

---

# The Mathematics of Reinforcement Learning

LEIF DÖRING

UNIVERSITY OF MANNHEIM

# Contents

<b>I</b>	<b>Multiarmed Bandits</b>	<b>3</b>
<b>1</b>	<b>Stochastic bandits</b>	<b>4</b>
1.1	A quick dive into two-stage stochastic experiments . . . . .	4
1.2	Introduction to stochastic bandits . . . . .	5
1.3	Algorithms: the exploration-exploitation trade-off . . . . .	13
1.3.1	Basic committ-then-exploit algorithm . . . . .	14
1.3.2	From greedy to UCB . . . . .	18
1.3.3	Boltzmann exploration . . . . .	28
1.3.4	Simple policy gradient for stochastic bandits . . . . .	30
1.4	Lower bounds for stochastic bandits . . . . .	34
1.4.1	A bit on relative entropy . . . . .	34
1.4.2	Mini-Max lower bounds (model-dependent) . . . . .	37
1.4.3	Asymptotic lower bound (model-dependent) . . . . .	39
<b>2</b>	<b>More bandits</b>	<b>42</b>
2.1	Adversarial bandits . . . . .	42
2.2	Contextual bandits . . . . .	42
<b>II</b>	<b>Tabular Reinforcement Learning</b>	<b>43</b>
<b>3</b>	<b>Basics: Stochastic Control and Dynamic Programming Methods</b>	<b>44</b>
3.1	Markov decision problems . . . . .	44
3.1.1	A quick dive into Markov chains . . . . .	44
3.1.2	Markov decision processes . . . . .	45
3.1.3	Stochastic control theory . . . . .	57
3.2	Basic tabular value iteration algorithm . . . . .	67
3.3	Basic policy iteration (actor-critic) algorithm . . . . .	70
3.3.1	Policy evaluation . . . . .	71
3.3.2	Policy improvement . . . . .	73
3.3.3	Policy iteration algorithms (tabular actor-critic) . . . . .	76
3.4	Stochastic control in finite time . . . . .	80
3.4.1	Setting and dynamic programming . . . . .	80
3.4.2	Dynamic programming algorithms . . . . .	85
<b>4</b>	<b>Simulation based dynamic programming methods</b>	<b>87</b>
4.1	A guiding example . . . . .	88
4.2	Monte Carlo policy evaluation and control . . . . .	89
4.2.1	First visit Monte Carlo policy evaluation . . . . .	90
4.2.2	Generalised policy iteration with first visit Monte Carlo estimation . . . . .	92
4.3	Asynchronous stochastic fixed point iterations . . . . .	93
4.3.1	Basic stochastic approximation theorem . . . . .	93
4.3.2	Asynchronous stochastic approximation . . . . .	96
4.4	One-step simulation-based dynamic programming TD(0) . . . . .	106

4.4.1	One-step policy evaluation (simulation-based policy evaluation) . . . . .	106
4.4.2	$Q$ -learning and SARSA (simulation-based value iteration for $Q$ ) . . . . .	110
4.4.3	Double $Q$ -learning . . . . .	115
4.5	Multi-step approximate dynamic programming . . . . .	120
4.5.1	$n$ -step TD for policy evaluation and control . . . . .	120
4.5.2	TD( $\lambda$ ) algorithms . . . . .	123
4.6	Tabular simulation based actor-critic . . . . .	132

# Introduction

The relevance of methods in artificial intelligence, or at least the belief in their relevance, has experienced significant fluctuations since its early days. The current wave appears distinct from its predecessors, with clear and easily accessible benefits for everyone. Present breakthroughs are predominantly rooted in advancements of deep learning, specifically the understanding of neural network functions, addressing highly complex regression problems. This progress is further fueled by the availability of massive datasets and affordable computational resources. While deep learning dominates, other artificial intelligence methods are poised to yield fundamental breakthroughs in the near future. These lecture notes focus on the area known as reinforcement learning, a method designed to address optimal decision problems. In this framework, an agent (or multiple agents) aims to find decisions that optimize a given target. Applications range from controlling robots for optimal task performance to making strategic decisions in games or assisting production planners in steering production optimally. Similar problems have been studied for decades across various research communities, each approaching them with different tools. Control theory in mathematics has traditionally dealt with physical control problems, such as steering vehicles or power plants optimally. Neuroscience seeks to understand how humans learn to act optimally in task performance. Operations research naturally addresses optimal decision-making in diverse areas, from investments to production planning. In computer science, the focus has been on teaching computers to play games at a superhuman level. Greater progress could have been achieved if these diverse communities had collaborated more closely in the past. Signs of increased collaboration are already evident in the adoption of different names for similar ideas. Notably, the term "reinforcement learning" originates from neuroscience and is relatively uncommon in mathematical literature.

The primary goal of these lecture notes is to present reinforcement learning concepts to mathematicians interested in understanding optimal decision-making through trial-and-error. Reinforcement learning formalizes the learning process in a cycle of three steps: try, observe, and improve. Despite its seemingly straightforward nature, many approximation algorithms precisely follow this structure. A secondary objective is to help computer scientists gain a deeper understanding of the mathematics behind reinforcement learning algorithms. There are a few text books that are very much recommended to be read in parallel while studying these lecture notes, most importantly

- "Reinforcement Learning" by Sutton and Barto, the classical introduction with a primary focus on concepts rather than mathematical details. Reading it in parallel with these notes is certainly very beneficial.
- "Neuro-dynamic Programming" by Bertsekas and Tsitsiklis, covering mathematical foundations also included in these notes. Despite being slightly outdated in topics it introduces powerful ideas predating their widespread adoption.
- "Bandit Algorithms" by Lattimore and Szepesvári, offering a comprehensive overview of bandit algorithms. Banit problems offer a very simple decision problem that allows an in-depth analysis and serves to illustrate difficulties for reinforcement learning in more complicated situations.

These lecture notes were written in 2023 to assist students in a one-semester course on reinforcement learning at the University of Mannheim. The content is continuously refined and expanded

to cover topics of interest to the author, his colleagues, and students. All feedback and comments are warmly welcomed.

## Part I

# Multiarmed Bandits

# Chapter 1

## Stochastic bandits

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



The chapter focuses 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 knowing 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 division 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

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

hold. The kernel  $\kappa$  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 defined in the usual way. We also use the notation  $\mathbb{P}(X \in A | Y) = \kappa(Y, A)$  from which it follows that  $\mathbb{E}[h(X, Y) | Y] = \int h(x, Y) \mathbb{P}(X \in dx | Y)$ . 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  $\mathbb{E}[h(X, Y) | Y] = \int h(x, Y) \mathbb{P}(X \in dx)$ .

## 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 canteen. Let's say the canteen offers four choices: vegan, vegetarian, classic, oriental. These are the four possible experiments, the random outcomes are the quality of the dish (measured in some way, such as on a 1-10 scale). There are plenty of other situations that immediately come to mind such as

- medical treatment of patients with different doses, outcome measures the success, for instance 1 for healed and 0 otherwise,
- placing different advertisements on websites, outcome measures the click rates.

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





The basic mathematical model behind a stochastic bandits is as follows. There are distributions  $P_{a_1}, \dots, P_{a_K}$  (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  $a_1, \dots, a_K$  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  $\mathcal{A} = \{a_1, \dots, a_K\}$  of experiments (arms to play) and denote by  $Q_a$  the expectation of the random outcome whose distribution we denote by  $P_a$ . Of course there could be other goals than finding the arm with the highest average outcome, but this is what we decide to optimize. Now fix a time-horizon  $n$  (which could be infinite), the number of rounds we can use for learning. A learning strategy is a sequence  $(\pi_t)_{t=1, \dots, n}$  of probability distributions on  $\mathcal{A}$  that only depend on what has been played prior to time  $t$ . Here  $\pi_t(\{a\})$  is the probability that the player chooses action  $a$  at time  $t$  and then receives the random outcome of the corresponding experiment.



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

The aim is to find a learning strategy that maximises the outcome of this procedure. To have an idea in mind think of the example of the university canteen. During your studies you might have  $n = 500$  choices in total. If you are not vegan or vegetarian you probably started without preferences, i.e.  $\pi_t(\text{vegan}) = \dots = \pi_t(\text{oriental}) = \frac{1}{4}$ , and over time learnt from experience how to change the following distributions  $\pi_t$  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  $\nu$  is called a stochastic bandit model. In these lectures we will always assume  $\mathcal{A} = \{a_1, \dots, a_K\}$  is finite,  $K$  is the number of arms. Often it will be useful to denote the arms by  $1, \dots, K$  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.

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

Typically one abbreviates  $Q_*$  for the largest action value  $Q_{a^*}$  and if there are several optimal arms the  $\operatorname{argmax}$  chooses any of them.

- A learning strategy for  $n$  rounds ( $n = +\infty$  is allowed) consists of
  - an initial distribution  $\pi_1$  on  $\mathcal{A}$ ,
  - a sequence  $(\pi_t)_{t=2, \dots, n}$  of kernels on  $\Omega_{t-1} \times \mathcal{A}$ ,



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

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

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

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



We will always assume that the action values  $Q_a$  are unknown but the bandit is a generative model, i.e. the random variables can be sampled. Everything that can be learnt about the model must be achieved by simulations (playing arms). In principle the learning strategy should be written down, most of the time we will construct the kernels  $\pi_t$  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  $\nu$  is a stochastic bandit model and  $(\pi_t)_{t=1, \dots, n}$  a learning strategy for  $n$  rounds. Then a stochastic process  $(A_t^\pi, X_t^\pi)_{t=1, \dots, n}$  on a probability space  $(\Omega, \mathcal{F}, \mathbb{P})$  is called a stochastic bandit process with learning strategy  $\pi$  if  $A_1^\pi \sim \pi_1$  and

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

for all  $t = 1, \dots, n$ . We will call  $A_t^\pi$  the action (the chosen arm) at time  $t$  and  $X_t^\pi$  the outcome (or reward) when playing arm  $A_t^\pi$  at time  $t$ . For notational convenience the superscripts  $\pi$  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  $\pi$ , 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  $(A^\pi, X^\pi)$ .

*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  $\pi$  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  $\pi$ .

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

$A_1$	$A_2$	$A_3$	$A_4$	
$\uparrow \pi_1$	$\uparrow \pi_2$	$\uparrow \pi_3$	$\uparrow \pi_4$	
$U_1$	$U_2$	$U_3$	$U_4$	$\dots$
$\mathbf{X}_1^{(\mathbf{a}_1)}$	$X_2^{(a_1)}$	$X_3^{(a_1)}$	$\mathbf{X}_4^{(\mathbf{a}_1)}$	$\dots$
$X_1^{(a_2)}$	$X_2^{(a_2)}$	$\mathbf{X}_3^{(\mathbf{a}_2)}$	$X_4^{(a_2)}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$X_1^{(a_K)}$	$\mathbf{X}_2^{(\mathbf{a}_K)}$	$X_3^{(a_K)}$	$X_4^{(a_K)}$	$\dots$

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

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

- $t = 1$ : Use  $U_1$  to sample from the discrete measure  $\pi_1(\cdot)$  an arm  $a$ , denote this arm by  $A_1$  and set  $X_1 := X_1^{(A_1)}$ .
- $t \mapsto t + 1$ : Use  $U_{t+1}$  to sample from the discrete measure  $\pi(\cdot; A_1, X_1, \dots, A_t, X_t)$  (relying only on the table to the left of column  $t + 1$ ) an arm  $a$ , denote this arm by  $A_t$  and 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 choosing the reward from row  $A_t$ . To see that this process  $(A, X)$  is indeed a bandit process with learning strategy  $\pi$  we need to check the defining properties. Let us denote by  $I_t^a(a_1, \dots, x_{t-1})$  a partition of  $[0, 1]$  into  $K$  disjoint intervals with lengths  $\pi_t(a; a_1, \dots, x_{t-1})$ . Then, using  $h(u, a_1, \dots, x_{t-1}) := \mathbf{1}_{u \in I_t^a(a_1, \dots, x_{t-1})}$  and

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

yields

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

The second property is derived as follows:

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

□

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  $a$ th arm is played for the  $n$ th time then the reward variable  $X_n^{(a)}$  is used instead of the reward variable corresponding to the time at which the  $a$ th arm is played for the  $n$ th time. In formulas, one sets  $X_t = X_{T_a(t)}^{(a)}$  instead of  $X_t = X_t^{(a)}$ , where  $T_a(t) = \sum_{s \leq t} \mathbf{1}_{X_s = a}$  is the number of times the  $a$ th arm was played before time  $t$ .

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

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

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

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

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

Now that bandit models are defined the next step is to discuss the questions of interest. In fact, with the targets of this lecture course in mind this is not completely clear as the goals of stochastic bandit theory and reinforcement learning are different. The reason is that the

research communities of statistics and AI traditionally have different examples in mind. While the stochastic bandit community originates from statistical question of optimal experimental design in medicine the AI community is more focused on artificial decision making (of computers). While both aim at developing and analysing algorithms that find the optimal arm, the different goals yield in different optimization goal. As an guiding example we go back to the two examples of medical treatment and advertisement. While in medical treatment every single round of learning refers to the medical treatment of an individual (which has the highest priority and the number of rounds is clearly limited say by  $n = 200$ ) in online advertisement it might be much less problematic to play many rounds (say  $n = 100k$ ) in the training procedure and to waste possible income. Here are 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 choosing wrong arms is minimized.

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

Let us first discuss the classical stochastic bandit approach (A). We already defined an optimal arm, an arm with maximal action value  $Q_a$ . Of course there might be several best arms, then  $a_*$  is chosen as any of them. Since the index set is not ordered there is no preference in which best arm to denote  $a_*$ . The goal is to maximise over all learning strategies the expectation  $\mathbb{E}[\sum_{t=1}^n X_t] = \sum_{t=1}^n \mathbb{E}[X_t]$  for a fixed time-horizon  $n$ . There is a simple upper bound for the reward until time  $n$ , which is  $nQ_*$ . Hence, if all  $Q_a$  would be known in advance then the stochastic bandit optimization problem would be trivial, just 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  $\nu$  is a bandit model and  $(\pi_t)_{t=1, \dots, n}$  a learning strategy. Then the (cumulated) regret is defined by

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

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  $\pi$  is clear from the context then we will shorten to  $R_n$ .

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



- Suppose a two-armed bandit with  $Q_1 = 1$  and  $Q_2 = -1$  and a learning



strategy  $\pi$  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  $\pi$  that  $R_n(\pi) \geq 0$  and  $\limsup_{n \rightarrow \infty} \frac{R_n(\pi)}{n} < \infty$ .
- Let  $R_n(\pi) = 0$ . Prove that  $\pi$  only chooses best arms. If there is only one best arm, then  $\pi$  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

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

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



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

There are different kind of bounds that one can aim for. First of all, one can aim for upper bounds and lower bounds for regret. In these lectures notes we mainly focus on upper bounds. Nonetheless, there are celebrated lower bounds due to Lai and Robbins that are not too hard to prove, 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  $\nu$  are called model-based, such bounds typically involve the differences between the action values  $Q_a$ ,
- bounds that only depend on  $n$  are called model independent, they are typically proved for entire classes of bandit models for which certain moment conditions are assumed.

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



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



quantities the understanding can still help us to understand how to work with the algorithms.

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



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

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

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



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

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

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

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

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

To compute the right hand side note that

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

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

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

□

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

We continue our discussion with the second perspective on how to analyse bandit algorithms. Approach (C) is more refined than (A), as an 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  $\nu$  is a bandit model and  $\pi$  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 regret decomposition lemma can be reformulated as follows:



**Lemma 1.2.10.**

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

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



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

The abstract discussion will become more accessible when analysing in detail specific algorithms. For explore-then-commit, using the regret-decomposition lemma, we will only estimate the regret while for the  $\varepsilon_n$ -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  $\varepsilon$ -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 ( $\varepsilon$ -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

We will now go into a few of the most popular algorithms. There is not much choice on how to design an algorithm. Essentially, all that can be done is to learn about arms that one is not too sure about (called exploration) and play arms that one expects to be good (called exploitation). We will not only present algorithms but also discuss theoretical regret bounds. Even though those won't be directly useful for our later understanding of reinforcement learning there is one



important learning: theoretical results allow to understand how to choose 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 commit-then-exploit algorithm

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



**Definition 1.3.1.** If  $(A_t, X_t)$  is a stochastic bandit process for some learning strategy  $\pi$ , 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  $\lim_{t \rightarrow \infty} \hat{Q}_a(t)$  is nothing but  $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t X_k^{(a)}$ , an average limit of an iid sequence which converges almost surely by the law of large numbers. The algorithm can be written in different ways. Either to try all arms  $m$ -times in a row or to alternate between the arms, for the pseudocode of Algorithm 1 we chose the latter. The learning strategy  $\pi_t$  is clearly an index strategy and can be written as

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

for arms denoted by  $1, \dots, K$ . 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  $\nu$

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

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

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

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

Obtain reward  $X_t$  by playing arm  $A_t$ ;

**end**

---

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  $\sigma^2$ .


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

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

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

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



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

The interesting question is the regret obtained during the exploitation phase if a suboptimal arm is played. It seems clear that the best arm is the most likely to be exploited as  $\hat{Q}_a(n) \approx Q_a(n)$  for large  $n$  by the law of large numbers. The regret thus comes from deviations of this large  $n$  principle, if the rewards of an arm exceed what they would yield on average. Since the simple explore-then-commit algorithm involves a lot of independence probabilities overestimation of arms can easily be estimated by inequalities which are known as concentration inequalities in probability theory.

*Proof, decomposing exploration and exploitation:* Without loss of generality (the order 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:

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

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

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

The appearing probability has the particularly nice form

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

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.  $\square$

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

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

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

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

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

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

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



**Definition 1.3.3.** A random variable  $X$  on a probability space  $(\Omega, \mathcal{A}, \mathbb{P})$  is called  $\sigma$ -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 acquainted to the definition:



- Show that every  $\sigma$ -subgaussian random variable satisfies  $\mathbb{V}[X] \leq \sigma^2$ .
- If  $X$  is  $\sigma$ -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  $\sigma_1$ -subgaussian and  $\sigma_2$ -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  $\sigma$ -subgaussian random variable is also  $\sigma'$ -subgaussian for every  $\sigma' > \sigma$  but this is not interesting as we will use  $\sigma$  to bound the variability ( $\sigma$  somehow measures the variance) as good as possible. This becomes clear in the next proposition, using  $\sigma$  larger than necessary only weakens the concentration inequality:



**Proposition 1.3.4.** If  $X$  is  $\sigma$ -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:

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

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

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



**Corollary 1.3.5. (Hoeffding's inequality)**

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

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

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

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

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

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

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

Also without the estimates the following logic is clear: If  $m$  is large (a lot of exploration) then the exploration regret is large (this is the first summand) and the exploitation regret is small (second summand). In the exercises you will explore numerically how to properly 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

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

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



The regret upper bound is minimized by

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

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



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

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

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

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



Do you think the explore-then-committ strategy with  $m$  from (1.1) is reasonable? No, it's cheating. The algorithm relies on  $n$ ,  $\sigma$ , and  $\Delta$  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 not accept dependence on  $n$ .
- The variance parameter  $\sigma$  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  $\sigma$  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 choose a best arm to obtain zero regret). Hence, the only non-trivial situation is that of unknown action values but known difference  $\Delta = Q_{a_1} - Q_{a_2}$ , but this is extremely special.

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  $\liminf_{n \rightarrow \infty} \frac{R_n(\pi)}{\log(n)} \geq \frac{\sigma^2}{\Delta}$  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 chooses 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 acquainted with the notation and simple modifications give useful algorithms. The plain idea is as follows: Take the past observations of each arm to define estimates  $\hat{Q}_a(t-1)$  of the action values  $Q_a$  at time  $t$  and then choose the maximal estimated arm, i.e.  $A_t := \arg\max_a \hat{Q}_a(t-1)$ . As always, since there is no preference order for the arms, if several estimated action values are equal we randomly choose 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  $\nu$ , vector  $\hat{Q}$ ,  $n$   
**Result:** actions  $A_1, \dots, A_n$  and rewards  $X_1, \dots, X_n$   
 Initialise  $T_a = 0$  for all  $a$ ;  
**while**  $t \leq n$  **do**  
     Set  $A_t = \operatorname{argmax}_a \hat{Q}_a$ ;  
     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  $R_1, \dots$  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  $R_n$ :

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

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 4.

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- $\sigma$ -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 subtracting the same constant from all arms to reduce the committal behavior effect described above. Again, it is unclear what constant should be subtracted without a priori knowledge on the action values. We will get back to this idea when we discuss the policy gradient method where this idea will return as the base-line trick.

### $\varepsilon$ -greedy bandit algorithms

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

---

#### Algorithm 3: $\varepsilon$ -greedy bandit algorithm

---

**Data:** bandit model  $\nu$ , exploration rate  $\varepsilon \in (0, 1)$ , vector  $\hat{Q}$ ,  $n$

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

```

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  $A_t = \operatorname{argmax}_a \hat{Q}_a$ ;
    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
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  $\varepsilon$ . 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  $\pi$  the learning strategy that first explores each arm once and then continues according to  $\varepsilon$ -greedy for some  $\varepsilon \in (0, 1)$  fixed. Show that the regret grows linearly:

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



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  $\varepsilon$ -greedy with reasonable regret bounds. We will only touch upon an example (introduced and studied in Auer et al.<sup>1</sup>) that one could call „explore-then- $\varepsilon$ -greedy with decreasing exploitation rate“. The algorithm replaces in the simple  $\varepsilon$ -greedy algorithm  $\varepsilon$  by the time-dependent exploration rate  $\varepsilon_t = \min\{1, \frac{CK}{d^2 t}\}$ . 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  $\varepsilon$ -greedy with decreasing exploration rate  $\varepsilon$ .



**Theorem 1.3.7. (Explore-then- $\varepsilon$ -greedy with decreasing  $\varepsilon$ )**

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  $\varepsilon$ -greedy algorithm with decreasing exploration rate  $\varepsilon_t = \min\{1, \frac{CK}{d^2 t}\}$  satisfies

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

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

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

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  $\Delta$  small and then compare the algorithm with the explore-then-commit algorithm 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

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

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



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

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

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





Suppose  $X_1, \dots, X_n$  are independent (not necessarily identically distributed!) random variables with variances  $\sigma_k^2$  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

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

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

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

Let  $x_0 := \frac{1}{2K} \sum_{s=1}^t \varepsilon_s$  with  $\varepsilon_n = \frac{CK}{d^2 n}$  the exploration rate from the algorithm. Suppose  $j$  is a suboptimal arm, we are going to estimate  $\mathbb{P}(A_t = j)$ . From the algorithm we obtain

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

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

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

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

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

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

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

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

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

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

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

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

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

□

We are not going to discuss further the  $\varepsilon$ -greedy algorithm. It turns out that that the dependence on the parameters  $K, \Delta_a, C$  is horribly bad compared to the UCB algorithm that is discussed next. Still, it is important to keep the previous algorithm in mind as  $\varepsilon$ -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 interpret 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

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

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 variables with mean  $Q$ , plugging-into Hoeffdings inequality 1.3.5 yields

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

Thus, the UCB exploration gives a simple control of the overestimation probabilities and with it the exploration caused by the optimism principle. The parameter  $\delta$  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  $\delta$

---

**Data:** bandit model  $\nu$ ,  $\delta \in (0, 1)$ ,  $n$ , vector  $\hat{Q}$

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

Initialise  $T_a = 0$  for all  $a$ ;

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

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

    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:

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

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  $\operatorname{UCB}(0, \delta) \equiv +\infty$  forces the algorithm to explore every arm at least once, a condition that reasonable algorithms should fulfill.



**Theorem 1.3.8.** Suppose  $\nu$  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  $\delta$  is fixed as  $\frac{1}{n^2}$  we skip the  $\delta$  from  $\operatorname{UCB}(t, \delta)$ . 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

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



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  $\{X_t^{(a)}\}_{t \leq n, a \in \mathcal{A}}$  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

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

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

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

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

First in words: If  $\omega \in G_{2,m}$  and  $\omega \in G_1$  then the UCB value for the  $m$ th 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  $t \leq n$  with  $T_a(t-1)(\omega) = m$  and  $A_t(\omega) = a$ . Using the definitions of  $G_1, G_{a,2}$ , and UCB yields

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

a contradiction to  $a = A_t(\omega) = \arg\max_b \text{UCB}_b(t-1)$ . 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:

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

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

holds that

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

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

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

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

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

Combining the above yields

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

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.  $\square$

Lecture 4

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



**Theorem 1.3.9.** Suppose  $\nu$  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. Pursuing this way leads to

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

Since  $\Delta$  can be chosen arbitrarily it suffices to minimise the righthand side as a function in  $\Delta$ . The minimum can be found easily by differentiation in  $\Delta$  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  $\sigma$ -subgaussian bandit models the UCB exploration bonus is modified as

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

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

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



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

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



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

It is very crucial to compare the constants appearing in the regret bounds. As mathematicians we tend to overestimate the importance of  $n$  and always think of  $n \rightarrow \infty$  (as we did in the formulation of Theorem 1.3.7). Keeping in mind logarithmic growth one quickly realises the importance of constants. As an example  $\log(1.000.000) \approx 13$  so that a constant  $\sqrt{K}$  or even worse  $1/\Delta$  can be much more interesting. As an example the constants (5 and  $1/\Delta$  appears even squared) in the regret bound of explore-then- $\epsilon$  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  $\text{SM}(\theta, x)$  on finite sets with  $d$  elements with probability weights

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

The weights are called Boltzmann weights of the vector  $x$ .

Typically, all  $\theta_i$  are equal, in which case  $\theta$  is called inverse temperature. The origin of such distributions lies in statistical physics. The softmax distributions is a so-called categorical distribution, written  $\text{Categorical}(p_1, \dots, p_d)$  which is also called a multinoulli or generalised Bernoulli distribution with probabilities  $p_1, \dots, p_d$ . For us, the following idea is much more important. If all  $\theta_k$  are non-negative then sampling from  $\text{SM}(\theta, x)$  is somewhat similar to the deterministic distribution  $M_x$  that only charges mass on the index  $\text{argmax}_k x_k$  because  $\text{SM}(\theta, x)$  assigns the highest probabilities to the coordinate with largest value  $x_k$ . But unlike the deterministic maximum distribution the softmax distribution only weights stronger the maximal element than the smaller elements. The role of  $\theta$  is important: for large  $\theta$  the softmax resembles the  $\text{argmax}$  distribution while for small  $\theta$  the distribution resembles the uniform distribution on  $\{1, \dots, d\}$ .

Replacing sampling from the (deterministic)  $\text{argmax}$  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  $\theta$  is crucial as the algorithm resembles two unfavorable algorithms with linear regret, both for small and large  $\theta$  (uniform and greedy exploration). In fact<sup>2</sup>, both constant and decreasing choices of

<sup>2</sup>N. Cesa-Bianchi, C. Gentile, G. Lugosi, G. Neu: Boltzmann exploration done right, NeuRIPS, (2017)

**Algorithm 5:** Simple Boltzmann exploration

---

**Data:** bandit model  $\nu$ , vector  $\hat{Q}$ , parameter  $\theta$   
**Result:** actions  $A_1, \dots, A_n$  and rewards  $X_1, \dots, X_n$   
 Initialise  $T_a = 0$  for all  $a$ ;  
**while**  $t \leq n$  **do**  
     Sample  $A_t$  from  $\text{SM}(\theta, \hat{Q})$ ;  
     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**

---

$\theta$  are unfavorable. To guess a better choice for  $\theta$  we need the following surprising lemma:



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

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

where  $g_1, \dots, g_d$  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  $\lambda_1, \lambda_2$ . Then  $\mathbb{P}(E_1 \leq E_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}$ . This is a simple integration exercise: with  $A = \{0 \leq x_1 \leq x_2\} \in \mathcal{B}(\mathbb{R}^2)$  the computation rules for vectors of independent random variables yields

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

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

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

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



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

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

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

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

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

where the  $g_a$  are iid Gumbel and independent of  $\hat{Q}$ . Does this look familiar? Of course, this is the greedy strategy with an additional random exploration bonus as we have seen in the UCB algorithm. Since typical values of  $g$  are around 1, motivated by the UCB algorithm a first idea should immediately come to mind:  $\theta^{-1}$  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-Bianchi et al. We will not go into detail here. The aim of this section was to link the Boltzmann exploration with greedy-type algorithms. Here is an interesting research question to think about. Let us forget that UCB-type exploration with Gumbel random exploration bonus is linked to the rather natural exploitation strategy given by Boltzmann softmax weights. It is then very natural to replace the Gumbel distribution by some other distribution. It seems most plausible that non-negative distributions should be more reasonable as a negative value turns the exploration bonus into an exploration malus which was never intended.



Use the simulation code provided for the exercises to play with different distributions and see if replacing the Gumbel distribution can be favorable. Choosing distributions that concentrate around 1 (Dirac measure at 1 gives UCB) might be reasonable, such as a Gamma distribution with shape 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 choosing one of the  $K$  arms and obtaining the corresponding outcome.



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

Keep in mind that since  $\mathcal{A}$  is assumed to be finite a distribution on  $\mathcal{A}$  is nothing but a probability vector, a vector of non-negative numbers that sums up to 1. A vector  $(0, \dots, 1, \dots, 0)$  with 1 for the  $a$ th arm (position) corresponds to playing only arm  $a$  and in this case  $Q_\pi = Q_a$ . A policy is optimal if the expected outcome  $Q_\pi$  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 learning 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 reinforcement 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  $\delta_a$  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  $\theta_*$  of  $J$ . Unfortunately, that approach has diverse difficulties which are topics of ongoing research:

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

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

some recent progress<sup>3</sup>. We will start with the second issue on how to deal with the gradient of  $J$  if  $J$  is unknown but samples can be generated. The trick that will occur a few times in this lecture course is the so-called log-trick, here in its simplest form:



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

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

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

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

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

or by a so-called batch of independent samples:

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

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

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

**Example 1.3.15.** Suppose  $d = |\mathcal{A}|$ , so that there is a parameter  $\theta_a$  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  $\theta_a$  needs to be sent to  $+\infty$ . The score-function is easily computed to be

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

<sup>3</sup>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)

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

if  $N = 1$ ,  $A_n = i$ , and  $X_i$  is a sample of arm  $i$ . Here is the first explanation why reinforcement learning is called reinforcement learning. Remember that the  $\theta_a$  are indirect parametrisations of the probabilities to play arm  $a$ . If at time  $n$  the policy current policy decides to play arm  $i$  then the probabilities are reinforced as follows. If the reward obtained by playing arm  $i$  is positive, let's interpret the positive reward as positive feedback, then the probability to play arm  $i$  is increased and all other probabilities are decreased. Similarly, if the feedback from playing arm  $i$  is negative, then the probability to do so in the next round is decreased and all other probabilities are increased. 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  $\pi_{\theta_0}$  is uniform on  $\mathcal{A}$ . If the first chosen arm, say  $a$ , yields a positive return then the the updated parameter vector  $\theta_1$  only has one positive entry, namely the  $a$ th entry. Hence, for the second update the  $a$ th arm is most likely to be chosen again. A way to reduce committal behavior is to subtract a baseline that is supposed to help distinguish good and bad arms<sup>4</sup>.



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-

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



tion)

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

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  $\nabla J$ , 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

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

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 (essentially 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  $Q(A) = \int_A f dP$ . 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

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



is called **relative entropic** (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 theoretic 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 derivative is the ratio of the densities.



**Lemma 1.4.2.** Suppose  $P \ll Q \ll \nu$ , then  $\frac{dP}{dQ} \frac{dQ}{d\nu} = \frac{dP}{d\nu}$ .

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

*Proof.*

$$P(B) = \int_B \frac{dP}{dQ} dQ = \int_B \frac{dP}{dQ} \frac{dQ}{d\nu} d\nu$$

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).  $\square$

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  $a_1, \dots, a_N$  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  $P \sim \text{Ber}(p)$  and  $Q \sim \text{Ber}(q)$  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 Lebesgue measure,

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

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

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

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  $p$  and  $q$  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

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

Next, we show that

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

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

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

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

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

the claim follows.  $\square$

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  $\sigma^2$ . What is unknown is the mean, it might be 0 or  $\Delta$ . Now we want to give a rule on how to determine if 0 or  $\Delta$  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

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

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  $\xi$ .

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 needed:



**Lemma 1.4.5. (Entropie-decomposition)**

Let  $k$  the number of arms and  $v = \{P_1, \dots, P_k\}$ ,  $v' = \{P'_1, \dots, P'_k\}$  the reward distributions of two  $k$ -armed bandits. Furthermore, let  $\pi$  a learning strategy and  $\mathbb{P}$  and  $\mathbb{P}'$  the probability distributions on  $(\mathbb{R} \times \{1, \dots, k\})^n$ , that describe the  $n$ -step bandit process for the two models under the learning strategy  $\pi$ . In this notation  $\mathbb{R}$  describes the reward and  $\mathcal{A} = \{1, \dots, k\}$  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  $\lambda$ ). The main observation is that the law of the bandit process for a given policy  $\pi$  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  $\rho$  is the counting measure on  $\{1, \dots, K\}$ . 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,

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

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

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



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

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

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

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

Plugging-into the definition of relative entropy yields

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

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 choosing  $\lambda$  to be Lebesgue measure replacing  $P_a$  by the densities  $p_a$  of  $P_a$ .  $\square$

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



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

$$R_n^*(\xi^k) \geq \frac{1}{27} \sqrt{(k-1)n}$$

holds for all  $n \geq k$ .

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

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

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

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

To save some notation we write  $\mathbb{P}$  and  $\mathbb{P}'$  for the laws of the bandit processes playing both policy  $\pi$  but on the different bandit models  $\mu$  and  $\mu'$ . 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  $\Delta$ . 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:

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

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  $\Delta$  when playing arm 1:

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

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

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

<sup>5</sup> 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  $\Delta$  (or just choosing  $\Delta = \sqrt{\frac{k-1}{4n}}$  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

<sup>5</sup>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  $\pi$  is called consistent over a class of bandits  $\mathcal{E}$  if for all  $\mu \in \mathcal{E}$  and all  $p > 0$  it holds that

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

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<sup>6</sup>:



**Theorem 1.4.8. (Lai-Robbins lower bound)**

If  $\pi$  is a consistent policy for a set  $\mathcal{E} = \mathcal{M}_1 \times \dots \times \mathcal{M}_k$  of bandit models, then the regret for all bandits  $\nu = (P_1, \dots, P_k)$  from  $\mathcal{E}$  satisfies

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

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:

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

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  $d_a = \frac{(Q_a - Q_*)^2}{\sigma^2} = \frac{\Delta_a^2}{\sigma^2} = D(P_a, P_*)$ . Thus, if all arms are Gaussian with variance  $\sigma^2$ , then the lower bound for UCB is

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

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  $\nu = (P_1, \dots, P_k) \in \mathcal{E}$ . Recalling the regret decomposition it is enough to prove that

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

holds for all suboptimal arm. Fix such an arm  $a$  and fix an  $\varepsilon > 0$  and define  $\nu'$  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 entropic decomposition for bandits yields

<sup>6</sup>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

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

First, from the regret decomposition it clearly holds that

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

Secondly, note that for  $\nu'$  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 played at least  $\frac{n}{2}$  times in total. Thus,

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

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

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

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

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

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

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

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

## Chapter 2

# More bandits

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

### **2.1 Adversarial bandits**

eine Vorlesung, naechstes Jahr

### **2.2 Contextual bandits**

eine Vorlesung, naechstes Jahr

## Part II

# Tabular Reinforcement Learning

## Chapter 3

# Basics: Stochastic Control and Dynamic Programming Methods

Lecture 6

### 3.1 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<sup>1</sup>. One should only keep in mind that the appearing  $Q$ -functions are not popular in stochastic optimal control,  $Q$ -functions are much more relevant to the reinforcement learning approaches developed in the next chapters.

#### 3.1.1 A quick dive into Markov chains

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

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

for all  $t \in \mathbb{N}_0$  and  $s_0, \dots, s_{t+1} \in \mathcal{S}$ . In words, transitions from a state to another do not depend on the past. The most important special case is that of a time-homogeneous Markov chain for which transition probabilities do not change over time. Time-homogeneous Markov chains are closely related to stochastic matrices (non-negative elements, all rows sum to 1). Given an initial distribution  $\mu$  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  $\mu$  and transition matrix  $P$  is uniquely determined through the basic path probabilities

$$\mathbb{P}(S_0 = s_0, \dots, S_n = s_n) = \mu(s_0)p_{s_0, s_1} \cdot \dots \cdot p_{s_{n-1}, s_n}. \quad (3.1)$$

---

<sup>1</sup>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

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

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

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

The computation tricks are always the same. Spell out the conditional probability, write the events of interest as disjoint union of simple paths, plug-in the path probability formula, and finally cancel from the appearing products. We always imagine a Markov chain to jump on a graph of states connected by arrows carrying the transition probabilities from states  $s$  to  $s'$ . Another way of expressing the Markov property is as follows: If  $\mathbb{P}(S_n = s') > 0$  and  $\tilde{\mathbb{P}} := \mathbb{P}(\cdot \mid S_n = s')$ , 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

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

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

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

Another way of expressing the Markov reward property is as follows: If  $\mathbb{P}(S_n = s') > 0$  and  $\tilde{\mathbb{P}} := \mathbb{P}(\cdot \mid S_n = s')$ , 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

$$\mathbb{E}_s[f(R_t, R_{t+1}, \dots) \mid S_t = s'] = \mathbb{E}_{s'}[f(R_0, R_1, \dots)]. \quad (3.2)$$

### 3.1.2 Markov decision processes

The situation of Markov decision processes (MDPs) is slightly more complicated. For MDPS we do not observe a game but take the role of a participant. We observe the Markov chain describing the match but can influence the transitions by taking actions. As an example, by substituting as a coach an additional attack player we increase the probability of players scoring goals (positive



reward) but decrease the probabilities of players winning duels which leads to larger probabilities to receive a goal (negative reward). The target will be to find a strategy (called policy) that maximises the expected reward of the game. Finding such a strategy, this is what reinforcement learning is all about!

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



**Definition 3.1.1. (Markov decision model)**

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

- (i)  $(\mathcal{S}, \bar{\mathcal{S}})$  is a measurable space, called the state space,
- (ii) For every  $s \in \mathcal{S}$ ,  $(\mathcal{A}_s, \bar{\mathcal{A}}_s)$  is a measurable space, called the action space of state  $s$ . The entire action space is defined to be  $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$ ,  $\mathcal{A}$  contains all actions and is equipped with the  $\sigma$ -algebra  $\bar{\mathcal{A}} = \sigma\left(\bigcup_{s \in \mathcal{S}} \bar{\mathcal{A}}_s\right)$ .
- (iii) A measurable set  $\mathcal{R} \subseteq \mathbb{R}$  of rewards with  $0 \in \mathcal{R}$ , its restricted Borel- $\sigma$ -algebra denoted by  $\bar{\mathcal{R}}$ .
- (iv) A function

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

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

- $(s, a) \mapsto p(B; s, a)$  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  $\sigma$ -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 3.1.5 below).

In most books you will find the notion  $p(\cdot | s, a)$ . In this course the notion „|“ will be used exclusively for conditional probabilities. 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 3.1.2. (Policy)**

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

- an initial Markov kernel  $\pi_0$  on  $\bar{\mathcal{A}} \times \mathcal{S}$ ,
- a sequence of probability kernels  $\pi = (\pi_t)_{t \in \mathbb{N}}$  on  $\bar{\mathcal{A}} \times ((\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S})$  such that

$$\pi_t(\mathcal{A}_s; s_0, a_0, \dots, s_{t-1}, a_{t-1}, s) = 1 \quad (3.3)$$

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

The set of all policies is denoted by  $\Pi$ .

A policy governs the choices of actions given the current state and all previous states-action pairs (not the rewards) seen before. Condition (3.3) 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 decision models are discrete and all  $\sigma$ -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  $\pi$  (plus an initial probability distribution over which state the stochastic process will start in). Let us formalize this stochastic process and prove its existence.



**Theorem 3.1.3. (Existence of (discrete) MDPs)**

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

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

To increase readability we will skip the  $\pi$  and often  $\mu$  from the notation.

<sup>2</sup> In words the mechanism goes as follows. In a state  $s$  an action is sampled according to the policy which is allowed to refer to the entire past of states and actions (not to the rewards!). The current and past state-action pairs  $(s, a)$  are used to sample the next state  $s'$  and the reward. Note: The reward is allowed to depend both on the current state-action pair  $(s, a)$  and the future state  $s'$ ! Typically, the reward will only depend on  $(s, a)$ , not on  $s'$ , but some examples require this greater generality.

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



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

<sup>2</sup>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  $\sigma$ -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.

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

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

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

In words: An initial state is chosen by  $\mu$ , the first action is sampled using  $\pi_0$ , 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  $\pi_i$ .



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

The exercise is important to get a feeling for the path probabilities. You will have to sum over all possible paths, i.e. over  $\sum_{a_0, r_0, s_0} \dots \sum_{a_T, r_T, s_T}$ , then pull out the factors that are independent of the summands to simplify backwards using the kernel property to obtain (here for the summands corresponding to  $T$ )

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

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



Measure for infinite time-horizon.

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

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

Consider the projections  $\pi_T^{T+1} : \Omega^{T+1} \rightarrow \Omega^T, \omega \mapsto \omega|_T$  and  $\pi_T : \Omega \rightarrow \Omega^T, \omega \mapsto \omega|_T$ . The projections simply remove the triple corresponding to time  $T + 1$  from the sequence. We now show the consistency property  $\mathbb{P}_T = \mathbb{P}_{T+1} \circ (\pi_T^{T+1})^{-1}$  for any  $T \in \mathbb{N}$ . If the consistency can be proved, then Kolmogorov's extension theorem gives a unique probability measure  $\mathbb{P}$  on  $(\Omega, \mathcal{F})$  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:

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

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

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



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

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



The properties claimed in the theorem hold.

The first two claims follow directly from the definition of the measure  $\mathbb{P}_\mu^\pi$  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  $\mathbb{P}_\mu^\pi$  (there is no dependence later than  $t$ ) yields

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

and

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

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

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



Check the other two claimed conditional probability identities. □



**Definition 3.1.4. (Markov decision process (MDP))**

Given a Markov decision model  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ , a policy  $\pi$ , and an initial distribution  $\mu$



on  $S$ , the stochastic process  $(S_t, A_t)_{t \in \mathbb{N}_0}$  on  $(\Omega, \mathcal{F}, \mathbb{P}_\mu^\pi)$  is called discrete-time Markov decision process (MDP),  $(R_t)_{t \in \mathbb{N}_0}$  the corresponding reward process. To abbreviate we will write  $\mathbb{P}_s^\pi$  instead of  $\mathbb{P}_{\delta_s}^\pi$ . The super- and subscript are sometimes removed from  $\mathbb{E}_\mu^\pi$  and  $\mathbb{P}_\mu^\pi$  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

$$\begin{aligned} & \mathbb{P}(S_0 = s_0, A_0 = a_0, R_0 = r_0, \dots, S_T = s_T, A_T = a_T, R_T = r_T) \\ &= \mu(s_0) \cdot \pi_0(a_0; s_0) \cdot \prod_{i=1}^T p(s_i, r_{i-1}; s_{i-1}, a_{i-1}) \cdot \pi_i(a_i; s_0, a_0, \dots, a_{i-1}, s_i) \quad (3.4) \\ & \cdot p(\mathcal{S} \times \{r_T\}, s_T, a_t), \end{aligned}$$

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

It is always very instructive to compare with the situation of an ordinary Markov chain that appears as a special case if  $\mathcal{A} = \{a\}$  and  $\mathcal{R} = \{0\}$ . 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 (3.1). No doubt, for MDPs the formula is more complicated but many probabilities can be computed similarly to the situation of a Markov chain. Playing with conditional probabilities it is instructive to check the following formulas (using a cancellation of the form  $\sum_i (a \cdot b_i) / \sum_i b_i = a$ ):

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

and

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

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

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

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

$$p(s', r; s, a) := h(s'; s, a) \cdot q(r; s, a) \quad (3.5)$$

and the corresponding Markov decision process samples next states/rewards independently. Now the Markov decision model in the sense of Definition 3.1.1 with Markov decision process from Theorem 3.1.3 has exactly the claimed transitions. Plugging-in  $p$  immediately gives

$$\begin{aligned} & \mathbb{P}(R_t = r_t | S_t = s_t, A_t = a_t, S_{t+1} = s_{t+1}) \\ \stackrel{\text{cond. prob.}}{=} & \frac{\mathbb{P}(S_{t+1} = s_{t+1}, R_t = r_t | S_t = s_t, A_t = a_t)}{\mathbb{P}(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t)} \\ \stackrel{3.1.3}{=} & \frac{p(s_{t+1}, r_t; s_t, a_t)}{p(\{s_{t+1}\} \times \mathcal{R}; s_t, a_t)} \\ \stackrel{(3.5)}{=} & p(\mathcal{S} \times \{r_t\}; s_t, a_t) \\ \stackrel{3.1.3}{=} & \mathbb{P}(R_t = r_t | S_t = s_t, A_t = a_t), \end{aligned}$$

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

**Example 3.1.6.** 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 3.1.7.** 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


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



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

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 3.1.8.** 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  $\Delta$  and we will always assume  $p(s, 0; s, a) = 1$  for 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 3.1.9. (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 through a wall (or additional obstacles). The aim is to move from the starting position  $S$  to the target position  $G$ . To make the problem a bit more difficult, there is a bomb  $B$  just before the goal  $G$ . If the robot lands in the bomb or in the target, the game is over. In

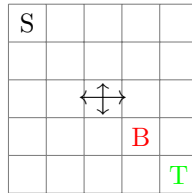


Figure 3.1: Grid world of size 5. The start is marked by  $S$ , the bomb by  $B$ , and the goal by  $G$ .

the following we will formulate this example as a Markov Decision Model. For this purpose, the individual objects from Definition 3.1.1 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  $S := \{(i, j), i, j \in \{0, 1, \dots, Z - 1\}\}$ . Since the state space  $S$  is finite, we will use as  $\sigma$ -algebra the power set, i.e.  $\mathcal{S} := \mathcal{P}(S)$ .

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 3.2: Representation of the state space of Example 3.1.9 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:

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

The advantage of this definition is that by simply adding states and actions we get the new state of the robot. In the following we want 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  $\bar{\mathcal{R}} := \mathcal{P}(\mathcal{R})$ . 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  $g : S \rightarrow \mathcal{R}$  which should return the associated reward for a state  $s \in S$ . The function is given by

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

In addition, we still need to define the set of terminal states  $\Delta$ . 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  $(s, a) \in S \times \mathcal{A}_s$ . 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 3.1.7. 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 3.1.10. (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 a discrete amount  $0, 1, 2, \dots, M$  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 than the ice vendor has in stock, the ice cream must be stored overnight and can then be sold the next day. There are costs for storing ice cream overnight, thus, the ice vendor should carefully decide about the amount of ice cream produced/bought in the morning. In addition, there are costs for the production/purchase of ice cream. For the sale of ice cream the vendor receives a fixed price per ice cream scoop. For simplicity, we also assume that revenues and costs do not change over time (which is realistic for ice cream, the price is typically adjusted once per year). The trade-off in this problem is simple. How much ice should be ordered in the morning so that the vendor can maximize the revenue (serve demand) but keep the costs low. The ice vendor situation can be formulated as Markov decision process. The state space is given by the stored amount of ice:  $\mathcal{S} = \{0, 1, \dots, M\}$ . An action models the amount of ice cream produced/ordered in a morning. If there are already  $s \in \mathcal{S}$  scoops in stock, the vendor can produce/order at most  $M - s$  many scoops. Thus,  $\mathcal{A}_s = \{0, 1, \dots, M - s\}$  and  $\mathcal{A} = \{0, \dots, M\}$ . For demand, we assume that it is independently identically distributed regardless of timing. Thus  $\mathbb{P}(D_t = d) = p_d$  for  $d \in \mathbb{N}$  holds for all time  $t \in \mathbb{N}$ . For simplicity, let us assume the demand can only take values in  $\{0, 1, \dots, M\}$ . In order to define the transition reward function and the reward, auxiliary functions are used to determine the revenues and costs. The selling price of a scoop of ice cream is given by a function  $f : \mathcal{S} \rightarrow \mathbb{R}_+$ , for instance  $f(s) = c \cdot s$ , where  $c > 0$  is the selling price for a scoop of ice cream. Similarly, we define a mapping  $o : \mathcal{A} \rightarrow \mathbb{R}_+$  for the production/purchase cost of the products and  $k : \mathcal{S} \rightarrow \mathbb{R}_+$  for the storage cost. Thus, for a pair  $(s, a) \in \mathcal{S} \times \mathcal{A}_s$  and another state  $s' \in \mathcal{S}$ , the gain is given by the mapping  $R : (\mathcal{S} \times \mathcal{A}_s) \times \mathcal{S} \rightarrow \mathbb{R}$  with

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

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

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

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 probabilities as

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

and the reward expectations as

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

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


**Definition 3.1.11. (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

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

The set of all Markov policies is denoted by  $\Pi_M$ .

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

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

The set of all stationary policies is denoted by  $\Pi_S$ .

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

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

The set of all deterministic stationary policies is denoted by  $\Pi_S^D$ .

In the stationary cases we typically write  $\pi$  instead of  $\varphi$ .

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  $R_1, \dots, R_n$  are the outcomes of playing the arms chosen according to  $\pi$ . 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 3.1.12. (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 (3.4). 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

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

Finally, if  $\varphi_t = \varphi$  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  $t$ .  $\square$



**Proposition 3.1.13. (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

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

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!  $\square$



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  $\pi_0$  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.

### 3.1.3 Stochastic control theory

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



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

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

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 3.4.
- 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

$$\begin{aligned} \mathbb{E}^{\pi} \left[ \sum_{t=0}^T R_t \right] &= \mathbb{E}^{\pi} \left[ \sum_{k=0}^{\infty} (1 - \gamma) \gamma^k \sum_{t=0}^k R_t \right] \\ &= (1 - \gamma) \mathbb{E}^{\pi} \left[ \sum_{t=0}^{\infty} \sum_{k=t}^{\infty} \gamma^k R_t \right] \\ &= (1 - \gamma) \mathbb{E}^{\pi} \left[ \sum_{t=0}^{\infty} \frac{\gamma^t}{1 - \gamma} R_t \right] \\ &= \mathbb{E}^{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \right] \end{aligned} \tag{3.6}$$

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  $\sigma$ -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 3.1.14. (Value functions)**

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

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

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

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

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  $\pi_0$ . In contrast,  $Q$  also fixes the first action  $a_0 = a$ . Think about chess. The state-value function shall describe 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  $V^\pi$  and  $Q^\pi$ .



**Proposition 3.1.15. (Bellman equations for  $Q^\pi$  and  $V^\pi$ )**

Suppose  $\pi$  is a stationary policy, then  $Q^\pi$  and  $V^\pi$  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

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

*Proof.* Recall from Proposition 3.1.13 that  $(S, A, R)$  is a Markov reward process so that by (3.2)

$$\mathbb{E}_{s,a}^\pi \left[ R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \mid S_1 = s', A_1 = a' \right] = \mathbb{E}_{s',a'}^\pi \left[ R_0 + \gamma R_1 + \gamma^2 R_2 + \dots \right] = Q^\pi(s', a').$$

Thus, using the formula of total probability,

$$\begin{aligned} & Q^\pi(s, a) \\ &= \mathbb{E}_{s,a}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_t \right] \\ &= \mathbb{E}_{s,a}^\pi [R_0] + \mathbb{E}_{s,a}^\pi \left[ \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \dots \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}_{s'}} \mathbb{E}_{s,a}^\pi \left[ R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \mid S_1 = s', A_1 = a' \right] \mathbb{P}_{s,a}^\pi(S_1 = s', A_1 = a') \end{aligned}$$

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

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:

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

□

The definition of  $V^\pi$  immediately allows to compute  $V^\pi$  from  $Q^\pi$ . For stationary policies the opposite also holds true:



**Corollary 3.1.16.** 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 (3.1.16).

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 3.1.17. (Banach fixed-point theorem)**

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

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

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  $X := \{v : \mathcal{S} \rightarrow \mathbb{R}\}$  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  $Y := \{q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}\}$  equipped with the supremum norm which is then nothing but the Banach space  $\mathbb{R}^{|\mathcal{S}| \cdot |\mathcal{A}|}$ .



**Definition 3.1.18. (Bellman expectation operator)**

Given a Markov decision model and a stationary policy  $\pi$  the operators

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

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 Bellman 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 3.1.15 that value functions are fixed points for Bellman expectation operators. Combined with the simple contractivity property with respect to the maximum norm the first major theorem follows:



**Theorem 3.1.19.** Both Bellman expectation operators are contractions with contraction constant  $\gamma$ . Their unique fixed points are the value functions  $V^\pi$  and  $Q^\pi$ , 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

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

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 3.1.20. (Optimal policy & optimal value function)**

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



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

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

is called optimal (state) value function.

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

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

is called optimal state-action value function.

(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 not 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  $X = \{v : \mathcal{S} \rightarrow \mathbb{R}\}$  and  $Y := \{q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}\}$ . 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 3.1.21. (Bellman optimality operators)**

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

(i) The non-linear system of equations

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

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

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

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

(ii) The non-linear system of equations

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





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

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

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

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

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



**Theorem 3.1.22.** 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

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

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

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

□

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 3.1.23.** 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 be 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 3.1.16). For didactic reasons let us first suppose that  $\pi$  is stationary. Then we get from Proposition 3.1.15

$$\begin{aligned} \sup_{\pi \in \Pi_S} Q^\pi(s, a) &\stackrel{3.1.15}{=} \sup_{\pi \in \Pi_S} \left( r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi(a'; s') Q^\pi(s', a') \right) \\ &= r(s, a) + \gamma \sup_{\pi \in \Pi_S} \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi(a', s') Q^\pi(s', a') \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} \sup_{\pi \in \Pi_S} Q^\pi(s', a'). \end{aligned}$$

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 advantage 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

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

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

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

$$\begin{aligned} \max_{a \in \mathcal{A}_s} \bar{Q}^*(s, a) &= \max_{a \in \mathcal{A}_s} \sup_{\pi \in \Pi_S} Q^\pi(s, a) \\ &= \sup_{\pi \in \Pi_S} \max_{a \in \mathcal{A}_s} Q^\pi(s, a) \\ &\stackrel{(*)}{=} \sup_{(\pi_t)_{t \geq 1}} \sup_{\pi_0} V^\pi(s) \\ &= \sup_{\pi \in \Pi_S} V^\pi(s) \\ &= \bar{V}^*(s). \end{aligned}$$

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  $\bar{V}^*$ :

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

**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

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

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.

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

than we still get

$$\mathbb{E}_{s, a}^{\pi} \left[ \sum_{t=1}^{\infty} \gamma^t R_t \mid S_1 = s', A_1 = a' \right] = \gamma Q^{\tilde{\pi}}(s', a'). \quad (3.7)$$

Before checking (3.7) 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

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

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 (3.7). As in the proof of the Markov reward property for stationary policies define the shifted process

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

and the new probability measure  $\tilde{\mathbb{P}} := \mathbb{P}_{s,a}^{\pi}(\cdot | S_1 = s', A_1 = a')$ . On the probability space  $(\Omega, \mathcal{F}, \tilde{\mathbb{P}}^{\pi})$ , the process  $(\tilde{S}_t, \tilde{A}_t)_{t \geq 0}$  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:

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

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

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 3.1.24. (Greedy policy)**

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

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

is called a greedy policy with respect to  $q$ . Sometimes we also write  $\operatorname{greedy}(q)$  instead of  $\pi_q$ . If  $\pi$  is greedy with respect to a  $q_v$  obtained from the  $V$ - $Q$ -transfer operator

$$q_v(s, a) := r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) v(s'), \quad s \in \mathcal{S}, a \in \mathcal{A}_s, \quad (3.8)$$

then we also write  $\pi_v$ .

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 3.1.25.** Suppose  $\pi_q$  is a greedy policy obtained from a  $q$ , then  $T^*q = T^{\pi_q}q$ . If  $q$  is obtained from a vector  $v$  through (3.8) then it also holds that  $T^*v = T^{\pi_q}v$ .

*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  $v$ :

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

where we used the definition of the Bellman operators and the definition of the greedy policy in the second equality.  $\square$

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



**Theorem 3.1.26. (Dynamic programming algorithm)**

An optimal policy  $\pi^*$  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, transferring 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 Bellman's state-action optimality equation. By uniqueness  $q$  equals  $Q^*$ . It remains to show  $V^* = V^{\pi_q}$ , then  $\pi_q$  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  $V^{\pi_q}(s) = \sum_{a \in \mathcal{A}_s} \pi_q(a; s) Q^{\pi_q}(s, a)$  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  $\pi_q$  is optimal.  $\square$

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  $\pi^*$  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 3.1.27.** A sequence  $(\pi^n)_{n \in \mathbb{N}}$  of policies for a Markov decision model is called a learning strategy, if  $\pi^{n+1}$  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  $\pi^*$ . What we mean by convergence will depend on the context, the minimal requirement is convergence of the value function to the optimal value function, i.e.  $\|V^{\pi^n} - V^*\|_\infty \rightarrow 0$ . In contrast to stochastic bandit theory the regret plays no role in reinforcement learning, the situation is too complex.

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

- value function based learning,
- policy based learning.

For value function the idea is to learn the optimal value function  $V^*$  (or optimal state-value function  $Q^*$ ) and then infer the optimal policy  $\pi^*$  by taking argmax (the greedy policy from Theorem 3.1.26). In contrast, policy learning tries to approximate directly the optimal policy  $\pi^*$ .



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

## 3.2 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 a unique fixed point of a contraction mapping  $T$  but also the convergence of the sequence  $(T^{n+1}v_0)_{n \in \mathbb{N}_0}$  to that fixed point for arbitrary  $v_0$ . We learnt that for  $T^*$  the unique fixed point corresponded to  $V^*$ . The idea of value iteration is to use this

**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  $V := V_{\text{new}}$   **for**  $s \in \mathcal{S}$  **do**

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

**end**   $\Delta := \max_{s \in \mathcal{S}} (|V_{\text{new}}(s) - V(s)|)$ **end**return  $V := V_{\text{new}}$  and the greedy policy with respect to  $V$ 

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 repeatedly.



**Theorem 3.2.1.** 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

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

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

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

Using continuity of the norm yields

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

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

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 3.2.2.** For  $\varepsilon > 0$  a policy  $\pi \in \Pi$  is called  $\varepsilon$ -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

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V(s'). \quad (3.9)$$

We now do the same and use (3.9) to define from the approximation  $V$  of  $V^*$  an approximate  $Q$ -function. Then the corresponding greedy policy is  $\varepsilon$ -optimal:



**Theorem 3.2.3.** Suppose  $V$  is the output of Algorithm 6 and  $Q$  is obtained from  $V$  using the transfer operator (3.9). Then the greedy policy  $\pi_Q$  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:

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

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

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

Rearranging this equation yields

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

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

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

This proves the claim.  $\square$



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



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

$$\|y_{n+1} - y^*\| \leq K \|y_n - y^*\|^\alpha, \quad n \in \mathbb{N}. \quad (3.10)$$

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

$$\|v_{n+1} - V^*\|_\infty = \|T^*v_n - T^*V^*\|_\infty \leq \gamma \|v_n - V^*\|_\infty, \quad n \in \mathbb{N}. \quad (3.11)$$

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

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

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



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

### 3.3 Basic policy iteration (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  $Q^\pi$  (or  $V^\pi$ ) for the currently best known policy  $\pi$ .
- 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.

### 3.3.1 Policy evaluation

We are now going to address the question on how to compute  $V^\pi(s)$  for a given policy  $\pi$ . 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  $\pi \in \Pi_s$ :

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

and that the value function  $V^\pi$  is a fixed point of the Bellman operator  $T^\pi$ .



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

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

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

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

where

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

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

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



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

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



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

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi. \quad (3.12)$$

*Proof.* This follows immediately as the Bellman expectation operator is a contraction (compare the exercise after Theorem 3.1.22). To compute the solution is straightforward linear algebra operation:

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

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

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

As seen with value iteration we can use the convergence properties of the iterates of a contraction mapping. We want to do the same for the operator  $T^\pi$  for a stationary policy. Using Banach's fixed point theorem convergence  $(T^\pi)^n v_0 \rightarrow V^\pi$  holds for any initialisation  $v_0$ . Implemented as an algorithm we obtain Algorithm 7.

---

**Algorithm 7:** Iterative policy evaluation (Naive)

---

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

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

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

$\Delta := 1$

**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 3.3.2.** Algorithm 7 terminates and  $\|V - V^\pi\|_\infty < \frac{\gamma \varepsilon}{1 - \gamma}$ .

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

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  $V_{\text{new}}(s)$ . This can be done more efficient by directly using available data, see Algorithm 8. The algorithm does not perform the matrix computation with  $T^\pi$  directly but row by row, it updates coordinate by coordinate instead of all coordinates at once.

**Algorithm 8:** Iterative policy evaluation (totally asynchronous updates)

---

**Data:** Policy  $\pi \in \Pi_S$ ,  $\varepsilon > 0$   
**Result:** Approximation  $V \approx V^\pi$   
Initialize  $V(s) = 0$  for all  $s \in \mathcal{S}$   
 $\Delta := 2\varepsilon$   
**while**  $\Delta > \varepsilon$  **do**  
     $\Delta := 0$   
    **for**  $s \in \mathcal{S}$  **do**  
         $v := V(s)$   
         $V(s) := \underbrace{\sum_{a \in \mathcal{A}_s} \pi(a; s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r; s, a) [r + \gamma V(s')]}_{T^\pi V(s) = (r_\pi + P_\pi)V(s)}$   
         $\Delta := \max(\Delta, |v - V(s)|)$   
    **end**  
**end**

---



Try to prove convergence of the totally asynchronous policy evaluation algorithm (without termination) to  $V^\pi$ . To do so enumerate  $\mathcal{S}$  as  $s_1, \dots, s_K$  and define

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

i.e.  $T^\pi$  is only applied to coordinate  $s$  while leaving all other coordinates unchanged. Then the inner loop of the algorithm performs nothing but the composition  $\bar{T}^\pi := (T_{s_K}^\pi \circ \dots \circ T_{s_1}^\pi)(V)$ , which is not the same as applying  $T^\pi$ ! Show that  $V^\pi$  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  $V^\pi$ .

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

Lecture 10

### 3.3.2 Policy improvement

We now aim to improve a given policy, that is to slightly change it such that its value function takes larger values. Mathematically speaking, for a policy  $\pi \in \Pi$  and value function  $V^\pi$  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  $\pi$  is not optimal, there always exists a strict improvement, e.g. the optimal policy. We now want to define a procedure to update a non-optimal policy to an improved policy. The key idea is to change the policy at a single state  $s \in \mathcal{S}$  to a particular action. For this we look at the action-value function of a stationary policy  $\pi \in \Pi_S$ . Recalling that

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

it becomes apparent that

$$\max_{a \in \mathcal{A}_s} Q^\pi(s, a) \geq V^\pi(s). \quad (3.13)$$

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  $\pi$  is at least as good as following the policy  $\pi$ . Recalling Definition 3.1.24 this suggests to use the greedy policy  $\pi_{Q^\pi}$  induced by the  $Q$ -function of the current policy  $\pi$  which then leads to the simple policy improvement algorithm. The improvement of the greedy policy improvement is a special case of

---

**Algorithm 9:** Greedy policy improvement

---

**Data:** policy  $\pi \in \Pi_S$ ,  $Q$ -function  $Q^\pi$   
**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**

---

the policy improvement theorem.



**Theorem 3.3.3. (Policy improvement theorem)**

Let  $\pi, \pi' \in \Pi_S$  be two stationary policies, then the following hold:

(i) If

$$V^\pi(s) \leq \sum_{a \in \mathcal{A}_s} \pi'(a; s) Q^\pi(s, a), \quad \forall s \in \mathcal{S}, \quad (3.14)$$

then  $\pi'$  is an improvement of  $\pi$ .

(ii) If there is a strict inequality in (3.14) for some state  $s$  then the improvement is strict.

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

On a heuristic level the theorem is trivial. If a stationary policy  $\pi'$  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  $\pi'$  is bigger than that for  $\pi$ .

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

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

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

$$V^\pi(s) \leq T^{\pi'} V^\pi(s) \leq T^{\pi'} T^{\pi'} V^\pi(s) \leq \dots \leq \lim_{k \rightarrow \infty} (T^{\pi'})^k V^\pi(s).$$

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,  $V^\pi(s) \leq V^{\pi'}(s)$  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  $\pi'$  satisfies the improvement condition:

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

Hence, (i) implies the claim.  $\square$

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



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

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

*Proof.* It follows from Lemma 3.1.16 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,  $Q^\pi$  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).  $\square$



**Corollary 3.3.5.** For a non-optimal policy  $\pi \in \Pi_S$  and the greedy policy  $\pi' = \text{greedy}(Q^\pi)$  obtained from  $Q^\pi$  is a strict policy improvement.

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

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



**Definition 3.3.6.** A policy  $\pi \in \Pi_S$  is called

- soft, if it fulfils

$$\pi(a; s) > 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s,$$

- $\varepsilon$ -soft for some  $1 \geq \varepsilon > 0$ , if it fulfils

$$\pi(a; s) > \frac{\varepsilon}{|\mathcal{A}_s|} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s,$$

- $\varepsilon$ -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  $\varepsilon$ ,



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  $\varepsilon$ -greedy learning strategies for stochastic bandits. Such policies are considered suboptimal as they lead to linear regret, they play suboptimal arms with probability  $\varepsilon$ . Similarly,  $\varepsilon$ -soft policies cannot be optimal as suboptimal actions must be played in contrast to greedy policies that only play optimal actions. Hence,  $\varepsilon$ -greedy policies will typically not improve policies, but they do improve all other  $\varepsilon$ -soft policies:



**Proposition 3.3.7.** If  $\pi \in \Pi_S$  is  $\varepsilon$ -soft, then the  $\varepsilon$ -greedy policy  $\pi'$  obtained from  $Q^\pi$  improves  $\pi$ .

*Proof.* This follows by checking the condition from the policy improvement theorem:

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

□

### 3.3.3 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  $V^\pi$  and then improve to  $\pi'$  using the greedy strategy obtained from  $Q^\pi$ . The above results show that every improvement step improves the policy and the limit of the procedure is  $\pi^*$ . Here is an illustration of the procedure:

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

The algorithm obtained this way is called policy iteration algorithm. Since there are many variations how to perform the policy evaluation (exact or approximatedly) we will meet several variants in the chapters below. We will first restrict ourselves to exact evaluations of  $V^\pi$  using the matrix inversion (3.12).



**Theorem 3.3.8. (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  $\pi^*$ .

*Proof.* By Corollary 3.3.5 in each iteration there is a strict improvement of the next policy  $\pi'$  (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 3.3.4 the termination of the algorithm, thus  $V^\pi = V^{\pi'}$ , implies that  $\pi'$  is optimal. □

**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  $V_{\text{new}}, \pi_{\text{new}}$   
stop = *False*  
**while** stop = *False* **do**  
    Policy evaluation (critic): Obtain  $V^\pi$  by computing (3.12).  
    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  $\pi_{\text{new}}$  from  $Q^\pi$   
    **if**  $Q^{\pi_{\text{new}}} = Q^\pi$  **then**  
        | stop = *True*  
    **end**  
**end**  
return  $\pi^* = \pi$

---

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  $V^{\pi_n}$  will not be computed explicitly but the policy iteration algorithm will be used in an approximate manner where  $V^\pi$  is estimated. One example is to replace the explicit evaluation of  $V^\pi$  by a few steps of the Banach fixed point iteration corresponding to  $T^\pi$ , compare Algorithms 7 or 8. Other examples will be discussed in Chapter 4. All algorithms alternating between value function estimation a policy improvement (not necessarily greedy)

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

will be called **generalised policy iteration algorithm**. The aim will be to generate algorithms that converge as quickly as possible to a policy  $\pi$  which is as close to  $\pi^*$  as possible.

**Algorithm 11:** Generalised policy iteration (actor-critic) paradigm

---

**Data:** initial policy  $\pi$   
**Result:** optimal policy  $\pi^*$  (or approximation of  $\pi^*$ )  
**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  $\pi_{\text{new}}$  by some algorithm.  
    Set  $\pi = \pi_{\text{new}}$ .  
**end**  
return  $\pi$

---

It is not clear at all if a generalised policy iteration algorithm converges to an optimal policy and when to stop the algorithm! The errors made by policy evaluation and improvement might easily build up. It is a very non-trivial task to find and tune approximate algorithms that converge. In Section 4.2.1 will come back to generalised policy iteration with  $\varepsilon$ -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  $\varepsilon$ -greedy policies that will be useful later to introduce exploitation into some algorithms. Let us extend the notion of optimality to the class of soft policies.



**Definition 3.3.9.** An  $\varepsilon$ -soft policy  $\pi^*$  is called  $\varepsilon$ -soft optimal if

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

There is a nice trick how to show convergence of  $\varepsilon$ -greedy policy iteration. Since  $\varepsilon$ -greedy policies play some action with probability  $\varepsilon$  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  $\varepsilon$ -soft policies.



**Theorem 3.3.10. ( $\varepsilon$ -greedy exact policy iteration)**

Started in any  $\varepsilon$ -soft policy the generalised policy iteration algorithm with exact policy evaluation and  $\varepsilon$ -greedy policy update converges to an  $\varepsilon$ -soft optimal policy.

*Proof.* To prove convergence of exact policy iteration with greedy policy update (Theorem 3.3.8) used the Bellman equation to guarantee that no further improvement means that the current policy is already optimal. This does not work for the  $\varepsilon$ -greedy policy iteration algorithm. To circumvent this problem we use a trick and “move the  $\varepsilon$ -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  $\varepsilon$ , the transition kernel will ignore the selected action and behave as if a uniformly random action was chosen. We can transform stationary  $\varepsilon$ -soft policies  $\pi$  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  $\varepsilon$ -soft policy  $\pi$  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  $\varepsilon$ -soft policies in the old MDP and all stationary policies in the new MDP. We now show, that the value functions  $V^{\pi}$  stay invariant with regard to this mapping. For that sake note that

$$\tilde{p}(s'; s, a) = (1 - \varepsilon)p(s'; s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} p(s'; s, b)$$

and

$$\tilde{r}(s, a) = (1 - \varepsilon)r(s, a) + \frac{\varepsilon}{|\mathcal{A}_s|} \sum_{b \in \mathcal{A}_s} r(s, b),$$

hence,

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

Similarly:

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

Combining the above yields

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

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

$$\sup_{\tilde{\pi} \in \tilde{\Pi}_S} \tilde{V}^{\tilde{\pi}}(s) = \sup_{\pi \in \Pi_S} V^\pi(s), \quad (3.15)$$

For the  $Q$ -functions we obtain

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

which implies that

$$\arg \max_{a \in \mathcal{A}_s} \tilde{Q}^{\tilde{\pi}}(s, a) = \arg \max_{a \in \mathcal{A}_s} Q^\pi(s, a).$$

Therefore greedy with respect to  $Q^\pi$  and  $\tilde{Q}^{\tilde{\pi}}$  is the same. Let  $\pi_n$  be an  $\varepsilon$ -soft policy, and let  $\pi_{n+1}$  be  $\varepsilon$ -greedy with regard to  $Q^{\pi_n}$ . Then  $\tilde{\pi}_{n+1} := f(\pi_{n+1})$  is greedy with respect to  $\tilde{Q}^{\tilde{\pi}_n}$ :

$$\tilde{\pi}_{n+1}(a; s) = \frac{\pi(a; s) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = \begin{cases} \frac{\left( (1 - \varepsilon) + \frac{\varepsilon}{|\mathcal{A}_s|} \right) - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = 1 & : a = a^*(s) \\ \frac{\frac{\varepsilon}{|\mathcal{A}_s|} - \frac{\varepsilon}{|\mathcal{A}_s|}}{1 - \varepsilon} = 0 & : a \text{ otherwise} \end{cases}.$$

Since we proved in Theorem 3.3.8 convergence of exact policy iteration with greedy policy updates the proof can be finished:

$$\sup_{\pi \in \Pi_S} V^\pi(s) \stackrel{(3.15)}{=} \sup_{\tilde{\pi} \in \tilde{\Pi}_S} \tilde{V}^{\tilde{\pi}}(s) = \tilde{V}^{\tilde{\pi}^*}(s) = \lim_{n \rightarrow \infty} \tilde{V}^{\tilde{\pi}_n}(s) = \lim_{n \rightarrow \infty} V^{\pi_n}(s)$$

□

On first view it is completely unclear why there might be any interest in  $\varepsilon$ -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,  $\varepsilon$ -greedy policies have pleasant exploration properties, they force the algorithm to look at all actions and not only the ones that are already known to be good. The reader might compare with the exploration-exploitation trade-off for stochastic bandits in Chapter 1.

## 3.4 Stochastic control in finite time

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 forgetfulness 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!<sup>3</sup>

### 3.4.1 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 restricted 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 control problem consists in optimising the state value function  $\mathbb{E}_s^\pi[\sum_{t=0}^{T-1} R_t]$  over all policies.

**Example 3.4.1.** 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  $\gamma$ ). 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.  $\pi$  consists of  $T - t$  Markov kernels. The definition of the state value function for  $T$ -step MDPs is as follows.



**Definition 3.4.2.** For any policy  $\pi \in \Pi_t^T$  the functions (vectors) defined by  $V_{T,T}^\pi \equiv 0$  and

$$V_{t,T}^\pi : \mathcal{S} \rightarrow \mathbb{R}, \quad s \mapsto \mathbb{E}_s^{\tilde{\pi}} \left[ \sum_{i=0}^{T-t-1} R_i \right] =: \mathbb{E}_{S_t=s}^\pi \left[ \sum_{i=t}^{T-1} R_i \right],$$

for  $t < T$  are called time-state value functions, where  $\tilde{\pi}$  is the policy  $\pi$  shifted by  $t$ , i.e.  $\tilde{\pi}_i = \pi_{t+i}$  for  $i = 0, \dots, T - t - 1$ . The function  $V_{0,T}^\pi$  is called state value function.

<sup>3</sup>at some point also include a terminal payout (which now is 0)

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



**Definition 3.4.3.** For any policy  $\pi \in \Pi_t^T$  the functions (matrices) defined by  $Q_{T,T}^\pi \equiv 0$  and

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

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

From now on we will drop the subscript  $T$ . We will just write  $V_t^\pi$  and  $Q_t^\pi$ , 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 3.1.16, we also get a lemma that describes the relation between the state value function and the state-action value function for a fixed policy:



**Proposition 3.4.4.** Given a Markovian policy  $\pi = (\pi_t)_{t \leq T}$  and a  $T$ -step Markov decision problem. Then the following relation between the state and state-action value function hold

$$\begin{aligned} V_t^\pi(s) &= \sum_{a \in \mathcal{A}_s} \pi_t(a; s) Q_t^\pi(s, a), \\ Q_t^\pi(s, a) &= r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^\pi(s') \end{aligned}$$

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

$$\begin{aligned} V_t^\pi(s) &= \sum_{a \in \mathcal{A}} \pi_t(a; s) \left[ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^\pi(s') \right], \\ Q_t^\pi(s, a) &= r(s, a) + \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_s} p(s'; s, a) \pi_{t+1}(a'; s') Q_{t+1}^\pi(s', a') \end{aligned}$$

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 3.1.15. Let's add the proof for completeness. Recall from Proposition 3.1.13 <sup>4</sup> that  $(S, A, R)$  is a Markov reward process so that by (3.2)

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

<sup>4</sup>at some point rewrite proposition for Markov policies, works equally

Thus, using the formula of total probability,

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

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

$$\begin{aligned}
V_t^\pi(s) &= \sum_{a \in \mathcal{A}_s} \pi_t(a; s) Q_t^\pi(s, a) \\
&= \sum_{a \in \mathcal{A}_s} \pi_t(a; s) \left( r(s, a) + \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi_{t+1}(a'; s') Q_{t+1}^\pi(s', a') \right) \\
&= \sum_{a \in \mathcal{A}_s} \pi_t(a; s) \left( r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^\pi(s') \right).
\end{aligned}$$

□

It is important to note that the system of equations is different from discounted MDP problems. First, the discounting factor disappears as  $\gamma = 1$  and, secondly, the linear systems are actually recursions that gradually simplify the system towards the terminal conditions  $Q_T^\pi \equiv 0$ ,  $V_T^\pi \equiv 0$ . 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 3.4.5.** For any  $t \leq T$  and a given Markov decision problem

- the function  $V_t^* : \mathcal{S} \rightarrow \mathbb{R}$  that takes values

$$V_t^*(s) := \sup_{\pi \in \Pi_t^T} V_t^\pi(s), \quad s \in \mathcal{S},$$

is called optimal time-state value function.

- the function  $Q_t^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  that takes values

$$Q_t^*(s, a) = \sup_{\pi \in \Pi_t^T} Q_t^\pi(s, a), \quad s \in \mathcal{S}, a \in \mathcal{A},$$

is called optimal time-state-action value function.

- a policy  $\pi^*$  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 3.4.6.** The following holds for the optimal time-state value function and the optimal time-state-action value function for any  $s \in \mathcal{S}$ :

- (i)  $V_t^*(s) = \max_{a \in \mathcal{A}_s} Q_t^*(s, a)$  for all  $t < T$ ,
- (ii)  $Q_t^*(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^*(s')$  for all  $t < T$

In particular,  $V^*$  and  $Q^*$  satisfy the following Bellman optimality equations (backwards recursions):

$$V_t^*(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) V_{t+1}^*(s') \right\}, \quad s \in \mathcal{S},$$

and

$$Q_t^*(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q_{t+1}^*(s', a'), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

for all  $t < T$ .

*Proof.* 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  $Q_t^*$ .  $\square$

For discounted infinite time horizon problems we could now show that it is sufficient to consider stationary greedy policies. **The stationarity is in general not true for finite time horizon MDPs.** Consider for example the ice-vendor MDP and assume that we close our store in the winter. Then the amount of ice cream we want to order depends strongly on the time horizon up to the closing date. It is clear that we would like to order with a different strategy if we can sell the ice cream for 6 weeks rather than just for 1 week. Given this observation it is clear that we can no longer restrict the set of policies to stationary policies and it follows also that we have no fixpoint relation in the value function. We can formulate the following theorem:



**Theorem 3.4.7. (Dynamic programming algorithm)**

Suppose  $v_t : \mathcal{S} \rightarrow \mathbb{R}$ ,  $t = 0, \dots, T$ , is a sequence of vectors that fulfill the Bellman optimality equations (backwards recursions)

$$v_t(s) = \begin{cases} 0 & : t = T \\ \max_{a \in \mathcal{A}_s} \left\{ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) v_{t+1}(s') \right\} & : t < T \end{cases},$$

then  $v_t = V_t^*$  for all  $t = 0, \dots, T$ . Similarly, if  $q_t : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ,  $t = 0, \dots, T$ , is a sequence of vectors that fulfill the Bellman state-action optimality equations (backwards recursions)

$$q_t(s, a) = \begin{cases} 0 & : t = T \\ r(s, a) + \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} q_{t+1}(s', a') & : t < T \end{cases},$$

then  $q_t = Q_t^*$  for all  $t = 0, \dots, T$ . Most importantly, an optimal (non-stationary)



policy is given by the greedy policy

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

*Proof.* Suppose  $q$  solves the optimality recursion and  $\pi^q$  is the greedy policy. By uniqueness  $q$  equals  $Q^*$ . It remains to show  $V^* = V^{\pi^q}$ , then  $\pi^q$  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  $\pi^q$ . 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  $\pi_q$  is optimal.  $\square$

The key tool to solve finite time horizon MDPs follows from Theorem 3.4.7 and is called backward induction. If the state-action space is not too large and we have access to the transition probabilities, then the optimal control problem can be solved backwards iteration. Let us recall the ice vendor example on finite time-horizon to get an idea why this is intuitive. In the last timestep, there is no opportunity to sell any more ice cream, the summer season is over. In the optimal situation we obviously have no more stock which corresponds to  $V_T^* \equiv 0$ . Then we go backwards in time step-by-step and consider what is optimal in every possible state. Suppose one day is left. The recursion gives

$$Q_{T-1}^*(s, a) = \underbrace{r(s, a)}_{\text{last days profit}} + \underbrace{\sum_{s' \in \mathcal{S}} p(s'; s, a) V_T^*(s')}_{=0, \text{ no future to be considered}} = r(s, a)$$

and the optimal policy becomes

$$\pi_{T-1}^*(s) = \arg \max_{a \in \mathcal{A}_s} r(s, a).$$

What does this mean? For the last step we need to chose the action that maximises the expected reward without any thought about the future. No surprise! If the expectations are know, nothing needs to be done. If not, this is nothing but the bandit problem! For the next time-step  $T-2$  the recursion gives

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

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 choosing 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

### 3.4.2 Dynamic programming algorithms

In the context of fully available dynamics the dynamic programming algorithms are very simple for finite MDPs. No equations must be solved, only recursion followed backwards must be computed from the terminal condition. Theorem 3.4.7 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:**  $V_t^\pi$  and  $Q_t^\pi$  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**

**end**

**end**

---

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

optimal control algorithm.

---

<sup>5</sup>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  $\pi^*$ 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 4

# 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 and accessible. Most importantly, the transition function  $p$  must be known explicitly. In many situations this is not the case. This sounds weird, on first glance one might think that there is no way to formulate Markov decision problems without knowing the decision problem. Indeed, this is not the case. There might be a 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 repeatedly usable, or even worse, too big to be stored at all. The state space might also be too big to learn about all possible decisions, but perhaps most states and actions are irrelevant.

In both cases there is no way to use 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 4.0.1.** 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 explicitly allows to compute the action values  $Q_a$ . Then one only needs to compare the action values to find the best action.

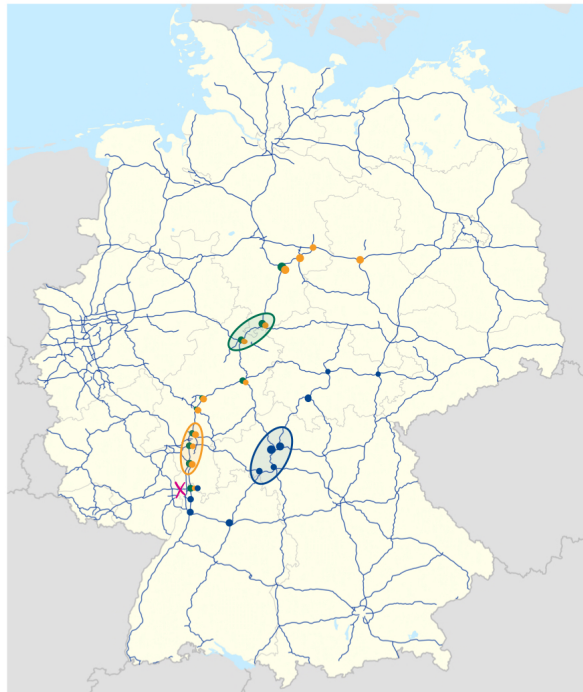
In this chapter we develop simulation based dynamic programming algorithms for the tasks we addressed with the dynamic programming algorithms based on Bellman's equations:

- policy evaluation, i.e. estimate  $V^\pi(s)$  for a given policy  $\pi$ ,
- optimal control, solve the stochastic control problem (i.e. find the optimal  $V^*$ ,  $Q^*$ ,  $\pi^*$ ).

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.

## 4.1 A guiding example

We start the discussion with a guiding example that should provide some intuition why the mathematical concepts introduced below are completely intuitive<sup>1</sup>. 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 fraction 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 Dreieck Funkturm) to Kreuz Mannheim. To compare two navigation systems  $\pi$  and  $\pi'$  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

<sup>1</sup>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.



With the bootstrapping ideas sketched above try to improve decision making (i.e. improve the navigation systems) of the current system using only temporal difference observations. What does this mean? Suppose an estimate  $\hat{Q}$  of the optimal  $Q$ -matrix is given and suppose a new state-action-reward-next state tuple  $(s, a, r, s')$  is given. How could the estimate  $\hat{Q}$  could be improved? It's not completely unlikely that you will develop an idea of what later will be called  $Q$ -learning.

To be honest, this navigation system example is not very realistic. A navigation system developed this way won't be competitive as it does not incorporate live-data which we all know is crucial and why google maps is as strong as it is. Nonetheless, the example captures the main ideas of this chapter.

## 4.2 Monte Carlo policy evaluation and control

The aim of this short section is to get a glimpse on what it means to use Monte Carlo for policy evaluation without any further thoughts. Recall that policy evaluation is important to know the value of a policy but also to perform generalised policy iteration (alternate between policy evaluation and policy improvement). The value function are expectations by definition expectations, thus, the most obvious model-free variant for policy evaluation is Monte Carlo estimate with  $N$  independent trajectories of the MDP.



**Definition 4.2.1.** 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  $\pi$ , then define

$$\hat{V}^\pi(s) = \frac{1}{N} \sum_{i=1}^N \underbrace{\sum_{t=0}^T \gamma^t R_t^i}_{= \hat{V}_i^\pi(s)} \quad (4.1)$$

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.

### 4.2.1 First visit Monte Carlo policy evaluation

Before we proceed note that the time-truncation in the estimate of  $V^\pi$  by (4.1) automatically induces a bias (i.e. the expectation of the estimator  $\hat{V}^\pi(s)$  is different from  $V^\pi(s)$ ). While there might be no way around such a truncation in practice (we cannot simulate an infinite length MDP) the problem does not occur in many situations. For examples most games do not run forever but stop in a terminating state. In that case one might replace  $T$  by  $\infty$  which in the case of termination is equal to a finite sum up to the termination time.



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

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

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 bootstrapping 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  $V^\pi$  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  $Q^\pi$  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.

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.

**Algorithm 14:** First visit Monte Carlo policy evaluation of  $V^\pi$ 


---

**Data:** Policy  $\pi \in \Pi_S$ , initial condition  $\mu$   
**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  $\mu$ .  
    Generate trajectory  $(s_0, a_0, r_0, s_1, \dots)$  started in  $\mu$  until time-horizon  $T$  using policy  $\pi$ .  
    **for**  $t = 0, 1, 2, \dots, T$  **do**  
        **if**  $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$  **then**  
             $v = \sum_{k=t}^T r_k$   
             $V_n(s_t) = \frac{1}{N(s_t)}v + \frac{N(s_t)-1}{N(s_t)}V_{n-1}(s_t)$   
        **end**  
        **else**  
             $V_n(s_t) = V_{n-1}(s_t)$   
        **end**  
         $N(s_t) = N(s_t) + 1$   
    **end**  
    return  $V_n$   
**end**

---



**Theorem 4.2.2.** The first visit Monte Carlo algorithms satisfy the following convergence properties for  $s \in \mathcal{S}$  and  $a \in \mathcal{A}$ :

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

*Proof.* Strong Markov property, law of large numbers. □

How can we achieve the condition of the theorem? There is not much we can manipulate, only the initial condition  $\mu$  and the policy  $\pi$ :

- Chose an initial distribution  $\mu$  (such as uniform) that puts mass on all states, this is called exploring start.
- Chose a policy  $\pi'$  that distributes mass more evenly on all possible actions than the target policy  $\pi$ , this is called policy exploration.

Comparing with stochastic bandits there is a similar exploration vs. exploitation trade-off for first visit Monte Carlo, playing with  $\mu$  and  $\pi$  has different advantages and disadvantages: Forcing more exploration through  $\mu$  and/or  $\pi$  improves the Monte Carlo convergence for states that are less favorable under  $\mu$  and/or  $\pi$  but downgrades convergence for states that are more favorable under  $\mu$  and  $\pi$ . Additionally, an error occurs as the Monte Carlo procedure estimates  $V^{\pi'}$  (resp.  $Q^{\pi'}$ ) instead of  $V^\pi$  (resp.  $Q^\pi$ ). 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  $\pi$  that is currently believed to be best. As for bandits, in practice one will first force more exploration by  $\pi$  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

**Algorithm 15:** First visit Monte Carlo policy evaluation of  $Q^\pi$ 


---

**Data:** Policy  $\pi \in \Pi_S$ , initial condition  $\mu$   
**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  $\mu$   
    Generate trajectory  $(s_0, a_0, r_0, s_1, \dots)$  started in  $\mu$  until time-horizon  $T$  using policy  $\pi$ .  
    **for**  $t = 0, 1, 2, \dots, T$  **do**  
        **if**  $(s_t, a_t) \notin \{(s_0, a_0), (s_1, a_1), \dots, (s_{t-1}, a_{t-1})\}$  **then**  
             $q = \sum_{k=t}^{\infty} r_k$   
             $Q_n(s_t, a_t) = \frac{1}{M(s_t, a_t)} q + \frac{M(s_t, a_t) - 1}{M(s_t, a_t)} Q_{n-1}(s_t, a_t)$   
             $M(s_t, a_t) = M(s_t, a_t) + 1$   
        **end**  
        **else**  
             $Q_n(s_t, a_t) = Q_{n-1}(s_t, a_t)$   
        **end**  
    **end**  
**end**

---



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**.

### 4.2.2 Generalised policy iteration with first visit Monte Carlo estimation

We have introduced an algorithm that can estimate the  $Q$ -values of a given policy only by simulations. Thus, there is a natural model-free procedure to approximate an optimal policy  $\pi^*$  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. 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). 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  $\varepsilon$ -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 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  $\varepsilon$ -greedy policy update and first visit Monte Carlos policy evaluation. Such a discussion should combine the advantage in effort/precision of first visit Monte Carlo policy evaluation with exploitation with the error committed by choosing  $\varepsilon$ -greedy policy update. As for bandits one should expect decreasing  $\varepsilon$  to be advantegous.

**Algorithm 16:** Monte Carlo generalised  $\varepsilon$ -greedy policy iteration

---

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

---



Simulate generalised  $\varepsilon$ -greedy policy iteration with first visit Monte Carlo evaluation for some MDP example with different choices of  $\varepsilon$ , 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<sup>2</sup>.

### 4.3 Asynchronous stochastic fixed point iterations

In order to justify the convergence of approximate dynamic programming algorithms we will use arguments that are mostly due to John Tsitsiklis and coauthors. The idea of the following is simple. Suppose we aim at solving the fixed point equation  $F(x) = x$  using Banachs fixed point iteration but cannot compute  $F(x)$  exactly but only have stochastic approximations  $\widehat{F}(x)$  (such as Monte Carlo estimates of an expectation). Can we still perform Banachs fixed point iteration as  $x(n+1) = \widehat{F}(x(n))$  and converge (almost surely) to the unique fixed point of  $F$ ? The answer is no but the update scheme  $x(n+1) = (1 - \alpha(n))x(n) + \alpha(n)\widehat{F}(s(n))$  converges to  $x^*$  if the estimats are unbiased with well-behaved second moments and the so-called step-sizes  $\alpha$  decay with a suitable rate. Writing Bellman operators with expectations this will readily lead us to different numerical schemes for policy evaluation and optimal control.

We will first discuss in the most simple settings the ideas of Robbins and Monro<sup>3</sup> to find roots of functions that can only be approximated and then turn towards Tsitsiklis' stochastic approximation theorem that serves as the main tool to prove convergence for all approximate dynamic programming algorithms.

#### 4.3.1 Basic stochastic approximation theorem

The simple Robbins Monro algorithm addresses the task to find zeros  $G(x) = 0$  for very particular functions  $G$ . From lectures on numerical analysis different algorithms (such as Newton's method  $x_{n+1} = x_n + \frac{G(x_n)}{G'(x_n)}$ ) might be known to the reader. The situation that Robbins and Monro addressed is different in the sense that they considered functions  $G$  for which there is no access to the true values  $G(x)$  but only to unbiased approximations  $\tilde{G}(x) = G(x) + \varepsilon$ , where  $\varepsilon$  is a mean-zero error. The situation to keep in mind is that of a function  $G$  that is defined as an expectation (such as  $V^\pi(s)$  or  $Q^\pi(s, a)$ ) that can be approximated for instance by Monte Carlo. The most generic Robbins-Monro approximation scheme is

$$x_{n+1} = x_n - \alpha_n y_n, \quad (4.2)$$

where  $\alpha_n$  are step-sizes specified below and  $y_n$  are stochastic approximations of  $G(x_n)$ . Here is a first simple convergence theorem, introducing the first stochastic approximation (SA) algorithm to approximate roots of nice enough functions:

<sup>2</sup>Check here for some results: <https://arxiv.org/pdf/1903.05926.pdf>

<sup>3</sup>H. Robbins and S. Monro: A stochastic approximation method. The Annals of Mathematical Statistics, 22:400–407, 1951.




**Theorem 4.3.1. (Simplest Robbins-Monro algorithm)**

Suppose there is some  $\kappa > 0$  such that  $G(x) - G(y) \geq \kappa(x - y)$  for all  $x, y \in \mathbb{R}$  and the stochastic process  $(x_n)$  is defined by (4.2), where  $y_n$  are stochastic approximations of  $G(x_n)$ :

$$y_n = G(x_n) + \varepsilon_n$$

and the processes are defined on a filtered probability space  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$  carrying all appearing random variables. Assume that there are deterministic step-sizes  $(\alpha_n)$  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 satisfying  $\mathbb{E}[\varepsilon_n | \mathcal{F}_n] = 0$ . Suppose furthermore that  $\sum_{n=0}^{\infty} \mathbb{E}[y_n^2] \alpha_n^2 < \infty$ , then  $x_n \xrightarrow{L^2} x_*$  for  $n \rightarrow \infty$ , where  $G(x_*) = 0$ .

The condition  $\sum_{n=0}^{\infty} \mathbb{E}[y_n^2] \alpha_n^2 < \infty$  is satisfied for instance if  $G$  is bounded and the errors have bounded conditional second moments, a condition that will occur in all theorems below.



The most important sequence satisfying the Robbins-Monro conditions is  $\alpha_n = \frac{1}{n}$ . Other obvious choices are  $\alpha_n = \frac{1}{n^p}$  for all  $\frac{1}{2} < p \leq 1$ .

The assumption on  $G$  is way too strong and the convergence mode weak (only  $L^2$ , thus, in probability), we discuss this simple version as it allows us to give a simple proof. We will later give a much more general version of the Robbins-Monro theorem but need to involve stability theory of ordinary differential equations. Here are two (very similar) classical situations:

- If  $G = F'$ , then the assumption on  $G$  is what is typically called strongly convex of  $F$  and the algorithm converges (in  $L^2$ ) to a critical point of  $F$ . If  $G$  is differentiable then the condition implies  $F'' \geq \kappa$ , a condition that is stronger than convex.
- If  $G$  is defined on  $[a, b]$  then the condition is for instance satisfied if  $G(a) < 0 < G(b)$  and has a unique root  $x_*$  where  $G' \geq \kappa$ .

In fact, the second example shows that Robbins-Monro is not really made for strongly convex functions. The most important situation to keep in mind is the one of functions defined through (unknown) expectations of random variables that can be sampled:

**Example 4.3.2.** Suppose  $G(x) = \mathbb{E}[f(x, Y)]$  for some random variable  $Y$  (and all expectations are finite). If we have access to  $f$  and samples  $\tilde{Y}$  of  $Y$  then we have access to stochastic approximations  $f(x, \tilde{Y}) = G(x) + (f(x, \tilde{Y}) - G(x)) =: G(x) + \varepsilon$  of  $f(x)$ . Keeping in mind that reinforcement learning is about reward functions (expectations) it might be little surprising that stochastic approximation is related to reinforcement learning.

*Proof of simple Robbins-Monro.* The proof mostly relies on a simple recursion result on positive sequences:



Suppose that  $z_n$  is a positive sequence such that  $z_{n+1} \leq (1 - a_n)z_n + c_n$ , where  $a_n, c_n$  are positive sequences satisfying

$$\sum_{n=1}^{\infty} a_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} c_n < \infty.$$

Then  $\lim_{n \rightarrow \infty} z_n = 0$ .

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

$$0 \leq z_n = \underbrace{x_n + \sum_{k=0}^{n-1} c_k}_{\text{converges for } n \rightarrow \infty} - \sum_{k=0}^{n-1} a_k z_k, \quad (4.3)$$

it has to hold true that  $\sum_{k=0}^{\infty} a_k z_k < \infty$ .

By assumption,  $\sum_{n=1}^{\infty} a_n = \infty$  and hence,  $\liminf_{n \rightarrow \infty} z_n = 0$ . Again, by 4.3.1, the existence of  $\lim_{n \rightarrow \infty} z_n$  follows directly. So,

$$\liminf_{n \rightarrow \infty} z_n = \lim_{n \rightarrow \infty} z_n = 0.$$



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

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

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

Now the assumption on  $G$  implies that

$$d_n \geq \kappa \mathbb{E}[(x_n - x_*)(x_n - x_*)] = \kappa z_n,$$

so that in total we derived the positive recursion

$$z_{n+1} \leq z_n(1 - 2\alpha_n \kappa) + \alpha_n^2 e_n.$$

Hence, the claim follows from the first step and shows very clearly the origin of the summability condition on the step sizes.  $\square$

From the probabilistic point of view it is important to note that almost nothing (like independence, adaptivity, etc.) was assumed on the stochastic approximations  $y_n$ , the catch is the weak  $L^2$ -convergence mode. We will later see a version with almost sure convergence that rely on the almost sure (super)martingale convergence theorem. In that case much more needs to be assumed.



Robbins-Monro type algorithms can equally be used to find roots of  $G(x) = a$ , the



corresponding algorithm becomes

$$x_{n+1} = x_n - \alpha_n(y_n - a), \quad (4.4)$$

where  $y$  are again approximations of  $G(x_n)$ .

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

**Example 4.3.3.** Suppose  $Z_1, Z_2, \dots$  is an iid sequence with finite mean  $\mu$ . Then

$$\theta_{n+1} = \theta_n + \frac{1}{n+1}(Z_n - \theta_n)$$

can be solved explicitly as

$$\theta_n = \frac{1}{n}\theta_0 + \frac{1}{n} \sum_{k=0}^{n-1} Z_k$$

and the righthand side converges to  $\mu$  by the law of large numbers. In fact, the convergence is even almost surely and in  $L^2$  if  $Z_i$  have finite variance. This way of writing the law of large numbers can be considered a stochastic approximation iteration to find a zero for  $G(\theta) = \mu - \theta$  with the Robbins-Monro iteration with step-sizes  $\alpha_n = \frac{1}{n+1}$  and unbiased approximations  $Z_n - \theta_n$  of  $G(\theta_n)$ . Keeping in mind that the convergence in the law of large numbers is very slow (order  $\sqrt{n}$ ) the example gives a first hint that stochastic approximation is a robust but slow algorithm.

### 4.3.2 Asynchronous stochastic approximation

We will now develop the main tools to prove convergence of Q-learning by proving a much more interesting version of the Robbins-Monro algorithm that results in almost sure convergence to fixedpoints. The proof is very interesting as it makes very transparent how the learning algorithms can be modified and still converge to the optimal state-action function.

The asynchronous stochastic approximation theorem that is needed to prove convergence of Q-learning must be an almost sure convergence theorem to give a useful convergence statement of Q-learning. Comparing with the proof of the simple Robbins-Monro theorem a stochastic version (with almost sure convergence) of the first step of the proof is needed. Formulated in the right way this is a consequence of the almost sure (super)martingale convergence theorem:



**Theorem 4.3.4. (Robbins-Siegmund Theorem)**

Let  $(\Omega, \mathcal{A}, \mathcal{F}, \mathbb{P})$  be a filtered probability space,  $(Z_k)_{k \in \mathbb{N}}$ ,  $(A_k)_{k \in \mathbb{N}}$ ,  $(B_k)_{k \in \mathbb{N}}$  and  $(C_k)_{k \in \mathbb{N}}$  be non-negative and  $\mathcal{F}$ -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

$$\mathbb{E}[Z_{k+1} \mid \mathcal{F}_k] \leq Z_k(1 + A_k) + B_k - C_k.$$

Then

1. there exists an almost surely finite random variable  $Z_\infty$  such that  $Z_k \rightarrow Z_\infty$  almost surely for  $k \rightarrow \infty$ ,
2. it holds true that  $\sum_{k=0}^{\infty} C_k < \infty$  almost surely.

*Proof.* The proof<sup>4</sup> 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  $(M_k)_{k \in \mathbb{N}}$ .

We define the auxiliary random variables

$$\widehat{Z}_k = \frac{Z_k}{\prod_{i=0}^{k-1} (1 + A_i)}, \quad \widehat{B}_k = \frac{B_k}{\prod_{i=0}^k (1 + A_i)}, \quad \widehat{C}_k = \frac{C_k}{\prod_{i=0}^k (1 + A_i)}$$

and observe that

$$\begin{aligned} \mathbb{E}[\widehat{Z}_{k+1} | \mathcal{F}_k] &= \left( \prod_{i=0}^{k-1} (1 + A_i)^{-1} \right) \mathbb{E}[Z_{k+1} | \mathcal{F}_k] = \left( \prod_{i=0}^{k-1} (1 + A_i)^{-1} \right) (Z_k(1 + A_k) + B_k - C_k) \\ &= \widehat{Z}_k + \widehat{B}_k - \widehat{C}_k. \end{aligned} \quad (4.5)$$

Our candidate for the supermartingale is

$$M_k = \widehat{Z}_k - \sum_{i=0}^{k-1} (\widehat{B}_i - \widehat{C}_i),$$

for which we observe

$$\begin{aligned} \mathbb{E}[M_{k+1} | \mathcal{F}_k] &= \mathbb{E}[\widehat{Z}_{k+1} | \mathcal{F}_k] - \sum_{i=0}^k (\mathbb{E}[\widehat{B}_i | \mathcal{F}_k] - \mathbb{E}[\widehat{C}_i | \mathcal{F}_k]) \leq \widehat{Z}_k + \widehat{B}_k - \widehat{C}_k - \sum_{i=0}^k (\widehat{B}_i - \widehat{C}_i) \\ &= \widehat{Z}_k - \sum_{i=0}^{k-1} (\widehat{B}_i - \widehat{C}_i) = M_k, \end{aligned}$$

where we have used (4.5) and that  $\widehat{B}_i, \widehat{C}_i$  are  $\mathcal{F}_k$ -measurable for  $i \leq k$ . In order to apply Doob's martingale convergence theorem, we need to verify  $\sup_{k \in \mathbb{N}} \mathbb{E}[M_k^-] < \infty$ . Since in general, it is not obvious that this property will hold, we introduce a localization.



Localization: We define the stopping time  $\tau_\varepsilon = \inf\{k \geq 1 : \sum_{i=0}^k \widehat{B}_i > \varepsilon\}$  for  $\varepsilon > 0$ . Show that there exists an integrable random variable  $M_\infty^\varepsilon$  with  $\lim_{k \rightarrow \infty} M_{k \wedge \tau_\varepsilon} = M_\infty^\varepsilon$  almost surely.

Since  $(B_k)_{k \in \mathbb{N}}$  is  $\mathcal{F}$ -adapted,  $\tau_\varepsilon$  is a stopping time with respect to  $\mathcal{F}$ . Moreover,  $(M_{k \wedge \tau_\varepsilon})_{k \in \mathbb{N}}$  is still a supermartingale, and additionally satisfies

$$M_{k \wedge \tau_\varepsilon} = \widehat{Z}_{k \wedge \tau_\varepsilon} - \sum_{i=0}^{(k \wedge \tau_\varepsilon)-1} \widehat{B}_i + \sum_{i=0}^{(k \wedge \tau_\varepsilon)-1} \widehat{C}_i \geq - \sum_{i=0}^{(k \wedge \tau_\varepsilon)-1} \widehat{B}_i \geq -\varepsilon,$$

since  $\sum_{i=0}^{(k \wedge \tau_\varepsilon)-1} \widehat{B}_i \leq \varepsilon$  by construction of the stopping time  $\tau_\varepsilon$ . Since  $(M_{k \wedge \tau_\varepsilon})_{k \in \mathbb{N}}$  is uniformly bounded from below (and due to the monotonic decrease of the expectation for supermartingales) we obtain

$$\sup_{k \in \mathbb{N}} \mathbb{E}[|M_{k \wedge \tau_\varepsilon}|] < \infty.$$

We are now ready to apply Doob's Martingale convergence theorem to find an integrable random variable  $M_\infty^\varepsilon$  with  $\lim_{k \rightarrow \infty} M_{k \wedge \tau_\varepsilon} = M_\infty^\varepsilon$  almost surely. Next, we have to remove the stopping time.

<sup>4</sup>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.



Remove localization: Show that  $\lim_{k \rightarrow \infty} M_k(\omega) < \infty$  almost surely.

Let  $(\varepsilon_n)_{n \in \mathbb{N}}$  be an increasing sequence with  $\lim_{n \rightarrow \infty} \varepsilon_n = \infty$ . First note, that for each  $n \in \mathbb{N}$  we have

$$\lim_{k \rightarrow \infty} M_{k \wedge \tau_{\varepsilon_n}}(\omega) = M_{\infty}^{\varepsilon_n}(\omega)$$

for almost all  $\omega \in \Omega$ . We observe that for each  $\omega \in \Omega$  with  $\sum_{i=0}^{\infty} \widehat{B}_i(\omega) < \infty$  there exists  $N \in \mathbb{N}$  such that  $\omega \in \{\tau_{\varepsilon_N} = \infty\}$ , i.e. for this  $\omega$  it holds

$$M_{k \wedge \tau_{\varepsilon_N}}(\omega) = M_k(\omega)$$

for all  $k \in \mathbb{N}$ , but similarly

$$\lim_{k \rightarrow \infty} M_k(\omega) = \lim_{k \rightarrow \infty} M_{k \wedge \tau_{\varepsilon_N}}(\omega) = M_{\infty}^{\tau_{\varepsilon_N}}(\omega) < \infty,$$

where the last inequality  $< \infty$  holds since  $\mathbb{E}[|M_{\infty}^{\tau_{\varepsilon_N}}|] < \infty$ .



Conclusion:

Finally, we move back to the assertion regarding  $(Z_k)_{k \in \mathbb{N}}$  and  $(C_k)_{k \in \mathbb{N}}$ . Observe that

$$-\infty < -\sum_{i=0}^{\infty} \widehat{B}_i(\omega) \leq \lim_{k \rightarrow \infty} M_k(\omega) = \lim_{k \rightarrow \infty} \widehat{Z}_k(\omega) - \sum_{i=0}^{k-1} (\widehat{B}_i(\omega) - \widehat{C}_i(\omega)) < \infty,$$

where  $\widehat{Z}_k(\omega), \widehat{B}_i(\omega), \widehat{C}_i(\omega) \geq 0$  implying that

$$\lim_{k \rightarrow \infty} \widehat{Z}_k(\omega) < \infty \quad \text{and} \quad \sum_{i=0}^{\infty} \widehat{C}_i(\omega) < \infty$$

for almost all  $\omega \in \Omega$ . Moreover, it holds true that

$$Z_k(\omega) = \widehat{Z}_k(\omega) \prod_{i=0}^{k-1} (1 + A_i(\omega)),$$

where both  $\widehat{Z}_k(\omega)$  and  $\prod_{i=0}^{k-1} (1 + A_i(\omega))$  converge for almost all  $\omega \in \Omega$ . The latter one follows by monotonicity and

$$0 \leq \prod_{i=0}^{k-1} (1 + A_i(\omega)) \leq \exp\left(\sum_{i=0}^{k-1} A_i(\omega)\right),$$

where the upper bound converges by assumption. Therefore,  $\lim_{k \rightarrow \infty} Z_k(\omega) = Z_{\infty}(\omega)$  exists for almost all  $\omega \in \Omega$ . Similarly, we have

$$\sum_{i=0}^k C_i(\omega) = \sum_{i=0}^k \widehat{C}_i(\omega) \prod_{j=0}^i (1 + A_j(\omega)) \leq \left( \prod_{j=0}^{\infty} (1 + A_j(\omega)) \right) \sum_{i=0}^k \widehat{C}_i(\omega)$$

which implies

$$\sum_{i=0}^{\infty} C_i(\omega) < \infty$$

for almost all  $\omega \in \Omega$ . □

Next we will show a useful extension of Robbins-Siegmund Theorem.



**Corollary 4.3.5.** Suppose  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$  is a filtered probability space carrying a non-negative adapted stochastic process  $(Z_n)$ . If

$$\mathbb{E}[Z_{n+1} | \mathcal{F}_n] \leq (1 - a_n + b_n)Z_n + c_n \quad (4.6)$$

for some non-negative  $(\mathcal{F}_n)$ -adaptive random variables  $a_n, b_n, c_n$  such that almost surely

$$\sum_{n=1}^{\infty} a_n = \infty, \quad \sum_{n=1}^{\infty} b_n < \infty, \quad \text{and} \quad \sum_{n=1}^{\infty} c_n < \infty,$$

then  $\lim_{n \rightarrow \infty} Z_n = 0$  almost surely.

*Proof.* We define  $A_n = b_n$ ,  $B_n = c_n$  and  $C_n = Z_n a_n$ . Then,

$$\mathbb{E}[Z_{n+1} | \mathcal{F}_n] \leq (1 + A_n)Z_n + B_n - C_n$$

with

$$\sum_{n=0}^{\infty} A_n < \infty \quad \text{and} \quad \sum_{n=0}^{\infty} B_n < \infty.$$

By Robbins-Siegmund Theorem we obtain that  $Z_n \rightarrow Z_\infty$  for  $n \rightarrow \infty$  almost surely and that  $\sum_{n=0}^{\infty} Z_n a_n < \infty$  almost surely.

As  $\sum_{n=0}^{\infty} a_n = \infty$  it follows that  $\liminf_{n \rightarrow \infty} Z_n = 0$ . As  $Z_n$  is convergent we obtain  $Z_\infty = \lim_{n \rightarrow \infty} Z_n = \liminf_{n \rightarrow \infty} Z_n = 0$  almost surely.  $\square$



**Lemma 4.3.6.** Suppose  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_t))$  is a filtered probability space carrying all appearing random variables. Suppose

- $\varepsilon(t)$  are  $\mathcal{F}_{t+1}$ -measurable with  $\mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0$  and  $\mathbb{E}[\varepsilon^2(t) | \mathcal{F}_t] \leq C$  for all  $t \in \mathbb{N}$ ,
- the step-sizes  $\alpha(t)$  are adapted with

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty$$

almost surely.

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

$$W(t+1) = (1 - \alpha(t))W(t) + \alpha(t)\varepsilon(t), \quad t \in \mathbb{N},$$

converges to 0 almost surely.

*Proof.* We are going to apply the corollary of Robbins-Siegmund theorem 4.3.5 to the sequence  $W^2$ :

$$\begin{aligned} \mathbb{E}[W(t+1)^2 | \mathcal{F}_t] &= \mathbb{E}[(1 - \alpha(t))^2 W^2(t) + \alpha^2(t)\varepsilon^2(t) + 2\alpha(t)(1 - \alpha(t))W(t)\varepsilon(t) | \mathcal{F}_t] \\ &= (1 - 2\alpha(t) + \alpha^2(t))W^2(t) + \mathbb{E}[\alpha^2(t)\varepsilon^2(t) + 2\alpha(t)(1 - \alpha(t))W(t)\varepsilon(t) | \mathcal{F}_t] \\ &\leq (1 - a_t + b_t)W^2(t) + c_t, \end{aligned}$$

with  $a_t = -2\alpha(t)$ ,  $b_t = \alpha^2(t)$ , and  $c_t = \alpha^2(t)C$ . Now the claim follows from the Robbins-Siegmund corollary.  $\square$

Note that assuming bounded conditional second moments makes our lives much easier, the theorem also holds under much less restricted assumption on the error. The condition is very strong and typically not satisfied for applications of stochastic approximation. Nonetheless, for convergence of Q-learning and TD-algorithms the assumption will turn out to be enough.



**Lemma 4.3.7.** Suppose  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_t))$  is a filtered probability space carrying all appearing random variables. Suppose the step-sizes  $\alpha(t)$  are adapted with  $\sum_{t=1}^{\infty} \alpha(t) = \infty$  almost surely. If  $A$  is a non-negative constant, then the non-negative stochastic process  $Y$  defined by some  $\mathcal{F}_0$ -measurable initial condition  $Y(0) \geq 0$  and the recursion

$$Y(t+1) = (1 - \alpha(t))Y(t) + \alpha(t)A, \quad t \in \mathbb{N},$$

converges to  $A$  almost surely as  $t \rightarrow \infty$ .

*Proof.* The non-negativity of  $Y$  follows from the recursive definition. Subtracting and iterating yields

$$Y(t+1) - A = (1 - \alpha(t))(Y(t) - A) = \dots = \prod_{s=1}^t (1 - \alpha_s)(Y(0) - A).$$

The right hand side vanishes almost surely in the limit because  $\sum_{t=1}^{\infty} \alpha_t = \infty$  almost surely:

$$\log \left( \prod_{s=1}^t (1 - \alpha(s)) \right) = \sum_{s=1}^t \log(1 - \alpha(s)) \leq \sum_{s=1}^t -\alpha(s) \rightarrow -\infty, \quad t \rightarrow \infty.$$

□

Before proceeding with asynchronous stochastic approximation let us think for a moment about the previous two lemmas. The formulation is extremely robust, all random variables are only assumed to be measurable, there is no more precise assumption. Also it is important to note that the recursive algorithms are very explicit. For example, there is no problem looking at the processes started at times (random or deterministic) different from 0, the convergence will still hold. In the proofs we will for instance use the notation  $(W(t : t_0))_{t \geq t_0}$  for the stochastic process with a given initial condition  $W(t_0 : t_0)$ , similarly for  $Y$ , at time  $t_0$  and recursive update

$$W(t+1 : t_0) = (1 - \alpha(t))W(t : t_0) + \alpha(t)\varepsilon(t), \quad t \geq t_0. \quad (4.7)$$

It obviously holds that  $W = W(\cdot : 0)$ . The convergence property of  $W(\cdot : t_0)$  is identical to that of  $W(\cdot : 0)$ . Since different  $W(\cdot : t_0)$  are defined through the same error variables these processes can be compared on a pathwise level („coupled“), a feature that will be crucial to prove asynchronous stochastic approximation below. Here is a first useful property:



**Lemma 4.3.8.** In the setting of Lemma 4.3.6 we consider for all  $t_0 \in \mathbb{N}$  the stochastic processes  $(W(t : t_0))_{t \geq t_0}$  from (4.7) with initial conditions  $W(t_0 : t_0) = 0$ . For every  $\delta > 0$  there exists some random variable  $T$  with values in  $\mathbb{N}$  such that  $|W(t : t_0)| < \delta$  for all  $T \leq t_0 \leq t$  almost surely.

*Proof.* From Lemma 4.3.6 we know that  $\lim_{t \rightarrow \infty} W(t : 0) = 0$  almost surely. Next, we note that

$$\begin{aligned}
W(t : 0) &= (1 - \alpha(t - 1))W(t - 1 : 0) + \alpha(t - 1)\varepsilon'(t - 1) \\
&= (1 - \alpha(t - 1))(1 - \alpha(t - 2))W(t - 2 : 0) + \alpha(t - 2)\varepsilon(t - 2) + \alpha(t - 1)\varepsilon(t - 1) \\
&= \dots \\
&= \prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : 0) + \sum_{s=t_0}^{t-1} \alpha(s) \prod_{k=s}^{t-2} (1 - \alpha(k))\varepsilon(s) \\
&= \prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : 0) + \underbrace{\prod_{s=t_0}^{t-1} (1 - \alpha(s))W(t_0 : t_0)}_{=0} + \sum_{s=t_0}^{t-1} \alpha(s) \prod_{k=s}^{t-2} (1 - \alpha(k))\varepsilon(s) \\
&= \underbrace{\prod_{s=t_0}^{t-1} (1 - \alpha(s))}_{\leq 1} W(t_0 : 0) + W(t : t_0)
\end{aligned}$$

holds for all  $t > t_0$ . Hence,  $|W(t : t_0)| \leq |W(t : 0)| + |W(t_0 : 0)|$  almost surely, i.e.  $W(t : t_0)$  can be controlled by  $W(t : 0)$ . Since for every  $\omega \in \Omega$ ,  $W(t : 0)(\omega)$  vanishes in the limit we can chose  $T(\omega)$  large enough so that  $|W(t : 0)(\omega)| < \frac{\delta}{2}$  for all  $t \geq T(\omega)$ . This implies the claim.  $\square$

For didactic reasons we next prove a theorem on fixed points of real-valued contractions. In a way the theorem is an approximate version of Banachs fixedpoint iteration set up for situations in which the contraction operator cannot be computed explicitly but only with an error. The theorem is not particularly interesting itself (do you know any interesting contraction on  $\mathbb{R}$ ?), but it will become much more interesting when extended to  $(\mathbb{R}^d, \|\cdot\|_\infty)$  to prove convergence of temporal difference schemes.



**Theorem 4.3.9. (Stochastic fixed point iteration for contractions on  $\mathbb{R}$ )**

Suppose  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$  is a filtered probability space on which all appearing random variables are defined. Suppose that

- $F : \mathbb{R} \rightarrow \mathbb{R}$  is a contraction, i.e. there is some  $\beta < 1$  such that  $|F(x) - F(y)| \leq \beta|x - y|$  for all  $x, y \in \mathbb{R}$ ,
- $\varepsilon(t)$  are  $\mathcal{F}_{t+1}$ -measurable with  $\mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0$  and  $\mathbb{E}[\varepsilon^2(t) | \mathcal{F}_t] \leq C$  for all  $t \in \mathbb{N}$ ,
- the step-sizes  $\alpha(t)$  are only (!) adapted with

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty.$$

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

$$x(t + 1) = x(t) + \alpha(t)(F(x(t)) + \varepsilon(t) - x(t)), \quad t \in \mathbb{N},$$

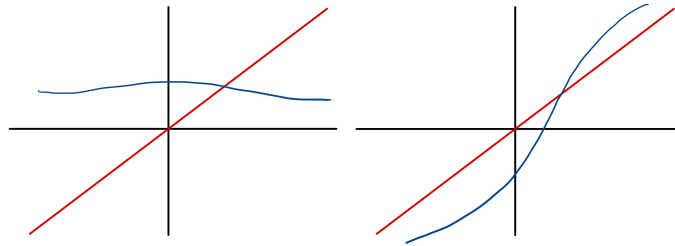
converges almost surely to the unique fixed point  $x^*$  of  $F$ .

Before going into the proof let us quickly motivate why we call the iterative scheme a stochastic fixed point iteration. For known  $F$  a Banach's fixed point iteration with approximated  $F$  looks like

$$x(t + 1) = F(x(t)) = x(t) + (F(x(t)) - x(t)) \approx x(t) + (F(x(t)) + \varepsilon(t) - x(t)).$$



Little surprisingly the scheme would not converge as the errors are not assumed to diminish and the scheme would fluctuate around  $x^*$  without converging. This is circumvented by decreasing the update size using  $\alpha(t)$ . Knowing other approximation schemes that find zeros or fixed points of real-valued functions one might ask why the scheme is so simple. For instance, there are no derivatives in the update scheme. The reason is the particularly simple class of functions. Real-valued contractions are simple functions, the bisecting line can never be crossed bottom-up, it must be crossed downwards (see the drawing).



contraction vs. non-contraction

Hence, if  $x > x^*$ , then  $F(x)$  is below the bisecting line implying  $F(x) - x < 0$ . Similarly,  $F(x) - x > 0$  if  $x < x^*$ . Thus, the scheme is set up such that  $x(t)$  is pushed left if  $x(t) > x^*$  and pushed right if  $x(t) < x^*$ . This little graphic discussion shows why fixed points of contractions (even in the approximate case) can be obtained using very simple approximation schemes that do not involve anything but (approximate) evaluations of the function.

*Proof.* The main idea of the proof is to compare the stochastic processes  $x$  to the stochastic processes  $W$  and  $Y$  from the previous lemmas. These processes are much simpler and we already know their almost sure limits.



Without loss of generality we can assume that  $F(x^*) = x^* = 0$ .

<sup>5</sup> This follows by defining the auxiliary function  $\tilde{F}(x) = F(x + x^*) - x^*$  which has fixed point 0. Suppose the statement holds for  $\tilde{F}$ . Then, subtracting  $x_*$  on both sides of the recursion yields

$$x(t + 1) - x^* = x_i(t) - x^* + \alpha(t)((\tilde{F}(x(t) - x^*) - x^*)) - (x(t) + \varepsilon(t) - x^*), \quad t \in \mathbb{N},$$

so that  $\tilde{x}(t) := x(t) - x^*$  satisfies the original recursion with  $x^* = 0$ . Since convergence of the shifted recursion to 0 is equivalent to convergence of the original recursion to  $x^*$  we may assume  $x^* = 0$ . Assuming  $x^* = 0$  is useful as we can use the estimate

$$|F(x)| = |F(x) - 0| = |F(x) - F(x^*)| \leq \beta|x - x^*| = \beta|x|$$

below.



$\sup_{t \geq 0} |x(t)|$  is finite almost surely, say by the (random) constants  $D_0$ .

<sup>6</sup> First chose  $\eta > 0$  such that  $\beta(1 + \eta) = 1$  and define  $M(t) := \max\{1, \max_{s \leq t} |x(s)|\}$  as well as  $\bar{\varepsilon}(t) = \frac{\varepsilon(t)}{M(t)}$ . Then  $\bar{\varepsilon}(t)$  is again  $\mathcal{F}_{t+1}$ -measurable with

$$\mathbb{E}[\bar{\varepsilon}(t) | \mathcal{F}_t] = \frac{1}{M_t} \mathbb{E}[\varepsilon(t) | \mathcal{F}_t] = 0 \quad \text{and} \quad \mathbb{E}[\bar{\varepsilon}^2(t) | \mathcal{F}_t] \leq \frac{C}{M_t} < C.$$

Hence,  $\bar{\varepsilon}$  is again an error sequence to which the above lemmas apply. Now let  $\bar{W}(\cdot : t_0)$  the recursively defined process from Lemma 4.3.8 started in  $\bar{W}(t_0 : t_0) = 0$ . Since  $\lim_{t \rightarrow \infty} \bar{W}(t :$

<sup>5</sup> nochmal checken

<sup>6</sup> da passt ein  $t_0$  noch nicht

$t_0) = 0$  almost surely there is some  $t_0$  such that  $|\bar{W}(t : t_0)| < \eta$  for all  $t \geq t_0$ . We are next going to show that

$$|x(t)| \leq M(t_0) + \bar{W}(t : t_0)M(t_0) \leq (1 + \eta)M(t_0), \quad t \geq t_0, \quad (4.8)$$

holds almost surely. By the choice of  $t_0$  we only need to prove the first inequality, which we will achieve by induction in  $t \geq t_0$ .

**Induction start:** The inequality clearly holds for  $t = t_0$  as  $\bar{W}(t_0 : t_0) = 0$  by definition and  $x(t) \leq M(t)$  holds for every  $t \in \mathbb{N}$ .

**Induction step:** For the induction step we will use the recursion to transfer the inequality from  $t$  to  $t + 1$ :

$$\begin{aligned} x(t+1) &= (1 - \alpha(t))x(t) + \alpha(t)F(x(t)) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{contraction}}{\leq} (1 - \alpha(t))x(t) + \alpha(t)\beta|x(t)| + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{induction}}{\leq} (1 - \alpha(t))(M(t_0) + \bar{W}(t : t_0)M(t_0)) + \alpha(t)\beta(1 + \eta)M(t_0) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{rearranging}}{=} ((1 - \alpha(t))\bar{W}(t : t_0) + \alpha(t)\varepsilon(t))M(t_0) + (1 - \alpha(t))M(t_0) + \alpha(t)\underbrace{\beta(1 + \eta)}_{=1}M(t_0) \\ &= \bar{W}(t+1 : t_0)M(t_0) + M(t_0) \leq (1 + \eta)M(t_0). \end{aligned}$$

That's it. Since  $(x_t)_{t \leq t_0}$  is clearly bounded the upper bound for  $t \geq t_0$  from (4.8) shows that  $x$  is bounded.

For the next step fix some  $\varepsilon > 0$  such that  $\beta(1 + 2\varepsilon) < 1$  and define recursively  $D_{k+1} = \beta(1 + 2\varepsilon)D_k$  so that  $\lim_{k \rightarrow \infty} D_k = 0$ .



There is a (random) sequence  $t_k \rightarrow \infty$  such that  $\sup_{t \geq t_k} |x(t)| \leq D_k$  almost surely.

Note that  $\lim_{k \rightarrow \infty} D_k = 0$ , hence, the proof of the theorem is complete once the claim is proved. The proof of this claim is carried out by induction.

**Induction start:** Set  $t_0 = 0$ , then the claim holds as  $D_0$  is the global upper bound of  $x$ .

**Induction step:** Suppose  $t_k$  is given. We now use the auxiliary processes from the lemmas above, but started at different times than time 0. The lemmas did not use anything but measurability and equally apply to changed starting times. They also apply to random starting times using the same error functions in the recursions. Let  $W(\cdot : s)$  the process from Lemma 4.3.6 started at time  $s$  in 0 and

$$\tau := \min \{s \geq t_k : W(t : s) < \beta\varepsilon D_k \forall t \geq s\}.$$

Note that  $\tau < \infty$  as  $\lim_{t \rightarrow \infty} W(t : s) = 0$  almost surely. Furthermore, denote by  $Y(\cdot : \tau)$  the process from Lemma 4.3.7 with  $A = \beta D_k$  but started at time  $\tau$  at  $D_k$ . The crucial inequality of the proof is the following:

$$|x(t) - W(t : \tau)| \leq Y(t : \tau), \quad \forall t \geq \tau, \quad (4.9)$$

where  $Y(\cdot : \tau)$  runs for  $t \geq \tau$  and is started in  $Y(\tau : \tau) = D_k$ . This again is proved by induction over  $t \geq \tau$ . The induction start  $t = \tau$  holds by definition of  $\tau$  because  $W(\tau, \tau) = 0$ ,  $Y(\tau : \tau) = D_k$ , and  $|x(t)| \leq D_k$  for all  $t \geq t_k$ . Next, suppose the estimate holds for some  $t \geq \tau$ . We use the defining recursions and the contraction property of  $F$  to prove the estimate for  $t + 1$ :

$$\begin{aligned} x(t+1) &= (1 - \alpha(t))x(t) + \alpha(t)F(x(t)) + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{Ind., contraction}}{\leq} (1 - \alpha(t))(Y(t : \tau) + W(t : \tau)) + \alpha(t)\beta|x(t)| + \alpha(t)\varepsilon(t) \\ &\stackrel{t \geq t_k}{\leq} (1 - \alpha(t))(Y(t : \tau) + W(t : \tau)) + \alpha(t)\beta D_k + \alpha(t)\varepsilon(t) \\ &\stackrel{\text{rearranging}}{=} (1 - \alpha(t))Y(t : \tau) + \alpha(t)\beta D_k + (1 - \alpha(t))W(t : \tau) + \alpha(t)\varepsilon(t) \\ &= Y(t+1 : \tau) + W(t+1 : \tau) \end{aligned}$$

In the same way we can show  $-Y(t+1 : \tau) + W(t+1 : \tau) \leq x(t+1)$ . This proves (4.9).

The reverse triangle inequality ( $|a| - |b| \leq |a - b|$ ) applied to (4.9) yields

$$|x(t)| \leq Y(t : \tau) + |W(t : \tau)|, \quad \forall t \geq \tau.$$

Since  $W(t : \tau) \leq \beta \varepsilon D_k$  for all  $t \geq \tau$  and  $Y(\cdot : \tau)$  converges to  $\beta D_k$  there is some index  $t_{k+1} > \tau \geq t_k$  for which  $|x(t)| < \beta \varepsilon D_k + (1 + \varepsilon) \beta D_k = D_{k+1}$  for all  $t \geq t_{k+1}$ . This proves the claim and the proof is complete.  $\square$

We continue with a proof of the main result of interest, the asynchronous stochastic approximation theorem from which convergences of temporal difference schemes follows readily. The wording asynchronous comes from the fact that the step-sizes  $\alpha_i$  do not need to be the same, the coupling between the equations only comes through  $F$ .



**Theorem 4.3.10. (Asynchronous stochastic fixed point iteration for  $\|\cdot\|_\infty$ -contractions on  $\mathbb{R}^d$ )**

Suppose  $(\Omega, \mathcal{A}, \mathbb{P}, (\mathcal{F}_n))$  is a filtered probability space on which all appearing random variables are defined. Suppose that

- $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$  is a contraction with respect to  $\|\cdot\|_\infty$ , i.e.

$$\|F(x) - F(y)\|_\infty \leq \beta \|x - y\|_\infty, \quad \forall x, y \in \mathbb{R}^d,$$

for some  $\beta \in (0, 1)$ ,

- $\varepsilon_i(t)$  are  $\mathcal{F}_{t+1}$ -measurable with  $\mathbb{E}[\varepsilon_i(t) | \mathcal{F}_t] = 0$  and there is some  $C > 0$  with  $\sup_{i,t} \mathbb{E}[\varepsilon_i^2(t) | \mathcal{F}_t] < C$ ,
- the step-sizes  $\alpha_i(t)$  are adapted with

$$\sum_{t=1}^{\infty} \alpha_i(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_i^2(t) < \infty.$$

Then for any  $\mathcal{F}_0$ -measurable initial condition  $x(0)$  the (asynchronous) stochastic approximation scheme

$$x_i(t+1) = x_i(t) + \alpha_i(t)(F_i(x(t)) - x_i(t) + \varepsilon_i(t)), \quad t \in \mathbb{N},$$

converges almost surely to the unique fixed point  $x^* \in \mathbb{R}^d$  of  $F$ .

*Proof.* The main point is to realise that the proof of the real-valued case can be extended analogously to the multi-dimensional situation. One only needs to be careful where the fixed point property of  $F$  (the coordinates are not necessarily contractions) is used and where the coupling of the equations through  $F$  appears. The first is surprisingly simple, the fixed point property was hardly used in the previous proof. It is only used to shift  $x^*$  to 0 and to ensure  $F$  growth at most linearly, thus, the recursion is bounded.



Without loss of generality we can assume that  $F(x^*) = x^* = 0$ .

The argument is exactly the same as in the previous proof.



$\sup_{t \geq 0} |x(t)|$  is finite almost surely.

The argument is exactly the same using the same  $\eta, \beta$ , and  $M$ . Then one defines the modified errors  $\bar{\varepsilon}_i = \frac{\varepsilon_i}{M}$  and from this the recursions  $W_i$ . The contraction with respect to the supremum

norm is used to estimate

$$|F_i(x(t))| = |F_i(x(t)) - F_i(x^*)| \leq \beta \|x(t) - x^*\|_\infty = \beta \|x(t)\|_\infty \leq \beta M_t,$$

allowing us to estimate in the same way

$$x_i(t+1) \leq W_i(t+1 : t_0)M(t_0) + M(t_0) \leq (1+\eta)M(t_0).$$

Hence, every coordinate process  $x_i$  is bounded, thus, also  $x$  is bounded.

Comparing with the rest of the previous proof  $F$  can be removed in the same way, only the linear growth is used to reduce to coordinate wise auxiliary processes  $W_i$  and  $Y_i$  and the argument works equally.



Go through the one-dimensional proof to check that there is a sequence  $t_k \rightarrow \infty$  such that  $\sup_{t \geq t_k} |x(t)| \leq D_k$  almost surely and  $\lim_{k \rightarrow \infty} D_k = 0$ .

Again, the argument is line by line the same for each coordinate  $x_i$ . Only  $\tau$  needs to be defined as

$$\tau := \min \{s : W_i(t : s) < \beta \varepsilon D_k \forall i = 1, \dots, d, t \geq s\}$$

to allow the same estimate for each  $x_i$ . □

The theorem is called asynchronous because the step-sizes  $\alpha_i(t)$  can be different for different coordinates in contrast to the synchronous case where they are equal. Most importantly, if only one coordinate of  $\alpha(t)$  is non-zero than only that coordinate is updated in step  $t$ . Such extreme case will be called totally asynchronous and we have already crossed such an update scheme in Algorithm 8 in which coordinates were updated by passing through one coordinate after the other.



We have only proved convergence of stochastic fixed point iterations. The proofs do not contain any hint on the quality of convergence. Indeed, stochastic approximation schemes converge terribly slow which is not surprising given the law of large number is a special case (Example (4.3.3)). We only keep in mind that errors with small variance will typically lead to faster (but still slow) convergence.

For reinforcement learning the setting of Example 4.3.2 will become relevant because Bellman operators can be written as

$$F_i(x) = \mathbb{E}_i[f(x, Z)] \tag{4.10}$$

so that approximations of the expectations using samples  $\tilde{Z}_i$  yield model-free learning (approximation) algorithms of the form

$$x_i(t+1) = x_i(t) + \alpha_i(t)(f(x(t), \tilde{Z}_i) - x_i(t)) \tag{4.11}$$

that converge almost surely because they can be rewritten as

$$x_i(t+1) = x_i(t) + \alpha_i(t)(F_i(x(t)) - x_i(t) + \varepsilon_i(t))$$

with the unbiased error terms  $\varepsilon_i(t) := f(x(t), \tilde{Z}_i) - F_i(x(t))$ . The algorithms will be asynchronous through the choice of  $\alpha$  that determines what coordinate to update and how strong the update effect should be. Using different fixed point equations (for  $V^\pi, Q^\pi, V^*, Q^*$ ) and different representations of  $F$  as expectation yields different algorithms with different advantages/disadvantages.



Many reinforcement learning algorithms are random versions of Banach's fixed point iteration to solve Bellman equations. The algorithms are asynchronous in the sense that the update of coordinates is not synchronous (i.e. all step-sizes  $\alpha_i$  are equal). Typically one is interested in the situation in which the vector  $\alpha_t$  is non-zero for exactly one entry. In this situation the vector  $x(t+1)$  gets an update in one coordinate only. We refer to this situation as **totally asynchronous** and every asynchronous algorithm needs to clarify where to update and what step-sizes to chose. The asynchronous choice of  $\alpha$  allows to balance the exploration of new states or state-action pairs. The second Robbins-Monro condition implies that such a totally asynchronous algorithm can only work if every state is visited infinitely often.

## 4.4 One-step simulation-based dynamic programming TD(0)

The first class of algorithms is extracted readily from writing Bellman operators as one-step expectation:

$$\begin{aligned}
 T^\pi V(s) &= \sum_{a \in \mathcal{A}} \pi(a; s) \left( r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) V(s') \right) = \mathbb{E}_s^\pi [R(s, A_0) + \gamma V(S_1)] \\
 T^\pi Q(s, a) &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}_{s'}} p(s'; s, a) \pi(a'; s) Q(s', a') = \mathbb{E}_s^\pi [R(s, a) + \gamma Q(S_1, A_1)] \\
 T^* Q(s, a) &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'; s, a) \max_{a' \in \mathcal{A}_{s'}} Q(s', a') = \mathbb{E}_s^{\pi^*} [R(s, a) + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')].
 \end{aligned}$$

Note that the same does not hold for the optimal Bellman operator for the state value function. These expressions are exactly of the form (4.10), thus, approximate fixed point iterations that only use one-step MDP samples can be used that converge to the unique fixed points. In the following we present following variants:

- SARS policy estimation of  $V^\pi$ ,
- SARSA policy estimation of  $Q^\pi$  based on,
- Q-learning and modifications such as SARSA and double-Q-learning.

Convergence proofs are similar, they follow immediately from Theorem 4.3.10 applied to the Bellman operators. More interesting is the practical implementation, i.e. the choice of asynchronous updates through  $\alpha$ . Finally, we prove convergence of the generalised policy iteration scheme resulting from SARSA updates of  $Q^\pi$  combined with policy improvement using policies that sufficiently explore but become greedy in the limit.



Compared to the direct Monte Carlo methods the methods presented below are much simpler to use. While Monte Carlo required an entire 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.

### 4.4.1 One-step policy evaluation (simulation-based policy evaluation)

We start with two simple model-free algorithms for evaluation of  $V^\pi$  and  $Q^\pi$ . In order to understand the tremendous freedom that stochastic approximation allows let us first formulate a general theorem and then transform them into useful algorithms.


**Theorem 4.4.1. (Convergence of one-step policy evaluation for  $V^\pi$ )**

Suppose  $\pi \in \Pi_S$  and  $V_0 \in \mathbb{R}^{|S|}$  is arbitrary and the asynchronous update rule is

$$V_{n+1}(s) = V_n(s) + \alpha_n(s)((R(s, a) + \gamma V_n(s'(s))) - V_n(s)), \quad s \in S,$$

with  $a \sim \pi(\cdot; s)$ ,  $s'(s) \sim p(\cdot; s, a)$  and step-sizes  $\alpha_n$  that only depend on the past steps and satisfy the Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s) < \infty \quad \text{a.s.}$$

for every  $s \in S$ . Then  $\lim_{n \rightarrow \infty} V_n(s) = V^\pi(s)$  almost surely, for every state  $s \in S$ .

It is important to note that the stochastic approximation algorithm does not impose any independence assumptions,  $\alpha_n$  only needs to be adapted to the algorithm and satisfy the decay properties. All possibilities from synchronous to totally asynchronous are perfectly fine as long as the Robbins-Monro conditions are satisfied.

*Proof of Theorem 4.4.1.* The convergence follows immediately from Theorem 4.3.10 once the algorithm is rewritten in the right way:

$$V_{n+1}(s) = V_n(s) + \alpha_n(s)(F(V_n)(s) - V_n(s) + \varepsilon_s(n))$$

with

$$F(V)(s) := \mathbb{E}_s^\pi[R(s, A_0) + \gamma V(S_1)]$$

and

$$\varepsilon_s(n) := (R(s, a) + \gamma V_n(s'(s))) - \mathbb{E}_s^\pi[R(s, A_0) + \gamma V_n(S_1)].$$

We need to be a bit careful with the filtration. The  $\sigma$ -algebras  $\mathcal{F}_n$  is generated by all random variables used to determine  $s, a, R(s, a)$  that appear in the definition of  $V(0), \dots, V(n)$ .

- In the proof of Theorem 3.1.26 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  $V$ ).
- The errors  $\varepsilon_s(n)$  are  $\mathcal{F}_{n+1}$ -measurable (they involve the next state-action pair  $(s, a)$ ) with  $\mathbb{E}[\varepsilon_s(n) | \mathcal{F}_n] = 0$  by definition. The assumed boundedness of the rewards also implies  $\sup_{n,s} \mathbb{E}[\varepsilon_s^2(n) | \mathcal{F}_n] \leq C$ .
- The step-sizes are adapted and satisfy  $\sum_{n=1}^{\infty} \alpha_n(s) = \infty$  a.s. and  $\sum_{n=1}^{\infty} \alpha_n^2(s) < \infty$  a.s. by assumption.

Hence, Theorem 4.3.10 can be applied and we get almost sure convergence to the unique fixpoint of  $F$  which is  $V^\pi$ .  $\square$

Let us turn the special situation of totally asynchronous updates into a concrete policy evaluation algorithm. There is a lot of freedom how to choose the step-size (it may for instance depend on the number of past updates of the coordinate as in the UCB bandit algorithm) and the coordinates. But the principle coordinate update rule must be

$$V_{\text{new}}(S_t) = V_{\text{old}}(S_t) + \alpha((R(S_t, A_t) + \gamma V_{\text{old}}(S_{t+1})) - V_{\text{old}}(S_t))$$

for  $a$  and  $s'$  chosen by the algorithm. Here is the simplest version of one-step value estimation in which we decide to choose  $a$  and  $s'$  from the value update as next coordinate to be updated. So, if we start in an initial state  $s$ , we sample  $a \sim \pi(\cdot; s)$  and  $s' \sim p(\cdot; s, a)$ . Then, we update the

**Algorithm 17:** SARS policy evaluation

---

**Data:** Policy  $\pi \in \Pi_S$   
**Result:** Approximation  $V \approx V^\pi$   
Initialize  $V \equiv 0$  and  $s$  arbitrary (for instance uniformly).  
**while** *not converged* **do**  
     $a \sim \pi(\cdot; s)$   
    Sample reward  $R(s, a)$ .  
    Sample next state  $s' \sim p(\cdot; s, a)$ .  
    Determine stepsize  $\alpha = \alpha(s)$ .  
    Update  $V(s) = V(s) + \alpha(R(s, a) + \gamma V(s')) - V(s)$   
    Set  $s = s'$ .  
**end**

---

value as given above. Then, to move on and to find a new suitable initial state, we take  $s'$  as new initial state and repeat the same update scheme. Since the scheme uses  $S$ - $A$ - $R$ - $S'$  to update the value of  $s$  we call the algorithm SARS policy evaluation.

There is a situation in which the above algorithm seems unsuitable. If the MDP terminates in finite times then states are not visited infinitely often so that the second Robbins-Monro condition is violated. Since the order of choosing states is completely irrelevant (this is governed by  $\alpha$  that only needs to be measurable) for convergence of stochastic approximation we can just repeatedly take rollout of finite length. As usually, a similar estimation procedure works for

**Algorithm 18:** SARS policy evaluation with termination

---

**Data:** Policy  $\pi \in \Pi_S$   
**Result:** Approximation  $V \approx V^\pi$   
Initialize  $V \equiv 0$   
**while** *not converged* **do**  
    Initialise  $s$  arbitrarily (for instance uniformly).  
    **while** *s not terminal* **do**  
         $a \sim \pi(\cdot; s)$   
        Sample reward  $R(s, a)$ .  
        Sample next state  $s' \sim p(\cdot; s, a)$ .  
        Determine stepsize  $\alpha = \alpha(s)$ .  
        Update  $V(s) = V(s) + \alpha(R(s, a) + \gamma V(s')) - V(s)$ .  
        Set  $s = s'$ .  
    **end**  
**end**

---

$Q$ . The disadvantage is that the algorithm requires a larger table to store all the values, the advantage is to get hold of  $Q$ -values that can be used for policy improvement. The analogous  $TD(0)$ -algorithm is based on

$$T^\pi Q(s, a) = \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma Q(S_1, A_1)]$$

It is not hard to guess that updates of the form

$$Q_{\text{new}}(S_t, A_t) = Q_{\text{old}}(S_t, A_t) + \alpha((R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})) - Q_{\text{old}}(S_t, A_t)),$$

where now every updated state-action pair requires a sample of the next state-action pair.



**Theorem 4.4.2. (Convergence of one-step policy evaluation for  $Q^\pi$ )**  
Suppose  $Q_0 \in \mathbb{R}^{|S| \times |A|}$  is arbitrary and, for  $n \in \mathbb{N}$ , the asynchronous update rule



is

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a)((R(s, a) + \gamma Q_n(s', a')) - Q_n(s, a)).$$

We assume for every  $n$  that  $s'(s, a) \sim p(\cdot; s, a)$  and  $a'(s, a) \sim \pi(\cdot; s'(s, a))$ , as well as  $\alpha_n$  depend only on the past steps and satisfy the Robbins-Monro conditions

$$\sum_{n=1}^{\infty} \alpha_n(s, a) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s, a) < \infty \quad \text{a.s.}$$

for every  $(s, a) \in \mathcal{S} \times \mathcal{A}$ . Then  $\lim_{n \rightarrow \infty} Q_n(s) = Q^\pi(s)$  almost surely, for every state-action pair  $(s, a)$ .

Again, note that there are no assumption whatsoever on the dependencies of  $\alpha_n$ , they only need to depend on the past only (be adapted). If the matrix  $\alpha_n$  only has one non-zero entry then the procedure is again called totally asynchronous.

*Proof.* Let us denote by  $(\tilde{S}_0, \tilde{A}_0, \dots)$  the state-action pairs defined by the algorithm. Furthermore, rewrite the updates as

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a)(F(Q_n)(s, a) - Q_n(s, a) + \varepsilon(s, a)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

with

$$F(Q)(s, a) := \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma Q(S_1, A_1)]$$

and

$$\varepsilon_n(\tilde{S}_n, \tilde{A}_n) := (R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1})) - \mathbb{E}_{\tilde{S}_n}^{\pi^{\tilde{A}_n}} [R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(S_1, A_1)].$$

and  $\varepsilon_n(s, a) = 0$  for  $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$ .

- In the proof of Theorem 3.1.26 it was shown that the Bellman expectation operator  $F = T^\pi$  is a  $\|\cdot\|_\infty$ -contraction.
- The errors  $\varepsilon_n(s)$  are  $\mathcal{F}_{n+1}$ -measurable with  $\mathbb{E}[\varepsilon_n(s) | \mathcal{F}_n] = 0$  by definition. The assumed boundedness of the rewards also implies  $\sup_{n,s} \mathbb{E}[\varepsilon_n^2(s) | \mathcal{F}_n] \leq C$ .
- The step-sizes are adapted and satisfy  $\sum_{n=1}^{\infty} \alpha_n(s) = \infty$  a.s. and  $\sum_{n=1}^{\infty} \alpha_n^2(s) < \infty$  a.s. by assumption.

Hence, Theorem 4.3.10 can be applied and we get almost sure convergence to the unique fixpoint of  $F$  which is  $Q^\pi$ .  $\square$

Similarly to the state value function a totally asynchronous algorithm is derived from the theorem. Now, we have to initialize a state-action pair, sample the reward and next state  $s'$ . The following action is then chosen according to the given policy and inserted into the estimate of the state-value function at the matrix position  $(s', a')$ . The state-action pair  $(s', a')$  is used for the next update except  $s'$  is a terminal state. Since the update uses  $S$ - $A$ - $R$ - $S'$ - $A'$  the policy algorithm is called SARSA. Here we only give the version with termination. In all algorithms above (and below) that depend on stochastic fixed point iterations the step-size  $\alpha$  needs to be set. For totally asynchronous learning there are two things that need to be specified for the algorithms:

- Which coordinates should be updated, i.e. which state or state-action pair is the only coordinate for which  $\alpha(n)$  is non-zero.
- How strongly should the coordinate be updated, i.e. what is the value  $\alpha_i(n)$  for the non-zero coordinate?



**Algorithm 19:** SARSA policy evaluation for  $Q^\pi$ 


---

**Data:** Policy  $\pi \in \Pi_S$   
**Result:** Approximation  $Q \approx Q^\pi$   
Initialize  $Q(s, a) = 0$  for all  $s \in S$   $a \in A$   
**while** *not converged* **do**  
    Initialise  $s$ .  
    Sample  $a \sim \pi(\cdot; s)$ .  
    **while**  $s$  *not terminal* **do**  
        Sample reward  $R(s, a)$ .  
        Sample next state  $s' \sim p(\cdot; s, a)$ .  
        Sample next action  $a' \sim \pi(\cdot; s')$ .  
        Determine step-size  $\alpha$ .  
        Update  $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma Q(s', a')) - Q(s, a))$   
        Set  $s = s', a = a'$ .  
    **end**  
**end**

---

The coordinate to be updated was already specified in the algorithms by  $s'$  for SARS (resp.  $(s', a')$  for SARSA). Due to the Robbins-Monro condition every state (resp. state-action pair) must be visited infinitely often to ensure the values of  $\alpha_i(n)$  might sum-up to  $+\infty$ . But what is good choice for the non-zero value of  $\alpha(n)$ ?



Even though invalid for the theory a popular choice for the step-size is a constant value  $\alpha$ . The most obvious choice to guarantee convergence of the algorithms is, for some  $\frac{1}{2} < p \leq 1$ ,

$$\alpha_s(n) = \frac{1}{(T_s(n) + 1)^p} \quad \text{resp.} \quad \alpha_{(s,a)}(n) = \frac{1}{(T_{(s,a)}(n) + 1)^p}$$

if  $T_s(n)$  denotes the number of times the state  $s$  was updated during the first  $n$  updates (resp.  $T_{(s,a)}(n)$  the number of times the state-action pair  $(s, a)$  was updated during the first  $n$  updates). The choice is reasonable because if states (or state-action pairs) are visited infinitely often, then

$$\sum_{n=1}^{\infty} \alpha_s(n) = \sum_{n=1}^{\infty} \frac{1}{n^p} = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_s^2(n) = \sum_{n=1}^{\infty} \frac{1}{n^{2p}} < \infty.$$

#### 4.4.2 $Q$ -learning and SARSA (simulation-based value iteration for $Q$ )

We continue with the most famous tabular control algorithms. Approximate value iteration to solve  $T^*Q = Q$  by iterating in an approximate way Bellman's state-action optimality operator  $T^*$ . Such algorithms directly learn the optimal state-action function  $Q^*$  without repeated policy improvement steps, they are simulation-based versions of value iteration for  $Q$ . The main advantage (but also disadvantage) of the  $Q$ -learning algorithm (and its modifications) is the flexibility, there are plenty of choices that can be made to explore the state-action space.



**Definition 4.4.3.** A learning algorithm is called **off-policy** if the choice of actions is not governed by the currently estimated policy but the exploration of actions can be driven by outside (random) sources. In contrast, an algorithm is called **on-policy** if the exploration only uses the currently estimated policy to explore actions. In many off-policy examples the actions are explored using a fixed policy that is typically called a **behavior policy**.

The term off-policy is not always used in the same way, also our definition is relatively vague. The most extreme scenario is that of a behavior policy that does not take into account the algorithmic learning at all. To have a clear idea in mind consider the following example. There are two ways of learning chess. Either observing repeated matches of other players or by repeatedly playing by oneself. The first is a typical off-policy situation, learning is accomplished by looking over someone's shoulders. Same with learning how to use a tool. Either looking over someones shoulds or trying the tool oneself. This illustration already shows the advantages and disadvantages. A disadvantage of off-policy learning is that it might be hard to learn from a bad player and more efficient to learn from a player that plays good and bad moves. In contrast, on-policy learning might lead to little exploration of actions and the algorithm focuses too much on what the algorithm believes to be good, similarly to the exploration-exploitation trade-off for stochastic bandits.



From the mathematics of convergence proofs the situation is relatively simple. In totally asynchronous approximate fixed point iterations for  $Q$  the vector  $\alpha$  governs the choice of state-action pairs. Since the sequence  $(\alpha_n)$  only needs to be adapted to the algorithm (its filtration) there is a lot of freedom on how to explore the actions. Essentially, we can explore in all ways that do not use future knowledge of the algorithm (how should we?) and explores all state-action pairs infinitely often.

No doubt, the remark only concerns convergence in the limit but not the speed of convergence. The mathematical theory of approximate learning of state-action value functions is not particularly well developed. In practise, algorithms work differently well on different examples. We start with the most direct algorithm, the approximate fixed point iteration with samples of Bellman's state-action optimality operator:  $Q$ -learning - Algorithm 21. There is a lot that can be changed

---

**Algorithm 20:**  $Q$ -learning (with behavior policy)
 

---

**Data:** Behavior policy  $\pi \in \Pi_S$

**Result:** Approximations  $Q \approx Q^*$ ,  $\pi = \text{greedy}(Q) \approx \pi^*$

Initialize  $Q$  (e.g.  $Q \equiv 0$ ).

**while** *not converged* **do**

    Initialize  $s$ .

**while**  $s$  *not terminal* **do**

        Sample  $a \sim \pi(\cdot; s)$ .

        Sample reward  $R(s, a)$ .

        Sample  $s' \sim p(\cdot; s, a)$ .

        Determine stepsize  $\alpha$ .

        Update  $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma \max_{a' \in \mathcal{A}_{s'}} Q(s', a')) - Q(s, a))$ .

        Set  $s = s'$ .

**end**

**end**

---

about  $Q$ -learning. To understand why let us check a convergence proof:



**Theorem 4.4.4. (Convergence of  $Q$ -learning)**

Suppose that a behavior policy  $\pi$  is such that all state-action pairs are visited infinitely often and the step-sizes are adapted and satisfy the Robbins-Monro conditions. Then  $\lim_{t \rightarrow \infty} Q_t(s, a) = Q^*(s, a)$  holds almost surely for all state-action pairs  $(s, a)$ .

*Proof.* The proof is exactly the same that we have seen above, now using the optimal Bellman operator

$$T^*Q(s, a) = \mathbb{E}_s^{\pi^a} [R(s, a) + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')]$$

on  $\mathbb{R}^{|S| \cdot |A|}$ . Please finish the proof yourself as an exercise:



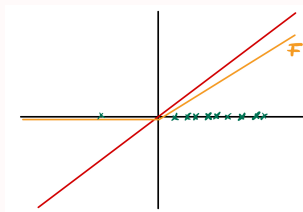
Rewrite the  $Q$ -learning algorithm as simulation-based fixed point iteration and check the conditions of Theorem 4.3.10 to prove the almost sure convergence.

□

Before discussing versions of  $Q$ -learning let us think for a moment what  $Q$ -learning really does.  $Q$ -learning challenges current estimates  $Q(s, a)$  of  $Q^*(s, a)$  by comparing them with new one-step estimates. If the  $Q$ -values are stored in a table, then the algorithm successively explores table entries and updates their values. In every step one entry of the table is updated as follows. The current estimate  $Q(s, a)$  is compared to a new estimate  $Q(s, a)$ , namely, by resampling only the first step (this is  $R(s, a)$ ) and then assuming the estimated future discounted reward from the new state  $s'$  is perfect (this is  $\max_{a'} Q(s', a')$ ). The table entry  $Q_{\text{new}}(s, a)$  is increased/decreased according to the new estimate  $R(s, a) + \max_{a'} Q(s', a')$  for  $Q^*(s, a)$  being larger/smaller than the old entry. Unfortunately, there is a huge drawback of  $Q$ -learning:  $Q$ -learning converges in the limit but can overestimate the  $Q$ -values by a lot. Here is an example that shows what can go wrong.



Stochastic approximation algorithms are not unbiased, i.e. the estimates  $x_n$  of  $x^*$  typically satisfy  $\mathbb{E}[x_n] \neq x^*$ . Here is a one-dimensional illustration that roughly captures the overestimation effect of  $Q$ -learning. Take  $F(x) = \gamma \max\{0, x\}$  for some  $\gamma < 1$ , this is a contraction.



An overestimating approximate fixedpoint iteration

Without error the algorithm quickly converges from the left to  $x^* = 0$  (even in one step if  $\alpha = 1$ ) because the steps are  $\alpha_n(F(x_n) - x_n) = -\alpha_n x_n$  while the convergence from the right takes some time as  $\alpha_n(F(x_n) - x_n) \approx 0$ . With errors the situation becomes worse even with  $x_0 = x^*$ . A positive error sends  $x_n$  to the positives from where the algorithm slowly comes back to 0 while after sending  $x_n$  to the negatives with a negative error the algorithm comes back to 0 much faster. The same effect happens for  $Q$ -learning. Estimated  $Q$ -values are typically too large. Variants of  $Q$ -learning that deal with the overestimation effect are discussed in the next section.

For convergence, the way the state-action pairs are chosen can be even more arbitrary as long as all state-action pairs are visited infinitely often and the step-sizes decrease according to the Robbins-Monro conditions. For instance, state-action pairs could also be chosen randomly with equal step-size  $\frac{1}{n}$ . We know from stochastic bandits that ignoring all learned information is not going to be beneficial. If we are interested in the performance of the algorithm during training, e.g. the total rewards during training (like the regret for bandit algorithms), we want to choose  $\pi$  such that *good* actions are taken more often but still enough exploration of new state-action pairs takes place. Thus, the exploration of table entries is often organised by  $\epsilon$ -greedy or Boltzman exploration.

A variant of  $Q$ -learning is SARSA which has the following update rule:

$$Q_{\text{new}}(S_t, A_t) = Q_{\text{old}}(S_t, A_t) + \alpha((R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})) - Q_{\text{old}}(S_t, A_t)),$$

where the choice of the next action must be on-policy (determined by  $Q$ , such as  $\varepsilon$ -greedy). The algorithm is not always going to converge, the right trade-off between exploration and exploitation of the best known estimate of  $Q^*$  (the current  $Q$ ) is needed. For  $\varepsilon$  too large the algorithm will play unfavorable state-action pairs too often, for  $\varepsilon$  too small the algorithm might miss to learn good actions. Since the updates use  $S$ - $A$ - $R$ - $S'$ - $A'$  this on-policy version of  $Q$ -learning is called SARSA control.

---

**Algorithm 21:** 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  $\alpha$ .  
     Sample reward  $R(s, a)$ .  
     Chose new policy  $\pi$  from  $Q$  (e.g.  $\varepsilon$ -greedy).  
     Sample next state  $s' \sim p(\cdot; s, a)$ .  
     Sample next action  $a' \sim \pi(\cdot; s')$ .  
     Update  $Q(s, a) = Q(s, a) + \alpha((R(s, a) + \gamma Q(s', a')) - Q(s, a))$ .  
     Set  $s = s'$ ,  $a = a'$ .  
   **end**  
**end**

---



SARSA simplifies if the policy update is greedy with respect to  $Q$ . In that case  $a' = \arg \max_a Q(s', a)$  so that the  $Q$ -updates are exactly the updates of  $Q$ -learning.

In contrast to  $Q$ -learning SARSA cannot be performed off policy.



Suppose for instance the actions  $a'$  are chosen uniformly then SARSA would not converge to  $Q^*$ . Think about it and try it in simulations!

Unlike  $Q$ -learning it is less obvious that SARSA converges to  $Q^*$ , in contrast to  $Q$ -learning the iteration is not an approximate version of the fixed point iteration for  $T^*$ ! The following proof is important as it is used in other settings as well, in the literature one can occasionally see reference to the SARSA convergence proof for modifications of  $Q$ -learning (for instance for double- $Q$  and clipping). The idea is to compare to the  $Q$ -learning iteration and then show the appearing error vanishes. Unfortunately, our version of approximative fixedpoint iteration only allows unbiased errors while the proof needs unbiased errors  $\mathbb{E}[\varepsilon_n | \mathcal{F}_n] \neq 0$ . For that sake a generalisation of Theorem 4.3.10 is needed.



Show that the statement of Theorem 4.3.10 also holds with  $\mathbb{E}[\varepsilon_i(n) | \mathcal{F}_n] \neq 0$  if

$$\sum_{n=1}^{\infty} \alpha_i(n) |\mathbb{E}[\varepsilon_i(n) | \mathcal{F}_n]| < \infty \quad (4.12)$$

holds almost surely. Going through the proof it will become clear that only Lemma 4.3.6 needs to be improved to this situation. Luckily the proof is short enough to see quickly (estimating  $W \leq 1 + W^2$ ) that (4.12) is enough to reduce the lemma to the corollary of Robbins-Siegmund theorem 4.3.5.

In fact, there are even more general versions that allow to suppress the summation condition of the next theorem. Since the most influential paper <sup>7</sup> seems to have an error (there is a circular

<sup>7</sup>S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms", Machine Learning, 38(3):287–308, 2000

argument in the last paragraph of the paper) we only use what has been developed in these lecture notes for the next convergence proof.



**Theorem 4.4.5. (Convergence of SARSA control for terminating MDPs)**

Suppose the step-sizes satisfy the Robbins-Monro conditions and the probabilities  $p_n(s, a)$  that the policy  $\pi_{n+1}$  is not greedy satisfies are such that

$$\sum_{n=1}^{\infty} \alpha_n(s, a)p_n(s, a) < \infty \quad \text{a.s.}$$

for all  $(a, s)$ . Then the SARSA algorithm converges to  $Q^*$  almost surely.

<sup>8</sup> The most important policy to which the theorem applies is  $\alpha_n$ -greedy because  $\sum \alpha_n^2(s, a) < \infty$  holds by assumption. Choosing  $\alpha_n(s, a) = \frac{1}{T_{s,a}(n)+1}$  the exploration rate in a state-action pair  $(s, a)$  decreases with the number of updates of  $Q(a, s)$ .

*Proof.* The proof is different from the ones before. The reason is that the updates are not directly estimates of a contraction operator. Nonetheless, a similar argument works. Adding a zero the algorithm can be reformulated in an approximate fixed point iteration with an error-term that is biased but with a bias decreasing to zero. We will use a variant of Theorem 4.3.10. The convergence also holds if

$$\sum_{n=1}^{\infty} \alpha_n(s, a) |\mathbb{E}[\varepsilon_n(s, a) | \mathcal{F}_n]| < \infty \quad \text{a.s.} \quad (4.13)$$

for all state-action pairs. Hence, we check the condition (4.13) instead of  $\mathbb{E}[\varepsilon_n(s, a) | \mathcal{F}_n] = 0$  with an appropriately chosen error-term. Let us denote by  $\tilde{S}_0, \tilde{A}_0, \dots$  the sequence of state-action pairs obtained from the algorithm. First, writing

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha_n(s, a)(T^*Q_n(s, a) - Q_n(s, a) + \varepsilon_n(s, a)), \quad s \in \mathcal{S}, a \in \mathcal{A}_s,$$

with

$$\varepsilon_n(\tilde{S}_n, \tilde{A}_n) = (R(\tilde{S}_n, \tilde{A}_n) + \gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1})) - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [R(\tilde{S}_n, \tilde{A}_n) + \gamma \max_{a'} Q_n(S_1, a')]$$

and  $\varepsilon(s, a) = 0$  for  $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$ . The errors  $\varepsilon_n(s, a)$  are  $\mathcal{F}_{n+1}$ -measurable. Furthermore,  $\mathbb{E}[\varepsilon(s, a) | \mathcal{F}_n] = 0$  for  $(s, a) \neq (\tilde{S}_n, \tilde{A}_n)$  and

$$\begin{aligned} & \mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ is greedy}\}} (\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ & \quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} (\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ greedy}\}} | \mathcal{F}_n] \underbrace{\mathbb{E}[\gamma \max_{a'} Q_n(\tilde{S}_{n+1}, a') - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]}_{=0} | \mathcal{F}_n] \\ & \quad + \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} | \mathcal{F}_n] \mathbb{E}[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n] \\ &= \mathbb{E}[\mathbf{1}_{\{\pi_{n+1}(\cdot; \tilde{S}_{n+1}) \text{ non-greedy}\}} | \mathcal{F}_n] \mathbb{E}[(\gamma Q_n(\tilde{S}_{n+1}, \tilde{A}_{n+1}) - \mathbb{E}_{\tilde{S}_n}^{\pi_{\tilde{A}_n}} [\gamma \max_{a'} Q_n(S_1, a')]) | \mathcal{F}_n]. \end{aligned}$$

Finally, recall that we assume throughout these lecture notes that rewards are bounded. Thus, the assumed boundedness of  $Q_0$  and the iteration scheme combined with  $\sum_{k=0}^{\infty} \gamma^k < \infty$  implies that  $|Q_n(s, a)| < C$  for some  $C$  and all  $n \in \mathbb{N}$ . Thus,

$$|\mathbb{E}[\varepsilon_n(\tilde{S}_n, \tilde{A}_n) | \mathcal{F}_n]| \leq C \mathbb{P}(\pi_{n+1}(\cdot; s') \text{ non greedy} | \mathcal{F}_n) = C p_n(s, a).$$

<sup>8</sup>problem mit der messbarkeit im

By assumption, the summation condition 4.13 is satisfied. By the boundedness of rewards, also  $\mathbb{E}[\varepsilon_n^2(s, a) | \mathcal{F}_n] \leq C < \infty$ . As  $T^*$  is a contraction and the Robbins-Monro conditions are satisfied, the iteration converges to  $Q^*$  almost surely.  $\square$

To get a feeling of the SARSA algorithm think about stochastic bandits seen as one-step MDPs.



To get a feeling for  $Q$ -learning and SARSA try to relate the algorithms with  $\alpha_n(s, a) = \frac{1}{T_{s,a}(n)+1}$  to the  $\varepsilon$ -greedy algorithm for stochastic bandits introduced in Chapter 1.

### 4.4.3 Double $Q$ -learning

The aim of this section is to introduce double variants of  $Q$ -learning that deal with overestimation of  $Q$ -learning. For a rough understanding of what goes wrong in  $Q$ -learning recall that the  $Q_n(s, a)$  are (random) estimates of the expectations  $Q^*(s, a)$ . In the updates of  $Q$ -learning we use the estimates  $\max_{a' \in \mathcal{A}_s} Q_n(s, a)$  of  $\max_{a' \in \mathcal{A}_s} Q^*(s, a)$  but those overestimate.



Suppose  $\hat{\mu}_1, \dots, \hat{\mu}_K$  are estimates for expectations  $\mu_1, \dots, \mu_K$ . Then the pointwise maximum  $\max_{i=1, \dots, K} \hat{\mu}_i$  overestimates  $\hat{\mu} = \max_{i=1, \dots, K} \mu_i$  because  $\mathbb{E}[\max \hat{\mu}_i] \geq \mathbb{E}[\mu_i]$  for all  $i$  so that  $\mathbb{E}[\max \hat{\mu}_i] \geq \max \mathbb{E}[\mu_i]$ . As a simple example suppose  $M_1, \dots, M_K$  are  $\text{Ber}(p)$ -distributed. They are all unbiased estimators of the mean but

$$\begin{aligned} \mathbb{E}[\max M_i] &= \mathbb{P}(\{M_1 = 0, \dots, M_K = 0\}) \cdot 0 + \mathbb{P}(\{M_1 = 0, \dots, M_K = 0\}^c) \cdot 1 \\ &= 1 - (1 - p)^K > p. \end{aligned}$$

Van Hasselt<sup>9</sup> 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 22: 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  $\alpha$ .

        Randomly choose  $\text{update} = A$  or  $\text{update} = B$

**if**  $\text{update} = A$  **then**

$a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$

            Update  $Q^A(s, a) = Q^A(s, a) + \alpha(R(s, a) + \gamma Q^B(s', a^*) - Q^A(s, a))$

**end**

**else**

$b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$

            Update  $Q^B(s, a) = Q^B(s, a) + \alpha(R(s, a) + \gamma Q^A(s', b^*) - Q^B(s, a))$

**end**

        Set  $s = s'$ .

**end**

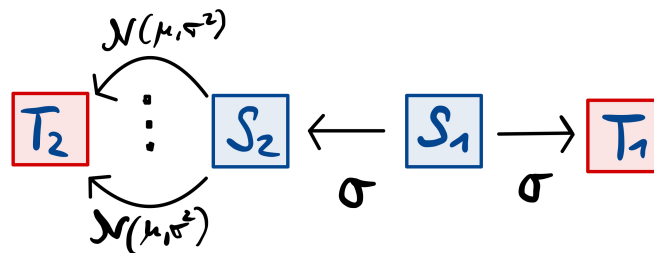
**end**

---

<sup>9</sup>H. van Hasselt, "Double Q-learning", NIPS 2010

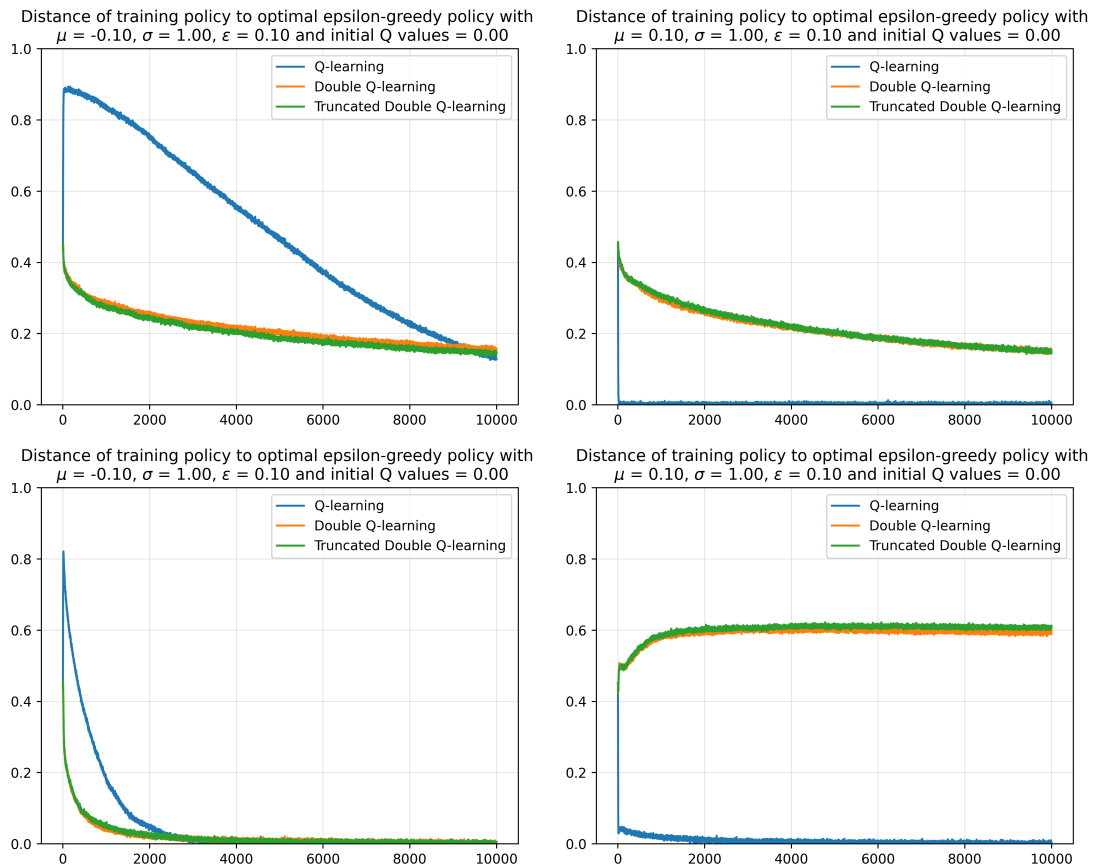
Similarly to SARSA (see the proof of Theorem 4.4.7) double  $Q$ -learning can be interpreted as classical  $Q$ -learning with an additional negatively biased term  $\hat{\epsilon}$ . We will not prove convergence of double  $Q$ -learning. The proof requires a stronger unbiased version of approximate dynamic programming. Instead we will introduce a truncation which allows us to follow the proof of convergence for SARSA. Interestingly, our new version of double  $Q$ -learning performs better than  $Q$ -learning and double  $Q$ -learning on some of the standard examples. Before stating the truncated double  $Q$ -learning algorithm let us check an example:

**Example 4.4.6.** The example MDP consists of two interesting states  $S_1, S_2$  and two terminal states  $T_1, T_2$ . The transition probabilities are indicated in the graphical representation. If action left/right is chosen in  $S_1$  then the MDP certainly moves to the left/right with zero reward. In state  $S_2$  there are several possible actions that all lead to  $T_2$  and yield a Gaussian reward with mean  $\mu$  and variance  $\sigma^2$  (typically equal to 1). Started in  $S_1$  an optimal policy clearly choses to terminate in  $T_1$  as the expected total reward is 0, a policy that also allows to go to  $S_2$  yields a negative expected total reward.



red states are terminating

In the classical example  $Q$  is initialised as 0 and  $\mu$  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  $\mu$  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  $\epsilon$ -greedy policy with  $\epsilon = 0.1$  was used. The plots show depending on the setup/initialisation either positive or negative bias can be beneficial.

Let us now proceed with what we call the truncated double  $Q$ -learning algorithm that contains other important double algorithms as special cases.



**Algorithm 23:** 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  $\alpha$ .        Randomly choose  $\text{update} = A$  or  $\text{update} = B$         **if**  $\text{update} = A$  **then**             $a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$              $\epsilon = \gamma \max(-C_- \alpha, \min(Q^B(s', a^*) - Q^A(s', a^*), C_+ \alpha))$             Update  $Q^A(s, a) = Q^A(s, a) + \alpha (R(s, a) + \gamma Q^A(s', a^*) - Q^A(s, a) + \epsilon)$         **end**        **else**             $b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$              $\epsilon = \gamma \max(-C_- \alpha, \min(Q^A(s', b^*) - Q^B(s', b^*), C_+ \alpha))$             Update  $Q^B(s, a) = Q^B(s, a) + \alpha (R(s, a) + \gamma Q^B(s', b^*) - Q^B(s, a) + \epsilon)$         **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:

$$\begin{aligned} Q_{n+1}^A(s, a) &= Q_n^A(s, a) + \alpha (R(s, a) + \gamma Q_n^B(s', a^*) - Q_n^A(s', a)) \\ &= Q_n^A(s, a) + \alpha (R(s, a) + \gamma Q_n^A(s', a^*) - Q_n^A(s, a) + \gamma (Q_n^B(s', a^*) - Q_n^A(s', a^*))) \end{aligned}$$

which can be written as

$$Q_{n+1}^A(s, a) = Q_n^A(s, a) + \alpha (T^*(Q_n^A)(s, a) + \epsilon_n(s, a) - Q_n(s, a)).$$

with errors

$$\epsilon_n(s, a) := \underbrace{[R(s, a) + \gamma Q_n^A(s', a^*) - T^*Q_n^A(s, a)]}_{=: \epsilon_n^Q(s, a)} + \underbrace{\gamma (Q_n^B(s', a^*) - Q_n^A(s', a^*))}_{=: \hat{\epsilon}_n(s, a)}.$$

Thus, from the point of view of approximate dynamic programming, double  $Q$ -learning is nothing but  $Q$ -learning with an additional error. Since the two equations are symmetric the error is negatively biased (the  $Q$ -function for some action should be smaller than the  $Q$ -function for the best action).



Make this intuition rigorous, show that  $\mathbb{E}[\hat{\epsilon}_n | \mathcal{F}_n] < 0$  almost surely.

Furthermore, looking carefully at the algorithm, truncated double  $Q$ -learning equals double  $Q$ -learning if  $C_+ = C_- = +\infty$  (or, in practice, just very large) as in that case

$$\max(-C_- \alpha, \min(Q_n^B(s', a^*) - Q_n^A(s', a^*), C_+ \alpha)) = Q_n^B(s', a^*) - Q_n^A(s', a^*).$$



**Theorem 4.4.7.** Consider the updating procedure of truncated double  $Q$ -learning in algorithm for  $Q^A$  and  $Q^B$ . Suppose that a behavior policy  $\pi$  is such that all state-action pairs are visited infinitely often and the step sizes are adapted and satisfy the Robbins-Monro conditions. Then

$$\lim_{t \rightarrow \infty} Q^A(s, a) = \lim_{t \rightarrow \infty} Q^B(s, a) = Q^*(s, a)$$

holds almost surely for all state-action pairs  $(s, a)$ .

Please note that truncated  $Q$ -learning was only introduced for these lectures notes as it allows us to prove convergence using Theorem 4.3.10. Interestingly, the algorithm performs pretty well on examples and it might be worth improving the algorithm by adaptive (depending on the past data) choice of  $C_+$  and  $C_-$  to learn the need of over- or understimation.

*Proof.* Because of the symmetry of the update procedure it is sufficient to prove the convergence of  $Q^A$ . As for SARSA the point is to use the reformulation as  $Q$ -learning with additional error and show that errors are sufficiently little biased. This is why we introduced the additional truncation in order to be able to check

$$\sum_{n=0}^{\infty} \alpha_n(s, a) |\mathbb{E}[\epsilon_n(s, a) | \mathcal{F}_n]| < \infty. \quad (4.14)$$

In the following we will write  $a^* = \arg \max_{a \in \mathcal{A}_{s'}} Q^A(s', a)$  and  $b^* = \arg \max_{a \in \mathcal{A}_{s'}} Q^B(s', a)$  for given  $s'$  and write

$$Q_{n+1}^A(s, a) = Q_n^A(s, a) + \alpha_n(s, a) (T^*(Q_n^A)(s, a) + \epsilon_n(s, a) - Q_n^A(s, a)).$$

with error

$$\begin{aligned} \epsilon_t(s, a) &:= \left[ R(s, a) + \gamma Q_n^A(s', a^*) - T^* Q_n^A(s, a) \right] \\ &\quad + \gamma \max(-C_- \alpha_n(s, a), \min(Q_n^B(s', a^*) - Q_n^A(s', a^*), C_+ \alpha_n(s, a))) \\ &=: \epsilon_n^Q + \hat{\epsilon}_n(s, a). \end{aligned}$$

All that remains to show is that the error term has bounded conditional second moments and the bias satisfies (4.14). Finite second moments follow as we assume in these lectures (for simplicity) that the rewards are bounded so that  $\sum_{k=0}^{\infty} \gamma^k = \frac{\gamma}{1-\gamma}$  implies boundedness. The  $Q$ -learning error is unbiased (sample minus expectation of the sample). The truncated double  $Q$ -learning error is also bounded:

$$\sum_{n=0}^{\infty} \alpha_n(s, a) |\mathbb{E}[\hat{\epsilon}_n(s, a) | \mathcal{F}_n]| \leq \max\{C_+, C_-\} \sum_{n=0}^{\infty} \alpha_n^2(s, a) < \infty. \quad (4.15)$$

Since  $T^*$  is a contraction and the learning rates satisfy the Robbins-Monro conditions the convergence follows from Theorem 4.3.10 with the modification.  $\square$



The interesting feature of truncated double  $Q$ -learning is the interpolation effect between  $Q$ -learning and double  $Q$ -learning. Large  $C$  makes the algorithm closer to double  $Q$ -learning, small  $C$  to  $Q$ -learning. It would be interesting to see if an adaptive choice of  $C$  (depending on the algorithm) could be used to combine the overestimation of  $Q$ -learning and the understimation of double  $Q$ -learning.

We finish this section with another variant of double  $Q$ -learning, so-called clipping<sup>10</sup>:

<sup>10</sup>Fujimoto, van Hoof, Meger: "Addressing Function Approximation Error in Actor-Critic Methods", ICML 2018

**Algorithm 24:** 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**  
    Initialize  $s$ .  
    **while**  $s$  *not terminal* **do**  
        Sample  $a \sim \pi(\cdot; s)$ .  
        Sample reward  $R(s, a)$ .  
        Sample  $s' \sim p(\cdot; s, a)$ .  
        Determine stepsize  $\alpha$ .  
        **if**  $update = A$  **then**  
             $a^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^A(s', a')$   
             $Q^A(s, a) = Q^A(s, a) + \alpha(R(s, a) + \gamma \min\{Q^A(s', a^*), Q^B(s', a^*)\} - Q^A(s, a))$ .  
        **end**  
        **else**  
             $b^* = \arg \max_{a' \in \mathcal{A}_{s'}} Q^B(s', a')$   
             $Q^B(s, a) = Q^B(s, a) + \alpha(R(s, a) + \gamma \min\{Q^A(s', b^*), Q^B(s', b^*)\} - Q^B(s, a))$ .  
        **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.

## 4.5 Multi-step approximate dynamic programming

The one-step approximate dynamic programming algorithms were derived rather directly from Theorem 4.3.10. We next turn towards finitely many steps forwards which is inbetween the one-step and infinitely many steps (aka Monte Carlo) approaches.

### 4.5.1 $n$ -step TD for policy evaluation and control

The idea of approximate is to solve fixed point equation in the case in which operators that are given by expectations can only be approximated by sampling. We now ignore this point of view

and recall what really happens in the updates. For SARSA policy evaluation those were

$$\underbrace{V_{\text{new}}(S_t)}_{\text{new estimate}} = V_{\text{old}}(S_t) + \alpha \left( \underbrace{R(S_t, A_t) + \gamma V_{\text{old}}(S_{t+1})}_{\text{reestimate of } V(S_t)} - \underbrace{V_{\text{old}}(S_t)}_{\text{old estimate}} \right)$$

and for evaluation of  $Q^\pi$

$$\underbrace{Q_{\text{new}}(S_t, A_t)}_{\text{new estimate}} = Q_{\text{old}}(S_t, A_t) + \alpha \left( \underbrace{R(S_t, A_t) + \gamma Q_{\text{old}}(S_{t+1}, A_{t+1})}_{\text{reestimate of } Q(S_t, A_t)} - \underbrace{Q_{\text{old}}(S_t, A_t)}_{\text{old estimate}} \right).$$

In every step the algorithms reestimate the state(-action) value function by resampling the first step and then continuing according to dynamic programming with the current estimate. The difference between new estimate and old estimate is called temporal-difference (TD) error. Weighting old and new estimates then leads to an increase for positive TD error (resp. a decrease for negative TD error). A natural generalisation for this reestimation procedure uses longer temporal differences, i.e. resample the next  $n$  steps and then continue according to the old estimate, i.e. reestimating with  $\sum_{i=1}^{n-1} \gamma^i R(S_{t+i}, A_{t+i}) + \gamma^n V^\pi(S_{t+n})$ . The corresponding algorithms are called  $n$ -step TD algorithms. We are not going to spell-out the details, the only difference is the update which is given below and that (compared to one-step) updates are stopped  $n$  steps before the termination as  $n$  steps in the future are used for the update.



Use the update rule

$$\underbrace{V_{\text{new}}(S_t)}_{\text{new estimate}} = V_{\text{old}}(S_t) + \alpha \left( \underbrace{\sum_{i=1}^{n-1} R(S_{t+i}, A_{t+i}) + \gamma V_{\text{old}}(S_{t+n})}_{\text{reestimate of } V(S_t)} - \underbrace{V_{\text{old}}(S_t)}_{\text{old estimate}} \right)$$

to write down pseudocode for  $n$ -step TD algorithms for evaluation of  $V^\pi$  and  $Q^\pi$  and prove the convergence by checking the  $n$ -step Bellman expectation equations <sup>a</sup>

$$T^\pi V(s) = \mathbb{E}_s^\pi \left[ R(s, A_0) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) \right] \quad (4.16)$$

and

$$T^\pi Q(s, a) = \mathbb{E}_s^{\pi^a} \left[ R(s, a) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n Q(S_n, A_n) \right]$$

and the conditions of Theorem 4.3.10 on the error term.

<sup>a</sup>write down?

Analogously, a new  $n$ -step SARSA-type control algorithm can be written down, we will compare the performance to 1-step SARSA in the implementation exercises.



Write pseudocode for an  $n$ -step SARSA control algorithm in the non-terminating case. Try to prove convergence in the same way we did for 1-step SARSA in Theorem 4.4.5.

To understand multistep methods better let us compare the Monte Carlo estimator of  $V^\pi$  from Section 4.2.1 with the one-step approximate dynamic programming estimator of  $V^\pi$  from Section 4.4.1. Looking closely at the  $n$ -step estimator a crucial observation can be made. For large  $n$  the TD update is essentially the Monte Carlo update. For a particular problem one can thus choose  $n$  such that the algorithm is closer to Monte Carlo (no reuse of samples) or closer to one-step approximate dynamic programming.

- The Monte Carlo estimator averages independent samples  $\hat{V}_k^\pi = \sum \gamma^t R(s_t, a_t)$  of the discounted total reward to get

$$\hat{V}^\pi(s) = \frac{1}{N} \sum_{k=1}^N \hat{V}_k^\pi(s).$$

The Monte Carlo estimator uses every sample  $R(s, a)$  once, whereas the dynamic programming estimator reuses samples (this is called bootstrapping) because the iteration scheme

$$V_{\text{new}}(s) = V_{\text{old}}(s) + \alpha(R(s, a) + \gamma V_{\text{old}}(s') - V_{\text{old}}(s))$$

reuses all samples  $R(s, a)$  that were used to estimate  $V_{\text{old}}(s')$ . From the practical point of view the bootstrapping is desirable if the sampling is expensive. Additionally, the reuse of estimates reduces the variance of the SARS estimator compared to the Monte Carlo estimator.

- Being unbiased Monte Carlo has a clear advantage to the unbiased algorithms obtained from stochastic approximation schemes for which we know nothing about the bias.

Let's turn these thoughts into a formal error decomposition. Suppose an estimate  $V$  of  $V^\pi$  is given, this is typically  $V_N$ . How close (in  $L^2$ ) is the new  $n$ -step estimate from the true vector  $V^\pi$ ?

**Proposition 4.5.1. (TD bias-variance decomposition)**

Suppose  $V$  is an old estimate of  $V^\pi$ , then there is a constant  $C$  (depending on the assumed bound for  $R$ ) such that

$$\begin{aligned}
 & \mathbb{E}_s^\pi \left[ \overbrace{\left( \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right)^2}^{\text{new estimation error}} \right] \\
 & \leq \underbrace{\gamma^{2n} \mathbb{E}_s^\pi [(V^\pi(S_n) - V(S_n))]^2}_{\text{old estimation bias}} + \underbrace{\mathbb{V}_s^\pi \left[ \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) \right] + \gamma^{2n} C \mathbb{V}_s^\pi [V(S_n)]}_{\text{n-step + future prediction variance}}.
 \end{aligned}$$

Before going through the proof let us discuss what can be learnt from the proposition. Recall that  $\gamma$  is fixed,  $V$  given by prior iterations of the algorithm, and  $n$  could be chosen.

- The first summand involves  $\gamma^{2n}$  which decreases in  $n$  and the squared error of the current estimate at time  $n$ .
- The second summand does not depend on the current estimate  $V$ , but is the variance of the  $n$ -step rewards under the target policy. This is the Monte Carlo variance for  $n$  steps.
- The last summand again involves  $\gamma^{2n}$  which decreases in  $n$  and the variance of the current  $n$ -step prediction.

The proof uses the classical bias-variance decomposition from statistics combined with the fact that  $T^\pi V^\pi = V^\pi$  and (4.16).

*Proof.*

$$\begin{aligned}
& \mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right)^2 \right] \\
&= \mathbb{E}_s^\pi \left[ \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right]^2 \\
&\quad + \mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) - \mathbb{E}_s^\pi \left[ \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) - V^\pi(S_0) \right] \right)^2 \right] \\
&\stackrel{(4.16)}{=} \left( -\gamma^n \mathbb{E}_s^\pi [V^\pi(S_n)] + \gamma^n \mathbb{E}_s^\pi [V(S_n)] \right)^2 \\
&\quad + \mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) - \mathbb{E}_s^\pi \left[ \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) \right] + \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) - \underbrace{(V^\pi(S_0) - \mathbb{E}_s^\pi [V^\pi(S_0)])}_{=0} \right)^2 \right] \\
&= \gamma^{2n} \left( \mathbb{E}_s^\pi [V^\pi(S_n)] - V(S_n) \right)^2 + \mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right)^2 \right] \\
&\quad + 2\mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right] \\
&\quad + \mathbb{E}_s^\pi \left[ \left( \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right)^2 \right].
\end{aligned}$$

If we can estimate the third summand we are done. The following estimate is pretty rough and uses the boundedness of  $R$ :

$$\begin{aligned}
& \mathbb{E}_s^\pi \left[ \left( \sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right] \\
&\leq \mathbb{E}_s^\pi \left[ \left| \left( \sum_{t=0}^{n-1} \gamma^t (R(S_t, A_t) - \mathbb{E}_s^\pi [R(S_t, A_t)]) \right) \gamma^n (V(S_n) - \mathbb{E}_s^\pi [V(S_n)]) \right| \right] \\
&\leq \frac{2C_R \gamma^n}{1-\gamma} \mathbb{E}_s^\pi [(V(S_n) - \mathbb{E}_s^\pi [V(S_n)])^2] = \frac{2C_R \gamma^n}{1-\gamma} \mathbb{V}_s^\pi [V(S_n)].
\end{aligned}$$

□

Now suppose a policy evaluation algorithm is run with adaptive choice of  $n$ , i.e. in every update  $n$  is adapted to the situation. Then, in theory,  $n$  would be chosen large if the current estimate has large variance or large error, otherwise small. Thus, it seems plausible that a policy evaluation algorithm based on  $n$ -step TD updates might decrease  $n$  over time.

### 4.5.2 TD( $\lambda$ ) algorithms

<sup>11</sup> There is a nice-trick in temporal different learning (learning by resampling segments). Instead of using  $n$  steps for a fixed number  $n$  one mixes different  $n$  or, alternatively, chooses  $n$  random. Since the Markov property is compatible with memoryless random variables only, it might not be surprising that geometric distributions (the only memoryless distributions) play a role. Recall that an integer valued random variables is called geometric with parameter  $\lambda \in (0, 1)$  if  $\mathbb{P}(X = k) = (1 - \lambda) \frac{1}{\lambda^k}$  for  $k \in \mathbb{N}_0$ . One interpretation is to decide successively with probability  $\lambda$  to first stop at 0, 1, 2, and so on. The striking fact of TD( $\lambda$ ) 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( $\lambda$ ) and Monte Carlo turns out to be most effective.

<sup>11</sup>mache notation Geo( $\lambda$ ) or Geo(1- $\lambda$ ) kompatibel mit Stochastik 1 Skript

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( $\lambda$ ) 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 difference update

$$\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left( \sum_{t=0}^{n-1} \gamma^t R(S_{t+k}, A_{t+k}) + \gamma^n V_{\text{old}}(S_{t+n}) - V_{\text{old}}(S_t) \right).$$

Here is a first simple algorithm that can be seen as a rigorous version (instead of stopping at some large time) of first visit Monte Carlo estimation of  $V^\pi$  for MDPs with infinite horizon:

---

#### Algorithm 25: First visit Monte Carlo for non-terminating MDPs

---

**Data:** Policy  $\pi \in \Pi_S$ , initial condition  $\mu$ ,  $\lambda \in (0, 1)$

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

Initialize  $V_0$  (e.g.  $V_0 \equiv 0$ ).

$n = 0$

**while** *not converged* **do**

    Sample  $T \sim \text{Geo}(\lambda)$ .

    Determine stepsizes  $\alpha_{n+1}(s)$  for every  $s \in S$ .

    Generate trajectory  $(s_0, a_0, s_1, \dots)$  until  $T$  using policy  $\pi$ .

**for**  $t = 0, 1, 2, \dots, T$  **do**

**if**  $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$  **then**

$V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t) \left( \sum_{i=0}^{T-1} \gamma^i R(s_{t+i}, a_{t+i}) + \gamma^T V_n(s_T) - V_n(s_t) \right)$

**end**

**end**

$n = n + 1$

**end**

Return  $V_n$ .

---



**Theorem 4.5.2.** Suppose  $\pi \in \Pi_S$ ,  $\lambda \in (0, 1)$ , and  $\alpha$  satisfies the Robbins-Monro conditions. Then  $V_n(s) \rightarrow V^\pi(s)$  almost surely for all  $s \in S$ .

*Proof.* As always the convergence follows from Theorem 4.3.10 once the algorithm is reformulated with a suitable contraction and (here: unbiased) error. Let us first introduce the contraction that we will interpret in two ways:

$$F(V)(s) := \mathbb{E}_s^\pi \left[ \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left( \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n) \right) \right]$$

It is easy to see that  $F : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$  is a contraction:

$$\begin{aligned} \|F(V) - F(W)\|_\infty &= \max_{s \in S} \left| \mathbb{E}_s^\pi \left[ \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} (\gamma^n V(S_n) - \gamma^n W(S_n)) \right] \right| \\ &\leq \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} \max_{s \in S} \gamma^n |\mathbb{E}_s^\pi[(V(s_n) - W(s_n))]| \\ &\leq \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} \max_{s \in S} \gamma^n |V(s) - W(s)| \\ &= \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} \gamma^n \|V - W\|_\infty \\ &= \mathbb{E}[\gamma^X] \|V - W\|_\infty \end{aligned}$$

for  $X \sim \text{Geo}(\lambda)$ . Furthermore, the unique fixed point is  $V^\pi$ :

$$\begin{aligned} F(V^\pi)(s) &= \sum_{n=1}^{\infty} (1-\lambda) \lambda^{n-1} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V^\pi(S_n) \right] \\ &\stackrel{(4.16)}{=} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(s) \\ &= V^\pi(s) \end{aligned}$$

To prove convergence of the two algorithms we give two interpretations on how to sample from the expectation defining  $F(V)$ . The two ways of sampling from the expectation gives the two first-visit algorithms. By Fubini's theorem  $F(V)(s)$  is the expectation of a two-stage stochastic experiment: first sample  $T \sim \text{Geo}(\lambda)$  and then  $\sum_{t=0}^{T-1} (\gamma^t R(S_t, A_t) + \gamma^T V(S_T))$  for a sample of the MDP started in  $s$  independently of  $T$ . This is exactly what the algorithm does so the convergence is exactly the one from one-step (or  $n$ -step) stochastic approximation writing

$$V_{n+1}(s_0) = V_n(s_0) + \alpha_{n+1}(s_0) (F(V_n)(s_0) + \varepsilon_n(s_0) - V_n(s_0))$$

with errors  $\varepsilon_n(s_0) = (\sum_{t=1}^{T-1} \gamma^t R(s_t, a_t) - \gamma^T V_n(s_T)) - F(V_n)(s)$ . We only need to be careful with the filtration and define  $\mathcal{F}_n$  as the  $\sigma$ -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,  $\varepsilon_n$  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 bounded as we assume  $R$  to be bounded in these lecture notes.  $\square$

Next, we come to the  $\lambda$ -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).<sup>12</sup> The algorithm We are not going to prove the convergence. Instead, we prove the convergence for an equivalent algorithm, the so-called first visit TD( $\lambda$ ) backwards algorithm.



There is no way of turning a forwards algorithm into an online algorithm, an algorithm where the current trajectory gradually updates the values of  $V$  because all future values are in use.

The surprising fact is that the first visit  $\lambda$ -return algorithm can actually transformed into an equivalent forwards algorithm. This is the main beautiful idea of TD( $\lambda$ ).

<sup>12</sup>schreiben mit termination, oder diskutieren, dass nach termination alles 0 ist



**Algorithm 26:** First visit  $\lambda$ -return algorithm

---

**Data:** Policy  $\pi \in \Pi_S$ ,  $\lambda \in [0, 1]$   
**Result:** Approximation  $V \approx V^\pi$   
Initialize  $V_0$  (e.g.  $V_0 \equiv 0$ ).  
Set  $n = 0$ .  
**while** *not converged* **do**  
    Determine stepsizes  $\alpha_{n+1}(s)$  for every  $s \in S$ .  
    Generate trajectory  $(s_0, a_0, s_1, \dots)$  using policy  $\pi$ .  
    **for**  $t = 0, 1, 2, \dots$  **do**  
        **if**  $s_t \notin \{s_0, s_1, \dots, s_{t-1}\}$  **then**  
             $V_{n+1}(s_t) = V_n(s_t) + \alpha_{n+1}(s_t)(\sum_{k=0}^{\infty} (1-\lambda)\lambda^{n-1}(\sum_{i=0}^{n-1} \gamma^i R(s_{t+k}, a_{t+k}) + \gamma^n V_{\text{old}}(s_{t+n}) - V_{\text{old}}(s_t)))$   
        **end**  
    **end**  
     $n = n + 1$   
**end**  
Return  $V_n$ .

---

**TD( $\lambda$ ) backwards algorithms**

To turn the  $\lambda$ -return algorithm into a backwards algorithm (e.g. only states from the past are used for every future update) a little neat lemma is needed.



**Lemma 4.5.3.** Suppose  $s_0, a_0, s_1, a_1, \dots$  is a state-action sequence,  $\lambda \in (0, 1)$ , and  $V : \mathcal{S} \rightarrow \mathbb{R}$ , then

$$\begin{aligned} & \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} \left( \sum_{t=0}^{n-1} \gamma^t R(s_{t+k}, a_{t+k}) + \gamma^n V(s_{t+n}) - V(s_t) \right) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)). \end{aligned}$$

There are two interesting point of this representation of the TD( $\lambda$ ) update. First, the formula is more pleasant as one infinite sum dissappeared. Secondly, the new formula also works for  $\lambda = 0$  and  $\lambda = 1$ . Plugging-in yields exactly the formulas for one-step (use  $0^0 = 1$ ) and Monte Carlo value (use a telescopic sum argument) updates.

*Proof.* For the proof we use that  $\sum_{n=t}^{\infty} \lambda^n = \lambda^t \sum_{n=0}^{\infty} \lambda^n = \frac{\lambda^t}{1-\lambda}$  so that

$$\begin{aligned} & (1-\lambda) \left( \sum_{n=1}^{\infty} \lambda^{n-1} \left( \sum_{t=0}^{n-1} \gamma^t R(s_t, a_t) + \gamma^n V(s_n) - V(s_0) \right) \right) \\ &= (1-\lambda) \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \sum_{n=t+1}^{\infty} \lambda^{n-1} + (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \gamma^n V(s_n) - V(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + (1-\lambda)\gamma V(s_{t+1})) - V(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

The last equation can be checked by using a telescoping sum (write out the sums):

$$\begin{aligned} \sum_{t=0}^{\infty} (\gamma\lambda)^t \gamma(1-\lambda)V(s_{t+1}) - V_{\text{old}}(s_0) &= \sum_{t=1}^{\infty} (\gamma\lambda)^{t-1} \gamma V(s_t) - \sum_{t=1}^{\infty} (\gamma\lambda)^t V(s_t) - V_{\text{old}}(s_0) \\ &= \sum_{t=0}^{\infty} (\gamma\lambda)^t (\gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

□

What do we learn from the lemma? Instead of updating once the  $\lambda$ -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 27.

---

**Algorithm 27:** Offline TD( $\lambda$ ) 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**  $\tilde{s} \in \mathcal{S}$  **do**

**if**  $N(\tilde{s}) \geq 1$  **then**

$V_{n+1}(\tilde{s}) = V_n(\tilde{s}) + \alpha(\tilde{s})(\gamma\lambda)^{t-k(\tilde{s})} (R(s, a) + \gamma V_n(s') - V_n(\tilde{s}))$

**end**

**end**

$s = s'$ ,  $t = t + 1$

**end**

$n = n + 1$

**end**

Return  $V_n$ .

---



**Theorem 4.5.4.** Suppose  $\pi \in \Pi_S$ ,  $\lambda \in (0, 1)$ , and  $\alpha$  satisfies the Robbins-Monro conditions. Then  $V_n(s) \rightarrow V^\pi(s)$  almost surely for all  $s \in \mathcal{S}$  in Algorithms 26 and 27.

*Proof.* By Lemma 4.5.3 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 4.5.2.  $F$  is a contraction with unique fixed point  $V^\pi$ . As always the algorithm is rewritten into a asynchronous stochastic approximation update. Without loss of generality, we assume  $k(s) = 0$  if state  $s$  has been visited. Else, we can

shift the sum again. From Proposition 4.5.3 we get

$$\begin{aligned} V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} (\gamma\lambda)^t (R(s_t, a_t) + \gamma V_n(s_{t+1}) - V_n(s_t)) \\ &= V_n(s_0) + \alpha_n(s_0) \sum_{i=1}^{\infty} (1-\lambda)\lambda^{i-1} \sum_{t=0}^{i-1} (\gamma^t R(s_t, a_t) + \gamma^i V_n(s_i)) - V_n(s_0) \\ &= V_n(s_0) + \alpha_n(s_0) (F(V_n)(s_0) - V_n(s_0) + \varepsilon_n(s_0)), \end{aligned}$$

with  $F(V)$  from above and error-term

$$\varepsilon_n(s_0) := \sum_{i=1}^{\infty} (1-\lambda)\lambda^{i-1} \sum_{t=0}^{i-1} (\gamma^t R(s_t, a_t) + \gamma^i V_n(s_i)) - F(V_n)(s_0)$$

for every  $s \in S$ . Moreover, the error-term  $\varepsilon_n(s)$  fulfills

$$\mathbb{E}_s^\pi[\varepsilon_n(s) \mid \mathcal{F}_n] = (F(V_n))(s) - (F(V_n))(s) = 0.$$

Again, the errors are bounded as we assume  $R$  to be bounded. Hence, we got all assumptions for Theorem 4.3.10 and convergence towards the fixpoint  $V^\pi$ .<sup>13</sup>  $\square$

We can now derive another algorithm that is much more common in practice. Essentially, we use the same algorithm as before but instead use every-visit updates instead of first visit updates. Another nice simplification turns this into the famous TD( $\lambda$ ) algorithm with eligibility traces.



**Lemma 4.5.5.** Suppose  $s_0, a_0, s_1, a_1, \dots$  is a state-action sequence,  $\lambda \in (0, 1)$ , and  $V : S \rightarrow \mathbb{R}$ , then

$$\begin{aligned} &\sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\ &= \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \underbrace{\sum_{t=0}^k (\gamma\lambda)^{k-t} \mathbf{1}_{s_t=s_0}}_{=: e_k(s_0)}. \end{aligned}$$

*Proof.* The proof follows from a Fubini flip using the indicator  $\mathbf{1}_{k \geq t} = \mathbf{1}_{t \leq k}$ :

$$\begin{aligned} &\sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\ &= \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (\gamma\lambda)^{k-t}. \end{aligned}$$

$\square$

Let us go back to the first visit algorithm (27) that implements first visit updates. Replacing first visit updates

$$V_{n+1}(s_0) = V_n(s_0) + \mathbf{1}_{\{\text{first visit of } s_0 \text{ at } t\}} \alpha_{n+1}(s_0) \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k))$$

<sup>13</sup>both proofs are incomplete. since a state might not be visited in a rollout it might not be sampled in a run, thus,  $\varepsilon_n$  is not unbiased. Solution: Restart in unvisited states as long as all states have been visited. Maths ok, run-time nightmare.

by the every visit update yields

$$\begin{aligned}
 V_{n+1}(s_0) &= V_n(s_0) + \alpha_{n+1}(s_0) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \\
 &\stackrel{\text{Lemma 4.5.5}}{=} V_n(s_0) + \alpha_{n+1}(s_0) \sum_{k=0}^{\infty} (R(s_k, a_k) + \gamma V(s_{k+1}) - V(s_k)) \underbrace{\sum_{t=0}^k (\gamma\lambda)^{k-t} \mathbf{1}_{s_t=s_0}}_{=: e_k(s_0)}.
 \end{aligned}$$

Implemented as an Algorithm the update immediately yields Algorithm 28, backwards TD( $\lambda$ ) with eligibility traces.

---

**Algorithm 28:** Offline TD( $\lambda$ ) with eligibility traces

---

**Data:** Policy  $\pi \in \Pi_S$ ,  $\lambda \in [0, 1)$   
**Result:** Approximation  $V \approx V^\pi$   
 Initialize  $V_n$  (e.g.  $V_n \equiv 0$ ) for all  $n \in \mathbb{N}$ .  
 $N = 0$   
**while not converged do**  
   Initialize  $e(s) = 0$  for all  $s \in S$   
   Initialize  $s$ .  
   Determine step-sizes  $\alpha(s)$ ,  $s \in S$ , for next 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**

---

Before proving convergence of offline TD( $\lambda$ ) with eligibility traces let us quickly discuss what the algorithm does. In fact, the simple mechanism is certainly a main reason for its success. The term  $e_k(s)$  is called the eligibility trace at time  $k$  in state  $s$ . It determines the effect of the previous visits in state  $s$  on the current value. If  $s$  was visited at times  $t_1, t_2, \dots$ , the reward after the current visit in state  $s_k$  needs to be considered in the value function in state  $s$ . The first visit in  $s$  still has an effect on the current visit in  $s_k$  of  $(\gamma\lambda)^{k-t_1}$ , the second visit has a larger effect of  $(\gamma\lambda)^{k-t_2}$  and so on. The first effects will vanish in the limit  $k \rightarrow \infty$ . In this algorithm it is important to note that we still use  $V_N$  until we are in a terminal state for the update with  $\Delta$ . Thus, the algorithm also does not bootstrap information of the beginning of a trajectory to later times.



There is an online version, see Algorithm 29, of TD( $\lambda$ ) 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  $\alpha$  (which?) this is the every visit Monte Carlo estimator.

This is because we can see in equation ?? that the value function is updated only when the trajectory ends and not in between. Note that for a functioning algorithm terminating MDPs are required then. For a similar convergence proof as ??, we necessarily require terminating MDPs. We need finite expected visits in each non-terminating state such that the update in ?? stays finite:

For every  $s \in S$ , let  $0 < m(s) := \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] < \infty$  and  $\tilde{\alpha}_n(s) := \alpha_n(s)m(s)$ . Then we can rewrite the every visit update scheme in the mathematical form

$$\begin{aligned} & V_{N+1}(s) \\ &= V_N(s) + \tilde{\alpha}_n(s) \left[ \frac{1}{m(s)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) - V_N(s) \right]. \end{aligned}$$

In this way backwards TD( $\lambda$ ) can be interpreted as stochastic approximation algorithm with step-sizes  $\tilde{\alpha}_n(s)$  that satisfy the Robbins-Monro condition if and only if  $\alpha_n(s)$  do.



**Theorem 4.5.6. (Convergence of TD( $\lambda$ ) with every-visit updates)**

Let for all  $s \in S$  with the first visit in  $k(s)$  the update of  $V$  be

$$V_{N+1}(s) = V_N(s) + \alpha_N(s) \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s} \sum_{k=t}^{\infty} ((\gamma\lambda)^{k-t} R(s_k, a_k) + \gamma V_N(s_{k+1}) - V_N(s_k)).$$

with  $\lambda < 1$  and the trajectory  $(s_0, a_0, s_1, \dots)$  sampled according to the policy  $\pi$  and  $m(s) < \infty$  for every  $s \in S$ .

Suppose that all states are visited infinitely often in the algorithm such that the step-sizes are adapted and satisfy the Robbins-Monro conditions are satisfied:

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \quad \text{a.s.} \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s) < \infty \quad \text{a.s.}$$

for every  $s \in S$ . Moreover, let the rewards be bounded and  $0 < \gamma < 1$ . Then  $\lim_{n \rightarrow \infty} V_N(s) = V^\pi(s)$  almost surely, for every state  $s \in S$ .

*Proof.* The filtration  $\mathcal{F}_N$  is defined to be generated by all random variables needed to for the  $N$ st rollout. As always we rewrite the update scheme into an asynchronous stochastic approximation scheme:

$$\begin{aligned} & V_{N+1}(s_0) \\ &= V_N(s_0) + \tilde{\alpha}_n(s_0) \left( \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) - V_N(s_0) \right) \\ &= V_N(s_0) + \tilde{\alpha}_n(s_0) (F(V_N)(s_0) - V_N(s_0) + \varepsilon_N(s_0)) \end{aligned}$$

with

$$F(V)(s) := \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V_N(S_{t+n})) \right]$$

and

$$\begin{aligned} \varepsilon_N(s_0) &:= \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(s_k, a_k) + \gamma^n V_N(s_{t+n})) \\ &\quad - F(V_N)(s_0) + V_N(s_0) - \frac{1}{m(s_0)} \sum_{t=0}^{\infty} \mathbf{1}_{s_t=s_0} V_N(s_0) \end{aligned}$$

Similarly to the previous proof,  $V^\pi$  is a fixed point of  $F$  because

$$\begin{aligned}
F(V^\pi)(s) &= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})) \right] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \mathbb{E}_s^\pi \left[ \sum_{k=t}^{t+n-1} (\gamma^{k-t} R(S_k, A_k) + \gamma^n V^\pi(S_{t+n})) \mid (S_t, A_t) \right] \right] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(S_t) \right] \\
&= \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V^\pi(s) \\
&= \frac{m(s)}{m(s)} V^\pi(s) = V^\pi(s)
\end{aligned}$$

for every  $s \in S$ . Similarly to the previous proof we also obtain that  $F$  is a contraction:

$$\begin{aligned}
\|F(V) - F(W)\|_\infty &= \max_{s \in S} \left| \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} (\gamma^n V(S_{t+n}) - \gamma^n W(S_{t+n})) \right] \right| \\
&\leq \max_{s \in S} \frac{1}{m(s)} \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \gamma^n \|V - W\|_\infty \right] \\
&\leq \max_{s \in S} \frac{1}{m(s)} \mathbb{E}[\gamma^X] \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} \mathbf{1}_{S_t=s} \right] \|V - W\|_\infty = \mathbb{E}[\gamma^X] \|V - W\|_\infty
\end{aligned}$$

for  $X \sim \text{Geo}(\lambda)$ . The error term  $\varepsilon_N(s)$  fulfills

$$\mathbb{E}_s^\pi[\varepsilon_N(s) \mid \mathcal{F}_N] = (F(V_N))(s) - (F(V_N))(s) - \frac{1}{m(s)} m(s) V_N(s) + V_N(s) = 0.$$

As the rewards are bounded, we also get

$$\mathbb{E}_s^\pi[\varepsilon_N^2(s) \mid \mathcal{F}_N] \leq C \quad \forall N \in \mathbb{N}, s \in S.$$

So, we got all assumptions for Theorem 4.3.10 and convergence towards the fixpoint  $V^\pi$ .  $\square$

The algorithm can also be modified by not waiting until termination before updating. Then,  $V_N$  is directly updated after each step. This is called online updating: We do not wait until the trajectory is terminated, but use a new version of  $V$  every time it has been updated. Using this update scheme, we can not use the forward and the backward view equivalently anymore: Instead to wait for the future rewards and update afterwards (forward view), we update the effect of previous states directly with a new version of the value function. The algorithm according to this scheme is now given by:

The convergence of the both versions of TD( $\lambda$ ) 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  $V^\pi$  (or  $Q^\pi$  or  $Q^*$ ) and immediately got an algorithm as approximate fixed point iteration. Here the approach got reversed. A very simple algorithm is written down and then proved to converge by rewriting into a very tricky contraction operator.

**Algorithm 29:** Online backwards TD( $\lambda$ ) with eligibility traces

---

**Data:** Policy  $\pi \in \Pi_{\mathcal{S}}$ ,  $\lambda \geq 0$   
**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**  $\tilde{s} \in \mathcal{S}$  **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**

---

**TD( $\lambda$ ) for Control**

To be written... We will only adapt the ideas of the last section to control algorithms now. No proofs - just algorithms!

**SARSA( $\lambda$ )****Q( $\lambda$ )**

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.  $\epsilon$ -greedy. So, if we play the greedy action in many cases, we can derive Q( $\lambda$ ) as well.

**4.6 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.