robbins actually assumed all $\mathcal{M}_a$ to be equal and an additional continuity property on the relative entropies of that class (continuity of the KL-distance as a function of the expectations). In that case the lower bound looks simpler because $d_a = D(P_a, P_*)$, where P_* is the law of the optimal arm:

$$\liminf_{n\to\infty} \frac{R_n(\pi)}{\log(n)} \geq \sum_{a\in\mathcal{A}} \frac{\Delta_a}{D(P_a,P_*)}$$

Here is an example in which a directly computation shows the same. If for instance $\mathcal{M}_a = \{\mathcal{N}(\mu,\sigma^2) : \mu \in \mathbb{R}\}$ for all a then $d_a = \frac{(Q_a-Q_*)^2}{\sigma^2} = \frac{\Delta^2}{\sigma^2} = D(P_a,P_*)$. Thus, if all arms are Gaussian with variance σ^2, then the lower bound for UCB is

$$\liminf_{n\to\infty} \frac{R_n}{\log(n)} \geq \sum_{a\neq a_*} \frac{\sigma^2}{\Delta_a}.$$

Comparing with the UCB upper bound of Theorem 1.3.8, $\sigma = 1$ this is pretty close. In fact, it is not too difficult to modify the UCB bonus so that the upper bound matches the lower bound, thus, the algorithm is asymptotically optimal. The term d_a in a way also measures the reward gap but rather the KL-distance from the optimal arm, not the mean distance.

Proof. Fix a consistent policy and suppose $\nu = (P_1, ..., P_k) \in \mathcal{E}$. Recalling the regret decomposition it is enough to prove that

$$\liminf_{n\to\infty} \frac{\mathbb{E}_\pi[T_a(n)]}{\log(n)} \geq \frac{1}{d_a}$$

holds for all suboptimal arm. Fix such an arm a and fix an $\varepsilon > 0$ and define ν' as the bandit model in which the arm distribution P_a is replaced by a distribution $P'_a \in \mathcal{M}_a$ such that $D(P_a,P'_a) \leq d_a + \varepsilon$ and $Q_{P'_a} > Q_*$. By the definition of the infimum such a bandit model exists in the class $\mathcal{E}$. Since only one arm differs the entropie decomposition for bandits yields

[6] T.L Lai, H. Robbins: "Asymptotically efficient adaptive allocation rules", Advances in Applied Mathematics, 1985, pp. 4-22

$D(\mathbb{P}, \mathbb{P}') \leq \mathbb{E}[T_a(n)](d_a + \varepsilon)$. We next follow an argument that is inspired by the proof of Theorem 1.4.6. First note that

$$R_n(\pi) + R'_n(\pi)) \geq \frac{n}{2}\Big(\mathbb{P}\Big(T_a(n) > \frac{n}{2}\Big)\Delta_a + \mathbb{P}\Big(T_a(n) \leq \frac{n}{2}\Big)(Q'_a - Q_*)\Big).$$

First, from the regret decomposition it clearly holds that

$$R_n(\pi)) \geq \Delta_a \mathbb{E}[T_a(n)] \geq \Delta_a \mathbb{E}[T_a(n)\mathbf{1}_{T_a(n)>n/2}] \geq \Delta_a \frac{n}{2}\mathbb{P}\Big(T_a(n) > \frac{n}{2}\Big).$$

Secondly, note that for ν' the best arm is a as the new arm distribution has a mean strictly larger than Q_*. Hence, if arm a is played less than $\frac{n}{2}$ times, then all other arms in are platyed at least $\frac{n}{2}$ times in total. Thus,

$$\begin{aligned}
R'_n(\pi) &= \sum_{i \neq a} \Delta'_i \mathbb{E}'[T_i(n)] \\
&\geq \sum_{i \neq a} (Q'_a - Q_*)\mathbb{E}'[T_i(n)] \\
&\geq \sum_{i \neq a} (Q'_a - Q_*)\mathbb{E}'[T_i(n)\mathbf{1}_{T_a(n)<n/2}] \\
&\geq \frac{n}{2}(Q'_a - Q_*)\mathbb{P}'\Big(T_a(n) \leq \frac{n}{2}\Big).
\end{aligned}$$

Chosing $A = \{T_a(n) > n/2\}$ we continue with Bretagnolle-Huber:

$$\begin{aligned}
R_n(\pi) + R'_n(\pi) &\geq \frac{n}{2}\Big(\mathbb{P}\Big(T_a(n) > \frac{n}{2}\Big)\Delta_a + \mathbb{P}'\Big(T_a(n) \leq \frac{n}{2}\Big)(Q'_a - Q_*)\Big) \\
&\geq \frac{n}{2}\min\{\Delta_a, Q'_a - Q_*\}\big(\mathbb{P}(A) + \mathbb{P}'(A^c)\big) \\
&\geq \frac{n}{4}\min\{\Delta_a, Q'_a - Q_*\}\exp\big(-\mathbb{E}[T_a(n)](d_a + \varepsilon)\big).
\end{aligned}$$

Rearranging and using the consistency property of the policy yields the claim:

$$\begin{aligned}
\liminf_{n\to\infty} \frac{\mathbb{E}[T_a(n)]}{\log(n)} &\geq \frac{1}{d_a + \varepsilon} \liminf_{n\to\infty} \frac{\log\left(\frac{n\min\{\Delta_a, Q'_a - Q_*\}}{4(R_n(\pi) + R'_n(\pi))}\right)}{\log(n)} \\
&\geq \frac{1}{d_a + \varepsilon}\Big(1 - \limsup_{n\to\infty} \frac{\log(R_n(\pi) + R'_n(\pi))}{\log(n)}\Big) = \frac{1}{d_a + \varepsilon}.
\end{aligned}$$

For the last equality we used that, for n large enough, $R_n(\pi) \leq Cn^p$ and $R'_n(\pi) \leq C'n^p$ so that

$$0 \leq \limsup_{n\to\infty} \frac{\log(R_n(\pi) + R'_n(\pi))}{\log(n)} \leq \limsup_{n\to\infty} \frac{\log(Cn^p + C'n^p)}{\log(n)} = \limsup_{n\to\infty} \frac{\log(C) + p\log(n)}{\log(n)} = p.$$

Since p can be chosen arbitrarily close to 0 the limit superior exists and is equal to 0. □

Part I

Tabular Reinforcement Learning

Chapter 2

Basics: Stochastic Control and Dynamic Programming Methods

Lecture 6

2.1 Markov decision problems

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 Putterman[1]. 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.

2.1.1 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 $(S_t)_{t\in\mathbb{N}}$ with values in some finite set $\mathcal{S}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ that satisfies the Markov property:

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

for all $t \in \mathbb{N}_0$ and $s_0, ..., s_{t+1} \in \mathcal{S}$. 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 P, 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

$$\mathbb{P}(S_0 = s_0, ..., S_n = s_n) = \mu(s_0) p_{s_0,s_1} \cdot ... \cdot p_{s_{n-1},s_n}. \tag{2.1}$$

[1] M. 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 $P(\cdot, s)$ on $\mathcal{S}$ for all $s \in \mathcal{S}$. Then the path probabilities can be written as

$$\mathbb{P}(S_0 = s_0, ..., S_n = s_n) = \mu(\{s_0\})P(\{s_1\}, s_0) \cdot ... \cdot P(\{s_n\}, s_{n-1}).$$

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

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

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'. Another way of expressing the Markov property is as follows: If $\mathbb{P}(S_n = s') > 0$ and $\tilde{\mathbb{P}} := \mathbb{P}(\cdot \,|\, S_n = s')$, then on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ the shifted process $(\tilde{S}_t)_{t\in\mathbb{N}} := (S_{t+n})_{t\in\mathbb{N}}$ is again a Markov chain with transition matrix P but started from s'. To train yourself in using the path probabilities please check the next claim:

Check that $\tilde{S}$ indeed is a Markov chain on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ with the same transitions as S. Hint: Compute path probabilities.

A Markov chain can be seen as a random process that can be observed from the 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 $(R_t)_{t\in\mathbb{N}}$ that is sampled together with the next state. There is a transition/reward-kernel p on $(\mathcal{R} \times \mathcal{S}) \times \mathcal{S}$ from which next state and reward are sampled, i.e. $p(r, s' \,;\, s)$ denotes the probability to transition the chain from s to s' and obtain reward r. More generally, the defining property of a reward Markov chain is

$$\mathbb{P}_\mu(S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} \,|\, S_0 = s_0, ..., S_t = s_t) = \mathbb{P}_\mu(S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} \,|\, S_t = s_t),$$

where the subscript refers to the initial distribution of S_0. If we think of a Markov chain as describing a game than the rewards might be direct consequences of the rules. As an example, $R_t = 1$ if and only if the player that we observe has scored a goal. The path probabilities are given by

$$\mathbb{P}_\mu(S_0 = s_0, R_0 = r_0, ..., S_n = s_n, R_n = r_n) = \mu(s_0) \cdot p(r_0, s_1 \,;\, s_0) \cdot ... \cdot p(r_n, s_n \,;\, s_{n-1})$$

Another way of expressing the Markov reward property is as follows: If $\mathbb{P}(S_n = s') > 0$ and $\tilde{\mathbb{P}} := \mathbb{P}(\cdot \,|\, S_n = s')$, then on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ the shifted process $(\tilde{S}_t, \tilde{R}_t)_{t\in\mathbb{N}} := (S_{t+n}, R_{t+n})_{t\in\mathbb{N}}$ is again a Markov reward chain started from s'. To train yourself in using the path probabilities please check the next claim:

Compute the path probabilities to check that $(\tilde{S}, \tilde{R})$ indeed is a Markov chain on $(\Omega, \mathcal{F}, \tilde{\mathbb{P}})$ with the same transitions as (S, R). Hint: Compute path probabilities.

In particular, and this is what we will need below

$$\mathbb{E}_s[f(R_t, R_{t+1}, ...)|S_t = s'] = \mathbb{E}_{s'}[f(R_0, R_1, ...)]. \tag{2.2}$$

2.1.2 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 particpant. 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 shophistication 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 loosing most features of interest.

Definition 2.1.1. (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 $s \in S$, $(\mathcal{A}_s, \bar{\mathcal{A}}_s)$ is a measurable space, called the action space of state s. The entire action space is defined to be $\mathcal{A} = \bigcup_{s \in S} \mathcal{A}_s$, $\mathcal{A}$ contains all actions and is equipped with the σ-algebra $\bar{\mathcal{A}} = \sigma(\bigcup_{s \in S} \bar{\mathcal{A}}_s)$.

(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

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

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

- $(s,a) \mapsto p(B\,;\, s,a)$ is $(\bar{\mathcal{A}} \otimes \bar{\mathcal{S}})$-$\bar{\mathcal{R}}$-measurable for all B,
- $B \mapsto p(B\,;\, s,a)$ 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 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 2.1.5 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. A Markov decision model does not fully describe a process that runs from state to state and returns rewards. Only the transitions $(s,a) \mapsto (s', r)$ is described but not how the next action a' is chosen. For that purpose an additional ingredient is

needed, the policy that can be chosen by the actor.

Definition 2.1.2. (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

$$\pi_t(\mathcal{A}_s; s_0, a_0, ..., s_{t-1}, a_{t-1}, s) = 1 \tag{2.3}$$

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

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 (2.3) 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 decison 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 $\mathcal{S}\times\mathcal{A}\times\mathcal{R}$ 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 2.1.3. (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 S. Then there exists a probability space $(\Omega, \mathcal{F}, \mathbb{P}^\pi_\mu)$ carrying a stochastic process $(S_t, A_t, R_t)_{t\in\mathbb{N}_0}$ with values in $S \times A \times R$ on $(\Omega, \mathcal{F}, \mathbb{P}^\pi_\mu)$ such that, for all $t \in \mathbb{N}$,

$$\begin{aligned}
&\mathbb{P}^\pi_\mu(S_0 = s_0, A_0 = a_0) = \mu(s_0)\pi_0(a_0\,;\, s_0)\\
&\mathbb{P}^\pi_\mu(A_t = a_t \,|\, S_0 = s_0, A_0 = a_0, ..., S_t = s_t) = \pi_t(a_t\,;\, s_0, a_0, ..., s_t),\\
&\mathbb{P}^\pi_\mu(S_{t+1} = s_{t+1}, R_t = r_t \,|\, S_t = s, A_t = a) = p(s_{t+1}, r_t\,;\, s, a).
\end{aligned}$$

To increase readability we will skipp the π and often μ from the notation.

[2] 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 safe quite a bit of time. The proof is essentially identical to the existence proof for Markov chains.

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

[2] todo: Schreibe lieber allgemeinen Beweis fuer Markov reward chains auf und zitiere hier. Vermutlich besser zugaenglich

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 $\Omega := (\mathcal{S} \times \mathcal{A} \times \mathcal{R})^T$, the trajectories of length T, i.e.

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

and $\mathcal{F}_T$ as the powerset. As for Markov chains the probabilities for paths are written down explicitly (skipping brackets to increase readability):

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

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 $\sum_{a_0,r_0,s_0} \cdots \sum_{a_T,r_T,s_T}$, 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)

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

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 $(\Omega, \mathcal{F})$ by $\Omega := (\mathcal{S} \times \mathcal{R} \times \mathcal{A})^\infty$, the trajectories of infinite length, and the corresponding σ-algebra $\mathcal{F} := (\bar{\mathcal{S}} \otimes \bar{\mathcal{A}} \otimes \bar{\mathcal{R}})^{\otimes \infty}$, the cylinder σ-algebra. The elements of Ω can be written as infinite sequences of the form

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

Consider the projections $\pi_T^{T+1} : \Omega^{T+1} \to \Omega^T, \omega \mapsto \omega|_T$ and $\pi_T : \Omega \to \Omega^T, \omega \mapsto \omega|_T$. 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 $(\Omega, \mathcal{F})$ 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:

$$\begin{aligned}&\mathbb{P}_{T+1}\big((\pi_T^{T+1})^{-1}(s_0, a_0, r_0, \ldots, s_T, a_T, r_T)\big)\\ &= \mathbb{P}_{T+1}(\cup_{s\in\mathcal{S}} \cup_{a\in\mathcal{A}} \cup_{r\in\mathcal{R}} \{(s_0, a_0, r_0, \ldots, s_T, a_T, r_T, s, a, r)\})\\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{r\in\mathcal{R}} \mathbb{P}_{T+1}(s_0, a_0, r_0, \ldots, s_T, a_T, r_T, s, a, r)\\ &\overset{\text{linearity}}{=} \mu(s_0) \cdot \pi_0(a_0 \,;\, s_0) \cdot \prod_{i=1}^{T} p(s_i, r_{i-1} \,;\, s_{i-1}, a_{i-1}) \cdot \pi_i(a_i \,;\, s_0, a_0, \ldots, a_{i-1}, s_i)\end{aligned}$$

$$\times \underbrace{\sum_{s\in\mathcal{S}} p(s, r_T\,;\, s_T, a_T)}_{p(\mathcal{S}\times\{r_T\}\,;\,S_T,a_T)} \cdot \underbrace{\sum_{a\in\mathcal{A}} \pi_{T+1}(a\,;\, s_0, a_0, \ldots, a_T, s_{T+1})}_{=1} \underbrace{\sum_{r\in\mathcal{R}} p(\mathcal{S}\times\{r\}\,;\, s_T, a_T)}_{=1}.$$

$$= \mu(s_0)\cdot\pi_0(a_0\,;\,s_0)\cdot\prod_{i=1}^{T} p(s_i, r_{i-1}\,;\, s_{i-1}, a_{i-1})\cdot\pi_i(a_i\,;\, s_0, a_0, \ldots, a_T, s_{T+1})$$
$$\cdot\, p(\mathcal{S}\times\{r_T\}\,;\, S_T, a_T)$$
$$= \mathbb{P}_T(s_0, a_0, r_0, \ldots, s_T, a_T, r_T).$$

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

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

The measurable space $(\Omega, \mathcal{F})$ 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 $(s_0, a_0, r_0, ...)$.

The properties claimed in the theorem hold.

The first two claims follow directly from the definition of the measure $\mathbb{P}^\pi_\mu$ and (S_0, A_0, R_0). We check the claim for A and leave the other claims as an exercise. First note that putting no restrictions to the rewards in the path probabilities leads to setting $p(\{s'\}\times\mathcal{R}\,;\, s, a)$ in the state-reward transitions (summing over all possibilities). Using the extension property of $\mathbb{P}^\pi_\mu$ (there is no dependence later than t) yields

$$\mathbb{P}^\pi_\mu(S_0 = s_0, A_0 = a_0, ..., A_{t-1} = a_{t-1}, S_t = s_t, A_t = a_t)$$
$$= \mu(s_0)\cdot\pi_0(a_0\,;\,s_0)\cdot\prod_{i=1}^{t} p(\{s_i\}\times\mathcal{R}\,;\, s_{i-1}, a_{i-1})\cdot\pi_i(a_i\,;\, s_0, a_0, \ldots, a_{i-1}, s_i)$$

and

$$\mathbb{P}^\pi_\mu(S_0 = s_0, A_0 = a_0, ..., A_{t-1} = a_{t-1}, S_t = s_t)$$
$$= \mathbb{P}^\pi_\mu(S_0 = s_0, A_0 = a_0, ..., A_{t-1} = a_{t-1}, S_t = s_t, A_t \in \mathcal{A})$$
$$= \mu(s_0)\cdot\pi_0(a_0\,;\,s_0)\cdot\prod_{i=1}^{t-1} p(\{s_i\}\times\mathcal{R}\,;\, s_{i-1}, a_{i-1})\cdot\pi_i(a_i\,;\, s_0, a_0, \ldots, a_{i-1}, s_i)$$
$$\times\, p(\{s_t\}\times\mathcal{R}\,;\, s_{t-1}, a_{t-1})\cdot\underbrace{\sum_{a\in\mathcal{A}}\pi_t(a; s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t)}_{=1}.$$

Hence, using the definition of conditional probability the products cancel in the fraction to give

$$\mathbb{P}^\pi_\mu(A_t = a_t\,|S_0 = s_0, A_0 = a_0, ..., A_{t-1} = a_{t-1}, S_t = s_t) = \pi_t(a_t\,;\, s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t)$$

Check the other two claimed conditional probability identities.

□

Definition 2.1.4. (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 S, the stochastic process $(S_t, A_t)_{t\in\mathbb{N}_0}$ on $(\Omega, \mathcal{F}, \mathbb{P}^\pi_\mu)$ is called discrete-time Markov decision process (MDP), $(R_t)_{t\in\mathbb{N}_0}$ the corresponding reward process. To abbreviate we will write $\mathbb{P}^\pi_s$ instead of $\mathbb{P}^\pi_{\delta_s}$. The super- and subscript are sometimes removed from $\mathbb{E}^\pi_\mu$ and $\mathbb{P}^\pi_\mu$ if there is no possible confusion about the initial distribution of 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

$$\begin{aligned}&\mathbb{P}(S_0 = s_0, A_0 = a_0, R_0 = r_0, \dots, S_T = s_T, A_T = a_T, R_T = r_T)\\ &= \mu(s_0)\cdot\pi_0(a_0\,;\,s_0)\cdot\prod_{i=1}^{T} p(s_i, r_{i-1}\,;\,s_{i-1}, a_{i-1})\cdot\pi_i(a_i\,;\,s_0, a_0, \dots, a_{i-1}, s_i) \qquad (2.4)\\ &\quad\cdot p(\mathcal{S}\times\{r_T\}, s_T, a_t),\end{aligned}$$

which gives the probabilitites an MDP follows a given path $(s_0, a_0, r_0, ..., s_T, a_T, r_T)$ of states, actions, and rewards. The formula will later explain the practical importance of the policy gradient theorem.

In 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 $\mathcal{R} = \{0\}$. In that case then the process (S_t) is a Markov chain with transition matrix $p_{s_i,s_j} = p(s_j, 0\,;\,s_i, a)$ and the path probabilities immediately simplify to the Markov chain formula (2.1). No doubt, for MDPs the formula is more complicated but many probabilities can be computed simmilarly 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 $\sum_i(a\cdot b_i)/\sum_i b_i = a$):

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

and

$$\begin{aligned}&\mathbb{P}(S_t = s_t, A_t = a_t, R_t = r_t, \dots, S_T = s_T, A_T = a_T, R_T = r_T \,|\, S_{t-1} = s)\\ &= \sum_{a\in\mathcal{A}_s}\pi_{t-1}(a\,;\,s_0, a_0, \dots, a_{t-2}, s)\prod_{i=t}^{T} p(s_i, r_i\,;\,s_{i-1}, a_{i-1})\cdot\pi_i(a_i\,;\,s_0, a_0, \dots, a_{i-1}, s_i),\end{aligned}$$

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 2.1.5. 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 through

$$p(s',r\,;\,s,a) := h(s'\,;\,s,a)\cdot q(r\,;\,s,a) \tag{2.5}$$

and the corresponding Markov decision process samples next states/rewards independently. Now the Markov decision model in the sense of Definition 2.1.1 with Markov decision process from Theorem 2.1.3 has exactly the claimed transitions. Plugging-in p immediately gives

$$\begin{aligned}
&\mathbb{P}(R_t = r_t \,|\, S_t = s_t, A_t = a_t, S_{t+1} = s_{t+1}) \\
&\overset{\text{cond. prob.}}{=} \frac{\mathbb{P}(S_{t+1} = s_{t+1}, R_t = r_t \,|\, S_t = s_t, A_t = a_t)}{\mathbb{P}(S_{t+1} = s_{t+1} \,|\, S_t = s_t, A_t = a_t)} \\
&\overset{2.1.3}{=} \frac{p(s_{t+1}, r_t\,;\,s_t,a_t)}{p(\{s_{t+1}\}\times\mathcal{R}\,;\,s_t,a_t)} \\
&\overset{(2.5)}{=} p(\mathcal{S}\times\{r_t\}\,;\,s_t,a_t) \\
&\overset{2.1.3}{=} \mathbb{P}(R_t = r_t \,|\, S_t = s_t, A_t = a_t),
\end{aligned}$$

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

Example 2.1.6. In many examples the rewards are deterministic functions of state, actions, and next state but themselves do not influence the next state (say $R_t = R(s_t, a_t, s_{t+1})$). An example appears in the ice vendor example below. In that situation the kernel p is simply

$$p(s',r\,;\,s,a) = \begin{cases} h(s'\,;\,s,a) & : r = R(s,a,s') \\ 0 & : r \neq R(s,a,s') \end{cases}$$

where h is the transition function from (s,a) to the next state s'. 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 2.1.7. For a Markov decision model we define state-action-state probabilities and expected rewards as well as the state-action expected rewards as follows:

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

for $s,s'\in\mathcal{S}$ and $a\in\mathcal{A}$ if the denominator is non-zero.

The notation will be used frequently, in reinforcement algorithms of the upcoming chapters. If for instance we are only interested in the state-action process, then it is more convenient to use $p(s';s,a)$ instead of $p(\{s'\}\times\mathcal{R};s,a)$ in computations with the path probabilities.

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

$$p(s';s,a) = \mathbb{P}(S_{t+1} = s' \,|\, S_t = s, A_t = a),$$

$$p(r; s, a) = \mathbb{P}(R_t = r \mid S_t = s, A_t = a),$$
$$r(s, a) = \mathbb{E}[R_t \mid S_t = s, A_t = a],$$
$$r(s, a, s') = \mathbb{E}[R_t \mid S_t = s, A_t = a, S_{t+1} = s']$$

holds if the events in the condition have positive probability.

To get an idea about Markov decision processes we will discuss two examples of simple Markov decision models, grid world and ice vendor. Many games, such as grid world, stop once a particular state is reached. Such states are called terminating states.

Definition 2.1.8. A state $s \in S$ satisfying $p(s\,;\, s, a) = 1$ for all $a \in \mathcal{A}_s$ is called a terminating state. The set of terminating states will be denoted by Δ and we will always assume $p(s, 0\,;\, s, a) = 1$ for all all $s \in \Delta$, $a \in \mathcal{A}_s$.

In words: Once a terminal state is hit the MDP will stop moving and all future rewards are 0. To get a feeling for the definitions we will spell out a representative example in detail:

Example 2.1.9. (Grid world)
Suppose we have a robot that we want to teach to walk through our 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 trough 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 bomb B just before the goal G. If the robot lands in the bomb or in the target, the game is over. In

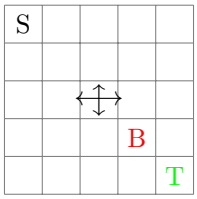


Figure 2.1: Grid world of size 5. The start is marked by S, the bomb by B, and the goal by G.

the following we will formulate this example as a Markov Decision Model. For this purpose, the individual objects from Definition 2.1.1 must be determined. The state space should contain all relevant information about the game. The garden can be represented by a two dimensional grid similar to a chess board. Thus, for a grid with side length $Z \in \mathbb{N}, Z > 2$, the state space is given by $S \coloneqq \{(i, j), i, j \in \{0, 1, \ldots, Z-1\}\}$. Since the state space S is finite, we will use as σ-algebra the power set, i.e. $\mathcal{S} \coloneqq \mathcal{P}(S)$.

0,0	0,1	0,2	0,3	0,4
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 2.2: Representation of the state space of Example 2.1.9 for a grid of length 5.

The second component of the Markov decision model is the entire action space $\mathcal{A}$ and the action spaces $\mathcal{A}_s$ for states $s \in S$. 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:

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

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 determine the action space for a state s using what is called masking (excluding actions from the set $\mathcal{A}$ of all possible actions). Let $s = (i,j) \in S$ then the valid actions for the state are given by

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

The agent may play all actions, except the agent is on the edge of the playing field. 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 $\overline{\mathcal{R}} \coloneqq \mathcal{P}(\mathcal{R})$. If the agent lands in the target, the reward is 10. If the agent runs into the bomb, 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. Next, we define the fourth and most complicated component of the Markov decision process, the transition/reward-function p. To define this function, we will first define an auxiliary function $g : S \to \mathcal{R}$ which should return the associated reward for a state $s \in \mathcal{S}$. The function is given by

$$g : \mathcal{S} \to \mathcal{R}, s \mapsto \begin{cases} 10 & : s = T \\ -10 & : s = B \\ -1 & : \text{otherwise} \end{cases}.$$

In addition, we still need to define the set of terminal states Δ. In our example, the set of terminal states consists of the target and the bomb, $\Delta = \{(Z-2,Z-2),(Z-1,Z-1)\}$. The transition function is defined by

$$p(s', r\,;\, s, a) = \mathbf{1}_{s\in\Delta} \cdot \mathbf{1}_{s'=s} \cdot \mathbf{1}_{r=0} + \mathbf{1}_{s\notin\Delta} \cdot \mathbf{1}_{s'=s+a} \cdot \mathbf{1}_{r=g(s+a)}$$

for $(s,a) \in \mathcal{S} \times \mathcal{A}_s$. The transition function looks a bit confusing and complicated at first sight, it is a compact way to write down all possible cases. 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 2.1.7. For all state-action pairs the transition probability are either 0 or 1, the next states and the rewards are deterministic for a chosen action:

$$p(s'; s, a) = \mathbf{1}_{s\in\Delta} \cdot \mathbf{1}_{s'=s} + \mathbf{1}_{s\notin\Delta} \cdot \mathbf{1}_{s'=s+a}.$$

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

$$r(s,a) = 0 \quad \text{if } s \text{ is terminal and otherwise} \quad r(s,a) = g(s+a).$$

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

We next formulate a simple supply optimisation problem in the setting of Markov decision processes, an ice vendor who has to dedice 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 2.1.10. (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 buy a discrete amount $0, 1, 2, \ldots, M$ of ice cream. Throughout the day, a random amount of ice cream is consumed, 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 then 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 order 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: $\mathcal{S} = \{0, 1, \ldots, M\}$. 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, $\mathcal{A}_s = \{0, 1, \ldots, M - s\}$ and $\mathcal{A} = \{0, \ldots, M\}$. 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 $\{0, 1, \ldots, M\}$. 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 $f : \mathcal{S} \to \mathbb{R}_+$, 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 $o : \mathcal{A} \to \mathbb{R}_+$ for the production/purchase cost of the products and $k : \mathcal{S} \to \mathbb{R}_+$ 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 $R : (\mathcal{S} \times \mathcal{A}_s) \times \mathcal{S} \to \mathbb{R}$ with

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

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

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

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 get the state-action-state transition probabilitites as

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

and the reward expectations as

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

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 needed to solve optimal control problems in the MDP setting.

Definition 2.1.11. (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

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

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

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

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

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

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

In the stationary cases we typically write 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. Lecture 7

As noted earlier, stochastic bandit models can be seen as special case of 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. In fact, there is another way of linking stochastic bandits to MDPs. The learning process using a learning strategy can also be seen as an MDP played with a non-Markovian policy as follows. 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 $R_1, ..., R_n$ 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.

There is a good reason that MDPs carry the word Markov. For Markov policies they are indeed Markov (reward) processes. The state-action process (S, A) is Markov so that together with the rewards the process (S, A, R) is a Markov reward process.

Proposition 2.1.12. (Markov property)
If $\pi \in \Pi_S$, then $(S_t, A_t)_{t\in\mathbb{N}}$ is a time-homogeneous Markov chain on $\mathcal{S} \times \mathcal{A}$ with two-step transitions

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

Proof. The proof is a computation with the path probabilities from (2.4). Since we only care for state-actions (S_t, A_t) we use the short-hand notation $p(s'\,;\, s, a)$ instead of $p(\{s'\} \times \mathcal{R}\,;\, s, a)$.

Plugging-in yields

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

Finally, if $\varphi_t=\varphi$ 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. □

Proposition 2.1.13. (Markov reward property)
If $\pi\in\Pi_S$ then $(S_t,A_t,R_t)_{t\in\mathbb{N}}$ satisfies the Markov reward process property

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

with time-homogeneous state/reward transition probabilities

$$p_{(s,a),(s',a',r')}=p(s',r\,;\,s,a)\pi(a'\,;\,s').$$

Proof.

The computation is almost identical to that of the Markov property for (S,A), do it!

□

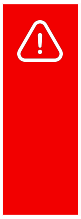
In what follows we will use the notation $\mathbb{P}_s^\pi$, $\mathbb{P}_{s,a}^\pi$ in different initialisations.

- Under $\mathbb{P}_{s,a}^\pi$ the Markov reward process (S,A,R) is started in (s,a). In state s we start with action a and then continue the Markov chain transitions and reward payoffs.
- Under $\mathbb{P}_s^\pi$ the Markov reward process (S,A,R) is started in the random initialisation $\delta_s\otimes\pi_0(\cdot\,;\,s)$. That is, in state s the first action is chosen by π_0 and then continue the Markov chain transitions and reward payoffs.

The notation is a bit annoying but the only way to avoid non-justified conditioning that one can see everywhere in the reinforcement learning literature.

2.1.3 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 trys to solve:



Given a Markov reward model and some terminal time T to defined later find a policy π that maximizes the expected sum of discounted future rewards

$$\mathbb{E}^{\pi}\Big[\sum_{t=0}^{T} R_t\Big].$$

There are three typical choices for T:

- $T = 0$ is called contextual bandit, for $|\mathcal{S}| = 1$ this is a stochastic bandit problem.
- $T = \min\{t : S_t = s'\}$ for some fixed state s'. The choice $R \equiv 1$ then counts the number of steps needed to reach s', justifying the name stochastic shortest path problems for this choice of T. We will not discuss stochastic shortest path problems in these notes.
- $T \in \mathbb{N}$ fixed. This is called finite-time stochastic control problem, we briefly touch upon finite-time problems in Section 2.4.
- The main focus of these notes is the case $T \sim \text{Geo}(1-\gamma)$, $\gamma \in (0,1)$, for some geometric random variable T independent of (S, A, R). It turns out that this situation is much simpler as there is additional Markovian structure (geometric random variables are memoryless). Given a fixed time the memoryless property says that the end is as far away as it was at the beginning. Integrating out the independent time-horizon gives

$$\begin{aligned}
\mathbb{E}^{\pi}\Big[\sum_{t=0}^{T} R_t\Big] &= \mathbb{E}^{\pi}\Big[\sum_{k=0}^{\infty}(1-\gamma)\gamma^k \sum_{t=0}^{k} R_t\Big] \\
&= (1-\gamma)\mathbb{E}^{\pi}\Big[\sum_{t=0}^{\infty}\sum_{k=t}^{\infty}\gamma^k R_t\Big] \\
&= (1-\gamma)\mathbb{E}^{\pi}\Big[\sum_{t=0}^{\infty}\frac{\gamma^t}{1-\gamma} R_t\Big] \\
&= \mathbb{E}^{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t R_t\Big]
\end{aligned} \tag{2.6}$$

  Thus, from the optimisation point of view it is equivalent to optimise the expected accumulated rewards up to a finite independent geometric time or to optimise the expected accumulated discounted rewards. Since it is more standard we will maily focus on the second point of view but keep in mind that the geometric interpretation might be more reasonable from a sampling perspective.

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

Assumption: The state space $\mathcal{S}$, the action space $\mathcal{A}$, and the set of rewards $\mathcal{R}$ are assumed to be finite. 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.

Definition 2.1.14. (Value functions)
For $\pi \in \Pi$ and $\gamma \in (0,1)$, the function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defined by

$$Q^\pi(s,a) = \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{\infty} \gamma^t R_t\Big]$$

is called state-action value function; or Q-function. The value function is defined by

$$V^\pi(s) = \mathbb{E}^\pi_s\Big[\sum_{t=0}^{\infty} \gamma^t R_t\Big] = \sum_{a\in\mathcal{A}_s} \pi_0(a\,;\,s) Q^\pi(s,a).$$

In words: The state value functions is the expected discounted total reward when $S_0 = s$ is fixed and A_0 is played according to π_0. In contrast, Q also fixes the first action $a_0 = a$. Think about chess. The state-value function shall desribe the expected success following a policy while state-action-value function is the expected success when fixing the first move.

For stationary policies the value function and state-value functions satisfy simple systems of equations. That will help us later to get our hands on V^π and Q^π.

Proposition 2.1.15. (Bellman equations for Q^π and V^π)
Suppose π is a stationary policy, then Q^π and V^π satisfy the following systems of linear equations:

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

and

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

Proof. Recall from Proposition 2.1.13 that (S,A,R) is a Markov reward process so that by (2.2)

$$\mathbb{E}^\pi_{s,a}\Big[R_1 + \gamma R_2 + \gamma^2 R_3 + ... \,\Big|\, S_1 = s', A_1 = a'\Big] = \mathbb{E}^\pi_{s',a'}\Big[R_0 + \gamma R_1 + \gamma^2 R_2 + ...\Big] = Q^\pi(s',a').$$

Thus, using the formula of total probability,

$$\begin{aligned}
&Q^\pi(s,a)\\
&= \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{\infty} \gamma^t R_t\Big]\\
&= \mathbb{E}^\pi_{s,a}\big[R_0\big] + \mathbb{E}^\pi_{s,a}\Big[\gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + ...\Big]\\
&= r(s,a) + \gamma \sum_{s'\in\mathcal{S}, a'\in\mathcal{A}_s} \mathbb{E}^\pi_{s,a}\Big[R_1 + \gamma R_2 + \gamma^2 R_3 + ... \,\Big|\, S_1 = s', A_1 = a'\Big]\mathbb{P}^\pi_{s,a}(S_1 = s', A_1 = a')
\end{aligned}$$

$$= r(s,a) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}_s} \mathbb{P}^{\pi}_{s,a}\big(S_1 = s', A_1 = a'\big) Q^{\pi}(s', a')$$
$$= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'\,;\, s, a)\pi(a'\,;\, s') Q^{\pi}(s', a').$$

The equation for V follows directly by plugging-in $V^{\pi}(s) = \sum_{a \in \mathcal{A}_s} \pi(a\,;\, s) Q^{\pi}(s,a)$ twice:

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

□

The definition of V^{π} immediately allows to compute V^{π} from Q^{π}. For stationary policies the opposite also holds true:

Corollary 2.1.16. Given a stationary policy $\pi \in \Pi_S$, the following relations between the state- and action-value functions hold:

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

Proof. This follows directly from the Bellman equation for Q plugging-in the definition $V^{\pi}(s) = \sum_{a \in \mathcal{A}_s} \pi(a\,;\, s) Q^{\pi}(s,a)$ for V. □

We will later see that linking Q and V is crucial, in particular computing Q from V via Corollary (2.1.16).

The classical theory of optimising Markov decision processes is very much about functional analysis. One of the most important theorem from basic functional analysis is Banach's fixed point theorem:

Theorem 2.1.17. (Banach fixed-point theorem)
Let $(U, ||\cdot||)$ be a Banach space, i.e. a complete normed vector space, and $T : U \to U$ a contraction, i.e. there exists $\lambda \in [0, 1)$ such that

$$\|Tu_1 - Tu_2\| \leq \lambda \, \|u_1 - u_2\|$$

for all $u_1, u_2 \in U$. Then

(i) there exists a unique fixed point, i.e. $u^* \in U$ such that $Tu^* = u^*$; and

(ii) for arbitrary $u_0 \in U$, the sequence $(u_n)_{n \in \mathbb{N}}$ defined by

$$u_{n+1} = Tu_n = T^{n+1} u_0$$

converges in U to u^*.

The Banach fixed point theorem 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 theory is the observation that contractions appear very naturally and lead to solution algorithms. For that sake we will always use the Banach space $(X, ||\cdot||_{\infty})$ consisting of $X := \{v : \mathcal{S} \to \mathbb{R}\}$ equipped with the maximum-norm. Since we assume that $\mathcal{S}$ is finite, X is nothing but $\mathbb{R}^{|\mathcal{S}|}$

where a vector is written as a function (mapping index to coordinate value). Similarly, we define $Y := \{q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}\}$ equipped with the supremum norm which is then nothing but the Banach space $\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$.

Definition 2.1.18. (Bellman expectation operator)
Given a Markov decision model and a stationary policy π the operators

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

mapping X into X and

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

mapping Y into Y are called Bellmann expectation operators.

There is a bit of ambiguity by denoting both operators by T. Since the number of arguments differs it will always be clear from the context which operator is meant. The operators might look familiar. We have actually already proved in Proposition 2.1.15 that value functions are fixed points for Bellman expectation operators. Combined with the simple contractivity property with respect to the maximums norm the first major theorem follows:

Theorem 2.1.19. Both Bellman expectation operators are contractions with contraction constant γ. Their unique fixed points are the value functions V^π and Q^π, i.e. $T^\pi V^\pi = V^\pi$ and $T^\pi Q^\pi = Q^\pi$.

Proof. We already proved the fixed point claims. Let us check the contraction property. Plugging-in yields

$$\begin{aligned} ||T^\pi v_1 - T^\pi v_2||_\infty &= \gamma \max_s \Big| \sum_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} \pi(a\,;\,s)p(s'\,;\,s,a)\big(v_1(s') - v_2(s')\big)\Big| \\ &\leq \gamma ||v_1 - v_2||_\infty \underbrace{\sum_{a \in \mathcal{A}_s} \pi(a\,;\,s) \underbrace{\sum_{s' \in \mathcal{S}} p(s'\,;\,s,a)}_{=1}}_{=1} \\ &= \gamma ||v_1 - v_2||_\infty. \end{aligned}$$

Please check yourself the same computation for the second Bellman operator. □

Since Bellman's expectation equations are linear systems they can in principle be solved by linear algebra methods. Since the linear operators are also contractions the linear systems $T^\pi v = v$ (resp. $T^\pi q = q$) have unique solutions. Still, it might be more reasonable to solve using Banach's fixed point iteration by iterating $v_{n+1} = T^\pi v_n$ for some starting vector v_0 (resp. $q_{n+1} = T^\pi q_n$ for some starting matrix q).

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 2.1.20. (Optimal policy & optimal value function)
For a given Markov decision model the following quantities will be of central importance:

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

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

is called optimal (state) value function.

(ii) The function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ 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.

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

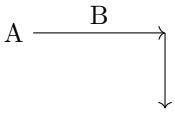
$$V^{\pi^*}(s) = V^*(s), \quad s \in \mathcal{S},$$

is called optimal policy.

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 no necessarily hold for infinite Markov decision models. We will see later that optimality of policies could have been defined alternatively by $Q^{\pi^*}(s,a) = Q^*(s,a)$ for all state-action pairs. We stick to the notion in terms of V as this is more common in the literature.

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. A priori it is absolutely not clear if there is one policy that is optimal for all starting states. It is also not clear if there is any way of characterising such a policy that leads to an algorithm. It is also not clear if that policy is very complicated. In fact, Bellman's results show that there is actually a stationary, even deterministic, policy that can be characterised by solving (non-linear) equations.

Lecture 8



We now turn towards the second operator of Bellman, the optimality operator. As for the expectation operator we work on $X = \{v : \mathcal{S} \to \mathbb{R}\}$ and $Y := \{q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}\}$. As long as we assume the Markov decision model to be finite X is nothing but $\mathbb{R}^{|\mathcal{S}|}$ and Y is $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$. To make X and Y Banach spaces we will fix the respective maximum-norms.

Definition 2.1.21. (Bellman optimality operators)
For a given Markov decision model we define the following operators:

(i) The non-linear system of equations

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

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

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

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 $T^* : V \to V$ 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 distinguised 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. To get a first little hand on the operator please do the following exercise.

All Bellman operators are monotone, i.e. if $u_1 \leq u_2$ it also holds that $T^\pi u_1 \leq T^\pi u_2$ and $T^* u_1 \leq T^* u_2$. The same holds for the matrix-valued operators T^π and T^*.

Similarly to the expectation operators also the optimality operators are contractions:

Theorem 2.1.22. Bellman's optimality operators are contractions and, thus, have unique fixedpoints.

Proof. To deal with the operators a simple inequality from analysis will be used:

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

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

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

Otherwise, the role of f and g is reversed. With $v_1, v_2 \in X$, the Bellman optimality operator and the triangle inequality yield

$$\begin{aligned}
\|T^* v_1 - T^* v_1\|_\infty &= \max_{s \in \mathcal{S}} |T^* v_1(s) - T^* v_2(s)| \\
&\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in S} p(s' \,;\, s,a) |v_1(s') - v_2(s'| \\
&\leq \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \gamma \sum_{s' \in S} p(s' \,;\, s,a) ||v_1 - v_2||_\infty \\
&= \gamma \, \|v_1 - v_2\|_\infty .
\end{aligned}$$

Hence, T^* is a contraction on $X = \{v : \mathcal{S} \to \mathbb{R}\}$ equipped with the maximum norm and thus has a unique fixed point. Next, we show the same for T^* on $Y = \{q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}\}$: With $q_1, q_2 \in Y$,

the second Bellman operator and the triangle inequality yield

$$\begin{aligned}
\|T^*q_1 - T^*q_2\|_\infty &= \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}_s} |T^*q_1(s,a) - T^*q_2(s,a)| \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in A} |\gamma \sum_{s'\in S} p(s'\,;\, s,a)\big(\max_{a'\in\mathcal{A}_s} q_1(s',a') - \max_{a'\in\mathcal{A}_s} q_2(s',a')\big)| \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in A} \gamma \sum_{s'\in S} p(s'\,;\, s,a) \max_{a'\in\mathcal{A}_s} |q_1(s',a') - q_2(s',a')| \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in A} \gamma \sum_{s'\in S} p(s'\,;\, s,a) \max_{s'\in\mathcal{S}} \max_{a'\in\mathcal{A}_s} |q_1(s',a') - q_2(s',a')| \\
&= \gamma \, \|q_1 - q_2\|_\infty \,.
\end{aligned}$$

□

We now come to the most important result of this section. The optimal value functions are uniquely characterised as fixed points of Bellman's optimality operators.

Theorem 2.1.23. The optimal value functions are the unique fixed points of Bellman's optimality operators, i.e. $T^*V^* = V^*$ and $T^*Q^* = Q^*$. Furthermore, it holds that $V^*(s) = \max_{a\in\mathcal{A}_s} Q^*(s,a)$.

Proof. It was shown previously that the operators are contractions on a Banach space, thus, have unique fixed points by Banach's fixed point theorem. Thus, the proof is complete if it can be shown that Q^* and V^* solve the optimal Bellman equations.

For didactic reasons we proceed in two steps. We first prove the theorem for simplified value functions that only take into account stationary policies, i.e. $\bar{Q}^*(s,a) = \sup_{\pi\in\Pi_S} Q^\pi(s,a)$ and $\bar{V}^*(s) = \sup_{\pi\in\Pi_S} V^\pi(s)$, as the proof is simpler. In a second part we prove the claim for all policies. There is an important consequence: There is no need to study non-stationary policies, the best expected reward can already obtained using only stationary policies!

Part 1 (stationary policies): We first prove that $\bar{Q}^*$ solves the optimal Bellman equation and next deduce the equation for $\bar{V}^*$.

Bellman equation for $\bar{Q}^*$: We proceed similarly to the proof of Bellman's expectation equation (see the proof of Lemma 2.1.16). For didactic reasons let us first suppose that π is stationary. Then we get from Proposition 2.1.15

$$\begin{aligned}
\sup_{\pi\in\Pi_S} Q^\pi(s,a) &\overset{2.1.15}{=} \sup_{\pi\in\Pi_S} \Big(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}_{s'}} p(s';s,a)\pi(a';s')Q^\pi(s',a') \Big) \\
&= r(s,a) + \gamma \sup_{\pi\in\Pi_S} \sum_{s'\in\mathcal{S}} \sum_{a'\in\mathcal{A}_{s'}} p(s';s,a)\pi(a',s')Q^\pi(s',a') \\
&= r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s';s,a) \max_{a'\in\mathcal{A}_{s'}} \sup_{\pi\in\Pi_S} Q^\pi(s',a').
\end{aligned}$$

The last equality needs a bit of explanation. The inequality „$\leq$“ is easily:

$$\sum_{a'\in\mathcal{A}_{s'}} \pi(a',s')Q^\pi(s',a') \leq \sum_{a'\in\mathcal{A}_{s'}} \pi(a',s') \sup_\pi Q^\pi(s',a') \leq \max_{a'\in\mathcal{A}_{s'}} \sup_\pi Q^\pi(s',a') \underbrace{\sum_{a\in\mathcal{A}_{s'}} \pi_0(a',s')}_{=1}.$$

For „$\geq$“ we argue as follows. Taking the supremum over all deterministic stationary policies only gives a lower bound. The advantag is that now there is a finite sum and a supremum over a finite set. We show that in that case supremum and sum can be exchanged. The claim follows. To justify let us check that

$$\sup_{\pi\in\Pi_S^D} \sum_{s\in\mathcal{S}} h(s,\pi(s)) = \sum_{s\in\mathcal{S}} \sup_{\pi\in\Pi_S^D} h(s,\pi(s))$$

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

$$\sup_{\pi\in\Pi_S^D}\sum_{s\in\mathcal{S}} h(s,\pi(s)) < \sum_{s\in\mathcal{S}}\sup_{\pi\in\Pi_S^D} h(s,\pi(s)),$$

and chose $\delta>0$ such that $\sum_{s\in\mathcal{S}}\sup_{\pi\in\Pi_S^D} h(s,\pi(s)) - \delta > \sup_{\pi\in\Pi_S^D}\sum_{s\in\mathcal{S}} h(s,\pi(s))$. 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

$$\begin{aligned}\sum_{s\in\mathcal{S}}\sup_{\pi\in\Pi_S^D} h(s,\pi(s)) &\leq \sum_{s\in\mathcal{S}} h(s,\pi^*(s)) + \varepsilon\\ &< \sup_{\pi\in\Pi_S^D}\sum_{s\in\mathcal{S}} h(s,\pi(s)) + \varepsilon\\ &\leq \sum_{s\in\mathcal{S}}\sup_{\pi\in\Pi_S^D} h(s,\pi(s)) + \varepsilon - \delta.\end{aligned}$$

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

Bellman equation for $\bar{V}^*$: Let us first prove $\max_{a\in\mathcal{A}_s}\bar{Q}^*(s,a) = \bar{V}^*(s)$:

$$\begin{aligned}\max_{a\in\mathcal{A}_s}\bar{Q}^*(s,a) &= \max_{a\in\mathcal{A}_s}\sup_{\pi\in\Pi_S} Q^\pi(s,a)\\ &= \sup_{\pi\in\Pi_S}\max_{a\in\mathcal{A}_s} Q^\pi(s,a)\\ &\overset{(*)}{=} \sup_{(\pi_t)_{t\geq1}}\sup_{\pi_0} V^\pi(s)\\ &= \sup_{\pi\in\Pi_S} V^\pi(s)\\ &= \bar{V}^*(s).\end{aligned}$$

Checking $(*)$ is not hard: „$\leq$“ is trivial, since the deterministic policy only charging a is smaller or equal to the supremum over all policies. „$\geq$“ follows from the inequality

$$V^\pi(s) = \sum_{a\in\mathcal{A}_s}\pi_0(a\,;\,s)Q^\pi(s,a) \leq \max_{a\in\mathcal{A}_s} Q^\pi(s,a)\underbrace{\sum_{a\in\mathcal{A}_s}\pi_0(a\,;\,s)}_{\leq 1} = \max_{a\in\mathcal{A}_s} Q^\pi(s,a).$$

Plugging-in twice into the equation for $\bar{Q}^*$ yields the claim for $\bar{V}^*$:

$$\begin{aligned}T^*\bar{V}^*(s) &= \max_{a\in\mathcal{A}_s}\Big\{r(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s'\,;\,s,a)\bar{V}^*(s')\Big\}\\ &= \max_{a\in\mathcal{A}_s}\Big\{r(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s'\,;\,s,a)\max_{a'\in\mathcal{A}_{s'}}\bar{Q}^*(s',a')\Big\}\\ &= \max_{a\in\mathcal{A}_s}\bar{Q}^*(a,s)\\ &= \bar{V}^*(s)\end{aligned}$$

Part 2 (non-stationary policies): Back to the non-stationary policies, the argument is similar only requires an additional computation to avoid the Markov reward property. First recall that the formular of total probability yields

$$Q^\pi(s,a) = r(s,a) + \gamma\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}_{s'}} p(s';s,a)\pi_0(a'\,;\,s')\mathbb{E}^\pi_{s,a}\Big[\sum_{t=1}^\infty\gamma^t R_t\Big|S_1=s',A_1=a'\Big]$$

for arbitrary policy. The expectation does not simplify as for stationary policies (Markov reward property), but it simplifies similarly well. If $\tilde{\pi}$ denotes the policy that is shifted by one index (we only forget the first step), i.e.

$$\tilde{\pi}_t(a\,;\,s_0,a_0,...,s_t) = \pi_{t+1}(a\,;\,s,a,s_0,a_0,...,s_t),\quad t\geq 0,$$

than we still get

$$\mathbb{E}^{\pi}_{s,a}\Big[\sum_{t=1}^{\infty}\gamma^t R_t\Big|S_1=s',A_1=a'\Big]=\gamma Q^{\tilde{\pi}}(s',a'). \tag{2.7}$$

Before checking (2.7) let us see how to finish the proof almost in the same way as above.

$$\begin{aligned}\sup_{\pi\in\Pi}Q^{\pi}(s,a)&=r(s,a)+\gamma\sup_{\tilde{\pi}}\sup_{\pi_0}\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}_{s'}}p(s';s,a)\pi_0(a',s')Q^{\tilde{\pi}}(s',a')\\&=r(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s';s,a)\max_{a'\in\mathcal{A}_{s'}}\sup_{\pi\in\Pi}Q^{\pi}(s',a').\end{aligned}$$

Again, the argument is that the stationary policy $\pi^*(a;s)=\operatorname{argmax}_{a\in\mathcal{A}_s}\sup_\pi Q^\pi(s,a)$ dominates all other policies because for all other policies $\tilde{\pi}$ it holds that

$$\sum_{a'\in\mathcal{A}_{s'}}\pi_0(a',s')Q^{\tilde{\pi}}(s',a')\le\sum_{a'\in\mathcal{A}_{s'}}\pi_0(a',s')\sup_{\tilde{\pi}}Q^{\tilde{\pi}}(s',a')\le\max_{a'\in\mathcal{A}_{s'}}\sup_{\tilde{\pi}}Q^{\tilde{\pi}}(s',a')\underbrace{\sum_{a\in\mathcal{A}_{s'}}\pi_0(a',s')}_{=1}.$$

The argument for V^* is exactly the same that we have seen for stationary policies above. To finish the proof we still need to check (2.7). As in the proof of the Markov reward property for stationary policies define the shifted process

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

and the new probability measure $\tilde{\mathbb{P}}:=\mathbb{P}^{\pi}_{s,a}(\cdot\,|\,S_1=s',A_1=a')$. On the probability space $(\Omega,\mathcal{F},\tilde{\mathbb{P}}^{\pi})$, the process $(\tilde{S}_t,\tilde{A}_t)_{t\ge0}$ is an MDP started in (s',a'), transition function p, and policy $(\tilde{\pi}_t)_{t\in\mathbb{N}_0}$. To see why, one only needs to compute the path probabilities:

$$\begin{aligned}&\tilde{\mathbb{P}}(\tilde{S}_0=s_0,\tilde{A}_0=a_0,\tilde{R}_0=r_0,...,\tilde{S}_t=s_t,\tilde{A}_t=a_t,\tilde{R}_t=r_t)\\&=\frac{\mathbb{P}^{\pi}_{s,a}(S_1=s_0,A_1=a_0,R_1=r_0,...,S_{t+1}=s_t,A_{t+1}=a_t,R_{t+1}=r_t,S_1=s',A_1=a')}{\mathbb{P}^{\pi}_{s,a}(S_1=s',A_1=a')}\\&=\frac{\delta_{(s',a')}(s_0,a_0)\cdot p(s_0\,;\,s,a)\cdot\pi_1(a_0;s,a,s_0)\cdot\prod_{i=0}^{t}p(s_{i+1},r_i\,;\,s_i,a_i)\cdot\pi_i(\{a_i\}\,;\,s,a,s_0,a_0,...,s_{i-1},a_{i-1},s_i)}{p(s_0\,;\,s,a)\cdot\pi_1(a_0;s,a,s_0)}\\&=\delta_{(s',a')}((s_0,a_0))\cdot\prod_{i=2}^{t+1}p(s_{i+1},r_{i+1}\,;\,s_i,a_i)\pi_i(a_i\,;\,s,a,s_0,a_0,...,s_{i-1},a_{i-1},s_i)\\&=\delta_{(s_0,a_0)}((s',a'))\cdot\prod_{i=1}^{t}p(s_i,r_{i-1}\,;\,s_{i-1},a_{i-1})\tilde{\pi}_i(a_i\,;\,s_0,a_0,...,s_{i-1},a_{i-1},s_i)\end{aligned}$$

Hence, $\tilde{\mathbb{E}}^{\tilde{\pi}}\Big[\sum_{t=0}^{\infty}\gamma^t\tilde{R}_t\Big]=V^{\tilde{\pi}}(s')$.

□

The proof showed that in order to find an optimal policy it is not needed to look at all policies, only stationary policies are needed. This is much more convenient, as the set of stationary policies is much smaller compared to all policies. Even more, there is a deterministic stationary optimal policy, a policy that assigns only one optimal action to every state. Yet, the situation is even better! The existence is not just theoretical but there is a way to get hands on such a policy by solving a system of equations, the Bellman optimality equation.

Definition 2.1.24. (Greedy policy)

Given a function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ 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 greedy(q) instead of π_q. If π is greedy with respect to a q_v obtained from the V-Q-transfer operator

$$q_v(s,a) := r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'\,;\,s,a) v(s'), \quad s \in \mathcal{S}, a \in \mathcal{A}_s, \tag{2.8}$$

then we also write π_v.

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. Here is a lemma that shows how to deal with greedy policies in the context of Bellman operators.

Lemma 2.1.25. Suppose π_q is a greedy policy obtained from a q, then $T^*q = T^{\pi_q}q$. If q is obtained from a vector v through (2.8) then it also holds that $T^*v = T^{\pi_v}v$.

Proof. Both claims readily follow from the definition of the Bellman operators:

$$\begin{aligned} T^{\pi_q}q(s,a) &= r(s,a) + \sum_{s' \in \mathcal{A}_s} \sum_{a' \in \mathcal{A}_{s'}} p(s'\,;\,s,a)\pi_q(s'\,;\,a')q(s',a') \\ &= r(s,a) + \sum_{s' \in \mathcal{A}_s} p(s'\,;\,s,a) \max_{a' \in \mathcal{A}_{s'}} q(s',a') = T^*q(s,a), \end{aligned}$$

where for the second equality we used the definition of the greedy policy. The same holds if q is obtained from a vector v:

$$\begin{aligned} T^{\pi_v}v(s) &= \sum_{a \in \mathcal{A}_s} \pi_v(a\,;\,s)\Big(r(s,a) + \sum_{s' \in \mathcal{A}_s} p(s'\,;\,s,a)v(s')\Big) \\ &= \max_{a \in \mathcal{A}_s} \Big(r(s,a) + \sum_{s' \in \mathcal{S}} p(s'\,;\,s,a)v(s')\Big) = T^*v(s), \end{aligned}$$

where we used the definition of the Bellman operators and the definition of the greedy policy in the second equality. □

As a consequence we learn how solving the Bellman state-action optimality equation yields an optimal policy.

Theorem 2.1.26. (Dynamic programming algorithm)
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. Since v is only a vector while q is a matrix it sounds first plausible (up to now) to solve $T^*v = v$ and then transfer to q_v. Solving the non-linear equation is the expensive step, transfering to q not. We will later see that typically we try to avoid transferring from V to Q but in this chapter on stochastic control that sounds like a good idea.

Proof. Suppose we have a solution q of Bellmann's state-action optimality equation. By uniqueness q equals Q^*. It remains to show $V^* = V^{\pi_q}$, then π_q is optimal by definition. But this follows from the previous lemma and the fact that $q = Q^*$:

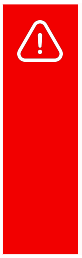
$$Q^* = T^*Q^* = T^{\pi_{Q^*}}Q^* = T^{\pi_q}Q^*.$$

Uniqueness of Bellman's expectation operator gives $Q^* = Q^{\pi_q}$. Finally, using the definition of the greedy policy and the relations $V^*(s) = \max_{a\in\mathcal{A}_s} Q^*(s,a)$ and $V^\pi(s) = \sum_{a\in\mathcal{A}_s} \pi(a\,;\,s)Q^\pi(s,a)$ between V and Q gives

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

We have thus proved that $V^* = V^{\pi_q}$ which shows that π_q is optimal. □

Let us summarise the findings made so far, the core of everything that will later be called valued-based algorithms.



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

From now on we will only focus on finding stationary optimal policies!

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 2.1.27. 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. $||V^{\pi^n} - V^*||_\infty \to 0$. 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 2.1.26). 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 $V^\pi(s')$ or $Q^\pi(s,a)$ from all other $Q^\pi(s',a')$. 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

2.2 Basic tabular value iteration algorithm

We have understood the problem and its ingredients and have learnt 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 approach of this section is to develop methods that approximate the optimal value functions. As seen in the previous section, the Banach fixed point theorem gives not only the existence of an 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 learnt 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. The algorithm is called a tabular algorithm as a table (here the vector V) is updated repeatadly.

Algorithm 6: Value iteration

Data: Accuracy $\varepsilon > 0$
Result: Approximation $V \approx V_*$, policy $\pi \approx \pi^*$
Initialize $V \equiv 0, V_{\text{new}} \equiv 0$
$\Delta := 1$
while $\Delta > \varepsilon$ **do**
  set $V := V_{new}$
  for $s \in \mathcal{S}$ **do**
    $$V_{\text{new}}(s) = \underbrace{\max_{a\in\mathcal{A}_s}\left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s';s,a)V(s')\right\}}_{(T^*V)(s)}$$
  end
  $\Delta := \max_{s\in\mathcal{S}}(|V_{\text{new}}(s) - V(s)|)$
end
return $V := V_{\text{new}}$ and the greedy policy with respect to V

Theorem 2.2.1. The value iteration Algorithm 6 terminates and the terminal vector satisfies $\|V - V^*\|_\infty \leq \frac{\gamma\,\varepsilon}{1-\gamma}$.

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

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

for $n \to \infty$. Now suppose the algorithm terminated after n steps, i.e. $V = v_n$ and $||v_n - v_{n-1}||_\infty <$

ε. Using the contraction property of T^* yields, for $m \in \mathbb{N}$,

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

Using continuity of the norm yields

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

Now the termination condition $||v_n - v_{n-1}||_\infty < \varepsilon$ implies the claim. □

The approximate value function from Algorithm 6 is clearly not equal to the optimal value function V^*. Hence, the effect on transferring to a policy needs to be analysed.

Definition 2.2.2. 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 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

$$Q(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'\,;\, s,a) V(s'). \tag{2.9}$$

We now do the same and use (2.9) to define from the approximation V of V^* an approximate Q-function. Then the corresponding greedy policy is ε-optimal:

Theorem 2.2.3. Suppose V is the output of Algorithm 6 and Q is obtained from V using the transfer operator (2.9). Then the greedy policy π_Q is $\frac{2\varepsilon\gamma}{1-\gamma}$-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:

$$\begin{aligned} T^*V(s) &= \max_{a\in\mathcal{A}_s}\Big\{r(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s'\,;\,s,a)V(s')\Big\} \\ &= \max_{a\in\mathcal{A}_s}\{Q(s,a)\} \\ &= \sum_{a\in\mathcal{A}_s}\pi_Q(a\,;\,s)Q(a,s) \\ &= \sum_{a\in\mathcal{A}_s}\pi(a\,;\,s)\Big(r(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s';\,s,a)V(s')\Big) \\ &= T^\pi V(s) \end{aligned}$$

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

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

Rearranging this equation yields

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

In the proof of the previous theorem we have already shown that $\|V - V^*\|_\infty \le \frac{\gamma}{1-\gamma}\,\|v_n - v_{n-1}\|_\infty$. Finally, using the terminal condition of Algorithm 6 yields

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

This proves the claim. □

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

Definition 2.2.4. Suppose $(V, ||\cdot||)$ is a normed space. For a sequence (y_n) in V with limit v^* we say the convergence is of order $\alpha > 0$ if there exists a constant $K < 1$ for which

$$\|y_{n+1} - y^*\| \le K\,\|y_n - y^*\|^\alpha\,, \quad n \in \mathbb{N}. \tag{2.10}$$

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 2.2.5. For all initialisations the convergence in Algorithm 6 (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 nth update of the iteration $v_n = T^*v_{n-1}$. The linear convergence rate follows directly from the fixed point property of T^*:

$$\|v_{n+1} - V^*\|_\infty = \|T^*v_n - T^*V^*\|_\infty \le \gamma\,\|v_n - V^*\|_\infty\,, \quad n \in \mathbb{N}. \tag{2.11}$$

In fact, the rate cannot be better as the following example shows. If Algorithm 6 is initialised with $v_0 := V^* + k\mathbf{1}$, it holds that

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

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

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.

2.3 Basic policy iteration (actor-critic) 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^π (or V^π) for the currently best known policy π.
- Policy improvement: Improve the best known policy by taking $\pi' = \text{greedy}(Q^\pi)$.

In contrast to value iteration the approach is more clever, it uses much more understanding of the optimal control problem. While value iteration is called a value-based method (only the value function is used, the policy is only obtained in the end) policy iteration is a policy-based method, the approach works directly uses the policy.

The approach is called an **actor-critic method** as it alternates between two steps. The critic evaluates the policy (computes the value function) which then the actor uses to improve the quality of the policy.

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

2.3.1 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 now.

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

$$T^\pi v(s) = \sum_{a \in \mathcal{A}_s} \pi(a\,;\, s)\Big(r(s,a) + \sum_{s' \in \mathcal{S}} p(s'\,;\, s,a)v(s')\Big).$$

and 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 Belman operator than takes the form

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

The equation looks complicated but ($\mathcal{S}$ is assumed finite) is just a system of linear equatinos 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

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

with $P^\pi \in \mathbb{R}^{|\mathcal{S}| \cdot |\mathcal{S}|}$ 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 2.3.1. The (affine) linear equation $V = r^\pi + \gamma P^\pi V$ has a unique solution. Hence, V^π can be computed explicitly as

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi. \tag{2.12}$$

Proof. This follows immediately as the Bellman expectation operator is a contraction (compare the exercise after Theorem 2.1.22). To compute the solution is straightforward linear algebra operation:

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

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

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 $(T^\pi)^n v_0 \to V^\pi$ holds for any initialisation v_0. Implemented as an algorithm we obtain Algorithm 7.

Algorithm 7: Iterative policy evaluation (Naive)

Data: Policy $\pi \in \Pi_S$, $\varepsilon > 0$
Result: Approximation $V \approx V^\pi$
Initialize $V \equiv 0, V_{\text{new}} \equiv 0$
$\Delta := 1$
while $\Delta > \varepsilon$ **do**
  Set $V = V_{\text{new}}$
  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 2.3.2. Algorithm 7 terminates and $\|V - V^\pi\|_\infty < \frac{\gamma\,\varepsilon}{1-\gamma}$.

Proof. The proof is identical to the proof of Theorem 2.2.1 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. □

Algorithm 7 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 8. 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 8: Iterative policy evaluation (totally asynchroneous 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 $s \in S$ **do**
    $v := V(s)$
    $V(s) := \underbrace{\sum_{a\in\mathcal{A}_s} \pi(a\,;\,s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r\,;\,s, a)\,[r + \gamma V(s')]}_{T^\pi V(s) = (r_\pi + P_\pi)V(s)}$
    $\Delta := \max(\Delta, |v - V(s)|)$
  end
end

Try to prove convergence of the totally asynchroneous policy evaluation algorithm

(without termination) to V^π. To do so enumerate $\mathcal{S}$ as $s_1, ..., s_K$ 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 $\bar{T}^\pi := (T_{s_K}^\pi \circ ... \circ T_{s_1}^\pi)(V)$, 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 $\bar{T}^\pi$, hence, without termination the algorithm converges to V^π.

Algorithms in which coordinates s are treated differently are called asynchroneous algorithms. In fact, there are other versions of the algorithm where coordinates are not sweeped in order but randomly. We will come back to such asynchroneous 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

2.3.2 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

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

The improvement is called strict if

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

If π is not optimal, there always exists a strict improvement, e.g. the optimal policy. We now want do 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 $s \in S$ to a particular action. For this we look at the action-value function of a stationary policy $\pi \in \Pi_S$. Recalling that

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

it becomes apparent that

$$\max_{a \in \mathcal{A}_s} Q^\pi(s, a) \geq V^\pi(s). \tag{2.13}$$

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 2.1.24 this suggests to use the greedy policy π_{Q^π} induced by the Q-function of the current policy π which then leads to the simple policy improvement algorithm. The improvement of the greedy policy improvement is a special case of the policy improvement theorem.

Theorem 2.3.3. (Policy improvement theorem)
Let $\pi, \pi' \in \Pi_S$ be two stationary policies, then the following hold:

(i) If

$$V^\pi(s) \leq \sum_{a \in \mathcal{A}_s} \pi'(a\,;\, s) Q^\pi(s, a), \quad \forall s \in \mathcal{S}, \tag{2.14}$$

Algorithm 9: Greedy policy improvement

Data: policy $\pi \in \Pi_S$, 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 $a \in \mathcal{A}_s \setminus \{a^*(s)\}$ **do**
    $\pi'(a\,;\, s) = 0$
  end
end

then π' is an improvement of π.

(ii) If there is a strict inequality in (2.14) for some state s then the improvement is strict.

(iii) For every policy $\pi \in \Pi_S$ the greedy policy obtained from Q^π improves π.

On a heuristic level the theorem is trivial. If a stationary policy π' is better for one step (in all states) then (by the Markov property) the policy will also lead to a larger reward if it is used in all time-steps. But then the value function for π' is bigger than that for π.

Proof. (i) Using definitions assumptions and the definition of Bellman expectation operators we get

$$\begin{aligned} V^\pi(s) &\le \sum_{a \in \mathcal{A}_s} \pi'(a\,;\, s) Q^\pi(s,a) \\ &= \sum_{a \in \mathcal{A}_s} \pi'(a\,;\, s)\Big(r(s,a) + \sum_{s' \in \mathcal{S}} p(s'\,;\, s,a) V^\pi(s')\Big) = T^{\pi'} V^\pi(s). \end{aligned}$$

Monotonicity of Bellman operators allows us to iterate the equation to obtain

$$V^\pi(s) \le T^{\pi'} V^\pi(s) \le T^{\pi'} T^{\pi'} V^\pi(s) \le \ldots \le \lim_{k \to \infty} (T^{\pi'})^k V^\pi(s).$$

By Banach's fixed point theorem the iterations of $T^{\pi'}$ converge uniformly (thus pointwise) to the unique fixed point of $T^{\pi'}$, which is $V^{\pi'}$. Thus, $V^\pi(s) \le V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

(ii) For strict inequalities the above steps imply strict inequalities.

(iii) Checking the greedy policy update satisfies the one-step improvement property is easy. With $a^*(s) \in \arg\max_{a \in \mathcal{A}_s} Q^\pi(s,a)$, the greedy policy π' satisfies the improvement condition:

$$V^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi(a\,;\, s) Q^\pi(s,a) \le \max_{a \in \mathcal{A}_s} Q^\pi(s,a) \underbrace{\sum_{a \in \mathcal{A}_s} \pi(a\,;\, s)}_{=1} = \sum_{a \in \mathcal{A}_s} \pi'(a\,;\, s) Q^\pi(s,a)$$

Hence, (i) implies the claim. □

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

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

Proof. It follows from Lemma 2.1.16 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 $Q^{\pi'} = Q^* = Q^\pi$ (and both are optimal). □

Corollary 2.3.5. For a non-optimal policy $\pi \in \Pi_S$ and the greedy policy $\pi' = \text{greedy}(Q^\pi)$ obtained from Q^π is a strict policy improvement.

Proof. The claim follows directly by Lemma 2.3.4. □

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 2.3.6. A policy $\pi \in \Pi_S$ is called

- soft, if it fulfils
$$\pi(a\,;\,s) > 0 \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}_s,$$
- 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.
$$\pi(a\,;\,s) = \begin{cases} (1-\varepsilon) + \frac{\varepsilon}{|\mathcal{A}_s|} & : a = a^*(s) \\ \frac{\varepsilon}{|\mathcal{A}_s|} & : a \text{ otherwise} \end{cases},$$
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 2.3.7. 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:

$$\begin{aligned} V^\pi(s) &= \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{a\in\mathcal{A}_s} Q^\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{a\in\mathcal{A}_s} Q^\pi(s,a) + \sum_a \pi(a\,;\,s)Q^\pi(s,a) \\ &= \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{a\in\mathcal{A}_s} Q^\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a\,;\,s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1-\varepsilon} Q^\pi(s,a) \\ &\leq \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{a\in\mathcal{A}_s} Q^\pi(s,a) + (1-\varepsilon)\max_a Q^\pi(s,a) \\ &= \sum_{a\in\mathcal{A}_s} \pi'(a\,;\,s)Q^\pi(s,a). \end{aligned}$$

□

2.3.3 Policy iteration algorithms (tabular actor-critic)

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 ist π^*. Here is an illustration of the procedure:

$$\pi_0 \nearrow V^{\pi_0} \searrow \pi_1 \nearrow V^{\pi_1} \searrow \pi_2 \nearrow \cdots \searrow \pi^*$$

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 10: Greedy exact policy iteration (actor-critic)

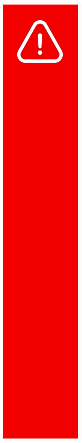
Data: initial policy $\pi \in \Pi_S$, initial value function V
Result: optimal policy $\pi^* \in \Pi_S^D$
initialise arbitrarily V_{new}, π_{new}
stop = *False*
while *stop* = *False* **do**
  Policy evaluation (critic): Obtain V^π by computing (2.12).
  set $Q^\pi(s,a) = \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r\,;\,s,a)\,[r+\gamma V^\pi(s')]$ for all a, s
  Policy improvement (actor): Obtain the improved greedy policy π_{new} from Q^π
  if $Q^{\pi_{new}} = Q^\pi$ **then**
    stop = *True*
  end
end
return $\pi^* = \pi$

the matrix inversion (2.12).

Theorem 2.3.8. (Greedy exact policy iteration)
Started in any policy the policy iteration Algorithm 16 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 2.3.5 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 ($|\Pi_S^D| = |\mathcal{S}|^{|\mathcal{A}|} < \infty$) the algorithm has to terminate in finitely many steps. By Lemma 2.3.4 the termination of the algorithm, thus $V^\pi = V^{\pi'}$, implies that π' is optimal. □

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 explicite evaluation of V^π by a few steps of the Banach fixed point iteration corresponding to T^π, compare Algorithms 7 or 8. Other examples will be discussed in Chapter 3. All algorithms alternating between value function estimation a policy improvement (not necessarily greedy)

$$\pi_0 \nearrow \underbrace{\hat{V}^{\pi_0}}_{\approx V^{\pi_0}} \searrow \pi_1 \nearrow \underbrace{\hat{V}^{\pi_1}}_{\approx V^{\pi_1}} \searrow \pi_2 \nearrow \cdots \searrow \pi$$

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 11: Generalised policy iteration (actor-critic) paradigm

Data: initial policy π
Result: optimal policy π^* (or approximation of π^*)
while *not converged* **do**
  Policy evaluation (critic): Obtain estimates for $\hat{V}^\pi$ and/or $\hat{Q}^\pi$ from **some** algorithm.
  Policy improvement (actor): Obtain (hopefully improved) policy π_{new} by **some** algorithm.
  Set $\pi = \pi_{new}$.
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 3.2.1 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 2.3.9. An ε-soft policy π^* is called ε-soft optimal if

$$V^{\pi^*}(s) = \sup_{\pi\ \varepsilon\text{-soft}} V^\pi(s) =: \tilde{V}^*(s), \quad \forall s \in \mathcal{S}.$$

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 2.3.10. **(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 2.3.8) 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,

$$\begin{aligned}
\sum_{a\in\mathcal{A}_s} \tilde{\pi}(a\,;\, s)\tilde{r}(s, a) &= \sum_{a\in\mathcal{A}_s} \Big(\frac{\pi(a\,;\, s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1-\varepsilon}\Big)\Big((1-\varepsilon)r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{b\in\mathcal{A}_s} r(s, b)\Big) \\
&= \sum_{a\in\mathcal{A}_s} \Big(\pi(a\,;\, s) - \frac{\varepsilon}{|\mathcal{A}_s|}\Big)r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{b\in\mathcal{A}_s} r(s, b)\underbrace{\sum_{a\in\mathcal{A}_s}\frac{\pi(a\,;\, s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1-\varepsilon}}_{=1} \\
&= \sum_{a\in\mathcal{A}_s} \pi(a\,;\, s)r(s, a).
\end{aligned}$$

Similarly:

$$\begin{aligned}
\sum_{a\in\mathcal{A}_s} \tilde{\pi}(a\,;\, s)\tilde{p}(y\,;\, s, a) &= \sum_{a\in\mathcal{A}_s} \Big(\frac{\pi(a\,;\, s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1-\varepsilon}\Big)\Big((1-\varepsilon)p(y\,;\, s, a) + \frac{\varepsilon}{|\mathcal{A}_s|}\sum_{b\in\mathcal{A}_s} p(y\,;\, s, b)\Big) \\
&= \sum_{a\in\mathcal{A}_s} \pi(a\,;\, s)p(y\,;\, s, a)
\end{aligned}$$

Combining the above yields

$$\begin{aligned}\tilde{T}^{\tilde{\pi}}V^{\pi}(s) &= \sum_{a\in\mathcal{A}_s}\tilde{\pi}(a\,;\,s)\Big[\tilde{r}(s,a)+\gamma\sum_{y\in\mathcal{S}}\tilde{p}(y\,;\,s,a)V^{\pi}(y)\Big]\\ &= \sum_{a\in\mathcal{A}_s}\pi(a\,;\,s)\Big[r(s,a)+\gamma\sum_{y\in\mathcal{S}}p(y\,;\,s,a)V^{\pi}(y)\Big]\\ &= T^{\pi}V^{\pi}(s) = V^{\pi}(s).\end{aligned}$$

Since the fixed point is unique it follows that $\tilde{V}^{\tilde{\pi}} = V^{\pi}$ and, as f is bijective,

$$\sup_{\tilde{\pi}\in\tilde{\Pi}_S}\tilde{V}^{\tilde{\pi}}(s) = \sup_{\pi\ \varepsilon\text{-soft}}V^{\pi}(s), \tag{2.15}$$

For the Q-functions we obtain

$$\begin{aligned}\tilde{Q}^{\tilde{\pi}}(s,a) &= \tilde{r}(s,a)+\gamma\sum_{y\in\mathcal{S}}\tilde{p}(y\,;\,s,a)V^{\pi}(y)\\ &= (1-\varepsilon)\Big(r(s,a)+\gamma\sum_{y\in\mathcal{S}}p(y\,;\,s,a)V^{\pi}(y)\Big)\\ &\qquad+\frac{\varepsilon}{|\mathcal{A}_s|}\sum_{b\in\mathcal{A}_s}\Big(r(s,b)+\gamma\sum_{y\in\mathcal{S}}p(y\,;\,s,b)\Big)\\ &= (1-\varepsilon)Q^{\pi}(s,a)+\frac{\varepsilon}{|\mathcal{A}_s|}\sum_{b\in\mathcal{A}_s}\Big(r(s,b)+\gamma\sum_{y\in\mathcal{S}}p(y\,;\,s,b)\Big),\end{aligned}$$

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 2.3.8 convergence of exact policy iteration with greedy policy updates the proof can be finished:

$$\sup_{\pi\ \varepsilon\text{-soft}}V^{\pi}(s) \overset{(2.15)}{=} \sup_{\tilde{\pi}\in\tilde{\Pi}_S}\tilde{V}^{\tilde{\pi}}(s) = \tilde{V}^{\tilde{\pi}^*}(s) = \lim_{n\to\infty}\tilde{V}^{\tilde{\pi}_n}(s) = \lim_{n\to\infty}V^{\pi_n}(s)$$

□

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.

2.3.4 Relation of policy iteration and value iteration

There is an important simple observation that we should always keep in mind as it reappears later in the discussion of SARSA and Q-learning. Value iteration is a short-cut of policy iteration, it combines the two phases of policy iteration in one. Why is that?

Recall that

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

and

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

Thus, $T^* v$ is nothing but $T^{greedy(v)} v$. But that means that iterating T^* is nothing but starting in some vector v and repeatadly chosing the greedy policy and computing its Bellman expectation operator once.

2.4 Stochastic control in finite time

So far we discussed stochastic control problems with geometrically distributed random time horizon (equivalently, infinite time horizon with discounted rewards). In this section the time horizon will be a fixed deterministic time T. The geometric time horizon is a simpler problem as the forgettfullness of geometric random variables leads to stationary optimal policies. This is different for deterministic time horizon, optimal policies $(\pi_t^*)_{t \leq T}$ will be non-stationary! [3]

2.4.1 Setting and dynamic programming

As in the previous section we will consider $(S_t, A_t, R_t)_{t \leq T}$ with the only difference that time is restricterd to $D = \{0, \dots, T\}$ for some $T \in \mathbb{N}$ fixed. Action and state spaces as well as transition probabilities p remain unchanged, policies $\pi = (\pi_t : t \in D)$ are defined as before. The finite-time stochastic contol problem consists in optimising the state value function $\mathbb{E}_s^\pi [\sum_{t=0}^{T-1} R_t]$ over all policies.

Example 2.4.1. An example to keep in mind is the ice vendor who has a three month summer season (Germans also love ice cream during winter, most others don't). The optimal 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 led to 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.

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 $T - t$ Markov kernels. The definition of the state value function for T-step MDPs is as follows.

Definition 2.4.2. For any policy $\pi \in \Pi_t^T$ the functions (vectors) defined by

[3] at some point also include a terminal payout (which now is 0)

$V^\pi_{T,T} \equiv 0$ and

$$V^\pi_{t,T} : \mathcal{S} \to \mathbb{R}, \quad s \mapsto \mathbb{E}^{\tilde{\pi}}_s\Big[\sum_{i=0}^{T-t-1} R_i\Big] =: \mathbb{E}^\pi_{S_t=s}\Big[\sum_{i=t}^{T-1} R_i\Big],$$

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 $i = 0, ..., T-t-1$. The function $V^\pi_{0,T}$ is called state value function.

Typically the time-state value function is defined as $V^\pi_{t,T}(s) = \mathbb{E}[\sum_{i=t}^{T-1} R_i \,|\, S_t = s]$. 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 $V^\pi_{t,T}$ is the total reward gained after time t. We also have to redefine the state-action value function

Definition 2.4.3. For any policy $\pi \in \Pi^T_t$ the functions (matrices) defined by $Q^\pi_{T,T} \equiv 0$ and

$$Q^\pi_{t,T} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad (s,a) \mapsto \mathbb{E}^{\tilde{\pi}}_{s,a}\Big[\sum_{i=0}^{T-t-1} R_i\Big] =: \mathbb{E}^\pi_{S_t=s,A_t=a}\Big[\sum_{i=t}^{T-1} R_i\Big],$$

for $t < T$ are called time-state-action value functions. The shifted policy $\tilde{\pi}$ is defined as in the previous definition. The function $Q^\pi_{0,T}$ is called state-action value function.

From now on we will drop the subscript T. We will just write V^π_t and Q^π_t, except in situation in which the emphasise lies on T. 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 2.1.16, we also get a lemma that discribes the relation between the state value function and the state-action value function for a fixed policy:

Proposition 2.4.4. Given a Markovian policy $\pi = (\pi_t)_{t \in \leq T}$ and a T-step Markov decision problem. Then the following relation between the state and state-action value function hold

$$V^\pi_t(s) = \sum_{a \in \mathcal{A}_s} \pi_t(a\,;\, s) Q^\pi_t(s,a),$$
$$Q^\pi_t(s,a) = r(s,a) + \sum_{s' \in \mathcal{S}} p(s'\,;\, s,a) V^\pi_{t+1}(s')$$

for all $t < T$. In particular, the Bellman expectation equations (backwards recursions)

$$V^\pi_t(s) = \sum_{a \in \mathcal{A}} \pi_t(a\,;\, s)\Big[r(s,a) + \sum_{s' \in \mathcal{S}} p(s'\,;\, s,a) V^\pi_{t+1}(s')\Big],$$
$$Q^\pi_t(s,a) = r(s,a) + \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_s} p(s'\,;\, s,a) \pi_{t+1}(a'\,;\, s') Q^\pi_{t+1}(s',a')$$

hold for $t < T$.

Proof. Expressing V_t in terms of Q_t is a matter of definition of $\mathbb{E}_s$, just as in the infinite horizon case. For $t < T$ we can use the Markov reward property of (S, A, R) and the formular of total probability to derive the recursions. This is exactly the same as in the proof of Proposition

2.1.15. Let's add the proof for completeness. Recall from Proposition 2.1.13 [4] that (S, A, R) is a Markov reward process so that by (2.2)

$$\mathbb{E}^{\pi}_{s,a}\Big[R_1 + R_2 + ... + R_{T-t-1} \,\Big|\, S_1 = s', A_1 = a'\Big] = \mathbb{E}^{\tilde{\pi}}_{s',a'}\Big[R_0 + R_1 + ... + R_{T-t-2}\Big] = Q^{\pi}_{t+1}(s', a').$$

Thus, using the formula of total probability,

$$\begin{aligned}
&Q^{\pi}_t(s,a)\\
&= \mathbb{E}^{\tilde{\pi}}_{s,a}\Big[\sum_{i=0}^{T-t-1} R_i\Big]\\
&= \mathbb{E}^{\tilde{\pi}}_{s,a}\big[R_0\big] + \mathbb{E}^{\tilde{\pi}}_{s,a}\Big[R_1 + R_2 ... + R_{T-t-1}\Big]\\
&= r(s,a) + \sum_{s'\in\mathcal{S}, a'\in\mathcal{A}_s} \mathbb{E}^{\tilde{\pi}}_{s,a}\Big[R_1 + R_2 + ... + R_{T-t-1} \,\Big|\, S_1 = s', A_1 = a'\Big]\mathbb{P}^{\tilde{\pi}}_{s,a}(S_1 = s', A_1 = a')\\
&= r(s,a) + \sum_{s'\in\mathcal{S}, a'\in\mathcal{A}_s} \mathbb{P}^{\tilde{\pi}}_{s,a}\big(S_1 = s', A_1 = a'\big)Q^{\pi}_{t+1}(s', a')\\
&= r(s,a) + \sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}_{s'}} p(s'\,;\, s,a)\tilde{\pi}_1(a'\,;\, s')Q^{\pi}_{t+1}(s', a')\\
&= r(s,a) + \sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}_{s'}} p(s'\,;\, s,a)\pi_{t+1}(a'\,;\, s')Q^{\pi}_{t+1}(s', a').
\end{aligned}$$

The equation for V follows directly by plugging-in $V^{\pi}_t(s) = \sum_{a\in\mathcal{A}_s}\pi_t(a\,;\,s)Q^{\pi}_t(s,a)$ twice:

$$\begin{aligned}
V^{\pi}_t(s) &= \sum_{a\in\mathcal{A}_s}\pi_t(a\,;\,s)Q^{\pi}_t(s,a)\\
&= \sum_{a\in\mathcal{A}_s}\pi_t(a\,;\,s)\Big(r(s,a) + \sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}_{s'}} p(s'\,;\,s,a)\pi_{t+1}(a'\,;\,s')Q^{\pi}_{t+1}(s',a')\Big)\\
&= \sum_{a\in\mathcal{A}_s}\pi_t(a\,;\,s)\Big(r(s,a) + \sum_{s'\in\mathcal{S}} p(s';\,s,a)V^{\pi}_{t+1}(s')\Big).
\end{aligned}$$

□

It is important to note that the system of equations is different from discounted MDP problems. First, the discounting factor dissapears as $\gamma = 1$ and, secondly, the linear systems are actually recursions that gradually simplify the system towards the terminal conditions $Q^{\pi}_T \equiv 0$, $V^{\pi}_T \equiv 0$. Or, reversed, starting with the zero-vector (resp. zero-matrix) a backward induction allows to compute V_t and Q_t recursively. There is no system of equations that needs to be solved!

Similarly to the discounted setting the optimal value functions are defined, now depending on the remaining time-horizon:

Definition 2.4.5. For any $t \le T$ and a given Markov decision problem

- the function $V^*_t : \mathcal{S} \to \mathbb{R}$ that takes values

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

  is called optimal time-state value function.

- the function $Q^*_t : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ that takes values

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

[4] at some point rewrite proposiiton for Markov policies, works equally

is called optimal time-state-action value function.

- a policy π^* that satisfies

$$V_t^{\pi}(s) = V_t^*(s), \quad s \in \mathcal{S}, t \leq T,$$

is called optimal.

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 2.4.6. 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. The proof is the same as in the infinite setting. First write

$$V_t^*(s) = \sup_{\pi} V_t^{\pi} = \sup_{\pi} \sum_{a \in \mathcal{A}_s} \pi_t(a\,;\, s) Q_t^{\pi}(s,a)$$

and then try to maximise the righthand side by playing greedy with respect to Q_t^*. □

For discounted infinite time horizion problems we could now show that it is suifficent 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 dependents stronly on the time horizion up to the closing date. It is clear that we would like to order with a different stategy 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 fixpoint relation in the value function. We can formulate the following theorem:

Theorem 2.4.7. (Dynamic programming algorithm)
Suppose $v_t : \mathcal{S} \to \mathbb{R}$, $t = 0, ..., T$, 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 $t = 0, ..., T$. Similarly, if $q_t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $t = 0, ..., T$, 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 $t = 0, ..., T$. Most importantly, an optimal (non-stationary) policy is given by the greedy policy

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

Proof. Suppose q solves the optimality recursion and π^q is the greedy policy. By uniqueness q equals Q^*. It remains to show $V^* = V^{\pi^q}$, then π^q is optimal by definition. But this follows from the previous lemma and the fact that $q = Q^*$. First for Q:

$$\begin{aligned} Q_t^*(s,a) &= r(s,a) + \sum_{s' \in \mathcal{S}} p(s';s,a) \max_{a' \in \mathcal{A}_{s'}} Q_{t+1}^*(s',a') \\ &= r(s,a) + \sum_{s' \in \mathcal{S}} p(s';s,a) \sum_{a' \in \mathcal{A}_s} \pi_{t+1}^q(a'\,;\,s') Q_{t+1}^*(s',a'). \end{aligned}$$

This means that also the sequence of matrices Q^* solves the Bellman expectation equation for π^q. Since the recursions have a unique solution it follows that $Q_t^* = Q_t^{\pi^q}$ for all $t \leq T$. Finally, using the definition of the greedy policy and the relations $V_t^*(s) = \max_{a \in \mathcal{A}_s} Q_t^*(s,a)$ and $V_t^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi_t(a\,;\,s) Q_t^\pi(s,a)$ between V and Q gives

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

We have thus proved that $V_t^* = V_t^{\pi_q}$ for all $t \leq T$ which shows that π_q is optimal. □

The key tool to solve finite time horizion MDPs follows from Theorem 2.4.7 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

$$Q_{T-2}^*(s,a) = \underbrace{r(s,a)}_{\text{todays profit}} + \underbrace{\sum_{s' \in \mathcal{S}} p(s'\,;\,s,a) \max_{a' \in \mathcal{A}_{s'}} Q_{T-1}^*(s',a')}_{\text{next days profit if next day is played optimally}}$$

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.

Finite-time control problems are similar to the stochastic bandits. If all quantities are known explicitly the problem is essentially trivial, the optimal policy is obtained by chosing largest Q-values and those can be computed without any effort. For discounted infinite-time problems at least the non-linear equation $T^*Q = Q$ must be solved which is not a trivial task. As for stochastic bandits the situation becomes interesting once the Q-values cannot be computed explicitly but must be estimated by interacting with the environment. This will be the subject of the next chapter.

5

2.4.2 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 must be computed from the terminal condition. Theorem 2.4.7 immediately translates into a simple

Algorithm 12: 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**

$$Q_t^\pi(s,a) := r(s,a) + \sum_{s'\in\mathcal{S}} p(s'\,;\, s,a) V_{t+1}^\pi(s')$$

    end

$$V_t^\pi(s) := \sum_{a\in\mathcal{A}_s} \pi_t(a\,;\, s) Q_t^\pi(s,a)$$

  end
end

optimal control algorithm.

[5] Sara: Wuerfelbeispiel

Algorithm 13: 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 $a \in \mathcal{A}_s$ **do**

$$Q_t^*(s,a) := r(s,a) + \sum_{s' \in \mathcal{S}} p(s' \,;\, s,a) V_{t+1}^*(s')$$

    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 3

Simulation based dynamic programming methods

In this chapter we turn from stochastic control theory to reinforcement learning, sample based algorithms to solve stochastic control problems without assuming explicit knowledge on the environment dynamic described by p. To understand what this means let us discuss major drawbacks of using Bellman equations that we try to overcome in this chapter:

- Dynamic programming algorithms are based on iterating Bellman operators the operators must be known andaccessible. Most importantly, the transition function p must be known explicity. 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 model (this is p) underlying a decision problem but the learning agent has no access to the physical model. For example one can play a computer game without knowing the transitions, one can drive a car without knowing the outcome of a steering decision.
- There is theoretical access to all ingredients of the decision problem but state (and/or action) space might be too big to be repeatadly 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 standard dynamic programming algorithms such as value or policy iteration, both algorithms need explicit knowledge of the transitions p 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 3.0.1. An algorithm to learn optimal policies 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 sample all appearing random variables.

In the situation of stochastic bandits (MDPs with one time-step) all algorithms were model-free. In fact, knowing all probabilities explicitliy allows to compute the action values Q_a. Then one only needs to compare the action values to find the best action.

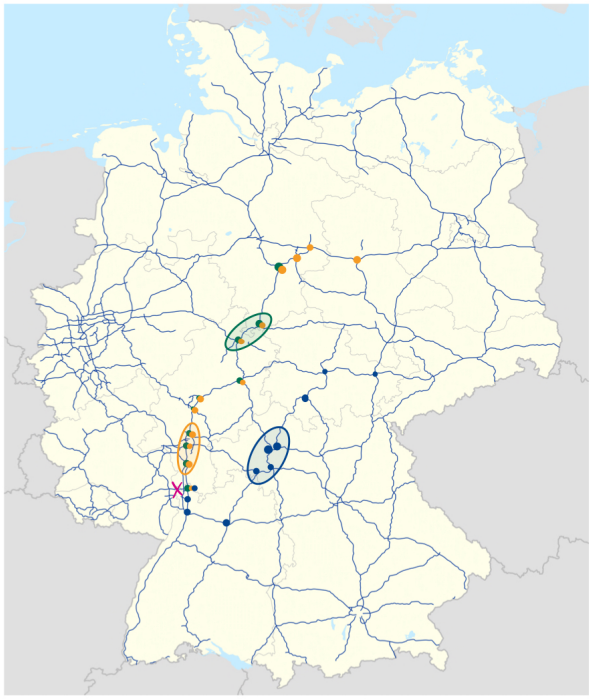
In this chapter we develop simulation based dynamic programming algorithms for the tasks we addressed with the dynamic programming algorithms based on Bellman's equations:

- 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 approach is very inefficient but helps us to sort some ideas. What turns out to be more efficient are stochastic versions of the Banach fixedpoint theorem, so-called stochastic approximation algorithms.

3.1 A guiding example

We start the discussion with a guiding example that should provide some intuition why the mathematical concepts introduced below are completely intuitive[1]. Suppose we have two high-way navigation systems that are supposed to bring us back to Mannheim, say to Kreuz Mannheim. Which one is better?



Temporal differences of 1, 2, and 3 steps

To formulate the problems the state space consists of autobahn drive-ups. The actions are the possible decisions of the navigation systems. The MDP has a terminating state, Kreuz Mannheim. The rewards are the times it takes to reach the next state. Since the terminal state is reached sufficiently fast, a discounting faction 0.99 has not much effect ($.99^{20} \approx .82$) so the total discounted reward is a good approximation of the time it takes from state s (e.g. Berlin Dreick Funkturm) to Kreuz Mannheim. To compare two navigation systems π and π' we should compute $V^{\pi}(s)$ and $V^{\pi'}(s)$ for all cities. There are two approaches that are natural.

- Drive the car repeatedly from every drive-up towards 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 that were computed before. Looking at the visualisation this is clearly not clever, many trajectories have common sections. Ideas that try to reuse samples for different estimates are called

[1]For 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

bootstrapping. The simplest thought 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.

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 simulation based (or sample based) dynamic programming methods.

There is a simple thougth from learning. If there are two unbiased estimates $\hat{X}$ and $\bar{X}$, then $\alpha\hat{X} + (1-\alpha)\bar{X}$ is also an unbiased estimate for all $\alpha \in (0,1)$. That mixture is hopefully better, e.g. it might have smaller variance. In reinforcement learing we will see the same again and again. With $\alpha = \frac{n-1}{n}$ this already appeared in (1.3) when updating the old estimate $\frac{1}{n-1}\sum_{k=1}^{n-1} X_i$ of $\mathbb{E}[X_1]$ with a further sample X_n. Now suppose you already have an estimate for the optimal Q-matrix Q^* based on a number of observations (s_i, a_i, r_i, s_i'). Assume there is an additional observation (s, a, r, s'). Do you have an idea how you would improve your estimate of $Q^*(s,a)$? If you have an idea that might likely be what we will later get to know under the name Q-learning. Think about the example of the German highway system (or how you would walk through Mannheim). You already have an idea of how to optimally navigate and get a new observation (s, a, r, s'), you walk from one block to another and record the time. Of course, you will not completely ignore your old knowledge (if for instance there is a big traffic jam) but you want to keep that new information (there can be a big traffic jam) into your estimate.

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. Nonetheless, the example captures the main ideas of this chapter.

3.2 Monte Carlo policy evaluation and control

The aim of this short section is to get a glimps 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). The value function are expectations by definition expectations, thus, the most obvious model-free variant for policy evaluation is Monte Carlo estimate with N indepent trajectories of the MDP.

Definition 3.2.1. A trajectory (S_t, A_t, R_t) of a Markov decision process is called a **rollout**. If several rollouts are used they are indexed with a superscript.

Here is the natural Monte carlo estimator of $V^\pi(s)$. If (S_t^i, A_t^i, R_t^i) are independent rollouts of the MDP started in s under policy π, then define

$$\hat{V}^\pi(s) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\sum_{t=0}^{T} \gamma^t R_t^i}_{=\hat{V}_i^\pi(s)} \tag{3.1}$$

for some large T and N. This naive approach is extremely inefficient as rollouts (S^i, A^i, R^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 rollout (S, A, R) of the Markov decision process.

3.2.1 First visit Monte Carlo policy evaluation

Before we proceed note that the time-truncation in the estimate of V^π by (3.1) 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 terminating state. In that case one might replace T by ∞ which in the case of termination is equal to a finite sum up to the termination time.

Recall from (2.6) that for discounted MDPs it holds that $V^\pi(s) = \mathbb{E}_s[\sum_{t=0}^{T} R_t]$ for an independent geometrically distributed random time-horizon. An unbiased estimator is then given by

$$\hat{V}^\pi(s) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\sum_{t=0}^{T^i} R_t^i}_{=\hat{V}_i^\pi(s)}$$

where the rollouts are independent and additionally the $T^i \sim \text{Geo}(1-\gamma)$ are independent samples of the independent random time-horizon.

For some reason the unbiased estimator is not very common in computer science literature. People prefer to truncate at some fixed time T (introducing bias) or assume the MDP is terminating. Terminating MDP is also not a very reasonable assumption as termination depends strongly on the policy and not only on the environment. For most non-trivial MDP one can chose a policy that only alternates between states and will not terminate.

Here is the first simple boostrapping idea, relying on the strong Markov property of Markov chains.

The strong Markov property of an MDP (every (reward) 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 only one rollout correlated estimates of V^π for many different starting points. Similarly, restarting at first visits of a given state-action pair (s, a) it is possible to extract from one rollout correlated estimates of Q^π for several different state-action pairs.

The idea is easy to implement. Simulate a number of rollouts and for every rollout 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 (1.3) a memory efficient way to avoid storing all past rewards.

Algorithm 14: First visit Monte Carlo policy evaluation of V^π

Data: Policy $\pi \in \Pi_S$, initial condition μ
Result: Approximation $V \approx V^\pi$
Initialize vectors $V_0 \equiv 0$ and $N \equiv 1$
$n = 0$
while *not converged* **do**
  $n = n + 1$
  Sample $T \sim \text{Geo}(1-\gamma)$.
  Sample s_0 from μ.
  Generate trajectory $(s_0, a_0, r_0, s_1, ...)$ started in μ until time-horizon $2T$ using policy π.
  for $t = 0, 1, 2, ..., T$ **do**
    if $s_t \notin \{s_0, s_1, ...s_{t-1}\}$ **then**
      $v = \sum_{k=t}^{T+t} r_k$
      $V_n(s_t) = \frac{1}{N(s_t)} v + \frac{N(s_t)-1}{N(s_t)} V_{n-1}(s_t)$
      $N(s_t) = N(s_t) + 1$
    end
    else
      $V_n(s_t) = V_{n-1}(s_t)$
    end
  end
  return V_n
end

By the law of large numbers the first visit Monte Carlo algorithm converges almost surely to the true value function (resp. 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. Of course we need to impose some stopping condition for the algorithm, but without stopping condition almost sure convergence is satisfied as long as all states (resp. state-action pairs) are visited infinitely often which is the case if all states (resp. state-action pairs) can be reached by the state-action Markov chain.

Theorem 3.2.2. The first visit Monte Carlo algorithms satisfy the following convergence properties for $s \in \mathcal{S}$ and $a \in \mathcal{A}$:

- If $N(s) \to \infty$ almost surely, then $V_n(s) \to V^\pi(s)$ almost surely.
- If $M(s, a) \to \infty$ almost surely, then $Q_n(s, a) \to Q^\pi(s, a)$ 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.

Algorithm 15: 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 matrices $Q_0 \equiv 0$ and $M \equiv 1$
$n = 0$
while *not converged* **do**
  $n = n + 1$
  Sample $T \sim \text{Geo}(1-\gamma)$.
  Sample s_0 from μ
  Generate trajectory $(s_0, a_0, r_0, s_1, ...)$ started in μ until time-horizon $2T$ using policy π.
  for $t = 0, 1, 2, ..., T$ **do**
    if $(s_t, a_t) \notin \{(s_0, a_0), (s_1, a_1), ...(s_{t-1}, a_{t-1})\}$ **then**
      $q = \sum_{k=t}^{T+t} r_k$
      $Q_n(s_t, a_t) = \frac{1}{M(s_t,a_t)} q + \frac{M(s_t,a_t)-1}{M(s_t,a_t)} Q_{n-1}(s_t, a_t)$
      $M(s_t, a_t) = M(s_t, a_t) + 1$
    end
    else
      $Q_n(s_t, a_t) = Q_{n-1}(s_t, a_t)$
    end
  end
end

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 rollout the sampled discounted rewards are clearly strongly dependent, thus, not allowing the use of the law of large numbers. If for instance one reward has an unreasonably large deviation from the mean then the estimates of the total reward will be large for several estimates. The corresponding (biased) algorithm is called **every visit Monte Carlo**.

3.2.2 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 simulations. 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 $\hat{Q}^\pi$ by performing the first visit Monte Carlo procedure and then improve the policy greedily by playing greedily from the matrix $\hat{Q}^\pi$. Unfortunately, there is an exploration problem, the infinite visitation condition of the first visit Monte Carlo theorem must not be satisfied. Suppose we start with a greedy policy with weights

on suboptimal actions. The first visit algorithm will only estimate the suboptimal Q-values. As in the stochastic bandit setting everything now depends on the initialisation of Q (which is typically the zero matrix) in relation to the unknown distribution of rewards. If for instance the rewards are all positive then the greedy update will again only play suboptimal actions. Just as for stochastic bandits, in order to force more exploration one will exchange the greedy policy update from policy iteration by ε-greedy policy update. Now there will be an error by replacing the original policy but the algorithm might converge to something more reasonable. Since the

Algorithm 16: Monte Carlo generalised ε-greedy policy iteration

Data: initial policy π
Result: approximation of π^*
while *not converged* **do**
  Policy evaluation: Obtain estimates $\hat{Q}^\pi$ using first visit Monte Carlo.
  Policy improvement: Obtain the ε-greedy policy π_{new} from $\hat{Q}^\pi$
  Set $\pi = \pi_{new}$.
end
return π

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 chosing ε-greedy policy update. As for bandits one should expect decreasing ε to be advantegous.

Simulate generalised ε-greedy policy iteration with first visit Monte Carlo evalution 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 schemes[2].

3.3 Stochastic fixed point iterations

In order to justify the convergence sample based 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 $x(n+1) = \widehat{F(x(n))}$ and still obtain convergence to the unique fixed point of F? The answer is generally no, the approximation errors are typically too large. Instead, the update scheme $x(n+1) = (1-\alpha(n))x(n) + \alpha(n)\widehat{F(x(n))}$ can be proved to converge to x_* if the approximation errors are unbiased and small enough and the so-called step-sizes α decay with a certain rate. Note that $\alpha \equiv 1$ gives the Banach iteration with approximations, $\alpha \equiv 0$ a constant sequence. Both do not converge so the it its non-trivial how to chose $\alpha(n)$.

The theory discussed below is typically called stochastic approximation. Since we will usually be interested in applying the theory to find fixed points we prefer to speak of stochastic fixed point algorithms.

We will first discuss in the most simple settings the ideas of Robbins and Monro[3] to find roots of

[2] Check here for some results: https://arxiv.org/pdf/1903.05926.pdf
[3] H. Robbins and S. Monro: A stochastic approximation method. The Annals of Mathematical Statistics, 22:400–407, 1951.

functions that can only be evaluated with approximation errors and then turn towards general stochastic fixed point iterations that we will later turn into sample based dynamic programming algorithms to solve stochastic control problems in a model-free way.

We will now deal with simple algorithms that are able to find zeros/fixed points for given functions. Since the theory goes back to Robbins and Monro in the 50s of the last century such algorithms are called Robbins-Monro algorithms. The original Robbins-Monro algorithm adresses 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 stochastic 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

$$x_{n+1} = x_n - \alpha_n y_n, \tag{3.2}$$

where α_n are so-called step-sizes specified below and y_n are stochastic estimates of $G(x_n)$. In contrast to the Newton approximation algorithm or other deterministic algorithm this algorithm is stochastic, thus, justifying the name stochastic approximation. We start with the most basic stochastic approximation theorem. The theorem will not be used for reinforcement learning but it allows us to understand easily where the so-called Robbins-Monro conditions on the step-sizes α_n stem from.

Theorem 3.3.1. (Simplest Robbins-Monro algorithm)
Let $(\Omega, \mathcal{F}, (\mathcal{F}_n), \mathbb{P})$ be a filtered probability space on which all random variables are defined. Suppose $G : \mathbb{R} \to \mathbb{R}$ has a zero x_* and satisfies $G(x) \geq \kappa(x - x_*)$ for some $\kappa > 0$. Define recursively the stochastic process

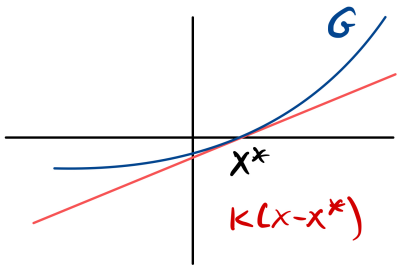
$$x_{n+1} = x_n - \alpha_n \underbrace{\big(G(X_n) + \varepsilon_n\big)}_{=:y_n}. \tag{3.3}$$

Assume that there are deterministic step-sizes (α_n) 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 that are conditionally unbiased, i.e. $\mathbb{E}[\varepsilon_n \,|\, \mathcal{F}_n] = 0$. Furthermore suppose that $\mathbb{E}[y_n^2] \leq A + B\mathbb{E}(x_n - x_*)^2$. Then $x_n \overset{L^2}{\to} x_*$ for $n \to \infty$, where $G(x_*) = 0$.

Before proving the theorem and giving an instructive example let us discuss the assumptions. The assumption on G implies that there can only be one zero. Every zero $\bar{x}$ to the right of x_* forces $\bar{x} = x_*$. In a drawing the situation looks like the graph plotted in the figure. From the drawing it is not surprising that the algorithm converges. If x_n is to the right of x_*, then (ignoring the error) the iteration will be pushed to the left as $G(x_n)$ is positive. Similarly, from the left of a zero iteration will be pushed to the right as $G(x_n)$ is negative.



Robbins-Monro assumption on G

There is a very natural class of examples. If $G = F'$, then the assumption on G is implied by what is called strong convexity of F. Strong convexity implies $F'(x) - F'(y) \geq \kappa(x - y)$ for all $x, y \in \mathbb{R}$. In that situation the theorem states convergence of the algorithm to a critical point of F. The simple Robbins-Monro theorem is a first (simple) version of convergence in L^2-sense of stochastic gradient descent towards the unique critical point (global minimum) for strongly convex functions. We will get to more refined results on stochastic gradient descent in later chapters.

Next, let us think about the step-sizes. Please keep the following in mind, that choice is almost always used.

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 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. If the errors are independent of $\mathcal{F}_n$, i.e. of the earlier recursion, than the error condition becomes $\mathbb{E}[\varepsilon_n] = 0$, the errors are unbiased.

Example 3.3.2. Here is the original setup of the Robbins-Monro paper. 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

$$y_n := f(x_n, \tilde{Y}) = G(x) + \underbrace{(f(x_n, \tilde{Y}) - G(x_n))}_{\varepsilon_n}$$

of $G(x)$. 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 is essentially the original proof of Robbins and Monro. It 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 $\lim_{n\to\infty} z_n = 0$.

Define $x_n := z_n + \sum_{k=0}^{n-1} a_k z_k - \sum_{k=0}^{n-1} c_k$. Then it holds true that $(x_n)_{\{n\in\mathbb{N}\}}$ is monotonically decreasing:

$$\begin{aligned} x_{n+1} &\leq z_n - a_n z_n + c_n + \sum_{k=0}^{n} a_k z_k - \sum_{k=0}^{n} c_k \\ &= z_n + \sum_{k=0}^{n-1} a_k z_k - \sum_{k=0}^{n-1} c_k \\ &= x_n \end{aligned}$$

As $x_n \geq -\sum_{k=0}^{n-1} c_k > -\infty$, $(x_n)_{\{n\in\mathbb{N}\}}$ converges. Moreover, since

$$0 \leq z_n = \underbrace{x_n + \sum_{k=0}^{n-1} c_k}_{\text{converges for } n\to\infty} - \sum_{k=0}^{n-1} a_k z_k, \tag{3.4}$$

it has to hold true that $\sum_{k=0}^{\infty} a_k z_k < \infty$. By assumption, $\sum_{k=1}^{\infty} a_k = \infty$ and hence, $\liminf_{n\to\infty} z_n = 0$. In order to obtain convergence to zero we still need to show that (z_n) converges. For that sake let us show that (z_n) is Cauchy. Using the above representation of z_n the Cauchy property follows because the series of the right hand side converge:

$$|z_{n+1} - z_n| \leq |x_{n+1} - x_n| + |c_n| + |a_n z_n| \to 0, \quad n \to \infty.$$

Thus, we proved $\liminf_{n\to\infty} z_n = \lim_{n\to\infty} z_n = 0$.

CD CD A positive recursion for the L^2-error.

Define $z_n = \mathbb{E}[(x_n - x_*)^2]$ and $d_n = \mathbb{E}[(x_n - x_*)(G(x_n) - G(x_*))]$. Then simple algebra leads to

$$\begin{aligned} z_{n+1} &= \mathbb{E}\big[(x_{n+1} - x_n + x_n - x_*)^2\big] \\ &= \mathbb{E}\big[(x_{n+1} - x_n)^2\big] + 2\mathbb{E}\big[(x_{n+1} - x_n)(x_n - x_*)\big] + \mathbb{E}\big[(x_n - x_*)^2\big] \\ &= \mathbb{E}\big[(\alpha_n y_n)^2\big] + 2\mathbb{E}\big[(-\alpha_n y_n)(x_n - x_*)\big] + \mathbb{E}\big[(x_n - x_*)^2\big] \\ &= \alpha_n^2 \mathbb{E}\big[y_n^2\big] - 2\alpha_n \big(\mathbb{E}\big[G(x_n)(x_n - x_*)\big] + \mathbb{E}\big[\varepsilon_n (x_n - x_*)\big]\big) + \mathbb{E}\big[(x_n - x_*)^2\big] \\ &\leq \alpha_n^2 (A + B z_n) - 2\alpha_n \big(\kappa \mathbb{E}\big[(x_n - x_*)^2\big] + \mathbb{E}\big[\mathbb{E}\big[\varepsilon_n \mid \mathcal{F}_n\big](x_n - x_*)\big]\big) + \mathbb{E}\big[(x_n - x_*)^2\big] \\ &= \alpha_n^2 e_n - 2\alpha_n d_n + z_n. \end{aligned}$$

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. □

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 different assumptions on G) 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 $F(x) = a$ (use $\tilde{G}(x) = F(x) - a$), the corresponding algorithm is

$$x_{n+1} = x_n - \alpha_n(F(x_n) - a + \varepsilon_n),$$

where y are again approximations of $G(x_n)$. Similarly, the algorithm can also be used to find fixed points (use $G(x) = F(x) - x$), the corresponding algorithm is

$$x_{n+1} = x_n - \alpha_n(F(x_n) - x_n + \varepsilon_n).$$

In fact, we will see below that for contractions F the assumptions on G of Robbins-Monro are not met forcing us to switch a sign.

We finish the first discussion of the stochastic approximation algorithms with an important example that shows that stochastic approximation can not expected to have good convergence rate:

Example 3.3.3. The simplest example is the function $G(x) = x - \mu$ which surprisingly leads to the law of large numbers. Suppose $Z_1, Z_2, ...$ is an iid sequence with mean μ and finite variance σ^2 and $G(x_n)$ is approximated by $x_n - Z_{n+1}$. Chosing $\alpha_n = \frac{1}{n+1}$ and $\mathcal{F}_n = \sigma(Z_1, ..., Z_n)$ the Robbins-Monro iteration becomes

$$x_{n+1} = x_n - \frac{1}{n+1}\big(G(x_n) + \varepsilon_n\big) = x_n + \frac{1}{n+1}\big(Z_{n+1} - x_n\big), \quad x_0 = 0,$$

with $\varepsilon_n = \mu - Z_{n+1}$. In fact, that recursion appeared before, compare (1.3). The solution is simply $x_n = \frac{1}{n}\sum_{k=1}^{n} Z_k$. Thus, if the conditions of Robbins-Monro hold, then the L^2 convergence to μ is nothing but the weak law of large numbers. Let us check the conditions of the theorem:

- G clearly satisfies the assumption.
- The assumed independence yields $\mathbb{E}[\varepsilon_n \,|\, \mathcal{F}_n] = \mathbb{E}[\mu - Z_{n+1}] = 0$.

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.

We now continue with stochastic approximations of fixed points with the ultimative goal to derive sample based approximations for the unique solution of $T^*Q = Q$. This will be the celebrated Q-learning algorithm. It was mentioned above that considering $G(x) = F(x) - x$ one could derive a stochastic approximation scheme

$$x_{n+1} = x_n - \alpha_n(F(x_n) - x_n + \varepsilon_n) = (1 - \alpha_n)x_n - \alpha_n(F(x_n) + \varepsilon_n), \quad n \in \mathbb{N},$$

for the fixedpoint of F. Unfortunately, the convergence in L^2-sense from the Robbins-Monro algorithm will not be strong enough for our purposes, additionally a contraction must not satisfy the assumptions of (a multivariate version) Robbins-Monro. Finally, we also want to allow the step-sizes to be random. Quite a lot of work will be needed to prove a fairly general stochastic fixed point theorem that is robust enough to be applied for several reinforcement learning algorithms.

Before diving into the theorem let us introduce some further notation.

In order to distinguish one-dimension and multivariate approximation schemes we write x_n for one-dimensional schemes and $x(n)$ for multivariate schemes, reserving the superscript $x_i(n)$ for the ith coordinate of the nth iterate.

For the applications we are mostly interested in max-norm contractions (such as the Bellman operators) but the theory can be carried out a bit more generally. For pseudo-contractions with respect to weighted norms.

Definition 3.3.4. For a vector $\vartheta \in \mathbb{R}^d$ with strictly positive entries the weighted maximum norm $\|\cdot\|_\vartheta$ is defined as

$$\|x\|_\vartheta = \max_{i\in\{1,\dots,d\}} \frac{|x_i|}{\vartheta_i}.$$

The most important special case is the constant vector $\vartheta = \mathbf{1}$ which leads to the usual maximum norm.

Definition 3.3.5. A function $F : \mathbb{R}^d \to \mathbb{R}^d$ is called a weighted maximum norm pseudo-contraction if there exists a $x^* \in \mathbb{R}^d$, a vector ϑ with only positive entries and a constant $\lambda \in [0, 1)$ such that:

$$\|F(x) - x^*\|_\vartheta \leq \lambda \|x - x^*\|_\vartheta, \quad x \in \mathbb{R}^d.$$

If F is a contraction with respect to $\|\cdot\|_\vartheta$ than F is clearly also a pseudo-contraction with the unique fixed point x^* (replace $x^* = F(x^*)$). In what follows we will analyse convergence of the multivariate Robbins-Monro fixed point iteration

$$\begin{aligned} x_1(n+1) &= (1-\alpha_1(n))x_1(n) + \alpha_1(n)(F_1(x(n)) + \varepsilon_1(n)) \\ &\dots = \cdots \\ x_d(n+1) &= (1-\alpha_d(n))x_d(n) + \alpha_d(n)(F_2(x(n)) + \varepsilon_d(n)) \end{aligned}$$

or, in short,

$$x_i(n+1) = (1-\alpha_i(n))x_i(n) + \alpha_i(n)(F_i(x(n)) + \varepsilon_i(n)), \tag{3.5}$$

where the subscript i refers to the dimension. In the light of the final remark of Section 3.1 the algorithm can be interpreted as keeping the old estimate of x^* with probaiblity $(1-\alpha)$ and using a newly estimated $F(x_n)$ with probability α. For pseudo-contractions we will derive conditions on the step-sizes and the error terms so that almost surely $x(n) \to x^*$ for $n \to \infty$.

Definition 3.3.6. A multivariate stochastic fixed point algorithm is called **synchroneous** if $\alpha_1(n) = ... = \alpha_d(n)$, i.e. all coordinates are updated equally in every step. Otherwise the algorithm is called **asynchroneous**. The algorithm is called totally asynchroneous if all except one step-size are zero, i.e. in every step only one coordinate is updated.

The notion should be compared to the value iteration algorithms from Section 2.3.1. The algorithms are often synchroneous but can also be run asynchroneously, compare the remark towards the Gauss-Seidel method in the end of Section 2.3.1.

For didactic reasons we first discuss 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 explictly 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 $\mathbb{R}^d$ to prove convergence of reinforcement learning algorithms.

Theorem 3.3.7. (Simple stochastic fixed point iteration on $\mathbb{R}$)
Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Suppose that

- $F : \mathbb{R} \to \mathbb{R}$ is a contraction,

- ε_n are $\mathcal{F}_{n+1}$-measurable with $\mathbb{E}[\varepsilon_n \,|\, \mathcal{F}_n] = 0$ and $\mathbb{E}[\varepsilon_n^2 \,|\, \mathcal{F}_n] \leq C$ for all $n \in \mathbb{N}$,
- the random step-sizes $\alpha_n \in [0, 1]$ are only (!) adapted with

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

Then the stochastic process defined recursively through some $\mathcal{F}_0$-measurable initial condition $x(0)$ and the recursion

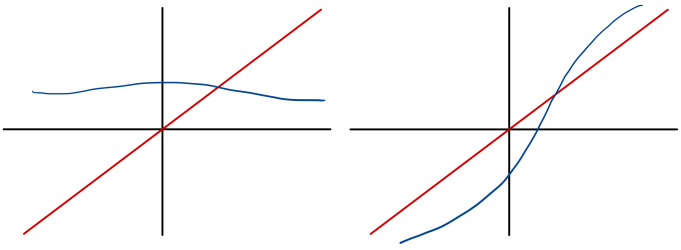
$$x_{n+1} = x_n + \alpha_n\big(F(x_n) + \varepsilon_n - x_n\big), \quad n \in \mathbb{N},$$

converges almost surely to the unique fixed point x^* of F.

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

$$x_{n+1} = F(x_n) = x_n + (F(x_n) - x_n) \approx x_n + (F(x_n) + \varepsilon_n - x_n).$$

Little surpisingly the scheme would not converge as the errors are not assumed to diminuish (only unbiased) and the scheme would fluctuate around x^* without converging. This is circumvented by decreasing the update size using α_n. 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 particulary 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 disecting line implying $F(x) - x < 0$. Similarly, $F(x) - x > 0$ if $x < x^*$. Thus, the scheme is set up such that x_n is pushed left if $x_n > x^*$ and pushed right if $x_n < x^*$. 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.

As mentioned earlier we will not prove Theorem 3.3.7, the assumption on the errors is too strong for our purposes in reinforcement learning. Here is the main theorem of this section:

Theorem 3.3.8. (Stochastic fixed point iteration for contractions on $\mathbb{R}^d$) Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Let $(x(n))$ be the sequence generated by the recursion

$$x_i(n+1) = (1 - \alpha_i(n))x_i(n) + \alpha_i(n)((F_i^n(x(n)) + \varepsilon_i(n) + b_i(n)), \quad n \in \mathbb{N}. \quad (3.6)$$

Let us pose the following assumptions:

- (F^n) all map $\mathbb{R}^d$ into itself, are pseudo-contractions for the same $||\cdot||_\vartheta$ norm

with the same x^* and $\lambda \in [0,1)$, i.e.

$$\|F^n(x) - x^*\|_\vartheta \leq \lambda \|x - x^*\|_\vartheta, \quad n \in \mathbb{N}.$$

- The non-negative stepsizes $\alpha_i(n)$ are $\mathcal{F}_n$-adapted and satisfy almost surely

$$\sum_{k=0}^{\infty} \alpha_i(k) = \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_i^2(k) < \infty$$

for all coordinates $i = 1, ..., d$.

- The errors $\varepsilon_i(n)$ satisfy the following conditions:
  - $\varepsilon_i(n)$ is $\mathcal{F}_{n+1}$-measurable,
  - $\mathbb{E}[\varepsilon_i(n)|\mathcal{F}_n] = 0$,
  - the conditional second moments satisfy

$$\mathbb{E}[\varepsilon_i^2(n) \,|\, \mathcal{F}_n] \leq A + B\|x(n)\|_\vartheta^2$$

for all coordinates $i = 1, ..., d$ and some constants $A, B \geq 0$.

- There exists a non-negative sequence of random variables (ω_n) that converges to zero almost surely such that the following holds

$$|b_i(n)| \leq \omega_n(\|x(n)\|_\vartheta + 1), \quad n \in \mathbb{N},$$

for all coordinates $i = 1, ..., d$.

Then $\lim_{n\to\infty} x(n) = x^*$ almost surely.

The theorem has plenty of assumptions. To get an overview let us check how to get back the simpler version from Theorem 3.3.7:

- $d = 1$,
- $b \equiv 0$,
- $F_n = F$ for all n, F is even a proper contraction,
- $B = 0$.

To make this section not even more horrible the proof is collected in the next section.

3.4 Proof of the general stochastic fixed point iteration

The proof is long but actually not too hard to follow if the main idea is understood. To keep it simple suppose $d = 1$ and $F(x) = \lambda x$ for some $\lambda \in (0,1)$. The fixed point is $x^* = 0$. Then the recursion simplifies to

$$x_{n+1} = (1 - \alpha_n)x_n + \alpha_n\big(\lambda x_n + \varepsilon_n\big) = (1 - \bar{\alpha}_n)x_n + \bar{\alpha}_n\bar{\varepsilon}_n$$

with $\bar{\alpha}_n = (1-\lambda)\alpha_n$ and $\bar{\varepsilon}_n = (1-\lambda)^{-1}\varepsilon_n$. This recursion is much simpler and we are going to use the supermartingale convergence theorem to prove it converges almost surely to 0 (Lemma 3.4.4 as an application of the Robbins-Siegmund theorem). The main proof of the theorem compares the multivariate recursion to this simple recursion. There are a couple of things that make everything messy. We need to deal with the different dimensions, use the growth conditions on F and the bias b, and then carefully compare with the simple recursions. The arguments are given in two steps. We first prove boundedness and then the convergence to the fixed point.

3.4.1 Proof of boundedness of the approximating sequence

Theorem 3.4.1. Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Let $(x(n))$ be the sequence generated by the recursion in Theorem 3.3.8. The assumptions for $b(n)$, $\alpha(n)$, and $\varepsilon(n)$ are as in Theorem 3.3.8, for all F^n we only assume the linear growth condition

$$\|F^n(x)\|_\vartheta \leq \lambda \|x\|_\vartheta + D$$

for some $\lambda \in (0,1)$ and $D \in \mathbb{R}$. Then the sequence (x_n) is bounded almost surely.

From Analysis we know that contractions (Lipschitz continuous functions) grow at most linearly. To recall and slightly extend to pseudo-contractions let us recall the short computation (triangle inequality):

$$\begin{aligned}\|F^n(x)\|_\vartheta &= \|F^n(x)) - x^* + x^*\|_\vartheta \\ &\leq \|F^n(x) - x^*\|_\vartheta + \|x^*\|_\vartheta \\ &\leq \lambda \|x - x^*\|_\vartheta + \|x^*\|_\vartheta \\ &\leq \lambda \|x\|_\vartheta + \underbrace{(1+\lambda)\|x^*\|_\vartheta}_{=:D}\end{aligned}$$

Thus, the assumptions of Theorem 3.4.1 are more general, the theorem will imply boundedness of the stochastic approximation scheme under the conditions of Theorem 3.3.8.

The proof of the theorem requires some foundations from stochastic recursions that we first collect.

Theorem 3.4.2. (Robbins-Siegmund Theorem)
Let $(\Omega, \mathcal{F}, (\mathcal{F}_n), \mathbb{P})$ be a filtered probability space, (Z_n), (A_n), (B_n) and (C_n) be non-negative and adapted stochastic processes, such that

$$\sum_{k=0}^{\infty} A_k < \infty \quad \text{and} \quad \sum_{k=0}^{\infty} B_k < \infty$$

almost surely. Moreover, suppose

$$\mathbb{E}[Z_{n+1} \mid \mathcal{F}_n] \leq Z_n(1 + A_n) + B_n - C_n, \quad n \in \mathbb{N}.$$

Then

- there exists an almost surely finite random variable Z_∞ such that $\lim_{n\to\infty} Z_n = Z_\infty$ almost surely,
- it holds that $\sum_{k=0}^{\infty} C_k < \infty$ almost surely.

Proof. The proof[4] is a nice application of Doob's almost sure martingale convergence theorem. Therefore, we are going to construct a supermartingale based on the stated stochastic processes.

Construction of a supermartingale (M_n).

We define the auxiliary random variables

$$\widehat{Z}_n = \frac{Z_n}{\prod_{k=0}^{n-1}(1+A_k)}, \quad \widehat{B}_n = \frac{B_n}{\prod_{k=0}^{n}(1+A_k)}, \quad \widehat{C}_n = \frac{C_n}{\prod_{k=0}^{n}(1+A_k)}$$

[4] H. Robbins and D. Siegmund. „A convergence theorem for non-negative almost supermartingales and some applications“, Optimizing methods in statistics, pages 233-257, Elsevier, 1971.

and observe that

$$\begin{aligned}\mathbb{E}[\widehat{Z}_{n+1} \mid \mathcal{F}_n] &= \Big(\prod_{k=0}^{n-1}(1+A_k)^{-1}\Big)\mathbb{E}[Z_{n+1} \mid \mathcal{F}_n] \\ &= \Big(\prod_{k=0}^{n-1}(1+A_k)^{-1}\Big)\big(Z_n(1+A_n)+B_n-C_n\big) \\ &= \widehat{Z}_n + \widehat{B}_n - \widehat{C}_n.\end{aligned} \tag{3.7}$$

Our candidate for the supermartingale is

$$M_n = \widehat{Z}_n - \sum_{k=0}^{n-1}(\widehat{B}_k - \widehat{C}_k), \quad n \in \mathbb{N}.$$

The supermartingale property can be checked readily:

$$\begin{aligned}\mathbb{E}[M_{n+1} \mid \mathcal{F}_n] &= \mathbb{E}[\widehat{Z}_{n+1} \mid \mathcal{F}_n] - \sum_{k=0}^{n}\Big(\mathbb{E}[\widehat{B}_k \mid \mathcal{F}_n] - \mathbb{E}[\widehat{C}_k \mid \mathcal{F}_n]\Big) \\ &\le \widehat{Z}_n + \widehat{B}_n - \widehat{C}_n - \sum_{k=0}^{n}(\widehat{B}_k - \widehat{C}_k) \\ &= \widehat{Z}_n - \sum_{k=0}^{n-1}(\widehat{B}_k - \widehat{C}_k) = M_n,\end{aligned}$$

where we have used (3.7) and that $\widehat{B}_k$, $\widehat{C}_k$ are $\mathcal{F}_n$-measurable for $k \le n$. In order to apply Doob's martingale convergence theorem, we need to verify $\sup_{n\in\mathbb{N}} \mathbb{E}[M_n^-] < \infty$. Since in general, it is not obvious that this property will hold, we introduce a localization.

CD CUD

Localization: We define the stopping time $\tau_\varepsilon = \inf\{n \ge 1 : \sum_{k=0}^{n} \widehat{B}_k > \varepsilon\}$ for $\varepsilon > 0$ and show that there exists an integrable random variable M_∞^ε with $\lim_{n\to\infty} M_{n\wedge\tau_\varepsilon} = M_\infty^\varepsilon$ almost surely.

Since (B_n) is adapted, τ_ε is a stopping time with respect to the filtration. Moreover, $(M_{n\wedge\tau_\varepsilon})$ is still a supermartingale, and additionally satisfies, using the non-negativity assumptions,

$$M_{n\wedge\tau_\varepsilon} = \widehat{Z}_{n\wedge\tau_\varepsilon} - \sum_{k=0}^{n\wedge\tau_\varepsilon-1}\widehat{B}_k + \sum_{k=0}^{n\wedge\tau_\varepsilon-1}\widehat{C}_k \ge -\sum_{k=0}^{n\wedge\tau_\varepsilon-1}\widehat{B}_k \ge -\varepsilon,$$

since $\sum_{k=0}^{n\wedge\tau_\varepsilon-1}\widehat{B}_k \le \varepsilon$ by construction of the stopping time τ_ε. Since $(M_{n\wedge\tau_\varepsilon})_{k\in\mathbb{N}}$ is uniformly bounded from below (and due to the monotonic decrease of the expectation for supermartingales) we obtain

$$\sup_{n\in\mathbb{N}} \mathbb{E}[|M_{n\wedge\tau_\varepsilon}|] < \infty.$$

We are now ready to apply Doobs Martingale convergence theorem to find an integrable random variable M_∞^ε with $\lim_{n\to\infty} M_{n\wedge\tau_\varepsilon} = M_\infty^\varepsilon$ almost surely. Next, we have to remove the stopping time.

CD CUD

Remove localization: Show that $\lim_{n\to\infty} M_n < \infty$ almost surely.

Let (ε_k) an increasing sequence with $\lim_{k\to\infty}\varepsilon_k = \infty$. First note that, for each $k \in \mathbb{N}$, we have

$$\lim_{n\to\infty} M_{n\wedge\tau_{\varepsilon_k}}(\omega) = M_\infty^{\varepsilon_k}(\omega)$$

for almost all $\omega \in \Omega$. We observe that for each $\omega \in \Omega$ with $\sum_{k=0}^{\infty} \widehat{B}_k(\omega) < \infty$ there exists some $N \in \mathbb{N}$ such that $\omega \in \{\tau_{\varepsilon_N} = \infty\}$, i.e. for this ω it holds that

$$M_{n\wedge\tau_{\varepsilon_N}}(\omega) = M_n(\omega)$$

for all $n \in \mathbb{N}$, but similarly

$$\lim_{n\to\infty} M_n(\omega) = \lim_{n\to\infty} M_{n\wedge\tau_{\varepsilon_N}}(\omega) = M_\infty^{\tau_N}(\omega) < \infty,$$

where the last inequality holds since $\mathbb{E}[|M_\infty^{\tau_N}|] < \infty$.

Conclusion of the argument

Finally, we move back to the assertion regarding (Z_n) and (C_n). Observe that almost surely

$$-\infty < -\sum_{k=0}^{\infty} \widehat{B}_k \le \lim_{n\to\infty} \left(\widehat{Z}_n - \sum_{k=0}^{n-1} (\widehat{B}_k - \widehat{C}_k) \right) = \lim_{n\to\infty} M_n < +\infty$$

because $\hat{B}_i \le B_i$ and $\sum_{k=0}^{\infty} B_k < \infty$ almost surely by assumption. The non-negativity assumption and convergence of $\sum_{k=0}^{\infty} B_k$ then imply almost sure existence of $\lim_{n\to\infty} \widehat{Z}_n$ and $\sum_{k=0}^{\infty} \widehat{C}_k$. Moreover, it holds true that

$$Z_n = \widehat{Z}_n \cdot \prod_{k=0}^{n-1} (1 + A_k),$$

where both $\widehat{Z}_n$ and $\prod_{k=0}^{n-1}(1+A_k)$ converge almost surely. The latter one follows by monotonicity and

$$\prod_{k=0}^{n-1} (1 + A_k) \le \exp\Big(\sum_{k=0}^{n-1} A_k \Big),$$

where the upper bound converges (is bounded) by assumption. Here we used the simple estimate $1 + x \le \exp(x)$. Therefore, $\lim_{n\to\infty} Z_n$ exists almost surely. Similarly, we have

$$\sum_{k=0}^{n} C_k = \sum_{k=0}^{n} \widehat{C}_k \cdot \prod_{j=0}^{k} (1 + A_j) \le \Big(\prod_{j=0}^{\infty} (1 + A_j) \Big) \sum_{k=0}^{n} \widehat{C}_k$$

which implies $\sum_{k=0}^{\infty} C_k < \infty$ almost surely. □

Next we will show a useful extension of Robbins-Siegmund theorem.

Corollary 3.4.3. Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space carrying a non-negative adapted stochastic process (Z_n). If

$$\mathbb{E}[Z_{n+1} \,|\, \mathcal{F}_n] \le (1 - a_n + b_n) Z_n + c_n, \quad n \in \mathbb{N}, \tag{3.8}$$

for some non-negative adaptive stochastic processes (a_n), (b_n), and (c_n) such that almost surely

$$\sum_{k=1}^{\infty} a_k = \infty, \quad \sum_{k=1}^{\infty} b_k < \infty, \quad \text{and} \quad \sum_{k=1}^{\infty} c_k < \infty.$$

Then $\lim_{n\to\infty} Z_n = 0$ almost surely.

Proof. Define $A_n = b_n$, $B_n = c_n$ and $C_n = Z_n a_n$. Then,

$$\mathbb{E}[Z_{n+1} \,|\, \mathcal{F}_n] \le (1 + A_n) Z_n + B_n - C_n, \quad n \in \mathbb{N},$$

with

$$\sum_{k=0}^{\infty} A_k < \infty \quad \text{and} \quad \sum_{k=0}^{\infty} B_k < \infty.$$

The Robbins-Siegmund theorem implies almost sure existence of $\lim_{n\to\infty} Z_n$ and that $\sum_{k=0}^{\infty} Z_k a_k < \infty$ almost surely. The assumed divergence $\sum_{k=0}^{\infty} a_k = \infty$ implies that $\lim_{n\to\infty} Z_n = 0$ almost surely. □

Lemma 3.4.4. Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space carrying all appearing random variables. Suppose

- ε_n are $\mathcal{F}_{n+1}$-measurable random variables with $\mathbb{E}[\varepsilon_n \,|\, \mathcal{F}_n] = 0$ and there is a pathwise bounded adapted process (D_n) such that $\mathbb{E}[\varepsilon_n^2 \,|\, \mathcal{F}_n] \le D_n$ for all $n \in \mathbb{N}$,
- $\alpha_n \in [0,1]$ are $\mathcal{F}_n$-measurable random variables, called step-sizes, satisfying

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

almost surely.

Then the stochastic process W defined by some $\mathcal{F}_0$-measurable initial condition $W(0)$ and the recursion

$$W_{n+1} = \big(1 - \alpha_n\big) W_n + \alpha_n \varepsilon_n, \quad n \in \mathbb{N},$$

converges to 0 almost surely.

To get a feeling just suppose the errors are iid standard Gaussians. If the α_n are essentially 0 then the sequence is essentially constant while the other extreme of α_n essentially 1 leads to a sequence of independent standard Gaussians. Both do not converge to 0. The right Robbins-Monro balance for α_n allows the error terms to pull the sequence towards the mean of the error which is zero.

Proof. We are going to apply the corollary to the Robbins-Siegmund theorem to the sequence W^2. For that sake let us derive the needed inequality:

$$\begin{aligned}
\mathbb{E}\big[W_{n+1}^2 \,\big|\, \mathcal{F}_n\big] &= \mathbb{E}\big[(1-\alpha_n)^2 W_n^2 + \alpha_n^2 \varepsilon_n^2 + 2\alpha_n(1-\alpha_n) W_n \varepsilon_n \,\big|\, \mathcal{F}_n\big] \\
&= (1 - 2\alpha_n + \alpha_n^2) W_n^2 + \mathbb{E}\big[\alpha_n^2 \varepsilon_n^2 \,\big|\, \mathcal{F}_n\big] + 2\alpha_n(1-\alpha_n) W_n \mathbb{E}\big[\varepsilon_n \,\big|\, \mathcal{F}_n\big] \\
&\le (1 - a_n + b_n) W_n^2 + c_n,
\end{aligned}$$

with $a_n = 2\alpha_n$, $b_n = \alpha_n^2$, and $c_n = \alpha_n^2 D_n^2$. Now the claim follows from the Robbins-Siegmund corollary. □

After these foundations we can now start with the main proof. First of all, let us get rid of the weighted norms by showing that without loss of generality equal weights can be used, i.e. $\vartheta = \mathbf{1}$. Transforming space by

$$\tilde{x}_i := \frac{x_i}{\vartheta_i}, \qquad \tilde{F}_i^n(x) := \frac{F_t(\vartheta_i x)}{\vartheta_i}, \qquad \tilde{b}_i := \frac{b_i}{\vartheta_i}, \quad \text{and} \quad \tilde{\varepsilon}_i(n) := \frac{\varepsilon_i(n)}{\vartheta_i}.$$

It follows immediately that $(\tilde{x}(n))$ solves the same recursion as $(x(n))$ but now with respect to $\tilde{F}$ and $\tilde{\varepsilon}$. Since the maximum norm of the transformed values equaly the weighted norm before the transformation, the conditions of the theorems hold with $\vartheta = \mathbf{1}$, e.g.

$$\|\tilde{F}_n(\tilde{x})\|_\infty = \|F_n(x(n))\|_\vartheta \le \lambda \|x(n)\|_\vartheta + D = \lambda \|\tilde{x}(n)\|_\infty + D.$$

From now on we will assume $\vartheta = \mathbf{1}$, i.e. we will work with $||\cdot||_\infty$ instead of $||\cdot||_\vartheta$.

Let us now start with the proof with a tedious choice of constants G_n. Instead of them carefully we could also choose the constants arbitrarily but would get less clean bounds for x, F, and b which would result in a much more nasty computation in the forth step.

There is an increasing adapted process (G_n) such that $||x(n)||_\infty \leq (1+\varepsilon)G_n$ for some $\varepsilon > 0$ and $\|F^n(x(n))\|_\infty + \omega_n(\|x(n)\|_\infty + 1) \leq G_n$.

Let λ and D be the constants from the theorem. Choose $G \geq 1$ such that $\lambda G + D < G$, for example $G := \max\{\frac{D+1}{1-\lambda}, 1\}$. Next, choose $\eta \in [0,1)$ such that $\lambda G + D = \eta G$. As $D > 0$ this implies $\lambda < \eta$. Finally, let us choose $\varepsilon > 0$ such that $(1+\varepsilon)\eta = 1$.

We will next define a recursive (random) sequence G_n. We set $G_0 = \max\{\|x(0)\|_\infty, G\}$ and define inductively

$$G_{n+1} = \begin{cases} G_n & : \|x(n+1)\|_\infty \leq (1+\varepsilon)G_n \\ G_0(1+\varepsilon)^\kappa & : \|x(n+1)\|_\infty > (1+\varepsilon)G_n, \end{cases},$$

where κ is chosen such that $G_0(1+\varepsilon)^{\kappa-1} < \|x(n+1)\|_\infty \leq G_0(1+\varepsilon)^\kappa$. As $\|\cdot\|_\infty$ is a continuous map, $\|x(n)\|_\infty$ is $\mathcal{F}_n$-measurable. G_n is determined only by $\|x(n)\|_\infty$ and operations that preserve measurability such as checking order, taking the max or multiplying a constant. Thus, the stochastic process (G_n) is adapted with respect to the given filtration. Furthermore, the sequence is non-decreasing and satisfies by construction

$$\|x(n)\|_\infty \leq (1+\varepsilon)G_n \tag{3.9}$$

as well as

$$\|x(n)\|_\infty \leq G_n \quad \text{if } G_{n-1} < G_n. \tag{3.10}$$

Next, since $\lambda < \eta$ we also have $\lambda\varepsilon + \eta < \eta\varepsilon + \eta = 1$. Thus, we can choose $\omega^* > 0$ such that

$$\lambda\varepsilon + \eta + \omega^*(2+\varepsilon) \leq 1. \tag{3.11}$$

By assumption the sequence (ω_n) converges to zero, hence, there is a $n^* \in \mathbb{N}_0$ such that $\omega_n \leq \omega^*$ for all $n \geq n^*$. We will now use this groundwork and prove

$$\|F^n(x(n))\|_\infty + \omega_n(\|x(n)\|_\infty + 1) \leq G_n, \quad \forall n \geq n^*. \tag{3.12}$$

Using the linear growth assumption, (3.9), the definition of η, and the fact that $G_n \geq G$ for all $n \in \mathbb{N}$ we obtain

$$\begin{aligned} \|F^n(x(n))\|_\infty &\leq \lambda\|x(n)\|_\infty + D \\ &\leq \lambda(1+\varepsilon)G_n + D \\ &= \lambda(1+\varepsilon)G_n + (\eta - \lambda)G \\ &\leq \lambda(1+\varepsilon)G_n + (\eta - \lambda)G_n \\ &= (\lambda(1+\varepsilon) + (\eta - \lambda))G_n = (\lambda\varepsilon + \eta)G_n. \end{aligned}$$

Applying this inequality, the definition of n^*, again (3.9), $G_n \geq 1$, and (3.11) we get, for $n \geq n^*$,

$$\begin{aligned} \|F^n(x(n))\|_\infty + \omega_n(\|x(n)\|_\infty + 1) &\leq (\lambda\varepsilon + \eta)G_n + \omega^*((1+\varepsilon)G_n + 1) \\ &\leq (\lambda\varepsilon + \eta)G_n + \omega^*((2+\varepsilon)G_n) \\ &= (\lambda\varepsilon + \eta + \omega^*(2+\varepsilon))G_n \\ &\leq G_n. \end{aligned}$$

That's it. It is important to note that not much happened, we only wrote down a possibly increasing sequence of boxes in which the solutions stays. This only becomes interesting if we can show that the G_n ultimately stop growing! This is what we do in the next steps.

QD CD Reduction to bounded second moment errors.

As a next step, we will rescale the noise terms to control the conditional second moments. This will later allow us to apply Lemma 3.4.4. For this purpose set

$$\tilde{\varepsilon}_i(n) = \frac{\varepsilon_i(n)}{G_n}, \quad n \in \mathbb{N}, i = 1, ..., d.$$

The assumptions on ε and the measurability of G_n yield

$$\mathbb{E}[\tilde{\varepsilon}_i(n) \,|\, \mathcal{F}_n] = \frac{\mathbb{E}[\varepsilon_i(n) \,|\, \mathcal{F}_n]}{G_n} = 0$$

and

$$\mathbb{E}\big[\tilde{\varepsilon}_i^2(n) \,\big|\, \mathcal{F}_n\big] = \frac{\mathbb{E}\big[\varepsilon_i^2(n) \,\big|\, \mathcal{F}_n\big]}{G_n^2} \leq \frac{A + B\|x(n)\|_\infty^2}{G_n^2} \leq \frac{A + B(1+\varepsilon)^2 G_n^2}{G_n^2} = \frac{A}{G_n^2} + B(1+\varepsilon)^2$$

Note that, because G_n is a non-decreasing process and $G_n \geq 1$ by construction, the right hand side is almost surely bounded.

Next, a multivariate version of Lemma 3.4.4 is needed:

QD CD For a given coordinate $i \in \{1, ..., n\}$ and $n_0 \in \mathbb{N}$, we recursively define the stochastic process $(\tilde{W}_i(n : n_0))_{n \geq n_0}$ as follows. The recursion is started in $\tilde{W}_i(n_0 : n_0) = 0$ and for all $n \geq n_0$ as

$$\tilde{W}_i(n+1 : n_0) = (1 - \alpha_i(n))\tilde{W}_i(n : n_0) + \alpha_i(n)\tilde{\varepsilon}_i(n).$$

Then, for all $\delta > 0$, there exists some $n_0 \in \mathbb{N}$ such that almost surely it holds that $|\tilde{W}_i(n : n_0)| \leq \delta$ for all $n \geq n_0$ and $i = 1, ..., d$.

The process $(\tilde{W}_i(n : 0))$ satisfies the step-size assumption of Lemma 3.4.4, as it is the step-size assumption of the theorem. The error term assumptions also hold, this was justified in the previous step. Thus, Lemma 3.4.4 implies the almost sure convergence

$$\lim_{n \to \infty} \tilde{W}_i(n : 0) = 0.$$

Now we use that all iterations were constructed from the same random variables (the error terms), this is called a coupling argument in probability theory. For every $s < n$ we can exploit the update and the fact that $\tilde{W}_i(k : k) = 0$ to write inductively

$$\begin{aligned}
&\tilde{W}_i(n : 0) \\
&= (1 - \alpha_i(n-1))\tilde{W}_i(n-1 : 0) + \alpha_i(n-1)\tilde{\varepsilon}_i(n-1) \\
&= (1 - \alpha_i(n-1))\tilde{W}_i(n-1 : 0) + (1 - \alpha_i(n-1)) \cdot \overbrace{\tilde{W}_i(n-1 : n-1)}^{=0} + \alpha_i(n-1)\tilde{\varepsilon}_i(n-1) \\
&= (1 - \alpha_i(n-1))\tilde{W}_i(n-1 : 0) + \tilde{W}_i(n : n-1) \\
&= (1 - \alpha_i(n-1))\tilde{W}_i(n-1 : 0) + \tilde{W}_i(n : n-1).
\end{aligned}$$

Plugging-in again yields equality to

$$\begin{aligned}
&\Big(\prod_{k=n}^{n-1}(1 - \alpha_i(k-2))\Big)\tilde{W}_i(n-2 : 0) + (1 - \alpha_i(n-1))\tilde{W}_i(n-1 : n-2) \\
&\qquad + \overbrace{\underbrace{(1 - \alpha_i(n-1))\tilde{W}_i(n-1 : n-1)}_{=0} + \alpha_i(n-1)\varepsilon_i(n-1)}^{\tilde{W}_i(n:n-1)} \\
&= \Big(\prod_{k=n-2}^{n-1}(1 - \alpha_i(k))\Big)\tilde{W}_i(s : 0) + \tilde{W}_i(n : n-2).
\end{aligned}$$

Iterating gives

$$\tilde{W}_i(n:0) = \Big(\prod_{k=s}^{n-1}(1-\alpha_i(k))\Big)\tilde{W}_i(s:0) + \tilde{W}_i(n:s)$$

for all $s < n$. As the stepsizes are ≤ 1 by assumption the product in the equation is also bounded by 1, hence,

$$|\tilde{W}_i(n:s)| \leq |\tilde{W}_i(n:0)| + |\tilde{W}_i(s:0)|.$$

Due to the convergence discussed above we find for every $\delta > 0$ some $n_0 \in \mathbb{N}_0$ with $|\tilde{W}_i(n:0)| \leq \frac{\delta}{2}$ for all $n \geq n_0$ and all $i = 1, ..., d$. This and the above inequality yield the claim from this step.

We now come to the main argument, comparing the multivariate sequence $(x(n))$ to the multivariate version $\tilde{W}$ of the simple recursion from Lemma 3.4.4.

CD CD

Suppose $n_0 \in \mathbb{N}$ is such that, for some $\varepsilon > 0$,

- $\|x(n_0)\|_\infty \leq G_{n_0}$,
- $||\tilde{W}(n:n_0)||_\infty \leq \varepsilon, \quad \forall n \geq n_0$,
- $\omega_n \leq \omega^*, \quad \forall n \geq n_0$.

Then we have $G_n = G_{n_0}$ for all $n \geq n_0$ and

$$\begin{aligned} -G_{n_0}(1+\varepsilon) &\leq -G_{n_0} + \tilde{W}_i(n:n_0)G_{n_0} \\ &\leq x_i(n) \\ &\leq G_{n_0} + \tilde{W}_i(n:n_0)G_{n_0} \\ &\leq G_{n_0}(1+\varepsilon). \end{aligned}$$

We give a proof by induction over n:

Induction start: For $n = n_0$ the desired equality obviously holds. Furthermore, because $\|x(n_0)\|_\infty \leq G_{n_0}$ by the first assumption, $\tilde{W}_i(n_0:n_0) = 0$, and $\varepsilon > 0$ the desired inequalities also hold.

Induction step: Let us assume the claims hold for fixed but arbitrary $n \geq n_0$. Then, using the recursion and the induction hypothesis, the assumptions on the noise terms, (3.12) and the third assumption yield

$$\begin{aligned} x_i(n+1) &= (1-\alpha_i(n))x_i(n) + \alpha_i(n)F_i^n(x(n)) + \alpha_i(n)\varepsilon_i(n) + \alpha_i(n)b_i(n) \\ &\leq (1-\alpha_i(n))(G_{n_0} + \tilde{W}_i(n:n_0)G_{n_0}) + \alpha_i(n)F_i^n(x(n)) + \alpha_i(n)\tilde{\varepsilon}_i(n)G_{n_0} \\ &\quad + \ \alpha_i(n)\omega_n(\|x(n)\|_\infty + 1) \\ &\leq (1-\alpha_i(n))(G_{n_0} + \tilde{W}_i(n:n_0)G_{n_0}) + \alpha_i(n)G_{n_0} + \alpha_i(n)\tilde{\varepsilon}_i(n)G_{n_0} \\ &= G_{n_0} + ((1-\alpha_i(n))\tilde{W}_i(n:n_0) + \alpha_i(n)\tilde{\varepsilon}_i(n))G_{n_0} \\ &= G_{n_0} + \tilde{W}_i(n+1:n_0)G_{n_0}. \end{aligned}$$

A symmetrical argument gives $-G_{n_0} + \tilde{W}_i(n+1:n_0)G_{n_0} \leq x_i(n+1)$. Now, using the second assumption we get $|x_i(n+1)| \leq G_{n_0}(1+\varepsilon)$. As $\varepsilon > 0$ was chosen arbitrarily, we conclude $G_{n+1} = G_{n_0}$ as desired. This finishes the inductive proof.

CD CD

Last step. Proof of boundedness.

We come to the end of the proof, where we tie the different results together to show the boundedness. The formal argument is by contradiction. For this purpose, let us assume the sequence $(x(n))$ is unbounded with positive probability. Then (3.9) implies that the sequence

(G_n) diverges to infinity. Because of this, (3.10) implies that the relation $\|x(n)\|_\infty \leq G_n$ holds for infinitely many values of n. Now, we can apply one of the previous steps and get that for an arbitrary $\varepsilon > 0$ there is some large enough n_0 such that

- $\|x(n_0)\|_\infty \leq G_{n_0}$,
- $||\tilde{W}(n : n_0)||_\infty \leq \varepsilon, \quad \forall n \geq n_0$.

Furthermore, as there are infinitely many n with $\|x(n)\|_\infty \leq G_n$, we can choose the n_0 large enough, such that we also have

$$\omega_n \leq \omega^*, \quad \forall n \geq n_0.$$

But these are exactly the assumptions of the previous step which then yields a contradiction to the assumed unboundedness.

3.4.2 Proof of convergence

We are now in a position to prove Theorem 3.3.8. As in the previous section we may restrict ourselfes to the maximum norm.

CD CD

Without loss of generality we can assume that $F^n(x^*) = x^* = 0$.

This can be seen by defining $\tilde{F}^n(\cdot) = F^n(\cdot + x^*) - x^*$. It follows readily that $\tilde{F}^n$ are pseudo-contractions again:

$$\|\tilde{F}^n(x) - 0\|_\infty = \|F^n(x + x^*) - x^*\|_\infty \leq \lambda \|x + x^* - x^*\|_\infty = \lambda ||x - 0||_\infty$$

Now suppose $(\tilde{x}_n)$ is the recursion obtained form $\tilde{F}^n$ and it is proved that $(\tilde{x}_n)$ converges to 0. Adding x^* to both side of the recursion shows that $(\tilde{x}_n + x^*)$ solves the same recursion as (x_n) for F. Thus, convergence of $(\tilde{x}_n)$ to 0 shows convergence of (x_n) to x^*.

CD CD

Preparations

Using the almost sure boundedness we can find a (random) $D_0 > 0$ such that $\|x(n)\|_\infty \leq D_0$ for all $n \in \mathbb{N}$. For an arbitrary $\varepsilon > 0$ with $\lambda + 2\varepsilon < 1$ we recursively define the sequence

$$D_{k+1} = (\lambda + 2\varepsilon) D_k, \quad k \in \mathbb{N}.$$

This sequence clearly converges to zero. As in the previous proof (simpler here as the normalisation by G_n is not needed) define for $n \geq n_0$

$$W_i(n + 1 : n_0) = (1 - \alpha_i(n)) W_i(n : n_0) + \alpha_i(n)\varepsilon_i(n), \quad W_i(n_0, n_0) = 0.$$

Since $||x_n||_\infty \leq D_0$ the assumption on ε imply that $\mathbb{E}[\varepsilon_n^2 \,|\, \mathcal{F}_n]$ is path-wise bounded. Thus, Lemma 3.4.4 implies that $(W(n : n_0)$ converges to zero almost surely for all $n_0 \in \mathbb{N}$.

CD CD

Outer induction in the parameter k. There exists an increasing sequence (n_k) such that $\|x(n)\| \leq D_k$ for all $n \geq n_k$.

Induction start: We already know that $\|x(n)\|_\infty \leq D_0$ so we can choose $n_0 = 0$.

Induction step: We assume $\|x(n)\|_\infty \leq D_k$ for all $n \geq n_k$. Recalling the assumption on b and the induction hypothesis shows that

$$|b_i(n)| \leq \omega_n(\|x(n)\|_\infty + 1) \leq \omega_n(D_k + 1), \quad n \geq n_k.$$

As we assumed (ω_n) to converge to zero this trivially yields the convergence to zero of the bias terms $(b(n))$. We can thus find for an arbitrary $\varepsilon > 0$ a $\tau_k \geq n_k$ such that $||b(n)||_\infty \leq \varepsilon D_k$ for

all $n \geq \tau_k$. We use this costruction to define a further stochastic process $(Y_n)_{n \geq \tau_k}$ for $n \geq \tau_k$ by the recursion

$$Y_i(n+1) = (1 - \alpha_i(n))Y_i(n) + \alpha_i(n)(\lambda + \varepsilon)D_k, \quad Y_i(\tau_k) = D_k.$$

This simple process converges almost surely to $(\lambda + \varepsilon)$ as can be seen as follows. Define $V_i(n) = Y_i(n) - (\lambda + \varepsilon)D_k$, then

$$\begin{aligned} V_i(n+1) &= Y_i(n+1) - (\lambda + \varepsilon)D_k \\ &= (1 - \alpha(n))Y_i(n) + \alpha(n)(\lambda + \varepsilon)D_k - (\lambda + \varepsilon)D_k \\ &= (1 - \alpha(n))Y_i(n) - (1 - \alpha(n))(\lambda + \varepsilon)D_K \\ &= (1 - \alpha(n))V_i(n), \end{aligned}$$

which converges to zero almost surely (by direct iteration or Lemma 3.4.4). In order to verify the outer induction the sequence (x_n) will be sandwiched between (W_n) and (Y_n):

Inner induction for k fixed. The following holds for all $n \geq \tau_k$:

$$-Y_i(n) + W_i(n : \tau_k) \leq x_i(n) \leq Y_i(n) + W_i(n : \tau_k) \tag{3.13}$$

We prove the claim by induction over $n \geq n_k$.

Induction start: Since $Y_i(\tau_k) = D_k$ and $W_i(\tau_k : \tau_k) = 0$ the inequality simplifies to $|x_i(n)| \leq D_k$ in the case $n = \tau_k$ but this holds due to the outer induction assumption as $\tau_k \geq n_k$.

Induction step: Suppose the inequalities hold for some $n > \tau_k$. Recall that $x^* = 0$ was assumed without loss of generality. Thus, $|F_i^n(x(n))| \leq \lambda \|x(n)\|_\infty \leq \lambda D_k$. We will only show the inequality on the righthand side in the following induction step as the one on the left can be derived using a symmetrical argument. Using the update rule, the induction hypothesis, and the definitions of the involved processes gives

$$\begin{aligned} x_i(n+1) &\leq (1 - \alpha_i(n))(Y_i(n) + W_i(n : \tau_k)) + \alpha_i(n)F_i^n(x(n)) + \alpha_i(n)\varepsilon_i(n) + \alpha_i(n)b_i(n) \\ &\leq (1 - \alpha_i(n))(Y_i(n) + W_i(n : \tau_k)) + \alpha_i(n)\lambda D_k + \alpha_i(n)\varepsilon_i(n) + \alpha_i\varepsilon D_k \\ &= \big((1 - \alpha_i(n))Y_i(n) + \alpha_i(n)\lambda D_k + \alpha_i(n)\varepsilon D_k\big) + ((1 - \alpha_i(n))W_i(n : \tau_k) + \alpha_i(n)\varepsilon_i(n)) \\ &= Y_i(n+1) + W_i(n+1 : \tau_k). \end{aligned}$$

This completes the inductive step and the inequality (3.13) is proved for all $n \geq \tau_k$.

Using the sandwich to prove convergence.

The outer induction can now be finished by combining the convergence properties of Y and W. Since W converges to zero and Y_i to $(\lambda + \varepsilon)D_k$ there is some n_{k+1} large enough so that $||x(n)||_\infty \leq (\lambda + 2\varepsilon)D_k = D_{k+1}$ for all $n \geq n_{k+1}$.

Finally, the theorem follows directly from the outer induction. The outer induction implies $\limsup_{n\to\infty} \|x(n)||_\infty \leq D_k$ for arbitrary k. Since D_k vanishes it follows that $\lim_{n\to\infty} \|x(n)\| = 0$.

3.5 Sample based dynamic programming

We will now translate stochastic approximation into algorithms to solve problems from optimal control in a model-free way. In this first section the simplest algorithms are provided, sample based versions of value iteration, both for policy evaluation using T^π and control using T^*. Please keep the following warning in mind:

We have only proved convergence of stochastic fixed point iterations, proofs did not have quantitative flavor. Stochastic approximation schemes converge terribly slow which is not surprising given the law of large number is a special case (Example

(3.3.3)).

For reinforcement learning the setting of Example 3.3.2 will become relevant because Bellman operators can be written as

$$F(x) = \mathbb{E}_x[f(Z)] \tag{3.14}$$

so that approximations of the expectations using samples $\tilde{Z}(1), \tilde{Z}(2), \ldots$ yield model-free learning algorithms of the form

$$x_i(n+1) = x_i(n) + \alpha_i(n)\big(f(x(n), \tilde{Z}_i(n)) - x_i(n)\big) \tag{3.15}$$

that converge almost surely because they can be rewritten as

$$x_i(n+1) = x_i(n) + \alpha_i(n)\big(F_i(x(n)) - x_i(n) + \varepsilon_i(n)\big)$$

with the unbiased error terms $\varepsilon_i(n) := f(x(n), \tilde{Z}_i(n)) - F_i(x(n))$. The algorithms will be asynchroneous 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. The first class of algorithms is extracted readily from writing Bellman operators as one-step expectation. For policy evaluation both versions of T^π, for V and Q, are actually expectation operators:

$$\begin{aligned} T^\pi V(s) &= \sum_{a\in\mathcal{A}_s} \pi(a\,;\, s)\Big(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'\,;\, s,a)V(s')\Big) \\ &= \mathbb{E}^\pi_s[R_0 + \gamma V(S_1)] \end{aligned}$$

and

$$\begin{aligned} T^\pi Q(s,a) &= r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \sum_{a'\in\mathcal{A}_s} p(s'\,;\, s,a)\pi(a'\,;\, s)Q(s',a') \\ &= \mathbb{E}^\pi_{s,a}[R_0 + \gamma Q(S_1, A_1)]. \end{aligned}$$

For the optimality operator the story is slightly different, only the variant for Q is an expectation:

$$\begin{aligned} T^* Q(s,a) &= r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'\,;\, s,a) \max_{a'\in\mathcal{A}_s} Q(s',a') \\ &= \mathbb{E}^\pi_{s,a}\big[R_0 + \gamma \max_{a'\in\mathcal{A}_{S_1}} Q(S_1,a')\big]. \end{aligned}$$

The policy π did not play a role in the final expectation as we only sample S_1 from p. Note that the same does not holds for the optimal Bellman operator for the state value function, the maximum operation would be outside of the expectation. These expressions are exactly of the form (3.14). In order to design stochastic fixed point algorithms to find fixed points of T^π (resp. T^*) all that is needed is the possibility to run the MDP for one step from all states (resp. state-action pairs). In this section we start with the algorithms that follow from the three Bellman operators formulated as expectation operators. Compared to the direct Monte Carlo methods the methods presented below are much simpler to use. While Monte Carlo required an entire rollout, 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.

3.5.1 Sample based policy evaluation algorithms

We start with a first algorithn, so-called TD(0) for policy evaluation. The name becomes clearer once we introduce TD(n) algorithms below. For completeness we give algorithms for V^π and Q^π even though they are essential equal.

Theorem 3.5.1. Suppose $\pi \in \Pi_S$ and the reward distributions have bounded second moments. For an initial vector $V_0 \in \mathbb{R}^{|\mathcal{S}|}$ define the (asynchroneous) update rule

$$V_s(n+1) = V_s(n) + \alpha_s(n)\big(r_n + \gamma V_{s'_n}(n) - V_s(n)\big), \quad s \in \mathcal{S},$$

with $a \sim \pi(\cdot\,;\, s)$, $(s'_n, r_n) \sim p(\cdot\,;\, s, a)$ and step-sizes α_n that only depend on the past steps and satisfy the Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_s(n) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_s^2(n) < \infty \quad \text{a.s.}$$

for every $s \in S$. Then $\lim_{n\to\infty} V(n) = V^\pi$ almost surely.

In words: in every round n and every state s one transition of the MDP must be sampled, resulting in action a_n, next state s'_n and reward r_n. The update rule then updates $V_n(s)$ using r_n and s'_n. If $\alpha_s(n) = 0$ then there is clearly no need to run the MDP from s. Thus, and this will be the case in the algorithms below, only one sample is required in the totally asynchroneous setting (only one $\alpha_s(n)$ different from zero). There is absolutely no assumption on the choice of α apart from the Robbins-Monroe conditions and adaptivity to the rounds before. The adaptivity is no assumption for a practical algorithm, how should an algorithm produce a sequence that uses random variables from future rounds?

Proof of Theorem 3.5.1. The convergence follows immediately from the general convergence theorem once the iteration is rewritten in the right way:

$$V_s(n+1) = V_s(n) + \alpha_s(n)\big(F_s(V(n)) - V_s(n) + \varepsilon_s(n)\big)$$

with

$$F_s(V) := \mathbb{E}_s^\pi[R_0 + \gamma V_{S_1}]$$

and

$$\varepsilon_s(n) := (r_n + \gamma V_{s'_n}(n)) - \mathbb{E}_s^\pi[R_0 + \gamma V_{S_1}(n)].$$

We need to be a bit careful with the filtration. The σ-algebras $\mathcal{F}_n$ is generated by all random variables that occur in all iterations that are needed to define $V(n)$. These are the random-step sizes, the reward samples, the actions, and the next states.

- In the proof of Theorem 2.1.26 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).
- By definition of the filtration 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 $\sup_s \mathbb{E}[\varepsilon_s^2(n)\,|\,\mathcal{F}_n] \le A + B\|V(n)\|_\infty^2$:

$$\begin{aligned}
&\mathbb{E}[\varepsilon_s^2(n)|\mathcal{F}_n] \\
&= \mathbb{E}[(r_n + \gamma V_{s'_n}(n))^2\,|\,\mathcal{F}_n] - 2\mathbb{E}_s^\pi[R_0 + \gamma V_{S_1}(n)]\mathbb{E}_s^\pi[r_n + \gamma V_{s'_n}(n)\,|\,\mathcal{F}_n] + (\mathbb{E}[R_0 + \gamma V_{S_1}(n)])^2 \\
&= \mathbb{E}_s^\pi[(R_0 + \gamma V_{S_1}(n))^2] - 2(\mathbb{E}_s^\pi[R_0 + \gamma V_{S_1}(n)])^2 + (\mathbb{E}_s^\pi[R_0 + \gamma V_{S_1}(n)])^2 \\
&\le \mathbb{E}_s^\pi[(R_0 + \gamma V_{S_1}(n))^2] \\
&= \mathbb{E}_s^\pi[R_0^2] + 2\gamma\mathbb{E}_s^\pi[R_0 V_{S_1}(n))^2] + \gamma^2\mathbb{E}_s^\pi[V_{S_1}(n)^2] \\
&\le C^2 + 2\gamma C\|V(n)\|_\infty + \gamma^2\|V(n)\|_\infty^2 \\
&\le C^2 + 2\gamma C(1 + \|V(n)\|_\infty^2) + \gamma^2\|V(n)\|_\infty^2
\end{aligned}$$

- The step-sizes are adapted (they are only allowed to use the random variables from the strict past) and satisfy almost surely the Robbins-Monroe conditions by assumption.

Hence, Theorem 3.3.8 can be applied and we get almost sure convergence to the unique fixpoint of F which is V^π. □

To repeat the important argument once more here is the version to compute Q^π:

Theorem 3.5.2. Suppose $\pi \in \Pi_S$ and the reward distributions have bounded second moments. For an initial matrix $Q_0 \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ define the (asynchroneous) update rule

$$Q_{s,a}(n+1) = Q_{s,a}(n) + \alpha_{s,a}(n)(r_n + \gamma Q_{s'_n,a'_n}(n) - Q_{s,a}(n)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s.$$

We assume for every n that $(s'_n, r_n) \sim p(\cdot\,; s, a)$ and $a'_n \sim \pi(\cdot\,; s'_n)$, as well as $\alpha(n)$ depend only on the past steps and almost surely satisfy the Robbins-Monro conditions for every $(s,a) \in \mathcal{S} \times \mathcal{A}$. Then $\lim_{n\to\infty} Q(n) = Q^\pi$ almost surely.

Proof. Write

$$Q_{s,a}(n+1) = Q_{s,a}(n) + \alpha_{s,a}(n)\big(F_{s,a}(Q(n)) - Q_{s,a}(n) + \varepsilon_{s,a}(n)\big), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

with

$$F_{s,a}(Q) := T^\pi(Q)(s,a) = \mathbb{E}^\pi_{s,a}[R_0 + \gamma Q_{S_1,A_1}]$$

and

$$\varepsilon_{s,a}(n) := (r_n + \gamma Q_{s'_n,a'_n}(n)) - \mathbb{E}^\pi_{s,a}[R_0 + \gamma Q_{S_1,A_1}(n)].$$

The σ-algebras $\mathcal{F}_n$ is generated by all random variables that occur in all iterations that are needed to define $V(n)$. These are the random-step sizes, the reward samples, the actions, and the next states.

- In the proof of Theorem 2.1.26 it was shown that the Bellman expectation operator $F = T^\pi$ is a $||\cdot||_\infty$-contraction.
- The errors $\varepsilon_{s,a}(n)$ are $\mathcal{F}_{n+1}$-measurable with $\mathbb{E}[\varepsilon_{s,a}(n) \,|\, \mathcal{F}_n] = 0$ by definition. The assumed boundedness of the rewards also implies $\sup_{s,a} \mathbb{E}[\varepsilon^2_{s,a}(n) \,|\, \mathcal{F}_n] \le A + B\|Q(n)\|^2_\infty$.
- The step-sizes are adapted and almost surely satisfy the Robbins-Monro conditions by assumption.

Hence, Theorem 3.3.8 can be applied and we get almost sure convergence to the unique fixpoint of F which is Q^π. □

In all algorithms above (and below) that depend on stochastic fixed point iterations the step-size α needs to be set. For totally asynchroneous 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 α for the non-zero coordinate?

Here is a straight forward choice:

In theory we assumed that there are samples (r, s', a') for every iteration in all state-action pairs (s, a). If $\alpha_{s,a}(n) = 0$ then there is no update at (s, a) so in practice the corresponding random variables will not be sampled. The Robbins-Monro summation condition implies that every state-action pair must be updated infinitely often to guarantee convergence of the algorithm. The most obvious choice to guarantee convergence of the algorithms is, for some $\frac{1}{2} < p \leq 1$,

$$\alpha_s(n) = \frac{1}{(T_s(n)+1)^p} \quad \text{resp. } \alpha_{s,a}(n) = \frac{1}{(T_{s,a}(n)+1)^p}$$

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

$$\sum_{n=1}^{\infty} \alpha_s(n) = \sum_{n=1}^{\infty} \frac{1}{n^p} = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_s^2(n) = \sum_{n=1}^{\infty} \frac{1}{n^{2p}} < \infty.$$

We finish the section wth pseudo-code, written in a totally asynchroneous manner. In every iteration only one item of the vector/matrix is updated. According to the theory the choice is governed by the step-sizes to be zero/non-zero and can depened on anything the algorithm has seen before. In case the next state (resp. state-action pair) is chosen as the previous s' (resp. (s', a')) a bit of care is needed if the MDP terminates. If s' is a terminating state than the next iteration needs to restart in some other non-terminating state.

Algorithm 17: Totally asynchroneous policy evaluation for V^π

Data: Policy $\pi \in \Pi_S$
Result: Approximation $V \approx V^\pi$
Initialize vector $V \equiv 0$.
while *not converged* **do**
  Determine s (for instance uniformly or $s = s'$).
  $a \sim \pi(\cdot\,;\, s)$.
  Sample reward $R(s, a)$.
  Sample next state $s' \sim p(\cdot\,;\, s, a)$.
  Determine stepsize $\alpha = \alpha(s)$.
  Update $V_s = V_s + \alpha\big(R(s, a) + \gamma V_{s'} - V_s\big)$.
end

Algorithm 18: Totally asynchroneous policy evaluation for Q^π

Data: Policy $\pi \in \Pi_S$
Result: Approximation $Q \approx Q^\pi$
Initialize matrix $Q \equiv 0$.
while *not converged* **do**
  Determine (s, a) (for instance uniformly or $(s, a) = (s', a')$).
  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 $\alpha = \alpha(s, a)$.
  Update $Q_{s,a} = Q_{s,a} + \alpha\big(R(s, a) + \gamma Q_{s',a'} - Q_{s,a}\big)$.
end

3.5.2 Q-learning and the SARSA trick

We continue with the most famous tabular control algorithm, Q-learning. Solving numerically $T^*Q = Q$ by iterating in a sample based way Bellman's state-action optimality operator T^*. 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. The mathematical theorem behind Q-learning is the following:

Theorem 3.5.3. Suppose the reward distributions have bounded second moments. For an initial matrix $Q_0 \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ define the (asynchroneous) update rule

$$Q_{s,a}(n+1) = Q_{s,a}(n) + \alpha_{s,a}(n)(r_n + \gamma \max_{a' \in \mathcal{A}_{s'_n}} Q_{s'_n,a'}(n) - Q_{s,a}(n)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s.$$

We assume for every n that $(s'_n, r_n) \sim p(\cdot\,; s, a)$, as well as $\alpha(n)$ depend only on the past steps and almost surely satisfy the Robbins-Monro conditions for every $(s,a) \in \mathcal{S} \times \mathcal{A}$. Then $\lim_{n\to\infty} Q(n) = Q^*$ almost surely.

Proof. The proof is exactly the same that we have seen above, now using the optimal Bellman operator

$$T^*Q(s,a) = \mathbb{E}_{s,a}\big[R_0 + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')\big]$$

on $\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$. Please finish the proof yourself as an excercise:

Rewrite the Q-learning algorithm as simulation-based fixed point iteration and check the conditions of Theorem 3.3.8 to prove the almost sure convergence.

□

Let us think a bit how the mathematical theorem can be turned into an algorithm. What we can do is to play the game that is made abstract in the MDP (S, A, R). Thus, we can run through state-action pairs and observe rewards, leading to a totally asynchroneous update scheme (only one $\alpha_{s,a}(n)$ is non-zero). As for the bandit algorithms the choice of state-action pairs to be updated is up to the algorithm design, this is again called exploration.

From the mathematics of convergence proofs the situation is relatively simple. In totally asynchroneous approximate fixed point iterations 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 (s,a) infinitely often as otherwise $\sum_k \alpha_k(s,a)$ would not be infinite.

There are four typical exploration examples that will remind a lot the different ways of exploring for bandit algorithms:

- choose (s,a) randomly, this is called **random walk exploration**,
- run through the MDP according to some fixed policy π, this is called a **behavior policy**,
- act ε-greedy from the current estimates $Q(n)$,
- act soft-ϵ-greedy from the current estimate $Q(n)$ (Boltzmann exploration).

Algorithm 19: Q-learning

Data: Behavior policy $\pi \in \Pi_S$
Result: Approximations $Q \approx Q^*$, greedy$(Q) = \pi \approx \pi^*$
Initialize Q (e.g. $Q \equiv 0$).
while *not converged* **do**
  Initialise s.
  while s *not terminal* **do**
    Sample action a. (e.g. randomly, according to a behavior policy, ε-(soft)greedy).
    Sample reward $R(s,a)$.
    Sample $s' \sim p(\cdot\,;\, s,a)$.
    Determine stepsize α.
    Update $Q_{s,a} = (1-\alpha)Q_{s,a} + \alpha\big((R(s,a) + \gamma \max_{a' \in \mathcal{A}_{s'}} Q_{s',a'})\big)$.
    Set $s = s'$.
  end
end

Similar to stochastic bandits the ε might be sent to zero. In theory one must be careful to properly choose ε to depend on (s,a) to ensure that all state-action pairs are still visited infinitely often. The pseudo-code in Algorithm 20 is written for exploration using a behavior policy, the convergence follows from Theorem 3.5.3 as long as π explores all state-action pairs infinitely often. There is a major difference in different exploration mechanisms. Random walk exploration updates even the most irrelevant actions as much as the most important. A good behavior policy updates much better and learns quicker the relevant Q-values while a poor behavior policy might not put much effort on the relevant Q-values, but how can we find a close to optimal behavior policy if finding an optimal policy is what the approach is made for (think of a behavior policy of the driving decisions of an old autonomous vehicle that is used to train a better version)? That's why most of the time one favors on-policy exploration using Boltzmann or greedy exploration.

Definition 3.5.4. An **exploration mechanism is called off-policy** if the choice of actions is not governed by the currently estimated policy. In contrast, an **exploration mechanism is called on-policy** if the exploration uses the currently estimated policy (Q-values) to explore actions.

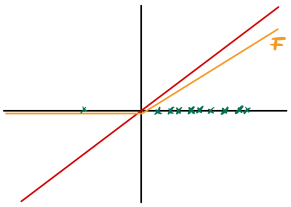
Before discussing the drawbacks of Q-learning let us think more intuitively why Q-learning makes sense. The algorithm challenges current estimates $Q(s,a)$ of $Q^*(s,a)$ by comparing them with new one-step estimates. If the Q-values are stored in a table, then the algorithm successifely explores table entries and updates their values. Let us think about the example from Section 3.1 and try to find the quickest travel directions from Berlin to Mannheim. Whenever a new sample $R(s,a)$ is available (this is the travel time from s to s' under driving decision a) then the old estimate of the expected driving time to Mannheim is updated as follows. Keep the old estimate with a factor $(1-\alpha)$ and update with a factor α by adding the driving time to s' and the best estimated time from s'.

$$Q(s,a) \leftarrow (1-\alpha)\underbrace{Q(s,a)}_{\text{old estimate}} + \alpha\Big(\underbrace{R(s,a)}_{\text{time to reach } s'} + \gamma \underbrace{\max_{a'} Q(s',a')}_{\text{estimated best remaining travel time from } s'}\Big)$$

The factor α is the trust we put into the new observation. The interpretation of vanishing α is that of increasing trust in the estimate which was built on more observations. Since Q-values influence each other the idea of bootstrapping discussed above becomes aparant. Samples $R(s,a)$ do not only help to improve $Q(s,a)$ but also in succeeding steps the Q-values that point to $Q(s,a)$. Think about it, that is exactly how you learn yourself to navigate!

Unfortunately, there is a couple of drawbacks to the Q-learning update rule. Let us discuss two, overestimation bias and exploration danger.

- 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.

- In a way Q-learning is a dangerous learning procedure, it is an optimistic way of learning. The learning procedure does not care a lot about dangerous actions (think about a car/robot and the action might result into a crash). Suppose the exploration forces the learner to go from (s, a) to (s', a') which is already known to be harmful, has a very small Q-value. The update mechanism of Q-learning will completely ignore (s', a'), it will not keep distance from dangerous actions. This might be the optimal solution of the problem but be dangerous in practice if the training requires to run a car or a robot in real.

There are ways around both problems that in the end are based on what is called the SARSA trick. The trick carries its name as it was introduced to prove[5] convergence of the algorithm SARAS that we explain below but is used in other settings as well.

In essence one studies variants of Q-learning with update rules

$$Q_{s,a} \leftarrow Q_{s,a} + \alpha\big((R(s,a) + \gamma \max_{a' \in \mathcal{A}_{s'}} Q_{s',a'}) - Q_{s,a} + b_{s,a}\big)$$

for some additional (negative) bias term b that is supposed to reduce the overly optimistic Q-values. Rewriting the update rule as in the Q-learning proof yields

$$Q_{s,a}(n+1) = Q_{s,a}(n) + \alpha_{s,a}(n)\big((T^*Q(n))(s,a) - Q_{s,a}(n) + \varepsilon_{s,a}(n) + b_{s,a}(n)\big),$$

which converges to Q^* almost surely if the additional bias b decays quickly enough.

In the next section we discuss so-called double Q-learning to reduce the overestimation, in this section let us look at SARSA to reduce the danger of playing dangerous actions. The danger can be reduced by only using the actions the algorithm really observed but not interfering with unobserved actions.

Definition 3.5.5. An update mechanism is called **online updating mechanism** if it only involves the currently observed state-action pairs while an **offline up-**

[5] S. Singh et al.: "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms", Machine Learning, 39, 287–308, 2000

dating mechanism can also use information of state-action pairs that are not currently observed.

The update mechanism of Q-learning is off-policy, it involves in the maximum state-action values of all hypothetical actions, actions that the learner could have used but did not. SARSA is a version of Q-learning with online updating mechanism, the update is as follows:

$$Q_{s,a} \leftarrow Q_{s,a} + \alpha\big((R(s,a) + \gamma Q_{s',a'}) - Q_{s,a}\big)$$

The name S-A-R-S'-A' stems from the use of state-actions (s,a) reward R and next state-actions (s',a').

Algorithm 20: 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**
    Determine stepsize α.
    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 $a' \sim \pi(\cdot\,;\, s')$.
    Update $Q_{s,a} = Q_{s,a} + \alpha\big((R(s,a) + \gamma Q_{s',a'}) - Q_{s,a}\big)$.
    Set $s = s'$, $a = a'$.
  end
end

Theorem 3.5.6. Suppose (Q_n) is the sequence of matrices obtained from the SARSA control algorithm. Assume the reward distributions have bounded second moments, the step-sizes satsify the Robbins-Monroe conditions almost surely and the update policy is GLIE, i.e.

- in the limit the policy is a.s. greedy with respect to the state-action value function,
- each state-action pair is visited infinitely often.

Then $\lim_{n\to\infty} Q(n) = Q_*$ almost surely.

Here are two examples for GLIE policies. One is to explore ε_n-greedy based on $Q(n)$ at each timestep with state-dependent $\varepsilon_n = \frac{1}{T_s(n)}$, another is a Boltzman exploration based on $Q(n)$

$$\pi_n(a\,;\, s) := \frac{e^{\log(T_s(n))Q_{s,a}(n)}}{\sum_b e^{\log(T_s(n))Q_{s,b}(n)}},$$

where $T_s(n)$ is the number of visits in state s. If states are visited infinitely often both policies are greedy in the limit. At the same time they allow enough exploration so that all state-action pairs are visited infinitely often.

Proof. The SARSA trick is to write SARSA as Q-learning with an additional bias term and then assume enough to ensure the bias term satisfies the condition of Theorem 3.3.8:

$$Q_{s,a}(n+1) = Q_{s,a}(n) + \alpha_{s,a}(n)\big(T^*Q_{s,a}(n) - Q_{s,a}(n) + \varepsilon_{s,a}(n) + b_{s,a}(n)\big)$$

with

$$\varepsilon_{s,a}(n) = (r_n + \gamma \max_{a'} Q_{s',a'}(n)) - T^* Q_n(s,a)$$

and

$$b_{s,a}(n) = (r_n + \gamma Q_{s'_n,a'_n}(n)) - (r_n + \gamma \max_{a'} Q_{s'_n,a'}(n)) = \gamma\big(Q_{s'_n,a'_n}(n) - \max_{a'} Q_{s'_n,a'}(n)\big) \le 0.$$

On top of the Q-learning convergence proof one needs to check the condition of Theorem 3.3.8 for the bias term. This is where the GLIE property comes into play as $b_{s,a}(n) = 0$ if a' is chosen greedily. [6] □

In a way the theorem is trivial and does not make much sense. We added a particular bias to the recursion of Q-learning and assumed it vanishes in the limit, that's about it. The point is the particular form of the bias. For SARSA the bias is chosen to be (strongly) negative if neighboring state-action pairs are dangerous in the sense that the Q-value for (s',a') deviates (strongly) from that for the best actions. Thus, if the exploration is based on Q (ε-greedy, Boltzmann) transitions to dangerous neighbors are less likely. This can be seen for instance in cliff walk where SARSA tends to prefer at the beginning trajectories that keep distance to the cliff and only converges later (due to the reduced bias) towards the optimal path. Q-learning instead directly learns to walk close to the cliff and has the tendency to fall down during the training process.

SARSA is actually a sample based version of policy iteration. Suppose the exploration mechanism is ε-greedy. Then in every step the algorithm evaluates with only one step the Q-function of the greedy policy of the step before. This is policy iteration.

3.6 ALTERS ZEUGS

Proof. 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

$$\begin{aligned}
&\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) \,|\, \mathcal{F}_n] \\
&= \mathbb{E}\big[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ is greedy}\}}(\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}^{\pi}_{\tilde{S}_n}{}^{\tilde{A}_n}[\gamma \max_{a'} Q_n(S_1, a')]) \,\big|\, \mathcal{F}_n\big] \\
&\quad + \mathbb{E}\big[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}}(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}^{\pi}_{\tilde{S}_n}{}^{\tilde{A}_n}[\gamma \max_{a'} Q_n(S_1, a')]) \,\big|\, \mathcal{F}_n\big] \\
&= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ greedy}\}} \,|\, \mathcal{F}_n] \underbrace{\mathbb{E}\big[\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}^{\pi}_{\tilde{S}_n}{}^{\tilde{A}_n}[\gamma \max_{a'} Q_n(S_1, a')] \,\big|\, \mathcal{F}_n\big]}_{=0} \\
&\quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}} \,|\, \mathcal{F}_n] \mathbb{E}\big[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}^{\pi}_{\tilde{S}_n}{}^{\tilde{A}_n}[\gamma \max_{a'} Q_n(S_1, a')]) \,\big|\, \mathcal{F}_n\big] \\
&= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}} \,|\, \mathcal{F}_n] \mathbb{E}\big[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}^{\pi}_{\tilde{S}_n}{}^{\tilde{A}_n}[\gamma \max_{a'} Q_n(S_1, a')]) \,\big|\, \mathcal{F}_n\big].
\end{aligned}$$

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,

$$\big|\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) \,|\, \mathcal{F}_n]\big| \le C\mathbb{P}(\pi_{n+1}(\cdot\,;\, s') \text{ non greedy} \,|\, \mathcal{F}_n) = C p_n(s,a).$$

By assumption, the summation condition 3.16 is satisfied. By the boundedness of rewards, also $\mathbb{E}[\varepsilon_n^2(s,a) \,|\, \mathcal{F}_n] \le C < \infty$. As T^* is a contraction and the Robbins-Monro conditions are satisfied, the iteration converges to Q^* almost surely. □

[6] irgendwann mal verstehbar aufschreiben

The most important policy to which the theorem applies is α_n-greedy because $\sum \alpha_n^2(s,a) < \infty$ holds by assumption. Chosing $\alpha_n(s,a) = \frac{1}{T_{s,a}(n)+1}$ 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 **??**. The convergence also holds if

$$\sum_{n=1}^{\infty} \alpha_n(s,a)\big|\mathbb{E}[\varepsilon_n(s,a)\,|\,\mathcal{F}_n]\big| < \infty \quad \text{a.s.} \tag{3.16}$$

for all state-action pairs. , the function F is bounded by some K, and the initial value of the iteration is bounded by K. Hence, we check the condition (3.16) instead of $\mathbb{E}[\varepsilon_n(s,a)\,|\,\mathcal{F}_n] = 0$ with an appropriately chosen error-term. Let us denote by $\tilde{S}_0, \tilde{A}_0, ...$ the sequence of state-action pairs obtained from the algorithm. First, writing

$$Q_{n+1}(s,a) = Q_n(s,a) + \alpha_n(s,a)\big(T^*Q_n(s,a) - Q_n(s,a) + \varepsilon_n(s,a)\big), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

with

$$\varepsilon_n(\tilde{S}_n, \tilde{A}_n) = (R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1})) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}\big[R(\tilde{S}_n, \tilde{A}_n) + \gamma \max_{a'} Q_n(S_1, a')\big]$$

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

$$\begin{aligned}
&\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n)\,|\,\mathcal{F}_n] \\
&= \mathbb{E}\big[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ is greedy}\}}(\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}[\gamma \max_{a'} Q_n(S_1, a')])\,\big|\,\mathcal{F}_n\big] \\
&\quad + \mathbb{E}\big[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}}(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}[\gamma \max_{a'} Q_n(S_1, a')])\,\big|\,\mathcal{F}_n\big] \\
&= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ greedy}\}}\,|\,\mathcal{F}_n]\underbrace{\mathbb{E}\big[\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}[\gamma \max_{a'} Q_n(S_1, a')]\,\big|\,\mathcal{F}_n\big]}_{=0} \\
&\quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}}\,|\,\mathcal{F}_n]\mathbb{E}\big[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}[\gamma \max_{a'} Q_n(S_1, a')])\,\big|\,\mathcal{F}_n\big] \\
&= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot\,;\tilde{S}_{n+1}) \text{ non-greedy}\}}\,|\,\mathcal{F}_n]\mathbb{E}\big[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}}[\gamma \max_{a'} Q_n(S_1, a')])\,\big|\,\mathcal{F}_n\big].
\end{aligned}$$

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,

$$\big|\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n)\,|\,\mathcal{F}_n]\big| \leq C\mathbb{P}(\pi_{n+1}(\cdot\,;\,s') \text{ non greedy}\,|\,\mathcal{F}_n) = Cp_n(s,a).$$

By assumption, the summation condition 3.16 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. □

We start with the SARSA algorithm, an on-policy variant of Q-Learning. For the proof we formulate a special case of the main stochastic approximation theorem:

Lemma 3.6.1. Suppose $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_n))$ is a filtered probability space on which all appearing random variables are defined. Let $(\Delta(n)) \subseteq \mathbb{R}^c$ a stochastic process

defined by the recursion

$$\Delta_i(n+1) = (1-\alpha_i(n))\Delta_i(n) + \alpha_i(n)G_i^n(\Delta(t)), \quad \Delta_0 \in \mathbb{R}^d.$$

We assume the following

- The stepsizes $\alpha_i(n)$ are adapted and satisfy the Robbins-Monroe conditions.
- $\|\mathbb{E}[G^n(\Delta(n))|\mathcal{F}_n]\|_\infty \leq \lambda\|\Delta(n)\|_\infty + c_n$ with $\lambda \in [0,1)$ and a non-negative sequence (c_n) of random variables converging to 0 almost surely.
- $\mathbb{V}[G_i^n(\Delta(n))|\mathcal{F}_n] \leq K(1+\|\Delta(n)\|_\infty)^2$ with a deterministic constant $K > 0$.

Then $\Delta(n)$ converges to 0 almost surely.

Proof. We will show that the lemma is a consequence of Theorem 3.3.8. For this purpose, we define the auxiliary functions

$$H_i^n(x) := \begin{cases} G_i^n(x) & : |\mathbb{E}[G_i^n(x)|\mathcal{F}_n]| \leq \lambda\|x\|_\infty \\ \operatorname{sign}(\mathbb{E}[(G_t(x))_i|\mathcal{F}_t])\lambda\|x\|_\infty & : \text{otherwise} \end{cases}$$

and $h^n(x) = G^n(x) - H^n(x)$. We can now define our building blocks from Theorem 3.3.8 as follows:

$$\begin{aligned} x(n) &:= \Delta(n), \\ F_i^n(x(n)) &:= \mathbb{E}\big[H_i^n(x(n)) \,\big|\, \mathcal{F}_n\big], \\ \varepsilon_i(n) &:= \big(H_i^n(x(n)) - \mathbb{E}[H_i^n(x(n))|\mathcal{F}_n]\big) + \big(h_i^n(x(n)) - \mathbb{E}[h_i^n(x(n))|\mathcal{F}_n]\big), \\ b_i^n &:= \mathbb{E}[h_i^n(x(n))|\mathcal{F}_n]. \end{aligned}$$

To apply Theorem 3.3.8 all conditions must be checked:

- Plugging-in most terms cancel so that the recursion for $(x(n))$ becomes

$$x_i(n+1) = (1-\alpha_i(n))x_i(n) + \alpha_i(n)\big(F_i(x_n) + \varepsilon_i(n) + b_i(n)\big)$$

- By construction, we have

$$\begin{aligned} \|F^n(x(n)) - 0\|_\infty &= \|F^n(x(n))\|_\infty \\ &= \|\mathbb{E}[H^n(x(n)) \,|\, \mathcal{F}_n]\|_\infty \\ &\leq \lambda\|x(n)\|_\infty \\ &= \lambda\|x(n) - 0\|_\infty \end{aligned}$$

  Thus, the mappings F^n are pseudo-contractions with $x^* = 0$.

- The conditions on the step-sizes are equal to those of the theorem.

- We have the $\mathcal{F}_{t+1}$-measurability and $\mathbb{E}[\varepsilon_i(t)|\mathcal{F}_t] = 0$ by construction. Then, we also have

$$\begin{aligned} \mathbb{E}[\varepsilon_i^2(n) \,|\, \mathcal{F}_n] &= \mathbb{V}[\varepsilon_i(n) \,|\, \mathcal{F}_n] \\ &= \mathbb{V}[H_i^n(x(n)) - G_i^n(x(n)) - H_i^n(x(n)) \,|\, \mathcal{F}_n] \\ &= \mathbb{V}[G_i^n(x(n)) \,|\, \mathcal{F}_n] \\ &\leq K(1+\|x(n)\|_\infty)^2 \\ &= K(1+2\|x(n)\|_\infty + \|x(n)\|_\infty^2) \\ &\leq K(1+2(1+\|x(n)\|_\infty^2) + \|x(n)\|_\infty^2) \\ &= 3K(1+\|x(n)\|_\infty^2) \end{aligned}$$

  With $A := 3K$ and $B := 3K$ the second moment assumtion on ε is satisfied.

- Finally, the condition for the bias terms needs to be checked:

$$\begin{aligned}|b_i(n)| &= |\mathbb{E}[h_i^n(\Delta(n))|\mathcal{F}_n]| \\ &= |\mathbb{E}[G_i^n(\Delta(n)) - H_i^n(\Delta(n))|\mathcal{F}_n]| \\ &\leq (|\mathbb{E}[G_i^n(\Delta(n))|\mathcal{F}_n]| - \lambda\|\Delta(n)\|_\infty)\mathbb{I}_{\{\mathbb{E}[G_i^n(\Delta(n))|\mathcal{F}_n]>\lambda\|\Delta(n)\|_\infty\}} \\ &\leq c_n \\ &\leq c_n(\|\Delta(n)\|_\infty + 1)\end{aligned}$$

We thus have the almost sure convergence of our sequence to $x^* = 0$. □

Proof. The proof is based on Lemma 3.6.1, interpreting matices $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ as vectors in $\mathbb{R}^d$ with $d = |\mathcal{S}| \cdot |\mathcal{A}|$. We define

- $\Delta_i(n) := Q_{s,a}(n) - Q^*_{s,a}$,
- $G_i^n(\Delta(n)) := R(s,a) + \gamma Q_{s',a'}(n) - Q^*_{s,a}$.

The tuples (s', a') are used for next state-action pair to safe additional indices. The update schedule of SARSA can now be written as

$$\Delta_i(n+1) = (1 - \alpha_i(n))\Delta_i(n) + \alpha_i(n)G_i^n(\Delta(n))$$

We are left only with checking the assumptions of Lemma 3.6.1:

- The step-size assumptions are imposed in the theorem.
-

$$\begin{aligned}G_i^n(\Delta(t)) &= R(s,a) + \gamma Q_{s',a'}(n) - Q_{s,a}(n) \\ &= R(s,a) + \gamma \max_{b\in\mathcal{A}} Q_{s',b}(n) - Q_{s,a}(n) + \gamma(Q_{s',a'}(n) - \max_{b\in\mathcal{A}} Q_{s',b}(n))\end{aligned}$$

  Now we realise that if we take the conditional expectation w.r.t the first two terms we get exactly the mapping F from the proof of theorem **??**. We know, that this function is a contraction with fixed point Q^*. Thus, we can use the above calculation and get:

$$\|\mathbb{E}[G^n(\Delta(n))|\mathcal{F}_n]\|_\infty \leq \lambda\|\Delta(n)\|_\infty + \gamma \max_{s'\in\mathcal{S}} |Q_n(s',a') - \max_{b\in\mathcal{A}} Q_n(s',b)|$$

If we take into account that a' is chosen according to our learning policy, which is GLIE, we conclude that the second summand is nothing but a non-negative sequence which converges to 0 a.s. Thus, assumption (II) also holds. Lastly, we realise that, similar to our proof of theorem **??**, assumption (III) follows from our simplifying assumption of bounded rewards. We conclude that $(\Delta(t))_{t\in\mathbb{N}_0}$ converges to 0, meaning nothing else than $\lim_{t\to\infty} Q_t(s,a) = Q^*(s,a)$ for all state-action pairs (s,a). Taking into account that π^t becomes greedy with respect to Q_t in the limit and that Q_t gets infinitely close to Q^*, we can derive that π^t converges to policy which is greedy w.r.t. Q^*, meaning exactly the optimal policy π^*. □

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.

3.6.1 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 $\hat{\mu}_1, ..., \hat{\mu}_K$ are estimates for expectations $\mu_1, ..., \mu_K$. Then the pointwise maximum $\max_{i=1,...,K} \hat{\mu}_i$ overestimates $\hat{\mu} = \max_{i=1,...,K} \mu_i$ 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 $M_1, ..., M_K$ are $\mathrm{Ber}(p)$-distributed. They are all unbiased estimators of the mean but

$$\begin{aligned}\mathbb{E}[\max M_i] &= \mathbb{P}(\{M_1 = 0, ..., M_K = 0\}) \cdot 0 + \mathbb{P}(\{M_1 = 0, ..., M_K = 0\}^c) \cdot 1 \\ &= 1 - (1-p)^K > p.\end{aligned}$$

Van Hasselt[7] 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.

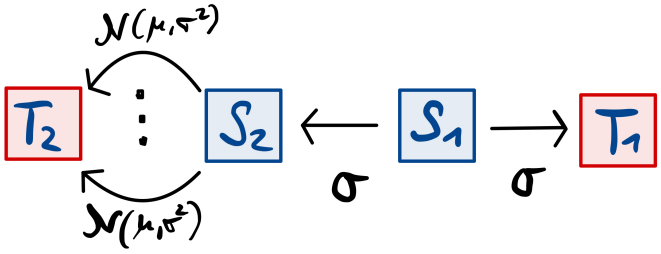
Algorithm 21: Double Q-learning (with behavior policy)

Data: Behavior policy $\pi \in \Pi_S$
Result: Approximations $Q \approx Q^*$, $\pi = \mathrm{greedy}(Q) \approx \pi^*$
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 $update = A$ or $update = B$
    if $update = A$ **then**
      $a^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$
      Update $Q^A(s, a) = Q^A(s, a) + \alpha\big(R(s, a) + \gamma Q^B(s', a^*) - Q^A(s, a)\big)$
    end
    else
      $b^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$
      Update $Q^B(s, a) = Q^B(s, a) + \alpha\big(R(s, a) + \gamma Q^A(s', b^*) - Q^B(s, a)\big)$
    end
    Set $s = s'$.
  end
end

Similarly to SARSA (see the proof of Theorem 3.6.3) double Q-learning can be interpreted as classical Q-learning with an additional negatively biased term $\hat{\varepsilon}$. 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:

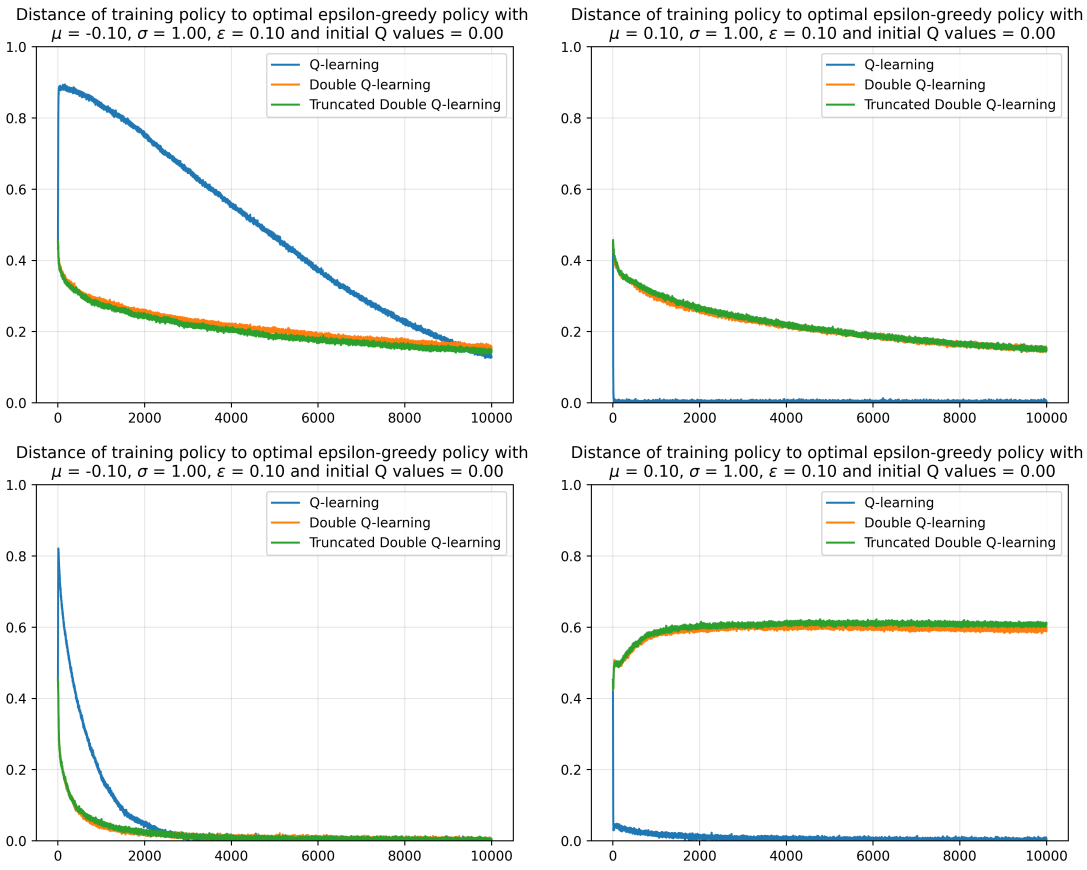
Example 3.6.2. The example MDP consists of two interesting states S_1, S_2 and two terminal states T_1, T_2. The transition probabilities are indicated in the graphical representation. If action left/right is chosen in S_1 then the MDP certainly moves to the left/right with zero reward. In state S_2 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.

[7] H. van Hasselt, "Double Q-learning", NIPS 2010



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 epsidodes 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 $\varepsilon = 0.1$ was used. The plots show depending on the setup/initialisation either positve or negative bias can be benefitial.

Let us now proceed with what we call the truncated double Q-learning algorithm that contains other important double algorithms as special cases.

Algorithm 22: 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 $update = A$ or $update = B$
    if $update = A$ **then**
      $a^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^A(s',a')$
      $\varepsilon = \gamma \max\big(-C_-\alpha, \min\big(Q^B(s',a^*) - Q^A(s',a^*), C_+\alpha\big)\big)$
      Update $Q^A(s,a) = Q^A(s,a) + \alpha\big(R(s,a) + \gamma Q^A(s',a^*) - Q^A(s,a) + \varepsilon\big)$
    end
    else
      $b^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^B(s',a')$
      $\varepsilon = \gamma \max\big(-C_-\alpha, \min\big(Q^A(s',b^*) - Q^B(s',b^*), C_+\alpha\big)\big)$
      Update $Q^B(s,a) = Q^B(s,a) + \alpha\big(R(s,a) + \gamma Q^B(s',b^*) - Q^B(s,a) + \varepsilon\big)$
    end
    Set $s = s'$.
  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:

$$\begin{aligned} Q^A_{n+1}(s,a) &= Q^A_n(s,a) + \alpha\big(R(s,a) + \gamma Q^B_n(s',a^*) - Q^A_n(s',a)\big) \\ &= Q^A_n(s,a) + \alpha\big(R(s,a) + \gamma Q^A_n(s',a^*) - Q^A_n(s,a) + \gamma\big(Q^B_n(s',a^*) - Q^A_n(s',a^*)\big)\big) \end{aligned}$$

which can be written as

$$Q^A_{n+1}(s,a) = Q^A_n(s,a) + \alpha\big(T^*(Q^A_n)(s,a) + \varepsilon_n(s,a) - Q_n(s,a)\big)\,.$$

with errors

$$\varepsilon_n(s,a) := \underbrace{\Big[R(s,a) + \gamma Q^A_n(s',a^*) - T^*Q^A(s,a)\Big]}_{=:\varepsilon^Q_n(s,a)} + \underbrace{\gamma\big(Q^B_n(s',a^*) - Q^A_n(s',a^*)\big)}_{=:\hat{\varepsilon}_n(s,a)}\,.$$

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). 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

$$\max\big(-C_-\alpha, \min\big(Q^B_n(s',a^*) - Q^A_n(s',a^*), C_+\alpha\big)\big) = Q^B_n(s',a^*) - Q^A_n(s',a^*).$$

Theorem 3.6.3. 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 infitely often and the step sizes are adapted and satisfy

the Robbins-Monro conditions. Then

$$\lim_{t\to\infty} Q^A(s,a) = \lim_{t\to\infty} Q^B(s,a) = Q^*(s,a)$$

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 **??**. 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

$$\sum_{n=0}^{\infty} \alpha_n(s,a)|\mathbb{E}[\varepsilon_n(s,a)|\mathcal{F}_n]| < \infty. \tag{3.17}$$

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

$$Q^A_{n+1}(s,a) = Q^A_n(s,a) + \alpha_n(s,a)\left(T^*(Q^A_n)(s,a) + \varepsilon_n(s,a) - Q^A_n(s,a)\right).$$

with error

$$\begin{aligned}\epsilon_t(s,a) &:= \Big[R(s,a) + \gamma Q^A_n(s',a^*) - T^*Q_nA(s,a)\Big] \\ &\quad + \gamma\max\big(-C_-\alpha_n(s,a), \min(Q^B_n(s',a^*) - Q^A_n(s',a^*), C_+\alpha_n(s,a))\big) \\ &=: \varepsilon^Q_n + \hat{\varepsilon}_n(s,a).\end{aligned}$$

All that remains to show is that the error term has bounded conditional second moments and the bias satisfies (3.17). 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:

$$\sum_{n=0}^{\infty} \alpha_n(s,a)\big|\mathbb{E}[\hat{\varepsilon}_n(s,a)|\mathcal{F}_n]\big| \le \max\{C_+,C_-\}\sum_{n=0}^{\infty}\alpha_n^2(s,a) < \infty. \tag{3.18}$$

Since T^* is a contraction and the learning rates satisfy the Robbins-Monro condititions the convergence follows from Theorem **??** with the modification. □

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 clipping[8]:

[8] Fujimoto, van Hoof, Meger: "Addressing Function Approximation Error in Actor-Critic Methods", ICML 2018

Algorithm 23: 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.
    if *update* $= A$ **then**
      $a^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$
      $Q^A(s,a) = Q^A(s,a) + \alpha\big(R(s,a) + \gamma \min\{Q^A(s', a^*), Q^B(s', a^*)\} - Q^A(s,a)\big)$.
    end
    else
      $b^* = \arg\max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$
      $Q^B(s,a) = Q^A(s,a) + \alpha\big(R(s,a) + \gamma \min\{Q^A(s', b^*), Q^B(s', b^*)\} - Q^B(s,a)\big)$.
    end
    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.

3.7 Multi-step approximate dynamic programming

The one-step approximate dynamic programming algorithms were derived rather directly from Theorem 3.3.8. We next turn towards finitely many steps forwards which is inbetween the one-step and infinitely many steps (aka Monte Carlo) approaches.

3.7.1 n-step TD for policy evaluation and control

To understand the idea of n-step temporal differences let us recall the idea behind one-step temporal differences for sample based policy evaluation in the setting of Q^π. The algorithm runs the MDP according to the stationary policy π and at every new state-action pair updates the

matrix according to

$$\underbrace{Q_{\text{new}}(S_t,A_t)}_{\text{new estimate}} = (1-\alpha)\underbrace{Q_{\text{old}}(S_t,A_t)}_{\text{old estimate}} + \alpha\big(\underbrace{R(S_t,A_t)+\gamma Q_{\text{old}}(S_{t+1},A_{t+1})}_{\text{reestimate of } Q(S_t,A_t)}\big).$$

In every step the algorithms reestimate the state(-action) values 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. 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.

Let us go through n-step SARSA value estimation for Q^π.

- Use the Markov property at time n to show that Q^π is also a fixed point of the matrix equation

$$Q(s,a) = \mathbb{E}_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n)\Big].$$

  Compare the proof of Proposition 2.1.15. Check the operator defined by the righthand side is a contraction.

- Use stochastic approximation to show that the sequence

$$\underbrace{Q_{\text{new}}(s,a)}_{\text{new estimate}} = (1-\alpha)\underbrace{Q_{\text{old}}(s,a)}_{\text{old estimate}} + \alpha\Big(\underbrace{\sum_{t=0}^{n-1} r_t + \gamma^n Q_{\text{old}}(s_n,a_n)}_{\text{reestimate of } Q(s,a)}\Big)$$

  converges to Q^π almost surely. Here $r_0,...,r_{n-1}$ are the rewards of an n-step rollout started in (s,a) and (s_n,a_n) is the state-action pair observed after n steps.

- For $n=1$ compare with Section 3.5.1.

- Write algorithmic pseudocode for a totally asynchroneous variant. The algorithm runs a rollout using π, every update requires the next n-steps ahead.

- Do the same for V^π.

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.

Mimic the SARSA control algorithm/proof for n-step temporal difference updates just as in the exercise above.

To understand multistep methods better let us compare the Monte Carlo estimator of Q^π from Section 3.2.1 with the one-step approximate dynamic programming estimator of Q^π from Section 3.5.1. 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 $\hat{Q}_k^\pi = \sum \gamma^t R_t$ of the discounted total reward to get

$$\hat{Q}^\pi(s) = \frac{1}{N}\sum_{k=1}^{N} \hat{Q}_k^\pi(s).$$

The Monte Carlo estimator uses every sample $R(s,a)$ once, whereas the dynamic programming estimator reuses (bootstraps) samples because the iteration scheme

$$Q_{\text{new}}(s,a) = (1-\alpha)Q_{\text{old}}(s,a) + \alpha\big(R(s,a) + \gamma Q_{\text{old}}(s',a')\big)$$

reuses all samples $R(s,a)$ that were used to estimate $Q_{\text{old}}(s',a')$. From the practical point of view the bootstrapping is desirable if the sampling is expansive. 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. For every MDP there will be some integer k for which n-step TD has the optimal performance.

Let's turn these thoughts into a formal error decomposition that highlights the advantages and disadvantages of increasing n.

Proposition 3.7.1. (TD bias-variance decomposition)
Suppose Q is some estimate for Q^π, then

$$\mathbb{E}^\pi_{s,a}\Big[\Big(\overbrace{\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n) - Q^\pi(s,a)}^{\text{reestimation error}}\Big)^2\Big]$$

$$\leq \underbrace{\gamma^{2n}\mathbb{E}^\pi_{s,a}[Q^\pi(S_n,A_n) - Q(S_n,A_n)]^2}_{\text{old estimation bias}} + \underbrace{\mathbb{V}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t\Big]}_{\text{Monte Carlo variance}} + \underbrace{\gamma^{2n}\mathbb{V}^\pi_{s,a}\big[Q(S_n,A_n)\big]}_{\text{old estimate variance}}.$$

Before going through the proof let us discuss what can be learnt from the proposition. Recall that $\gamma \in (0,1)$ is fixed, Q 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 Q, but is the variance of the n-step rewards under the target policy. The Monte Carlo variance increases with n.
- The last summand again involves γ^{2n} which decreases in n and the variance of the current n-step prediction.

Now suppose a policy evaluation algorithm is run with adaptive choice of n, i.e. in every update n is adapted to the situation. The estimate suggests to uses large n if Q is not a good and/or a noisy approximation of Q^π and small n if Q can be expected to be a good approximation. It is thus reasonable to start with larger n (less bootstrapping, more Monte Carlo) and over time decrease n. [9]

Proof of Proposition 3.7.1. We use the classical bias-variance decomposition formula $\mathbb{E}[Z^2] = \mathbb{E}[Z]^2 + \mathbb{E}[(Z-\mathbb{E}[Z])^2]$, sorting the summands, and $Q^\pi(s,a) = \mathbb{E}_{s,a}[\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q^\pi(S_n,A_n)]$

[9] Bild einfuegen, breit im Baum vs. tief im Baum. Soll bootstrapping klarer machen

to compute

$$
\begin{aligned}
&\mathbb{E}^\pi_{s,a}\Big[\Big(\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n) - Q^\pi(s,a)\Big)^2\Big] \\
&= \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n) - Q^\pi(s,a)\Big]^2 \\
&\quad + \mathbb{E}^\pi_{s,a}\Big[\Big(\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n) - Q^\pi(s,a) - \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t + \gamma^n Q(S_n,A_n) - Q^\pi(s,a)\Big]\Big)^2\Big] \\
&= \gamma^n\Big(\mathbb{E}^\pi_{s,a}[Q(S_n,A_n)] - \mathbb{E}^\pi_{s,a}[Q^\pi(S_n,A_n)]\Big)^2 \\
&\quad + \mathbb{E}^\pi_{s,a}\Big[\Big(\sum_{t=0}^{n-1}\gamma^t R_t - \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t\Big] + \gamma^n\big(Q(S_n,A_n) - \mathbb{E}^\pi_{s,a}[Q(S_n,A_n)]\big) - \underbrace{\big(Q^\pi(s,a) - \mathbb{E}^\pi_{s,a}[Q^\pi(S_0,A_0)]\big)}_{=0}\Big)^2\Big] \\
&= \gamma^{2n}\Big(\mathbb{E}^\pi_{s,a}[Q^\pi(S_n,A_n) - Q(S_n,A_n)]\Big)^2 + \mathbb{E}^\pi_{s,a}\Big[\Big(\sum_{t=0}^{n-1}\gamma^t(R_t - \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t\Big]\Big)^2\Big] \\
&\quad + 2\mathbb{E}^\pi_{s,a}\Big[\Big(\sum_{t=0}^{n-1}\gamma^t R_t - \mathbb{E}^\pi_{s,a}\Big[\sum_{t=0}^{n-1}\gamma^t R_t\Big]\Big)\gamma^n\big(Q(S_n,A_n) - \mathbb{E}^\pi_{s,a}[Q(S_n,A_n)]\big)\Big] \\
&\quad + \gamma^{2n}\mathbb{E}^\pi_{s,a}\Big[\big(Q(S_n,A_n) - \mathbb{E}^\pi_{s,a}[Q(S_n,A_n)]\big)^2\Big].
\end{aligned}
$$

The third summand equals 0 because (S_n, A_n) is independent of the rewards $R_0, ..., R_{n-1}$. Thus, the expectation factorises and $\mathbb{E}^\pi_{s,a}[Q(S_n,A_n) - \mathbb{E}^\pi_{s,a}[Q(S_n,A_n)]] = 0$. □

3.7.2 TD(λ) algorithms

[10] 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, choses 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 rollout 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

[10] mache notation Geo(λ) or Geo(1-λ) kompatibel mit Stochastik 1 Skript

difference update

$$\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\Big(\sum_{t=0}^{n-1}\gamma^t R(S_{t+k},A_{t+k})+\gamma^n V_{\text{old}}(S_{t+n})-V_{\text{old}}(S_t)\Big)\Big).$$

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 24: 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 $(s_0, a_0, s_1, ...)$ until T using policy π.
  for $t = 0, 1, 2, ..., T$ **do**
    if $s_t \notin \{s_0, s_1, ...s_{t-1}\}$ **then**
      $V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t)\big(\sum_{i=0}^{T-1}\gamma^i R(s_{t+i},a_{t+i}) + \gamma^T V_n(s_T) - V_n(s_t)\big)$
    end
  end
  $n = n + 1$
end
Return V_n.

Theorem 3.7.2. Suppose $\pi \in \Pi_S$, $\lambda \in (0,1)$, and α satisfies the Robbins-Monro conditions. Then $V_n(s) \to V^\pi(s)$ almost surely for all $s \in \mathcal{S}$.

Proof. As always the convergence follows from Theorem **??** once the algorithm is reformulated with a suitable contraction and (here: unbiased) error. Let us first introduce the contraction that we will interprete in two ways:

$$F(V)(s) := \mathbb{E}_s^\pi\Big[\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\Big(\sum_{t=0}^{n-1}\gamma^t R(S_t,A_t)+\gamma^n V(S_n)\Big)\Big]$$

It is easy to see that $F : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is a contraction:

$$\begin{aligned}
||F(V)-F(W)||_\infty &= \max_{s\in S}\Big|\mathbb{E}_s^\pi\Big[\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}(\gamma^n V(S_n)-\gamma^n W(S_n))\Big]\Big| \\
&\le \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\max_{s\in S}\gamma^n|\mathbb{E}_s^\pi[(V(s_n)-W(s_n))]| \\
&\le \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\max_{s\in S}\gamma^n|V(s)-W(s)| \\
&= \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\gamma^n||V-W||_\infty \\
&= \mathbb{E}[\gamma^X]\,||V-W||_\infty
\end{aligned}$$

for $X \sim \text{Geo}(\lambda)$. Furthermore, the unique fixed point is V^π:

$$\begin{aligned} F(V^\pi)(s) &= \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\mathbb{E}_s^\pi\Big[\sum_{t=0}^{n-1}\gamma^t R(S_t,A_t)+\gamma^n V^\pi(S_n)\Big] \\ &\overset{(??)}{=} (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}V^\pi(s) \\ &= V^\pi(s) \end{aligned}$$

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 $\sum_{t=0}^{T-1}\big(\gamma^t R(S_t,A_t)+\gamma^T V(S_T)$ 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

$$V_{n+1}(s_0) = V_n(s_0) + \alpha_{n+1}(s_0)\big(F(V_n)(s_0) + \varepsilon_n(s_0) - V_n(s_0)\big)$$

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 rollouts. Since we assumed the step-sizes are fixed for each rollout they are $\mathcal{F}_n$-measurable. Furthermore, ε_n is $\mathcal{F}_{n+1}$-measurable (only random variables from rollout $n+1$ 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 bouned as we assume R to be bounded in these lecture notes. □

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 rollout 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 rollout (in contrast for instance to the one-step algorithms). [11] The algorithm We are not going

Algorithm 25: 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 $(s_0, a_0, s_1, ...)$ using policy π.
  for $t = 0, 1, 2, ...$ **do**
    if $s_t \notin \{s_0, s_1, ...s_{t-1}\}$ **then**
      $V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t)(\sum_{n=t}^{\infty}(1-\lambda)\lambda^{n-1}(\sum_{k=0}^{n-1}\gamma^i R(s_{t+k}, a_{t+k}) +$
      $\gamma^n V_{\text{old}}(s_{t+n}) - V_{\text{old}}(s_t))))$
    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.

[11] schreiben 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 suprising fact is that the first visit λ-return algorithm can actually 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 neet lemma is needed.

Lemma 3.7.3. Suppose $s_0, a_0, s_1, a_1, ...$ is a state-action sequence, $\lambda \in (0,1)$, and $V : \mathcal{S} \to \mathbb{R}$, then

$$\begin{aligned}&\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\Big(\sum_{t=0}^{n-1}\gamma^t R(s_{t+k},a_{t+k})+\gamma^n V(s_{t+n})-V(s_t)\Big)\Big)\\ =&\sum_{t=0}^{\infty}(\gamma\lambda)^t\big(R(s_t,a_t)+\gamma V(s_{t+1})-V(s_t)\big).\end{aligned}$$

There are two interesting point of this representation of the TD(λ) update. First, the formula is more pleasant as one infinite sum dissapeared. 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

$$\begin{aligned}&(1-\lambda)\Big(\sum_{n=1}^{\infty}\lambda^{n-1}\Big(\sum_{t=0}^{n-1}\gamma^t R(s_t,a_t)+\gamma^n V(s_n)-V(s_0)\Big)\Big)\\ =&(1-\lambda)\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\sum_{n=t+1}^{\infty}\lambda^{n-1}\Big)+(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^n V(s_n)-V(s_0)\\ =&\sum_{t=0}^{\infty}\big(\gamma\lambda)^t(R(s_t,a_t)+(1-\lambda)\gamma V(s_{t+1})\big)-V(s_0)\\ =&\sum_{t=0}^{\infty}(\gamma\lambda)^t\big(R(s_t,a_t)+\gamma V(s_{t+1})-V(s_t)\big)\end{aligned}$$

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 26.

Algorithm 26: 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 rollout.
  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**
        $V_{n+1}(\tilde{s}) = V_n(\tilde{s}) + \alpha(\tilde{s})(\gamma\lambda)^{t-k(\tilde{s})}\big(R(s,a) + \gamma V_n(s') - V_n(s)\big)$
      end
    end
    $s = s'$, $t = t + 1$
  end
  $n = n + 1$
end
Return V_n.

Theorem 3.7.4. Suppose $\pi \in \Pi_S$, $\lambda \in (0, 1)$, and α satisfies the Robbins-Monro conditions. Then $V_n(s) \to V^\pi(s)$ almost surely for all $s \in \mathcal{S}$ in Algorithms 25 and 26.

Proof. By Lemma 3.7.3 the updates of V_n per rollout 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 3.7.2. F is a contraction with unique fixed point V^π. As always the algorithm is rewriten 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 3.7.3 we get

$$\begin{aligned} V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} (\gamma\lambda)^t \big(R(s_t, a_t) + \gamma V_n(s_{t+1}) - V_n(s_t)\big) \\ &= V_n(s_0) + \alpha_n(s_0) \sum_{i=1}^{\infty} (1-\lambda)\lambda^{i-1} \sum_{t=0}^{i-1} \big(\gamma^t R(s_t, a_t) + \gamma^i V_n(s_i)) - V_n(s_t)\big) \\ &= V_n(s_0) + \alpha_n(s_0)\big(F(V_n)(s_0) - V_n(s_0) + \varepsilon_n(s_t)\big), \end{aligned}$$

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 bouned as we assume R to be bounded. Hence, we got all assumptions for Theorem **??** and convergence towards the fixpoint V^π. [12] □

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 simplfication turns this into the famous TD(λ) algorithm with elicability traces.

Lemma 3.7.5. Suppose $s_0, a_0, s_1, a_1, ...$ is a state-action sequence, $\lambda \in (0,1)$, and $V : \mathcal{S} \to \mathbb{R}$, then

$$\begin{aligned}&\sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} \big((\gamma\lambda)^{k-t} R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big)\\ &= \sum_{k=0}^{\infty} \big(R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big) \underbrace{\sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathbf{1}_{S_t=s}}_{=:e_k(S_0)}.\end{aligned}$$

Proof. The proof follows from a Fubini flip using the indicator $\mathbf{1}_{k\geq t} = \mathbf{1}_{t\leq k}$:

$$\begin{aligned}&\sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} \big((\gamma\lambda)^{k-t} R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big)\\ &= \sum_{k=0}^{\infty} \big(R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (\gamma\lambda)^{k-t}.\end{aligned}$$

□

Let us go back to the first visit algorithm (26) that implements first visit updates. Replacing first vistig updates

$$V_{n+1}(s_0) = V_n(s_0) + \mathbf{1}_{\{\text{first visit of } s_0 \text{ at } t\}} \alpha_{n+1}(s_0) \sum_{k=t}^{\infty} \big((\gamma\lambda)^{k-t} R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big)$$

by the every visit update yields

$$\begin{aligned}V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} \big((\gamma\lambda)^{k-t} R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big)\\ &\overset{\text{Lemma 3.7.5}}{=} V_n(s_0) + \alpha_{n+1}(s_0) \sum_{k=0}^{\infty} \big(R(s_k,a_k) + \gamma V(s_{k+1}) - V(s_k)\big) \underbrace{\sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathbf{1}_{S_t=s_0}}_{=:e_k(s_0)}.\end{aligned}$$

Implemented as an Algorithm the update immediately yields Algorithm 27, 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 $t_1, t_2, ...$, 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 $k \to \infty$. In this algorithm it

[12] both proofs are incomplete. since a state might not be visited in a rollout 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 27: 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 $\alpha(s)$, $s \in \mathcal{S}$, for next rollout.
  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 28, of TD(λ) with eligibility traces in which V is updated during the rollout (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 $0 < m(s) := \mathbb{E}_s^\pi\big[\sum_{t=0}^\infty \mathbf{1}_{S_t=s}\big] < \infty$ and $\tilde{\alpha}_n(s) := \alpha_n(s)m(s)$. Then we can rewrite the every visit update scheme in the mathematical form

$$\begin{aligned} &V_{N+1}(s) \\ &= V_N(s) + \tilde{\alpha}_n(s)\Big[\frac{1}{m(s)}\sum_{t=0}^\infty \mathbf{1}_{s_t=s}(1-\lambda)\sum_{n=1}^\infty \lambda^{n-1}\sum_{k=t}^{t+n-1}(\gamma^{k-t}R(s_k,a_k) + \gamma^n V_N(s_{t+n})) - V_N(s))\Big]. \end{aligned}$$

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 3.7.6. (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

$$V_{N+1}(s) = V_N(s) + \alpha_N(s) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s} \sum_{k=t}^{\infty} \big((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V_N(s_{k+1}) - V_N(s_k)\big).$$

with $\lambda < 1$ and the trajectory $(s_0, a_0, s_1, ...)$ 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 $s \in \mathcal{S}$. Moreover, let the rewards be bounded and $0 < \gamma < 1$. Then $\lim_{n\to\infty} V_N(s) = V^\pi(s)$ 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 Nst rollout. As always we rewrite the update scheme into an asynchronous stochastic approximation scheme:

$$\begin{aligned}
&V_{N+1}(s_0) \\
&= V_N(s_0) + \tilde{\alpha}_n(s_0)\Big(\frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) - V_N(s_0))\Big) \\
&= V_N(s_0) + \tilde{\alpha}_n(s_0)\big(F(V_N)(s_0) - V_N(s_0) + \varepsilon_N(s_0)\big)
\end{aligned}$$

with

$$F(V)(s) := \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V_N(S_{t+n}))\Big]$$

and

$$\begin{aligned}
\varepsilon_N(s_0) := &\frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) \\
&- F(V_N)(s_0) + V_N(s_0) - \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} V_N(s_0)
\end{aligned}$$

Similarly to the previous proof, V^π is a fixed point of F because

$$\begin{aligned}
F(V^\pi)(s) &= \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} \big(\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})\big)\Big] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \mathbb{E}_s^\pi \Big[\sum_{k=t}^{t+n-1} \big(\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})\big) \,\Big|\, (S_t, A_t)\Big]\Big] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(S_t)\Big] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}\Big](1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(s) \\
&= \frac{m(s)}{m(s)} V^\pi(s) = V^\pi(s)
\end{aligned}$$

for every $s \in S$. Similarly to the previous proof we also obtain that F is a contraction:

$$\begin{aligned}||F(V) - F(W)||_\infty &= \max_{s\in S} \Big| \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1}(\gamma^n V(S_{t+n}) - \gamma^n W(S_{t+n})) \Big] \Big| \\ &\leq \max_{s\in S} \frac{1}{m(s)} \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s}(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1}\gamma^n ||V - W||_\infty \Big] \\ &\leq \max_{s\in S} \frac{1}{m(s)} \mathbb{E}\big[\gamma^X\big] \mathbb{E}_s^\pi \Big[\sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \Big] ||V - W||_\infty = \mathbb{E}\big[\gamma^X\big] ||V - W||_\infty\end{aligned}$$

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 **??** and convergence towards the fixpoint V^π. □

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 28: 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 $s \in \mathcal{S}$
  Initialize s.
  Determine step-sizes $\alpha(s)$, $s \in \mathcal{S}$, for next rollout.
  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 interation. 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.

3.8 Tabular simulation based actor-critic

So far we discussed several methods on how to turn value-iteration into sample based model-free algorithms. But how about policy iteration? Can policy iteration also be turned into a sample based model-free algorithm? The answer is yes, and this is called actor-critic.

Part II

Non-Tabular Reinforcement Learning

Up to now we studied so-called tabular algorithms under the basic assumption a table for all state-action pairs can be safed. We then developed algorithms that explicitly dealt with all state-action pairs for instance by computing or estimating the matrices Q^π or Q^*. This approach is only feasibilty if the number of state-action pairs is small enough to at least safe such a table (not to think about operating with the tables, i.e. multiplications, inversions, ect.). This assumption is widely unrealistic. It makes sense for gridworld examples but immediately fails for interesting control problems such as playing a computer game. The number of actions is often not the problem but the number of states might be huge (or even infinite). In such situations tabular algorithms are not feasible and further approximations are indispensible. In this part we will discuss a few of the classical approaches:

- **Policy approximation**. The set of all policies is replaced by a parametric family for which the best policy is approximated using (stochastic) gradient descent,
- **Value function approximation**. for which the Q-matrix (or value function) is replaced by an approximate family $(Q_\theta)_{\theta\in\Theta}$ and instead of learning all entries $Q(s,a)$ separately we try to find the best approximating matrix (in some sense) Q_{θ^*} by optimising a suitable errorfunction using (stochast) gradient descent.
- **Actor-critic**. Mixtures of policy approximation and value function approximation.

All these methods have in common that they somehow involve a (stochastic) gradient algorithm. We thus start with in introductory chapter introducing some of the most classical results.

Chapter 4

A quick dive into gradient descent methods

We will focus during this section[1] on gradient descent methods for functions $f : \mathbb{R}^d \to \mathbb{R}$, 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.

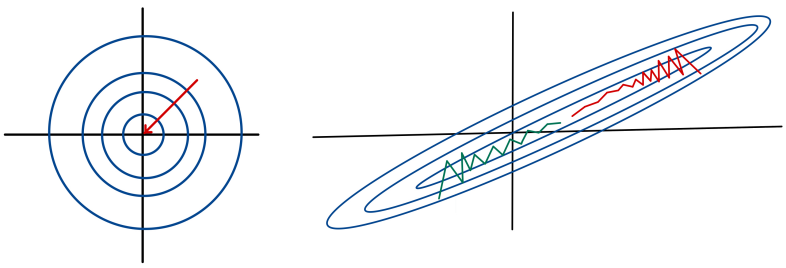
Before we get started recall from basic analysis that the negative gradient is the direction of the steepest descent, i.e.

$$\min_{||d||=1} \underbrace{f'(x)d}_{\text{slope in direction } d} = \nabla f(x).$$

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 chose 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 uses equivalent definitions that are more straightforward to be used in the proofs.



[1] For 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.

To get a first impression in what follows we will go through the most basic theory, for L-smooth (not too step) and strongly convex (not too flat) functions. What is more relavant 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.

Definition 4.0.1. We call a function $f : \mathbb{R}^d \to \mathbb{R}$ L-smooth if f is differentiable and the gradient ∇f is L-Lipschitz continuous, i.e.

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^d. \tag{4.1}$$

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.

4.1 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 4.1.1. (Descent lemma)
Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth for some $L > 0$. Then it holds that

$$f(x + y) \leq \underbrace{f(x) + y^T \nabla f(x) + \frac{L}{2}\|y\|^2}_{\text{tangent at } x \text{ plus quadratic}} \tag{4.2}$$

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. □

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 4.1.2. Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth and $(x_k)_{k\in\mathbb{N}}$ be defined as

$$x_{k+1} = x_k - \bar{\alpha}\nabla f(x_k), \quad x_0 \in \mathbb{R}^d,$$

with non-negative step-size $\bar{\alpha} \in [\varepsilon, \frac{2-\varepsilon}{L}]$ 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. Since f is assumed to be L-smooth we can apply the descent Lemma 4.1.1 (with the choice $y \equiv -\bar{\alpha}\nabla f(x_k)$ and $x \equiv x_k$),

$$\begin{aligned} f(x_k - \bar{\alpha}\nabla f(x_k)) - f(x_k) &\leq (-\bar{\alpha}\nabla f(x_k))^\top \nabla f(x_k) + \frac{L}{2}\|\bar{\alpha}\nabla f(x_k)\|^2 \\ &= \bar{\alpha}\|\nabla f(x_k)\|^2(\frac{\bar{\alpha}L}{2} - 1). \end{aligned} \tag{4.3}$$

In fact, this inequality justifies the name descent lemma. Due to our choice of $\bar{\alpha} \leq \frac{2-\varepsilon}{L}$ we can bound

$$\frac{\bar{\alpha}L}{2} - 1 \leq -\frac{\varepsilon}{2} < 0.$$

We reformulate the inequality (4.3) and obtain

$$f(x_k) - f(x_{k+1}) = f(x_k) - f(x_k - \bar{\alpha}\nabla f(x_k)) \geq \frac{\varepsilon}{2}\bar{\alpha}\|\nabla f(x_k)\|^2 \geq \frac{\varepsilon^2}{2}\|\nabla f(x_k)\|^2. \tag{4.4}$$

Assume that $(x_{n_k})_{k\in\mathbb{N}}$ is a sub-sequence with limit point $\bar{x} \in \mathbb{R}^d$ so that also $\lim_{k\to\infty} f(x_{n_k}) = f(\bar{x})$ by the continuity of f. By the Cauchy criterion for converging sequence s we also obtain

$$\lim_{k\to\infty} (f(x_{n_{k+1}}) - f(x_{n_k})) = 0.$$

Then, using the (4.4), it then follows that

$$\|\nabla f(\bar{x})\| = \lim_{k\to\infty} \|\nabla f(x_k)\|^2 = 0.$$

□

The proof was simple and the result is very weak. There is no statement about convergence and no way to know if the limit points are minima. For this more assumptions on f are needed.

4.2 Gradient descent for L-smooth, convex functions

We will now study the convergence of gradient descent under the additional assumption of convexity.

Definition 4.2.1. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called convex if

$$f(y) \geq f(x) + (y - x)^T \nabla f(x)$$

for all $x, y \in \mathbb{R}^d$.

Recall from basic analysis that the definition has a geometric interpretation. The prime example is a quadratic function $f(x) = x^T Ax$ for a positive definite matrix A but also linear functions are convex. At every point the tangent plane lies below the graph of f. For convex and L-smooth f convergence of the gradient descent recursion to a global minimum can be proved if a global minimum exists (linear functions are convex but do not posess minima). In theorems we will always assume a global minimum exists.

All local minima of convex functions must be global minima, in particular, all global minima have same height.

In what follows we will always denote by f_* the height at a (all) global minimum. In order to talk about convergence rates one typically considers an error function $e : \mathbb{R}^d \to \mathbb{R}$ with the property $e(x) \geq 0$ for all $x \in \mathbb{R}^d$ and $e(x_*) = 0$. A typical choice is $e(x) = f(x) - f_*$. Recall that convergence is called linear (it is actually exponentially fast) if $e(x_{k+1}) \leq ce(x_k)$ for some $c \in (0, 1)$ and sublinear if convergence is slower. For a first order methode (only first derivatives are used) convergence will not be linear. For convex and smooth function we can at least prove the following sublinear convergence rate in case a global minimum exists.

Theorem 4.2.2. Let $f : \mathbb{R}^d \to \mathbb{R}$ 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 mimima we do not aim to prove convergence of x_n to x_* but instead convergence of $f(x_n)$ to f_*.

Proof. We again apply the descent lemma 4.1.1 to the recursive scheme ($y = x_{k+1} - x_k$, $x = x_k$) to obtain

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \bar{\alpha}\|\nabla f(x_k)\|^2 + \frac{L\bar{\alpha}^2}{2}\|\nabla f(x_k)\|^2 \\ &= f(x_k) + \bar{\alpha}\Big(\frac{L\bar{\alpha}}{2} - 1\Big)\|\nabla f(x_k)\|^2. \end{aligned}$$

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 and obtain

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) + \bar{\alpha}\Big(\frac{L\bar{\alpha}}{2} - 1\Big)\|\nabla f(x_k)\|^2 \\ &\leq f(x_*) - \frac{\bar{\alpha}}{\bar{\alpha}}(x_* - x_k)^T\nabla f(x_k) + \bar{\alpha}\Big(\frac{L\bar{\alpha}}{2} - 1\Big)\|\nabla f(x_k)\|^2 \\ &= f(x_*) + \frac{1}{\bar{\alpha}}\Big[\frac{1}{2}\|x_* - x_k\|^2 + \frac{\bar{\alpha}^2}{2}\|\nabla f(x_k)\|^2 - \frac{1}{2}\|(x_* - x_k) + \bar{\alpha}\nabla f(x_k)\|^2\Big] \\ &\quad + \bar{\alpha}\Big(\frac{L\bar{\alpha}}{2} - 1\Big)\|\nabla f(x_k)\|^2, \end{aligned}$$

where we have used the polarisation formula $-\langle a,b\rangle = \frac{1}{2}\|a\|^2 + \frac{1}{2}\|b\|^2 - \frac{1}{2}\|a+b\|^2$ for $a,b \in \mathbb{R}^d$. Rearanging the righthand side yields

$$\begin{aligned} f(x_{k+1}) &\leq f(x_*) + \frac{1}{2\bar{\alpha}}\big(\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2\big) + \bar{\alpha}\Big(\frac{L\bar{\alpha}}{2} - \frac{1}{2}\Big)\|\nabla f(x_k)\|^2 \\ &\leq f(x_*) + \frac{1}{2\bar{\alpha}}\big(\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2\big), \end{aligned}$$

where we have used again that $\bar{\alpha} \leq \frac{1}{L}$. Taking the sum over all iterations gives

$$\begin{aligned} \sum_{k=0}^{N}\big(f(x_{k+1}) - f(x_*)\big) &\leq \frac{1}{2\bar{\alpha}}\sum_{k=0}^{N}\big(\|x_* - x_k\|^2 - \|x_* - x_{k+1}\|^2\big) \\ &= \frac{1}{2\bar{\alpha}}\big(\|x_* - x_0\|^2 - \|x_* - x_{N+1}\|^2\big) \\ &\leq \frac{1}{2\bar{\alpha}}\|x_* - x_0\|^2, \end{aligned}$$

where we have applied a telescoping sum. With the decrease of $(f(x_k))_{k\in\mathbb{N}}$ it follows that

$$\sum_{k=0}^{N} f(x_{k+1}) \geq (N+1) f(x_{N+1}),$$

and therefore, the assertion follows with

$$f(x_{N+1}) - f(x_*) \leq \frac{1}{N+1}\sum_{k=0}^{N}\big(f(x_{k+1}) - f(x_*)\big) \leq \frac{1}{N+1}\frac{1}{2\bar{\alpha}}\|x_* - x_0\|^2 =: \frac{c}{N+1}.$$

□

Definition 4.2.3. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called μ-strongly convex if

$$f(y) \geq \underbrace{f(x) + (y-x)^T\nabla f(x) + \frac{\mu}{2}\|y-x\|^2}_{\text{tangent at } x \text{ plus quadratic}}$$

for all $x, y \in \mathbb{R}^d$.

Linear functions are not strongly convex, $f(x,y) = x^2$ is not strongly convex as it only bends in the x direction. While f is an example of a convex function with infinitely many global minima (the y-axis) all strongly convex functions have a unique global maxima and no further local minima.

Show that a strongly convex function has a unique global minimum.

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 4.2.4. Let $f : \mathbb{R}^d \to \mathbb{R}$ 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 $\bar{\alpha} < \frac{1}{L}$. Then 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. The constant can be chosen as $c = (1 + \mu\bar{\alpha})^{-1/2}$.

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)

$$f(x_n) - f(x_*) \leq \frac{1}{2\mu}||\nabla f(x_n) - \nabla f(x_*)||^2 \leq \frac{L}{2\mu}||x_n - x_*||^2 \leq (c^2)^n \frac{L}{2\mu}||x_0 - x_*||$$

so that the theorem significantly strengthens the previous theorem on only convex functions.

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 4.2.3 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-stronly convex implies convexity we can use the calculations from the previous proof of Theorem 4.2.2, where we have derived that

$$f(x_{k+1}) - f(x_*) \leq \frac{1}{2\bar{\alpha}}\left(||x_k - x_*||^2 - ||x_{k+1} - x_*||^2\right)$$

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 $c = \sqrt{\frac{1}{1+\mu\bar{\alpha}}} \in (0, 1)$. □

For a choice $\bar{\alpha}$ close to $\frac{1}{L}$ the rate is essentially

$$\|x_k - x_*\| \leq \left(\sqrt{\frac{1}{1 + \frac{\mu}{L}}}\right)^k \|x_0 - x_*\|.$$

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.

Suppose $f(x) = x^T Ax$ is a quadratic function. Show that L and μ can be described by the spectrum of A. L is the two times the largest singular value and μ two times the smallest singular value. Thus, convergence is fast if the Eigenvalues are similar while convergence can be slow if the Eigenvalues differ a lot. The latter corresponds to the situation of narrow vallays, the curvatures is big in some direction and small in another direction.

4.3 Gradient descent for L-smooth functions with PL inequality

In the applications for policy gradient methods the assumption of convexity (actually concavitiy 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.

Definition 4.3.1. A function $f : \mathbb{R}^d \to \mathbb{R}$ satisfies the (strong) PL inequality if for some constant $C > 0$

$$\|\nabla f(x)\|^2 \geq C(f(x) - f_*), \quad x \in \mathbb{R}^d, \tag{4.5}$$

holds for all $x \in \mathbb{R}^d$, where $f_* = \min_{x \in \mathbb{R}^d} f(x) > -\infty$.

In fact, the PL inequality is exactly what gradient descent needs for convergence to a global minimum.

Theorem 4.3.2. Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth and satisfied the PL condition holds for some $C = 2r$ with $r \in (0, L)$. 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

$$e(x_k) := f(x_k) - f_* \leq c^k (f(x_0) - f_*),$$

where $c = 1 - \frac{r}{L} \in (0, 1)$.

Note again that since there are no unique mimima we do not aim to prove convergence of x_n to x_* but instead convergence of $f(x_n)$ to $f(x_*)$.

Proof. Using the descent Lemma 4.1.1 as in the proofs above (this only uses L-smooth) yields

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \bar{\alpha}(1 - \frac{L\bar{\alpha}}{2}) \|\nabla f(x_k)\|^2 \\ &= f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2. \end{aligned}$$

Using the PL-inequality yields

$$f(x_{k+1}) \overset{(4.6)}{\leq} f(x_k) - \tfrac{r}{L}(f(x_k) - f_*).$$

Subtracting f_* from both sides, we get

$$f(x_{k+1}) - f_* \leq (1 - \tfrac{r}{L})(f(x_k) - f_*)$$

and the claim follows by induction. □

The idea behind inequality (4.6), also known as PL or Polyak-Lojasiewicz inequality (also called gradient domination) 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 4.4.1 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.

Prove that μ-strong convexity and L-smoothness imply the PL condition (4.6). What is C?

Condition (4.6) can be relaxed a little bit to remain convergence of gradient descent but just with a sublinear convergence rate.

Definition 4.3.3. A function $f : \mathbb{R}^d \to \mathbb{R}$ satisfies the weak PL inequality if for some constant $C > 0$

$$\|\nabla f(x)\| \geq C(f(x) - f_*), \quad x \in \mathbb{R}^d, \tag{4.6}$$

holds for all $x \in \mathbb{R}^d$, where $f_* = \min_{x\in\mathbb{R}^d} f(x) > -\infty$.

We will see below that for very special parametrisations of policies the weak PL inequalities holds for the parametrised value function in reinforcement learning. The convergence can still be proved but unfortunately is much slower, weak PL is not similar to strong convexity.

Theorem 4.3.4. Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth and satisfied the PL condition holds for some $C = 2r$ with $r \in (0, L)$. 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 sublinearly in the sense 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

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L}4r^2(f(x_k) - f_*)^2.$$

Substracting f_* on both sides of the inequality yields the recursion

$$e(x_{k+1}) \leq e(x_k) - \frac{2r^2}{L}e(x_k)^2. \tag{4.7}$$

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

$$0 \leq a_{n+1} \leq (1 - qa_n)a_n, \quad n \geq 0,$$

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

$$\frac{1}{a_n} - \frac{1}{a_0} = \sum_{k=0}^{n-1}\Big(\frac{1}{a_{k+1}} - \frac{1}{a_k}\Big) \geq nq.$$

Reordering we get our claim

$$a_n \leq \frac{1}{nq + \frac{1}{a_0}} \overset{a_0 \leq \frac{1}{q}}{\leq} \frac{1}{(n+1)q}.$$

If we can show that $e(x_k) \leq \frac{L}{2r^2}$ for all k, then the claim follows by applying this to (4.7). As in the previous proof the descent lemma and $\bar{\alpha} = \frac{1}{L}$ give

$$f_* \leq f(x_{k+1}) \leq f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2,$$

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 $e(x_k) \leq \frac{L}{2r^2}$. □

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 (4.6) 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.

4.4 Gradient descent with diminishing step-sizes

Typically, the smoothness parameter L is unknown, especially in a model-free optimization procedure as policy gradient where we only want to assume that we can sample from the MDP. Thus, how could one choose a step-size that satisfies $\bar{\alpha} < \frac{2}{L}$ or $\bar{\alpha} < \frac{1}{L}$? There are plenty of work arounds. Here we show that convergence also works if quickly diminishing step-sizes are used. If step-sizes decay too strongly (being summable) the algorithm might produce a sequence that stops moving away from a stationary point/optimum. We cover the case of convergence to a stationary point by only assuming L-smoothness in the context on diminishing step-sizes. All other results can be transferred to diminishing step-sizes as well.

Theorem 4.4.1. Let $f : \mathbb{R}^d \to \mathbb{R}$ 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

$$\lim_{k\to\infty} \alpha_k = 0 \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k = \infty.$$

Then for the sequence $(f(x_k))_{k\in\mathbb{N}}$ it holds true that either

$$\lim_{k\to\infty} f(x_k) = -\infty \quad \text{or} \quad \lim_{k\to\infty} \nabla f(x_k) = 0.$$

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. As in all proofs above we use the descent lemma to obtain

$$f(x_{k+1}) \leq f(x_k) - \alpha_k\Big(1 - \frac{L\alpha_k}{2}\Big)\|\nabla f(x_k)\|^2.$$

Since we have assumed that $\lim_{k\to\infty} \alpha_k = 0$, 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 $\lim_{k\to\infty} f(x_k) = -\infty$ or $\lim_{k\to\infty} f(x_k) = M$ for some $M \in \mathbb{R}$. Suppose that we are in the case where $\lim_{k\to\infty} f(x_k) = M$. It follows

$$\sum_{k=k_0}^{K} \frac{\alpha_k}{2}\|\nabla f(x_k)\|^2 \leq \sum_{k=k_0}^{K} \big(f(x_k) - f(x_{k+1})\big) = f(x_{k_0}) - f(x_K)$$

for ever $K > k_0$. For $K \to \infty$ this implies

$$\sum_{k=k_0}^{\infty} \frac{\alpha_k}{2}\|\nabla f(x_k)\|^2 \leq f(x_{k_0}) - M < \infty. \tag{4.8}$$

Since $\sum_{k=k_0}^{\infty} \alpha_k = \infty$ it follows that

$$\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

In order to prove $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$ we will now prove that $\limsup_{k\to\infty} \|\nabla f(x_k)\| = 0$. Suppose that $\limsup_{k\to\infty} \|\nabla f(x_k)\| \geq \varepsilon$ for some $\varepsilon > 0$ and consider two sub-sequences $(m_j)_{j\in\mathbb{N}}$ and $(n_j)_{j\in\mathbb{N}}$ with $m_j < n_j < m_{j+1}$ such that

$$\frac{\varepsilon}{3} < \|\nabla f(x_k)\|, \quad \text{for } m_j \leq k < n_j$$

and

$$\|\nabla f(x_k)\| \leq \frac{\varepsilon}{3}, \quad \text{for } n_j \leq k < m_{j+1}.$$

These sequences exists because $\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0$. Moreover, let $\bar{j} \in \mathbb{N}$ be sufficiently large such that

$$\sum_{k=m_{\bar{j}}}^{\infty} \alpha_k\|\nabla f(x_k)\|^2 \leq \frac{\varepsilon^2}{9L}.$$

Using L-smoothness for $j \geq \bar{j}$ and $m_j \leq m \leq n_j - 1$ it holds true that

$$\begin{aligned}
\|\nabla f(x_{n_j}) - \nabla f(x_m)\| &\leq \sum_{k=m}^{n_j-1} \|\nabla f(x_{k+1}) - \nabla f(x_k)\| \\
&\leq L\sum_{k=m}^{n_j-1} \|x_{k+1} - x_k\| \\
&= \frac{3\varepsilon}{3\varepsilon} L\sum_{k=m}^{n_j-1} \alpha_k\|\nabla f(x_k)\| \\
&\leq L\frac{3}{\varepsilon}\sum_{k=m}^{n_j-1} \alpha_k\|\nabla f(x_k)\|^2 \\
&\leq L\frac{3}{\varepsilon}\frac{\varepsilon^2}{9L} = \frac{\varepsilon}{3},
\end{aligned}$$

where we have used that $\|\nabla f(x_k)\| > \frac{\varepsilon}{3}$ for $m_j \leq k \leq n_j - 1$. This implies that

$$\|\nabla f(x_m)\| \leq \|\nabla f(x_{n_j})\| + \|\nabla f(x_{n_j}) - \nabla f(x_m)\| \leq \|\nabla f(x_{n_j})\| + \frac{\varepsilon}{3} \leq \frac{2\varepsilon}{3}$$

and therefore $\|\nabla f(x_m)\| \leq \frac{2\varepsilon}{3}$ for all $m \geq m_{\bar{j}}$. This is in contradiction to $\limsup_{k\to\infty} \|\nabla f(x_k)\| \geq \varepsilon$ and we have proved that

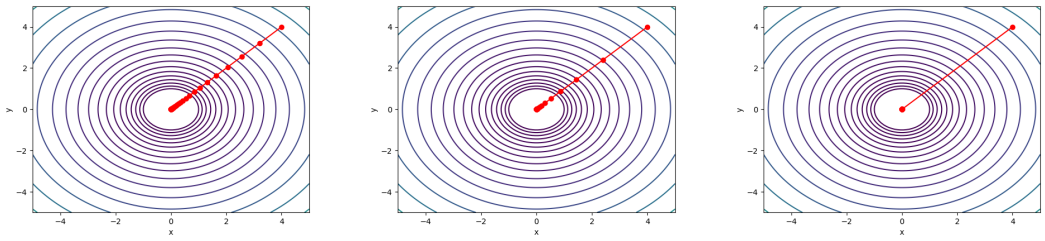
$$\limsup_{k\to\infty} \|\nabla f(x_k)\| = \liminf_{k\to\infty} \|\nabla f(x_k)\| = \lim_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

Finally, let $\bar{x} \in \mathbb{R}^d$ be an accumulating point of $(x_k)_{k\in\mathbb{N}}$. Since $(f(x_k))_{k\geq k_0}$ is decreasing, it follows by continuity that

$$\nabla f(\bar{x}) = \lim_{k\to\infty} \nabla f(x_k) = 0.$$

□

There is an important point to keep in mind. Chosing step-sizes smaller than possible will slow down convergence a lot. As an example think about $f(x) = ||x||^2$. The negative gradient $-\nabla f(x, y) = -(2x, 2y)$ points directly to the minimum and gradient descent with step-size $\alpha = \frac{1}{L} = \frac{1}{2}$ converges in just one step to the global minimum. Gradient descent with smaller (or even diminishing step-sizes) might require many steps.



Gradient descent on $f(x) = ||x||^2$ with constant step-sizes $\alpha = \frac{1}{10}, \frac{1}{5}, \frac{1}{2}$.

4.5 Stochastic gradient descent methods

We consider the following setup. Let $(\Omega, \mathcal{A}, \mathbb{P})$ be the underlying probability space, $Z : \Omega \times \mathbb{R}^d \to \mathbb{R}^p$ be a random variable on $(\Omega, \mathcal{A}, \mathbb{P})$ for every $x \in \mathbb{R}^d$ with distribution μ_x. We are interested in solving the optimization problem

$$\min_{x\in\mathbb{R}^d} F(x),$$

where the cost function $F : \mathbb{R}^d \to \mathbb{R}$ is defined as expectation function in form

$$F(x) = \mathbb{E}_{Z\sim\mu_x}[f(x, Z)] = \int_{\mathbb{R}^p} f(x, z)\mu_x(\mathrm{d}z), \quad x \in \mathbb{R}^d,$$

for a function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$. The mechanism that creates random variables $Z \sim \mu_x$ is often called an oracle. We assume that we can repeatedly ask the oracle for suggestions (samples). Here is a standard set of assumption that typically does not pose any problems:

1. The function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$ is $\mathcal{B}(\mathbb{R}^d) \otimes \mathcal{B}(\mathbb{R}^p)/\mathcal{B}(\mathbb{R})$-measurable.
2. For every $z \in \mathbb{R}^p$ the function $x \mapsto f(x, z)$ is continuously differentiable.
3. For every $x \in \mathbb{R}^d$ we have

$$\mathbb{E}_{Z\sim\mu_x}[|f(x, Z)| + \|\nabla_x f(x, Z)\|] < \infty$$

Algorithm 29: Plain vanilla stochastic gradient descent method (SGD)

Data: Initial random variable $X_0 : \Omega \to \mathbb{R}^d$
Result: Approximation of a stationary point
Set $k = 0$
while *not converged* **do**
  Determine α_k
  Sample Z_{k+1} from μ_{X_k}
  Set $X_{k+1} = X_k - \alpha_k \nabla_x f(X_k, Z_{k+1})$
  $k \mapsto k + 1$
end

In fact, writing $G(x) = \nabla F(x)$ the algorithm very much looks like stochastic approximation.

$$X_{k+1} = X_k - \alpha_k(G(X_k) + \varepsilon_k)$$

with unbiased error terms $\varepsilon_k = \nabla_x f(X_k, Z_{k+1}) - \nabla F(X_k)$. It is thus not too surprising that the algorithm will converge towards a zero of ∇F if the error terms are sufficiently well behaved (such as bounded variance). Indeed, the following simple variant of SGD is also proved using the Robbins-Siegmund theorem:

Theorem 4.5.1. (SGD almost sure convergence for L-smooth function) Let $F : \mathbb{R}^d \to \mathbb{R}$ be L-smooth with $F_* = \inf_{x \in \mathbb{R}^d} F(x) > -\infty$ that satisfies

$$\mathbb{E}_{Z \sim \mu_x}[\|\nabla_x f(x, Z) - \mathbb{E}[\nabla_x f(x, Z)]\|^2] \leq c(1 + (F(x) - F_*)). \tag{4.9}$$

Suppose $(\alpha_k)_{k \in \mathbb{N}}$ are non-negative step-sizes that are deterministic or adapted to the stochastic gradient scheme and satisfy almost surely the Robbins-Monro conditions

$$\sum_{k=0}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Moreover, let X_0 be a random variable such that $\mathbb{E}[F(X_0)] < \infty$ and $(X_k)_{k \in \mathbb{N}}$ the sequence of random variables generated by the stochastic gradient Algorithm 29. Then $(F(X_k))_{k \in \mathbb{N}}$ converges almost surely to some finite random variable F_∞ and

$$\lim_{k \to \infty} \|\nabla F(X_k)\|^2 = 0$$

almost surely.

This first theorem on stochastic gradient descent extends Theorem 4.1.2 to the stochastic setting. The assumptions are weak and also the convergence statement is weak. Under stronger assumptions on F (such as convexity, strong convexity, gradient domination) there are also stronger stochastic convergence theorems in the spirit above. There are two points that should be mentioned. Due to the errors step-sizes must tend to zero in all settings and convergence rates in the stochastic setting are much slower.

Proof. For the proof we assume additionally that gradient and expectation can be interchanged (no big problem)

$$\mathbb{E}_{Z \sim \mu_x}[\nabla_x f(x, Z)] = \nabla F(x). \tag{4.10}$$

We define the natural filtration $\mathcal{F} = (\mathcal{F}_k)_{k \in \mathbb{N}}$ through $\mathcal{F}_k = \sigma(X_m, m \leq k) = \sigma(X_0,\ Z_m, m \leq k)$ and note that $(\alpha_k)_{k \in \mathbb{N}}$ is $\mathcal{F}$-adapted per construction. Using the descent lemma and the

polarisation formula $\langle x, y\rangle = \frac{1}{2}(||x||^2 + ||y||^2 + ||x-y||^2)$ obtain (pathwise) that

$$\begin{aligned} F(X_{k+1}) &= F(X_k - \alpha_k \nabla_x f(X_k, Z_{k+1})) \\ &\leq F(X_k) - \alpha_k \langle \nabla F(X_k), \nabla_x f(X_k, Z_{k+1})\rangle + \alpha_k^2 \frac{L}{2} \|\nabla_x f(X_k, Z_{k+1})\|^2 \\ &= F(X_k) - \alpha_k \|\nabla_x F(X_k)\|^2 + \alpha_k \langle \nabla_x F(X_k), \varepsilon_k\rangle \\ &\quad + \alpha_k^2 \frac{L}{2}\big(\|\nabla_x F(X_k)\|^2 - \langle \nabla F(X_k), \varepsilon_k\rangle + \|\varepsilon_k\|^2\big), \end{aligned}$$

where $\varepsilon_k := \nabla F(X_k) - \nabla_x f(X_k, Z_{k+1})$. Using (4.10) and (4.9) we obtain by construction in Algorithm 29 that

$$\mathbb{E}[\varepsilon_k \mid \mathcal{F}_k] = 0$$

and

$$\mathbb{E}[\|\varepsilon_k\|^2 \mid \mathcal{F}_k] \leq c(1 + (F(X_k) - F_*)).$$

This yields

$$\begin{aligned} \mathbb{E}[\overbrace{F(X_{k+1}) - F_*}^{Z_{k+1}\geq 0} \mid \mathcal{F}_k] &\leq (F(X_k) - F_*) - \alpha_k\Big(1 - \frac{L}{2}\alpha_k\Big)\|\nabla F(X_k)\|^2 + \frac{L}{2}\alpha_k^2 c(1 + (F(X_k) - F_*)) \\ &= (1 + c\frac{L}{2}\alpha_k^2)\underbrace{(F(X_k) - F_*)}_{Z_k\geq 0} + \underbrace{c\frac{L}{2}\alpha_k^2}_{B_k\geq 0} - \underbrace{\alpha_k\Big(1 - \frac{L}{2}\alpha_k\Big)\|\nabla F(X_k)\|^2}_{C_k\geq 0}. \end{aligned}$$

Here we used the assumption that F is bounded below to deduce that the sequence (Z_n) is non-negative. We can assume without loss of generality that $\alpha_k \leq (1-\varepsilon)\frac{2}{L}$ for some $\varepsilon \in (0,1)$, otherwise let k be sufficiently large, such that $(1 - \frac{L}{2}\alpha_k) \geq \varepsilon > 0$. We can now apply Theorem 3.4.2 to deduce that $\lim_{k\to\infty} F(X_k) - F_*$ exists almost surely and is finite, as well as

$$\varepsilon \sum_{k=0}^{\infty} \alpha_k \|\nabla F(X_k)\|^2 \leq \sum_{k=0}^{\infty} \alpha_k (1 - \frac{L}{2}\alpha_k)\|\nabla F(X_k)\|^2 < \infty$$

almost surely. Since we have assumed $\sum_{k=0}^{\infty} \alpha_k = \infty$ almost surely, using the same argument as in the proof of Theorem 4.4.1 pathwise we obtain

$$\lim_{k\to\infty} \|\nabla F(X_k)\|^2 = 0$$

almost surely. □

There are many other variants of the theorem, with weaker/stronger assumptions on F or the oracle that result in weaker/stronger convergence statements. Here is one other variant that is used in the context of reinforcement learning[2].

Suppose instead of (4.9) the condition so-called expected smoothness (or ABC) condition

$$\mathbb{E}_{Z\sim\mu_x}[\|\nabla_x f(x, Z)\|^2] \leq 2A(F(x) - F_*) + B||\nabla F(x)||^2 + C.$$

Then SGD with constant step-sizes $\eta \in (0, \frac{2}{LB})$ satisfies

$$\min_{k\leq K} \mathbb{E}[\|\nabla F(X_k)\|^2] \leq \frac{2(F(X_0) - F_*)(1 + L\eta^2 A)^K}{\eta K(2 - LB\eta)} + \frac{LC\eta}{2 - LB\eta}$$

[2]Yuan, Gower, and Lazaric: "A general sample complexity analysis of vanilla policy gradient", AISTATS 2020

and, if $A = 0$, then one can also show

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|\nabla F(X_k)\|]^2 = \mathbb{E}[\|\nabla F(X)\|]^2 \leq \frac{2(F(X_0)-F_*)(1)^K}{\eta K(2-LB\eta)} + \frac{LC\eta}{2-LB\eta},$$

where X is drawn uniformly from $X_1, ..., X_K$. For a prespecified accuracy ε one can then solve for K to achieve the following consequence. For constant step size $\gamma = \min\{\frac{1}{\sqrt{LAK}}, \frac{a}{LB}, \frac{\varepsilon}{2LC}\}$ and a number of iterations $K \geq \frac{12(F(X_0)-F_*)L}{\varepsilon^2}\max\{B, \frac{12A(F(X_0)-F_*)}{\varepsilon^2}, \frac{2C}{\varepsilon^2}\}$ the gradients can be bounded by $\min_{k\leq K}\mathbb{E}[\|\nabla F(X_k)\|^2] \leq \varepsilon$.

4.6 Regularisation

dieses Jahr noch nicht. [3]

[3] discuss regularisation in non-linear optimisation! take previous example as an example

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 considerd. (1) How to set up an algorithm, how to compute the gradients? This is answered with the policy gradient theorems that suprisingly 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 chosing 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 explicitely, then Algorithm 30 is the kind of algorithm we aim for. Unfortunately, there are plenty of problems.

Algorithm 30: Plain vanilla policy gradient algorithm (with exact gradietns)

Data: Initial parameter θ_0
Result: Approximate policy $\pi^\theta \approx \pi^{\theta^*}$
Set $n = 0$.
while *not converged* **do**
  Choose step-size α.
  Calculate $K = \nabla J(\theta)|_{\theta=\theta_n}$.
  Update $\theta_{n+1} = \theta_n - \alpha K$.
  Set $n = n + 1$.
end
return π^{θ_n}.

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 Policy gradient theorems

[1]

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 30. To start the discussion we first prove formulas for $\nabla J(\theta)$ that go back to Sutton, McAlister, Singh, and Mansour[2]. 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 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 proceeed with the discounted infinite-time case.

5.1.1 Finite-time undiscounted MDPs

In this didactic section we wil 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 5.1.1. 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

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1} \nabla_\theta\big(\log \pi^\theta(A_t\,;\,S_t)\big) R_0^T\Big].$$

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!

[1] Notation anpassen. Rewards, Ubergangswkeiten haben alte Form. Ueberall auf Anfangsbed μ

[2] Sutton, McAlister, Singh, Mansour: "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NIPS, 1999

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

$$\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big) = \delta_s(s_0)\delta_0(r_0)\prod_{t=0}^{T-1}\pi^\theta(a_t\,;\,s_t)p(\{s_{t+1}, r_{t+1}\}\,;\,s_t, a_t)$$

from (2.4). Define

$$\mathcal{T} = \big\{\tau = (r_0, s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T)\,:\, r_t \in \mathcal{R}, s_t \in \mathcal{S}, \forall t \le T,\, a_t \in \mathcal{A}, \forall t < T\big\}$$

as the set of all trajectories. From the definition of the value function for a T-step MDP we deduce that

$$J_s(\theta) = V^{\pi^\theta}(s) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1} R_{t+1}\Big] = \sum_{\tau\in\mathcal{T}}\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\sum_{t=0}^{T-1} r_{t+1}.$$

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

$$\begin{aligned}
&\nabla_\theta\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\\
&= \nabla_\theta(\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big))\frac{\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)}{\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)}\\
&= \nabla_\theta\big(\log(\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big))\big)\,\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\\
&= \nabla_\theta\Big(\log(\delta_s(s_0)) + \log(\delta_0(r_0)) + \sum_{t=0}^{T-1}\big(\log(\pi^\theta(a_t;s_t)) + \log(p(\{s_{t+1},r_{t+1}\};s_t,a_t))\big)\Big)\\
&\quad\times\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\\
&= \sum_{t=0}^{T-1}\nabla_\theta\big(\log(\pi^\theta(a_t;s_t))\big)\,\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big).
\end{aligned}$$

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^{π^θ} is differentiable. We conclude with

$$\begin{aligned}
\nabla_\theta J_s(\theta) &= \nabla_\theta V^{\pi^\theta}(s)\\
&= \sum_{\tau\in\mathcal{T}}\nabla_\theta\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\sum_{t=0}^{T-1} r_{t+1}\\
&= \sum_{\tau\in\mathcal{T}}\mathbb{P}_s^{\pi^\theta}\big((S,A,R) = \tau\big)\sum_{t=0}^{T-1}\nabla_\theta\big(\log(\pi^\theta(a_t;s_t))\big)\sum_{t=0}^{T-1} r_{t+1}\\
&= \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1}\nabla_\theta\big(\log(\pi^\theta(A_t;S_t))\big)\sum_{t=0}^{T-1} R_{t+1}\Big]\\
&= \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1}\nabla_\theta\big(\log(\pi^\theta(A_t;S_t))\big)R_0^T\Big].
\end{aligned}$$

□

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 5.1.2. If π is a policy, then the vector $\nabla_\theta(\log(\pi^\theta(a\,;\,s)))$ is called the score-function of π^θ.

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 choosen 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 5.1.3. 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

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1} \nabla_\theta\big(\log \pi^\theta(A_t\,;\,S_t)\big) R_t^T\Big].$$

Proof. From Theorem 5.1.1 we have

$$\begin{aligned}\nabla_\theta J_s(\theta) &= \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1} \nabla_\theta\big(\log(\pi^\theta(A_t;S_t))\big)\sum_{t=0}^{T-1} R_{t+1}\Big] \\ &= \sum_{t'=0}^{T-1}\sum_{t^*=0}^{T-1} \mathbb{E}_s^{\pi^\theta}\Big[\nabla_\theta\big(\log(\pi^\theta(A_{t'};S_{t'}))\big)R_{t^*+1}\Big].\end{aligned}$$

For every $t^* < t'$ we see

$$\begin{aligned}&\mathbb{E}_s^{\pi^\theta}\Big[\nabla_\theta\big(\log(\pi^\theta(A_{t'};S_{t'}))\big)R_{t^*+1}\Big] \\ &= \sum_{\tau\in\mathcal{T}} \mathbb{P}_s^{\pi^\theta}\big((S,A,R)=\tau\big)\nabla_\theta\big(\log(\pi^\theta(a_{t'};s_{t'}))\big)r_{t^*+1} \\ &= \sum_{a_0}\sum_{s_1}\sum_{r_1}\cdots\sum_{a_{T-1}}\sum_{s_T}\sum_{r_T}\prod_{t=0}^{T-1}\pi^\theta(a_t;s_t)p(\{s_{t+1},r_{t+1}\};s_t,a_t)\big(\nabla_\theta\big(\log(\pi^\theta(a_{t'};s_{t'}))\big)r_{t^*+1}\big) \\ &= \sum_{a_0}\sum_{s_1}\sum_{r_1}\cdots\sum_{s_{t^*}}\sum_{r_{t^*}}\prod_{t=0}^{t^*}\pi^\theta(a_t;s_t)p(\{s_{t+1},r_{t+1}\};s_t,a_t)\sum_{a_{t^*}}\pi^\theta(a_{t^*};s_{t^*}) \\ &\quad\times\sum_{s_{t^*+1}}\sum_{r_{t^*+1}} p(\{s_{t^*+1},r_{t^*+1}\};s_{t^*},a_{t^*})r_{t^*+1}\sum_{a_{t^*+1}}\pi^\theta(a_{t^*+1};s_{t^*+1})\times\dots\end{aligned}$$

$$\times \sum_{s_{t'}} \sum_{r_{t'}} p(\{s_{t'}, r_{t'}\}; s_{t'}, a_{t'}) \underbrace{\sum_{a_{t'}} \pi^\theta(a_{t'}; s_{t'}) \nabla_\theta \big(\log(\pi^\theta(a_{t'}; s_{t'})) \big)}_{= \sum_{a_{t'}} \nabla_\theta \pi^\theta(a_{t'}; s_{t'}) = \nabla_\theta \sum_{a_{t'}} \pi^\theta(a_{t'}; s_{t'}) = \nabla_\theta 1 = 0} \times \overbrace{1}^{\text{sum of probabilities}}$$

$$= 0.$$

Now the claim follows. □

In the last step we will replace the reward-to-go with the Q-function.

Theorem 5.1.4. 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

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta} \Big[\sum_{t=0}^{T-1} \nabla_\theta \big(\log \pi^\theta(A_t \,;\, S_t) \big) Q_t^{\pi^\theta}(S_t, A_t) \Big].$$

Proof. For the proof we use the property of a Markov property of Markov reward processes:

$$\mathbb{E}_s^{\pi^\theta} \big[R_t^T \,|\, S_t = s, A_t = a \big] = Q_t^{\pi^\theta}(s, a).$$

Then the claim follows from Theorem 5.1.3:

$$\begin{aligned}
\nabla_\theta J_s(\theta) &= \sum_{t=0}^{T-1} \mathbb{E}_\mu^{\pi^\theta} \big[\nabla_\theta \big(\log(\pi^\theta(A_t; S_t)) \big) R_t^T \big] \\
&= \sum_{t=0}^{T-1} \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} \mathbb{E}_\mu^{\pi^\theta} \big[\nabla_\theta \big(\log(\pi^\theta(A_t; S_t)) \big) R_t^T \,\Big|\, S_t = s, A_t = a \big] \mathbb{P}_\mu^{\pi^\theta}(S_t = s, A_t = a) \\
&= \sum_{t=0}^{T-1} \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} \nabla_\theta \big(\log(\pi^\theta(a; s)) \big) \mathbb{E}_\mu^{\pi^\theta} \big[R_t^T \,\big|\, S_t = s, A_t = a \big] \mathbb{P}_\mu^{\pi^\theta}(S_t = s, A_t = a) \\
&= \sum_{t=0}^{T-1} \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} \nabla_\theta \big(\log(\pi^\theta(a; s)) \big) Q_t^T(s, a) \mathbb{P}_\mu^{\pi^\theta}(S_t = s, A_t = a) \\
&= \mathbb{E}_s^{\pi^\theta} \Big[\sum_{t=0}^{T-1} \nabla_\theta \big(\log(\pi^\theta(A_t; S_t)) \big) Q_t^{\pi^\theta}(S_t, A_t) \Big]
\end{aligned}$$

□

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 5.1.5. 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.

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

$$\theta_{t+1} = \theta_t + \alpha \, \mathbb{E}_{\mu}^{\pi^\theta} \Big[\sum_{t=0}^{T-1} \nabla_\theta \big(\log \pi^\theta(A_t \,;\, S_t) \big) Q_t^{\pi^\theta}(S_t, A_t) \Big]$$

or

$$\theta_{t+1} = \theta_t + \alpha \, \mathbb{E}_{\mu}^{\pi^\theta} \Big[\sum_{t=0}^{T-1} \nabla_\theta \big(\log \pi^\theta(A_t \,;\, S_t) \big) \sum_{t'=t}^{T-1} R_{t+1} \Big].$$

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

$$\tilde{\nabla} J_\mu(\theta) = \frac{1}{K} \sum_{i=1}^{K} \Big[\sum_{t=0}^{T-1} \nabla_\theta \big(\log \pi^\theta(a_t^i \,;\, s_t^i) \big) \sum_{t'=t}^{T-1} r_{t'+1} \Big], \tag{5.1}$$

with K trajectories $(s_0^i, a_0^i, s_1^i, r_1^i, \ldots, a_{T-1}^i, s_T^i, r_T^i)$ sampled according to the current policy π^θ with initial distribution μ.
This famous algorithm, even not in exactly that form, is due to Ronald Williams[3]. 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 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.

[3]Williams: "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", Machine Learning, 8, 229-256, 1992

Algorithm 31: 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 $\left(s_0^i, a_0^i, s_1^i, r_1^i, \dots, a_{T-1}^i, s_T^i, r_T^i\right)$ using policy $\pi^{\theta_{l-1}}$.
  end
  Determine step-size α.
  $\tilde{\nabla} J(\theta_{l-1}) = \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{T-1} \nabla_\theta \big(\log \pi^{\theta_{l-1}}(a_t^i \,;\, s_t^i) \big) \sum_{t'=t}^{T-1} r_{t'+1}^i \right]$
  $\theta_l = \theta_{l-1} - \alpha \tilde{\nabla} J(\theta_{l-1})$
end
Set $l = l + 1$.

5.1.2 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 parametrised policy π^θ is given by

$$J_s(\theta) = V^{\pi^\theta}(s) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\Big] = \sum_{a \in \mathcal{A}_s} \pi^\theta(a; s) Q^{\pi^\theta}(s, a). \tag{5.2}$$

The infinite sum makes the computation of the gradient more challenging.

Lemma 5.1.6. Under the assumption that J_s is differentiable for every start state $s \in \mathcal{S}$ we have a closed form of the gradient

$$\nabla J_s(\theta) = \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}_{s'}} \rho_s^{\pi^\theta}(s') \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a),$$

where $\rho_\mu^\pi(s') = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_\mu^\pi(S_t = s') = \mathbb{E}_\mu^\pi[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t = s'}]$.

Proof. The trick is essentially simple but tedious to write down. One applies the gradient to the righthand side of (5.2) to find ∇J (this involves dynamic programming) in a the sum/product from the cain rules. Repeatadly applying the chain rule again, formally using an induction, leads to the claim. With the notation $p(s \to s'; n, \pi) = \mathbb{P}_s^\pi(S_n = s')$ we first show the following identity by induction:

For all $n \in \mathbb{N}$ it holds that

$$\begin{aligned} \nabla J_s(\theta) = &\sum_{t=0}^{n} \sum_{s' \in \mathcal{S}} \gamma^t p(s \to s'; t, \pi^\theta) \sum_{a \in \mathcal{A}_{s'}} \nabla \pi^\theta(a; s') Q^{\pi^\theta}(s', a) \\ &+ \sum_{s' \in \mathcal{S}} \gamma^{n+1} p(s \to s'; n+1, \pi^\theta) \nabla J_{s'}(\theta). \end{aligned} \tag{5.3}$$

By the chain rule we have that

$$\begin{aligned}
\nabla J_s(\theta) &= \sum_{a\in\mathcal{A}_s}\Big(\nabla\pi^\theta(a;s)Q^{\pi^\theta}(s,a)+\pi^\theta(a;s)\nabla Q^{\pi^\theta}(s,a)\Big)\\
&= \sum_{a\in\mathcal{A}_s}\Big(\nabla\pi^\theta(a;s)Q^{\pi^\theta}(s,a)+\pi^\theta(a;s)\nabla\big(r(s,a)+\gamma\sum_{s'\in\mathcal{S}}p(s';s,a)V^{\pi^\theta}(s')\big)\Big)\\
&= \sum_{a\in\mathcal{A}_s}\Big(\nabla\pi^\theta(a;s)Q^{\pi^\theta}(s,a)+\pi^\theta(a;s)\gamma\sum_{s'\in\mathcal{S}}p(s';s,a)\nabla J_{s'}(\theta)\Big)\\
&= \sum_{a\in\mathcal{A}_s}\Big(\nabla\pi^\theta(a;s)Q^{\pi^\theta}(s,a)+\pi^\theta(a;s)\gamma\sum_{s'\in\mathcal{S}}p(s';s,a)\Big(\sum_{a'\in\mathcal{A}_{s'}}\\
&\quad\Big(\nabla\pi^\theta(a';s')Q^{\pi^\theta}(s',a')+\pi^\theta(a';s')\gamma\sum_{s''\in\mathcal{S}}p(s'';s',a')\nabla J_{s''}(\theta)\Big)\Big)\Big)\\
&= \sum_{t=0}^{1}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\
&\quad+\sum_{s''\in\mathcal{S}}\gamma^2 p(s\to s'';2,\pi^\theta)\nabla J_{s''}(\theta).
\end{aligned}$$

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

$$\begin{aligned}
\nabla J_s(\theta) &= \sum_{t=0}^{n}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\
&\quad+\sum_{s'\in\mathcal{S}}\gamma^{n+1}p(s\to s';n+1,\pi^\theta)\nabla J_{s'}(\theta)\\
&= \sum_{t=0}^{n}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\
&\quad+\sum_{s'\in\mathcal{S}}\gamma^{n+1}p(s\to s';n+1,\pi^\theta)\\
&\quad\times\Big(\sum_{a'\in\mathcal{A}_{s'}}\Big(\nabla\pi^\theta(a';s')Q^{\pi^\theta}(s',a')+\pi^\theta(a';s')\gamma\sum_{s''\in\mathcal{S}}p(s'';s',a')\nabla J_{s''}(\theta)\Big)\Big)\\
&= \sum_{t=0}^{n+1}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\
&\quad+\sum_{s'\in\mathcal{S}}\gamma^{n+1}p(s\to s';n+1,\pi^\theta)\cdot\Big(\sum_{a'\in\mathcal{A}_{s'}}\pi^\theta(a';s')\gamma\sum_{s''\in\mathcal{S}}p(s'';s',a')\nabla J_{s''}(\theta)\Big)\\
&= \sum_{t=0}^{n+1}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\
&\quad+\sum_{s''\in\mathcal{S}}\gamma^{n+2}p(s\to s'';n+2,\pi^\theta)\nabla J_{s''}(\theta).
\end{aligned}$$

The remainder term in (5.3) vanishes for $n\to\infty$.

$$\Big|\sum_{s''\in\mathcal{S}}\gamma^{n+1}p(s\to s'';n+1,\pi^\theta)\nabla J_{s''}(\theta)\Big|\le\gamma^{n+1}|\mathcal{S}|\max_{s\in\mathcal{S}}|\nabla J_s(\theta)|\to 0,\quad n\to\infty.$$

Hence, we proved that

$$\begin{aligned}\nabla J_s(\theta) &= \sum_{t=0}^{\infty}\sum_{s'\in\mathcal{S}}\gamma^t p(s\to s';t,\pi^\theta)\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\ &= \sum_{s'\in\mathcal{S}}\rho_s^{\pi^\theta}(s')\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a).\end{aligned}$$

The exchange of limits and sums is justified as we assume $\mathcal{S}$ to be finite. □

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 5.1.6 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 5.1.7. The discounted state-visitation measure under policy π for starting state $s\in\mathcal{S}$ is given by

$$d_\mu^\pi(s) := \frac{\rho_\mu^\pi(s)}{\sum_{s'}\rho_\mu^\pi(s')}.$$

First note that, using Fubini's theorem,

$$\sum_{s'\in\mathcal{S}}\rho_\mu^{\pi^\theta}(s')=\sum_{s'\in\mathcal{S}}\mathbb{E}_\mu^\pi\Big[\sum_{t=0}^\infty\gamma^t\mathbf{1}_{S_t=s'}\Big]=\frac{1}{1-\gamma}.$$

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 5.1.8. Under the assumption that $J_s(\theta)$ is differentiable for every start state $s\in\mathcal{S}$ the gradient can be expressed as

$$\nabla J_s(\theta)=\frac{1}{1-\gamma}\mathbb{E}_{S\sim d_s^{\pi^\theta},A\sim\pi^\theta(\cdot\,;\,S)}\big[\nabla\log(\pi^\theta(A\,;\,S))Q^{\pi^\theta}(S,A)\big].$$

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 5.1.6 we have

$$\begin{aligned}\nabla J_s(\theta) &= \sum_{s'\in\mathcal{S}}\rho^{\pi^\theta}(s')\sum_{a\in\mathcal{A}_{s'}}\nabla\pi^\theta(a;s')Q^{\pi^\theta}(s',a)\\ &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}}\nabla\log(\pi^\theta(a;s'))Q^{\pi^\theta}(s',a)\pi^\theta(a;s')d_s^{\pi^\theta}(s')\\ &= \frac{1}{1-\gamma}\mathbb{E}_{S\sim d_s^{\pi^\theta},A\sim\pi^\theta(\cdot\,;\,S)}\big[\nabla\log(\pi^\theta(A\,;\,S))Q^{\pi^\theta}(S,A)\big],\end{aligned}$$

where we used the definition of the state-visitation measure and the log-trick. The second equality gives the first claim, the final equality the second claim. □

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:

Lemma 5.1.9. If $T \sim \mathrm{Geo}(1-\gamma)$ is independent of the MDP, then $d^\pi(s) = \mathbb{E}_s^\pi\big[\sum_{t=0}^T \mathbf{1}_{S_t=s}\big]$. In particular, the sampling of d^π can be carried out as follows. First run teh MDP until an independent $\mathrm{Geo}(1-\gamma)$ time and then sample from the obtained occupation measure.

This is easily seen as

$$\begin{aligned}
\mathbb{E}_s^\pi\Big[\sum_{t=0}^{T} \mathbf{1}_{S_t=s}\Big] &= \mathbb{E}_s^\pi\Big[\sum_{t=0}^{\infty} \mathbf{1}_{t\le T}\mathbf{1}_{S_t=s}\Big] \\
&\overset{\text{Fubini}}{=} \sum_{t=0}^{\infty} \mathbb{E}_s^\pi\big[\mathbf{1}_{t\le T}\mathbf{1}_{S_t=s}\big] \\
&\overset{\text{ind.}}{=} \sum_{t=0}^{\infty} \mathbb{P}(t\le T)\mathbb{E}_s^\pi\big[\mathbf{1}_{S_t=s}\big] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^\pi\big[\mathbf{1}_{S_t=s}\big] \\
&= \mathbb{E}_s^\pi\Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{S_t=s}\Big]
\end{aligned}$$

[4]

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 5.1.10. Suppose that $(s,a) \mapsto \nabla_\theta\big(\log \pi^\theta(a\,;\,s)\big)Q^{\pi^\theta}(s,a)$ is bounded. Then

$$\nabla_\theta J_s(\theta) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta\big(\log \pi^\theta(A_t\,;\,S_t)\big)Q^{\pi^\theta}(S_t,A_t)\Big].$$

[5] 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 imples the Q-function is bounded) the assumption of the theorem holds.

Proof.

$$\begin{aligned}
\nabla_\theta J_s(\theta) &= \sum_{t=0}^{\infty} \gamma^t \sum_{s'\in\mathcal{S}} \mathbb{P}_s^{\pi^\theta}(S_t=s') \sum_{a\in\mathcal{A}} \nabla\pi^\theta(a\,;\,s')Q^{\pi^\theta}(s',a) \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^\theta}\Big[\sum_{a\in\mathcal{A}} \nabla\pi^\theta(a\,;\,S_t)Q^{\pi^\theta}(S_t,a)\Big]
\end{aligned}$$

[4] allgemeines lemma schreiben, das gleiche braucht man auch bei dem performance difference lemma. $E[\sum \gamma^t F(S_t,A_t)] = \sum_{s,a} d(s)\pi(a,s)F(s,a)$

[5] brauche auch die version mit reward to go, beweis wie oben mit reinbedingen

$$
\begin{aligned}
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^\theta} \Big[\sum_{a \in \mathcal{A}} \pi^\theta(a\,;\, S_t) \nabla \log \pi^\theta(a\,;\, S_t) Q^{\pi^\theta}(S_t, a) \Big] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_s^{\pi^\theta} \Big[\nabla \log \pi^\theta(A_t\,;\, S_t) Q^{\pi^\theta}(S_t, A_t) \Big] \\
&= \mathbb{E}_s^{\pi^\theta} \Big[\sum_{t=0}^{\infty} \gamma^t \nabla \log \pi^\theta(A_t\,;\, S_t) Q^{\pi^\theta}(S_t, A_t) \Big]
\end{aligned}
$$

The final eschenge of sum and expectation is justified by dominated convergence and the assumption of the theorem. □

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. Under certain assumptions on the policy parametrisation this can be done.

Assumption 5.1.11. The policy π^θ is differentiable with respect to θ and $\nabla \log(\pi^\theta(a;s))$ exists and is L_Θ-smooth and has bounded norm for any $(s,a) \in \mathcal{S} \times \mathcal{A}$, i.e.

$$\|\nabla \log(\pi^{\theta_1}(a;s)) - \nabla \log(\pi^{\theta_2}(a|s))\| \leq L_\Theta \|\theta_1 - \theta_2\|, \quad \text{for any } \theta_1, \theta_1 \in \Theta,$$

$$\|\nabla \log(\pi^\theta(a;s))\| \leq B_\Theta, \quad \text{for any } \theta \in \Theta,$$

for some $L_\Theta, B_\Theta > 0$.

This assumption is for example fulfilled by tabular and linear softmax parametrisations or by Gaussian policies.

Show that the tabular and linear softmax parametrisation fulfills Assumption 5.1.11.

Using a computation similar to the one for the occupation measure gives the following theorem[6].

Proposition 5.1.12. Suppose that the policy parametrisation fulfills Assumption 5.1.11 and $T \sim \text{Geo}(1-\gamma)$, $T' \sim \text{Geo}(1-\gamma^{1/2})$ are independent of each other and the MDP, then

$$\nabla J_s(\theta) = \frac{1}{1-\gamma} \mathbb{E}_s^{\pi^\theta} \Big[\nabla \log(\pi^\theta(S_T\,;\, A_T)) \sum_{t=T}^{T+T'} \gamma^{(t-T)/2} R(S_t, A_t) \Big].$$

[7] 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. We split the proof into two steps.

The term $\hat{Q}^{\pi^\theta}(s,a) := \sum_{t'=0}^{T'} \gamma^{t'/2} R(S_{t'}, A_{t'})$ is an unbiased estimator of the Q-

[6] Zhang, Koppel, Zhu, Basar: "Global Convergence of Policy Gradient Methods to (Almost) Locally Optimal Policies", SIAM Journal on Control and Optimization, 2020

[7] doppeltes s, da sollte ein mu stehen

CD CD

function,

$$\mathbb{E}_s^{\pi_a^\theta}\Big[\sum_{t'=0}^{T'}\gamma^{t'/2}R(S_{t'},A_{t'})\Big]=Q^{\pi^\theta}(s,a).$$

First note that by the bounded reward assumption

$$\mathbb{E}_s^{\pi_a^\theta}\Big[\sum_{t=0}^{N}\mathbf{1}_{0\le t\le T'}\gamma^{t/2}R_{t+1}\Big]\le R_*\mathbb{E}_{T'}\Big[\sum_{t=0}^{N}\mathbf{1}_{0\le t\le T'}\gamma^{t/2}\Big].$$

If we write $E_s^{\pi^\theta}$ we mean the expectation with respect to all appearing random variables, i.e. also of T'. When we bound R by R_*, only the expectation with respect to T' is left, which is why we denote it with $\mathbb{E}_{T'}$. In the following we will write $\mathbb{E}_{T'}[\mathbb{E}_s^{\pi^\theta}[\cdot]]$ to highlight the independence of T' to the MDP. As the RHS of the inequality is monotonically increasing and the limit exists, we can use the monotone convergence theorem to show that

$$\begin{aligned}
\mathbb{E}_s^{\pi_a^\theta}\Big[\hat{Q}^{\pi^\theta}(s,a)\Big]&=\mathbb{E}_{T'}\Big[\mathbb{E}_s^{\pi_a^\theta}\Big[\sum_{t'=0}^{T'}\gamma^{t'/2}R(S_{t'},A_{t'})\Big]\Big]\\
&=\mathbb{E}_{T'}\Big[\mathbb{E}_s^{\pi_a^\theta}\Big[\lim_{N\to\infty}\sum_{t'=0}^{N}\mathbf{1}_{0\le t'\le T'}\gamma^{t'/2}R(S_{t'},A_{t'})\Big]\Big]\\
&=\lim_{N\to\infty}\sum_{t'=0}^{N}\mathbb{E}_{T'}\Big[\mathbb{E}_s^{\pi_a^\theta}\Big[\mathbf{1}_{0\le t'\le T'}\gamma^{t'/2}R(S_{t'},A_{t'})\Big]\Big]\\
&=\lim_{N\to\infty}\sum_{t'=0}^{N}\mathbb{E}_s^{\pi_a^\theta}\Big[\mathbb{E}_{T'}[\mathbf{1}_{0\le t'\le T'}]\gamma^{t'/2}R(S_{t'},A_{t'})\Big]\\
&=\lim_{N\to\infty}\sum_{t'=0}^{N}\mathbb{E}_s^{\pi_a^\theta}\Big[\gamma^{t'}R(S_{t'},A_{t'})\Big]\\
&=\lim_{N\to\infty}\mathbb{E}_s^{\pi_a^\theta}\Big[\sum_{t'=0}^{N}\gamma^{t'}R(S_{t'},A_{t'})\Big]\\
&=Q^{\pi^\theta}(s,a),
\end{aligned}$$

where we have used that T' is independent of the MDP and that $\mathbb{E}_{T'}[\mathbf{1}_{0\le t'\le T'}]=\mathbb{P}(T'\ge t)=\gamma^{t/2}$ by the Geo$(1-\gamma^{1/2})$ distribution. In the last equation we used the dominated convergence theorem to bring the limit back inside the expectation.

CD CD

For the second step we define the estimator $\widehat{\nabla}J_s(\theta)=\frac{1}{1-\gamma}\nabla\log(\pi^\theta(A_T;S_T))\hat{Q}(S_T,A_T)$. Then by definition

$$\frac{1}{1-\gamma}\mathbb{E}_s^{\pi^\theta}\Big[\nabla\log(\pi^\theta(S_T\,;\,A_T))\sum_{t=T}^{T+T'}\gamma^{(t-T)/2}R(S_t,A_t)\Big]=\mathbb{E}_s^{\pi^\theta}\Big[\widehat{\nabla}J_s(\theta)\Big].$$

It remains to show that $\mathbb{E}_s^{\pi^\theta}\Big[\widehat{\nabla}J_s(\theta)\Big]=\nabla J_s(\theta)$.

As T and T' are independent and independent of the MDP, we have directly that

$$\mathbb{E}_s^{\pi^\theta}\Big[\widehat{\nabla}J_s(\theta)\Big]=\frac{1}{1-\gamma}\mathbb{E}_s^{\pi^\theta}\Big[\nabla\log(\pi^\theta(S_T\,;\,A_T))\sum_{t=T}^{T+T'}\gamma^{(t-T)/2}R(S_t,A_t)\Big]$$

$$
\begin{aligned}
&= \frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_{s,T,T'}\Big[\nabla \log(\pi^\theta(S_T\,;\,A_T))\hat{Q}(S_T,A_T)\Big]\\
&= \frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_{s,T}\Big[\nabla \log(\pi^\theta(S_T\,;\,A_T))\mathbb{E}^{\pi^\theta_{A_T}}_{T',S_T}[\hat{Q}(S_T,A_T)]\Big]\\
&= \frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_{s,T}\Big[\nabla \log(\pi^\theta(S_T\,;\,A_T))Q(S_T,A_T)\Big],
\end{aligned}
$$

where the last equality is due to step 1. As Q is bounded by $\frac{R_*}{1-\gamma}$ and $\|\nabla \log(\pi^\theta(a;s))\| \leq B_\Theta$ by Assumption 5.1.11, we can again use the monotone convergence theorem as in step 1 to show that

$$
\begin{aligned}
\mathbb{E}^{\pi^\theta}_s\Big[\widehat{\nabla} J_s(\theta)\Big] &= \frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_{s,T}\Big[\nabla \log(\pi^\theta(S_T\,;\,A_T))Q(S_T,A_T)\Big]\\
&= \frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_{s,T}\Big[\sum_{t=0}^{\infty}\mathbf{1}_{t=T}\nabla \log(\pi^\theta(S_t\,;\,A_t))Q(S_t,A_t)\Big]\\
&= \lim_{N\to\infty}\sum_{t=0}^{N}\mathbb{E}_T[\mathbf{1}_{t=T}]\frac{1}{1-\gamma}\mathbb{E}^{\pi^\theta}_s\Big[\nabla \log(\pi^\theta(S_t\,;\,A_t))Q(S_t,A_t)\Big]\\
&= \lim_{N\to\infty}\sum_{t=0}^{N}\gamma^t\mathbb{E}^{\pi^\theta}_s\Big[\nabla \log(\pi^\theta(S_t\,;\,A_t))Q(S_t,A_t)\Big]\\
&= \mathbb{E}^{\pi^\theta}_s\Big[\sum_{t=0}^{\infty}\gamma^t\nabla \log(\pi^\theta(S_t\,;\,A_t))Q(S_t,A_t)\Big]\\
&= \nabla J_s(\theta),
\end{aligned}
$$

where we used the independence of T to the MDP and that $\mathbb{E}_T[\mathbf{1}_{t=T}]\frac{1}{1-\gamma} = \frac{\mathbb{P}(T=t)}{1-\gamma} = \gamma^t$. The last equation is due to the policy gradient theorem 5.1.10 □

Using this proposition results in the following REINFORCE algorithm with geometric rollouts.

Algorithm 32: Mini-batch REINFORCE for infinite time horizon

Data: Initial parameter θ_0, $K \geq 1$, initial state s
Result: Approximate policy $\pi^{\theta_L} \approx \pi^{\theta^*}$
$l = 1$
while *not converged* **do**
  for $i = 1, \dots, K$ **do**
    Sample $T_i \sim \text{Geo}(1-\gamma)$
    Sample trajectory $\Big(s^i_0 = s, a^i_0, s^i_1, r^i_1, \dots, a^i_{T-1}, s^i_{T_i}, r^i_{T_i}, a^i_{T_i}\Big)$ using policy $\pi^{\theta_{l-1}}$.
    Sample $T'_i \sim \text{Geo}(1-\gamma^{\frac{1}{2}})$
    Set $\tilde{s}^i_0 = s^i_{T_i}$ and $\tilde{a}^i_0 = a^i_{T_i}$
    Sample trajectory $\Big(\tilde{s}^i_0, \tilde{a}^i_0, \tilde{s}^i_1, \tilde{r}^i_1, \dots, \tilde{a}^i_{T'_i-1}, \tilde{s}^i_{T'_i}, \tilde{r}^i_{T'_i}, \tilde{a}^i_{T'_i}\Big)$ using policy $\pi^{\theta_{l-1}}$.
  end
  Determine step-size α.
  $\widehat{\nabla} J_s(\theta_{l-1}) = \frac{1}{1-\gamma}\frac{1}{K}\sum_{i=1}^{K}\Big[\nabla_\theta\big(\log \pi^{\theta_{l-1}}(a^i_{T_i}\,;\,s^i_{T_i})\big)\sum_{t'=0}^{T'_i-1}\gamma^{t'/2}\tilde{r}^i_{t'+1}\Big]$
  $\theta_l = \theta_{l-1} - \alpha\widehat{\nabla} J_s(\theta_{l-1})$
end
Set $l = l+1$.

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. We will see this in the subsequent section. Combined with an additional PI inequality (only known for tabular softmax parametrisation) then also yields convergence to an optimal policy.

5.1.3 Convergence of REINFORCE

Under certain assumptions we can prove convergence of REINFORCE to stationary points using Theorem 4.5.1. First, we will need L-smoothness, which can be shown under Assumption 5.1.11. Secondly, we need an unbiased estimator with bounded variance of the gradient $\nabla J(\theta)$ for any $\theta \in \mathbb{R}^d$. This can be achieved using the geometric rollouts discussed in the previous section. The unbiasedness has already be shown in Proposition 5.1.12 and the bounded variance is discussed below. The algorithm we consider is the REINFORCE Algorithm 32.
First we prove L-smoothness of the objective.

Lemma 5.1.13. Under Assumption 5.1.11, the objective $J_{s_0}(\theta)$ is L-smooth with $L = \frac{R_* L_\Theta}{(1-\gamma)^2} + \frac{(1+\gamma)R_* B_\Theta^2}{(1-\gamma)^3}$, where R_* is the maximal reward of the bounded reward assumption.

Proof. From the policy gradient theorem 5.1.8 we obtain for an initial state s_0 that

$$\nabla J_{s_0}(\theta) = \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}, a\in\mathcal{A}} d_{s_0}^{\pi^\theta}(s)\pi^\theta(a;s)\nabla \log(\pi^\theta(a;s))Q^{\pi^\theta}(s,a).$$

Now note that we can rewrite Q to be

$$Q^{\pi^\theta}(s,a) = \sum_{t'=0}^{\infty} \gamma^{t'} \sum_{s'\in\mathcal{S}, a'\in\mathcal{A}} p((s,a)\to s';t',\pi^\theta)\pi^\theta(a';s')r(s',a'),$$

where $p((s,a)\to s';t',\pi^\theta) = \mathbb{P}_s^{\pi_a^\theta}(S_{t'}) = s')$. Using the definition of the state-visitation measure and this form of Q^{π^θ} we obtain for the gradient

$$\begin{aligned}
&\nabla J_{s_0}(\theta)\\
&= \sum_{s\in\mathcal{S}, a\in\mathcal{A}} \sum_{t=0}^{\infty} \gamma^t p(s_0\to s;t,\pi^\theta)\pi^\theta(a;s)\nabla\log(\pi^\theta(a;s))Q^{\pi^\theta}(s,a)\\
&= \sum_{t=0}^{\infty}\gamma^t \sum_{s\in\mathcal{S}, a\in\mathcal{A}} p(s_0\to s;t,\pi^\theta)\pi^\theta(a;s)\nabla\log(\pi^\theta(a;s)) \sum_{t'=0}^{\infty}\gamma^{t'} \sum_{s'\in\mathcal{S}, a'\in\mathcal{A}} p((s,a)\to s';t',\pi^\theta)\pi^\theta(a';s')r(s',a')\\
&= \sum_{t=0}^{\infty}\sum_{t'=0}^{\infty}\gamma^{t+t'} \sum_{s\in\mathcal{S}, a\in\mathcal{A}, s'\in\mathcal{S}, a'\in\mathcal{A}} p(s_0\to s;t,\pi^\theta)\pi^\theta(a;s)p((s,a)\to s';t',\pi^\theta)\pi^\theta(a';s')\nabla\log(\pi^\theta(a;s))r(s',a').
\end{aligned}$$

For notation simplicity we define

$$f_{t,t'}^{s_0,\theta}(s,a,s',a') := p(s_0\to s;t,\pi^\theta)\pi^\theta(a;s)p((s,a)\to s';t',\pi^\theta)\pi^\theta(a';s')$$

and get

$$\nabla J_{s_0}(\theta) = \sum_{t=0}^{\infty}\sum_{t'=0}^{\infty}\gamma^{t+t'} \sum_{s\in\mathcal{S}, a\in\mathcal{A}, s'\in\mathcal{S}, a'\in\mathcal{A}} f_{t,t'}^{s_0,\theta}(s,a,s',a')\nabla\log(\pi^\theta(a;s))r(s',a').$$

For any θ_1 and θ_2 we can analyse the difference of gradients using this representation as follows.

$$
\begin{aligned}
&\|\nabla J_{s_0}(\theta_1) - \nabla J_{s_0}(\theta_2)\| \\
&= \Big\| \sum_{t=0}^{\infty}\sum_{t'=0}^{\infty} \gamma^{t+t'} \cdot \Big\{ \sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} f_{t,t'}^{s_0,\theta}(s,a,s',a')\big(\nabla\log(\pi^{\theta_1}(a;s)) - \nabla\log(\pi^{\theta_2}(a;s))\big) r(s',a') \\
&\quad + \sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} \big(f_{t,t'}^{s_0,\theta_1}(s,a,s',a') - f_{t,t'}^{s_0,\theta_2}(s,a,s',a')\big)\nabla\log(\pi^{\theta_2}(a;s)) r(s',a') \Big\} \Big\| \\
&\le \sum_{t=0}^{\infty}\sum_{t'=0}^{\infty} \gamma^{t+t'} \cdot \Big\{ \sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} f_{t,t'}^{s_0,\theta}(s,a,s',a') \big\|\nabla\log(\pi^{\theta_1}(a;s)) - \nabla\log(\pi^{\theta_2}(a;s))\big\| |r(s',a')| \\
&\quad + \sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} |f_{t,t'}^{s_0,\theta_1}(s,a,s',a') - f_{t,t'}^{s_0,\theta_2}(s,a,s',a')| \, \|\nabla\log(\pi^{\theta_2}(a;s))\| \, |r(s',a')| \Big\}.
\end{aligned}
$$

Using L_Θ-smoothness of the log-policy and the bounded reward assumption we have for the first term, that

$$
\sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} f_{t,t'}^{s_0,\theta}(s,a,s',a') \big\|\nabla\log(\pi^{\theta_1}(a;s)) - \nabla\log(\pi^{\theta_2}(a;s))\big\| |r(s',a')| \le R_* L_\Theta \|\theta_1 - \theta_2\|. \tag{5.4}
$$

For the second term we have to work a little harder to bound the difference of the probability densities $|f_{t,t'}^{s_0,\theta_1}(s,a,s',a') - f_{t,t'}^{s_0,\theta_2}(s,a,s',a')|$. Therefore, we first denote by $\mathcal{T}_t$ all trajectories from 0 to t, i.e.

$$
\mathcal{T}_t = \{\tau = \{s_0, a_0, \dots, s_t, a_t\} | s_0, a_0 \in \mathcal{A}_0, \dots, s_t \in \mathcal{S}, a_t \in \mathcal{A}_{s_t}\}
$$

and rewrite $f_{t,t'}^{s_0,\theta}(s,a,s',a')$ as follows

$$
\begin{aligned}
f_{t,t'}^{s_0,\theta}(s,a,s',a') &= p(s_0 \to s; t, \pi^\theta)\pi^\theta(a;s) p((s,a) \to s'; t', \pi^\theta)\pi^\theta(a';s') \\
&= \sum_{\tau\in\mathcal{T}_{t+t'}} \mathbf{1}_{s_t=s,a_t=a,s_{t'}=s',a_{t'}=a'} \prod_{n=0}^{t+t'} \pi^\theta(a_n;s_n) \prod_{n=0}^{t+t'-1} p(s_{n+1};s_n,a_n).
\end{aligned}
$$

So,

$$
\begin{aligned}
&f_{t,t'}^{s_0,\theta_1}(s,a,s',a') - f_{t,t'}^{s_0,\theta_2}(s,a,s',a') \\
&= \sum_{\tau\in\mathcal{T}_{t+t'}} \mathbf{1}_{s_t=s,a_t=a,s_{t'}=s',a_{t'}=a'} \Big(\prod_{n=0}^{t+t'} \pi^{\theta_1}(a_n;s_n) - \prod_{n=0}^{t+t'} \pi^{\theta_2}(a_n;s_n) \Big) \prod_{n=0}^{t+t'-1} p(s_{n+1};s_n,a_n).
\end{aligned}
$$

Then, using Taylor expansion (or also the more dimensional mean value or Lagrange theorem) of $\theta \mapsto \prod_{n=0}^{t+t'} \pi^\theta(a_n;s_n)$ we obtain the existence of a $\tilde\theta \in [\theta_1, \theta_2]$ such that

$$
\begin{aligned}
\Big| \prod_{n=0}^{t+t'} \pi^{\theta_1}(a_n;s_n) - \prod_{n=0}^{t+t'} \pi^{\theta_2}(a_n;s_n) \Big| &\le \Big| (\theta_1 - \theta_2)^T \nabla_\theta \Big(\prod_{n=0}^{t+t'} \pi^\theta(a_n;s_n) \Big)\Big|_{\theta=\tilde\theta} \Big| \\
&\le \|\theta_1 - \theta_2\| \Big\| \sum_{n=0}^{t+t'} \nabla\pi^{\tilde\theta}(a_n;s_n) \prod_{m=0,m\neq n}^{t+t'} \pi^{\tilde\theta}(a_m;s_m) \Big\| \\
&\le \|\theta_1 - \theta_2\| \sum_{n=0}^{t+t'} \big\|\nabla\log(\pi^{\tilde\theta}(a_n;s_n))\big\| \prod_{m=0}^{t+t'} \pi^{\tilde\theta}(a_m;s_m) \\
&\le \|\theta_1 - \theta_2\| (t+t'+1) B_\Theta \prod_{m=0}^{t+t'} \pi^{\tilde\theta}(a_m;s_m).
\end{aligned}
$$

Finally, we follow for the second term, that

$$\begin{aligned}
&\sum_{s\in\mathcal{S},a\in\mathcal{A},s'\in\mathcal{S},a'\in\mathcal{A}} |f_{t,t'}^{s_0,\theta_1}(s,a,s',a') - f_{t,t'}^{s_0,\theta_2}(s,a,s',a')| \, \|\nabla \log(\pi^{\theta_2}(a;s))\| \, |r(s',a')| \\
&\leq \sum_{\tau\in\mathcal{T}_{t+t'}} \Big| \prod_{n=0}^{t+t'} \pi^{\theta_1}(a_n;s_n) - \prod_{n=0}^{t+t'} \pi^{\theta_2}(a_n;s_n) \Big| \prod_{n=0}^{t+t'-1} p(s_{n+1};s_n,a_n) B_\Theta R_* \\
&\leq \|\theta_1-\theta_2\|(t+t'+1)B_\Theta^2 R_* \underbrace{\sum_{\tau\in\mathcal{T}_{t+t'}} \prod_{m=0}^{t+t'} \pi^{\theta}(a_m;s_m) \prod_{n=0}^{t+t'-1} p(s_{n+1};s_n,a_n)}_{=1} \\
&= \|\theta_1-\theta_2\|(t+t'+1)B_\Theta^2 R_*.
\end{aligned}$$

All in all, we obtain

$$\begin{aligned}
\|\nabla J_{s_0}(\theta_1) - \nabla J_{s_0}(\theta_2)\| &\leq \sum_{t=0}^{\infty}\sum_{t'=0}^{\infty} \gamma^{t+t'} (R_* L_\Theta \|\theta_1-\theta_2\| + \|\theta_1-\theta_2\|(t+t'+1)B_\Theta^2 R_*) \\
&\leq L\|\theta_1-\theta_2\|,
\end{aligned}$$

for $L = \frac{R_* L_\Theta}{(1-\gamma)^2} + \frac{(1+\gamma)R_* B_\Theta^2}{(1-\gamma)^3}$, where we used that

$$\sum_{t=0}^{\infty}\sum_{t'=0}^{\infty} \gamma^{t+t'}(t+t'+1) = \frac{1+\gamma}{(1-\gamma)^3}.$$

□

Secondly, we show that the estimator used in Algorithm 32 has bounded variance.

Lemma 5.1.14. The estimator $\widehat{\nabla} J_s(\theta)$ proposed in Algorithm 32 for $K=1$ has bounded variance. More precisely,

$$\|\widehat{\nabla} J_s(\theta)\| \leq \frac{B_\Theta R_*}{(1-\gamma)(1-\gamma^{1/2})}$$

and therefore,

$$\mathbb{E}[\|\widehat{\nabla} J_s(\theta) - \nabla J(\theta)\|^2] \leq C$$

for $C = R_*^2 B_\Theta^2 \Big(\frac{1}{(1-\gamma)^2(1-\gamma^{1/2})^2} + \frac{2}{(1-\gamma)^3(1-\gamma^{1/2})} + \frac{1}{(1-\gamma)^4} \Big)$.

Proof. By the definition of $\widehat{\nabla} J_s(\theta)$ we have that

$$\begin{aligned}
\|\widehat{\nabla} J_s(\theta)\| &= \Big\| \frac{1}{1-\gamma} \nabla_\theta \log(\pi^\theta(A_T\,;\,S_T)) \sum_{t'=0}^{T'-1} \gamma^{t'/2} R(S_{T+t'}, A_{T+t'}) \Big\| \\
&= \frac{1}{1-\gamma} \|\nabla_\theta \log(\pi^\theta(A_T\,;\,S_T))\| \sum_{t'=0}^{T'-1} \gamma^{t'/2} |R(S_{T+t'}, A_{T+t'})| \\
&\leq \frac{B_\Theta R_*}{(1-\gamma)} \sum_{t'=0}^{T'-1} \gamma^{t'/2} \\
&= \frac{B_\Theta R_*}{(1-\gamma)(1-\gamma^{1/2})}.
\end{aligned}$$

For the second claim, first note that

$$\begin{aligned}\|\nabla J_s(\theta)\| &= \|\frac{1}{1-\gamma}\mathbb{E}_{S\sim d_s^{\pi^\theta}, A\sim\pi^\theta(\cdot\,;\,S)}\big[\nabla\log(\pi^\theta(A\,;\,S))Q^{\pi^\theta}(S,A)\big]\|\\ &\leq \frac{1}{1-\gamma}\mathbb{E}_{S\sim d_s^{\pi^\theta}, A\sim\pi^\theta(\cdot\,;\,S)}\big[\|\log(\pi^\theta(A\,;\,S))\|\,Q^{\pi^\theta}(S,A)\big]\\ &\leq \frac{1}{1-\gamma}\mathbb{E}_{S\sim d_s^{\pi^\theta}, A\sim\pi^\theta(\cdot\,;\,S)}\big[B_\Theta\,\frac{R_*}{1-\gamma}\big]\\ &= \frac{B_\Theta R_*}{(1-\gamma)^2}.\end{aligned}$$

Hence, we have that

$$\begin{aligned}&\mathbb{E}[\|\widehat{\nabla}J_s(\theta)-\nabla J_s(\theta)\|^2]\\ &\leq \mathbb{E}[\|\widehat{\nabla}J_s(\theta)\|^2] + 2\mathbb{E}[\|\widehat{\nabla}J_s(\theta)\|]\|\nabla J_s(\theta)\| + \|\nabla J_s(\theta)\|^2\\ &\leq \frac{R_*^2B_\Theta^2}{(1-\gamma)^2(1-\gamma^{1/2})^2} + 2\frac{R^*B_\Theta}{(1-\gamma)(1-\gamma^{1/2})}\frac{R_*B_\Theta}{(1-\gamma)^2} + \frac{R_*^2B_\Theta^2}{(1-\gamma)^4}\\ &= R_*^2B_\Theta^2\Big(\frac{1}{(1-\gamma)^2(1-\gamma^{1/2})^2} + \frac{2}{(1-\gamma)^3(1-\gamma^{1/2})} + \frac{1}{(1-\gamma)^4}\Big).\end{aligned}$$

Define $C = R_*^2B_\Theta^2\Big(\frac{1}{(1-\gamma)^2(1-\gamma^{1/2})^2} + \frac{2}{(1-\gamma)^3(1-\gamma^{1/2})} + \frac{1}{(1-\gamma)^4}\Big)$ proves the claim. □

Theorem 5.1.15. Assume the objective function $\theta \mapsto J(\theta)$ is L-smooth and the policy parametrisation fulfills Assumption 5.1.11. Consider the stochastic process $(\theta_n)_{n\geq 0}$ generated in Algorithm 32 for $K = 1$, where the step-sizes $(\alpha_l)_{l\in\mathbb{N}}$ (deterministic or $\mathcal{F}$-adapted) satisfy

$$\alpha_l > 0, \quad \sum_{l=0}^{\infty}\alpha_l = \infty \quad \text{and} \quad \sum_{l=0}^{\infty}\alpha_l^2 < \infty$$

(almost surely). Then,

$$\lim_{l\to\infty}\|\nabla J_s(\theta_l)\|^2 = 0$$

almost surely.

Proof. First note that we consider a maximisation problem instead of a minimisation problem and therefore have to consider $-J$ in the SGD Theorem 4.5.1. By the bounded reward assumption we know that $J_{s,*} = \sup_{\theta\in\mathbb{R}^d} J_s(\theta) \leq \frac{R_*}{1-\gamma} < \infty$, i.e. $\inf_\theta -J_s(\theta) > -\infty$. Furthermore, we know that J_s (and thus also $-J_s$) is L-smooth. We have by $\widehat{\nabla}J_s(\theta)$ from Algorithm 32 an estimator which fulfills the equations (4.10) and (4.9) shown in Proposition 5.1.12 and Lemma 5.1.14. Hence, defining

- $F = -J_s$,
- $x = \theta$ and Z takes the roll of the MDP rollouts up to the geometric length, i.e. $Z(x) = (T, T', S_T, A_T, S_{T+1}, \ldots, S_{T+T'}, A_{T+T'})(\theta)$ under policy π^θ,
- $\nabla_x f(x,Z) = \frac{1}{1-\gamma}\nabla_\theta\log(\pi^\theta(S_T\,;\,A_T))\sum_{t=T}^{T+T'}\gamma^{(t-T)/2}R(S_t,A_t)$,

leads to almost sure convergence:

$$\lim_{l\to\infty}\|\nabla J_s(\theta_l)\|^2 = 0.$$

□

5.1.4 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 5.1.16. Assume that (S, A, R) is a T-step MDP with finite state-action spaces and consider a stationary differentiable parametrized family of policies $\{\pi^\theta : \theta \in \Theta\}$. For any $b \in \mathbb{R}$ the gradient of $J(\theta)$ can also be written as

$$\nabla_\theta J(\theta) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1} \nabla_\theta\big(\log \pi^\theta(A_t\,;\,S_t)\big)\underbrace{\big(Q_t^{\pi^\theta}(S_t,A_t)-b\big)}_{\text{or } (R_t^T-b) \text{ or } (R_0^T-b)}\Big].$$

Proof. The computation is very similar to the bandit case, we redo the log-trick and see that for every $t \le T-1$

$$\begin{aligned}
\mathbb{E}_s^{\pi^\theta}\big[\nabla_\theta\big(\log \pi^\theta(A_t\,;\,S_t)\big)b\big] &= b\sum_{s_t\in\mathcal{S}}\sum_{a_t\in\mathcal{A}_s}\mathbb{P}_s^{\pi^\theta}(S_t=s_t)\pi^\theta(a_t\,;\,s_t)\nabla_\theta\big(\log\pi^\theta(a_t\,;\,s_t)\big)\\
&= b\sum_{s_t\in\mathcal{S}}\mathbb{P}_s^{\pi^\theta}(S_t=s_t)\sum_{a_t\in\mathcal{A}_s}\nabla_\theta\pi^\theta(a_t\,;\,s_t)\\
&= b\sum_{s_t\in\mathcal{S}}\mathbb{P}_s^{\pi^\theta}(S_t=s_t)\nabla_\theta\underbrace{\sum_{a_t\in\mathcal{A}}\pi^\theta(a_t\,;\,s_t)}_{=1} = 0.
\end{aligned}$$

Thus, the additional term vanishes in all three representations of the gradient so that $-b$ can be added. □

Show that the constant baseline b can be replaced by any deterministic state-dependent baseline $b : \mathcal{S} \to \mathbb{R}$, i.e.

$$\nabla_\theta J(\theta) = \mathbb{E}_s^{\pi^\theta}\Big[\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(A_t\,;\,S_t)\big)\big(Q_t^{\pi^\theta}(S_t,A_t)-b(S_t)\big)\Big].$$

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 proof 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 know 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 uses 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 behaviro policy π^b and compensate by inserting the likelihood ratio. Here is the version of the finite-time non-discounted case:

Proposition 5.1.17. (Likelihood ratio trick for policy gradients) Suppose π^b is a behavior policy (with strictly positive weights), then

$$\nabla_\theta J_\mu(\theta) = \mathbb{E}_\mu^{\pi^b}\Big[\prod_{i=0}^{T-1}\frac{\pi^\theta(A_i\,;\,S_i)}{\pi^b(A_i\,;\,S_i)}\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(A_t\,;\,S_t)\big)Q_t^{\pi^\theta}(S_t,A_t)\Big]$$

Proof. The claim follows from the definition of discrete expectations $\mathbb{E}[g(X)] = \sum_k g(k)\mathbb{P}(X=k)$ and the MDP likelihood trick:

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}_\mu^{\pi^\theta}\Big[\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(A_t\,;\,S_t)\big)Q_t^{\pi^\theta}(S_t,A_t)\Big] \\
&= \sum_{\tau\text{: trajectory}}\Big(\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(a_t\,;\,s_t)\big)Q_t^{\pi^\theta}(s_t,a_t)\Big)\mathbb{P}_\mu^{\pi^\theta}(\tau) \\
&= \sum_{\tau\text{: trajectory}}\Big(\frac{\mathbb{P}_\mu^{\pi^\theta}(\tau)}{\mathbb{P}_\mu^{\pi^b}(\tau)}\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(a_t\,;\,s_t)\big)Q_t^{\pi^\theta}(s_t,a_t)\Big)\mathbb{P}_\mu^{\pi^b}(\tau) \\
&= \sum_{\tau\text{: trajectory}}\Big(\prod_{i=0}^{T-1}\frac{\pi^\theta(a_i\,;\,s_i)}{\pi^b(a_i\,;\,s_i)}\sum_{t=0}^{T-1}\nabla_\theta\big(\log\pi^\theta(a_t\,;\,s_t)\big)Q_t^{\pi^\theta}(s_t,a_t)\Big)\mathbb{P}_\mu^{\pi^b}(\tau)
\end{aligned}$$

$$= \mathbb{E}_{\mu}^{\pi^b}\Big[\sum_{t=0}^{T-1}\prod_{i=0}^{T-1}\frac{\pi^\theta(A_i\,;\,S_i)}{\pi^b(A_i\,;\,S_i)}\nabla_\theta\big(\log\pi^\theta(A_t\,;\,S_t)\big)Q_t^{\pi^\theta}(S_t,A_t)\Big]$$

□

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 succesful. 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 instance[8]. 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.

5.1.5 Natural policy gradient

[8]Metelli, Papini, Faccio, Restelli: "Policy Optimization via Importance Sampling", NIPS 2018

Chapter 6

Reinforcement learning with function approximation

Tabular methods generally assumed small state- and action-spaces. Small in the sense that in principle all Q-values $Q(s,a)$ could be considered separately in a table. In reality the number of states/actions is typically way too big. Additionally, it seems unpractical to estimate (learn) all Q-values separately as small changes in states might only have little impact on $Q(s,a)$. For instance, if s is a picture such as the state of an Atari game. In that case improved learning of $Q(s,a)$ should also effect $Q(s',a)$ for s' similar (in some sense) to s. As an illustration one might think of a function $f:\mathbb{N}^d \to \mathbb{R}$ that should be learned in some learning context. If $f(n) = a \cdot n + b$ is (affine) linear it would be completely unreasonable to learn all values $f(n)$ separately, only the vectors a and b would have to be estimated. If linear functions are a good approximation to the true function f then it might still be much more reasonable to estimate two parameters for an approximation instead of approximating all true values $f(n)$. Exactly this happens in RL with function approximation, the state-value function V or the state-action value function Q is approximated using a parametric class of functions V_w (resp. Q_w) and instead of finding estimates $\hat{V}(s)$ for all $s \in S$ (or $\hat{Q}(s,a)$ for all s,a) a parameter-vector w is estimated that best approximates $V(s)$ for all s (or $Q(s,a)$ for all s,a). In this chapter we will mostly deal with linear function approximation, in practice V_w or Q_w are neural network functions and w is the vector with all weights. As in dynamic programming there are different possible tasks:

- policy evaluation with function approximation, i.e. find V_w from the approximating parametric class that best approximates V^π (or Q_w that best approximates Q^π),
- policy iteration with function approximation, i.e. alternate between policy evaluation with function approximation and policy improvement, which can either be model-based or model-free using samples,
- value iteration with function approximation or a sample based Q-learning with function approximation,
- policy gradient with function approximation.

The good news is that even though it's complicated to make it work, in many settings dynamic programming or policy gradient with function approximation works. Bad news are lack of mathematical understanding. In very restricted settings proofs can be given but for the most relevant use of function approximation there are still large gaps in the understanding.

6.1 Policy evaluation with function approximation

In this section we deal with the evaluation of a fixed policy π, i.e. the computation of the state-value function V^π and the state-action-value function Q^π. Of course, we already know

exact and approximative tabular solution methods that use the theory of dynamic programming and stochastic approximation to obtain estimates of V^π and Q^π.
The aim of this section is to adapt the algorithms from the tabular case to obtain more general methods that do not require us to store estimates of every entry of V^π and Q^π. Estimates of V^π and Q^π will not be stored as lookup-tables (or vectors), where estimates for each of the entries of $V^\pi(s)$ and/or $Q^\pi(a,s)$ individually. Instead, we will use a parametrized family $\{f_w : w \in \mathbb{R}^d\}$. Typically the number d of parameters will be much smaller than the number of states (or state-action pairs) which means that we will have to tune far fewer parameters than in our previous tabular algorithms. While this is definitely an advantage, we cannot expect convergence to the value functions V^π and Q^π once approximations are introduced. One obvious reason for that is that they may not be within the set of functions that we can represent exactly using our chosen parametrization. In this new setting, the following questions appear naturally:

(i) Does a given algorithm converge to something?

(ii) If the algorithm does convergence, is the limit close (in some sense) to the true value function V^π (resp. Q^π)?

(iii) If the algorithm does not converge, does it at least oscillate within some small region around V^π (resp. Q^π)?

6.1.1 Function approximation

We will broadly classify the parametrized approximation architectures as either linear or non-linear. Linear architectures approximate the state-value function as follows:

$$f_w(s) = w \cdot \phi(s) = \sum_{i=1}^{d} w_i \phi_i(s),$$

where $\phi : \mathcal{S} \to \mathbb{R}^d$ and $\phi_i(s)$ denotes the i-th entry of $\phi(s)$. Here ϕ is a fixed mapping from $\mathcal{S}$ to $\mathbb{R}^d$ called the feature extractor, ϕ should extract the most important features of a state. Feature vectors can be anything, a good choice for concrete problems is clearly a tricky task. Essentially, using feature extraction the state-space is reduced to a subset of $\mathbb{R}^d$ and we approximate a function $\mathbb{R}^d \to \mathbb{R}$ by linear functions only. The linear architecture for Q^π is entirely analogous:

$$f_w(s,a) = w \cdot \phi(s,a) = \sum_{i=1}^{d} w_i \phi_i(s,a),$$

with $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$. In this section we will only deal with policy evaluation for the state-value function, the action-values can be handled similarly as the Bellman operator is essentially the same. As we will see soon, convergence of many algorithms can only be guaranteed in the case of linear approximation architectures. Still, non-linear parametrizations like neural network funcations are very popular in practical applications of RL (in this particular example, w would denote the weights and biases of the network). Fortunately, linear methods still allow for a good deal of flexibility as we will discuss below.

Example 6.1.1. Let us partition $\mathcal{S}$ into m disjoint subsets $\mathcal{S}_i$, i.e. $\mathcal{S} = \dot{\bigcup}_{i=1}^{m} \mathcal{S}_i$ with $\mathcal{S}_i \cap \mathcal{S}_j = \varnothing$ for all $i \neq j$. Define

$$\phi : \mathcal{S} \to \mathbb{R}^m \text{ such that } \phi_i : s \mapsto \mathbf{1}_{S_i}(s).$$

Then states from a fixed partition set are treated equally and the linear function approximation simplifies to $f_w(s) = w_i$ for all $s \in \mathcal{S}_i$. This particularly simple linear function approximation is referred to as state eaggregation. State aggregation can for instance be useful in a computer game where many of the possible images on the screen may have near identical state values because many features (for instance the sun in the background of super mario) of the image are irrelevant for the game.

It is not surprising that many rigorous results exist for state aggregation. In a way it first simplifies the problem and then treats the remaining problem as a tabular problem.

Tabular reinforcement learning can be seen as reinforcement learning with function approximation. Tabular is nothing but state aggregation with the finest partition $\mathcal{S}_i = \{s_i\}$.

Example 6.1.2. From the more familiar tasks of regression and interpolation we already that polynomials are good at approximating real-valued functions. It is not so surprising, then, that polynomials can be used in tasks where the states can be expressed as numbers, e.g. positions or angles in a robotics task. Suppose, for example, $\mathcal{S} \subseteq \mathbb{R}^2$ and let

$$\phi(s) = (1, s_1, s_2, s_1 s_2) \quad \text{so that} \quad f_w(s) = w_1 + w_2 s_1 + w_3 s_2 + w_4 s_1 s_2,$$

where the last term takes into account interactions between the dimensions. Of course, we could also use higher-dimensional polynomials to model more complex interactions, say $\phi(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_1^2 s_2^3, s_1^3, s_2^3)$ if we expect V^π to behave like a cubic function. This already highlights how we should incorporate prior knowledge about the nature of the task into the construction of useful feature vectors.

6.1.2 Sample based policy evaluation with function approximation

All function approximation algorithms must use some notion of error between functions to evaluate the quality of the function approximation. To give simple algorithms the notion should be useful and simple. The most classical one is the weighted mean-squared error:

Definition 6.1.3. Let μ a (discrete) probability measure on $\mathcal{S}$ and $\{f_w : w \in \mathbb{R}^d\}$ a set of approximation architectures. We define the mean square value error as

$$E(w) = \frac{1}{2} \sum_{s \in \mathcal{S}} \mu(s) \big(V^\pi(s) - f_w(s)\big)^2 = \frac{1}{2} \mathbb{E}_{S \sim \mu} \big[(V^\pi(S) - f_w(S))^2 \big].$$

The weights μ are supposed to indicate the importance of the respective states, thus, a natural choice is the (discounted) state visitation measure. Ideally, we want to find w such that $E(w)$ attains its global minimum. If our parametrized function class contains the true value function the value of the minimum will, of course, be 0. But there is no reason that should be the case, and typically it won't. It should be noted that it is not clear that E is the right performance objective for reinforcement learning. Our reason to even learn the value function is (at least most of the time) to obtain a better policy. However, the best value function approximation need not be the one that minimizes E. Even so, it is not obvious what a sensible alternative goal would be.

The following discussion will be kept somewhat informal to illustrate the important points, a formal treatment follows below. Least-squares problem as minimising E appear in many contexts, solution methods are quite standard. If the approximation architecture is linear the minimisation problem can be solved explicitly but involves non-trivial matrix inversions. In general, as long as the approximation architecture is differentiable the minimum (at least a local minimum) can be obtained using gradient descent on the parameters as the gradient can be obtained easily from the chain rule. This yields

$$\begin{aligned} \nabla_w E(w) &= \frac{1}{2} \sum_{s \in \mathcal{S}} \mu(s) \nabla_w \big(V^\pi(s) - f_w(s)\big)^2 \\ &= -\sum_{s \in \mathcal{S}} \mu(s) \big(V^\pi(s) - f_w(s)\big) \nabla_w f_w(s) \\ &= \mathbb{E}_{S \sim \mu} \big[\big(f_w(S) - V^\pi(S)\big) \nabla_w f_w(S) \big]. \end{aligned} \tag{6.1}$$

What we see is that the gradient takes precisely the form needed for stochastic gradient algorithms (SGD). In order for SGD algorithms to work, a minimal necessary requirement is that we should use unbiased estimates of the gradient of the potential function. For the sake of argument, assume just for now that we somehow have access to the value function. As suggested by the calculations above, a sample based SGD algorithm for calculating the best function approximation would be given by

$$w_{n+1} = w_n + \alpha_n \left(V^\pi(s_{n+1}) - f_{w_n}(s_{n+1})\right) \nabla_w f_{w_n}(s_{n+1}), \quad w_0 \in \mathbb{R}^d,$$

where the states $s_1, ...$ are sampled independently according to μ and step-sizes that satisfy decay rates depending on the problem (most of the time the Robbins-Monro conditions work). Obviously, we do not have access to the value function in practice. Therefore, we will have to replace $V^\pi(s_n)$ by estimates U_t which leads to the following general algorithm:

$$w_{n+1} = w_n + \alpha_n \left(U_{n+1} - f_{w_n}(s_{n+1})\right) \nabla_w f_{w_n}(s_{n+1}), \quad w_0 \in \mathbb{R}^d. \tag{6.2}$$

If f is a linear architecture, then the algorithm simplifies to

$$w_{n+1} = w_n + \alpha_n \left(U_{n+1} - w_n \cdot \phi(s_{n+1})\right) \phi(s_{n+1}), \quad w_0 \in \mathbb{R}^d.$$

We can get creative here and chose U_t as one of the estimates of the value function from sample-based dynamic programming. In theory the simplest choice of U is a Monte Carlo estimator where independent samples are generated:

Example 6.1.4. Setting $U_n = \sum_{t=0}^\infty \gamma^t R(S_{t+1}^n, A_{t+1}^n)$ with rollouts $(S_t^n, A_t^n)_{t\geq 0}$ sampled according to the policy π with starting state $S_0^n = s_n$, the Monte Carlo version of policy evaluation with function approximation is

$$w_{n+1} = w_n + \alpha_n \Big(\underbrace{\sum_{t=0}^\infty \gamma^t R(S_{t+1}^{n+1}, A_{t+1}^{n+1})}_{\text{Theoretical Monte Carlo estimator for } V^\pi(s_{n+1})} - f_{w_n}(s_{n+1}) \Big) \nabla_w f_{w_n}(s_{n+1}), \quad w_0 \in \mathbb{R}^d.$$

This is somewhat a double stochastic gradient scheme, sampling independently from states and total reward distributions. We will see below that for linear function architectures convergence follows directly from standard stochastic gradient theorems.

Similarly to sample-based dynamic programming the Monte Carlo approach is inefficient, there is no bootstrapping of rollouts. Rollouts are only used once, with the advantage that there is a lot of independence that facilitates convergence guarantees. Reusing rollouts by including the already estimated value function yields a one-step temporal difference approach (TD(0)). Recall that in $TD(0)$ a value function estimation is improved by only resampling one times-step and then reusing the old estimate of V^π:

$$V^\pi(s) \leftarrow R(s,a) + \gamma V^\pi(s'),$$

where $a \sim \pi(\cdot\,;\, s)$ and $s' \sim p(\cdot\,;\, s, a)$ is a one-step sample. While this sounds plausible, a formal justification is throught the Bellman operator written in terms of expectations. Since V^π is unknown it is plausible to replace V^π by the best-known current approximation of V^π. The corresponding update formula is as follows:e

Example 6.1.5. The TD(0) update rule for policy evaluation with function approximation is

$$w_{n+1} = w_n + \alpha_n \big(\underbrace{R(s_{n+1}, a) + \gamma f_{w_n}(s')}_{\text{TD(0) estimator for } V^\pi(s_{n+1})} - f_{w_n}(s_{n+1}) \big) \nabla_w f_{w_n}(s_{n+1}), \quad w_0 \in \mathbb{R}^d, \tag{6.3}$$

where $a \sim \pi(\cdot\,;\, s_{n+1})$ and $s' \sim p(\cdot\,;\, s_{n+1}, a)$.

Analogously to sample based dynamic programming one-step temporal differences are also replaced by n-steps or a TD(λ) version:

Example 6.1.6. The N-step temporal difference update rule for policy evaluation with function approximation is

$$w_{n+1} = w_n + \alpha_n \Big(\underbrace{\sum_{t=0}^{N-1} \gamma^t R(S_t^{n+1}, A_t^{n+1}) + \gamma^N f_{w_n}(S_N^n)}_{N\text{-step TD estimator for } V^\pi(s_{n+1})} - f_{w_n}(s_{n+1}) \Big) \nabla_w f_{w_n}(s_{n+1}), \tag{6.4}$$

where $A_0^n, S_0^n, ..., S_N^n$ are MDP rollouts started in s_n.

Using approximators f_w to estimate V^π destroys the gradient descent property of the algorithm. The update rule is not anymore obtained from the chain rule of the least-square error! In reinforcement learning such algorithms are often called semi-gradient, they look like gradient descent updates but aren't. As a matter of fact, most algorithms do not converge, and if so, the analysis does not follow from gradient descent theorems but from the very specific form of the iteration.

Up to now the discussion was informal and, in fact, problematic. There is a big problem with the chain rule. If, as we did for the TD(0) update, the value function is replaced by a function approximation f_w, then the dependence of w yields a different derivative and the gradient descent update is not (6.4). Nonetheless, the algorithm could still be reasonable and (at least for linear function approximation) actually is.

Theorem 6.1.7. (Stochastic Gradient Descent)
Suppose $(\Omega, \mathcal{A}, \mathcal{F}_n)$ is a filtered probability with $\mathcal{F}_{n+1}$-measurable errors ε_n and define the recursion

$$r_{n+1} = r_n - \alpha_t \big(\nabla g(r_n) + \varepsilon_n \big)$$

where $g : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is a cost function. We assume that the stepsizes α_n are nonnegative, $\mathcal{F}_n$-measurable and satisfy

$$\sum_{n=0}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=0}^{\infty} \alpha_n^2 < \infty.$$

g is assumed to be L-smooth, i.e. g is differentiable and the gradient ∇g is L-Lipschitz continuous. In addition, we assume that the noise terms are $\mathcal{F}_{n+1}$-measurable and conditionally unbiased in the sense that

$$\mathbb{E}[\varepsilon_n \,|\, \mathcal{F}_n] = 0$$

and have sufficiently small variances

$$\mathbb{E}[\|\varepsilon_n\|^2 \,|\, \mathcal{F}_n] \leq A + B\|\nabla g(r_n)\|^2$$

for some constants $A, B \in \mathbb{R}$. Under these assumptions the sequence $(g(r_n))_{n\in\mathbb{N}}$ converges almost surely and $\lim_{n\to\infty} \nabla g(r_n) = 0$. Moreover, every limit point of $(r_n)_{n\in\mathbb{N}}$ is a stationary point of g.

Note that the theorem only requires L-smoothness on g but no convexity assumption. Hence, the result is only convergence to a critical point but not convergence to a global minimum [1]. To prove at least something we can show without much effort the convergence of the gradient descent based value prediction algorithm with linear function approximation in the case the value function is estimated without bias (e.g. with Monte Carlo estimation):

[1] The proof of this theorem can be found in Bertsekas' and Tsitsiklis' book *Neuro-Dynamic Programming*

Theorem 6.1.8. Suppose $(w_n)_{n\in\mathbb{N}}$ is the sequence from Example 6.1.4 initialised in some $w_0 \in \mathbb{R}^d$, i.e. approximate policy evaluation with Monte Carlo estimates. If features and rewards are bounded and the approximation architecture is linear, then $(w_n)_{n\in\mathbb{N}}$ converges almost surely to the global mimimum of E.

The linear least-squared problem can also be solved explicitly by matrix inversion. In contrast to gradient descent methods (with provable convergence or not) the direct approach does not extend to non-linear approximation architectures. To give a first impression we discuss a simple convergence proof for the simple linear situation with Monte Carlo estimates.

Proof. Since the f_w are assumed linear the mean-squared error is a quadratic function. Thus, it is L-smooth (the derivative is linear) and has a unique global minimum (which is the only critical point). Furthermore, the gradient is

$$\nabla_w E(w) \overset{(6.1)}{=} \mathbb{E}_\mu \left[(f_w(S) - V^\pi(S)) \nabla_w f_w(S) \right].$$

To link the update scheme formally to the stochastic gradient scheme we write

$$w_{n+1} = w_n + \alpha_n \big[(U_{n+1} - f_w(S_{n+1})) \nabla_w f_{w_n}(S_{n+1}) \big] = w_n + \alpha_n \big[\nabla_w E(w_n) + \varepsilon_n \big],$$

with

$$\varepsilon_n := (f_{w_n}(S_{n+1}) - U_{n+1}) \nabla_w f_{w_n}(S_{n+1}) - \nabla_w E(w_n),$$

where U_n is a Monte Carlo sample of $V^\pi(S_n)$ generated independently of $S_n \sim \mu$. Next, denote by $\mathcal{F}_n$ the filtration generated by all random variables of the algorithm to produce w_n. These are all α_k with $k < n$ and all rollouts (S^k, A^k) that result in U_k with $k < n$. Then U_n is independent of $\mathcal{F}_n$ and $\mathcal{F}_{n+1}$-measurable (thus, ε_n is $\mathcal{F}_{n+1}$-measurable). Furthermore, S_n is independent of $\mathcal{F}_n$. Hence, the tower property and measurabilty/independence properties of conditional expectation yield

$$\begin{aligned}
&\mathbb{E}[\varepsilon_n \mid \mathcal{F}_n] \\
&= \mathbb{E}[f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \mathbb{E}[U_{n+1} \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \nabla_w E(w_n) \\
&= \mathbb{E}[f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \mathbb{E}[\mathbb{E}[U_{n+1} \nabla_w f_{w_n}(S_{n+1}) \,|\, \sigma(\mathcal{F}_n, S_{n+1})] \,|\, \mathcal{F}_n] - \nabla_w E(w_n) \\
&= \mathbb{E}[f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \mathbb{E}[\mathbb{E}[U_{n+1} \,|\, \sigma(\mathcal{F}_n, S_{n+1})] \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \nabla_w E(w_n) \\
&= \mathbb{E}[f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \mathbb{E}[\mathbb{E}[U_{n+1} \mid S_{n+1}] \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \nabla_w E(w_n) \\
&= \mathbb{E}[f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \mathbb{E}[V^\pi(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) \,|\, \mathcal{F}_n] - \nabla_w E(w_n) \\
&\overset{\text{ind.}}{=} \mathbb{E}_{S_{n+1}\sim\mu}[(f_{w_n}(S_{n+1}) - V^\pi(S_n)) \nabla_w f_{w_n}(S_{n+1})] - \nabla_w E(w_n) \\
&= 0.
\end{aligned}$$

[2] Finally, the second moments satisfy

$$\begin{aligned}
&\mathbb{E}\left[\|\varepsilon_n\|^2 \,|\, \mathcal{F}_n \right] \\
&= \mathbb{E}\left[\|2 \left(f_{w_n}(S_{n+1}) - U_{n+1}\right) \nabla_w f_{w_n}(S_{n+1}) - \nabla_w E(w_n)\|^2 \,\middle|\, \mathcal{F}_n \right] \\
&\leq \mathbb{E}\left[\left(2 \left\|U_n \nabla_w f_{w_n}(S_{n+1})\right\|_2 + \|2 f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) - E(w_n)\|\right)^2 \,\middle|\, \mathcal{F}_n \right] \\
&= \mathbb{E}\left[4U_n^2 \|\nabla_w f_{w_n}(S_{n+1})\|^2 \,\middle|\, \mathcal{F}_n \right] \\
&\quad + \mathbb{E}\left[2|U_n| \cdot \|\nabla_w f_{w_n}(S_n)\|_2 \cdot \|2 f_{w_n}(S_n) \nabla_w f_{w_n}(S_n) - E(w_n)\| \,|\, \mathcal{F}_n \right] \\
&\quad + \mathbb{E}\left[\|2 f_{w_n}(S_{n+1}) \nabla_w f_{w_n}(S_{n+1}) - E(w_n)\|^2 \,|\, \mathcal{F}_n \right] \\
&\leq C_1 \cdot \mathbb{E}[U_{n+1}^2 \,|\, \mathcal{F}_n] + C_2 \cdot \mathbb{E}[|U_{n+1}| \,|\, \mathcal{F}_n] + C_3 \\
&= C_1 \cdot \mathbb{E}[\mathbb{E}[U_{n+1}^2 \mid S_n] \mathcal{F}_n] + C_2 \cdot \mathbb{E}[\mathbb{E}[|U_{n+1}| \,|\, S_n] | \mathcal{F}_n] + C_3 \\
&\leq C_1 K_1 + C_2 K_2 + C_3
\end{aligned}$$

[2] warum sind die gradienten beschrankt? stimmt nicht

where C_1,C_2,$C_3 \in \mathbb{R}$ exist due to the finiteness of the state space and K_1,$K_2 \in \mathbb{R}$ exist due to our assumption on the variance of U_t. By the SGD theorem, it immediately follows that w_t converges to a stationary point of E. Since E is a finite sum of convex functions if we use a linear architecture, it is a convex function itself. Therefore, the only stationary point is a global minimum which proves the theorem.

□

The proof did not require much. The linear architecture was used to ensure the least-square error is an L-smooth (here quadratic) function, and that the error has a global minimum. The algorithm might still converge to a stationary point for other estimators and other function approximations. Unfortunately, the sampling will usually have a bias as the true value function is replaced by the current estimate. For linear architectures with TD(λ) estimators we refer to the famous paper of Tsitskilis and van Roy[3].

What we did in this section works equally well for approximate evaluation of the Q-function for a given policy. The minimisation goal becomes

$$E(w) = \sum_{s\in\mathcal{S},a\in\mathcal{A}} \mu(s)\pi(a\,;\,s)\big(Q^\pi(s,a) - f_w(s,a)\big)^2.$$

The generic gradient descent motivated minimisation algorithm is

$$w_{n+1} = w_n + \alpha_n \left(U_{n+1} - f_w(s_{n+1},a_{n+1})\right) \nabla_w f_w(s_{n+1},a_{n+1}), \quad w_0 \in \mathbb{R}^d,$$

where s_n is drawn from the reference measure μ on $\mathcal{S}$, a_n according to $\pi(\cdot\,;\,s_n)$, and U_n are estimates for $Q^\pi(s_n,a_n)$. In the simplest situation of independent Monte Carlo estimates and linear function approximations $f_w(s,a) = w\cdot\phi(s,a)$ the same stochastic gradient argument shows convergence of $(w_n)_{n\in\mathbb{N}}$ to the best linear approximation of Q^π.

6.2 Approximate policy improvement

Explain how policy evaluation is done approximately and what is known.

- Replace for large action space $\mathrm{argmax_a Q(s,a)}$ by an approximation such as softmax or Gumbel softmax. Explain why this is useful and prove what is known.
- Make link to policy gradient, seeing a gradient step as approximation to taking best action. Also make connection to natural gradient.

6.3 Generic bounds for policy iteration with approximations

So far we discussed both ingredients for policy iteration (policy evaluation, policy improvement) with function approximation, i.e. replacing the true targets by targets that are simpler to obtain. Since both steps involve errors it is a priori not clear how strongly the errors accumulate while iterating. In this section we give a generic result on how errors propagate.

Proposition 6.3.1. Consider a policy improvement algorithm that produces a sequence of stationary policies π_k and a corresponding sequence of approximate state-value functions $\widehat{V^{\pi_k}}$ with given accuracies

- Policy evaluation error: $||\widehat{V^{\pi_k}} - V^{\pi_k}||_\infty \leq \varepsilon$ for all $k \in \mathbb{N}$
- Policy improvement error: $||T^{\pi_{k+1}}\widehat{V^{\pi_k}} - T^*\widehat{V^{\pi_k}}||_\infty \leq \delta$ for all $k \in \mathbb{N}$

[3] J. Tsitsiklis, B. van Roy: "An Analysis of Temporal-Difference Learning with Function Approximation", 1997, IEEE Transactions on Automatic Control, Vol 42(5)

It then holds that

$$\limsup_{k\to\infty} ||V^* - V^{\pi_k}||_\infty \leq \frac{\delta + 2\gamma\varepsilon}{(1-\gamma)^2}.$$

Before we turn to the proof, let us discuss what this statement really means. In approximative policy iteration there are two error sources:

- errors obtained in policy evaluation (estimation errors and approximation errors),
- errors obtained in finding the greedy policy corresponding to the estimated value functions $\widehat{V^{\pi_k}}$.

While the first error is clear from the statement, the second error is more hidden. It becomes clearer recalling an important fact from dynamic programming.

A policy π is greedy with respect to some function $V \in \mathbb{R}^n$ if and only if $T^*V = T^\pi V$.

Thus, δ measures, in some sense, the deviation of policies π_{k+1} from the greedy policy obtained from $\widehat{V^{\pi_k}}$. But why do we not just estimate the policies in some sense as measures, for instance in total deviation? Using the definitions the error measure used in the proposition turns out to be the deviation from the best policy (the greedy one) weighted by the future impact of actions when playing the action:

$$\begin{aligned}
&(T^*\widehat{V^{\pi_k}})(s) - (T^{\pi_{k+1}}\widehat{V^{\pi_k}})(s) \\
&= \max_{a\in\mathcal{A}_s}\Big\{r(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s'\,;\,s,a)\widehat{V^{\pi_k}}(s')\Big\} - \sum_{a\in\mathcal{A}_s}\pi_{k+1}(a\,;\,s)\Big(r(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s';\,s,a)\widehat{V^{\pi_k}}(s')\Big) \\
&= \sum_{a\in\mathcal{A}_s} \underbrace{\big(\text{greedy}(\widehat{V^{\pi_k}})(a\,;\,s) - \pi_{k+1}(a\,;\,s)\big)}_{\text{deviation from best policy}} \underbrace{\Big(r(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s';\,s,a)\widehat{V^{\pi_k}}(s')\Big)}_{\text{currently estimated Q-value}}
\end{aligned}$$

The proposition is very general and sheds some light to different situations.

(i) If all errors are zero, an explicit policy iteration can be performed, then the proposition yields another proof of convergence.

(ii) Suppose the model is known, everything is computable but policy evaluation is performed with function approximation. Then the Q-functions can be computed and the greedy policies obtained without error. In that case $\delta = 0$ and ε measures the approximation error of the value function using the approximation class. The proposition shows how badly the errors can at most accumulate.

What might be possible reasons why $\delta \neq 0$? One way this can happen is if there is no model of the system available. Without knowledge of the transition probabilities, we cannot compute Q^{π_k} and consequently cannot compute the greedy policy of V^{π_k} either. Another reason could be that we do indeed have a model available, but we do not want to use a strictly greedy policy in the next step to ensure sufficient exploration (e.g. an ε-greedy method).

As a side note, by letting $\varepsilon, \delta = 0$ we recover the convergence result for the exact policy iteration procedure.

The above proposition does not only apply in the case of function approximation but in the tabular setting as well, where it might still happen that $\varepsilon \neq 0$ if we use an approximate method for the policy evaluation like Monte-Carlo. Is the bound provided by the proposition any good? Clearly, the term $(1-\gamma)^2$ in the denominator is annoying if γ is close to 1, but the good news is that the bound only depends linearly on ε and δ.

Two properties of Bellmann's expectation operator T^π (and T^*) will be used:

(i) *Monotonicity*: $V(s) \geq U(s)$ for all $s \in \mathcal{S}$ implies $T^\pi V(s) \geq T^\pi U(s)$ for all $s \in \mathcal{S}$.

(ii) Let $c \in \mathbb{R}$ and let $\mathbf{1} \in \mathbb{R}^{|\mathcal{S}|}$ the vector with 1 in every component. Then,

$$T^\pi(V + c\mathbf{1})(s) = T^\pi V(s) + \gamma c, \quad \forall s \in \mathcal{S}.$$

Both properties follow immediately by the definition or by writing $T^\pi V(s) = \mathbb{E}_s^\pi[R(s, A_0) + \gamma V(S_1)]$. The proposition is proved in two steps. A first lemma deals with a one-step version of the proposition, which is then extended to n steps. Taking limits will then prove the proposition.

Lemma 6.3.2. (generalised policy improvement lemma)
Let π and $\bar{\pi}$ be some policies and let $V \in \mathbb{R}^{|\mathcal{S}|}$ be a vector. If

- $\|V - V^\pi\|_\infty \leq \varepsilon$ for some $\varepsilon > 0$,
- $||T^*V - T^{\overline{\pi}}V||_\infty \leq \delta$ for some $\delta > 0$,

then

$$\max_{s \in \mathcal{S}} \left(V^\pi(s) - V^{\overline{\pi}}(s)\right) \leq \frac{\delta + 2\gamma\varepsilon}{1 - \gamma}.$$

Why do we call the lemma generalised policy improvement? If $V = V^\pi$ and $\pi = \bar{\pi}$ then the choices $\varepsilon = \delta = 0$ yield the policy improvement lemma for the greedy policy obtained from V^π because T^*V^π is nothing but the value function of the greedy policy obtained from V^π. The generalisation tells us that value weakening can be controled in the approximate policz iteration step. We will later apply the lemma to $\pi = \pi_k$, $\bar{\pi} = \pi_{k+1}$ and $V = \widehat{V^{\pi_k}}$ for fixed $k \in \mathbb{N}$. The assumptions of the proposition imply the assumption of the lemma for this particular choice.

Proof. Define $\xi = \max_{s \in \mathcal{S}} \left(V^\pi(s) - V^{\overline{\pi}}(s)\right)$ so that

$$V^{\overline{\pi}}(s) + \xi \geq V^\pi(s), \quad \forall s \in \mathcal{S}.$$

Using the assumption we obtain

$$0 \leq -\left(T^*V\right)(s) + \left(T^{\overline{\pi}}V\right)(s) + \delta \leq -\left(T^\pi V\right)(s) + \left(T^{\overline{\pi}}V\right)(s) + \delta$$

for all s. We have used our assumption on $\overline{\pi}$ in the first inequality and the general inequality $T^*V(s) \geq T^\pi V(s)$ in the second one. We can now derive the main inequality that we need,

$$\begin{aligned}
V^\pi(s) - V^{\overline{\pi}}(s) &= V^\pi(s) - T^{\overline{\pi}}V^{\overline{\pi}}(s) \\
&\leq \left(V^\pi(s) - T^{\overline{\pi}}V^\pi(s) + \gamma\xi\right) + \left(-\left(T^\pi V\right)(s) + \left(T^{\overline{\pi}}V\right)(s) + \delta\right) \\
&\leq ||V^\pi - T^\pi V||_\infty + ||T^{\overline{\pi}}V - T^{\overline{\pi}}V^\pi||_\infty + \gamma\xi + \delta \\
&= ||T^\pi V^\pi - T^\pi V||_\infty + ||T^{\overline{\pi}}V - T^{\overline{\pi}}V^\pi||_\infty + \gamma\xi + \delta \\
&\leq 2\gamma||V^\pi - V||_\infty + \gamma\xi + \delta \\
&\leq 2\gamma\varepsilon + \gamma\xi + \delta,
\end{aligned}$$

where we have used additionally the fixedpoint and contraction properties of the Bellman expectation operators. The definition of ξ implies that $\xi \leq 2\gamma\varepsilon + \gamma\xi + \delta$ which is equivalent to

$$\xi \leq \frac{\delta + 2\gamma\varepsilon}{1 - \gamma}$$

which proves the statement. □

Next, define

$$\xi_k = ||V^{\pi_k} - V^{\pi_{k+1}}||_\infty \quad \text{and} \quad \zeta_k = \max_{s \in \mathcal{S}} \left(V^*(s) - V^{\pi_k}(s)\right).$$

We will now derive a bound for ζ_k which involves in the upper bound the term estimated in the previous lemma for the right choices of $\pi, \bar{\pi}$.

Lemma 6.3.3. Suppose the assumptions from Proposition 6.3.1 hold, then

$$\zeta_{k+1} \leq \gamma\zeta_k + \gamma\xi_k + \delta + 2\gamma\varepsilon$$

for all $k \in \mathbb{N}$.

Proof. First note that

$$V^*(s) - \zeta_k \leq V^{\pi_k}(s), \quad \forall k \in \mathbb{N}.$$

Monotonicity and linearity of T^* as well as the fixed point property yield

$$T^*V^{\pi_k}(s) \geq T^*(V^* - \zeta_k\mathbf{1})(s) = T^*V^*(s) - \gamma\zeta_k = V^*(s) - \gamma\zeta_k, \quad \forall s \in \mathcal{S}.$$

Using the assumption and monotonicity of T^{π_k} this yields

$$\begin{aligned}
T^{\pi_{k+1}}V^{\pi_k}(s) &\geq T^{\pi_{k+1}}(\widehat{V^{\pi_k}} - \varepsilon\mathbf{1})(s) \\
&= T^{\pi_{k+1}}\widehat{V^{\pi_k}}(s) - \gamma\varepsilon \\
&\geq T^*\widehat{V^{\pi_k}}(s) - \delta - \gamma\varepsilon \\
&\geq T^*(V^{\pi_k} - \varepsilon\mathbf{1})(s) - \delta - \gamma\varepsilon \\
&= T^*V^{\pi_k}(s) - \delta - 2\gamma\varepsilon \\
&\geq T^*(V^* - \zeta_k\mathbf{1})(s) - \delta - 2\gamma\varepsilon \\
&= V^*(s) - \gamma\zeta_k - \delta - 2\gamma\varepsilon
\end{aligned}$$

for all $s \in \mathcal{S}$, where we have used our assumptions on the approximation error of $\widehat{V^{\pi_k}}$ and the computational error of the corresponding greedy policy. It follows that

$$\begin{aligned}
V^{\pi_{k+1}}(s) &= T^{\pi_{k+1}}V^{\pi_{k+1}}(s) \\
&\geq T^{\pi_{k+1}}(V^{\pi_k} - \xi_k\mathbf{1})(s) \\
&= T^{\pi_{k+1}}V^{\pi_k}(s) - \gamma\xi_k \\
&\geq V^*(s) - \gamma\zeta_k - \delta - 2\gamma\varepsilon - \gamma\xi_k
\end{aligned}$$

for all $s \in \mathcal{S}$. Rearranging this yields

$$0 \leq V^*(s) - V^{\pi_{k+1}}(s) \leq \gamma\zeta_k + \gamma\xi_k + \delta + 2\gamma\varepsilon$$

for all $s \in \mathcal{S}$ from which we conclude the result. □

We can now easily prove the original proposition by using a familiar trick from introductory analysis courses.

Proof of Proposition 6.3.1: The second lemma yields

$$\zeta_{k+1} - \gamma\zeta_k \leq \gamma\xi_k + \delta + 2\gamma\varepsilon$$

which, combined with the first lemma (using $\pi = \pi_k$, $\bar{\pi} = \pi_{k+1}$, $V = \widehat{V^{\pi_k}}$ and the assumption of the proposition to check the assumptions of the lemma), gives

$$\zeta_{k+1} - \gamma\zeta_k \leq \gamma\frac{\delta + 2\gamma\varepsilon}{1-\gamma} + \delta + 2\gamma\varepsilon.$$

Taking the limit superior on both sides yields

$$(1-\gamma)\limsup_{k\to\infty}\zeta_k \leq \gamma\frac{\delta + 2\gamma\varepsilon}{1-\gamma} + \delta + 2\gamma\varepsilon.$$

Simplifying the righthand side and dividing by $(1-\gamma)$ proves the claim. □

6.4 Q-learning with linear function approximation

Let us quickly recall what we learnt in dynamic programming and the sample based analogue. Dynamic programming lead to two kind of algorithms:

- Policy iteration algorithms based on Banach's fixed point theorem. Iterating T^π or T^* leads to iterative algorithms that approximate the value function V^π (resp. Q^π) or the optimal value-function V^* (resp. Q^*). If V^* or Q^* is known, then an optimal policy is obtained by taking the greedy policy.

- Using that T^π and T^* for Q (not for V!) can be written as expectations the stochastic approximation theorem for $||\cdot||_\infty$-contractions gives sample based algorithms to approximate V^π, Q^π, and Q^*. Since the mathematical structure is the same (stochastic approximation of fixedpoint) the algorithms looked very similar, using the respective operator.

So far we discussed policy evaluation with function approximation and gave a rigorous justification of the simplest update rule

$$w_{n+1} = w_n + \alpha_n\big(\underbrace{V^\pi(s_{n+1})}_{\approx U_{n+1}} - f_{w_n}(s_{n+1})\big)\nabla_w f_{w_n}(s_{n+1}), \quad w_0 \in \mathbb{R}^d,$$

in terms of stochastic gradient descent. The update minimises (at least for linear approximation architectures) the least-squared error. Similarly, for Q-learing with function approximation the goal is to minimise

$$E(w) = \frac{1}{2}\sum_{s\in\mathcal{S}, a\in\mathcal{A}_s} \mu(s,a)\big(Q^*(s,a) - f_w(s,a)\big)^2 = \frac{1}{2}\mathbb{E}_{(S,A)\sim\mu}\big[(Q(S,A) - f_w(S,A))^2\big]$$

for some reference measure μ. Just as before a stochastic gradient algorithm to minimise E is

$$w_{n+1} = w_n + \alpha_n\big(Q^*(s_{n+1}, a_{n+1}) - f_{w_n}(s_{n+1}, a_{n+1})\big)\nabla_w f_{w_n}(s_{n+1}, a_{n+1}), \quad w_0 \in \mathbb{R}^d.$$

As for the temporal difference approach to policy evaluation with function approximation we use the Bellman operator. Recalling that Q^* is the unique fixedpoint of $T^*Q(s,a) = \mathbb{E}_s^a\big[R(s,a) + \gamma \max_{a'\in\mathcal{A}_{S_1}} Q(S_1, a')\big]$ we replace the unknown Q^* by a sample of the expectation and the unknown Q by the best known approximation f_w.

Definition 6.4.1. The update scheme

$$\begin{aligned} &w_{n+1} \\ &= w_n + \alpha_n\big(R(s_{n+1}, a_{n+1}) + \max_a f_{w_n}(s_{n+1}, a) - f_{w_n}(s_{n+1}, a_{n+1})\big)\nabla_w f_{w_n}(s_{n+1}, a_{n+1}) \end{aligned}$$

is called Q-learning with function approximation. If $\{f_w = w^T\Phi : w \in \mathbb{R}^d\}$ one speaks of Q-learning with linear function approximation, if $\{f_w : w \in \mathbb{R}^d\}$ are neural network functions one speaks of deep Q-learning (DQN). The choice of state-action pairs (s_n, a_n) is handled differently in different versions of the algorithm.

6.5 Policy gradient with linear function approximation

Similarly to stochastic approximation

$$w_{n+1} = w_n + \alpha_n\Big(R(s_n, a_n) + \gamma \max_{a'} f_w(s', a') - f_w(s_n, a_n)\Big)\nabla_w f_w(s_n, a_n),$$

6.6 A quick dive into neural networks

6.6.1 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 outcoming action signals. Inspired from neuroscience a century of research in computer science has led to incredible results in artificial intelligence. Let us first describe the simplest model of a neuron, and then continue with artificial networks of neurons (so-called neural networks or perceptrons). The simplest model[4] is

$$f(x) = \mathbf{1}_{[0,\infty)}\Big(\sum_{i=1}^{d} w_i x_i + b\Big).$$

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 6.6.1. For $\sigma : \mathbb{R} \to \mathbb{R}$, $w \in \mathbb{R}^d$, and $b \in \mathbb{R}$, the function $f : \mathbb{R}^d \to \mathbb{R}$ defined by

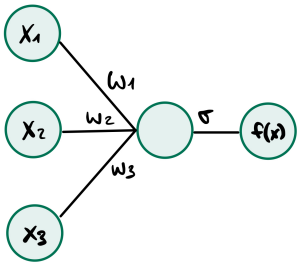
$$f(x) = \sigma\Big(\sum_{i=1}^{d} w_i x_i + b\Big)$$

is called an artificial neuron with activation function σ, weights $w_1, ..., w_d$ for the signals $x_1, ..., x_d$ and bias b.

The weights can be interpreted as importance of the signal for the neuron, the bias as susceptability of the neuron.

Example 6.6.2. Here are some typcial activation functions:

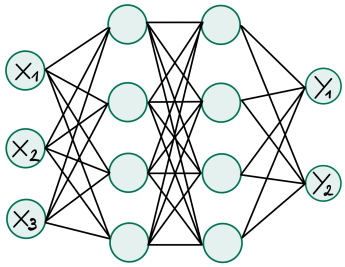
- $\sigma(s) = \mathbf{1}_{[0,\infty)}(s)$, the Heavyside function, yields a neuron that either fires or not,
- $\sigma(s) = s$ yields a linear neuron which is not really used,
- $\sigma(s) = \max\{0, s\}$ is called rectified linear unit (ReLU), a very typical practical choice,
- $\sigma(s) = \frac{1}{1+e^{-s}} = \frac{e^s}{1+e^s}$ 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

[4] McCulloch, Pitts: "A logical calculus of ideas immanent in nervous activity", Bull. Math. Biophys., 5:115-133, 1943

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.



Graphical representation of a neural network with 2 hidden layers, 3 input dimensions, 2 output dimensions

Here is a mathematical formulisation of a simple network of neurons using the same activation function, such networks of artificial neurons are also called multilayer perceptrons[5]:

Definition 6.6.3. (Feedforward neural networks)
Let $d, L \in \mathbb{N}$, $L \geq 2$, and $\sigma : \mathbb{R} \to \mathbb{R}$. Then a multilayer perceptron (or fully connected feedforward neural network) with d-dimensional input, L layers, and activation function σ is a function $F : \mathbb{R}^d \to \mathbb{R}^k$ that can be written as

$$F(x) = T_L(\sigma(T_{L-1}(...\,\sigma(T_1(x))))),$$

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 applies a final normalisation with the softmax function $\Phi(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}}$ to the output vector:

$$F(x) = \Phi(T_L(\sigma(T_{L-1}(...\,\sigma(T_1(x)))))).$$

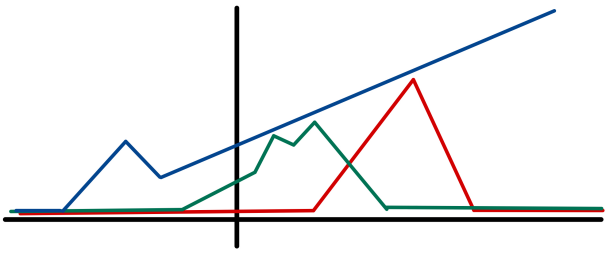
[6] The multilayer perceptron is quite special in it's structure. All neurons of a fixed layer receive the same input vector (the outputs of the neurons before) which is then multiplied with the weight-vector of the neuron, shifted, and plugged-into the activation function to yield the signal. To visualise 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 6.6.4. The architecture of a feedforwards neural network is the chosen number of layers, units, and fixed zero-weights.

[5] F. Rosenblatt: "The perceptron: a probabilistic model for information storage and organization in the brain" Psychological review, 65(6):386, 1958

[6] include drawings

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 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 $\mathrm{ReLU}(-1) - 2\mathrm{ReLU}(0) + \mathrm{ReLU}(1)$ gives a spike at 0, with $\mathrm{ReLU}(b) = \max\{0, b\}$.

There are other classes of artificial neural networks that are much more relavant 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.

6.6.2 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 results[7].

Theorem 6.6.5. (Approximation of Boolean functions)
Every Boolean function $f : \{0,1\}^d \to \{0,1\}$ 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 $u \in \{0,1\}^d$,

$$\mathbf{1}_{x=u} = \mathbf{1}_{[0,\infty)}\Big(\sum_{i=1}^{d}(2x_iu_i - x_i - u_i)\Big) = \sigma\Big(\sum_{i=1}^{d}(2x_iu_i - x_i - u_i)\Big).$$

[7] Check 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

$$f(x) = \sigma\Big(-1 + \sum_{u\in A} \mathbf{1}_{x=u}\Big) = \sigma\Big(-1 + \sum_{u\in A} \sigma\Big(\sum_{i=1}^{d}(2x_iu_i - x_i - u_i)\Big)\Big).$$

This, in fact, this is nothing but a neural network with two hidden layers. The first with $|A|$ units, the second with one unit. □

Definition 6.6.6. 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 $g : K \to \mathbb{R}$ 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 $\lim_{x\to+\infty} \sigma(x) = 1$ and $\lim_{x\to-\infty} f(x) = 0$. 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 6.6.7. (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 $||f - h||_\infty \le C\,\omega(h, 1/p)$ with C independent of h.

In the estimate the so-called modulus of continuity appears:

$$\omega(h, \delta) = \sup_{x,y\in\mathbb{R}:|x-y|\le\delta} |h(x) - h(y)|$$

For a function $h : \mathbb{R} \to \mathbb{R}$ 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 $\omega(h, \delta) \le L\delta$, 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

$$\sup_{x\in[0,1]} |h(x) - h_p(x)| = \sup_{x\in[0,1]} \Big|h(x) - \sum_{i=1}^{p} h(x_i)\mathbf{1}_{[x_{i-1},x_i)}(x)\Big| \le \omega(h, 1/p)$$

by the definition of h_p and ω. With a telescopic sum we rewrite

$$h_p(x) = h(x_1) + \sum_{i=1}^{\lfloor px\rfloor} \big(h(x_{i+1}) - h(x_i)\big).$$

Next, we define a the shallow neural network function with p units as

$$f_p(x) = h(x_1)\sigma(\alpha) + \sum_{i=1}^{p-1} \big(h(x_{i+1}) - h(x_i)\big)\sigma\big(\alpha(px - i)\big) \in \mathcal{F}_p$$

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

$$\big|\sigma\big(\alpha(px - i)\big) - \mathbf{1}_{i \leq \lfloor px \rfloor}\big| \leq \frac{\varepsilon}{p}$$

holds for all $i \notin \{\lfloor px \rfloor, \lfloor px \rfloor + 1\}$ because then $|\alpha(px - i)| > \alpha$. It then follows that

$$\begin{aligned}
&\big|f_p(x) - h_p(x)\big| \\
&= \Big|h(x_1)(\sigma(\alpha) - 1) + \sum_{i=1}^{p-1} \big(h(x_{i+1}) - h(x_i)\big)\big(\sigma(\alpha(px - i)) - \mathbf{1}_{i \leq \lfloor px \rfloor}\big)\Big| \\
&\leq \frac{\varepsilon}{p}\big(|h(x_1)| + (p-2)\omega(h, 1/p)\big) + |h(x_{\lfloor px \rfloor + 1}) - h(x_{\lfloor px \rfloor})||\sigma(\alpha(px - \lfloor px \rfloor)) - 1| \\
&\quad + |h(x_{\lfloor px \rfloor + 2}) - h(x_{\lfloor px \rfloor + 1})||\sigma(\alpha(px - \lfloor px \rfloor - 1)) - 1|.
\end{aligned}$$

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

$$\begin{aligned}
\sup_{x \in [0,1]} |h(x) - f_p(x)| &\overset{\Delta}{\leq} \sup_{x \in [0,1]} |h(x) - h_p(x)| + \sup_{x \in [0,1]} |h_p(x) - f_p(x)| \\
&\leq \omega(h, 1/p) + \delta + 2(||\sigma||_\infty + 1)\omega(h, 1/p).
\end{aligned}$$

Now the claim follows. Covering a compact set $K \subseteq \mathbb{R}$ by finitely many intervals it then follows that for all $h : K \to \mathbb{R}$ 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:

CD CD

The set

$$\mathcal{E} := \Big\{g : K \to \mathbb{R} \,\Big|\, g(x) = \sum_{i=1}^{N} s_i e^{\langle v^i, x \rangle}, x \in \mathbb{R}^d, N \in \mathbb{N}, v^i \in \mathbb{R}^d, s_i \in \{-1, 1\}\Big\}$$

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:

CD CD

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 $f : \mathbb{R}^d \to \mathbb{R}$ 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 $\bar{K}_i \subseteq R$ 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 $\bar{f}_i:\mathbb{R}^d\to\mathbb{R}$ as

$$\bar{f}_i(x)=s_i f_i\Big(\sum_{k=1}^{d}v_k^i x_k\Big)$$

so that

$$\begin{aligned}\sup_{x\in K}\big|\bar{f}_i(x)-s_ie^{\langle v^i,x\rangle}\big|&=\sup_{x\in K}\big|s_if_i(\langle v^i,x\rangle)-s_ie^{\langle v^i,x\rangle}\big|\\&=\sup_{x\in\bar{K}_i}|s_if_i(x)-s_ie^x|<\frac{\varepsilon}{N}.\end{aligned}$$

To see that $\bar{f}_i$ is a neural network function with one hidden layer note that

$$\bar{f}_i(x)=\sum_{l=1}^{p}s_iw_l\sigma\Big(\sum_{k=1}^{d}v_k^ix_k-b_l\Big).$$

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 $f=\sum_{i=1}^{N}\bar{f}_i$:

$$\sup_{x\in K}|f(x)-g(x)|\le\sum_{i=1}^{N}\sup_{x\in K}\big|\bar{f}_i(x)-s_ie^{\langle v^i,x\rangle}\big|<\varepsilon.$$

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 $f:\mathbb{R}^d\to\mathbb{R}$ with $\sup_{x\in K}|f(x)-h(x)|<\frac{\varepsilon}{2}$. Then $\sup_{x\in K}|h(x)-f(x)|\le\sup_{x\in K}|h(x)-g(x)|+\sup_{x\in K}|g(x)-f(x)|<\varepsilon$. This proves the theorem. □

6.6.3 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: $\{(x_i,y_i):i\le N\}$. 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 featue 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 imageNet[8] with more than fourteen million labeled images. There are two amazing features why neural networks are used to map images to feature vectors:

[8] imageNet 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 dozents 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 6.6.8. 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.

[9] In the algorithm and the proof we allowed a generic function Φ to normalise the outputs so the chain of concatenations is

$$F(x) = T_L(\sigma(T_{L-1}(...\,\sigma(T_1(x))))),$$

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 **??**[10] for a graphical representation to ease the understanding. [11] 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 chooses 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

$$\begin{aligned}
\frac{\partial}{\partial w_{i,j}^L}\frac{1}{2}||F_w(x) - y||^2 &= \frac{\partial}{\partial w_{i,j}^L}\sum_{s=1}^{N_L}\frac{1}{2}\Big(\Phi\Big(\sum_{t=1}^{N_{L-1}} w_{t,s}^L V_t^{L-1}\Big) - y_s\Big)^2 \\
&= \frac{\partial}{\partial w_{i,j}^L}\frac{1}{2}\Big(\Phi\Big(\sum_{t=1}^{N_{L-1}} w_{t,j}^L V_t^{l-1}\Big) - y_j\Big)^2 \\
&= \Big(\Phi\Big(\sum_{t=1}^{N_{l-1}} w_{t,j}^L V_t^{L-1}\Big) - y_j\Big)\Phi'(h_j^L)\frac{\partial}{\partial w_{i,j}^L}\sum_{t=1}^{N_{L-1}} w_{t,j}^L V_t^{L-1} \\
&= (V_j^L - y_j)\Phi'(h_j^L)V_i^{L-1} \\
&=: \delta_j^L\, V_i^{L-1}.
\end{aligned}$$

[9] Φ muss weg weil es nicht koordinatenweise wirkt, dazu passen die indices nicht
[10] bildchen noch malen
[11] put drawing

There are two crucial points involved in the second and fourth equality. For the second equality we use that only the jth term depends on $w^L_{i,j}$ (compare the graphical representation of the network) so that all other derivatives equal zero. Similarly, for the fourth equality only the term $t = i$ depends on $w^L_{i,j}$ so that all other derivatives vanish and the tth term simply has the derivative V^{L-1}_i.

Last hidden layer: To compute the partial derivative with respect to the final weights $w^{L-1}_{i,j}$ we proceed similarly. Writing the norm and the definition of the neural network yields

$$\begin{aligned}
&\frac{\partial}{\partial w^{L-1}_{i,j}} \frac{1}{2} ||F_w(x) - y||^2 \\
&= \frac{\partial}{\partial w^{L-1}_{i,j}} \sum_{s=1}^{N_L} \frac{1}{2} \Big(\Phi\Big(\sum_{t=1}^{N_{L-1}} w^L_{t,s} \sigma\Big(\sum_{r=1}^{N_{l-2}} w^{L-1}_{r,t} V^{L-2}_r \Big)\Big) - y_s \Big)^2 \\
&= \sum_{s=1}^{N_L} (V^L_s - y_s) \Phi'(h^L_s) \frac{\partial}{\partial w^{L-1}_{i,j}} \sum_{t=1}^{N_{L-1}} w^L_{t,s} \sigma\Big(\sum_{r=1}^{N_{l-2}} w^{L-1}_{r,t} V^{L-2}_r \Big) \\
&= \sum_{s=1}^{N_L} (V^L_s - y_s) \Phi'(h^L_s) \sum_{t=1}^{N_{L-1}} w^L_{t,s} \sigma'\Big(\sum_{r=1}^{N_{l-2}} w^{L-1}_{r,t} V^{L-2}_r \Big) \frac{\partial}{\partial w^{L-1}_{i,j}} \sum_{r=1}^{N_{l-2}} w^{L-1}_{r,t} V^{L-2}_r \\
&= \sum_{s=1}^{N_L} (V^L_s - y_s) \Phi'(h^L_s) \sum_{t=1}^{N_{L-1}} w^L_{t,s} \sigma'(h^{L-2}_j) \frac{\partial}{\partial w^{L-1}_{i,j}} \sum_{r=1}^{N_{l-2}} w^{L-1}_{r,t} V^{L-2}_r .
\end{aligned}$$

All derivatives of the last term disappear, except for $r = i$ and $t = j$. The remaining derivative equals 1, thus, the expression simplifies to

$$\begin{aligned}
\frac{\partial}{\partial w^{L-1}_{i,j}} \frac{1}{2} ||F_w(x) - y||^2 &= V^{L-2}_i \, \sigma'(h^{L-2}_j) \sum_{s=1}^{N_L} (V^L_s - y_s) \Phi'(h^L_s) w^L_{j,s} \\
&= V^{L-2}_i \, \sigma'(h^{L-2}_j) \sum_{s=1}^{N_L} \delta^L_j w^L_{j,s} \\
&=: V^{L-2}_i \, \delta^{L-1}_j .
\end{aligned}$$

Well, anyone who likes playing with indices is warmly invited to turn this into a clean induction :).[12] □

Since the derivatives of σ repeatadly appear it is clearly desirable to chose 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 $\delta^L_j = \Phi'(h^L_j)$ and $\delta^l_i = \sigma'(h^{l-1}_i) \sum_{j=1}^{N_l} w^l_{i,j} \delta^l_j$.

6.6.4 Using neural networks to approximate value functions

6.6.5 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.

[12] write 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

- $h_i^l := \sum_{k=1}^{N_{l-1}} w_{k,l}^l V_k^{l-1}$, $i = 1, ..., N_l$,
- $V_i^l := \sigma(h_i^l)$, $i = 1, ..., N_l$,

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

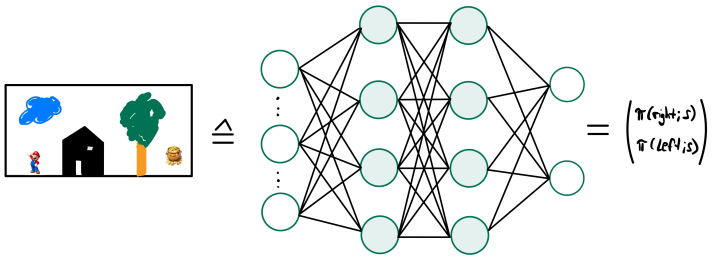
$$\pi^\theta(a\,;s) = \frac{e^{\theta_a^T \cdot \Phi(s)}}{\sum_{a'} e^{\theta_{a'}^T \Phi(s)}}, \quad \theta = (\theta_{a_1}, ..., \theta_{a_k}) \in \mathbb{R}^{ad},$$

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 visulisation of the idea. An image (which is nothing but a long pixel vector) is fed into a neural network, the output vector are the probabilities $\pi(a_1\,;\,s), ..., \pi(a_k\,;\,s)$. In that case a softmax function transforms the final hidden layer into probabilities.



Direct policy parametrisation using neural networks

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 $\pi(a\,;\,s)$.

6.7 Deep Q-learning (DQN)

6.8 Deep policy gradient (actor critic methods)

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 combinded with further idea it does. In this section we will discuss some of the most important ideas for practical use of REINFORCE.

6.8.1 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.
- Valued-based algorithms that evaluate the value-function (or Q-function). Such methods are indirect, they do not try to optimise 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 in 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 [13],[14]:

- 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 as 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. [15]

Proposition 6.8.1. If $\mathbf{Q}^{\pi^\theta}$ 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

$$\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}_s} d^{\pi^\theta}(s)\pi^\theta(a\,;\,s)\big(Q^{\pi^\theta}(s,a) - \mathbf{Q}_w^{\pi^\theta}(s,a)\big)\nabla_w \mathbf{Q}_w^{\pi^\theta}(s,a) = 0,$$

then

$$\nabla J_s(\theta) = \frac{1}{1-\gamma} \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;\,s)\nabla \log(\pi^\theta(a;s'))\mathbf{Q}_w^{\pi^\theta}(s',a).$$

[16] The exact same formulas also hold with the other policy gradient representations from Section 5.1.2.

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

$$\sum_{s'\in\mathcal{S}} d^{\pi^\theta}(s') \sum_{a\in\mathcal{A}_{s'}} \nabla_\theta \pi^\theta(a\,;\,s)\big(Q^{\pi^\theta}(s',a) - \mathbf{Q}_w^{\pi^\theta}(s',a)\big) = 0$$

[13] R. Sutton, D. McAllester, S. Singh, Y. Mansour: "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NIPS 1999

[14] V. Konda, J. Tsitsiklis: "Actor-Critic Algorithms", NIPS 1999

[15] ueberall μ in gradient einarbeiten, auch fuer d^π

[16] Bedeutung beider Eigenschaften ausfuehren

Using the policy gradient theorem (version of Theorem 5.1.6) and the above yields the claim:

$$\begin{aligned}\nabla J_s(\theta) &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;\,s)\nabla\log(\pi^\theta(a;s'))Q^{\pi^\theta}(s',a)\\ &\quad - \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;\,s)\nabla_\theta\log(\pi^\theta(s,a))\big(Q^{\pi^\theta}(s',a)-\mathbf{Q}_w^{\pi^\theta}(s',a)\big)\\ &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;s\,)\nabla\log(\pi^\theta(a;s'))\mathbf{Q}_w^{\pi^\theta}(s',a)\end{aligned}$$

□

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 6.8.2. 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

$$\mathbf{Q}_w^{\pi^\theta}(s,a) = w^T\cdot\Big(\Phi(s,a) - \sum_{a'}\pi^\theta(a'\,;\,s)\Phi(s,a')\Big).$$

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 6.8.3. A similar example can be obtained for a continuous action space. Suppose π^θ is a linear Gaussian policy, i.e.

$$\pi^\theta(a\,;s) \sim \mathcal{N}\big(\theta^T\cdot\Phi(s),\sigma^2\big)$$

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 as the connection of critic computation and actor update is nothing else then a regular update of the policy gradient algorithm (Proposition 6.8.1). 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,

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 critcial point w of the weighted approximation error, i.e.

$$\nabla_w \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}_s} d^{\pi^{\theta_n}}(s)\pi^{\theta_n}(a\,;\,s)\big(Q^{\pi^{\theta_n}}(s,a) - \mathbf{Q}_w^{\pi^\theta}(s,a)\big)^2 = 0.$$

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

- the gradient will be approximated as in the REINFORCE algorithm replacing the Q-function by the estimated Q-function $\mathbf{Q}_w^{\pi^\theta}$.

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 then 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 6.8.4. 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

$$\begin{aligned}\nabla J_s(\theta) &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;\,s)\nabla\log(\pi^\theta(a;s'))\big(Q^{\pi^\theta}(s',a) - V^{\pi^\theta}(s')\big)\\ &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}_{s'}} d^{\pi^\theta}(s')\pi^\theta(a\,;\,s)\nabla\log(\pi^\theta(a;s'))A^{\pi^\theta}(s',a).\end{aligned}$$

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 $\hat{A}^{\pi^\theta}$ of the unknown A^{π^θ} is derived.
- Plug $\hat{A}^{\pi^\theta}$ 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 $V_w : \mathcal{S} \to \mathbb{R}$ 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 $y_1, ..., y_n$ 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 $\hat{A}^{\pi^\theta}$ of A^{π^θ} from the gradient step is computed.

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 $(s_0^i, a_0^i, ...)$.
  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 $\hat{A}(s_0^i, a_0^i) = R(s_0^i, s_0^i) + \hat{V}_w^\pi(s_1^i) - \hat{V}_w^\theta(s_0^i)$.
  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

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. [17]

6.8.2 Soft actor-critic (SAC)

6.8.3 Proximal policy optimisation (PPO)

[17] bespreche sampling aus d^π durch stoppen nach geometrischer zeit. und bedeutung von $d^\pi \pi$

Chapter 7

Monte Carlo Tree Search