Reinforcement Learning

Summary
Fabian Damken
July 25, 2022

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In this course we will look at lots of methods from the domain of *reinforcement learning (RL)*. RL is an approach for agent-oriented learning where the agent learns by repeatedly acting with the environment and from rewards. Also, it does not know how the world works in advance. RL is therefore close to how humans learn and tries to tackle the fundamental challenge of artificial intelligence (AI):

> "The fundamental challenge in artificial intelligence and machine learning is learning to make good decisions under uncertainty." (Emma Brunskill)

RL is so general that every AI problem can be phrased in its framework of learning by interacting. However, the typical setting is that at every time step, an agent perceives the state of the environment and chooses an action based on these perceptions. Subsequently, the agent gets a numerical reward and tries to maximize this reward by finding a suitable strategy. This procedure is illustrated in Figure 1.1.

1.1 Artificial Intelligence

The core question of AI is how to build "intelligent" machines, requiring that the machine is able to adapt to its environment and handle unstructured and unseen environments. Classically, AI was an "engine" producing answers to various queries based on rules designed by a human expert in the field. In (supervised) machine learning (ML), the rules are instead learned from a (big) data set and the "engine" produces answers based on the data. However, this approach (leaning from labeled data) is not sufficient for RL as demonstrations might be imperfect, the correspondence problem, and that we cannot demonstrate everything. We can break these issues down as follows: supervised learning does not allow "interventions" (trial-and-error) and evaluative feedback (reward).

The core idea leading to RL was to not program machines to simulate an adult brain, but to simulate a child's brain that is still learning. RL formalizes this idea of intelligence to interpret rich sensory input and choosing complex actions. We know that this may be possible as us humans do it all the time. This lead to the RL view on AI depicted in Figure 1.1 and is based on the hypothesis that learning from a scalar reward is sufficient to yield intelligent behavior (Sutton and Barto, 2018).

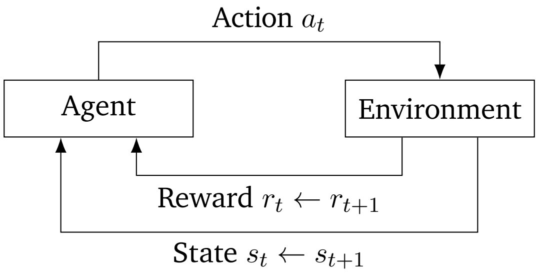

Action a_t
Agent
Environment
Reward $r_t \leftarrow r_{t+1}$
State $s_t \leftarrow s_{t+1}$

Figure 1.1: The Reinforcement Learning Cycle

	actions *do not* change the state of the world	actions change the state of the world
no model	(Multi-Armed) Bandits	Reinforcement Learning
known model	Decision Theory	Optimal Control, Planning

Table 1.1: Problem Classification

1.2 Reinforcement Learning Formulation

RL tries to *maximize the long-term reward* by finding a strategy/policy with the general assumption that it is easier to assess a behavior by specifying a cost than specifying the behavior directly. In general, we have the following things different to most (un)supervised settings:

- no supervision, but only a reward signal
- feedback (reward) is always delayed and not instantaneous
- time matters, the data is sequential and by no means i.i.d.
- the agent's actions influence the subsequent data, i.e., the agent generates its own data

In addition to this, RL is challenged by a numerous complicated factors and issues, e.g., dynamic state-dependent environments, stochastic and unknown dynamics and rewards, exploration vs. exploitation, delayed rewards (how to assign a temporal credit), and very complicated systems (large state spaces with unstructured dynamics). For designing an RL-application, we usually have to choose the state representation, decide how much prior knowledge we want to put into the agent, choose an algorithm for learning, design an objective function, and finally decide how we evaluate the resulting agent. By all these decisions, we want to reach a variety of goals, e.g., convergence, consistency, good generalization abilities, high learning speed (performance), safety, and stability. However, we are usually pretty restricted in terms of computation time, available data, restrictions in the way we act (e.g., safety constraints), and online vs. offline learning.

This sounds like a lot and, in fact, is! We therefore often limit ourselves onto specific (probably simpler) sub-problems and solve them efficiently under some assumptions. Some common flavors of the RL problem are, for instance:

- *Full:* no additional assumptions, the agent can only probe the environment through the state dynamics and its actions; the agent has to understand the environment
- *Filtered State and Sufficient Statistics:* assumption of a local Markov property (i.e., the next state only depends on the current state and action, and not on the past), decomposable rewards (into specific time steps); we can show that every problem is a (probably infinite) instance of this assumption, but how to filter the state to get such properties?
- *Markovian Observable State:* assume that we can observe the state fulfilling the Markov property directly
- *Further Simplifications:* contextual bandits (the dynamics do not depend on the action or the past and current state at all); bandits (only a single state)

We can summarize the different RL-like problems in a matrix, see Table 1.1.

1.2.1 Components

To solve an RL problem, we need three ingredients:

1. Model Learning
    - we want to approximate and learn the state transfer using methods from supervised learning
    - need to generate actions for model identification
    - estimation of the model or the model's parameters
2. Optimal Control/Planning
    - generation of optimal control inputs
3. Performance Evaluation

1.3 Wrap-Up

- why RL is crucial for AI and why all other approaches are ultimately doomed
- background and characteristics of RL
- classification of RL problems
- core components of RL algorithms

2 Preliminaries

In this chapter we cover some preliminaries that are necessary for understanding the rest of the course. Note that most of this content is dense and should be used as a reference throughout this course as oppose to an actual introduction to the topic.

2.1 Functional Analysis

Definition 1 (Normed Vector Space)**.** A *normed vector space* is a vector space $\mathcal{X}$ over X equipped with a *norm* $\|\cdot\| : \mathcal{X} \to \mathbb{R}$ that has the following properties:

1. $\|x\| \geq 0$ for all $x \in \mathcal{X}$ and $\|x\| = 0$ iff $x = 0$ (non-negativity)
2. $\|\alpha x\| = |\alpha|\|x\|$ for all $\alpha \in X$ and $x \in \mathcal{X}$ (homogenity)
3. $\|x_1 + x_2\| \leq \|x_1\| + \|x_2\|$ for all $x_1, x_2 \in \mathcal{X}$ (triangle inequality)

For the rest of this course we usually use real finite-dimensional vectors spaces $\mathcal{X} = \mathbb{R}^d$, $d \in \mathbb{N}^+$, the L_∞-norm $\|\cdot\|_\infty$, and (weighted) L_2-norms $\|\cdot\|_{2,\rho}$.

Definition 2 (Complete Vector Space)**.** A vector space $\mathcal{X}$ is *complete* if every Cauchy sequence[1] in $\mathcal{X}$ has a limit in $\mathcal{X}$.

Definition 3 (Contraction Mapping)**.** Let $\mathcal{X}$ be a vector space equipped with a norm $\|\cdot\|$. An operator $T : \mathcal{X} \to \mathcal{X}$ is called an *α-contraction mapping* if $\exists \alpha \in [0, 1) : \forall x_1, x_2 \in \mathcal{X} : \|Tx_1 - Tx_2\| \leq \alpha\|x_1 - x_2\|$. If only $\exists \alpha \in [0, 1] : \forall x_1, x_2 \in \mathcal{X} : \|Tx_1 - Tx_2\| \leq \alpha\|x_1 - x_2\|$, T is called *non-expanding*.

Definition 4 (Lipschitz Continuity)**.** Let $\mathcal{X}$ and $\mathcal{Y}$ be vector spaces equipped with norms $\|\cdot\|_X$ and $\|\cdot\|_Y$, respectively. A function $f : \mathcal{X} \to \mathcal{Y}$ is called *Lipschitz-continuous* if $\exists L \geq 0 : \forall x_1, x_2 \in \mathcal{Y} : \|f(x_1) - f(x_2)\|_Y \leq L\|x_1 - x_2\|_X$.

Remark 1. *Obviously, every contraction mapping is also Lipschitz-continuous with Lipschitz-constant $L \triangleq \alpha$ and is therefore continuous. Also, the product of two Lipschitz-continuous mappings is Lipschitz-continuous and therefore $T^n = T \circ \cdots \circ T$ is Lipschitz-continuous, too.*

Definition 5 (Fixed Point)**.** Let $\mathcal{X}$ be a vector space equipped and let $T : \mathcal{X} \to \mathcal{X}$ be an operator. Then $x \in \mathcal{X}$ is a *fixed point* of T if $Tx = x$.

Theorem 1 (Banach Fixed Point Theorem)**.** *Let $\mathcal{X}$ be a complete vector space with a norm $\|\cdot\|$ and let $T : \mathcal{X} \to \mathcal{X}$ be an α-contraction mapping. Then T has a unique fixed point $x^* \in \mathcal{X}$ and for all $x_0 \in \mathcal{X}$ the sequence $x_{n+1} = Tx_n$ converges to x^* geometrically, i.e., $\|x_n - x^*\| \leq \alpha^n\|x_0 - x^*\|$.*

[1] This section is already overflowing with mathematical rigor compared to the rest of the course, so we will skip the definition of a Cauchy sequence here.

2.2 Statistics

This section introduces some concepts of statistics, but you should

2.2.1 Monte-Carlo Estimation

Let X be a random variable with mean $\mu = \mathbb{E}[X]$ and variance $\sigma^2 = \mathrm{Var}[X]$ and let $\{x_i\}_{i=1}^n$ be i.i.d. realizations of X. We then have the *empirical mean* $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^n x_i$ and we can show that $\mathbb{E}[\hat{\mu}_n] = \mu$ and $\mathrm{Var}[\hat{\mu}_n] = \sigma^2/n$. Also, if the sample size n goes to infinity, we have the *strong* and *weak law of large numbers,* respectively:

$$P\big(\lim_{n\to\infty} \hat{\mu}_n = \mu\big) = 1 \qquad\qquad \lim_{n\to\infty} P\big(|\hat{\mu}_n - \mu| > \epsilon\big) = 0$$

Also, we have the *central limit theorem:* no matter the distribution of P, its mean value converges to a normal distribution, $\sqrt{n}(\hat{\mu}_n - \mu) \overset{D}{\rightarrow} \mathcal{N}(0, \sigma^2)$.

2.2.2 Bias-Variance Trade-Off

When evaluating/training a ML model, the error is due to two factors (illustrated in Figure 2.1):

- *bias,* i.e., the distance to the expected prediction
- *variance,* i.e., the variability of a prediction for a given data point

In general, we want to minimize both, but we can only minimize one of them! This is known as the *bias-variance trade-off.*

2.2.3 Important Sampling

If we want to estimate the expectation of some function $f(x)$ for $x \sim p(x)$, but cannot sample from $p(x)$ (which is often the case for complicated models), we can instead use the following relation(s):

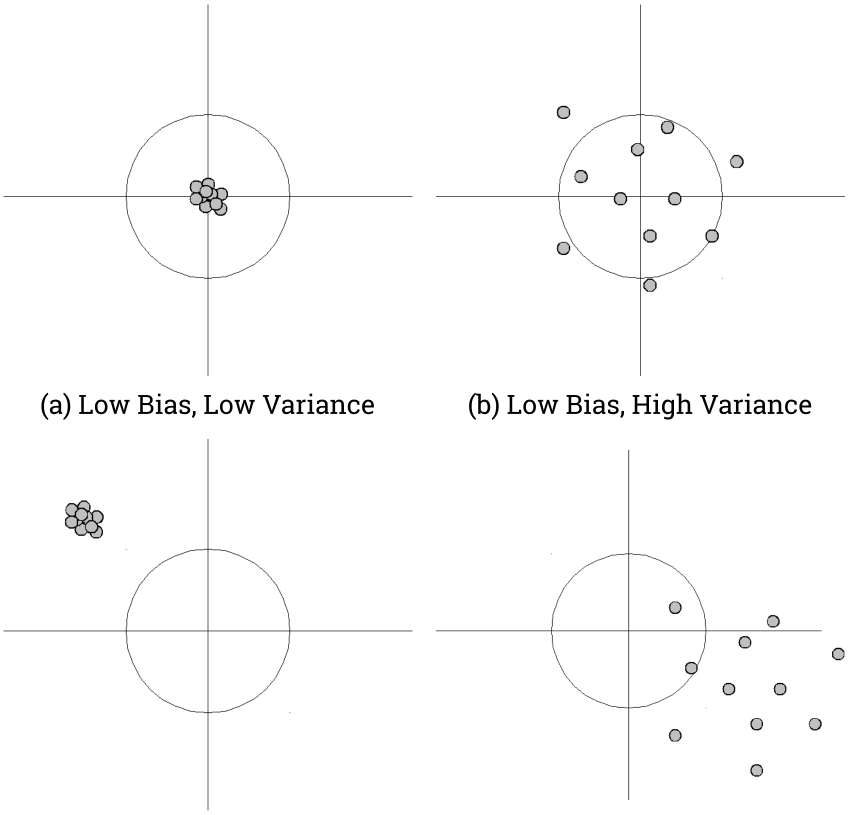
$$\mathbb{E}_{x\sim p}\big[f(x)\big] = \sum_x f(x)p(x) = \sum_x f(x)\frac{p(x)}{q(x)}q(x) = \mathbb{E}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$
$$\mathbb{E}_{x\sim p}\big[f(x)\big] = \int f(x)p(x)\,\mathrm{d}x = \int f(x)\frac{p(x)}{q(x)}p(x)\,\mathrm{d}x = \mathbb{E}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

and sample from a surrogate distribution $q(x)$. This approach obviously has problems if q does not cover p sufficiently well along with other problems. See Bishop, 2006, Chapter 11 for details.

2.2.4 Linear Function Approximation

A basic approximator we will need often is the linear function approximator $f(\boldsymbol{x}) = \boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x})$ with weights $\boldsymbol{w}$ and features $\boldsymbol{\phi}(\boldsymbol{x})$. As the weights are optimized and the features are designed, we have lots of variability here. Actually, constructing useful features is the influential step on the approximation quality. Most importantly, features are the only point where we can introduce interactions between different dimensions. A good representations therefore captures all dimensions and all (possibly complex) interaction.

We will now go over some frequently used features.



(a) Low Bias, Low Variance

(b) Low Bias, High Variance

(c) High Bias, Low Variance

(d) High Bias, High Variance

Figure 2.1: Bias-Variance Trade-Off; Source: Bernhard Thiery (CC BY-SA 3.0)

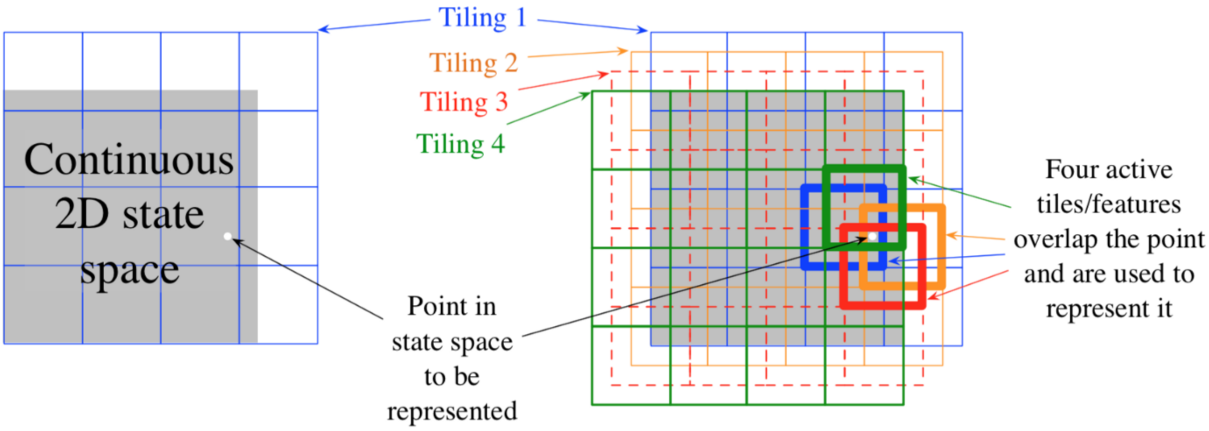


Figure 2.2: Tile Coding; Source: `https://towardsdatascience.com/reinforcement-learning-tile-coding-implementation-7974b600762b`

Polynomial Features *Polynomial features* are particularly simple and capture the interaction between dimensions by multiplication. For instance, the first- and second-order polynomial features of a two-dimensional state $\boldsymbol{x} = (x_1, x_2)^\top$ are:

$$\boldsymbol{\phi}_{P1}(\boldsymbol{x}) = (1, x_1, x_2, x_1x_2)^\top \qquad \boldsymbol{\phi}_{P2}(\boldsymbol{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1x_2^2, x_1^2x_2, x_1^2, x_2^2)$$

However, the number of features grows *exponentially* with the dimension!

Fourier Basis Fourier series can be used to approximate periodic functions by adding sine and cosine waves with different frequencies and amplitudes. Similarly, we can use them for general function approximation of functions with bounded domain. As it is possible to approximate any even function with just cosine waves and we are only interested in bounded domains, we can set this domain to positive numbers only and can therefore approximate any function. For one dimension, the n-th order *Fourier (cosine) basis* is

$$\phi_m(x) = \cos(\pi m \tilde{x}), \quad m = 0, 1, \dots, n.$$

and $\tilde{x}$ is a normalized version of x, i.e., $\tilde{x} = (x - x_{\text{max}})/(x_{\text{max}} - x_{\text{min}})$.

Coarse Coding *Coarse coding* divides the space into M different regions and produced M-dimensional coding features for which the j-th entry is 1 iff the data point lies withing the respective region; all values the data point does not lie in are 0. Features with this codomain are also called *sparse*.

Tile Coding *Tile coding* is a computationally efficient form of coarse coding which use square *tilings* of space. It uses N tilings, each composed of M tiles. The features "vector" is then an $N \times M$ matrix where a single value is 1 iff x lies inside the tile and 0 otherwise. Figure 2.2 shows an illustration of this coding.

Radial Basis Functions *Radial basis functions (RBFs)* are a generalization of coarse coding where the features are in the interval $(0, 1]$. A typical RBF is the Gaussian

$$\phi_j(\boldsymbol{x}) = \exp\left\{ -\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|_2^2}{2\sigma_j^2} \right\}$$

with center $\boldsymbol{c}_j$ and bandwidth σ_j^2.

Neural Networks A very powerful alternative to hand-crafting features are *neural networks (NNs)*. By stacking multiple layers of learned features, they are very powerful prediction machines.

2.2.5 Likelihood-Ratio Trick

Suppose we need to differentiate the expectation of some function $f(x)$ w.r.t. θ where $x \sim p_\theta(\cdot)$. However, we cannot directly calculate $\mathbb{E}_{x\sim p_\theta}[f(x)]$ or "differentiate through sampling." Instead, we can use the identity

$$\frac{\mathrm{d}}{\mathrm{d}z}\log h(z) = \frac{h'(z)}{h(z)} \qquad \implies \qquad f'(z) = h(z)\,\frac{\mathrm{d}}{\mathrm{d}z}\log h(z)$$

to reformulate the derivative of the expectation as

$$\frac{\partial}{\partial\theta}\mathbb{E}_{x\sim p_\theta}[f(x)] = \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)\,\mathrm{d}x = \int f(x)\left(\frac{\partial}{\partial\theta}p_\theta(x)\right)p_\theta(x)\,\mathrm{d}x = \mathbb{E}_{x\sim p_\theta}\left[f(x)\,\frac{\partial}{\partial\theta}p_\theta(x)\right].$$

While this is a very powerful approach, the gradient estimator exhibits high variance!

2.2.6 Reparametrization Trick

Suppose we need to differentiate the expectation of some function $f(x)$ w.r.t. θ where $x \sim p_\theta(\cdot)$. However, we cannot directly calculate $\mathbb{E}_{x\sim p_\theta}[f(x)]$ or "differentiate through sampling." Instead, we reformulate the expectation with a function $x = g_\theta(\varepsilon)$ that separates the random components ε from the deterministic ones θ such that we can reparameterize the expectation as

$$\mathbb{E}_{x\sim p_\theta}[f(x)] = \mathbb{E}_\varepsilon\big[f\big(g_\theta(\varepsilon)\big)\big].$$

For instance, if $p_\theta(x) = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$ is a Gaussian, $g_\theta(\varepsilon) = \mu_\theta + \sigma_\theta\varepsilon$ with $\varepsilon \sim \mathcal{N}(0,1)$. We can now simply use the chain rule to take the derivative w.r.t. θ. Compared to the likelihood-ratio trick, this estimator has less variance!

2.3 Miscellaneous

Finally, this section contains all the stuff that does not fit into the categories before.

2.3.1 Useful Integrals

The following hold for a distribution $p_\theta(x)$:

$$\int \frac{\partial}{\partial\theta}p_\theta(x)\,\mathrm{d}x = 0 \qquad\qquad \int \frac{\partial}{\partial\theta}\log p_\theta(x)\,\mathrm{d}x = \int \frac{\frac{\partial}{\partial\theta}p_\theta(x)}{p_\theta(x)}\,\mathrm{d}x = 0$$

The first identity can be shown by swapping the integral and derivative and using the normalization condition of probability densities. For the second we use integration by parts with $f' = \frac{\partial}{\partial\theta}p_\theta(x)$, for which $f = 0$ due to the first integral. Hence, the second follows.

3 Markov Decision Processes and Policies

In this chapter we will develop the groundwork for all upcoming chapters and define some important mathematical concepts.

3.1 Markov Decision Processes

A *Markov decision process (MDP)* describes the environment for RL *formally* for the case where we can fully observe the environment, i.e., we directly "see" the state. Also, the current state fully characterized the system and future states are independent from the past *(Markov property)*. This mathematical framework allows precision and rigorous reasoning on, for instance, optimal solutions and convergence (note, however, that we will only touch the tip of the iceberg in theoretical analysis and we will be less rigorous than some mathematician may wish). The nice this of MDPs is there wide applicability: we can frame almost all RL problems as MDPs. Most of the remaining chapter here focuses on fully observable and finite MDPs, i.e., the number of states and actions is finite. Table 3.1 shows an overview over different Markov models.

We now went over some mathematical definitions for building up the "Markovian framework."

Definition 6 (Markov Property)**.** A stochastic process X_t is *Markovian* or *fulfills the Markov property* if $P_t(S_{t+1} = s' \mid S_t = s, S_{t-1} = k_{t-1}, \dots, S_0 = k_0) = P_t(S_{t+1} = s' \mid S_t = s)$ for all t.

Definition 7 (Stationary Transition Probabilities)**.** If $P_t(S_{t+1} = s' \mid S_t = s)$ is time invariant, $p_{ss'} \coloneqq P_t(S_{t+1} = s' \mid S_t = s)$ are the *stationary transition probabilities.*

Definition 8 (State Transition Matrix)**.** With the transition probabilities $p_{ss'}$, let $\boldsymbol{P}_{ss'} \coloneqq p_{ss'}$ for all s, s' be the *transition matrix.*

Definition 9 (Markov Chain)**.** A *Markov chain* is a tuple $\langle \mathcal{S}, \boldsymbol{P}, \iota \rangle$ with the (finite) set of discrete-time states $S_t \in \mathcal{S}$, $n \coloneqq |\mathcal{S}|$, transition matrix $\boldsymbol{P} \in [0,1]^{n \times n}$, and the initial state distribution $\iota_i = P(S_0 = i)$.

Definition 10 (Probability Row Vector)**.** The vector $\boldsymbol{p}_t \coloneqq \sum_{i=1}^{n} P(S_t = i)\boldsymbol{e}_i^\top$ with the i-th unit vector $\boldsymbol{e}_i$ and includes the probability of being in the i-th state at time step t.

Theorem 2 (Chapman-Kolmogorov for Finite Markov Chains)**.** *The probability row vector $\mathbf{p}_{t+k}$ at time step $t + k$ starting from $\mathbf{p}_t$ at time step t is given by $\mathbf{p}_{t+k} = \mathbf{p}_t \boldsymbol{P}^k$.*

	All states observable?	
Actions?	Yes	No
Yes	Markov Decision Process	Partially Observable MDP
No	Markov Chain	Hidden Markov Model

Table 3.1: Types of Markov Models

Proof. Assume w.l.o.g. $t = 0$ We proof this by induction. For the base case, let $k = 1$. Let $\boldsymbol{p}_0 = (p_{0,1}, p_{0,2}, \dots, p_{0,n})$ be an arbitrary probability row vector. By linearity, we have

$$\boldsymbol{p}_0 \boldsymbol{P}_{ss'} = \sum_{i=1}^{n} p_{0,1} \boldsymbol{e}_i^\top \boldsymbol{P} = \sum_{i=1}^{n} p_{0,1} \boldsymbol{P}_i$$

where $\boldsymbol{P}_i$ is the i-th row of $\boldsymbol{P}$. Rewriting this equation in terms of explicit transition probabilities, we have

$$\begin{aligned} &= \sum_{i=1}^{n} P(S_0 = i) \sum_{j=1}^{n} \boldsymbol{e}_j^\top P(S_1 = j \mid S_0 = i) = \sum_{j=1}^{n} \boldsymbol{e}_j^\top \sum_{i=1}^{n} P(S_0 = i) P(S_1 = j \mid S_0 = i) \\ &= \sum_{j=1}^{n} \boldsymbol{e}_j^\top \sum_{i=1}^{n} P(S_1 = j, S_0 = i) = \sum_{j=1}^{n} \boldsymbol{e}_j^\top P(S_1 = j) = \sum_{j=1}^{n} p_{1,j} \boldsymbol{e}_j^\top = \boldsymbol{p}_1. \end{aligned}$$

The first equality is due to the definition of $\boldsymbol{P}_i$, the third is due to the definition of conditional probabilities, the fourth is due to marginalizing out S_0, and the final is just another application of the definit ion of the probability row vector. For the induction step $k \to k + 1$, assume that $\boldsymbol{p}_k = \boldsymbol{p}_t \boldsymbol{P}^k$ holds for some k. We then have $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k \boldsymbol{P} = \boldsymbol{p}_0 \boldsymbol{P}^k \boldsymbol{P} = \boldsymbol{p}_0 \boldsymbol{P}^{k+1}$ where the first equality is due to the base case and the second is due to the induction hypothesis. □

Definition 11 (Steady State)**.** A probability row vector $\boldsymbol{p}$ is called a *steady* state if an application of the transition matrix does not change it, i.e., $\boldsymbol{p} = \boldsymbol{p}\boldsymbol{P}$.

Remark 2. *While the steady state is in general not independent of the initial state (consider, for instance, $\boldsymbol{P} = \boldsymbol{I}$), it gives insights in which states of the Markov chain are visited in the long run.*

Definition 12 (Absorbing, Ergodic, and Regular Markov Processes)**.** A Markov process is called …

- …*absorbing* if it has at least one *absorbing state* (i.e., a state that can never be left) and if that state can be reached from every other state (not necessarily in one step).
- …*ergodic* if all states are *recurrent* (i.e., visited an infinite number of times) and *aperiodic* (i.e., visited without a systematic period).
- …*regular* if some power of the transition matrix has only positive (non-zero) elements.

3.1.1 Example

3.2 Markov Reward Processes

Definition 13. Markov Reward Process A *Markov reward process* is a tuple $\langle \mathcal{S}, \boldsymbol{P}, R, \gamma, \iota \rangle$ with the (finite) set of discrete-time states $S_t \in \mathcal{S}$, $n := |\mathcal{S}|$, transition matrix $\boldsymbol{P}_{ss'} = P(s' \mid s)$, reward function $R : \mathcal{S} \to \mathbb{R} : s \mapsto R(s)$, discount factor $\gamma \in [0, 1]$, and the initial state distribution $\iota_i = P(S_0 = i)$. We call $r_t = R(s_t)$ the immediate reward at time step t.

3.2.1 Time Horizon, Return, and Discount

Note that in 13 we did not clearly specify how the reward is computed. Especially we did not define how much time steps the reward “looks” into the future. For this we generally have three options: finite, indefinite, and infinite. The first computes the reward for a fixed and finite number of steps, the second until some stopping criteria is met, and the third infinitely.

Definition 14 (Cumulative Reward)**.** The *cumulative reward* summarizes the reward signals of a Markov reward process (MRP). We define the following:

$$J_t^{\text{total}} := \sum_{k=1}^{T} r_{t+k} \qquad J_t^{\text{average}} := \frac{1}{T} \sum_{k=1}^{T} r_{t+k} \qquad J_t \equiv J_t^{\text{discounted}} := \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k,$$

For an infinite horizon, we take the limit of these as $T \to \infty$.

Theorem 3. *The cumulative discounted reward fulfills the recursive relation* $J_t = r_{t+1} + \gamma J_{t+1}$.

Proof. $J_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k = r_{t+1} + \sum_{k=t+2}^{T} \gamma^{k-t-1} r_k = r_{t+1} + \gamma \sum_{k=t+2}^{T} \gamma^{k-t-2} r_k = r_{t+1} + \gamma J_{t+1}$ □

Definition 15 (Return)**.** The *return* $J(\tau)$ of a trajectory $\tau = (s_t)_{t=1}^{T}$ is the discounted reward $J(\tau) := J_0(\tau)$.

Remark 3. *The infinite horizon discounted cumulative reward for* $r_t = 1$ *(for all t) is a geometric series and we have* $J_t = \lim_{T\to\infty} \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k = \sum_{k=0}^{\infty} \gamma^k = 1/(1-\gamma)$ *for* $\gamma < 1$*. If the reward is lower/upper bounded by* $r_{\min}/r_{\max}$*, we have* $J_t \in \big[r_{\min}/(1-\gamma), r_{\max}/(1-\gamma)\big]$*. Similarly, the return is lower/upper-bounded.*

We can interpret the discount factor γ as a "measure" how important future rewards are to the current state (how delayed vs. immediate the reward is). For instance, $\gamma \approx 0$ yields myopic evaluation and $\gamma \approx 1$ yields far-sighted evaluation. An alternative interpretation is that the discount factor is the probability that the process continues (such that the discounted return is the expected return w.r.t. the discount factor). Despite the obvious advantage that including a discount factor prevents the return from diverging, we also have a couple of other reasons why it makes sense to weigh future rewards less:

- we might be *uncertain* about the future (e.g., with imperfect) models
- if the reward is *financial,* immediate rewards earn more interest than delayed rewards
- *animal and human behavior* also shows preference for immediate rewards—and why try to mimic biology in the end

However, sometimes we still use *undiscounted* MRPs (i.e., $\gamma = 1$), for instance if all sequences are guaranteed to terminate.

3.2.2 Value Function

Definition 16 (Value Function for MRP)**.** The *state value function* for a MRP is $V(s) := \mathbb{E}_{\boldsymbol{P}}[J_t \mid s_t = s]$ for any t.That is, the *expected* return starting from state s where the expectation is w.r.t. the state dynamics.

Theorem 4 (Bellman Equation)**.** *For all states* $s \in \mathcal{S}$*, we have* $V(s) = R(s) + \gamma \mathbb{E}\big[V(s_{t+1}) \,\big|\, s_t = s\big]$.

Proof. $V(s) = \mathbb{E}\big[J_t \,\big|\, s_t = s\big] = R(s) + \gamma \mathbb{E}\big[J_{t+1} \,\big|\, s_t = s\big] = R(s) + \gamma \mathbb{E}\big[V(s_{t+1}) \,\big|\, s_t = s\big]$ □

The Bellman equation allows us to decompose the value of any state into its immediate reward and the value of the subsequent states (in expectation). As we only consider discrete MRPs, we can also express the Bellman equation in matrix form,

$$\boldsymbol{V} = \boldsymbol{R} + \gamma \boldsymbol{P}\boldsymbol{V} \qquad \Longleftrightarrow \qquad \boldsymbol{V} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} \boldsymbol{R}, \tag{3.1}$$

where $\boldsymbol{V}$ and $\boldsymbol{R}$ are columns vectors with the values and rewards, respectively, and $\boldsymbol{P}$ is the transition matrix. We can therefore directly solve this linear equation and get the values of the states! However, for n states the complexity is $\mathcal{O}(n^3)$ and hence this is only possible for small MRPs. For large MRPs, a variety of efficient iterative methods exist. In the following chapters, we will cover *dynamic programming* (chapter 4) *Monte-Carlo evaluation* (chapter 5) and *temporal difference learning* (chapter 6).

3.2.3 Example

3.3 Markov Decision Processes

So far, we only considered processes *without* actions, i.e., we were not able to interact with the process. The next natural extension is from MRPs to MDPs:

Definition 17 (Markov Decision Process)**.** A *Markov decision process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, R, \gamma, \iota \rangle$ with the (finite) set of discrete-time states $S_t \in \mathcal{S}$, $n \coloneqq |\mathcal{S}|$, (finite) set of actions $A_t \in \mathcal{A}$, $m \coloneqq |\mathcal{A}|$, transition matrix $\boldsymbol{P}_{ss'}^{a} = P(s' \,|\, s, a)$, reward function $R : \mathcal{S} \times \mathcal{A} : (s, a) \mapsto R(s, a)$, discount factor $\gamma \in [0, 1]$, and the initial state distribution $\iota_i = P(S_0 = i)$. We call $r_t = R(s_t)$ the immediate reward at time step t.

An interesting—yet philosophical—question is, whether a scalar reward is adequate to formulate a goal? The big hypothesis underlying its usage is the *Sutton hypothesis* that wall we mean by goals can be formulated as the maximization of a sum of immediate rewards. While this hypothesis might be wrong, it turns out to be so simple and flexible that we just use it. Also, it forces us to simplify our goal and to actually formulate *what* we want instead of *why*. Hence, the goal must be outside of the agent's direct control, i.e., it must not be a component of the agent. However, the agent must be able to measure successes explicitly and frequently.

In order to reason about an agent and what it might do, we first have to introduce *policies*.

3.3.1 Policies

A *policy* defines, at any point in time, what action an agent takes, i.e., it fully defines the *behavior* if the agent. Policies are very flexible and can be Markovian or history-dependent, deterministic or stochastic, stationary or non-stationary, etc.

Definition 18 (Policy)**.** A *policy* π is a distribution over actions given the state s, i.e., $\pi(a \,|\, s) = P(a \,|\, s)$.

Note that we can reduce a deterministic policy $a = \pi(s)$ to a stochastic one using $\pi(a \,|\, s) = \mathbb{1}\big[a = \pi(s)\big]$ for discrete and $\pi(a \,|\, s) = \delta\big(a - \pi(s)\big)$ for continuous action spaces. Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, R, \gamma, \iota \rangle$ and a policy π, let $\mathcal{M}^{\pi} = \langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}^{\pi}, R^{\pi}, \gamma, \iota \rangle$ be the policy π's MRP with

$$\boldsymbol{P}_{ss'}^{\pi} = \mathbb{E}_{a \sim \pi(\cdot \,|\, s)}\big[\boldsymbol{P}_{ss'}^{a}\big] \qquad\qquad R^{\pi}(s) = \mathbb{E}_{a \sim \pi(\cdot \,|\, s)}\big[R(s, a)\big]. \tag{3.2}$$

This allows us to apply theory of MRPs and Markov chains to MDPs. However, it is often useful to exploit the action distribution instead of reducing it to the state dynamics.

Value Functions

Like for MRPs, we define the value function for MDPs:

Definition 19 (Value Function for MDP)**.** The *state value function* of a MDP is $V^{\pi}(s) \coloneqq \mathbb{E}_{\boldsymbol{P},\pi}[J_t \,|\, s_t = s]$ for any t. That is, the *expected* return starting from state s where the expectation is w.r.t. the state dynamics and policy.

However, as we seek to maximize the return and therefore want to steer towards the largest point of the value function, it is helpful to also define the *action* value function:

Definition 20 (Action Value Function)**.** The *action value function* of a MDP is $Q^{\pi}(s, a) \coloneqq \mathbb{E}_{\boldsymbol{P},\pi}[J_t \,|\, s_t = s, a_t = a]$ for any t. That is, the *expected* return starting from state s, taking action a in the first step and subsequently following policy π where the expectation is w.r.t. the state dynamics and policy.

Hence, if we know the action value function for some policy π, we can easily choose the action that steers the system to the largest return achievable following π by locally maximizing $Q(s, a)$ over a for a given state s: $\pi(s) = \arg\max_a Q(s, a)$.

Similar to MRPs, we can also decompose the state and action value function according to a Bellman equation.

Theorem 5 (Bellman Expectation Equation). *For all states $s \in \mathcal{S}$, we have the following decompositions:*

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\pi,\boldsymbol{P}}\big[R(s,a) + \gamma V^\pi(s_{t+1}) \,\big|\, s_t = s\big] = \mathbb{E}_{a\sim\pi}\big[Q^\pi(s,a)\big] \\ Q^\pi(s,a) &= R(s,a) + \gamma \mathbb{E}_{\pi,\boldsymbol{P}}\big[Q^\pi(s_{t+1}, a_{t+1}) \,\big|\, s_t = s, a_t = a\big] \end{aligned}$$

Note that the Q-function decomposition is very similar to the MRP-decomposition of the state value function.

Proof. Left as an exercise for the reader (hint: plug in the definitions of the individual components). □

Due to the reformulation (3.2), we can reformulate the Bellman equation analogous to (3.1) as $\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi$ which we can solve in closed form. However, also analogous to the MRP-case, this is inefficient for high-dimensional state spaces.

Definition 21 (Bellman Operator). The *Bellman operator* for V and Q is an operator T^π mapping from state and action value functions to state and action value functions. It is defined as follows:

$$\begin{aligned} (T^\pi V)(s) &= \mathbb{E}_{\pi,\boldsymbol{P}}\big[R(s,a) + \gamma V(s_{t+1}) \,\big|\, s_t = s\big] \\ (T^\pi Q)(s,a) &= R(s,a) + \gamma \mathbb{E}_{\pi,\boldsymbol{P}}\big[Q(s_{t+1}, a_{t+1}) \,\big|\, s_t = s, a_t = a\big] \end{aligned}$$

Theorem 6. *The Bellman operator is an α-contraction mapping w.r.t. $\lVert\cdot\rVert_\infty$ if $\gamma \in (0, 1)$.*

Proof. □

Remark 4. *With these operators, we can compactly write the Bellman equation(s) as $T^\pi V^\pi = V^\pi$ and $T^\pi Q^\pi = Q^\pi$ and the policy's state and action value functions are the* unique *respective fixed points of T^π.*

Optimality

Definition 22 (Optimality). The optimal state/action-value function is the maximum value over all policies:

$$V^*(s) \coloneqq \max_\pi V^\pi(s) \qquad\qquad Q^*(s,a) \coloneqq \max_\pi Q^\pi(s,a).$$

The optimal value function then specifies the *best* possible performance in the MDP and we call an MDP *solved* when we know the optimal value function.

Definition 23 (Policy Ordering). For two policies π, π', we write $\pi \geq \pi'$ iff $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

Theorem 7. *For any Markov decision process there exists a optimal policy π^* with $\forall \pi : \pi^* \geq \pi$ and all policies achieve the unique optimal state- and action-value functions (22). There exists a deterministic optimal policy.*

Remark 5. *We can recover the optimal deterministic policy by maximizing $Q^*(s, a)$ over a:*

$$\pi^*(s \,|\, a) = \begin{cases} 1 & \textit{if } a = \arg\max_{a\in\mathcal{A}} Q^*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

With these definitions at hand, we can take a look at Bellman's principle of optimality:

“An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.”
(Richard Bellman, 1957)

This principle is formalized in the *Bellman optimality equation.*

Theorem 8 (Bellman Optimality Equation)**.** *For all states $s \in \mathcal{S}$, we have the following decompositions:*

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a) = \max_{a \in \mathcal{A}} \Big\{ R(s,a) + \gamma \mathbb{E}_{\boldsymbol{P}}\big[V^*(s_{t+1}) \,\big|\, s_t = s\big] \Big\}$$
$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{\boldsymbol{P}}\big[V^*(s_{t+1}) \,\big|\, s_t = s\big] = R(s,a) + \gamma \mathbb{E}_{\boldsymbol{P}}\big[\max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a') \,\big|\, s_t = s\big]$$

Proof. Left as an exercise for the reader (hint: plug in the definitions of the individual components). □

Definition 24 (Bellman Optimality Operator)**.** The *Bellman optimality operator* for V and Q is an operator T^* mapping from state- and action-value functions to state- and action-value functions. It is defined as follows:

$$(T^*V)(s) = \max_{a \in \mathcal{A}} \Big\{ R(s,a) + \gamma \mathbb{E}_{\boldsymbol{P}}\big[V(s_{t+1}) \,\big|\, s_t = s\big] \Big\}$$
$$(T^*Q)(s,a) = R(s,a) + \gamma \mathbb{E}_{\boldsymbol{P}}\big[\max_{a' \in \mathcal{A}} Q(s_{t+1}, a') \,\big|\, s_t = s\big]$$

Theorem 9. *The Bellman optimality operator is an α-contraction mapping w.r.t. $\|\cdot\|_\infty$ if $\gamma \in (0,1)$.*

Proof. □

Remark 6. *With these operators, we can compactly write the Bellman equation(s) as $T^*V^* = V^*$ and $T^*Q^* = Q^*$ and the optimal state- and action-value functions are the* unique *fixed points of T^*. Also, repeated application of T^* to any state- or action-value function converges to the optimal state- or action-value function.*

While we had closed-form solutions for the MRP and policy value functions, it is not possible to solve the Bellman optimality equation in closed form as its a nonlinear equation system (due to the involved maximizations). In the following chapters we will look at a variety of methods for solving this problem iteratively, starting with *dynamic programming.*

3.3.2 Example

3.4 Wrap-Up

- definition of Markov reward processes and Markov decision processes
- definition of the two value functions and how to compute them
- definition of an optimal policy
- the Bellman equation
- the Bellman expectation and optimality equations

4 Dynamic Programming

4.1 Finite Horizon DP

4.2 Policy Iteration

4.2.1 Policy Evaluation

4.2.2 Policy Improvement

4.2.3 Using the Action-Value Function

4.2.4 Examples

4.3 Value Iteration

4.3.1 Principle of Optimality

4.3.2 Convergence

4.3.3 Example

4.4 Policy vs. Value Iteration

4.5 Efficiency

5 Monte-Carlo Algorithms

5.1 Policy Evaluation

5.2 Example

6 Temporal Difference Learning

6.1 Temporal Differences vs. Monte-Carlo

6.1.1 Bias-Variance Trade-Off

6.1.2 Markov Property

6.1.3 Backup

6.2 Bootstrapping and Sampling

6.3 TD(λ)

6.3.1 n-Step Return

6.3.2 λ-Return

6.3.3 Eligibility Traces

6.4 Example

6.5 Wrap-Up

7 Tabular Reinforcement Learning

7.0.1 Monte-Carlo Methods

Generalized Policy Iteration

Greediness and Exploration vs. Exploitation

ϵ-Greedy Exploration and Policy Improvement

Monte-Carlo Policy Iteration and Control

GLIE Monte-Carlo Control

7.0.2 TD-Learning: SARSA

Convergence

n-Step

Eligibility Traces and SARSA(λ)

Example

7.1 Off-Policy Methods

7.1.1 Monte-Carlo

7.1.2 TD-Learning

Importance Sampling

Q-Learning

Convergence

Example

7.2 Remarks

7.3 Wrap-Up

8 Function Approximation

8.1 On-Policy Methods

8.1.1 Stochastic Gradient Descent

8.1.2 Gradient Monte-Carlo

. . . with Linear Function Approximation

8.1.3 Semi-Gradient Methods

. . . with Linear Function Approximation

8.1.4 Least-Squares TD

Semi-Gradient SARSA

8.2 Off-Policy Methods

8.2.1 Semi-Gradient TD

8.2.2 Divergence

8.3 The Deadly Triad

8.4 Offline Methods

8.4.1 Batch Reinforcement Learning

8.4.2 Least-Squares Policy Iteration

8.4.3 Fitted Q-Iteration

8.5 Wrap-Up

9 Policy Search

9.1 Policy Gradient

9.1.1 Computing the Gradient

Finite Differences

Least-Squares-Based Finite Differences

Likelihood-Ratio Trick

9.1.2 REINFORCE

Gradient Variance and Baselines

Example

9.1.3 GPOMDP

9.2 Natural Policy Gradient

9.3 The Policy Gradient Theorem

9.3.1 Actor-Critic

9.3.2 Compatible Function Approximation

Example

9.3.3 Advantage Function

9.3.4 Episodic Actor-Critic

9.4 Wrap-Up

10 Deep Reinforcement Learning

10.1 Deep Q-Learning: DQN

10.1.1 Replay Buffer

10.1.2 Target Network

10.1.3 Minibatch Updates

10.1.4 Reward- and Target-Clipping

10.1.5 Examples

10.2 DQN Enhancements

10.2.1 Overestimation and Double Deep Q-Learning

10.2.2 Prioritized Replay Buffer

10.2.3 Dueling DQN

10.2.4 Noisy DQN

10.2.5 Distributional DQN

10.2.6 Rainbow

10.3 Other DQN-Bases Methods

10.3.1 Count-Based Exploration

10.3.2 Curiosity-Driven Exploration

10.3.3 Ensemble-Driven Exploration

10.4 Wrap-Up

11 Deep Actor-Critic

11.1 Surrogate Loss

11.1.1 Kakade-Langford-Lemma

11.1.2 Practical Surrogate Loss

11.2 Advantage Actor-Critic (A2C)

11.3 On-Policy Methods

11.3.1 Trust-Region Policy Optimization (TRPO)

Practical Implementation

Conjugate Gradient

11.3.2 Proximal Policy Optimization (PPO)

11.4 Off-Policy Methods

11.4.1 Deep Deterministic Policy Gradient (DDPG)

11.4.2 Twin Delayed DDPG (TD3)

11.4.3 Soft Actor-Critic (SAC)

11.5 Wrap-Up

12 Frontiers

12.1 Partial Observability

12.2 Hierarchical Control

12.2.1 The Options Framework

12.3 Markov Decision Processed Without Reward

12.3.1 Intrinsic Motivation

12.3.2 Inverse Reinforcement Learning

12.4 Model-Based Reinforcement Learning

12.5 Wrap-Up