3 Stochastic optimal control and reinforcement learning

There are different types of machine learning including supervised learning, self-supervised learning and unsupervised learning. *Reinforcement learning refers to a learner or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions.* This is based on evaluative information from the environment and could be called action-based learning. Reinforcement learning implies a cause and effect relationship between actions and reward. It implies goal directed behavior at least insofar as the agent has an understanding of reward versus lack of reward or punishment. *Optimal Control* (OC) and *Reinforcement Learning* (RL) both deal with *sequential decision-making in deterministic and stochastic environments*, but they approach the problem from different perspectives and have distinct characteristics:

Foundation and historical roots:

- OC: Stems from the field of control theory. It traditionally focuses on the mathematical modeling and understanding of systems, often with a well-defined model of the environment in mind.
- RL: Emerged from the realm of artificial intelligence and, to some extent, psychology, inspired by behavioral learning theories. RL often operates in scenarios where the model of the environment is unknown.

Knowledge about the environment:

- OC: Typically assumes a known model of the environment. This model describes state transitions and rewards (or costs) as a function of states and actions.
- RL: Can operate with or without an explicit model of the environment. Model-free RL methods, like Q-learning or policy gradient methods, directly learn a policy or value function without needing to know the dynamics or reward structure.

Solution techniques:

- OC: Methods in OC, such as dynamic programming (Value Iteration, Policy Iteration), rely heavily on the known model to find the optimal policy.
- RL: Employs a wider range of techniques, including but not limited to dynamic programming. Many algorithms, especially model-free ones, rely on interaction with the environment to learn. RL also incorporates deep learning (Deep RL) to handle large-scale problems with high-dimensional input spaces.

Application:

- OC: Traditionally applied to problems with smaller state and action spaces due to the computational intensity of exact methods. However, approximations are possible for larger problems.
- RL: Given its roots in AI, RL has been widely applied to large-scale and high-dimensional problems, especially with the advent of deep learning. Examples include playing Atari games, Go, and various robotics tasks.

Exploration vs. exploitation:

- OC: When formulated classically, the OC problem typically assumes full knowledge of the system, and hence the concepts of exploration and exploitation are not central.
- RL: The trade-off between exploration (trying new actions to discover their effects) and exploitation (choosing known good actions) is a central theme in RL, especially in model-free settings.

Goal:

- OC: The primary goal is to find the optimal policy (or control law) given the system dynamics and reward function.
- RL: While the end goal is also to find an optimal policy, RL places a significant emphasis on learning from interaction. This can involve learning about the environment's dynamics, the reward structure, or directly about the optimal policy or value function.

3.1 Background material on Machine and Deep Learning

Machine Learning (ML) and Deep Learning (DL) are a subset of artificial intelligence (AI) that enables computers to learn from data and make decisions with minimal human intervention. ML and DL algorithms are classified into supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses labeled data to train models, unsupervised learning identifies structures in unlabeled data, and reinforcement learning involves agents optimizing cumulative rewards in an environment. For further reading, we refer to:

- Hastie, T., Tibshirani, R., & Friedman, J, The Elements of Statistical Learning, Springer, 2009.
- Bishop, C. M., & Bishop, H., Deep Learning: Foundation and Concepts, Springer, 2024.
- Prince, S. J. D., Understanding Deep Learning. Cambridge University Press, 2023, `https://udlbook.github.io/udlbook/`.
- Zhang, A., et. al, Dive into Deep Learning, Cambridge University Press, 2023, `https://d2l.ai/`.

3.2 Theoretical foundation

Since we are in a stochastic setting, it is important to understand the *basics of probability theory*. We use sans-serif letters such as $\mathsf{x}, \boldsymbol{\mathsf{x}}$, and $\boldsymbol{\mathsf{X}}$ to represent *random variables* (scalars, vectors and matrices) and serif letters such as $x, \mathbf{x}$, and $\mathbf{X}$ to represent the corresponding *deterministic variables* or events. For an introduction to probability theory, please refer to the brief introduction given in the Appendix A and for a consistent summary to [3.1]. A comprehensive overview can be found in [3.2].

3.2.1 Stochastic dynamical systems

We consider stochastic, nonlinear dynamical systems with discrete time evolution of the form

$$\boldsymbol{\mathsf{x}}_{k+1} = \mathbf{f}_d(\boldsymbol{\mathsf{x}}_k, \boldsymbol{\mathsf{u}}_k, \boldsymbol{\mathsf{w}}_k), \quad k = 0, \dots, N-1 \ , \tag{3.1}$$

and initial condition $\boldsymbol{\mathsf{x}}_0$ drawn from the initial condition distribution $\mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0)$. Here,

- k denotes the discrete time or epoch index,
- $\boldsymbol{\mathsf{x}}_k \in \mathcal{X} \subset \mathbb{R}^n$ is the stochastic state vector and
- $\boldsymbol{\mathsf{u}}_k \in \mathcal{U}(\boldsymbol{\mathsf{x}}_k) \subset \mathcal{U} \subset \mathbb{R}^m$ is the stochastic input vector. The input vector $\boldsymbol{\mathsf{u}}_k$ is restricted to a nonempty subset $\mathcal{U}(\boldsymbol{\mathsf{x}}_k) \subset \mathcal{U}$ that depends on $\boldsymbol{\mathsf{x}}_k$.
- $\boldsymbol{\mathsf{w}}_k \in \mathcal{W}$ is the disturbance vector.
- The horizon length is given by N.
- The mapping $\mathbf{f}_d : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \rightarrow \mathcal{X}$ describes how the system state $\boldsymbol{\mathsf{x}}$ evolves over time.
- The sequence $\boldsymbol{\mathsf{W}}_{[0:N-1]} = (\boldsymbol{\mathsf{w}}_0, \boldsymbol{\mathsf{w}}_1, \dots, \boldsymbol{\mathsf{w}}_{N-1})$ is assumed to be uncorrelated.

Let us use $\mathsf{p}(\mathbf{i}_k)$ to denote the probability distribution, with observation $\mathbf{i}_k$, of the stochastic vector $\boldsymbol{\mathsf{i}}_k \in \mathcal{I}$ at time index k, i.e., $\boldsymbol{\mathsf{i}}_k \sim \mathsf{p}(\mathbf{i}_k)$. A probability distribution is a description of how likely a stochastic variable is to take on each of its possible states. The method of describing this distribution varies based on the nature of the variable, being either discrete or continuous. Using this concept, it is actually possible to write the dynamics of the distribution with the *law of total probability*, cf. (A.12), as

$$\begin{aligned}\mathsf{p}(\mathbf{x}_{k+1}) &= \mathbb{E}_{\boldsymbol{\mathsf{x}}_k, \boldsymbol{\mathsf{u}}_k \sim \mathsf{p}(\mathbf{x}_k, \mathbf{u}_k)}\{\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\} \\ &= \int_{\mathcal{X}} \int_{\mathcal{U}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \mathsf{p}(\mathbf{x}_k, \mathbf{u}_k) \mathrm{d}\mathbf{u}_k \mathrm{d}\mathbf{x}_k \ ,\end{aligned} \tag{3.2}$$

where $\mathsf{p}_{\mathsf{x}} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \rightarrow [0,1]$ encodes the stochastic dynamics as a conditional distribution of the next state $\mathbf{x}_{k+1}$ as a function of the current state $\mathbf{x}_k$ when applying the control input $\mathbf{u}_k$ [3.3, p. 13]. Thus, (3.1) can be alternatively written as

$$\boldsymbol{\mathsf{x}}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, 1, \dots, N-1 \ . \tag{3.3}$$

The fact that at this point two expressions (3.1) and (3.3) are given for the system dynamics is due to the fact that, depending on the problem, one of the two representations is to be preferred in order to obtain mathematically consistent and clear expressions. In the control engineering context, (3.1) is preferred over (3.3), whereas (3.3) is widely used in the reinforcement learning context. We will utilize both notations. Note that by construction, the system (3.1) satisfies the Markov property, see [3.4], which states that $\mathbf{x}_{k+1}$ depends only on quantities of the previous time step k, i.e.

$$\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) = \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, \dots, N-1. \tag{3.4}$$

Note 3.1. In the reinforcement learning literature, the control input $\mathbf{u}_k$ is referred to as action $\mathbf{a}_k$ and the state $\mathbf{x}_k$ is denoted by $\mathbf{s}_k$.

3.2.2 Rollouts, rewards and return

Let us introduce the *state sequence*

$$\mathbf{X}_{[0:N]} = (\mathbf{x}_0, \dots, \mathbf{x}_N) \tag{3.5}$$

and *input sequence*

$$\mathbf{U}_{[0:N-1]} = (\mathbf{u}_0, \dots, \mathbf{u}_{N-1}) \ , \tag{3.6}$$

which describes the time evolution of the state and input variables. We denote

$$\boldsymbol{\tau}_{[0,N]} = (\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1 \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N) \tag{3.7}$$

as the N-step *rollout*[1]. In optimal control, we are typically interested in minimizing a total cost. In the reinforcement learning literature, the reward is used instead of the cost. Consequently, the accumulative costs (total cost) are minimized and accumulative rewards (return) are maximized. Throughout this work we will adopt the logic and notation commonly found in the reinforcement learning literature. Note that the rewards can be positive and negative.

Definition 3.1. (Discounted return, see [3.5]) The *future discounted return* of a rollout $\boldsymbol{\tau}_{[k:N]}$ is

$$\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) = \sum_{i=k}^{N} \gamma^{i-k} \mathsf{r}_i = \gamma^{N-k} \mathsf{r}_N + \sum_{i=k}^{N-1} \gamma^{i-k} \mathsf{r}_i \tag{3.8}$$

with discount factor $\gamma \in [0, 1]$. It discounts and accumulates the *random immediate reward* $\mathsf{r}_k : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ as well as the *random terminal reward* $\mathsf{r}_N : \mathcal{X} \to \mathbb{R}$.

[1] Rollouts are also frequently called trajectories.

The smaller γ is chosen, the lower future rewards in r_k will be weighted, thereby giving more significance to immediate rewards. Conversely, rewards received for high occupancies in the future are weighted more heavily. Thus, this parameter essentially represents the distance of foresight into the future. Note that if R_k is the value received at time index k, then

$$\begin{aligned}\mathsf{R}_k &= \mathsf{r}_k + \gamma\mathsf{r}_{k+1} + \gamma^2\mathsf{r}_{k+2} + \ldots + \gamma^{N-k-1}\mathsf{r}_{N-1} + \gamma^{N-k}\mathsf{r}_N \\ &= \mathsf{r}_k + \gamma\Big(\mathsf{r}_{k+1} + \gamma\mathsf{r}_{k+2} + \gamma^2\mathsf{r}_{k+3} + \ldots + \gamma^{N-k-2}\mathsf{r}_{N-1} + \gamma^{N-k-1}\mathsf{r}_N\Big) \\ &= \mathsf{r}_k + \gamma\mathsf{R}_{k+1}\ . \end{aligned} \tag{3.9}$$

So, the further away a reward is from the initial state $\mathbf{x}_k$, the less actual reward we will receive from it. For the sake of completeness, we introduce the *reward sequence*

$$\mathbf{r}_{[0:N]} = (\mathsf{r}_0, \ldots, \mathsf{r}_N)\ . \tag{3.10}$$

There is a finite and infinite horizon setting in RL. They mainly differ in the time frame over which the agent plans and makes decisions:

- In the *finite horizon setting*, the agent operates over a fixed, known number of time steps, i.e, N is finite.
- In the *infinite horizon setting*, the agent considers an infinite number of future steps, i.e., $N = \infty$.

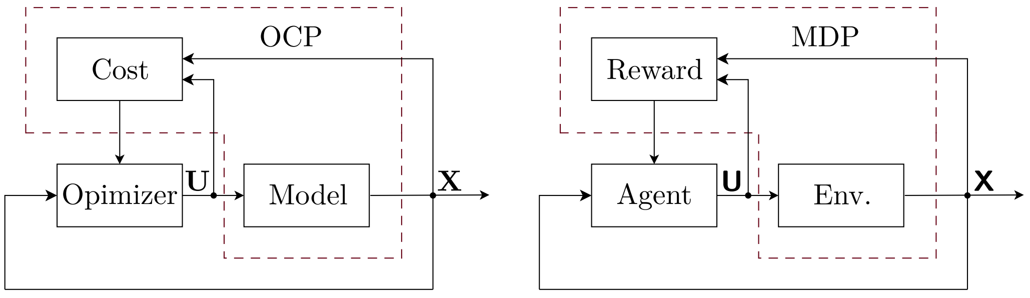
Note 3.2. There is no consensus within the community about whether the reward corresponding to the state $\mathbf{x}_k$ is gained at time index k as in [3.5], or time index $k+1$, as in [3.6]. Here, it is assumed that the reward is gained at time index k and is denoted by r_k.

There are two settings for learning and optimization:

- The *episodic/sequential setting*, where the experience is broken up into a series of episodes/sequences. It is our goal to maximize the cumulative reward within a single episode. This setting is typical in scenarios like games or simulations where tasks are naturally episodic and reset after reaching a conclusion.
- The *continuing setting*, where the task doesn't have clear episode boundaries. It is our goal to maximize the cumulative reward over an infinite or very long time horizon, i.e., $N = \infty$. This is more reflective of real-world scenarios like stock market trading or ecosystem management, where the process is ongoing and doesn't reset periodically. In this setting, the notion of discounting future rewards often becomes essential to ensure that the cumulative reward doesn't diverge to infinity.

3.2.3 Different terminologies and interaction models

In control engineering, when addressing a (deterministic) *Optimal Control Problem* (OCP), we refer to a simulation model, a cost function, and a numerical solution method to solve the dynamical optimization problem. Hence, we adopt the Optimizer-Model-Cost interaction framework, as illustrated in Figure 3.1a. In reinforcement learning, we consider the agent-environment-reward interaction model[2], rooted in the principles of the (stochastic) *Markov Decision Process* (MDP). This interaction model is depicted in Figure 3.1b. In RL,



(a) Optimizer-model-cost interaction model. (b) Agent-environment-reward interaction model.

Figure 3.1: Interaction models.

the accumulative rewards (return) are maximized and, in OC, the accumulative costs (total cost) are minimized. In control engineering, the environment corresponds to the controlled system, plant or model and the agent to the decision maker or controller. In essence, while there exists a linguistic divergence between control engineering and computational intelligence terminologies, their underlying semantics largely converge. This dichotomy is primarily a byproduct of the distinct historical and academic roots of the two fields. For more information about the different terminologies in Control Systems Engineering and Computational Intelligence, see [3.7, p. 43].

3.2.4 Markov Decision Process

Markov Decision Processes (MDP) formally describe an environment for reinforcement learning. A Markov Process is a memoryless random process, i.e., a sequence of random states with the Markov property (3.4).

Definition 3.2. (Markov Decision Process) A (stationary[a]) Markov Decision Process (MDP) is a fully observable, probabilistic state model. A finite-horizon, discount-reward MDP $\mathcal{M}$ is a tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{R}, \mathsf{p}, \mathsf{p}_{\mathsf{x}_0}, \gamma, N \rangle$ containing:

- $\mathcal{X}$ is the state space
- $\mathcal{U}$ is the input space
- $\mathcal{R}$ is the reward space

[2]Frequently referred to simply as the agent-environment interaction model.

- p is the dynamic function
- $\mathsf{p}_{\mathsf{x}_0}$ is the initial condition function
- $\gamma \in [0, 1]$ is a discount factor and
- N is the horizon length.

[a]The dynamic function is time-independent.

Note 3.3. Markov models are categorized as either fully or partially observable as well as either autonomous or non-autonomous, cf. Table 3.1. In the partially observable setting, the agent only has access to the output[a] $\mathbf{y}_k$ at each time step k. This model is called *Partially Observable Markov Decision Process* (POMDP) and in addition to the MDP, there is an output-transition probability $\mathsf{o}(\mathbf{y}_k \mid \mathbf{x}_k, \mathbf{u}_k)$. In the autonomous case, there is no input $\mathbf{u}_k$.

[a]Called observation in the reinforcement learning literature.

Table 3.1: Types of Markov Models.

	Fully observable	**Partially observable**
Auto.	Markov Chain	Hidden Markov Model
Non-auto.	Markov Decision Process	Partially Observable Markov Decision Process

The MDP and agent together thereby give rise to an episode

$$\mathbf{x}_0, \mathbf{u}_0, \mathsf{r}_0,\ \mathbf{x}_1, \mathbf{u}_1, \mathsf{r}_1,\ \ldots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathsf{r}_{N-1},\ \mathbf{x}_N, \mathsf{r}_N\ , \tag{3.11}$$

where $\mathbf{x}_k \in \mathcal{X}, \mathbf{u}_k \in \mathcal{U}$ and $\mathsf{r}_k \in \mathcal{R}$.

$$\begin{tikzcd} \mathbf{x}_0 \arrow[r, "\mathbf{u}_0", "\mathsf{r}_0"'] & \mathbf{x}_1 \arrow[r, "\mathbf{u}_1", "\mathsf{r}_1"'] & \mathbf{x}_2 \arrow[r, dash] & \cdots \arrow[r] & \mathbf{x}_N \end{tikzcd}$$

Figure 3.2: Episode.

The states, inputs, and rewards are either *discrete* or *continuous* random variables.

- Discrete variables: These are variables that take values on a finite set. For example, inputs in a Tic-Tac-Toe game are discrete because there are a limited number of squares where a player can place a symbol.

  Example 3.1 (Tic-Tac-Toe game). In the Tic-Tac-Toe game, the board configuration at any given time serves as the outcomes. Each outcome can be represented as a 3×3 matrix or a vector with 9 elements with entries from $\{\mathrm{X}, \mathrm{O}, \mathrm{empty}\}$. That's a total of $3^9 = 19683$ different ways the 3×3 grid can be filled in. The sample space is the set of all possible outcomes and can be written as $\boldsymbol{\xi} \in \Xi = \{\mathrm{X}, \mathrm{O}, \mathrm{empty}\}^9$. For each element x of the random vector $\boldsymbol{\mathsf{x}}$, a possible mapping from the sample space the real numbers is $\mathsf{x}(\mathrm{X}) = 1$, $\mathsf{x}(\mathrm{O}) = 2$, $\mathsf{x}(\mathrm{empty}) = 3$. Hence, $\boldsymbol{\mathsf{x}}(\boldsymbol{\xi}) \in \mathbb{R}^9$ and $\boldsymbol{\mathsf{x}} \in \mathcal{X} = \{1, 2, 3\}^9 \subseteq \mathbb{R}^9$. The least number of moves required to win a Tic-Tac-Toe game is 5, while the maximum is 9. Assuming player one, using X, goes first and player two, using O, follows, the game proceeds with alternating moves until the board is filled with five X's and four O's. The inputs are discrete and correspond to the placement of $\{\mathrm{X}, \mathrm{O}\}$ in an empty cell. At the start of the game, there are 9 possible inputs, one for each cell on the board. Thus, the first player has $9+7+5+3+1 = 25$ and the second has $8+6+4+2 = 20$ possible moves, not accounting for games ending before the board is full. In a Tic-Tac-Toe game, the reward structure is often based on the game's outcome, which is inherently discrete. The goal of the game is to get three in a row - horizontally, vertically, or diagonally. Play continues until a player achieves this goal or all the spaces are filled with X's and O's. Win-Lose-Draw: A simple way to define the reward is to give a positive value for a win (1), a negative value for a loss (-1), and a smaller value or zero for a draw (0), i.e. $\mathcal{R} = \{1, -1, 0\}$, with $|\mathcal{R}| = 3$. In more sophisticated models, future rewards can be discounted to prioritize short-term gains and a negative reward could be given for each move to encourage the agent to win in fewer steps. This discussion reveals that even elementary games such as Tic-Tac-Toe are founded on complex mathematical concepts.

X	O	X
X	X	O
O	O	X

(a) Win $r = 1$.

X	O	X
X	O	
O	O	X

(b) Loss $r = -1$.

X	O	X
X	O	O
O	X	X

(c) Draw $r = 0$.

Figure 3.3: Reward stucture of the Tic-Toc-Toe game.

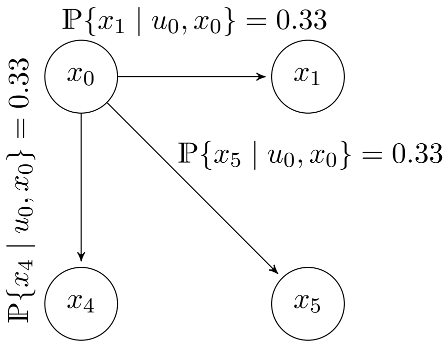
Example 3.2 (Frozen Lake). In the Frozen Lake game, the environment is a grid composed of safe tiles (frozen surface) and dangerous tiles (holes). For examlpe, it can be represented as a 4×4 matrix or a vector with 16 elements. In the environment's code, each tile is represented by a letter as follows: (S: starting point, safe), (F: frozen surface, safe), (H: hole, stuck forever) and (G: goal, safe). Each tile can take one of four values $\{\mathrm{S}, \mathrm{F}, \mathrm{H}, \mathrm{G}\}$. By default, the environment is always in the same configuration. The agent A must navigate from the starting point (S) to the goal (G) without falling into holes. The agent's location in the grid is the state and hence there are $16 = |\mathcal{X}| = \{0, 1, \ldots, 15\}$ possible states. Possible inputs are $\{\leftarrow, \downarrow, \rightarrow, \uparrow\}$. We can map the inputs to the real line via $\{\mathsf{u}(\leftarrow) = 0, \mathsf{u}(\downarrow) = 1, \mathsf{u}(\rightarrow) = 2, \mathsf{u}(\uparrow) = 3\}$. A sequence of inputs leads the agent to the goal. The agent must learn the optimal path, avoiding holes, and reaching the goal with a minimum number of moves. The game is deterministic in the non-slippery version. In the slippery version, the action the agent takes only has 33% chance of succeeding. In case of failure, one of the three other inputs is randomly taken instead, see Figure 3.4.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

(a) Configuration of the environment.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(b) Current location of the agent as state encoding.


$\mathbb{P}\{x_1 \mid u_0, x_0\} = 0.33$
x_0
x_1
$\mathbb{P}\{x_4 \mid u_0, x_0\} = 0.33$
$\mathbb{P}\{x_5 \mid u_0, x_0\} = 0.33$
x_4
x_5

(c) Slippery version.

Figure 3.4: 4×4 Frozen Lake game - configuration, encoding, and stochastic version.

- Continuous variables: These are variables that can take an infinite number of values within a given range. For example, the angle of a rotatory pendulum is a continuous state because it can vary within a range.

Example 3.3 (Rotatory pendulum). The state of a pendulum is typically characterized by its angle θ and angular velocity ω. These variables are continuous and can vary within a specific range, i.e. $\mathbf{x}^\top = [\theta, \omega] \in \mathbb{R}^2$ and $\mathcal{X} = [0\,\mathrm{rad}, 2\pi\,\mathrm{rad}] \times [-2\,\mathrm{rad/s}, 2\,\mathrm{rad/s}]$ for example. The input for a pendulum could be the torque τ applied to it, which is also a continuous variable, i.e. $\mathsf{u} = \tau \in \mathbb{R}^1$ and $\mathcal{U} = [-1\,\mathrm{Nm}, 1\,\mathrm{Nm}]$. A negative reward can be given based on the angular deviation from the upright position, usually $\theta = 0\,\mathrm{rad}$. The closer the pendulum is to being upright, the smaller the absolute value of the negative reward. To encourage efficient control, the reward can also include a term that penalizes large torque inputs, hence a possible deterministic reward is $r = -(|\theta| + d\tau^2)$, with coefficient $d > 0$.

Value-discrete case

Suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *finite sets* of cardinality $|\mathcal{X}|$, $|\mathcal{U}|$, and $|\mathcal{R}|$. Let us assume that the random variables $\mathbf{x}'$ and r have well defined discrete probability distributions dependent only on the preceding state $\mathbf{x}$ and input $\mathbf{u}$. That is, for particular values of these random variables, there is a *dynamic function* $\mathsf{p} : \mathcal{X} \times \mathcal{R} \times \mathcal{X} \times \mathcal{U} \rightarrow [0, 1]$, which is equivalent to the *conditional probability* $\mathbb{P}\{\cdot \mid \cdot\}$. It allows us to predict the future state $\mathbf{x}'$ and current reward r:

$$\mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) = \mathbb{P}\{\mathsf{x}_{k+1} = \mathbf{x}', \mathsf{r}_k = r \mid \mathsf{x}_k = \mathbf{x}, \mathsf{u}_k = \mathbf{u}\} \ . \tag{3.12}$$

The dynamic function gives rise to the *state-transition function* $\mathsf{p}_\mathsf{x} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is given by

$$\mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) = \sum_{r \in \mathcal{R}} \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \tag{3.13}$$

and the *reward function* $\mathsf{p}_\mathsf{r} : \mathcal{R} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is

$$\mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.14}$$

The function p_x defines the dynamics of the MDP

$$\mathsf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \tag{3.15}$$

and the function p_r gives the rewards of the MDP

$$\mathsf{r} \sim \mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.16}$$

The *expected reward* is a two-argument function $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$:

$$\bar{r}(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\mathsf{r} \sim \mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u})}\{\mathsf{r}\} = \sum_{r \in \mathcal{R}} \sum_{\mathbf{x}' \in \mathcal{X}} r \, \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.17}$$

Note 3.4. In reinforcement literature, the random immediate reward is often defined as a three-argument function $\mathsf{R} : \mathcal{X} \times \mathcal{U} \times \mathcal{X}' \to \mathbb{R}$ which also incorporates the next state $\mathbf{x}' \in \mathcal{X}'$, see [3.6]. Note that we can always us the law of total probability (A.12) and state-transition function p_x to convert the formulations because

$$\bar{r}(\mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}' \in \mathcal{X}} R(\mathbf{x}, \mathbf{u}, \mathbf{x}') \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.18}$$

For MDPs with *discrete state and input spaces*, we avoid introducing a second index. We denote a specific value-discrete state by $\mathbf{x}_i \in \mathcal{X}$ and a specific value-discrete input by $\mathbf{u}_l \in \mathcal{U}$. In contrast, x_k and u_k represent the state and input at a specific time index k. We define the i-to-j-for-l *state-transition probabilities* as

$$p_{ij}^{\mathbf{u}_l} = \mathsf{p}_\mathsf{x}(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{u}_l) = \mathbb{P}[\mathsf{x}_{k+1} = \mathbf{x}_j, \mid \mathsf{x}_k = \mathbf{x}_i, \mathsf{u}_k = \mathbf{u}_l] \ , \tag{3.19}$$

which can be summarized in the *state-transition probability tensor* $\mathbf{P}$, with tensor elements

$$\mathbf{P}^{\mathbf{u}_l}[i, j] = p_{ij}^{\mathbf{u}_l} \ . \tag{3.20}$$

The rows of the state-transition probability sum up to one for each input $\mathbf{u}_l$, i.e.

$$\sum_{j=0}^{|\mathcal{X}|-1} p_{ij}^{\mathbf{u}_l} = 1 \ . \tag{3.21}$$

for all $\mathbf{x}_i \in \mathcal{X}$ and $\mathbf{u}_l \in \mathcal{U}$.

Example 3.4. For $\mathcal{X} = \{x_0, x_1, x_2\}$ and $\mathcal{U} = \{u_0\}$, with state-transition probability matrix

$$\mathbf{P}^{u_0} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} p_{00}^{u_0} & 0 & 0 \\ p_{10}^{u_0} & 0 & 0 \\ p_{20}^{u_0} & 0 & 0 \end{bmatrix} , \tag{3.22}$$

we can draw a state-transition diagram shown in Figure 3.5.

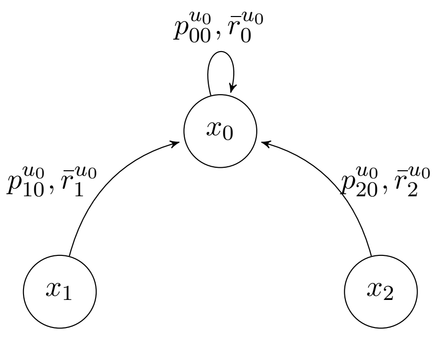

$p_{00}^{u_0}, \bar{r}_0^{u_0}$
x_0
$p_{10}^{u_0}, \bar{r}_1^{u_0}$
$p_{20}^{u_0}, \bar{r}_2^{u_0}$
x_1
x_2

Figure 3.5: A state-transition diagram.

Value-continuous case

Now suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *continuous sets*. Then, the *dynamic function* induces a *state-transition function* $\mathsf{p}_\mathsf{x} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0,1]$ of the form

$$\mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) = \int_{\mathcal{R}} \mathsf{p}(\mathbf{x}_{k+1}, r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}r_k \tag{3.23}$$

and a *reward function* $\mathsf{p}_\mathsf{r} : \mathcal{X} \times \mathcal{U} \to [0,1]$, which is

$$\mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) = \int_{\mathcal{X}} \mathsf{p}(\mathbf{x}_{k+1}, r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}\mathbf{x}_{k+1} \ . \tag{3.24}$$

The function p_x defines the dynamics of the MDP

$$\mathsf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \tag{3.25}$$

and the function p_r gives the rewards of the MDP

$$\mathsf{r}_k \sim \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) \ . \tag{3.26}$$

We can can once again determine the *expected rewards* for state-input pairs as a two-argument function $\bar{r}_k : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$,

$$\bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) = \mathbb{E}_{\mathsf{r}_k \sim \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k)}\{\mathsf{r}_k\} = \int_{\mathcal{R}} r_k \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}r_k \ . \tag{3.27}$$

3.2.5 Control policy

A control law or a control policy[3] is a function that tells us which is the best input to choose in each state. A policy can be deterministic or stochastic and stationary or nonstationary.

Definition 3.3. (Policy, see [3.3, p. 15]). A (stationary) *stochastic policy* $\pi : \mathcal{X} \times \mathcal{U} \to [0, 1]$ is a distribution over inputs given states

$$\mathbf{u}_k \sim \pi(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.28}$$

A *stationary and deterministic policy* $\pi : \mathcal{X} \to \mathcal{U}$ is a function $\boldsymbol{\mu} : \mathcal{X} \to \mathcal{U}$ that maps the state vector $\mathbf{x}_k$ to control inputs

$$\mathbf{u}_k = \boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.29}$$

A *nonstationary and deterministic policy* $\pi : \mathcal{X} \times \{0, \dots, N-1\} \to \mathcal{U}$ is a sequence of functions

$$\left(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_{N-1}\right) \ , \tag{3.30}$$

where $\boldsymbol{\mu}_k$ maps the state vector $\mathbf{x}_k$ to control inputs

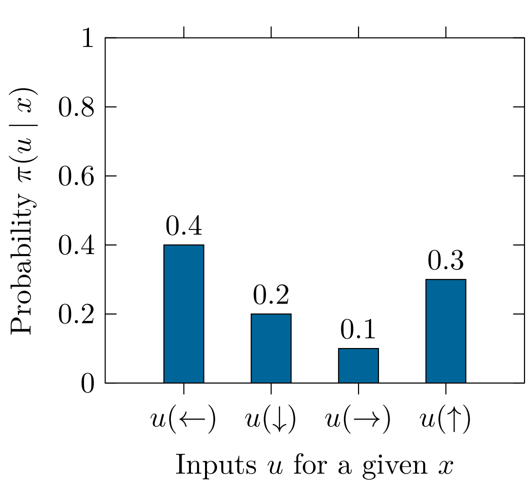
$$\mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) \ . \tag{3.31}$$

The policy is admissible if $\boldsymbol{\mu}_k(\mathbf{x}_k) \in \mathcal{U}(\mathbf{x}_k)$ holds for all $\mathbf{x}_k \in \mathcal{X}_k$ and k. The deterministic policy (3.31) can be written as

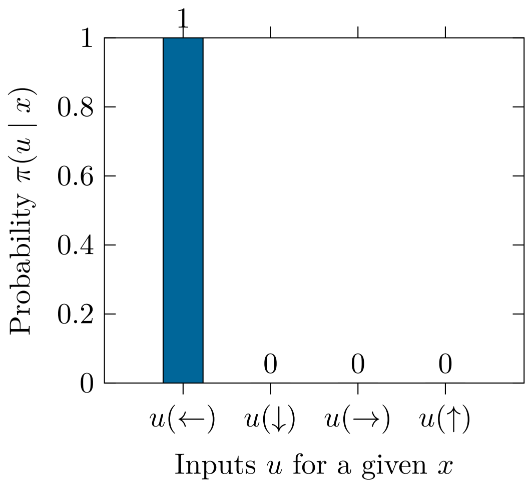
$$\pi(\mathbf{u}_k \mid \mathbf{x}_k) = \begin{cases} 1 & \text{if } \mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) \\ 0 & \text{else} \ . \end{cases} \tag{3.32}$$

Example 3.5. In the discrete case, a stochastic policy denoted as $\pi(\mathbf{u} \mid \mathbf{x})$ is a conditional distribution over the inputs $\mathbf{u} \in \mathcal{U}$ given the state $\mathbf{x} \in \mathcal{X}$, $\pi(\mathbf{u} \mid \mathbf{x}) = \mathbb{P}(\mathbf{u} \mid \mathbf{x})$. As an example, if a robot has four inputs $\mathcal{U} = \{\mathsf{u}(\leftarrow), \mathsf{u}(\downarrow), \mathsf{u}(\rightarrow), \mathsf{u}(\uparrow)\}$. The policy at a state $x \in \mathcal{X}$ for such a set of inputs $\mathcal{U}$ is a distribution where the probabilities of the four inputs could be $[0.4, 0.2, 0.1, 0.3]$. Note that we should have $\sum_{u \in \mathcal{U}} \pi(u \mid x) = 1$ for any state x. A deterministic policy is a special case of a stochastic policy in that the distribution $\pi(u \mid x)$ only gives non-zero probability to one particular input, e.g., $[1, 0, 0, 0]$ for our example with four inputs, see Figure 3.6.

[3]It is also often referred to simply as policy.



(a) Stochastic policy.



(b) Deterministic policy.

Figure 3.6: 4×4 Frozen Lake game - policy.

Theorem 3.1. *(Probability of observing a rollout, see [3.8, p. 11]) The probability distribution of observing a N-step rollout $\boldsymbol{\tau}$ under a policy π is*

$$\mathsf{p}(\boldsymbol{\tau} \mid \pi) = \mathsf{p}^{\pi}(\boldsymbol{\tau}) = \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.33}$$

Proof. We can proof (3.33) by induction using the product rule (A.11) and the Markov property (3.4). With $\mathsf{p}^{\pi}(\mathbf{x}_0) \doteq \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0)$ and $\mathsf{p}^{\pi}(\mathbf{u}_k \mid \mathbf{x}_k) \doteq \pi(\mathbf{u}_k \mid \mathbf{x}_k)$, we get

$$\begin{aligned}
\mathsf{p}^{\pi}(\mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_0) &= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \quad \underbrace{\mathsf{p}(\mathbf{x}_1, \mathbf{u}_0 \mid \mathbf{x}_0)}_{=\mathsf{p}_{\mathsf{x}}(\mathbf{x}_1 \mid \mathbf{x}_0, \mathbf{u}_0)\pi(\mathbf{u}_0 \mid \mathbf{x}_0)} \\
\mathsf{p}^{\pi}(\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_1, \mathbf{u}_0) &= \mathsf{p}^{\pi}(\mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_0) \underbrace{\mathsf{p}(\mathbf{x}_2, \mathbf{u}_1 \mid \mathbf{x}_1, \cancel{\mathbf{x}_0, \mathbf{u}_0})}_{=\mathsf{p}_{\mathsf{x}}(\mathbf{x}_2 \mid \mathbf{x}_1, \mathbf{u}_1)\pi(\mathbf{u}_1 \mid \mathbf{x}_1)} \\
&= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \qquad (3.34) \\
&\vdots \\
\mathsf{p}^{\pi}(\mathbf{x}_N, \dots, \mathbf{x}_0, \mathbf{u}_{N-1}, \dots, \mathbf{u}_0) &= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \ .
\end{aligned}$$

□

Whenever any policy π, whether deterministic or probabilistic, is implemented, the resulting process is a Markov process. In value-discrete case, the associated *state-transition probability tensor* is denoted by $\mathbf{P}^{\pi}$. In the deterministic case, $\pi \in \Pi_d$, then at time index

k, the corresponding input $\mathbf{u}_k$ equals $\boldsymbol{\mu}_k(\mathbf{x}_k)$ and we have

$$\mathbf{P}^\pi[i,j] = p^\pi_{ij} = \mathbb{P}[\mathsf{x}_{k+1} = \mathbf{x}_j \mid \mathsf{x}_k = \mathbf{x}_i, \mathsf{u}_k = \boldsymbol{\mu}_l] \ . \tag{3.35}$$

In the stochastic case, $\boldsymbol{\pi} \in \Pi_p$ and

$$p^\pi_{ij} = \mathbb{E}_{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}\left\{p^{\mathbf{u}}_{ij}\right\} \ . \tag{3.36}$$

3.2.6 Value functions

In reinforcement learning, value functions play a central role in representing and estimating the expected future rewards or returns an agent can obtain from states and inputs. They serve as a foundational concept for many algorithms and provide insight into the quality of different decisions. In the finite horizon setting, we have:

Definition 3.4. (Action-Value Function Q^π for a MDP $\mathcal{M}$ and policy π, see [3.6]). The *action-value function*[a] $Q^\pi : \mathcal{X}\times\mathcal{U} \to \mathbb{R}$ for a MDP $\mathcal{M}$ and policy π gives the expected return if you start in state $\mathbf{x}$, take an arbitrary control input $\mathbf{u}$ (which may not have to come from the control law), and then forever after act according to policy π, i.e.

$$\begin{aligned} Q^\pi(\mathbf{x},\mathbf{u}) &= \mathbb{E}_\pi\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathsf{x}_k = \mathbf{x}, \mathsf{u}_k = \mathbf{u}\right\} && (3.37a)\\ &= \mathbb{E}_{\substack{\mathsf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathsf{x}_{k+1}|\mathsf{x}_k,\mathsf{u}_k)\\ \mathsf{u}_k\sim\pi(\mathsf{u}_k|\mathsf{x}_k)}}\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathsf{x}_k = \mathbf{x}, \mathsf{u}_k = \mathbf{u}\right\} \ . && (3.37b)\end{aligned}$$

[a]Also Q-value function.

The action-value function depends on $\mathbf{x}, \mathbf{u}, \pi$ and $\mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u})$ but is independent of $\mathsf{X}_{[k+1:N]}$ and $\mathsf{U}_{[k+1:N-1]}$ since they are eliminated by taking the expectation. The action-value function $Q^\pi(\mathbf{x},\mathbf{u})$ evaluates how good it is to pick an input $\mathbf{u}$ being in state $\mathbf{x}$.

Definition 3.5. (State-Value Function V^π, see [3.6]). The *state-value function* $V^\pi : \mathcal{X} \to \mathbb{R}$ gives the expected return if you start in state $\mathbf{x}$ and always act according to policy π, i.e.

$$\begin{aligned} V^\pi(\mathbf{x}) &= \mathbb{E}_\pi\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathsf{x}_k = \mathbf{x}\right\} && (3.38a)\\ &= \mathbb{E}_{\mathsf{u}_k\sim\pi(\mathsf{u}_k|\mathsf{x}_k)}\{Q^\pi(\mathsf{x}_k,\mathsf{u}_k) \mid \mathsf{x}_k = \mathbf{x}\} \ . && (3.38b)\end{aligned}$$

Intuitively, for a fixed policy π, the state-value function $V^\pi_k(\mathbf{x})$ evaluates how good the situation is in state $\mathbf{x}$. Clearly, in the deterministic case, the action-value function and state-value function are related via

$$\begin{aligned} V^\pi(\mathbf{x}_k) &= \mathbb{E}_{\mathsf{u}_k\sim\pi(\mathsf{u}_k|\mathsf{x}_k)}\{Q^\pi(\mathsf{x}_k,\mathsf{u}_k) \mid \mathsf{x}_k = \mathbf{x}\}\\ &\overset{(A.7)}{=} \int_\mathcal{X} Q^\pi(\mathbf{x}_k,\mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k)\mathrm{d}\mathbf{x}_k\\ &\overset{(3.32)}{=} Q^\pi(\mathbf{x}_k,\boldsymbol{\mu}_k(\mathbf{x}_k)) \ . \end{aligned} \tag{3.39}$$

Moreover, by definition, the optimal value function produces the maximum return

$$V^*(\mathbf{x}) = \max_{\pi \in \Pi} V^\pi(\mathbf{x}) \quad \text{and} \quad Q^*(\mathbf{x}, \mathbf{u}) = \max_{\pi \in \Pi} Q^\pi(\mathbf{x}, \mathbf{u}) \ . \tag{3.40}$$

Definition 3.6. (Advantage Function A^π for a MDP $\mathcal{M}$, see [3.9]). The *advantage function* $A^\pi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ of a MDP $\mathcal{M}$ for a given policy π, indicates how much better the control input $\mathbf{u}$ is in state $\mathbf{x}$ compared to the average, i.e.

$$A^\pi(\mathbf{x}, \mathbf{u}) = Q^\pi(\mathbf{x}, \mathbf{u}) - V^\pi(\mathbf{x}) \ . \tag{3.41}$$

Advantage functions measure the relative benefit of choosing a specific input over others in a given state.

Note 3.5. The action-value, state-value, and advantage functions are functions of the time index k in the finite horizon setting.

Example 3.6 (Frozen Lake game continued). In the infinite horizon and value discrete case, the value function can be organized in a table. For instance, for a 4×4 Frozen Lake grid, there are 16 states (each cell is a state) and 4 inputs $\{\leftarrow, \downarrow, \rightarrow, \uparrow\}$. The Q-table is a 16×4 matrix with rows representing states and columns representing inputs. So want to calculate $16 \times 4 = 64$ action-state values. Table 3.2 shows the Q-Table for the example at hand.

State	$u = \mathsf{u}(\leftarrow) = 0$	$u = \mathsf{u}(\downarrow) = 1$	$u = \mathsf{u}(\rightarrow) = 2$	$u = \mathsf{u}(\uparrow) = 3$
$x = 0$	$Q^\pi(0,0)$	$Q^\pi(0,1)$	$Q^\pi(0,2)$	$Q^\pi(0,3)$
$x = 1$	$Q^\pi(1,0)$	$Q^\pi(1,1)$	$Q^\pi(1,2)$	$Q^\pi(1,3)$
...	...	...	...	...
$x = 14$	$Q^\pi(14,0)$	$Q^\pi(14,1)$	$Q^\pi(14,2)$	$Q^\pi(14,3)$
$x = 15$	$Q^\pi(15,0)$	$Q^\pi(15,1)$	$Q^\pi(15,2)$	$Q^\pi(15,3)$

Table 3.2: Q-Table for the 4×4 Frozen Lake game.

3.2.7 Bellman Expectation Equation

The *Bellman Expectation Equation* serves as the foundational framework for reinforcement learning. It provides a recursive decomposition of the value function, which is essential for evaluating value functions and finding optimal policies.

Value-discrete case

Suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *finite sets* of cardinality $|\mathcal{X}|$, $|\mathcal{U}|$, and $|\mathcal{R}|$. The state-value function can be decomposed into an expected immediate reward plus the discounted values of the successor state. To show this, we start from the definition of the state-value function (3.38a) and substitute $\mathsf{R}_k = \mathsf{r}_k + \gamma \mathsf{R}_{k+1}$ to get

$$\begin{aligned} V^\pi(\mathbf{x}) &= \mathbb{E}_\pi\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}\} \\ &= \mathbb{E}_\pi\{\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}\} + \gamma \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}\} \ . \end{aligned} \tag{3.42}$$

Note that the distribution $\mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x})$ is a marginal distribution of a distribution that also contains the variables $\mathbf{u}$ and $\mathbf{x}'$, respectively. Thus, we have

$$\mathsf{p}_\mathsf{r}(r \mid \mathbf{x}) = \sum_{\mathbf{x}'\in\mathcal{X}} \sum_{\mathbf{u}\in\mathcal{U}} \mathsf{p}(\mathbf{x}', \mathbf{u}, r_k \mid \mathbf{x}) = \sum_{\mathbf{x}'\in\mathcal{X}} \sum_{\mathbf{u}\in\mathcal{U}} \mathsf{p}(\mathbf{x}', r_k \mid \mathbf{u}, \mathbf{x}) \pi(\mathbf{u} \mid \mathbf{x}) \ , \tag{3.43}$$

where we used $\pi(\mathbf{u} \mid \mathbf{x}) \doteq \mathsf{p}(\mathbf{u} \mid \mathbf{x})$. The first part in (3.42) is the *expected reward* following π because

$$\begin{aligned} \bar{r}^\pi(\mathbf{x}) = \mathbb{E}_\pi\{\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}\} &= \sum_{r_k\in\mathcal{R}} r_k \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}) \\ &= \sum_{\mathbf{u}\in\mathcal{U}} \pi(\mathbf{u} \mid \mathbf{x}) \underbrace{\sum_{r_k\in\mathcal{R}} \sum_{\mathbf{x}'\in\mathcal{X}} r_k \, \mathsf{p}(\mathbf{x}', r_k \mid \mathbf{u}, \mathbf{x})}_{\overset{(3.17)}{=} \bar{r}(\mathbf{u}|\mathbf{x})} \ . \end{aligned} \tag{3.44}$$

Let us assume that $R_{k+1} \in \Gamma$ is also a random variable from the finite set Γ. Thus, for the second part in (3.42) follows after some reformulations

$$\begin{aligned} \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}\} &= \sum_{R_{k+1}\in\Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}) \\ &\overset{(A.12)}{=} \sum_{\mathbf{x}'\in\mathcal{X}} \sum_{R_{k+1}\in\Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}', \mathbf{x}) \mathsf{p}(\mathbf{x}' \mid \mathbf{x}) \\ &= \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}(\mathbf{x}' \mid \mathbf{x}) \underbrace{\sum_{R_{k+1}\in\Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}', \mathbf{x})}_{=\mathbb{E}_\pi\{\mathsf{R}_{k+1}|\mathbf{x}'=\mathbf{x}'\}=V^\pi(\mathbf{x}')} \\ &\overset{(A.12)}{=} \sum_{\mathbf{u}_l\in\mathcal{U}} \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \pi(\mathbf{u}|\mathbf{x}) V^\pi(\mathbf{x}') \\ &= \mathbb{E}_{\substack{\mathbf{x}'\sim\mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}}\{V^\pi(\mathbf{x}')\} \ . \end{aligned} \tag{3.45}$$

Finally, we have found the *discrete Bellman Expectation Equation of the state-value function*

$$V^\pi(\mathbf{x}) = \bar{r}^\pi(\mathbf{x}) + \gamma \mathbb{E}_{\substack{\mathbf{x}'\sim\mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}}\{V^\pi(\mathbf{x}')\} \tag{3.46a}$$

$$= \sum_{\mathbf{u}\in\mathcal{U}} \pi(\mathbf{u} \mid \mathbf{x}) \left[\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^\pi(\mathbf{x}') \right] \ . \tag{3.46b}$$

We can draw similar conclusions for the action-valve function. From (3.37a), we get

$$\begin{aligned} Q^\pi(\mathbf{x},\mathbf{u}) &= \mathbb{E}_\pi\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \\ &= \mathbb{E}_\pi[\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}] + \gamma \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \\ &= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) \sum_{\mathbf{u}\in\mathcal{U}} \pi(\mathbf{u}'|\mathbf{x}') Q^\pi(\mathbf{x}',\mathbf{u}') \ . \end{aligned} \tag{3.47}$$

Hence, in equivalence to (3.46), we find the *discrete Bellman Expectation Equation of the action-value function*

$$Q^\pi(\mathbf{x},\mathbf{u}) = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\substack{\mathbf{x}'\sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u}'\sim\pi(\mathbf{u}'|\mathbf{x}')}}\{Q^\pi(\mathbf{x}',\mathbf{u}')\} \tag{3.48a}$$

$$= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\{V^\pi(\mathbf{x}')\} \tag{3.48b}$$

$$= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) V^\pi(\mathbf{x}') \ . \tag{3.48c}$$

The last two reformulations are due to the relation (3.39).

Value-continuous case

Now suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *continuous sets*. The *continuous Bellman Expectation Equations* can be derived in a similar manner as previously demonstrated. For the *state-value function*, it is given by

$$V^\pi(\mathbf{x}_k) = \bar{r}^\pi(\mathbf{x}_k) + \gamma \mathbb{E}_{\substack{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k) \\ \mathbf{u}_k\sim\pi(\mathbf{u}_k|\mathbf{x}_k)}}\{V^\pi(\mathbf{x}_{k+1})\} \ . \tag{3.49}$$

The recursion for the *action-value function* is

$$Q^\pi(\mathbf{x}_k,\mathbf{u}_k) = \bar{r}(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\substack{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k) \\ \mathbf{u}_{k+1}\sim\pi(\mathbf{u}_{k+1}|\mathbf{x}_{k+1})}}\{Q^\pi(\mathbf{x}_{k+1},\mathbf{u}_{k+1})\} \tag{3.50a}$$

$$= \bar{r}(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)}\{V^\pi(\mathbf{x}_{k+1})\} \ . \tag{3.50b}$$

The last reformulation is once again due to the relation (3.39).

Example 3.7 (Deterministic Bellman Expectation equation). Furthermore, in the *deterministic case*, the Bellman Expectation Equation (3.49) simplifies to the Bellman Equation

$$V^\pi(\mathbf{x}_k) = \bar{r}^\pi(\mathbf{x}_k) + \gamma V^\pi(\mathbf{x}_{k+1}) \ . \tag{3.51}$$

Within this framework, Figure 3.7 illustrates the significance of the Bellman Equation. Here,

- $V^\pi(\mathbf{x}_k)$ can be viewed as a predicted performance,
- $\bar{r}^\pi(\mathbf{x}_k)$ represents the observed immediate reward, and
- $V^\pi(\mathbf{x}_{k+1})$ offers a current prediction of future control inputs.

> This concept is called *bootstrapping* in reinforcement learning. It refers to the idea of using the estimates of future value functions to update the current estimate of the value function. In other words, we're "bootstrapping" our value updates based on other estimated values – "Learn a guess from a guess". These concepts are further explored and utilized in the ensuing discussion on temporal difference learning.

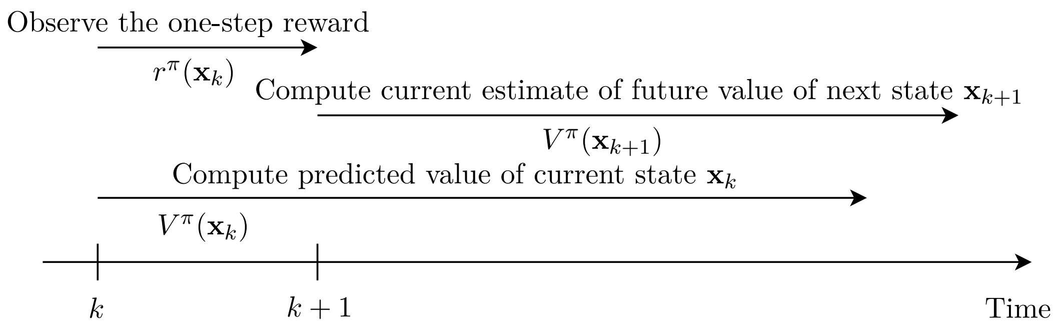

Observe the one-step reward
$r^\pi(\mathbf{x}_k)$
Compute current estimate of future value of next state $\mathbf{x}_{k+1}$
$V^\pi(\mathbf{x}_{k+1})$
Compute predicted value of current state $\mathbf{x}_k$
$V^\pi(\mathbf{x}_k)$
k
$k+1$
Time

Figure 3.7: The temporal difference perspective of the Bellman Equation illustrates how the equation embodies the processes of action-taking, observation, assessment, and enhancement inherent in reinforcement learning [3.10, p. 469].

3.2.8 Bellman Optimality Equation

The previously obtained results motivate the next theorem.

Value-discrete case

Theorem 3.2. *(Value-discrete Bellman Optimality Equation, see [3.11, p. 12]) Let us define the optimal action-state value function $Q^*: \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ by*

$$Q^*(\mathbf{x},\mathbf{u}) = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_\mathbf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\{V^*(\mathbf{x}')\} \ . \tag{3.52}$$

Then, Q^* satisfies the following Bellman Optimality Equation

$$Q^*(\mathbf{x},\mathbf{u}) = \bar{r}(\mathbf{x}',\mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_\mathbf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\left\{\max_{\mathbf{u}'\in\mathcal{U}} Q^*(\mathbf{x}',\mathbf{u}')\right\} \tag{3.53}$$

with

$$V^*(\mathbf{x}) = \max_{\mathbf{u}\in\mathcal{U}} Q^*(\mathbf{x},\mathbf{u}) \ . \tag{3.54}$$

Moreover, every policy $\pi \in \Pi_d$ such that

$$\boldsymbol{\mu}^*(\mathbf{x}) = \arg\max_{\mathbf{u}\in\mathcal{U}} Q^*(\mathbf{x}, \mathbf{u}) \tag{3.55}$$

is optimal.

Proof. Since Q^* is defined by (3.52), it follows that

$$\max_{\mathbf{u}\in\mathcal{U}} Q^*(\mathbf{x}, \mathbf{u}) = \max_{\mathbf{u}\in\mathcal{U}} \Big[\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \mathbb{E}_{\mathsf{x}' \sim \mathsf{p}_{\mathsf{x}}(\mathsf{x}'|\mathbf{x},\mathbf{u})} \{V^*(\mathbf{x}')\}\Big] = V^*(\mathbf{x}) \ . \tag{3.56}$$

This establishes (3.54) and (3.55). Substituting from (3.54) into (3.52) gives (3.53). □

Value-continuous case

In the value-continous case, we the *Bellman Optimality Equation* reads as

$$Q^*_k(\mathbf{x}_k, \mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathsf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)} \left\{ \max_{\mathbf{u}_{k+1}\in\mathcal{U}} Q^*_{k+1}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) \right\} \ . \tag{3.57}$$

Obviously, the Bellman Optimality Equation can be used to find the optimal value function V^* and optimal policy π^*.

3.3 Model-based reinforcement learning

In model-based reinforcement learning, we assume to know the dynamic and reward function $\mathsf{p_x}$ and $\mathsf{p_r}$, respectively.

3.3.1 Infinite horizon Dynamic Programming

Next we show the *Value and Policy Iteration* used in reinforcement learning. In this section, we will restrict our self to deterministic policies, the infinite horizon setting ($N = \infty$) and the value-discrete case. In this case, the value-discrete Bellman Expectation equation (3.50) with (3.19) can be written as

$$
\begin{aligned}
Q^\pi(\mathbf{x},\mathbf{u}) &= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p_x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) V^\pi(\mathbf{x}') \\
Q^\pi(\mathbf{x}_i,\mathbf{u}_l) &= \bar{r}(\mathbf{x}_i,\mathbf{u}_l) + \gamma \sum_{\mathbf{x}_j\in\mathcal{X}} \mathsf{p_x}(\mathbf{x}_j \mid \mathbf{x}_i,\mathbf{u}_l) V^\pi(\mathbf{x}_j) \\
&= \bar{r}(\mathbf{x}_i,\mathbf{u}_l) + \gamma \sum_{j=1}^{|\mathcal{X}|} p_{ij}^{\mathbf{u}_l} V^\pi(\mathbf{x}_j)\ .
\end{aligned}
\tag{3.58}
$$

Value Iteration

Value iteration is the computation of the state-value function V^π by the Dynamic Programming described in Theorem 3.2. The iteration is performed for the entire state space starting from arbitrarily initialized residual reward [3.6, p. 83]. The iteration rule in Theorem 3.2 represents a fixed point iteration for V^π and converges against V^*, see [3.9]. Therefore, we introduce the *state-value vector*

$$
\mathbf{v}^\pi = \begin{bmatrix} V^\pi(\mathbf{x}_0) & \dots & V^\pi\left(\mathbf{x}_{|\mathcal{X}|}\right) \end{bmatrix}^\top \in \mathbb{R}^{|\mathcal{X}|+1}
\tag{3.59}
$$

and define the *Bellman Iteration Map*[4] $\mathrm{B} : \mathbb{R}^{|\mathcal{X}|+1} \to \mathbb{R}^{|\mathcal{X}|+1}$ via

$$
\mathbf{y} \mapsto (\mathrm{B}(\mathbf{y}))[i] = \max_{\mathbf{u}_l\in\mathcal{U}} \left[\bar{r}(\mathbf{x}_i,\mathbf{u}_l) + \gamma \sum_{j=1}^{|\mathcal{X}|} p_{ij}^{\mathbf{u}_l} \mathbf{y}[j] \right].
\tag{3.60}
$$

Theorem 3.3. *(Theorem 3 in [3.11]) The map* B *is monotone and a contraction with respect to the ℓ_∞-norm. Therefore, the fixed point $\mathbf{v}^{(\infty)}$ of the map* B *satisfies the relation*

$$
\mathbf{v}^{(\infty)}[i] = \max_{\mathbf{u}_l\in\mathcal{U}} \left[\bar{r}(\mathbf{x}_i,\mathbf{u}_l) + \gamma \sum_{j=0}^{|\mathcal{X}|-1} p_{ij}^{\mathbf{u}_l} \mathbf{v}^{(\infty)}[j] \right]\ .
\tag{3.61}
$$

[4]Called Backup operator in the reinforcement learning literature.

See [3.11] for a proof. Given an initial guess $\mathbf{v}^{(0)} \in \mathbb{R}^{|\mathcal{X}|+1}$, one can iteratively employ the Bellman iteration. These iterations will converge to the unique fixed point, denoted as $\mathbf{v}^{(\infty)}$, of the operator B. The importance of this iterative process is elaborated in the subsequent theorem.

Theorem 3.4. *(Theorem 4 in [3.11]) Define $\mathbf{v}^{(\infty)} \in \mathbb{R}^{|\mathcal{X}|+1}$ to be the unique fixed point of* B*, and define $\mathbf{v}^* \in \mathbb{R}^{|\mathcal{X}|+1}$ to equal $V^*(\mathbf{x}), \mathbf{x} \in \mathcal{X}$, where $V^*(\mathbf{x})$ is defined in* (3.40)*. Then $\mathbf{v}^{(\infty)} = \mathbf{v}^*$.*

See [3.11] for a proof. Therefore, the optimal value vector can be computed using the Bellman iteration. However, knowing the optimal value vector does not, by itself, give us an optimal policy. Therefore, after the convergence Bellman iteration, a policy determination according to (3.55) is performed.

The *Infinite Horizon Value Iteration Algorithm* 1 show how to use Value Iteration to find an optimal policy π^*.

Algorithm 1: Infinite Horizon Value Iteration Algorithm

```
Data: $\theta$ is a small number
Result: Find $V^*(\mathbf{x})$ and $\boldsymbol{\mu}^*$ ,$\forall \mathbf{x} \in \mathcal{X}$
/* Initialization                                              */
1  Initialize $V(\mathbf{x})$ arbitrarily;
2  $V^\pi(\mathbf{x}) \leftarrow 0$ ,$\forall \mathbf{x} \in \mathcal{X}$;
/* Loop until convergence                                        */
3  $\Delta \leftarrow 0$;
/* Optimal Value Determination                                   */
4  while $\Delta < \theta$ do
5      for each $\mathbf{x} \in \mathcal{X}$ do
6          $v \leftarrow V^\pi(\mathbf{x})$;
7          $Q^\pi(\mathbf{x}) \leftarrow \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p_x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) V^\pi(\mathbf{x}')$ ;
8          $V^\pi(\mathbf{x}) \leftarrow \max_{\mathbf{u} \in \mathcal{U}} [Q^\pi(\mathbf{x})]$;
9          $\Delta \leftarrow \max(\Delta, |v - V^\pi(\mathbf{x})|)$;
10     end
11 end
/* Optimal Policy Determination                                  */
12 $Q^*(\mathbf{x},\mathbf{u}) \leftarrow \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p_x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) V^\pi(\mathbf{x}')$;
13 $\boldsymbol{\mu}^*(\mathbf{x}) \leftarrow \arg\max_{\mathbf{u} \in \mathcal{U}} Q^*(\mathbf{x},\mathbf{u})$;
14 return  $\mu^*$ ,$\forall \mathbf{x} \in \mathcal{X}$
```

Note 3.6. Python code of the *Infinite Horizon Value Iteration Algorithm* 1 for the Frozen Lake Example can be found here.

Policy Iteration

When the model is fully known, following Bellman Expectation equation, we can use Dynamic Programming (DP) to iteratively evaluate value functions and improve the policy.

- *Policy Evaluation* (value update) is to compute the state-value functions $V^{\pi}(\mathbf{x})$ for a given policy π according to the Bellman Expectation equation (3.49).
- Based on the state-value functions, *Policy Improvement* (policy update) generates a better policy $V^{\pi'}(\mathbf{x}) \geq V^{\pi}(\mathbf{x})$ by acting greedily.

Once again, we show Policy Iteration for the infinite horizon ($N = \infty$) and value-discrete case ($\mathcal{X}$ and $\mathcal{U}$ finite) only.

Policy Evaluation

Define the *state-value vector* $\mathbf{v}^{\pi} \in \mathbb{R}^{|\mathcal{X}|}$ via

$$\mathbf{v}^{\pi} = \begin{bmatrix} V^{\pi}(\mathbf{x}_0) & \dots & V^{\pi}\left(\mathbf{x}_{|\mathcal{X}|}\right) \end{bmatrix}^{\top}, \tag{3.62}$$

and the *reward vector* $\mathbf{r}^{\pi} \in \mathbb{R}^{|\mathcal{X}|+1}$ via

$$\mathbf{r}^{\pi} = \begin{bmatrix} \bar{r}_0^{\pi} & \cdots & \bar{r}_{|\mathcal{X}|}^{\pi} \end{bmatrix}^{\top} . \tag{3.63}$$

Then, the value-discrete infinite horizon Bellman Expectation equation (3.49) reads in vector notation as

$$\mathbf{v}^{\pi} = \mathbf{r}^{\pi} + \gamma \mathbf{P}^{\pi} \mathbf{v}^{\pi} . \tag{3.64}$$

Policy Evaluation involves computing the state-value vector $\mathbf{v}^{\pi}$ for a given policy π solving the Bellman Expectation equation. We also refer to it as the prediction problem. In this context, it is important to note that the induced matrix norm $\|\mathbf{P}^{\pi}\|_{\infty}$ is equal to 1, and since $\gamma < 1$, the matrix $\mathbf{I} - \gamma\mathbf{P}^{\pi}$ is nonsingular. Consequently, for any reward vector $\mathbf{r}^{\pi}$, there exists a unique $\mathbf{v}^{\pi}$ that satisfies (3.49). In principle it is possible to deduce from (3.49)

$$\mathbf{v}^{\pi} = (\mathbf{I} - \gamma\mathbf{P}^{\pi})^{-1}\mathbf{r}^{\pi} . \tag{3.65}$$

However, applying this formula can be challenging in practical applications of Markov Decision Problems, mainly due to the large size of the state space $\mathcal{X}$, represented by the integer $|\mathcal{X}|$. Additionally, matrix inversion has a cubic complexity, making it impractical to invert the matrix $\mathbf{I} - \gamma\mathbf{P}^{\pi}$. Consequently, alternative approaches need to be explored. The Contraction Mapping Theorem provides a feasible solution in such cases.

Theorem 3.5. *(Iterative policy evaluation, Theorem 2 in [3.11]) The map $\mathbf{y} \mapsto \mathrm{T}(\mathbf{y}) = \mathbf{r}^{\pi} + \gamma\mathbf{P}^{\pi}\mathbf{y}$ is monotone and is a contraction with respect to the ℓ_{∞}-norm, with contraction constant γ. Therefore, we can choose some vector $\mathbf{y}^{(0)}$ arbitrarily, and then define*

$$\mathbf{y}^{(i+1)} = \mathbf{r}^{\pi} + \gamma\mathbf{P}^{\pi}\mathbf{y}^{(i)} . \tag{3.66}$$

Then $\mathbf{y}^{(i)}$ converges to the state value vector $\mathbf{v}^{\pi}$.

See [3.11] for a proof. Similar results can be obtained for the value-continuous case, see [3.4]. Therefore, the transition tensor $\mathbf{P}^{\pi}$ is replaced by a transition operator.

Policy Improvement

Given a state-value function $V^\pi(\mathbf{x})$ of a policy π, it is possible to perform a *policy improvement step*, cf. [3.6, pp. 84–85]. The result is a potentially better strategy π', where $V^{\pi'}(\mathbf{x}) \geq V^\pi(\mathbf{x})$ holds. To obtain π', the previous strategy π is no longer followed, but instead the *greedy policy* is chosen, which maximizes the expected reward:

$$\pi' \leftarrow \arg\max_{\mathbf{u}\in\mathcal{U}} Q^\pi(\mathbf{x},\mathbf{u}) = \arg\max_{\mathbf{u}\in\mathcal{U}} \Big[\bar{r}^\pi + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_\mathbf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\big\{V^\pi(\mathbf{x}')\big\}\Big] \ . \tag{3.67}$$

This can be proven by the Policy Improvement Theorem 3.6:

Theorem 3.6. *(Policy Improvement Theorem) We consider two policies $\pi(\mathbf{u} \mid \mathbf{x})$ and $\pi'(\mathbf{u} \mid \mathbf{x})$. Let us define*

$$Q^\pi(\mathbf{x},\pi') = \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{Q^\pi(\mathbf{x},\mathbf{u})\} \ . \tag{3.68}$$

If $\forall \mathbf{x} \in \mathcal{X}$, we have that $Q^\pi(\mathbf{x},\pi') \geq V^\pi(\mathbf{x})$, then it holds that

$$V^\pi(\mathbf{x}) \leq Q^\pi(\mathbf{x},\pi') \leq V^{\pi'}(\mathbf{x}), \ \forall x \ . \tag{3.69}$$

This means that π' is at least as good a policy as π.

Proof. By expanding Q^π, we can get that $\forall \mathbf{x} \in \mathcal{X}$,

$$\begin{aligned} V^\pi(\mathbf{x}) \leq Q^\pi(\mathbf{x},\pi') &= \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{Q^\pi(\mathbf{x},\mathbf{u})\} \\ &= \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\left\{\mathsf{r}_k + \gamma \underbrace{V^\pi(\mathbf{x}')}_{\leq Q^\pi(\mathbf{x}',\pi')}\right\} \\ &\leq \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{\mathsf{r}_k + \gamma Q^\pi(\mathbf{x}',\pi')\} \\ &= \mathbb{E}_{\mathbf{u},\mathbf{u}'\sim\pi'}\Big\{\mathsf{r}_k + \gamma \mathsf{r}_{k+1} + \gamma^2 V^\pi(\mathbf{x}'')\Big\} \\ &\leq \dots \\ &\leq \mathbb{E}_{\mathbf{u},\mathbf{u}',\mathbf{u}''\dots\sim\pi'}\Big\{\mathsf{r}_k + \gamma \mathsf{r}_{k+1} + \gamma^2 \mathsf{r}_{k+2} + \dots\Big\} \\ &= V^{\pi'}(\mathbf{x}) \ . \end{aligned} \tag{3.70}$$

□

The *Infinite Horizon Policy Iteration Algorithm* 2 shows how to use policy evaluation and policy improvement to find an optimal policy π^*.

Algorithm 2: Infinite Horizon Policy Iteration Algorithm

```
Data: $\theta$ is a small number
/* Initialization                                          */
1  Initialize $V^\pi(\mathbf{x})$ arbitrarily;
2  Randomly initialize policy $\boldsymbol{\mu}$;
/* Policy Evaluation                                       */
3  $\Delta \leftarrow 0$;
4  while $\Delta < \theta$ do
5      for each $\mathbf{x} \in \mathcal{X}$ do
6          $v^\pi \leftarrow V^\pi(\mathbf{x})$;
7          $V^\pi(\mathbf{x}) \leftarrow \bar{r}^\pi + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^\pi(\mathbf{x}')$;
8          $\Delta \leftarrow \max(\Delta, |v^\pi - V^\pi(\mathbf{x})|)$;
9      end
10 end
/* Policy Improvement                                      */
11 policy is stable $\leftarrow$ true;
12 for each $\mathbf{x} \in \mathcal{X}$ do
13     old-policy $\leftarrow \pi$;
14     $Q^\pi(\mathbf{x}, \mathbf{u}) \leftarrow \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^\pi(\mathbf{x}')$;
15     $\boldsymbol{\mu}(\mathbf{x}) \leftarrow \arg\max_{\mathbf{u} \in \mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{u})$;
16     if old-policy $\neq \pi$ then
17         policy is stable $\leftarrow$ false;
18     end
19 end
20 if policy is stable then
21     return $V^\pi \approx V^*$ and $\pi \approx \pi^*$;
22 else
23     go to Policy Evaluation;
24 end
```

Finally, we compare Value Iteration and Policy Iteration and draw some conclusions:

Aspect	**Value Iteration**	**Policy Iteration**
Convergence Speed	Faster to optimal value function	Fewer iterations required and converges to the optimal policy faster
Implementation	Simpler, no separate policy needed	More complex with policy evaluation and update
Efficiency per Iteration	Slower due to all states sweep	More efficient with early policy stop
Computational Expense	Varied, generally moderate	Computationally expensive per iteration

Table 3.3: Comparison of Value Iteration and Policy Iteration.

Note 3.7. Python code of the *Infinite Horizon Policy Iteration Algorithm* 2 for the Frozen Lake Example can be found here.

We will now have a look at a value-continuous example, in particular we investigate the well known *Linear Quadratic Regulator* (LQR) problem from Control Engineering.

Example 3.8 (The Linear Quadratic Regulator problem - infinite time horizon and deterministic case). Consider the problem of the discrete-time linear quadratic regulator (LQR), characterized by deterministic dynamics

$$\mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\mathbf{u}_k \ , \tag{3.71}$$

with k the discrete time index. Let the pair $(\mathbf{\Phi}, \mathbf{\Gamma})$ be reachable. In this example, the state space $\mathcal{X} = \mathbb{R}^n$ and input space $\mathcal{U} = \mathbb{R}^m$ are infinite. The return over an infinite horizon, based on deterministic immediate rewards r_i, is expressed as

$$R_k(\boldsymbol{\tau}_{[k:\infty]}) = \frac{1}{2}\sum_{i=k}^{\infty} r_i(\mathbf{x}_i, \mathbf{u}_i) = \frac{1}{2}\sum_{i=k}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \mathbf{u}_i^\top \mathbf{R}\mathbf{u}_i\Big) \ . \tag{3.72}$$

The return R_k depends on all future states $\mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ and all future control inputs $\mathbf{u}_k, \mathbf{u}_{k+1}, \ldots$. We select a stationary control law $\mathbf{u}_k = \boldsymbol{\mu}(\mathbf{x}_k)$ and write the associated action-value function as

$$Q^\pi(\mathbf{x}_k, \mathbf{u}_k) = r(\mathbf{x}_k, \mathbf{u}_k) + V^\pi(\mathbf{x}_{k+1}) \tag{3.73a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.73b}$$

and the associated state-value function as

$$V^\pi(\mathbf{x}_k) = \frac{1}{2}\sum_{i=k}^{\infty} r_i^\pi = \frac{1}{2}\sum_{i=k}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \boldsymbol{\mu}^\top(\mathbf{x}_i)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_i)\Big) \tag{3.74a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.74b}$$

Note that the state-value function for a fixed control law depends only on the initial state $\mathbf{x}_k$. A difference equation equivalent to the infinite sum (3.74a) is given by

$$\begin{aligned} V^\pi(\mathbf{x}_k) &= \frac{1}{2}\Big(\mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^\top(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k)\Big) + \frac{1}{2}\sum_{i=k+1}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \boldsymbol{\mu}^\top(\mathbf{x}_i)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_i)\Big) \\ &= \frac{1}{2}\Big(\mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^\top(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k)\Big) + V^\pi(\mathbf{x}_{k+1}) \ . \end{aligned} \tag{3.75}$$

Equation (3.75) is exactly the Bellman Equation (3.49) for the LQR problem. Assuming a quadratic state-value function

$$V^\pi(\mathbf{x}_k) = \frac{1}{2}\mathbf{x}_k^\top \mathbf{P}\mathbf{x}_k \ , \tag{3.76}$$

with positive-definite matrix $\mathbf{P}$, yields the Bellman Equation

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{P}\mathbf{x}_k = \mathbf{x}_k^{\top}\mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^{\top}(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k) + \mathbf{x}_{k+1}^{\top}\mathbf{P}\mathbf{x}_{k+1} \ , \tag{3.77}$$

which, using the state equation (3.71), can be written as

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^{\top}(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k) + (\boldsymbol{\Phi}\mathbf{x}_k + \boldsymbol{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k))^{\top}\mathbf{P}(\boldsymbol{\Phi}\mathbf{x}_k + \boldsymbol{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k)) \ . \tag{3.78}$$

Assuming a stationary state feedback control law $\boldsymbol{\mu}(\mathbf{x}_k) = -\mathbf{K}\mathbf{x}_k$ with control gain $\mathbf{K}$, we get

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{P}\mathbf{x}_k = \mathbf{x}_k^{\top}\mathbf{Q}\mathbf{x}_k + \mathbf{x}_k^{\top}\mathbf{K}^{\top}\mathbf{R}\mathbf{K}\mathbf{x}_k + \mathbf{x}_k^{\top}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})^{\top}\mathbf{P}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})\mathbf{x}_k \ . \tag{3.79}$$

For all state trajectories, we find

$$(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})^{\top}\mathbf{P}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K}) - \mathbf{P} + \mathbf{Q} + \mathbf{K}^{\top}\mathbf{R}\mathbf{K} = \mathbf{0} \ . \tag{3.80}$$

This is a Lyapunov equation in Josephs form. That is, the Bellman Equation (3.77) for the discrete-time LQR problem is equivalent to a Lyapunov equation.

3.3.2 Finite horizon Dynamic Programming

Finite horizon Dynamic Programming (DP) is based on Bellman's optimality principle, which states that every optimal solution of (3.40) is composed of optimal partial solutions. The basic idea of DP is to divide the problem into subproblems, which can be solved more easily, and to assemble these partial solutions into the overall solution. We will show the value-continuous Dynamic Programming Algorithm for MDPs only.

Theorem 3.7. *(Bellman's Optimality Principle, see [3.3, p. 20]).*
Let $\pi^* = \left(\boldsymbol{\mu}_0^*, \boldsymbol{\mu}_1^*, \dots, \boldsymbol{\mu}_{N-1}^*\right)$ *be the optimal policy for*

$$V_0^{\pi}(\mathrm{x}_0) = \mathbb{E}_{\pi}\left\{\mathsf{R}_0\left(\boldsymbol{\tau}_{[0:N]}\right) \mid \mathbf{x}_0 = \mathrm{x}_0\right\} \ . \tag{3.81}$$

Considering the subproblem of (3.81) *where, starting from the state* x_{k+1}*, the expected returns*

$$V_{k+1}^{\pi}(\mathrm{x}_{k+1}) = \mathbb{E}_{\pi}\left\{\mathsf{R}_{k+1}\left(\boldsymbol{\tau}_{[k+1:N]}\right) \mid \mathbf{x}_{k+1} = \mathrm{x}_{k+1}\right\} \ . \tag{3.82}$$

are to be maximized, then the truncated policy $\left(\boldsymbol{\mu}_{k+1}^*, \boldsymbol{\mu}_{k+2}^*, \dots, \boldsymbol{\mu}_{N-1}^*\right)$ *is optimal for the subproblem.*

The underlying idea behind the Principle of Optimality is straightforward. If the truncated policy $\left(\boldsymbol{\mu}_{k+1}^*, \boldsymbol{\mu}_{k+2}^*, \dots, \boldsymbol{\mu}_{N-1}^*\right)$ were not optimal as stated, we would be able to increase the return further by switching to an optimal policy for the subproblem once we reach

$\mathbf{x}_{k+1}$. Based on Theorem 3.2, the *Dynamic Programming Algorithm* can be stated. The algorithm constructs optimal value functions

$$V_N^*(\mathbf{x}_N), V_{N-1}^*(\mathbf{x}_{N-1}), , \dots, V_0^*(\mathbf{x}_0) \tag{3.83}$$

starting from $V_N^*(\mathbf{x}_N)$ and proceeding backwards in time.

Proposition 3.1. (Dynamic Programming Algorithm, see [3.3, p. 23] - backward pass). For every initial state $\mathbf{x}_0$, the maximum expected return $V_0^*(\mathbf{x}_0)$ of the basic problem is equal to $V_0^\pi(\mathbf{x}_0)$, given by the last step of the following algorithm, which proceeds from $\gamma^N \bar{r}_N$ backward in time from $k = N-1$ to $k = 0$, for all $\mathbf{x}_k \in \mathcal{X}$:

$$\begin{aligned} V_N^*(\mathbf{x}_N) &= \gamma^N \bar{r}_N(\mathbf{x}_N) \\ V_k^*(\mathbf{x}_k) &= \max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k, \mathbf{u}_k) \ , \end{aligned} \tag{3.84}$$

with

$$Q_k^*(\mathbf{x}_k, \mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} | \mathbf{x}_k, \mathbf{u}_k)} \{ V_{k+1}^*(\mathbf{x}_{k+1}) \} \ . \tag{3.85}$$

Furthermore, if $\mathbf{u}_k^* = \boldsymbol{\mu}_k^*(\mathbf{x}_k)$ satisfies the Bellman iteration (3.84) for each $\mathbf{x}_k$ and k, the policy $\pi^* = \left(\boldsymbol{\mu}_0^*, \boldsymbol{\mu}_1^*, \dots, \boldsymbol{\mu}_{N-1}^*\right)$ is optimal.

Once the value functions $V_0^*(\mathbf{x}_0), \dots, V_N^*(\mathbf{x}_N)$ have been obtained, we can use the following forward algorithm to construct an optimal control input sequence $\left(\mathbf{u}_0^*, \mathbf{u}_1^*, \dots, \mathbf{u}_{N-1}^*\right)$ and the corresponding state trajectory $\left(\mathbf{x}_0^*, \mathbf{x}_1^*, \dots, \mathbf{x}_{N-1}^*\right)$ for a given initial state $\mathbf{x}_0$.

Proposition 3.2. (Construction of an optimal rollout, see [3.7, p. 12] - forward pass). Starting at the initial condition $\mathsf{x}_0^* \sim \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0^*)$, compute forward in time from $k = 0$ to $k = N-1$ the control input via

$$\mathbf{u}_k^* = \boldsymbol{\mu}_k^*(\mathbf{x}_k^*) = \arg\max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k^*, \mathbf{u}_k) \tag{3.86}$$

with

$$Q_k^*(\mathbf{x}_k^*, \mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k^*, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}\left(\mathbf{x}_{k+1}^* | \mathbf{x}_k^*, \mathbf{u}_k\right)} \{ V_{k+1}^*(\mathbf{x}_{k+1}) \} \tag{3.87}$$

and optimal state trajectory according to

$$\mathbf{x}_{k+1}^* \sim \mathsf{p}_\mathsf{x}\left(\mathbf{x}_{k+1}^* \mid \mathbf{x}_k^*, \mathbf{u}_k^*\right) \ . \tag{3.88}$$

See [3.3, p. 25] and [3.7, p. 11] for more information on the Dynamic Programming Algorithm. The *Finite Horizon Value Iteration Algorithm* 3 shows how to use Value Iteration to find an optimal policy π^*.

Algorithm 3: Finite Horizon Value Iteration Algorithm

```
   Data: $\epsilon$ is a small number
   Result: Find $V_k^*(\mathbf{x}_k)$ and $\pi^*$ ,$\forall \mathbf{x}_k \in \mathcal{X}$
   /* Initialization                                   */
 1 Initialize $V_k^\pi(\mathbf{x}_k)$ arbitrarily, expect the $V_N^\pi(\mathbf{x}_N)$;
 2 $V_k^\pi(\mathbf{x}_k) \leftarrow 0, \forall \mathbf{x}_k \in \mathcal{X}$;
   /* Loop until convergence                             */
 3 $\Delta \leftarrow 0$;
   /* Optimal Value Determination                        */
 4 for $k = N-1, \dots, 0$ do
 5     for each $\mathbf{x}_k \in \mathcal{X}$ do
 6         $v_k \leftarrow V_k^\pi(\mathbf{x}_k)$;
 7         $Q_k^\pi(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathsf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathsf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)} \left\{ V_{k+1}^\pi(\mathsf{x}_{k+1}) \right\}$;
 8         $V_k^\pi(\mathbf{x}_k) \leftarrow \max_{\mathbf{u}_k \in \mathcal{U}} Q_k^\pi(\mathbf{x}_k, \mathbf{u}_k)$;
 9         $\Delta \leftarrow \max(\Delta, |v_k - V_k^\pi(\mathbf{x}_k)|)$;
10     end
11 end
   /* Optimal Policy Determination                       */
12 $Q_k^*(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathsf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathsf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)} \left\{ V_{k+1}^*(\mathsf{x}_{k+1}) \right\}$;
13 $\boldsymbol{\mu}_k^*(\mathbf{x}_k) \leftarrow \arg\max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k, \mathbf{u}_k)$;
14 return $\pi^*$ ,$\forall \mathbf{x}_k \in \mathcal{X}$
```

Example 3.9 (The Linear Quadratic Regulator problem - finite time horizon and stochastic case). In the discrete-time LQR design, the optimal policy π^* is determined for a linear time-invariant system with discrete-time evolution

$$\mathsf{x}_{k+1} = \boldsymbol{\Phi}_k \mathsf{x}_k + \boldsymbol{\Gamma}_k \mathbf{u}_k + \mathsf{w}_k \ . \tag{3.89}$$

The disturbance w_k is characterized by the following properties, see [3.12],

$$\begin{aligned} \mathbb{E}\{\mathsf{w}_k\} &= \mathbf{0} \\ \mathbb{E}\left\{\mathsf{w}_k \mathsf{w}_k^\top\right\} &= \mathbf{W} \ . \end{aligned} \tag{3.90}$$

The deterministic rewards are given by

$$r_k(\mathbf{x}_k, \mathbf{u}_k) = \frac{1}{2}(\mathbf{x}_k^\top \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^\top \mathbf{R}_k \mathbf{u}_k) \tag{3.91}$$

$$r_N(\mathbf{x}_N) = \frac{1}{2}\mathbf{x}_N^\top \mathbf{Q}_N \mathbf{x}_N \ , \tag{3.92}$$

where $\mathbf{Q}_k \in \mathbb{R}^{n \times n}$ is positive semi-definite and $\mathbf{R}_k \in \mathbb{R}^{m \times m}$ is positive definite for all $k = 0, \dots, N$. To determine the optimal policy π^* in the form

$$\mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) = -\mathbf{K}_k \mathbf{x}_k \ , \tag{3.93}$$

we apply the Dynamic Programming Algorithm from Theorem 3.1.

Starting from the terminal reward $r_N(\mathbf{x}_N)$, a backward recursion is performed based on the Bellman iteration (3.84). We initialize by

$$V_N(\mathbf{x}_N) = \frac{1}{2}\mathbf{x}_N^\top \mathbf{Q}_N \mathbf{x}_N \equiv \frac{1}{2}\mathbf{x}_N^\top \mathbf{P}_N \mathbf{x}_N \ . \tag{3.94}$$

Substituting the system dynamics $\mathbf{x}_N = \mathbf{\Phi}_k \mathbf{x}_{N-1} + \mathbf{\Gamma}_k \mathbf{u}_{N-1} + \mathbf{w}_{N-1}$ results in

$$\begin{aligned} V_{N-1}(\mathbf{x}_{N-1}) &= \min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \left[r_{N-1}(\mathbf{x}_{N-1}\ , \mathbf{u}_{N-1}) + \mathbb{E}_{\mathbf{w}_{N-1}}\{V_N^*(\mathbf{x}_N)\}\right] \\ &= \frac{1}{2}\min_{\mathbf{u}_{N-1}\in\mathbb{R}^m}\left[\mathbf{x}_{N-1}^\top \mathbf{Q}_{N-1}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^\top \mathbf{R}_{N-1}\mathbf{u}_{N-1} + \mathbf{x}_N^\top \mathbf{P}_N \mathbf{x}_N\right] \ . \end{aligned} \tag{3.95}$$

After rearranging, we have

$$\begin{aligned} V_{N-1}^*(\mathbf{x}_{N-1}) = \frac{1}{2}\min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \Big[& \mathbf{x}_{N-1}^\top(\mathbf{Q}_{N-1} + \mathbf{\Phi}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1})\mathbf{x}_{N-1} \\ & + \mathbf{u}_{N-1}^\top(\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1})\mathbf{u}_{N-1} \\ & + 2\mathbf{u}_{N-1}^\top(\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1})\mathbf{x}_{N-1} \\ & + \mathbb{E}_{\mathbf{w}_{N-1}}\Big\{\mathbf{w}_{N-1}^\top \mathbf{P}_N \mathbf{w}_{N-1}\Big\}\Big] \ . \end{aligned} \tag{3.96}$$

Note that this optimization problem is convex in $\mathbf{u}_{N-1}$ as $\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1} > 0$. Therefore, any local minimum is a global minimum, and therefore we can simply apply the first order optimality conditions. Differentiating gives

$$\left(\frac{\partial}{\partial \mathbf{u}_{N-1}} V_{N-1}^*\right)(\mathbf{x}_{N-1}) = (\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1})\mathbf{u}_{N-1} + (\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1})\mathbf{x}_{N-1} \tag{3.97}$$

and setting this to zero yields

$$\mathbf{u}_{N-1}^* = -(\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1})^{-1}(\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1})\mathbf{x}_{N-1} \ , \tag{3.98}$$

which we write

$$\mathbf{u}_{N-1}^* = -\mathbf{K}_{N-1}\mathbf{x}_{N-1} \ , \tag{3.99}$$

which is a time-varying linear feedback control law. Plugging this feedback policy into (3.96) results in

$$\begin{aligned} V_{N-1}^*(\mathbf{x}_{N-1}) = \mathbf{x}_{N-1}^\top(&\mathbf{Q}_{N-1} + \mathbf{K}_{N-1}^\top \mathbf{R}_{N-1}\mathbf{K}_{N-1} \\ &+ (\mathbf{\Phi}_{N-1} + \mathbf{\Gamma}_{N-1}\mathbf{K}_{N-1})^\top \mathbf{P}_N(\mathbf{\Phi}_{N-1} + \mathbf{\Gamma}_{N-1}\mathbf{K}_{N-1}))\mathbf{x}_{N-1} \ . \end{aligned} \tag{3.100}$$

Critically, this implies that the cost is always a positive semi-definite quadratic function of the state. Because the optimal policy is always linear, and the optimal cost is always quadratic, the DP recursion may be recursively performed backward in time and the minimization may be performed analytically. Following the same procedure, we can write the DP recursion for the discrete-time LQR controller. The optimal feedback gain $\mathbf{K}_k$ according to control law $\mathbf{u}_k^* = -\mathbf{K}_k\mathbf{x}_k$ is obtained from a *backward calculation* from $k = N-1$ to $k = 0$ starting from $\mathbf{P}_N = \mathbf{Q}_N$:

$$\mathbf{K}_k = (\mathbf{R}_k + \mathbf{\Gamma}_k^\top \mathbf{P}_{k+1}\mathbf{\Gamma}_k)^{-1}(\mathbf{\Gamma}_k^\top \mathbf{P}_{k+1}\mathbf{\Phi}_k) \tag{3.101}$$

$$\mathbf{P}_k = \mathbf{Q}_k + \mathbf{K}_k^\top \mathbf{R}_k\mathbf{K}_k + (\mathbf{\Phi}_k + \mathbf{\Gamma}_k\mathbf{K}_k)^\top \mathbf{P}_{k+1}(\mathbf{\Phi}_k + \mathbf{\Gamma}_k\mathbf{K}_k)\ . \tag{3.102}$$

To determine the optimal control sequence $\mathbf{U}_{[0:N-1]}^* = \left(\mathbf{u}_0^*, \mathbf{u}_1^*, \dots, \mathbf{u}_{N-1}^*\right)$ and optimal state sequence $\mathbf{X}_{[0:N]}^* = (\mathbf{x}_0^*, \mathbf{x}_1^*, \dots, \mathbf{x}_N^*)$, a *forward calculation* from $k = 0$ to $k = N-1$ is performed starting from $\mathbf{x}_0 = \mathbf{x}_0^*$ using the optimal control law and discrete-time evolution

$$\begin{aligned}
\mathbf{u}_0^* &= -\mathbf{K}_0\mathbf{x}_0^* & &\rightarrow & \mathbf{x}_1^* &= \mathbf{\Phi}_k\mathbf{x}_0^* + \mathbf{\Gamma}_k\mathbf{u}_0^* \\
&\vdots & & & &\vdots \\
\mathbf{u}_{N-1}^* &= -\mathbf{K}_{N-1}\mathbf{x}_{N-1}^* & &\rightarrow & \mathbf{x}_N^* &= \mathbf{\Phi}_k\mathbf{x}_{N-1}^* + \mathbf{\Gamma}_k\mathbf{u}_{N-1}^*\ .
\end{aligned} \tag{3.103}$$

The relation offers multiple insights. Even when $\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{Q}, \mathbf{R}$ are constant, the policy is still time-varying. This phenomenon arises because early control efforts lead to reduced state costs in all subsequent time steps. As the episode nears its end, the benefits of this early control wane, and the control effort decrease. However, for a linear time-invariant system where $(\mathbf{\Phi}, \mathbf{\Gamma})$ is reachable, the feedback gain $\mathbf{K}_k$ approach a constant value as the episode length grows infinitely.

3.4 Model-free reinforcement learning

The methods described before require knowledge of the system dynamics. However, this is not the case in model-free reinforcement learning, where the dynamics are unknown. As a result, it becomes necessary to develop solution methods that do not rely on this explicit knowledge.

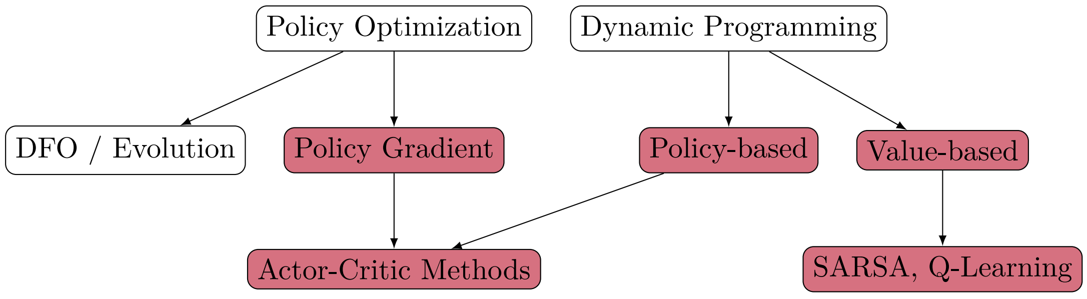


Figure 3.8: Classification of model-free RL algorithms [3.13, p. 4].

In reinforcement learning (RL), there are numerous options for approximating elements like policies, value functions, and dynamic models. Figure 3.8 provides a *classification of model-free RL algorithms.* There are three main approaches: (i.) *Policy Optimization*, (ii.) *Approximate Dynamic Programming*, and (iii.) *Hybrid methods.*

Policy Optimization techniques focus on optimizing the policy, the function mapping the agent's state to its next control input. They consider reinforcement learning as a numerical optimization problem where we optimize the expected reward with respect to the policy's parameters. There are two ways to optimize a policy: *Derivative-free Optimization* (DFO) algorithms and policy gradient methods. DFO algorithms work by perturbing the policy parameters, assessing the performance, and then moving towards better performance. They're simple to implement but scale poorly with increased parameters. On the other hand, *Policy Gradient Methods* estimate the policy improvement direction using various quantities measured by the agent, thus avoiding parameter perturbation. They are more complex to implement but can optimize larger policies than DFO algorithms.

Approximate Dynamic Programming (ADP), is based on learning value functions, predicting the agent's potential reward. ADP algorithms strive to satisfy certain consistency equations that the true value functions obey. Known algorithms for exact solutions in finite state-input RL problems are *policy-based* and *value-based.* They can be combined with function approximation in multiple ways; currently, leading descendents of value iteration focus on *approximating Q-functions.*

Lastly, *Actor-Critic* methods incorporate elements from both policy optimization and dynamic programming. They optimize a policy using value functions for faster optimization and often utilize ideas from approximate dynamic programming for fitting the value functions.

We discuss five key model-free reinforcement learning algorithms. Table 3.4 summarizes the use of these model-free reinforcement learning algorithms. It categorizes these algorithms based on their applicability in discrete and continuous input spaces. On-policy and off-policy learning refer to how an algorithm learns a policy. On-policy learning improves the policy used to make decisions, while off-policy learning improves a different policy from the one used to generate data.

Algorithm	Policy	Discrete	Continuous	Tabular
Monte Carlo	on	Yes	No	Yes
SARSA	on	Yes	No	Yes
Q-Learning	off	Yes	No	Yes
Deep Q-Network	off	Yes	Yes	No
Policy Gradient	on/off	Yes	Yes	No

Table 3.4: Use of reinforcement learning algorithms in different input spaces.

In the following section, we will specifically focus on what are known as *tabular solution methods*, see [3.6, p. 23], which are applicable in scenarios with an infinite horizon as well as discrete input and state spaces.

3.4.1 Generalized Policy Iteration

In model-free RL, the use of the action-value function $Q^\pi(\mathbf{x},\mathbf{u})$ is favored over the state-value function $V^\pi(\mathbf{x})$ because $Q^\pi(\mathbf{x},\mathbf{u})$ eliminates the need to know the state-transition and reward function explicitly, which is consistent with the goal of model-free RL to learn optimal policies without a model of the environment, cf. (3.67). Let us introduce the action-value table/matrix

$$\mathbf{Q}^\pi = \begin{bmatrix} Q^\pi(\mathbf{x}_0,\mathbf{u}_0) & Q^\pi(\mathbf{x}_0,\mathbf{u}_1) & \dots & Q^\pi\left(\mathbf{x}_0,\mathbf{u}_{|\mathcal{U}|-1}\right) \\ Q^\pi(\mathbf{x}_1,\mathbf{u}_0) & Q^\pi(\mathbf{x}_1,\mathbf{u}_1) & \dots & Q^\pi\left(\mathbf{x}_1,\mathbf{u}_{|\mathcal{U}|-1}\right) \\ \vdots & \ddots & \vdots & \\ Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1},\mathbf{u}_0\right) & Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1},\mathbf{u}_1\right) & \dots & Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1},\mathbf{u}_{|\mathcal{U}|-1}\right) \end{bmatrix}. \tag{3.104}$$

The *Generalized Policy Iteration* (GPI) algorithm refers to an iterative procedure to improve the policy when combining policy evaluation and improvement, cf. Figure 3.9:

$$\pi^{(0)} \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(0)} \xrightarrow{\text{improve}} \pi^{(1)} \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(1)} \dots \xrightarrow{\text{improve}} \pi^* \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(*)} \tag{3.105}$$

In GPI, the action-value function $\mathbf{Q}^\pi$ is approximated repeatedly to be closer to the true value $(\mathbf{Q}^\pi)^{(*)}$ of the current policy and in the meantime, the policy π is improved repeatedly to approach optimality, i.e., π^*. This policy iteration process works and always converges to the optimality, independent of the granularity and other details of the two processes. Almost all reinforcement learning methods are well described as GPI [3.6, p. 86].

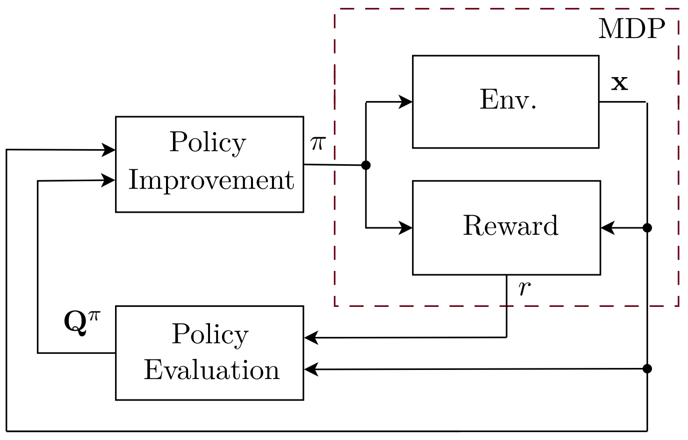


Figure 3.9: Generalized Policy Iteration.

Exploration vs. Exploitation

Efficient decision-making is often challenged by the need to balance exploration and exploitation. Exploration involves seeking new information to improve future decisions, while exploitation leverages current knowledge to maximize immediate rewards.

- Greedy input: When an agent chooses an input with the highest estimated value at the moment. The agent exploits its current knowledge by selecting the greedy input.
- Non-greedy input: When the agent does not choose the action with the highest estimated value and forgoes the immediate reward, hoping to gather more information about other inputs.
- Exploration: This enables the agent to improve its knowledge of each input, which hopefully leads to long-term benefits.
- Exploitation: The agent can choose the greedy input to obtain the highest reward for a short-term advantage. However, a purely greedy input selection can result in suboptimal behavior.

This creates a dilemma between exploration and exploitation, as an agent cannot explore and exploit simultaneously. Striking a balance between *exploration* and *exploitation* is crucial in model-free RL.

Policy Improvement

Policy improvement refers to acting greedily and exploiting current knowledge. In view of (3.67), given an action-value function $Q^\pi(\mathbf{x}, \mathbf{u})$, the greedy policy

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \begin{cases} 1 & \text{if } \mathbf{u} = \arg\max_{\mathbf{w}\in\mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w}) \\ 0 & \text{else} \end{cases}, \tag{3.106}$$

is selected to maximize the expected reward based on current knowledge. Several strategies exist to balance exploration and exploitation:

ϵ-Greedy Method
With ϵ-greedy, at each step in state $\mathbf{x}$, the agent selects a random input with a fixed probability $\epsilon \in [0, 1]$:

- With probability $\frac{\epsilon}{|\mathcal{U}|} + 1 - \epsilon$, choose $\mathbf{u} = \arg\max_{\mathbf{w}\in\mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w})$ and
- with probability $\frac{\epsilon}{|\mathcal{U}|}$, select any input uniformly at random.

Hence, the ϵ-greedy policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \begin{cases} \frac{\epsilon}{|\mathcal{U}|} + 1 - \epsilon & \text{if } \mathbf{u} = \arg\max_{\mathbf{w}\in\mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w}) \\ \frac{\epsilon}{|\mathcal{U}|} & \text{else} \end{cases}. \tag{3.107}$$

Compared to the greedy method (3.67), the ϵ-greedy method helps in exploring the input space.

Softmax Strategy
A downside of ϵ-greedy exploration is that it randomly picks any possible input for exploration. This means it's just as likely to pick the worst option as it is to pick the almost best one. In contrast, the softmax[5] strategy utilizes input-selection probabilities which are determined by ranking the action-value function estimates using a Boltzmann distribution. The softmax policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \operatorname*{soft\,max}_{\mathbf{u}\in\mathcal{U}} \beta Q^\pi(\mathbf{x}, \mathbf{u}) = \frac{\exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x}, \mathbf{u})\right)}{\sum_{\mathbf{w}\in\mathcal{U}} \exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x}, \mathbf{w})\right)} . \tag{3.108}$$

Here, β is the temperature parameter that controls the stochasticity of the policy. A low β makes the policy more deterministic, choosing the action with the highest Q-value with higher probability. A high β makes the policy more exploratory, distributing the selection probability more uniformly across inputs. In practice, the agent uses the Softmax Strategy to randomly select the next input by sampling from the distribution.

[5] Actually, the softmax function is a softened version of the argmax function. Therefore, it could be more appropriately termed softargmax.

Upper Confidence Bound
The Upper Confidence Bound (UCB) strategy addresses the exploration-exploitation trade-off by selecting inputs based on both their estimated value and the uncertainty of those estimates. It adds a term to the action-value function that increases with the uncertainty of the action value, encouraging exploration of less certain actions. The UCB policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \arg\max_{\mathbf{w}\in\mathcal{U}} \left(Q^\pi(\mathbf{x},\mathbf{w}) + c\sqrt{\frac{\ln(t)}{N(\mathbf{x},\mathbf{w})}} \right) , \tag{3.109}$$

where

- c is a parameter that controls the degree of exploration.
- t is the total number of times the state $\mathbf{x}$ has been visited.
- $N(\mathbf{x},\mathbf{w})$ is the number of times input $\mathbf{w}$ has been chosen in state $\mathbf{x}$.

The exploration term $c\sqrt{\frac{\ln t}{N(\mathbf{x},\mathbf{w})}}$ decreases as $N(\mathbf{x},\mathbf{w})$ increases, reducing exploration over time as more information is gathered. This method balances the need to explore actions with high uncertainty and the need to exploit actions with high estimated rewards.

Policy Evaluation

If the dynamics of the environment are unknown, several methods can be used for *policy evaluation*. In policy evaluation, we apply the update rule

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow \hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \alpha\delta \tag{3.110}$$

to improve the estimated action-value function $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$. Here, $\alpha \in (0,1]$ is the learning rate and δ denotes a residual. From a control engineering perspective, (3.110) is an integrator. It updates $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ until the residual δ approaches zero.

Monte Carlo methods
The principle of Monte Carlo (MC) methods is to sample rollouts $\boldsymbol{\tau}$ using the current policy π and approximate the expectation $Q^\pi(\mathbf{x},\mathbf{u}) = \mathbb{E}_\pi\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}$ to compute the empirical mean return[6]

$$\hat{R}_k = \sum_{j=k}^{N} \gamma^{j-k} r_j \tag{3.111}$$

for each state-input pair. Then, with residual

$$\delta = \frac{1}{N(\mathbf{x},\mathbf{u})} \sum_{k=1}^{N(\mathbf{x},\mathbf{u})} \left[\hat{R}_k - \hat{Q}^\pi(\mathbf{x},\mathbf{u})\right] , \tag{3.112}$$

[6]Also referred to as Monte Carlo return.

where $N(\mathbf{x},\mathbf{u})$ is the total number of times the state-input pair is visited, we get from (3.110) the Monte Carlo update rule

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow \hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \frac{1}{N(\mathbf{x},\mathbf{u})}\sum_{k=1}^{N(\mathbf{x},\mathbf{u})}\Big[\hat{R}_k - \hat{Q}^\pi(\mathbf{x},\mathbf{u})\Big] . \tag{3.113}$$

An update law for the visitation count $N(\mathbf{x},\mathbf{u})$ is mathematically formulated as

$$N(\mathbf{x},\mathbf{u}) \leftarrow N(\mathbf{x},\mathbf{u}) + \mathbb{1}_{\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}} .$$

Here, $\mathbb{1}_{\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}}$ is the binary indicator function[7]. The equation (3.113) represents a simple average of the returns. After each episode, for each state-input pair visited, we calculate the return $\hat{R}_k$ from that point until the end of the episode and then update the Q-value estimate for that state-action pair accordingly. This particular MC method is called every-visit MC because we compute the return every time a state is visited in the episode. MC methods need to learn from complete episodes to compute and all the episodes must eventually terminate.

SARSA: On-policy Temporal-Difference Learning
On-policy Temporal-Difference (TD) learning combines the principles of Monte Carlo methods with Dynamic Programming (DP). Under the certainty equivalence assumption, we get from the Bellman Expectation Equation, cf. (3.50),

$$Q^\pi(\mathbf{x},\mathbf{u}) = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma Q^\pi(\mathbf{x}',\mathbf{u}') . \tag{3.114}$$

The key idea in On-policy TD learning is to update the action-value function $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ using the TD residual

$$\delta = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma\hat{Q}^\pi(\mathbf{x}',\mathbf{u}') - \hat{Q}^\pi(\mathbf{x},\mathbf{u}) \tag{3.115}$$

and to follow the SARSA update rule, see, e.g., [3.6, p. 120],

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow \hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \alpha\Big[\bar{r}(\mathbf{x},\mathbf{u}) + \gamma\hat{Q}^\pi(\mathbf{x}',\mathbf{u}') - \hat{Q}^\pi(\mathbf{x},\mathbf{u})\Big] . \tag{3.116}$$

For $\alpha = 0$, the value $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ remains unchanged, while for $\alpha = 1$, the old value $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ is replaced entirely by the so-called TD-target[8] $\bar{r}(\mathbf{x},\mathbf{u})+\gamma\hat{Q}^\pi(\mathbf{x}',\mathbf{u}')$. This becomes evident when we express (3.116) as a first-order low-pass filter in the form

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow (1-\alpha)\hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \alpha\Big[\bar{r}(\mathbf{x},\mathbf{u}) + \gamma\hat{Q}^\pi(\mathbf{x}',\mathbf{u}')\Big] . \tag{3.117}$$

On-policy Temporal-Difference learning is called SARSA because of this data sequence used for calculation – State, Action, Reward, State, Action. The estimate $\hat{Q}^\pi$ is updated based on its own estimation, which is a form of bootstrapping, as described in [3.6, p. 89]. TD learning can learn from incomplete rollouts and hence we don't need to track the

[7]If $\mathbf{x}_k = \mathbf{x}$ and $\mathbf{u}_k = \mathbf{u}$, the function $\mathbb{1}_{\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}}$ returns 1, otherwise, it returns 0.
[8]In machine learning, a target value refers to the actual value you aim to predict with your model. It's the outcome or label that the algorithm is being trained to forecast in supervised learning.

rollouts up to termination.

Q-learning: Off-policy Temporal-Difference Learning
Q-learning is an extension of TD-learning that goes beyond predicting the value function Q^π to enable the determination of an optimal policy π^*. The update rule for Q-learning involves updating the estimate $\hat{Q}^\pi$ at each time step, considering both the observed state $\mathbf{x}$ and the corresponding action $\mathbf{u}$. Note that the Bellman Optimality Equations (3.53) reads as

$$Q^*(\mathbf{x},\mathbf{u}) = \bar{r}_k(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)}\left\{\max_{\mathbf{w}\in\mathcal{U}} Q^*(\mathbf{x}',\mathbf{w})\right\} . \tag{3.118}$$

Since we do not have access to the optimal values Q^*, the idea in Off-policy TD learning is to update the estimate of the action-value function $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ using the residual

$$\delta = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^\pi(\mathbf{x}',\mathbf{w}) - \hat{Q}^\pi(\mathbf{x},\mathbf{u}) . \tag{3.119}$$

The Q-learning update law than becomes

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow \hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \alpha\left[\bar{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^\pi(\mathbf{x}',\mathbf{w}) - \hat{Q}^\pi(\mathbf{x},\mathbf{u})\right] . \tag{3.120}$$

The estimated action-value function $\hat{Q}^\pi$ directly approximates optimal action-value function Q^*, independent of the policy being followed. There are a couple of extensions of TD- and Q-learning that are designed to address its limitations and improve its applicability and performance. Key extensions include:

- Double Q-Learning [3.6, p. 125]: Addresses the overestimation of action-state values by decoupling the improvement and evaluation of the input in the action-state value update.
- Eligibility Traces [3.6, p. 287]: The more theoretical view is that Eligibility Traces are a bridge from TD to Monte Carlo methods (forward view). According to the other view, an Eligibility Trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action (backward view). The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. TD(λ) [3.6, p. 293]: Extends the basic TD method by considering a series of TD errors for all time steps until the end of the episode, weighted by a factor λ. The factor $\lambda \in [0,1]$ allows you to balance the trade-off between bias (accuracy) and variance (stability) in the learning updates.
- $Q(\lambda)$ [3.6, p. 312]: This extension applies the concept of Eligibility Traces to Q-Learning. In $Q(\lambda)$, every input-value pair (Q-value) has an associated eligibility trace, which decays over time but gets bumped up when visited. The action-state value updates are then applied to all pairs proportionally to their eligibility, allowing for quicker propagation of information through the state-input space.

Finally, we compare Monte Carlo, SARSA and Q-Learning and draw some conclusions:

Aspect	**Monte Carlo**	**SARSA**	**Q-Learning**
Type of Learning	Episodic	On-policy TD	Off-policy TD
Convergence Speed	Slow (high variance)	Moderate	Fast (high bias)
Policy Dependency	Optimal policy	Sensitive to current policy	Learns optimal policy from any policy
Suitability	Episodic tasks	Close to current strategy tasks	Complex environments with suboptimal actions

Table 3.5: Comparison of Monte Carlo, SARSA, and Q-Learning.

Note 3.8. Python code of the *Monte Carlo Algorithm*, the *SARSA Algorithm* and *Q-Learning Algorithm* for the Frozen Lake Example can be found here.

3.4.2 Approximate solution methods

For large state and input spaces, methods where the state value function V^π or the action-state value function Q^π are represented by means of a table are no longer manageable. The reason for this is the high memory requirements of the table, since a values must be stored for each state or state-input pair. Moreover, for a continuous state space, an infinite number of values would have to be stored, and the computational cost is considerable has to be applied to each state or state-input pair, respectively. Instead of a table, a function approximator parameterized with the vector $\boldsymbol{\phi}$ can be used to represent the value functions in the form

$$V^\pi(\mathbf{x}) \approx \hat{V}^\pi(\mathbf{x};\boldsymbol{\phi}) = \hat{V}^\pi_{\boldsymbol{\phi}}(\mathbf{x}) \tag{3.121a}$$

$$Q^\pi(\mathbf{x},\mathbf{u}) \approx \hat{Q}^\pi(\mathbf{x},\mathbf{u};\boldsymbol{\phi}) = \hat{Q}^\pi_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{u}) \ . \tag{3.121b}$$

As a result, the memory requirement is significantly reduced to only the parameter vector $\boldsymbol{\phi}$. Additionally, by employing a function approximator, the agent can generalize from previously visited states or state-input pairs to unvisited ones. In an ideal scenario where $V^\pi(\mathbf{x})$ and $Q^\pi(\mathbf{x},\mathbf{u})$ are known, one could utilize supervised learning methods to approximate them with $\hat{V}^\pi$ and $\hat{Q}^\pi$. By generating rollouts

$$\boldsymbol{\tau}^{(e)} = \left(\mathbf{x}_0^{(e)}, \mathbf{u}_0^{(e)}, \mathbf{x}_1^{(e)}, \mathbf{u}_1^{(e)}, \dots, \mathbf{x}_{N-1}^{(e)}, \mathbf{u}_{N-1}^{(e)}, \mathbf{x}_N^{(e)}\right) \tag{3.122}$$

for $e = 0, 1, \dots, E$, with data

$$\mathcal{D} = \left\{\mathbf{x}_k^{(i)}, V^\pi\left(\mathbf{x}_k^{(e)}\right)\right\}_{e=0}^{E} \tag{3.123a}$$

$$\mathcal{D} = \left\{\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right), Q^\pi\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right)\right\}_{e=0}^{E} \ , \tag{3.123b}$$

the *regression task* can be formulated as follows

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{e\in\mathcal{D}} \left(\hat{V}^\pi_{\boldsymbol{\phi}}\left(\mathbf{x}_k^{(e)}\right) - V^\pi\left(\mathbf{x}_k^{(e)}\right)\right)^2 \tag{3.124}$$

and

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{e\in\mathcal{D}} \left(\hat{Q}^\pi_{\boldsymbol{\phi}}\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right) - Q^\pi\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right)\right)^2 \ . \tag{3.125}$$

3.4.3 Deep Q-Network

Q-learning can experience instability and divergence when combined with nonlinear function approximators and bootstrapping. The Deep Q-Network (DQN) addresses these issues by introducing two key innovations to stabilize and enhance the training process:

- *Experience replay*: Instead of using sequential rollouts for updates, DQN stores a collection of steps $\mathbf{e}_k = (\mathbf{x}_k, \mathbf{u}_k, \mathrm{r}_k, \mathbf{x}_{k+1})$ in a replay memory $\mathcal{D}_k = \{\mathbf{e}_0, \dots, \mathbf{e}_k\}$. This memory contains a diverse range of experiences from multiple past episodes. During training, mini-batches of experiences are randomly sampled from this memory to update the Q-values. This technique not only improves the efficiency of data utilization but also breaks the correlation between consecutive samples and mitigates the non-stationary distribution issues, leading to more stable and reliable learning.

- *Periodically updated the target network*: In standard Q-learning, the same network estimates both the current and the target Q-values, leading to a moving target problem that can cause harmful correlations and oscillations. DQN addresses this by employing two separate networks: a primary network with parameters $\boldsymbol{\phi}$ for the current Q-value estimation and a target network with parameters $\boldsymbol{\phi}^-$ that remains fixed for a set number of steps C. The target network's sole purpose is to generate the stable target Q-values for the updates. Every C steps, the primary network's weights are copied to the target network, ensuring that the targets are only periodically updated, which significantly enhances the stability of the learning process.

The objective function for DQN is formulated as follows:

$$L(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{x},\mathbf{u},\mathsf{r},\mathbf{x}')\sim U(\mathcal{D})}\left[\left(\mathsf{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^{\pi}_{\boldsymbol{\phi}^-}(\mathbf{x}',\mathbf{w}) - \hat{Q}^{\pi}_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{u})\right)^2\right] \tag{3.126}$$

Here, $U(\mathcal{D})$ represents a uniform distribution over the replay memory $\mathcal{D}$ and $\boldsymbol{\phi}^-$ denotes the parameters of the target network.

3.4.4 Policy Gradients

Policy gradients are a class of algorithms in reinforcement learning that optimize policies directly. They work by calculating the gradient of the expected reward concerning the policy parameters and then adjusting the parameters in the direction that increases the expected reward. This approach is particularly powerful for high-dimensional or continuous input spaces and allows for stochastic policies, offering a more nuanced way of exploring and exploiting the environment. They form the basis for many advanced reinforcement learning methods and are fundamental to understanding and developing new algorithms in the field.

The term *Policy Gradient* (PG) covers methods where a policy is parameterized by the parameter vector $\boldsymbol{\theta}$, i.e.

$$\pi(\mathbf{u}_k \mid \mathbf{x}_k; \boldsymbol{\theta}) = \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.127}$$

With this parameterized policy, a multivariate distribution

$$\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0) \prod_{k=0}^{N-1} \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \tag{3.128}$$

can be introduced, which gives the probability density of observing a N-step rollout $\boldsymbol{\tau}$ under a the policy $\pi_{\boldsymbol{\theta}}$. Thus, with the *discounted return* (3.8), we obtain the parameterized action-value function

$$Q_k^{\pi_\theta}(\mathbf{x}_k, \mathbf{u}_k) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\mathsf{R}_k(\boldsymbol{\tau}) \mid \mathbf{x}_k = \mathbf{x}_k, \mathbf{u}_k = \mathbf{u}_k\} \ , \tag{3.129}$$

and analogously the parameterized state-value function

$$V_k^{\pi_\theta}(\mathbf{x}_k) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\mathsf{R}_k(\boldsymbol{\tau}) \mid \mathbf{x}_k = \mathbf{x}_k\} \ . \tag{3.130}$$

Now, the optimal control problem can be formulated as an optimization over the parameter vector $\boldsymbol{\theta}$, see [3.8, p. 96], in the form

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \ , \tag{3.131}$$

with expected return

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)}\{V_0^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_0)\} = \int_{\mathcal{T}} \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\mathsf{R}_0(\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau} \ . \tag{3.132}$$

To solve (3.131) with (3.132), a gradient ascent method can be employed

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}} \ , \tag{3.133}$$

where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ represents the gradient of the objective function $J(\boldsymbol{\theta})$, and α denotes the step size or learning rate. Policy gradient methods optimize a policy directly by following the gradient of the expected reward with respect to policy parameters. The distinction between Deterministic and Stochastic Policy Gradients is in the way inputs are chosen by the policy:

- Deterministic Policy Gradients (DPG): The policy directly maps states to inputs deterministically. In Deep Deterministic Policy Gradient (DDPG), a noise process is added to the output of the actor network to promote exploration during learning. This controlled randomness in the output enables the deterministic policy to explore the action space effectively.
- Stochastic Policy Gradients (SPG): The policy outputs a probability distribution over inputs, meaning the input is sampled from this distribution, introducing randomness. SPG approaches are often advantageous in settings where exploration is essential, as the stochasticity encourages diverse input selection in the learning process.

3.4.5 Deterministic and stochastic policies

Deterministic policy
For a deterministic policy, we have

$$\mathbf{u} = \boldsymbol{\mu}_\theta(\mathbf{x}) \ . \tag{3.134}$$

The policy $\boldsymbol{\mu}_\theta(\mathbf{x})$ is typically represented by neural networks or other parameterized function approximators. These functions map from the state space to the input space and are designed to capture complex, nonlinear relationships between states and inputs.
Multivariate Gaussian distribution
A multivariate Gaussian distribution (or multivariate normal distribution) is described by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. A diagonal Gaussian distribution is a special case where the covariance matrix only has entries on the diagonal. As a result, we can represent it by a vector. Hence, a popular choice for policy model $\pi_{\boldsymbol{\theta}}(\mathbf{u} \mid \mathbf{x})$ is the *diagonal*

Gaussian policy model. A diagonal Gaussian policy always has a neural network that maps from states to mean inputs $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviations are typically represented by standalone parameters. Given the mean input $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$, and a vector $\mathbf{z}$ of noise from a spherical Gaussian ($\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$), an input sample can be computed with

$$\mathbf{u} = \boldsymbol{\mu}_{\theta}(\mathbf{x}) + \boldsymbol{\sigma}_{\theta} \odot \mathbf{z} \ , \tag{3.135}$$

where $\odot$ denotes the element wise product of two vectors. In a diagonal Gaussian policy model, exploration is achieved by the additive noise term. The mean determines the expected input, while the diagonal elements of the covariance matrix determine the amount of exploration or noise added to each input. When an input is sampled from this distribution, the noise leads to exploration by encouraging slightly different inputs to be taken, even in the same state. This process allows the model to explore various strategies and potentially discover better ones. Note that the log-likelihood of a m-dimensional input, for a diagonal Gaussian with mean $\boldsymbol{\mu} = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma} = \boldsymbol{\sigma}_{\boldsymbol{\theta}}$, is given by

$$\ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u} \mid \mathbf{x})\right) = -\frac{1}{2}\left(\sum_{i=1}^{m}\left(\frac{(\mathbf{u}[i] - \boldsymbol{\mu}[i])^2}{\boldsymbol{\sigma}[i]^2} + 2\ln\left(\boldsymbol{\sigma}[i]\right)\right) + m\ln\left(2\pi\right)\right) \ . \tag{3.136}$$

3.4.6 Stochastic Policy Gradient

To obtain the Policy Gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ for the stochastic policy (3.127), the following procedure is performed. From (3.132), follows

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \int_{\mathcal{T}} \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}}\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \ . \tag{3.137}$$

Using the identity[9]

$$\nabla_{\boldsymbol{\theta}}\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \frac{\nabla_{\boldsymbol{\theta}}\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right)}{\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})} = \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \ , \tag{3.138}$$

we obtain

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}}\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau})\right\} \ . \tag{3.139}$$

The term $\ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right)$ can be split into three components using (3.128), i.e.,

$$\ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \ln\left(\mathsf{p}_0(\mathsf{x}_0)\right) + \sum_{k=0}^{N-1} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathsf{x}_k)\right) + \sum_{k=0}^{N-1} \ln\left(\mathsf{p}_{\mathsf{x}}(\mathsf{x}_{k+1} \mid \mathsf{x}_k, \mathbf{u}_k)\right) \ . \tag{3.140}$$

Taking the gradient with respect to the parameter vector eliminates the terms that depend on the system dynamics results in

$$\nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathsf{x}_k)\right) = \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathsf{x}_k)\right) \ . \tag{3.141}$$

[9] Known as the log-derivative-trick.

Substituting (3.141) into (3.139) with (3.129) yields the *Stochastic Policy Gradient* (SPG) for finite time horizons[10], see [3.14],

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\mathsf{R}_0\right\} . \tag{3.142}$$

Note 3.9. With a parameterized deterministic policy $\pi_{\boldsymbol{\theta}} : \mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$, the multivariate distribution (3.155) simplifies to

$$\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\Big|_{\mathbf{u}_k=\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.143}$$

Taking the logarithm yields

$$\ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \ln\left(\mathsf{p}_0(\mathbf{x}_0)\right) + \sum_{k=0}^{N-1} \ln\left(\mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\right)\Big|_{\mathbf{u}_k=\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.144}$$

The gradient with respect to $\boldsymbol{\theta}$ is computes as

$$\begin{aligned} \nabla_\theta \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) &= \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \ln(\mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k))\Big|_{\boldsymbol{\theta}(\mathbf{x}_k)} , \\ &= \nabla_{\boldsymbol{\theta}}\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)\nabla_{\mathbf{u}_k} \ln\left(\mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\right)\big|_{\mathbf{u}_k=\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} , \end{aligned} \tag{3.145}$$

where the term containing the system dynamics does not cancel out anymore, unlike in (3.141). Thus, the system dynamics must then be known to calculate the policy gradient.

There are two important variants of the Stochastic Policy Gradient:

- *Action-Value Function Policy Gradient*: Incorporates all return information, making it theoretically complete and accurate.
- *Advantage Function Policy Gradient*: Reduces variance in gradient estimates, leading to more stable and efficient learning.

Action-Value Function Policy Gradient

Firstly, the Stochastic Policy Gradient can be formulated in term of the return R_k according to (3.8). Causality requires that future inputs and policies at time k do not depend on past rewards at time i. Thus, for $i < k$, we have

$$0 = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\mathbf{r}_i\} . \tag{3.146}$$

[10] $\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}$ is the so-called score function.

With the definition of the return R_k, we get from (3.142)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\left(\sum_{i=0}^{k-1}\gamma^{i-k} r_i + \underbrace{\sum_{i=k}^{N-1}\gamma^{i-k} r_i}_{=\mathsf{R}_k}\right)\right\} \tag{3.147}$$

and with the causality condition (3.146), we find another formulation of the Stochastic Policy Gradient

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\mathsf{R}_k\right\} . \tag{3.148}$$

Secondly, the Stochastic Policy Gradient can be written in term of the state-value function $Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)$. Define $\boldsymbol{\tau}_{[:,k]} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_k, \mathbf{u}_k)$ as the rollout up to time k, and $\boldsymbol{\tau}_{[k,:]}$ as the remainder of the rollout after that. Using the *law of total expectation* (A.9), we can break up (3.142) into:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{k=0}^{N-1} \mathbb{E}_{\boldsymbol{\tau}_{[:,k]}\sim\pi_{\boldsymbol{\theta}}}\Big\{\mathbb{E}_{\boldsymbol{\tau}_{[k,:]}\sim\pi_{\boldsymbol{\theta}}}\Big\{\nabla_\theta \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\mathsf{R}_k \mid \boldsymbol{\tau}_{[:,k]}\Big\}\Big\} . \tag{3.149}$$

The gradient of the log-probability is constant with respect to the inner expectation, due to its dependency on $\mathbf{x}_k$ and $\mathbf{u}_k$, and it can be extracted:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{k=0}^{N-1} \mathbb{E}_{\boldsymbol{\tau}_{[:,k]}\sim\pi_{\boldsymbol{\theta}}}\Big\{\nabla_\theta \ln\left(\pi_\theta(\mathbf{u}_k \mid \mathbf{x}_k)\right)\mathbb{E}_{\boldsymbol{\tau}_{[k,:]}\sim\pi_{\boldsymbol{\theta}}}\Big\{\mathsf{R}_k \mid \boldsymbol{\tau}_{[:,k]}\Big\}\Big\} . \tag{3.150}$$

Lastly, the Markov property implies

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)\right\} . \tag{3.151}$$

Since $\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})$ is unknown, the expectation $\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\cdot\}$ is approximated from $e = 1, \dots E$ rollouts $\boldsymbol{\tau}^{(e)}$, collected in the set $\mathcal{D} = \{\boldsymbol{\tau}^{(e)}\}$, by the *empirical mean* as

$$\hat{\mathbf{g}} = \widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{\tau}\in\mathcal{D}}\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{x}_k \mid \mathbf{u}_k)\right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) . \tag{3.152}$$

Advantage Function Policy Gradient

The estimated policy gradient has high variance[11], resulting in poor convergence properties [3.15]. To reduce variance, a *baseline* $b_k(\mathbf{x}_k) \in \mathbb{R}$ can be subtracted from Q_k^π in (3.142) without affecting the expectation, see [3.8, p. 98] for a proof, resulting in

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}}\big(\ln \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\big)\Big(Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) - b_k(\mathbf{x}_k)\Big)\right\} . \tag{3.153}$$

[11]We say that a method has high variance if you get different results when you run it multiple times. A method with less variance will give you more similar results when you run it multiple times.

Intuitively, the variance is reduced when subtracting a baseline because the operation effectively re-centers the reward signal around a mean value. A widely used baseline is the value function $b_k(\mathbf{x}_k) = V_k^\pi(\mathbf{x}_k)$. See [3.16] for different baselines and variants of the Stochastic Policy Gradient. From (3.153), we then get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) \right\} , \tag{3.154}$$

with *Advantage function* according to (3.41). The Advantage function quantifies the relative benefit of taking a specific input in a given state compared to the average input for that state.

- By comparing $Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)$ (expected reward for a specific input) with the basline $V_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k)$ (average reward under the policy), the advantage function isolates how much better or worse an input is relative to this baseline.
- Inputs with a positive advantage $A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) > 0$ are better than the policy's average input, and the policy is encouraged to select these inputs more frequently.
- Inputs with a negative advantage $A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) < 0$ are worse than the average, and the policy is discouraged from taking them.

3.4.7 Deterministic Policy Gradient

The policy gradient (PG) for deterministic policy of the form $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$ is developed in [3.17]. The authors demonstrated that the Deterministic Policy Gradient (DPG) acts as a special case of the Stochastic Policy Gradient (SPG). For a parameterized deterministic policy $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$, we introduce the multivariate distribution

$$\mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.155}$$

The cost function to be minimized

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)}\{V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0)\} = \int_{\mathcal{T}} \mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \tag{3.156}$$

is differentiated with respect to $\boldsymbol{\theta}$, to obtain

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)}\{\nabla_{\boldsymbol{\theta}} V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0)\} . \tag{3.157}$$

To determine the DPG, $\nabla_{\boldsymbol{\theta}} V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0)$ must be known. The calculation is then carried out for the general case $\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k)$, following the proof in [3.17]:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) &= \nabla_{\boldsymbol{\theta}} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \\
&\overset{(3.50)}{=} \nabla_{\boldsymbol{\theta}} \Big(\bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} | \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \Big\{ V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \Big\} \Big) \\
&= \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \nabla_{\boldsymbol{\theta}} \int_{\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1} ,
\end{aligned} \tag{3.158}$$

The last part can be expanded taking the gradient inside the integral

$$\begin{aligned}&\gamma \int_{\mathcal{X}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \\ &+ \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1} \;.\end{aligned} \tag{3.159}$$

Putting things together gives

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} &\left(\bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \underbrace{\int \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1}}_{= \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))}\left\{ V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\}} \right) \\ &+ \gamma \underbrace{\int \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1}}_{= \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))}\left\{ \nabla_{\theta} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\}} \;.\end{aligned} \tag{3.160}$$

And finally we get

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) &= \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)\big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} \\ &\quad + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \left\{ \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\} \;.\end{aligned} \tag{3.161}$$

Continuing the recursion in (3.161), substituting this into (3.157), and combining the expectation over $\boldsymbol{\tau}$, one obtains an expression for the DPG:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) \Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} \right\}. \tag{3.162}$$

3.4.8 Actor-Critic Methods

Two main components in policy gradient are the policy model and the value function. It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the policy update, such as by reducing gradient variance in policy gradients, and that is exactly what the Actor-Critic method does. Actor-critic methods consist of two models:

- The *Critic* updates the value function parameters $\boldsymbol{\phi}$ and depending on the algorithm it could be action-value function $Q_{\boldsymbol{\phi}}^{\pi}$ or state-value function $V_{\boldsymbol{\phi}}^{\pi}$.
- The *Actor* updates the policy parameters $\boldsymbol{\theta}$ for $\pi_{\boldsymbol{\theta}}$, in the direction suggested by the critic.

Algorithm 4 shows a vanilla Actor-Critic Stochastic Policy Gradient Algorithm and Algorithm shows a Vanilla Actor-Critic Deterministic Policy Gradient Algorithm.

Algorithm 4: Vanilla Actor-Critic Stochastic Policy Gradient Algorithm

```
   /* Initialization                                                      */
1  Initialize policy parameters $\boldsymbol{\theta}$ arbitrarily;
2  Initialize value function parameters $\boldsymbol{\phi}$ arbitrarily;
3  for $e = 0, 1, 2, \dots, E$ episodes do
       /* Reset the environment and initial state                         */
4      Initialize state $\mathbf{x}_0$;
       /* Collect rollouts using policy $\pi^{(e)} = \pi\left(\boldsymbol{\theta}^{(e)}\right)$ in environment   */
5      $\mathcal{D}^{(e)} = \boldsymbol{\tau}^{(1)}, \dots, \boldsymbol{\tau}^{(E)}$;
6      for $k = 0, 1, 2, \dots, N-1$ time steps do
           /* Compute Monte Carlo return                                  */
7          $\hat{R}_k^{(e)} \leftarrow \sum_{j=k}^{N-1} \gamma^{j-k} r_j$;
           /* Compute Advantage estimates                                 */
8          $\hat{A}_k^{(e)} \leftarrow \hat{R}_k^{(e)} - \hat{V}^{\pi}_{\boldsymbol{\phi}^{(e)}}$;
9      end
       /* Estimate policy gradient                                        */
10     $\hat{\mathbf{g}}^{(e)} \leftarrow \frac{1}{\left|\mathcal{D}^{(e)}\right|} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(e)}} \hat{A}_k^{(e)}$;
       /* Actor learning - compute policy update                          */
11     $\boldsymbol{\theta}^{(e+1)} \leftarrow \boldsymbol{\theta}^{(e)} + \alpha \hat{\mathbf{g}}^{(e)}$;
       /* Critic learning - fit the value function by regression          */
12     $\boldsymbol{\phi}^{(e+1)} \leftarrow \arg\min_{\boldsymbol{\phi}} \frac{1}{\left|\mathcal{D}^{(e)}\right| N} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \left(\hat{V}^{\pi}_{\boldsymbol{\phi}} - \hat{R}_k^{(e)}\right)^2$;
13 end
```

3.4.9 Off-Policy Policy Gradient

The vanilla versions of the actor-critic method are exclusively on-policy: training samples are collected according to the *target policy*, which is the same policy we aim to optimize. A simpler strategy involves two policies: the *target policy* π, which is studied and optimized, and the *behavior policy* β, which drives exploratory actions. Here, learning is based on data "off" the target policy, and the entire mechanism is labeled as off-policy learning. Off-policy methods offer several distinct advantages, including:

- The off-policy approach doesn't require a complete rollout, and it can reuse past episodes (via experience replay), resulting in significantly improved sample efficiency[12].
- The sample collection employs a behavior policy different from the target policy, facilitating more effective exploration.

[12]Sample efficiency in the context of control technology and machine learning refers to the ability of a system to achieve high performance with a limited number of data samples.

Nearly all off-policy methods employ *importance sampling* to estimate expected values from a distribution different than the sample source. In the context of off-policy learning, returns are weighted by the importance sampling ratio, which quantifies the relative likelihood of specific trajectories under the target and behavior policies. Given an initial state $\mathbf{x}_k$, the probability of a future state-input trajectory under any policy π is given by (3.33). Thus, the relative probability of the trajectory under the target and behavior policies, that it the *importance sampling ratio*, is

$$\rho_{[k:N-1]} \doteq \frac{\prod_{j=k}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{j+1} \mid \mathbf{x}_j, \mathbf{u}_j)\pi(\mathbf{u}_j \mid \mathbf{x}_j)}{\prod_{j=k}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{j+1} \mid \mathbf{x}_j, \mathbf{u}_j)\beta(\mathbf{u}_j \mid \mathbf{x}_j)} = \prod_{j=k}^{N-1} \frac{\pi(\mathbf{u}_j \mid \mathbf{x}_j)}{\beta(\mathbf{u}_j \mid \mathbf{x}_j)} \; . \tag{3.163}$$

Although the trajectory probabilities depend on the MDP's transition probabilities, which are generally unknown, they appear identically in both the numerator and denominator, and thus cancel. The importance sampling ratio ends up depending only on the two policies and the sequence, not on the MDP.

Remember that we aim to estimate the expected returns under the target policy π, but only possess returns R_k^β from the behavior policy β. These returns exhibit an incorrect expectation denoted by $V^\beta(\mathbf{x}) = \mathbb{E}_\beta\left\{\mathsf{R}_k^\beta \mid \mathsf{x}_k = \mathbf{x}\right\}$. To correct this, importance sampling is employed. Using the ratio $\rho_{[k:N-1]}$, we can transform these returns to align with the desired expectation

$$V^\pi(\mathbf{x}) = \mathbb{E}_\pi\left\{\rho_{[k:N-1]}\mathsf{R}_k^\beta \mid \mathsf{x}_k = \mathbf{x}\right\} \; . \tag{3.164}$$

Now let's see how off-policy policy gradient is computed. Since the training observations are sampled by $\mathsf{u}_k \sim \beta(\mathbf{u}_k \mid \mathbf{x}_k)$, we can rewrite the gradient following certain calculations, see [3.18] for a proof,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\beta\left\{\sum_{k=0}^{N-1} \frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)}{\beta(\mathbf{u}_k \mid \mathbf{x}_k)} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathsf{x}_k, \mathsf{u}_k)\right\} \; . \tag{3.165}$$

Here, $\frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\beta(\mathbf{u}_k|\mathbf{x}_k)}$ is the *importance weight.* In essence, when implementing policy gradient in the off-policy setting, we can adjust it with a weighted sum, where the weight is the ratio of the target policy to the behavior policy.

3.4.10 Proximal Policy Optimization

Schulman proposed *Proximal Policy Optimization* (PPO) in 2017, and it has since become a widely used method in continuous model-free RL due to its simplicity and strong performance. PPO-clip is an actor-critic method and makes use of a generalized advantage estimate and a clipped surrogate objective function, see [3.19]. It's worth noting that there are other variants of PPO, such as PPO-KL, which incorporate an adaptive Kullback-Leibler penalty term, see [3.16]. Our discussion will focus exclusively on PPO-Clip. This approach is straightforward to implement and has been shown to work well in practice. It doesn't require a second-order optimization process, making it computationally less intensive.

Generalized Advantage Estimation

PPO employs a Generalized Advantage Estimation (GAE) as its advantage function. The purpose of GAE is to significantly reduce the variance of the estimator while keeping the bias introduced as low as possible [3.20, p. 136]. This goal mirrors the fundamental principles of Eligibility Traces, cf. [3.6, p. 125]. Our present discussion is fundamentally grounded in the methodologies outlined in the supplementary material provided by [3.21]. Generally, the Monte Carlo return (3.111) serves as an unbiased estimator of the expected return at a specific state. However, each reward r_k may vary due to the environment dynamics, leading to a high variance in the overall estimation. To mitigate this, an employe an n-step return

$$\begin{aligned}\hat{R}_{[k:k+n]} &= r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \ldots + \gamma^n r_{k+n} \\ &= \sum_{j=k}^{n-1} \gamma^{j-k} r_j + \gamma^n \hat{V}^{\pi}_{k+n}(\mathbf{x}_{k+n}) \ ,\end{aligned} \tag{3.166}$$

where we estimate the remaining return using a value function estimate $\hat{V}^{\pi}(\mathbf{x})$. It offers a lower-variance but slightly biased estimate by truncating the sum of returns after n steps. Particular instances such as $\hat{R}_{[k:k+1]} = r_k + \gamma \hat{V}^{\pi}_{k+1}(\mathbf{x}_{k+1})$, used in Q-learning, and $\hat{R}_{[k,\infty]} = \sum_{j=k}^{N} \gamma^{j-k} r_j = \hat{R}_k$, which reverts to the original Monte Carlo return, illustrate the variance-bias trade-off. An alternative for balancing bias and variance is the λ-return, computed as a weighted average of n-step returns, see [3.6, p. 289], and defined as

$$\hat{R}_k(\lambda) = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{R}_{[k:k+n]} \ . \tag{3.167}$$

Assuming all rewards after step N are zero, such that $\hat{R}_{[k:k+n]} = \hat{R}_{[k:N]}$ for all $n \geq N-k$, the infinite sum can be calculated according to

$$\hat{R}_k(\lambda) = (1-\lambda) \sum_{n=1}^{N-k-1} \lambda^{n-1} \hat{R}_{[k:k+n]} + \lambda^{N-k-1} \hat{R}_{[k:N]} \ . \tag{3.168}$$

Here, $\lambda = 0$ yields $\hat{R}_{[k,k+1]}$, and $\lambda = 1$ provides the full Monte Carlo return $\hat{R}_{[k,\infty]}$. Intermediate values of $\lambda \in (0,1)$ produces interpolants that can be used to balance the bias and variance of the value estimator. Updating the value function with the λ-return leads to the so-called TD(λ) algorithm, and its application for advantage estimation results in the Generalized Advantage Estimator GAE(λ):

$$\hat{A}^{\pi}_k(\lambda) = \hat{R}_k(\lambda) - \hat{V}^{\pi}_k(\mathbf{x}_k) \ . \tag{3.169}$$

For instance, setting $\lambda = 0$ yields the expression

$$\hat{A}^{\pi}_k(0) = \hat{R}_{[k:k+1]} - \hat{V}^{\pi}_k(\mathbf{x}_k) = r_k + \gamma \hat{V}^{\pi}_{k+1}(\mathbf{x}_{k+1}) - \hat{V}^{\pi}_k(\mathbf{x}_k) = \delta_k \ , \tag{3.170}$$

which is equivalent to the TD residual (3.115). Notice that $\hat{R}_k(0) = \hat{R}_{[k:k+1]} = r_k + \gamma \hat{V}^{\pi}_{k+1}(\mathbf{x}_{k+1})$ is the 1-step estimator for $\hat{Q}^{\pi}_k(\mathbf{x}_k, \mathbf{u}_k)$. Choosing a value closer to zero introduces more bias due to the immediate approximation, resulting in lower variance, as discussed in [3.16]. Conversely, employing $\lambda = 1$ leads to high variance and nearly zero bias due to the summation of rewards, i.e.,

$$\hat{A}^{\pi}_{k}(1) = \hat{R}_{[k:N]} - \hat{V}^{\pi}_{k}(\mathbf{x}_k) = \sum_{j=k}^{N} \gamma^{j-k} r_j \ . \tag{3.171}$$

which is equivalent to the Monte Carlo return (3.111). To summarize, adjusting the value of λ allows control over the tradeoff between bias and variance. A smaller value, such as $\lambda = 0$, reduces variance at the expense of introducing more bias, while a larger value, like $\lambda = 1$, results in higher variance with minimal bias due to reward summation.

Clipping

PPO-clip updates policies via

$$\theta^{(i+1)} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{u}_k \sim \pi_{\boldsymbol{\theta}^{(i)}}} \left\{ L(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}) \right\} \ , \tag{3.172}$$

typically taking multiple steps of (usually mini-batch) Stochastic Gradient Ascent (SGA) to maximize the objective function. The (clipped surrogate) objective function L is given by, see [3.19],

$$L\Big(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}\Big) = \min\left(l_k A^{\pi_{\boldsymbol{\theta}^{(i)}}}(\mathbf{x}_k, \mathbf{u}_k),\ \mathrm{clip}(l_k, \epsilon) A^{\pi_{\boldsymbol{\theta}^{(i)}}}(\mathbf{x}_k, \mathbf{u}_k)\right) \ , \tag{3.173}$$

with likelihood ratio

$$l_k = l_k\Big(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}\Big) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)}{\pi_{\boldsymbol{\theta}^{(i)}}(\mathbf{x}_k, \mathbf{u}_k)} = \frac{\text{new policy}}{\text{old policy}} \tag{3.174}$$

and clipping operator

$$\mathrm{clip}(l_k, \epsilon) = \begin{cases} 1-\epsilon & \text{if} \quad l_k < 1-\epsilon \\ 1+\epsilon & \text{if} \quad l_k > 1+\epsilon \\ l_k & \text{else} \ . \end{cases} \tag{3.175}$$

The hyperparameter ϵ is a small positive constant, typically set to a value like 0.2, see Figure 3.10 for an illustration. The purpose of this ϵ-clipping is to prevent overly large policy updates, thereby maintaining stability in the learning process. This is achieved by ensuring that the current policy does not deviate excessively from the older one. The term l_k represents the probability ratio between the current and old policy. Let's interpret the implications of l_k:

- If $l_k > 1$, it signifies that for a given state $\mathbf{x}_k$, the corresponding action $\mathbf{u}_k$ is more probable under the current policy than it was under the old policy.
- Conversely, if l_k is between 0 and 1, the action $\mathbf{u}_k$ is less likely under the current policy compared to the old one.

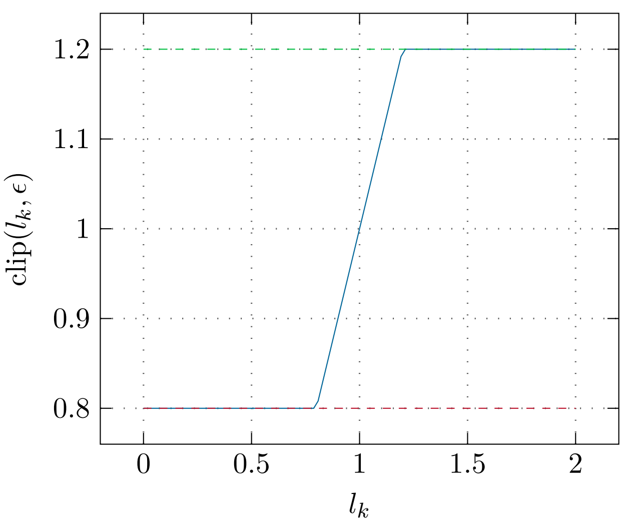


Figure 3.10: Clipping function (3.175) with $\epsilon = 0.2$.

This likelihood ratio serves as a straightforward method to gauge the divergence between the old and current policy. Algorithm 5 shows vanilla Proximal Policy Optimization with Clipping.

Algorithm 5: Proximal Policy Optimization with Clipping

```
   /* Initialization                                                        */
1  Initialize policy parameters $\boldsymbol{\theta}$, old policy parameters $\boldsymbol{\theta}_{\text{old}}$;
2  Initialize value function parameters $\boldsymbol{\phi}$;
3  for $e = 1, 2, 3, \ldots$ episodes do
       /* Collect rollouts using the old policy $\pi^{(e)} = \pi\Big(\boldsymbol{\theta}^{(e)}\Big)$ in
          environment                                                       */
4      $\mathcal{D}^{(e)} = \{\boldsymbol{\tau}^{(1)}, \ldots, \boldsymbol{\tau}^{(E)}\}$;
5      for $k = 0, 1, 2, \ldots, N-1$ time steps do
           /* Compute $\lambda$-returns                                     */
6          $\hat{R}_k^{(e)}(\lambda) \leftarrow (1-\lambda) \sum_{n=1}^{N-k-1} \lambda^{n-1} \hat{R}_{[k:k+n]} + \lambda^{N-k-1} \hat{R}_{[k:N]}$;
           /* Compute generalized advantage estimates                       */
7          $\hat{A}_k^{(e)} \leftarrow \hat{R}_k^{(e)}(\lambda) - \hat{V}_{\boldsymbol{\phi}}^{\pi}(\mathbf{x}_k)$;
8      end
       /* Actor learning - update policy parameters via SGA in
          mini-batch                                                        */
9      $l_k^{(e)} \leftarrow = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)}{\pi_{\boldsymbol{\theta}^{(e)}}(\mathbf{x}_k, \mathbf{u}_k)}$ ;
10     $\boldsymbol{\theta}^{(e+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \frac{1}{\left|\mathcal{D}^{(e)}\right|} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \min\Big(l_k^{(e)} \hat{A}_k^{(e)},\ \text{clip}\Big(\epsilon, l_k^{(e)}\Big) \hat{A}_k^{(e)}\Big)$;
       /* Critic learning - fit the value function by regression           */
11     $\boldsymbol{\phi}^{(e+1)} \leftarrow \arg\min_{\boldsymbol{\phi}} \frac{1}{\left|\mathcal{D}^{(e)}\right| N} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \Big(\hat{V}_{\boldsymbol{\phi}}^{\pi}(\mathbf{x}_k) - \hat{R}_k^{(e)}(\lambda)\Big)^2$;
12 end
```

3.5 Literatur

[3.1] M. P. Deisenroth, A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020. [Online]. Available: https://mml-book.github.io/.

[3.2] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge University Press, 2013, vol. 1.

[3.3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2005, vol. 1.

[3.4] M. L. Puterman, *Markov Decision Processes*. John Wiley & Sons, 2005.

[3.5] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2009.

[3.6] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 2018.

[3.7] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[3.8] M. Sugiyama, *Statistical Reinforcment Learning*. CRC Press, 2015.

[3.9] D. Silver, *Reinforcement learning: An introduction*, 2015. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[3.10] F. L. Lewis, D. L. Varbie, and V. Syrmos, *Optimal Control*. John Wiley & Sons, 2012.

[3.11] M. Vidyasagar, "A tutorial introduction to reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2304.00803v1.

[3.12] R. Stengel, *Optimal Control and Estimation*. Dover Publications, 1986.

[3.13] J. Schulman, "Optimizing expectations: From deep reinforcement learning to stochatic computation graphs," Ph.D. dissertation, University of California, Berkeley, 2016.

[3.14] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, pp. 1238–1274, 2013.

[3.15] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[3.16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *ICLR 2015*, 2015. [Online]. Available: arXiv:1506.02438.

[3.17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmille, "Deterministic policy gradient algorithms," in *Proceedings ofthe 31th International Conference on Machine Learning, ICML 32*, 2014.

[3.18] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *Proceedings of the 29 th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

[3.19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: arXiv:1707.06347.

[3.20] L. Graesser and W. Keng, *Foundation of Deep Reinforcement Learning*. Addison-Wesley, 2020.

[3.21] X. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," 2018. [Online]. Available: arXiv:1804.02717.

[3.22] C. M. Bischof, *Pattern Recognition and Machine Learning*. Springer, 2006.

A Basics of probability theory

A brief summary of basic results from probability theory is given next. For an introduction to probability theory, we refer to [3.1] and [3.2]. We use sans-serif letters such as $\mathsf{x}, \boldsymbol{\mathsf{x}}$, and $\boldsymbol{\mathsf{X}}$ to represent *random variables* (scalars, vectors and matrices) and serif letters such as $x, \mathbf{x}$, and $\mathbf{X}$ to represent the corresponding *deterministic variables* or observations.

A.1 Variables

Quantitative variables are categorized as either discrete or continuous.

- Categorical Variable: Categorical variables contain a finite number of categories or distinct groups. Categorical data might not have a logical order. Categorical data may not follow a logical sequence. Examples include variables like gender, type of material, and methods of payment.
- Discrete Variable: Discrete variables are numeric variables that have a countable number of values between any two values. Discrete variables are exclusively numeric. Common examples include the number of customer complaints or the count of defects or flaws.
- Continuous Variable: Continuous variables are numeric variables that have an infinite number of values between any two values. They can be in the form of numeric data or date/time values. Typical examples are the measurement of a component's length or the exact moment a payment is processed.

A.2 Random variables

A random variable is a mapping from the sample space Ξ to the real numbers $\mathbb{R}$. It assigns a numerical value to each outcome $\xi \in \Xi$. We denote random variables using sans-serif letters, i.e., $\mathsf{x} : \Xi \to \mathbb{R}$, whereas the numerical values are denoted using serif letters, i.e., $x = \mathsf{x}(\xi)$. If the range of the random variable, i.e., the set of possible values of x, is countable, we say that the random variable is discrete. Otherwise, it is continuous.

> *Example* A.1. An example of a discrete random variable is the mapping that assigns integers from 1 to 6 to the outcomes of a die roll. For instance, x(⚀) $= 1$, x(⚁) $= 2$, x(⚂) $= 3$, x(⚃) $= 4$, x(⚄) $= 5$, x(⚅) $= 6$. This process is called categorical variable encoding in machine learning.

Each outcome is mapped to a real number and consequently any event $\mathcal{A} \subseteq \Xi$ corresponds to a subset $\mathcal{A}' \subseteq \mathbb{R}$ of the real line. Conversely, any subset $\mathcal{A}'$ of the real line identifies an event in the sample space. The relationship between $\mathcal{A}$ and $\mathcal{A}'$ can be written as

$$\mathcal{A}' = \{x : x = \mathsf{x}(\xi),\ \xi \in \mathcal{A}\} \quad \text{and} \quad \mathcal{A} = \{\xi : \mathsf{x}(\xi) \in \mathcal{A}'\}\ . \tag{A.1}$$

Based on the correspondence, we denote $\mathcal{A}'$ as an event and assign the probability

$$\mathbb{P}\{\mathsf{x}(\xi) \in \mathcal{A}'\} = \mathbb{P}\{\xi \in \mathcal{A}\}\ . \tag{A.2}$$

A.3 Probability distribution

We write $\mathsf{x} \sim \mathsf{p}(x)$ to denote that a random variable x is distributed according to $\mathsf{p}(x)$, with observation x. We will use the expression *probability distribution* not only for the discrete probability mass function (PMF) but also for the continuous probability density function (PDF). This is in line with most machine learning literature, cf. [3.1, 3.22]. We rely on the context to distinguish the different use of the phrase *probability distribution*.

A.4 Expectation

The expectation or expected value of a deterministic function $g : \mathbb{R} \to \mathbb{R}$ of a univariate random variable $\mathsf{x} \sim \mathsf{p}(x)$ is given by

$$\mathbb{E}_{\mathsf{x}\sim\mathsf{p}(x)}\{g(\mathsf{x})\} = \sum_{x\in\mathcal{X}} \mathsf{p}(x)g(x)\ . \tag{A.3}$$

In the continuous case, we have

$$\mathbb{E}_{\mathsf{x}\sim\mathsf{p}(x)}\{g(\mathsf{x})\} = \int_{\mathcal{X}} \mathsf{p}(x)g(x)\mathrm{d}x\ . \tag{A.4}$$

For multivariate random variables, we define the expected value element wise, see, e.g., [3.1, p. 187] for more information.

A.5 Probability of an event

Next we show how to rewrite the probability of arbitrary events as an expectation. This involves the introduction of the indicator function of a set $\mathcal{A}' \subseteq \mathbb{R}$, defined as a function taking values 0 or 1:

$$\mathrm{I}_{\mathcal{A}'}(x) = \begin{cases} 1, & x \in \mathcal{A}', \\ 0, & \text{else}\ . \end{cases} \tag{A.5}$$

Setting $g(\mathsf{x}) = \mathrm{I}_{\mathcal{A}'}(\mathsf{x})$, results in

$$\mathbb{E}_{\mathsf{x}\sim\mathsf{p}(x)}\{\mathrm{I}_{\mathcal{A}'}(\mathsf{x})\} = \mathbb{P}\{\mathsf{x} \in \mathcal{A}'\} = \begin{cases} \sum_{x\in\mathcal{A}'} \mathsf{p}(x) & \text{if} \quad \mathsf{x} \quad \text{is discrete} \\ \int_{\mathcal{A}'} \mathsf{p}(x)\mathrm{d}x & \text{if} \quad \mathsf{x} \quad \text{is continuous}\ . \end{cases} \tag{A.6}$$

Since the indicator function $\mathrm{I}_{\mathcal{A}'}(x)$ denotes whether x belongs to $\mathcal{A}'$ or not, its expectation measures the average with which x falls inside $\mathcal{A}'$, which coincides with the probability of x being in $\mathcal{A}'$.

A.6 Conditional expectation

We can use conditional distributions to compute conditional expectations. More specifically, the conditional expectation of a random variable $\mathsf{z} = g(\mathsf{x})$ given an observation $\mathsf{y} = y$ is defined as

$$\mathbb{E}_{\mathsf{x}\sim\mathsf{p}(x)}\{g(\mathsf{x}) \mid \mathsf{y} = y\} = \begin{cases} \sum_{x\in\mathcal{X}} g(x)\mathsf{p}(x \mid y) & \text{if} \quad \mathsf{x} \quad \text{is discrete} \\ \int_{\mathcal{X}} g(x)\mathsf{p}(x \mid y)\mathrm{d}x & \text{if} \quad \mathsf{x} \quad \text{is continuous .} \end{cases} \tag{A.7}$$

Example A.2. For example, if $\mathcal{X}$ and $\mathcal{U}$ are continuous, the rollout $\boldsymbol{\tau} \in \mathcal{T}$ is continuous as well. For a given policy π and function $\mathsf{R} : \mathbb{R} \to \mathbb{R}$, then

$$\mathbb{E}_{\boldsymbol{\tau}\sim\mathsf{p}(\cdot)}\{\mathsf{R}(\boldsymbol{\tau}) \mid \pi\} = \int_{\mathcal{T}} \mathsf{p}(\boldsymbol{\tau} \mid \pi)\mathsf{R}(\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau} = \int_{\mathcal{T}} \mathsf{p}^{\pi}(\boldsymbol{\tau})\mathsf{R}(\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau} = \mathbb{E}_{\boldsymbol{\tau}\sim\mathsf{p}^{\pi}(\cdot)}\{\mathsf{R}(\boldsymbol{\tau})\} \ . \tag{A.8}$$

Here, $\mathsf{p}^{\pi}(\boldsymbol{\tau})$ is the probability distribution of rollouts obtained by executing the policy π.

A.7 Law of total expectation

The law of total expectation (also law of iterated expectations) states that if x is a random variable whose expected value $\mathbb{E}\{\mathsf{x}\}$ is defined, and y is any random variable on the same probability space, then

$$\mathbb{E}\{\mathsf{x}\} = \mathbb{E}\{\mathbb{E}\{\mathsf{x} \mid \mathsf{y}\}\} \ , \tag{A.9}$$

i.e., the expected value of the conditional expected value of x given y is the same as the expected value of x.

A.8 Rules of probability

The two fundamental rules of probability theory are, see [3.1, p. 187],

$$\text{Sum rule} \quad \mathsf{p}(x) = \begin{cases} \sum_{y\in\mathcal{Y}} \mathsf{p}(x, y) & \text{if} \quad \mathsf{y} \quad \text{is discrete} \\ \int_{\mathcal{Y}} \mathsf{p}(x, y)\mathrm{d}y & \text{if} \quad \mathsf{y} \quad \text{is continuous} \end{cases} \tag{A.10}$$

$$\text{Product rule} \quad \mathsf{p}(x, y) = \mathsf{p}(y \mid x)\mathsf{p}(x) \ . \tag{A.11}$$

Here, $\mathsf{p}(x, y)$ represents the *joint probability*, one says "the probability of x and y". Similarly, $\mathsf{p}(y \mid x)$ denotes the *conditional probability*, it is the "the probability of y given x". Lastly, $\mathsf{p}(x)$ is the *marginal probability* and is simply phrased as "the probability of x". The sum and product rule can be combined to get the *law of total probability.* It is a fundamental rule relating marginal probabilities to conditional probabilities

$$\mathsf{p}(y) = \mathbb{E}_{\mathsf{x}\sim\mathsf{p}(x)}\{\mathsf{p}(y \mid x)\} = \begin{cases} \sum_{x\in\mathcal{X}} \mathsf{p}(x, y) = \sum_{x\in\mathcal{X}} \mathsf{p}(y \mid x)\mathsf{p}(x) & \text{if} \quad \mathsf{x}, \mathsf{y} \quad \text{are discrete} \\ \int_{\mathcal{X}} \mathsf{p}(x, y)\mathrm{d}x = \int_{\mathcal{X}} \mathsf{p}(y \mid x)\mathsf{p}(x)\mathrm{d}x & \text{if} \quad \mathsf{x}, \mathsf{y} \quad \text{are continuous} \end{cases} \tag{A.12}$$

A.9 The chain rule of conditional probabilities

Any joint probability distribution over many random variables can be decomposed into conditional distributions over only one variable. This observation is known as the chain rule

$$\mathsf{p}(x_N, \dots, x_0) = \mathsf{p}(x_0) \prod_{k=0}^{N-1} \mathsf{p}(x_{k+1} \mid x_k, \dots, x_0). \tag{A.13}$$

This follows directly from the definition of product rule (A.11). For example, if we apply the definition twice, we get

$$\begin{aligned} \mathsf{p}(x_2, x_1, x_0) &= \mathsf{p}(x_2 \mid x_1, x_0)\mathsf{p}(x_1, x_0) \\ \mathsf{p}(x_1, x_0) &= \mathsf{p}(x_1 \mid x_0)\mathsf{p}(x_0) \\ \mathsf{p}(x_2, x_1, x_0) &= \mathsf{p}(x_2 \mid x_1, x_0)\mathsf{p}(x_1 \mid x_0)\mathsf{p}(x_0) \ . \end{aligned} \tag{A.14}$$

A.10 Distributions

A.10.1 Bernoulli distribution

The (discrete) Bernoulli distribution will be referred to as $\mathcal{B}(x;\mu)$, where $x \in \{0,1\}$. It represents for example the result of flipping coin. The values 1 occurs with success probability μ and $x = 0$ occurs with failure probability $1-\mu$, respectively. Thus for random variable x, which is Bernoulli distributed, we write

$$\mathsf{x} \sim \mathcal{B}ern(x;\mu) = \mu^x (1-\mu)^{1-x} \ . \tag{A.15}$$

A.10.2 Normal distribution

The (continuous) normal distribution will be referred to as $\mathcal{N}(x;\mu,\sigma^2)$, where $x \in \mathbb{R}$. Thus when a random variable x is normally distributed with mean μ and standard deviation σ, one may write

$$\mathsf{x} \sim \mathcal{N}\Big(x;\mu,\sigma^2\Big) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \ . \tag{A.16}$$

A.10.3 Beta Distribution

The (continuous) beta distribution will be referred to as $\mathcal{B}eta(x;\alpha,\beta)$, where $x \in [0,1]$. It is a family of distributions characterized by two positive parameters, α and β. The beta distribution is often used to model the distribution of probabilities or proportions. The probability density function of a beta-distributed random variable x is given by

$$\mathsf{x} \sim \mathcal{B}eta(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \ , \tag{A.17}$$

where $B(\alpha,\beta)$ is the beta function, defined as

$$B(\alpha,\beta)=\int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\mathrm{d}t=\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}\ , \tag{A.18}$$

with $\Gamma(\cdot)$ denoting the gamma function.

A.11 Importance sampling

Often we encounter problems involving the evaluation of expectations of functions with respect to certain probability distributions. Direct evaluation of these expectations via analytical methods is often impractical, requiring instead the use of numerical methods such as **Monte Carlo** techniques. However, when dealing with rare events or distributions with heavy tails, standard Monte Carlo sampling becomes inefficient. This is where **importance sampling** comes into play. Importance sampling is a variance reduction technique that improves the efficiency of Monte Carlo estimation by focusing more sampling effort on important regions of the sample space that contribute more significantly to the quantity of interest. Let x be a random variable with probability density function $\mathsf{p}(x)$, and let $h(x)$ be a function of interest. We aim to estimate the expectation

$$\mathbb{E}\{h(\mathsf{x})\}=\int_{\mathcal{X}} h(x)\mathsf{p}(x)\,\mathrm{d}x\ . \tag{A.19}$$

Using a Monte Carlo approach, this expectation can be estimated by sampling $\mathsf{x}_0,\mathsf{x}_1,\dots,\mathsf{x}_{N-1}$ from $\mathsf{p}(x)$ and computing

$$\hat{\mu}=\frac{1}{N}\sum_{i=0}^{N-1} h(\mathsf{x}_i)\ . \tag{A.20}$$

However, when $h(x)$ is significant only in regions where $\mathsf{p}(x)$ is small, the variance of $\hat{\mu}$ can be large, requiring many samples to achieve a reliable estimate. Importance sampling addresses this by changing the sampling distribution. Instead of sampling directly from $\mathsf{p}(x)$, we sample from a different distribution $\mathsf{q}(x)$, known as the **importance distribution**, which places more emphasis on the regions where $h(x)$ is large. The expectation can then be rewritten as

$$\mathbb{E}\{h(\mathsf{x})\}=\int_{\mathcal{X}} h(x)\frac{\mathsf{p}(x)}{\mathsf{q}(x)}\mathsf{q}(x)\,\mathrm{d}x\ . \tag{A.21}$$

The corresponding Monte Carlo estimator becomes

$$\hat{\mu}_{\mathrm{IS}}=\frac{1}{N}\sum_{i=0}^{N-1} h(\mathsf{x}_i)\frac{\mathsf{p}(\mathsf{x}_i)}{\mathsf{q}(\mathsf{x}_i)}\ , \tag{A.22}$$

where $\mathsf{x}_i\sim\mathsf{q}(x)$. The term $\frac{\mathsf{p}(x)}{\mathsf{q}(x)}$ is called the **importance weight**.

- **Importance Distribution** $\mathsf{q}(x)$: This should be chosen to be "‘close"’ to $\mathsf{p}(x)$, but more concentrated in the regions where $h(x)$ is large.

- **Importance Weight** $\mathsf{w}(x) = \frac{\mathsf{p}(x)}{\mathsf{q}(x)}$: Compensates for the change in the sampling distribution, ensuring that the estimator remains unbiased.

One of the primary goals of importance sampling is to reduce the variance of the estimator compared to standard Monte Carlo. The variance of the importance sampling estimator is given by

$$\mathrm{Var}(\hat{\mu}_{\mathrm{IS}}) = \frac{1}{N} \int_{\mathcal{X}} \left[h(x) \frac{\mathsf{p}(x)}{\mathsf{q}(x)} - \mathbb{E}\{h(\mathsf{x})\} \right]^2 \mathsf{q}(x) \,\mathrm{d}x \ . \tag{A.23}$$

The variance depends on how well $\mathsf{q}(x)$ is chosen. If $\mathsf{q}(x)$ is chosen such that it places more weight on regions where $h(x)$ is large and $\mathsf{q}(x)$ is small, the variance can be significantly reduced.

- **Support Matching**: $\mathsf{q}(x) > 0$ wherever $h(x)p(x) > 0$.
- **Tail Behavior**: $\mathsf{q}(x)$ should capture the tail behavior of $h(x)\mathsf{p}(x)$ if the tail significantly contributes to the expectation.
- **Balance Between Accuracy and Complexity**: A well-chosen $\mathsf{q}(x)$ balances improved efficiency against the increased computational cost of evaluating the importance weights.
- **Choice of Importance Distribution**: The performance of importance sampling heavily depends on the choice of $\mathsf{q}(x)$. Often, it is beneficial to use domain knowledge or pre-existing approximations to design $\mathsf{q}(x)$.
- **Computational Cost**: While importance sampling can reduce variance, the cost of evaluating the importance weights and sampling from $\mathsf{q}(x)$ should be considered.
- **Bias and Consistency**: Despite being unbiased in theory, practical implementations may introduce bias due to numerical issues or poor choice of $\mathsf{q}(x)$.

A.12 Surprise and Entropy

Surprise $\mathsf{s}(x)$ is "inversely"[1] related to probability $\mathsf{p}(x)$. Surprise is measured in nats and defined by

$$\mathsf{s}(x) = \ln(\mathsf{p}(x)) \ . \tag{A.24}$$

Events with low probability are more surprising, while events with high probability are less surprising. The intuition is that rare events (low probability) contain more information or are more surprising when they occur compared to common events (high probability). In essence, surprise in information theory provides a mathematical way to quantify how

[1]The primary reason for using the logarithm, and not the inverse, is its additive property. When dealing with independent events, the total surprise should be the sum of the surprises of the individual events. In addition, the logarithmic function provides a more intuitive scale for measuring surprise. Very small probabilities translate into larger values of surprise in a way that is more manageable and interpretable.

much new information is gained when observing an event, based on how unexpected that event was. (Shannon) *Entropy* is the average surprise (or average information content) of a random variable given by

$$\mathbb{H}[x] = \begin{cases} -\sum_{x\in\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{p}(x)) & \text{if} \quad \mathsf{x} \quad \text{is discrete} \\ -\int_{\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{p}(x))\mathrm{d}x & \text{if} \quad \mathsf{x} \quad \text{is continuous} \; . \end{cases} \tag{A.25}$$

Entropy is crucial in information theory. It measures the expected amount of surprise (information) inherent in the distribution $\mathsf{p}(x)$ of the random variable x.

A.13 Relative Entropy

The *Kullback-Leibler (KL) divergence* is a statistical measure that quantifies the difference between two probability distributions. It is also known as relative entropy. For two probability distributions $\mathsf{p}(x)$ and $\mathsf{q}(x)$, the KL divergence is mathematically defined, for discrete random variable, as

$$D_{\mathrm{KL}}(\mathsf{p} \parallel \mathsf{q}) = -\sum_{x\in\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{q}(x)) + \sum_{x\in\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{p}(x)) = \sum_{x\in\mathcal{X}} \mathsf{p}(x)\ln\left(\frac{\mathsf{p}(x)}{\mathsf{q}(x)}\right) \tag{A.26}$$

and, for continuous random variables, as

$$D_{\mathrm{KL}}(\mathsf{p} \parallel \mathsf{q}) = -\int_{\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{q}(x))\mathrm{d}x + \int_{\mathcal{X}} \mathsf{p}(x)\ln(\mathsf{p}(x))\mathrm{d}x = \int_{\mathcal{X}} \mathsf{p}(x)\ln\left(\frac{\mathsf{p}(x)}{\mathsf{q}(x)}\right)\mathrm{d}x \; . \tag{A.27}$$

KL divergence measures the expected excess surprise from using distribution $\mathsf{q}(x)$ as a model instead of the true distribution $\mathsf{p}(x)$. It can be thought of as the information lost when $\mathsf{q}(x)$ is used to approximate $\mathsf{p}(x)$. Key properties of KL divergence include

- It is always non-negative.
- It equals zero if and only if the two distributions are identical.
- It is not symmetric, meaning $D_{\mathrm{KL}}(\mathsf{p} \parallel \mathsf{q}) \neq D_{\mathrm{KL}}(\mathsf{q} \parallel \mathsf{p})$.
- It does not satisfy the triangle inequality, so it's not a true metric.

In essence, KL divergence provides a way to measure how one probability distribution diverges from another.

A.14 Max Entropy formulation

Policy Improvement appears reasonable but has some drawbacks:

1. **Indistinguishably Close Values**: When two options have nearly the same values, we expect both to have roughly equal chances of being chosen. However, this model does not account for this nuance. In the extreme case of identical values, an arbitrary rule is required to break ties.

2. **Lack of Smoothness**: Policy Improvement is not smooth. Changes in values only impact the decision if they alter which item has the highest value. This binary approach overlooks the gradations in value.

3. **Inclusion of Lower Values**: We aim for a decision model where items with lower values are still occasionally chosen. This reflects a more realistic scenario where all options have a non-zero probability of selection, aligning with real-world decision-making processes.

By addressing these issues, we can develop a more robust and flexible decision model. A way to solve this is using the Entropy (A.25) as a regularizer to improve the properties of Policy Improvement, i.e.

$$V^\pi(\mathbf{x}) = \max_{\mathbf{u}\in\mathcal{U}} \left[Q^\pi(\mathbf{x},\mathbf{u}) + \beta \mathbb{H}(\pi(\mathbf{u} \mid \mathbf{x})) \right] \ , \tag{A.28}$$

with parameter β. After some calculations, we get the log-sum-exp[2] formulation

$$V^\pi(\mathbf{x}_k) = \log\operatorname{sum}\exp_{\mathbf{u}\in\mathcal{U}} Q^\pi(\mathbf{x},\mathbf{u}) = \beta \log \sum_{\mathbf{u}\in\mathcal{U}} \exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x},\mathbf{u})\right) \tag{A.29}$$

and the softmax[3] formulation, that is

$$\pi \leftarrow \operatorname{soft}\max_{\mathbf{u}\in\mathcal{U}} Q^\pi(\mathbf{x},\mathbf{u}) = \frac{\exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x},\mathbf{u})\right)}{\sum_{\mathbf{w}\in\mathcal{U}} \exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x},\mathbf{w})\right)} \ . \tag{A.30}$$

One can also interpret the softmax as an energy-based model, where the state-value function represent the energy and β is a temperature-like parameter.

[2]Actually a softmax operator.
[3]Actually a softargmax operator.