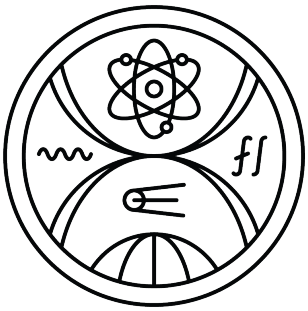
Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

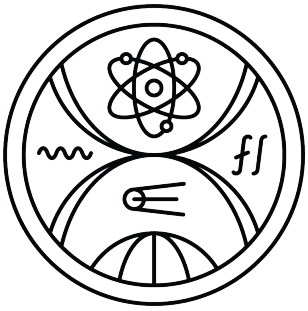


Neuro-symbolic approach to reinforcement learning in robotics

Master's Thesis

2025
Bc. Tomáš Bisták

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics



Neuro-symbolic approach to reinforcement learning in robotics

Master's Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	doc. RNDr. Martin Homola, PhD.

Bratislava, 2025
Bc. Tomáš Bisták

48865456

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Tomáš Bisták
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Neuro-symbolic approach to reinforcement learning in robotics

Annotation: Reinforcement learning (RL) has become a standard machine learning approach to solving sequential decision tasks, including robotics. Yet, purely experience-driven RL is a computationally intensive black-box with low level of explainability. Neuro-symbolic approaches offer to meet these challenges, also thanks to transparency of the symbolic level.

Aim: 1. Become familiar with RL literature and with selected neuro-symbolic approaches applicable to sequential tasks, with focus on robotic manipulation.
2. Extend Delfosse et al.'s (2023) model to robotics domain and implement the reasoning component for predicting the consequences of robot's actions.
3. Using a task with objects on a table, compare the hybrid approach with pure RL approach in terms of explainability and accuracy of predictions.

Literature: Delfosse Q., Shindo H., Dhami D., Kersting K. (2023) Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction. NIPS, https://arxiv.org/abs/2306.01439
Yang W.-C. , Marra G., Rens G., De Raedt L. (2023) Safe Reinforcement Learning via Probabilistic Logic Shields. IJCAI, https://www.ijcai.org/proceedings/2023/0637.pdf
Gomez R., Sridharan M., Riley H. (2020) What do you really want to do? Towards a Theory of Intentions for Human-Robot Collaboration. Annals of Math. and Artif. Intel. https://doi.org/10.1007/s10472-019-09672-4

Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	doc. RNDr. Martin Homola, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	05.10.2023

Approved: 05.12.2023 prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

Student Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Bisták
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Neuro-symbolic approach to reinforcement learning in robotics
Neuro-symbolový prístup k učeniu posilňovaním v robotike

Anotácia: Učenie posilňovaním (RL) sa stalo štandardným prístupom strojového učenia k riešeniu sekvenčných rozhodovacích úloh, vrátane robotiky. Napriek tomu je takéto učenie výpočtovo náročná čierna skrinka s nízkou úrovňou vysvetliteľnosti. Neuro-symbolové prístupy ponúkajú riešenia k týmto výzvam, aj vďaka transparentnosti symbolovej úrovne.

Cieľ: 1. Preštudujte literatúru o RL a o vybraných neuro-symbolových prístupoch aplikovateľných na sekvenčné úlohy, so zameraním na robotickú manipuláciu.
2. Rozšírte model Delfosse a spol. (2023) na oblasť robotiky a implementujte komponent usudzovania na predpovedanie dôsledkov akcií robota.
3. Na vybratej úlohe s objektami na stole porovnajte hybridný prístup s čistým prístupom pomocou RP z hľadiska vysvetliteľnosti a presnosti predikcií.

Literatúra: Delfosse Q., Shindo H., Dhami D., Kersting K. (2023) Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction. NIPS, https://arxiv.org/abs/2306.01439
Yang W.-C. , Marra G., Rens G., De Raedt L. (2023) Safe Reinforcement Learning via Probabilistic Logic Shields. IJCAI, https://www.ijcai.org/proceedings/2023/0637.pdf
Gomez R., Sridharan M., Riley H. (2020) What do you really want to do? Towards a Theory of Intentions for Human-Robot Collaboration. Annals of Math. and Artif. Intel. https://doi.org/10.1007/s10472-019-09672-4

Vedúci: prof. Ing. Igor Farkaš, Dr.
Konzultant: doc. RNDr. Martin Homola, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 05.10.2023

Dátum schválenia: 05.12.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Declaration: I hereby certify that I have written this thesis independently without any help from third parties and without using any sources or aids other than those indicated.

. . . . . .

Bratislava, 2025 Bc. Tomáš Bisták

Acknowledgments: I would like to thank prof. Ing. Igor Farkaš, Dr. for his supervision, willingness, and patience during my work on this thesis. I am also thankful to doc. RNDr. Martin Homola, PhD. for his precious advice on symbolic and neuro-symbolic methods in artificial intelligence. Lastly, I give particular gratitude to my father, Ing. Pavol Bisták, PhD., for his valuable support.

Abstract

Keywords:

Abstrakt

Kľúčové slová:

Contents

List of Figures

List of Tables

Nomenclature

General

x	scalar
$\boldsymbol{v}$	vector

Reinforcement Learning

$\mathcal{M}$	Markov decision process
$\mathcal{T}$	set of decision points/time steps
T	terminal time step
t	time step
$\mathcal{S}$	set of states
$\mathcal{S}^T$	set of terminal states
$\mathcal{S}^N$	set of non-terminal states
s_0	initial-state probability distribution
S_t	state at time step t
$\mathcal{A}$	set of actions
$\mathcal{A}(s)$	actions admissible in state s
A_t	action at time step t
$\mathcal{R}$	set of rewards
R_t	reward at time step t
p	dynamics
γ	discount factor

τ	trajectory
π	policy
G_t	return after time step t

Acronyms

HRL hierarchical reinforcement learning. 17, 26, 30

MDP Markov decision process. xi, 13–18, 20, 21, 25, 26

oSMDP options semi-Markov decision process. xi, 18, 19

RL reinforcement learning. 13, 15–17, 19, 21, 23, 27–30

1 Introduction

[Rie+20], [Gla+22]

Part I

State of the Art

2 Logics

3 Artificial Neural Networks

4 Reinforcement Learning

This chapter presents the reinforcement learning (RL) framework in more detail. After formally defining RL's basic model and objectives, we briefly introduce some classical solution methods.

4.1 Standard Setting

The description of the standard RL setting provided in this section is based on the treatises of Sutton and Barto [SB18] and Puterman [Put94].

4.1.1 Agent-Environment Interaction

The fundamental concern of RL resides in training an *agent* through *continual interaction* with an *environment* that changes under the agent's intervention and responds with *numerical feedback* implicitly encoding the learning goals. This interaction model is formalized as a *(discrete-time) Markov decision process (MDP)*.[1]

Definition 1 (Discrete-Time Markov Decision Process)**.** *A* ***(discrete-time) Markov decision process*** *is a 6-tuple* $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ *where*

- $\mathcal{T} = \{0, 1, \ldots, T\}$ *is a non-empty discrete set of* ***decision points*** *or* ***time steps****, with* T *being either a random variable over* $\mathbb{N}^+$ *or infinity,*
- $\mathcal{S} = \mathcal{S}^T \cup \mathcal{S}^N$ *is a non-empty set of* ***states*** *composed of a set of* ***terminal states*** $\mathcal{S}^T$ *and a set of* ***non-terminal states*** $\mathcal{S}^N$*, with* $\mathcal{S}^T \cap \mathcal{S}^N = \emptyset$*, such that* $\mathcal{S}^T = \emptyset$ *if and only if* $T = \infty$*,*
- $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}(s)$ *is a non-empty set of* ***actions****, with* $\mathcal{A}(s)$ *being the set of* ***admissible actions*** *in a state* $s \in \mathcal{S}$ *such that* $\mathcal{A}(s) = \emptyset$ *if and only if* $s \in \mathcal{S}^T$*,*
- $s_0 : \mathcal{S} \to [0, 1]$ *is an* ***initial-state probability distribution****,*
- $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ *defines the process's* ***dynamics*** *via a conditional probability distribution* $p(\cdot, \cdot \mid s, a)$ *over* $\mathcal{S} \times \mathcal{R}$ *for every state* $s \in \mathcal{S}$ *and every action* $a \in \mathcal{A}(s)$*, with* $\mathcal{R} \subseteq \mathbb{R}$ *being a set of* ***rewards****, and*

[1] There also exists a continuous-time formulation [Put94]. Still, we restrict our discussion solely to the discrete-time setting since it best suits our application.

- $\gamma \in [0, 1]$ *is a* ***discount factor*** *such that* $\gamma = 1$ *only if* $T \neq \infty$.

If $\mathcal{S}, \mathcal{A}$, *and* $\mathcal{R}$ *are all finite, we call* $\mathcal{M}$ *a* ***finite (discrete-time) Markov decision process****. Moreover, a discrete-time Markov decision process in which* T *is a random variable is said to frame a* ***finite-horizon*** *or* ***episodic task****. In contrast, a process with* $T = \infty$ *is said to represent an* ***infinite-horizon*** *or* ***continuing task****.*

The agent-environment interaction a given MDP $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ expresses can be understood as follows. The interaction itself extends in time and is divided into non-overlapping successive periods, the beginnings of which determine a discrete set of time steps $t = 0, 1, \ldots, T$. The environment is initially in a state $S_0 \sim s_0$. At each time step $t \neq T$, the agent observes the current state of the environment $S_t \in \mathcal{S}$ and performs an action $A_t \in \mathcal{A}(S_t)$. Applying the action A_t in the state S_t causes a possibly stochastic *transition* to a next state S_{t+1} in the environment (observed in the following time step) associated with a reward R_{t+1} provided to the agent where $(S_{t+1}, R_{t+1}) \sim p(\cdot, \cdot \mid S_t, A_t)$. When the agent is faced with a continuing task, this feedback loop repeats indefinitely, creating a *trajectory*

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots).$$

For episodic tasks, this process continues until the environment reaches a terminal state $S_T \in \mathcal{S}^T$ at some point T, resulting in a finite trajectory

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$$

called an *episode*. Once an episode is finished, the environment is reset to one of the initial states according to the distribution s_0 to start a new episode.

This agent-environment interaction is also depicted in Figure 4.1, which further highlights the *Markov property* of MDPs meaning that each transition only depends on the current state of the environment and the action taken in that state. Previous interaction has thus no effect on future outcomes.

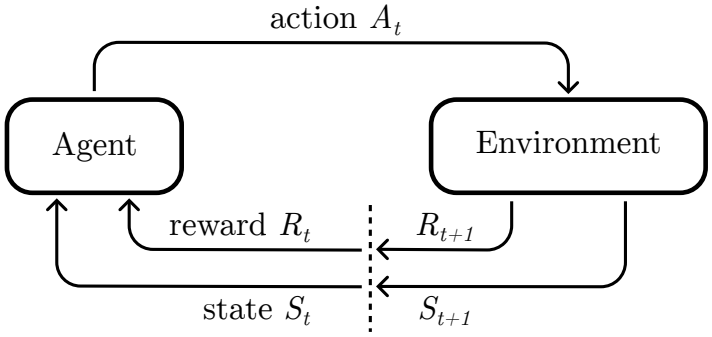


Figure 4.1: The agent-environment interaction in an MDP.

4.1.2 Policies, Returns, and Values

Having a formal model of the agent-environment interaction, we can now closely look at what defines an agent and how its behavior is assessed in the context of RL.

Each agent clearly must possess a possibly stochastic way of deciding which action to take in every non-terminal state of the given environment, since the environment's dynamics are unknown to the agent. Such a description of the agent's behavior is called a *policy* and is also sufficient to fully describe the agent within RL.

Definition 2 (Policy)**.** *A **policy** for a given MDP $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ is any function $\pi : \mathcal{S}^N \times \mathcal{A} \to [0, 1]$ that maps every non-terminal state $s \in \mathcal{S}^N$ to a probability distribution $\pi(\cdot \mid s)$ over the admissible actions $\mathcal{A}(s)$.*

To evaluate the agent's course of action, we use *returns*, i.e., the cumulative rewards that the agent collects from the environment during interaction from a specific time step onward.

Definition 3 (Return)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP. The **return** G_t after a time step $t \in \mathcal{T}$ over a trajectory $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots)$ in $\mathcal{M}$ is defined as*

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{4.1}$$

where T is either the point of termination of τ if any, or infinity otherwise.

Notice the application of the discount factor in Definition 3, whose role here is twofold. First, it serves as a mechanism to account for the uncertainty of future rewards. Second, its presence avoids infinite returns in the case of continuous tasks.

Focusing on returns rather than on immediate rewards enables us to study the characteristics of the agent's behavior in the long term. In particular, we are interested in what return the agent can expect to receive if it follows a given policy, for which we introduce the notions of *state values* and *action values*.

Definition 4 (State and Action Values)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP, and let π be a policy for $\mathcal{M}$.*

*The **state-value function** $v_\pi : \mathcal{S} \to \mathbb{R}$ for π is defined for every state $s \in \mathcal{S}$ as*

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &\equiv \mathop{\mathbb{E}}_{\substack{A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ldots \\ A_i \sim \pi(\cdot \mid S_i),\quad (S_{i+1}, R_{i+1}) \sim p(\cdot, \cdot \mid S_i, A_i)}} \left[G_t \mid S_t = s \right]. \end{aligned} \tag{4.2}$$

*The **action-value function** $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for π is defined for every state $s \in \mathcal{S}$ and every action $a \in \mathcal{A}(s)$ as*

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &\equiv \mathop{\mathbb{E}}_{\substack{R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ldots \\ A_i \sim \pi(\cdot \mid S_i),\quad (S_{i+1}, R_{i+1}) \sim p(\cdot, \cdot \mid S_i, A_i)}} \left[G_t \mid S_t = s, A_t = a \right]. \end{aligned} \tag{4.3}$$

Value functions may also be expressed via the following *Bellman equations* derived from the definitions of values and returns. For brevity, we introduce their forms only for the case of finite MDPs.

Definition 5 (Bellman Equations)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be a finite MDP, and let π be a policy for $\mathcal{M}$.*

*The **Bellman equation for** v_π is given for every state $s \in \mathcal{S}$ by*

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{(s',r) \sim p(\cdot,\cdot \mid s,a)} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]. \tag{4.4}$$

*The **Bellman equation for** q_π is given for every state $s \in \mathcal{S}$ and every action $a \in \mathcal{A}(s)$ by*

$$q_\pi(s,a) = \sum_{(s',r) \sim p(\cdot,\cdot \mid s,a)} p(s', r \mid s, a) \left[r + \gamma q_\pi(s', a') \right]. \tag{4.5}$$

4.1.3 Optimality and Learning Objectives

The ultimate goal of RL is to teach the agent to behave optimally within a given environment in the long term, i.e., to follow a policy that leads the agent to the best (maximal) returns possible. Such a policy need not be unique, however, all of these *optimal policies* share the same *optimal value functions*.

Definition 6 (Optimal State and Action Values)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP.*

*The **optimal state-value function** $v_* : \mathcal{S} \to \mathbb{R}$ for $\mathcal{M}$ is defined for every state $s \in \mathcal{S}$ as*

$$v_*(s) = \max_\pi v_\pi(s). \tag{4.6}$$

*The **optimal action-value function** $q_* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for $\mathcal{M}$ is defined for every state $s \in \mathcal{S}$ and every action $a \in \mathcal{A}(s)$ as*

$$q_*(s,a) = \max_\pi q_\pi(s,a). \tag{4.7}$$

Definition 7 (Optimal Policy)**.** *An **optimal policy** π^* for a given MDP $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ is any policy for $\mathcal{M}$ such that $v_{\pi^*} = v_*$ (or that $q_{\pi^*} = q_*$).*

Further applying the Bellman equations for finite MDPs, we obtain the *Bellman optimality equations* as alternative definitions of the optimal value functions. These equations and their counterparts for the general case are often at the core of the methods for constructing an optimal policy.

Definition 8 (Bellman Optimality Equations)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be a finite MDP.*

*The **Bellman optimality equation for** v_* is given for every state $s \in \mathcal{S}$ by*

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{(s',r) \sim p(\cdot,\cdot|s,a)} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right]. \tag{4.8}$$

*The **Bellman optimality equation for** q_* is given for every state $s \in \mathcal{S}$ and every action $a \in \mathcal{A}(s)$ by*

$$q_*(s,a) = \sum_{(s',r) \sim p(\cdot,\cdot|s,a)} p(s', r \mid s, a) \left[r + \gamma \max_{a' \in \mathcal{A}(s')} q_*(s', a') \right]. \tag{4.9}$$

With the above-discussed concept of optimality, we are ready to formulate the standard *reinforcement-learning problem* precisely.

Definition 9 (Reinforcement-Learning Problem)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP. Find an optimal policy π^* for $\mathcal{M}$.*

4.2 Hierarchical Reinforcement Learning

When modeling real-world problems in the standard RL framework, we often have to construct MDPs with quite complex state and action spaces focused on low-level features to capture the intricacies of the actual interaction. A canonic example of this is the problem of object manipulation using a robotic arm, in which the agent can only control the arm's joints to provide an end-to-end solution. Hence, the exact configuration of the arm needs to be part of the current state and actions must correspond to primitive joint move.

Such an overload with raw aspects of interaction may hamper the agent's efforts to abstract from these details and reach the ultimate goal. One way of tackling this issue is to inject explicit abstraction hierarchies into the agent-environment interaction to equip the agent with our prior domain knowledge or to ensure a specific conceptual decomposition. We describe this approach of hierarchical reinforcement learning (HRL) in the following.

4.2.1 Options Framework

The options framework [SPS99] is an HRL method for introducing temporal abstraction. More precisely, this framework enables the agent to take a compound macro-action with a longer completion horizon if admissible and to remember this intention throughout several consecutive steps of simple actions until a terminating condition for the selected macro-action is satisfied. These macro-actions are called *options* and are formally defined as below.

Definition 10 (Option)**.** *Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP. An **option** within $\mathcal{M}$ is any triple $\langle \mathcal{I}, \pi, \beta \rangle$ where*

- *$\mathcal{I} \subseteq \mathcal{S}^N$ is an **initiation set**,*
- *$\pi : \mathcal{S}^N \times \mathcal{A} \to [0, 1]$ is a policy for $\mathcal{M}$, and*
- *$\beta : \mathcal{S} \to [0, 1]$ is a **termination condition**, with $\beta(s) = 1$ for all $s \in \mathcal{S}^T$.*

*An option $\langle \mathcal{I}, \pi, \beta \rangle$ is an **admissible option** in a given state $s \in \mathcal{S}$ if and only if $s \in \mathcal{I}$ and no other option is currently active.*

Fixing a set of options for a selected (discrete-time) MDP gives us an extended agent-environment interaction model framed as a *(discrete-time) options semi-Markov decision process (oSMDP)*. This decision process is considered *semi-Markov* because the execution of a single option terminates under potentially stochastic conditions and may span multiple time steps.

Definition 11 (Discrete-Time Options Semi-Markov Decision Process)**.** *A **(discrete-time) options semi-Markov decision process** is a tuple $\mathcal{M}_\mathcal{O} = \langle \mathcal{M}, \mathcal{O} \rangle$ where*

- *$\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ is a (discrete-time) MDP,*
- *$\mathcal{O}$ is a non-empty set of options within $\mathcal{M}$.*

The agent-environment interaction described by an oSMDP $\mathcal{M}_\mathcal{O} = \langle \mathcal{M}, \mathcal{O} \rangle$ slightly differs from that expressed via the underlying MDP $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$. At each time step $t \neq T$ in which no option is activated, the agent can now initiate any option $O_t = \langle \mathcal{I}, \pi, \beta \rangle \in \mathcal{O}$ admissible in the current state of the environment $S_t \in \mathcal{S}$. After this initiation, the agent interacts with the environment by performing actions from $\mathcal{A}$ according to π, with the probability $\beta(S_{t'})$ of terminating O_t in any successive state $S_{t'} \in \mathcal{S}$ where $t' > t$. Upon termination of O_t in a state $S_{t'} \in \mathcal{S}^N$, the agent receives the cumulative discounted reward $R_{t:t'} = \sum_{k=t+1}^{t'} \gamma^{k-t-1} R_k$ and chooses among the admissible options in $S_{t'}$.

Figure 4.2 illustrates this extended agent-environment interaction defined by an oSMDP for better understanding.

4.2.2 Meta-Controller/Controller Model

The options framework itself does not deal with how policies for the different options are obtained, i.e., whether they should be given or learned by the agent. The meta-controller/controller model [Kul+16] addresses this question by identifying three components of the agent: a meta-controller, a controller, and a critic.

The *meta-controller* is responsible for learning a policy π over the set of options in the given oSMDP $\mathcal{M}_\mathcal{O}$ observing transitions $S_t \xrightarrow{O_t} S_{t'}$ and *extrinsic rewards* $R^{(\text{ext})}_{t:t'}$ after each selected option O_t as explained in Section 4.2.1.

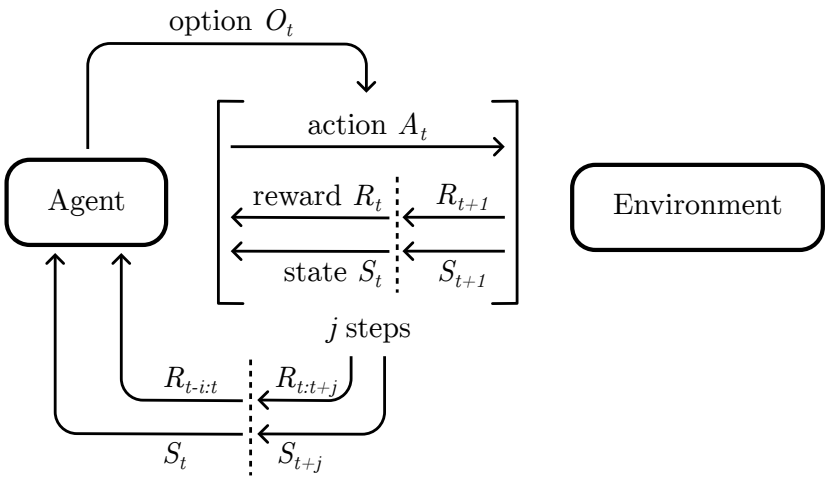


Figure 4.2: The agent-environment interaction in an oSMDP.

The *controller* is tasked with learning policies π_o over the set of actions in $\mathcal{M}_\mathcal{O}$ for the respective options o in $\mathcal{M}_\mathcal{O}$ based on transitions $S_t \xrightarrow{A_t} S_{t+1}$ and *intrinsic rewards* $R_t^{(\text{int})}$ after each individual action A_t taken.

The role of the *critic* is to calculate the intrinsic rewards from the action-induced transitions for the controller. The differentiation between intrinsic and extrinsic rewards allows for an inevitable decoupling of the goal of learning how to execute a particular option from the overall learning objectives, which need not be aligned.

With this internal structure of the agent, the meta-controller/controller model assumes that the meta-controller and the controller act and learn as a whole. The controller thus instructs the agent to behave according to the corresponding policy once the meta-controller initiates an option. This results in the flow of the agent-environment interaction shown in Figure 4.3.

Note that the meta-controller and the controller are designed to learn simultaneously in the original model. However, we may as well (pre-)train the controller in advance and only combine it with the meta-controller afterward. Another alternative is to (pre-)train both components separately. These possible variations demonstrate that this model is rather flexible and offers space for experimentation.

4.2.3 Other Hierarchical Extensions

The options framework and the meta-controller/controller model presented in previous sections both seek to introduce temporal abstraction via a two-level hierarchy of actions. Since time is not the only dimension where hierarchies can be established, other hierarchical extensions of the standard RL framework exist [BM03; HML22]. We briefly discuss the two most notable ones in this section.

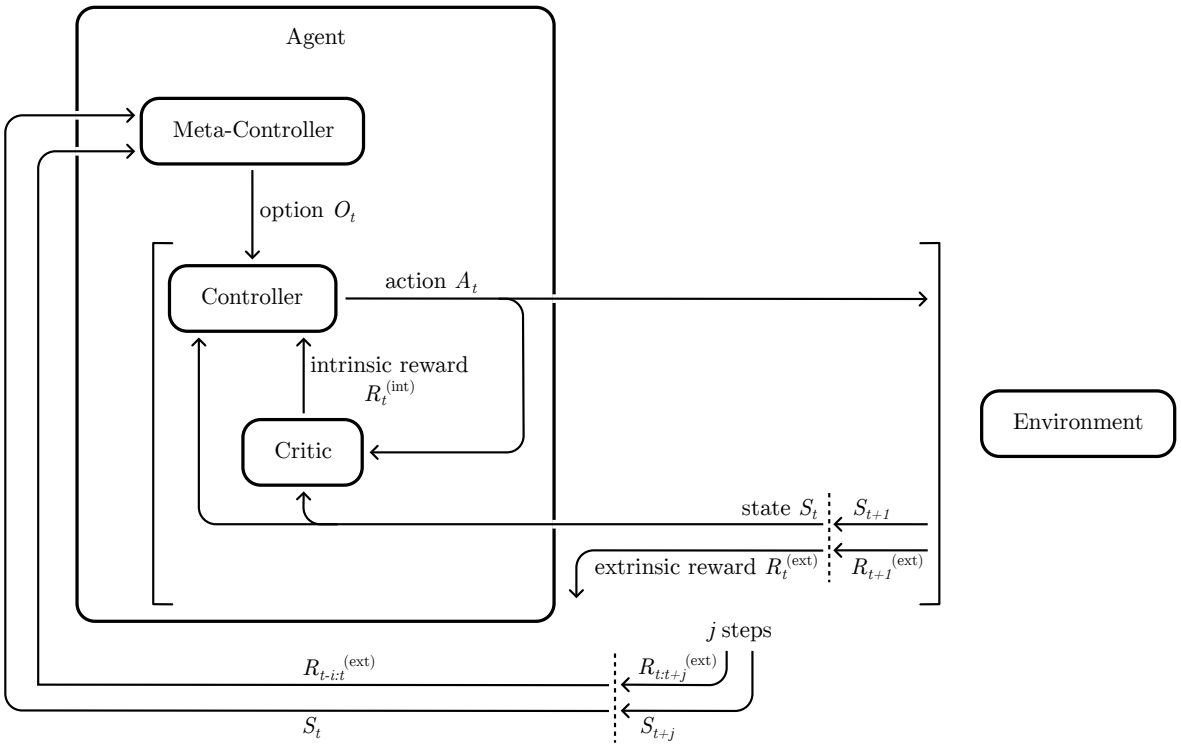


Figure 4.3: The agent-environment interaction in the meta-controller/controller model.

Feudal Reinforcement Learning

Feudal reinforcement learning [DH92] provides a mechanism for working with spatial abstraction by creating a multi-level vertical hierarchy over the state space. At each level, the state space is partitioned into groups of "close-by" states, where the closeness of states is determined by a designer-defined metric. The lower the level in the hierarchy, the finer the resolution of the partitioning. In addition, these views of the state space are aligned such that each group at a certain level is divided into several groups at the level immediately below.

Although the agent in this setting still learns a single policy as described in Section 4.1.2, this hierarchical structuring of the state space opens an opportunity for the agent to do so by examining value functions at different resolutions.

MAXQ Value-Function Decomposition

The MAXQ framework [Die00] tries to achieve task-based abstraction by constructing a hierarchy over the action space of a given MDP $\mathcal{M}$. Particularly, the task represented by $\mathcal{M}$ is decomposed into sub-tasks, each described by its own MDP where performing an action consists in either taking a primitive action from $\mathcal{M}$ or executing another sub-task (without cyclical references). This procedure yields a hierarchy in the form of a task graph with the different sub-tasks and the original task as nodes and oriented edges indicating the sub-task inter-dependencies.

Due to the potentially recursive use of sub-tasks as actions, we may consider this method of hierarchization as a generalization of the options framework with one caveat. The reward the agent receives here after deciding to execute another sub-task i following the respective policy π_i in a state s is defined as the expected return on solving that sub-task under π_i from s with a special intrinsic reward upon successful completion, or an approximation thereof. This feedback scheme breaks down the value function into components corresponding to the individual sub-tasks and thus facilitates abstraction.

4.3 Solution Techniques

Having examined a range of RL settings, we now present several techniques to achieve RL's objective, i.e., to teach the agent to behave optimally by finding an optimal policy maximizing the expected return. We assume that the environment dynamics are unknown to the agent. The only way to discover which actions yield the greatest returns in the individual states is thus to infer that from experience. The nature of this experience varies between the standard and the hierarchical frameworks, leading to different approaches. However, the methods developed for either case can be distinguished based on how they organize the learning process. The search for an optimal policy may be guided by information about state and action values. Alternatively, it is possible to directly optimize a parametric policy so that values are implicitly stored in the policy's parameters instead of being explicitly computed or approximated. Another option is to combine these ideas into one. In this section, we progressively discuss selected solution methods from all these three categories.

We begin by describing techniques depending on approximations of value functions as auxiliary measures of policy quality. Our attention then turns to policy-based methods, some incorporating value-function approximation. Finally, we study a few state-of-the-art neuro-symbolic approaches to RL.

The sub-sections on value-based and policy-based methods draw largely upon the work of Sutton and Barto [SB18].

4.3.1 Value-Based Methods

Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be a finite MDP. A simple way to find an optimal policy for $\mathcal{M}$, which we consider in this section, would be to iteratively optimize a certain initial policy based on the returns we expect it to generate as given by its value functions. This procedure is known as *generalized policy iteration* and can be described in more detail as follows.

We start with an arbitrary policy π_0 for $\mathcal{M}$. Next, *policy evaluation* is carried out, determining the state-value function v_{π_0} or the action-value function q_{π_0} for π_0, e.g.,

with Bellman equations. Consequently, a new policy π_1 is formed through the process of *policy improvement* that greedily optimizes π_0 w.r.t. v_{π_0} or q_{π_0} for all $s \in \mathcal{S}$:

$$\pi_1(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{(s',r) \sim p(\cdot,\cdot \mid s,a)} p(s', r \mid s, a) \left[r + \gamma v_{\pi_0}(s') \right], \text{ or} \tag{4.10}$$

$$\pi_1(s) = \arg\max_{a \in \mathcal{A}(s)} q_{\pi_0}(s, a). \tag{4.11}$$

Policy evaluation and improvement are then repeated until the last policy performs sufficiently well since there is a guarantee that the constructed sequence of policies $\pi_0, \pi_1, \ldots$ approaches an optimal policy as the number of iteration steps grows to infinity.

Generalized policy iteration sets forth a theoretically sound scheme for producing desired results. In practice, we have to take into account that the environment does not reveal its dynamics p. The value functions can thus only be approximated from collected experience. As state and action values are defined in terms of expectations, we estimate them via sample averages. Recalculating these estimates from scratch after each policy improvement would, however, require a considerable amount of agent-environment interaction, which we strive to minimize. To this end, we maintain a single approximation of the state-value function or the action-value function, denoted V_π and Q_π, respectively, and initialized such that $V_\pi(s) = 0$ or $Q_\pi(s, a) = 0$ for all terminal states $s \in \mathcal{S}^T$ and actions $a \in \mathcal{A}(s)$. Policy evaluation then reduces to updating the value-function approximation with a fresh sample of limited size to better resemble the value function of the last policy. Although approximating the state-value function for the purposes of policy improvement still demands knowing p, it may benefit policy-based methods introduced later, which justifies studying this case as well.

Unfortunately, this realization of policy evaluation raises a subtle issue regarding what policies we create and keep track of. Generalized policy iteration prescribes that the previous policy should be greedily optimized in the policy-improvement step to generate a new, deterministic policy. Behaving according to such a policy while gathering samples for the update of our value-function approximation may result in crucial states never being visited and important actions never being tried. Hence, our estimates of their values would potentially be far from the truth. This conflict between the need for sub-optimal behavior to correctly assess the advantages of taking different paths and the ultimate goal of policy optimization is called *the problem of exploration and exploitation*. In particular, the agent must continuously explore various options and simultaneously exploit the already-gained knowledge of the task to the maximum.

One solution to this situation is to always work with *soft policies*, that is, policies assigning non-zero probabilities to all actions from $\mathcal{A}(s)$ in every state $s \in \mathcal{S}$. Especially common are *ϵ-greedy policies*, under which the agent chooses its action uniformly randomly from $\mathcal{A}(s)$ with a small probability $\epsilon > 0$ (so, each with the chance $\frac{\epsilon}{\mathcal{A}(s)}$)

and otherwise selects greedily the (supposedly) optimal action. Policies obtained by policy improvement modified to output soft policies can then be directly used for the approximate policy evaluation. We refer to the methods adopting this technique as *on-policy* because they optimize the same policy that dictates the agent's behavior during interaction. A noticeable downside of this approach is that we may at best reach an optimal soft policy. Therefore, another possibility is to iteratively optimize a deterministic *target policy* according to values estimated from experience sampled with a *behavior policy* – always a soft version of the last target policy. This separation enables us to arrive at an optimal policy but necessitates extra care when converting expectations between different probability distributions. Due to the complexity of these *off-policy* methods and the fact that we do not employ them in our research, we do not discuss this topic further and direct an interested reader to the book from Sutton and Barto [SB18]. In the rest of this chapter on RL, we assume that on-policy methods are utilized.

As a final remark on practical implementation, it is inevitable to highlight that generalized policy iteration constructs a sequence of policies, while only ever using the last one. We thus merely store the last (current) policy π and update it in place.

In the remainder of this section, we first present two fundamental ways of performing policy evaluation and then examine how to cope with tasks with continuous state spaces.

On-Policy Monte Carlo

Suppose that $\mathcal{M}$ encodes an episodic task. The minimal experience allowing us to update the current value-function approximation purely based on interaction data is naturally a single episode where the agent follows the current policy. Having the trajectory $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$ from such a sample episode suffices to compute the returns G_t in that episode and update the estimates for states S_t or state-action pairs (S_t, A_t) in τ respectively as

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha \left[G_t - V_\pi(S_t)\right], \text{ or} \tag{4.12}$$

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha \left[G_t - Q_\pi(S_t, A_t)\right], \tag{4.13}$$

where $\alpha \in (0, 1]$ is the learning rate determining the weight of recent observations relative to past experience. These update rules constitute a policy evaluation technique called on-policy Monte Carlo (MC), with which we are guaranteed to converge to an optimal soft (ϵ-greedy) policy.

Note that there exist two variants of on-policy MC, namely, the *first-visit* and the *every-visit* method. The former updates the estimates exclusively with the returns after the first occurrence of the state or the state-action pair in the trajectory so that

all the sampled returns are mutually independent. The latter considers the return after every appearance as a valid sample. The convergence property holds for both versions and the differences between them are minor.

On-Policy Temporal Difference

One of the problems of on-policy MC is its reliance on the assumption of always terminating agent-environment interaction. On-policy temporal difference (TD) learning addresses this issue by replacing the whole sampled returns with returns from partial trajectories of a fixed length combined with the current estimate for the values of endpoints. More precisely, the *one-step TD* method for the approximation of the state-value function requires just one transition, from the current state S_t to a new state S_{t+1} yielding a reward R_{t+1}, with $(S_{t+1}, R_{t+1}) \sim p(\cdot, \cdot \mid S_t, A_t)$, and caused by an action $A_t \sim \pi(\cdot \mid S_t)$, after which it performs the update:

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha \left[R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)\right], \tag{4.14}$$

where $\alpha \in (0, 1]$ is the learning rate and the difference $R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)$ is called the *TD error*. The one-step TD method for the action-value-function approximation may need one extra transition, from the state S_{t+1} to a state $S_{t+2} \sim p(\cdot, R_{t+2} \mid S_{t+1}, A_{t+1})$ after an action $A_{t+1} \sim \pi(\cdot \mid S_{t+1})$, in which case we have the *SARSA* update:

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) - Q_\pi(S_t, A_t)\right], \tag{4.15}$$

or it may calculate the expected next action value under the current policy instead, making the *expected SARSA* update:

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_{a \in \mathcal{A}(S_{t+1})} Q_\pi(S_{t+1}, a) - Q_\pi(S_t, A_t)\right]. \tag{4.16}$$

The above-described *bootstrapping* is the essence of TD learning and enables us to improve the policy online, i.e., at every time step[2] and without waiting for the possibly indefinite interaction to terminate. Yet, on-policy TD learning converges to an optimal soft (ϵ-greedy) policy and usually does so even faster than on-policy MC when the learning rate is constant. These two aspects contribute to the trend of TD being often preferred over MC.

Apart from one-step TD techniques, there are also *n-step TD* methods that generalize the presented concept to partial trajectories of arbitrary length n, moving closer to MC as $n \to \infty$. TD methods with *eligibility traces* try to provide an extension along the same axis by looking into the past rather than the future. For a more detailed analysis of these approaches see [SB18].

[2]Starting from $t = 1$ if SARSA is used.

Parametric Approximations

Up to this point, we assumed that our MDP $\mathcal{M}$ was finite as stated at the beginning of this sub-section on value-based methods. The finiteness of $\mathcal{M}$ eases the representation of V_π, Q_π, and π, which can be kept as tables or mappings. This option is no longer possible when we allow for continuous state spaces. Moreover, policy improvement with an approximation of the state-value function becomes much less viable. Neither can we readily update the current policy to correspond to the latest approximation of the action-value function. However, having Q_π at least enables us to determine the policy for the current state, which is all we need to choose an action and continue learning.

The outlined concerns lead to the conclusion that the previously presented methods must be adjusted. This can be accomplished by capitalizing on differentiable parametric function-approximation techniques such as ANNs and only maintaining a parametric action-value-function approximation $\hat{q}_{\boldsymbol{w}}$ differentiable w.r.t. its vector of parameters $\boldsymbol{w} \in \mathbb{R}^d$, for $d \in \mathbb{N}^+$. Policy evaluation and improvement then merge into a single update of $\boldsymbol{w}$ minimizing the *mean squared action error*

$$\overline{QE}(\boldsymbol{w}) = \mathbb{E}_\pi \left[\left(q_\pi(S_t, A_t) - \hat{q}_{\boldsymbol{w}}(S_t, A_t) \right)^2 \right], \tag{4.17}$$

where π is the current policy induced by $\hat{q}_{\boldsymbol{w}}$.

Applying SGD transforms the original on-policy MC update to

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[G_t - \hat{q}_{\boldsymbol{w}}(S_t, A_t) \right] \nabla_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_t, A_t). \tag{4.18}$$

In the case of on-policy SARSA or on-policy expected SARSA, we come to the following respective *semi-gradient* updates:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[R_{t+1} + \gamma \hat{q}_{\boldsymbol{w}}(S_{t+1}, A_{t+1}) - \hat{q}_{\boldsymbol{w}}(S_t, A_t) \right] \nabla_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_t, A_t), \text{ or} \tag{4.19}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[R_{t+1} + \gamma \sum_{a \in \mathcal{A}(S_{t+1})} Q_\pi(S_{t+1}, a) - \hat{q}_{\boldsymbol{w}}(S_t, A_t) \right] \nabla_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_t, A_t). \tag{4.20}$$

We call the above two TD updates semi-gradient because, for simplicity, they neglect the contribution of the estimates of the next state-action values to the overall error.

For the later discussion on policy-based methods, we also consider having an approximation $\hat{v}_{\boldsymbol{w}}$ of the state-value function differentiable w.r.t. its parameter vector $\boldsymbol{w} \in \mathbb{R}^d$, for $d \in \mathbb{N}^+$. Our objective would be to minimize the *mean squared state error* $\overline{VE}$ analogous to $\overline{QE}$:

$$\overline{VE}(\boldsymbol{w}) = \mathbb{E}_\pi \left[\left(v_\pi(S_t) - \hat{v}_{\boldsymbol{w}}(S_t) \right)^2 \right], \tag{4.21}$$

where π is the current policy induced by $\hat{v}_{\boldsymbol{w}}$. Via gradient and semi-gradient optimization, respectively, we thus obtain the on-policy MC update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[G_t - \hat{v}_{\boldsymbol{w}}(S_t) \right] \nabla_{\boldsymbol{w}} \hat{v}_{\boldsymbol{w}}(S_t), \tag{4.22}$$

and the on-policy TD update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[R_{t+1} + \gamma \hat{v}_{\boldsymbol{w}}(S_{t+1}) - \hat{v}_{\boldsymbol{w}}(S_t) \right] \nabla_{\boldsymbol{w}} \hat{v}_{\boldsymbol{w}}(S_t). \tag{4.23}$$

We conclude by noting that parametric approximation provides a means to tackle continuous state spaces through self-organized abstraction. Therefore, combining this technique with HRL can fuse the merits of this emergent phenomenon and the designer-controlled behavior.

4.3.2 Policy-Based Methods

In this sub-section, we study policy-based methods, which search for an optimal policy without the intermediate stage of approximating value functions.

Let $\mathcal{M} = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, s_0, p, \gamma \rangle$ be an MDP for which we would like to find an optimal policy. The procedure policy-based methods follow is similar to generalized policy iteration in the sense that they still iteratively optimize an arbitrary initial policy. However, the policy, $\pi_{\boldsymbol{\theta}}$, is now represented by a function differentiable w.r.t. its vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, with $d' \in \mathbb{N}^+$, for instance, by an ANN. These parameters fulfill the role of value-function approximations in the value-based methods. At each optimization step, the parameters are thus updated through gradient ascent to maximize a certain *performance measure* J depending on $\boldsymbol{\theta}$ and expressed in terms of values:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{4.24}$$

where $\alpha \in (0, 1]$ is the learning rate. If $\mathcal{M}$ frames an episodic task, the performance measure is naturally defined as a value of a particular initial state $s_0^* \in \mathcal{S}$:

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0^*), \tag{4.25}$$

which gives the following gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\sum_{a \in \mathcal{A}(S_t)} q_{\pi_{\boldsymbol{\theta}}}(S_t, a) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a \mid S_t) \right]. \tag{4.26}$$

Continuous tasks require a re-definition of values via average rewards per time step. Since our research setting is episodic, we do not consider the continuous case in this discussion.

Employing policy-based solution techniques may bring various benefits. For example, they are agnostic to what probability distributions over the actions for the individual states the policy specifies. This freedom of choice empowers us to use a policy with distributions that smoothly converge to a deterministic optimum, e.g., an ANN with a soft-max output layer, instead of an ϵ-greedy policy that attains merely a soft optimum. In addition, policy-based methods can also handle continuous action spaces, simply by working with policies that define continuous probability distributions.[3]

[3] The summation in Equation (4.26) is then replaced with integration.

We now present a few notable policy-based techniques, which mainly differ in how they approximate the gradient from Equation (4.26) and perform the policy update.

REINFORCE

REINFORCE [Wil92] builds on the idea of value-based on-policy MC. Hence, it collects the entire trajectory $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$ of a sample episode under π. The sample returns G_t for each pair (S_t, A_t) from τ are then passed as estimates of $q_{\pi_\theta}(S_t, A_t)$ into Equation (4.26). This approximation results in the update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha G_t \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(A_t \mid S_t)}{\pi_{\boldsymbol{\theta}}(A_t \mid S_t)}. \tag{4.27}$$

Actor-Critic

The family of actor-critic (AC) methods borrows the core principles from value-based on-policy TD. The sample returns are again replaced with returns from partial trajectories and values of endpoints. To this end, these methods keep an approximation $\hat{v}_{\boldsymbol{w}}$ of the state-value function differentiable w.r.t. its parameter vector $\boldsymbol{w} \in \mathbb{R}^d$, for $d \in \mathbb{N}^+$.[4] One-step AC then updates the parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ after each transition, caused by an action $A_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid S_t)$ from the current state S_t to a new state S_{t+1} producing a reward R_{t+1}, where $(S_{t+1}, R_{t+1}) \sim p(\cdot, \cdot \mid S_t, A_t)$, as follows:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[R_{t+1} + \gamma \hat{v}_{\boldsymbol{w}}(S_{t+1}) - \hat{v}_{\boldsymbol{w}}(S_t)\right] \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(A_t \mid S_t)}{\pi_{\boldsymbol{\theta}}(A_t \mid S_t)}, \tag{4.28}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[R_{t+1} + \gamma \hat{v}_{\boldsymbol{w}}(S_{t+1}) - \hat{v}_{\boldsymbol{w}}(S_t)\right] \nabla_{\boldsymbol{w}} \hat{v}_{\boldsymbol{w}}(S_t). \tag{4.29}$$

Notice that the update of policy parameters involves comparing the estimate of G_t with the approximation of the current state's value, effectively transforming this term into the TD error. Such a comparison to virtually any state-dependent baseline can also be performed in REINFORCE, without hindering convergence. On the contrary, using reasonable baselines, like state values, often accelerates learning.

4.3.3 Neuro-Symbolic Approaches

Parametric function-approximation techniques, especially neuronal computational models, are key enablers to the more widespread use of RL due to their suitability for online, incremental optimization and ability to process continuous state and action spaces, characteristic of many real-world control tasks. However, these methods of sub-symbolic AI tend to require sampling large amounts of experience when dealing

[4] The state-value function approximation need not be parametric, but doing so leads to a more uniform approach.

with complex problems and provide little insight into the agent's decision-making governed by the learned parameters. Symbolic representations lend themselves to mitigate both issues by introducing high-level abstractions. The combination of sub-symbolic and symbolic methods thus gave rise to a new class of neuro-symbolic approaches to RL, state-of-the-art instances of which we present in this sub-section.

Programmatically Interpretable Reinforcement Learning

Programmatically Interpretable Reinforcement Learning (PIRL) [Ver+18] aims to eliminate the lack of transparency and understandability of purely ANN-based policies. To this end, PIRL produces policies that bear the form of programs in a functional programming language with an adequately designed set of valid syntactical constructs and their semantics. Such a program's execution fed with values for the individual features of the current state simply returns the action that should be taken.

These functional programs are synthesized from *sketches*, i.e., sets of parametric template structures that can be combined. An example member of a sketch could be the if-then-else template "**if** $x > c$ **then** t_1 **else** t_2", where x is a placeholder for a numerical state feature, c is a constant, and t_1, t_2 are other templates. Given a sketch and a neural policy trained to solve an RL problem using any conventional RL method, PIRL searches the space of all programs determined by the sketch for a programmatic policy that best mimics the neural one. Hence, PIRL operates ex-post because an already trained neural policy needs to be provided, from which the programmatic policy is constructed offline.

The absence of interaction between the sub-symbolic and symbolic layer means that PIRL cannot increase the efficiency of learning the underlying neural policy. However, this separation allows PIRL to interpret the reasoning of any neural agent.

Neural Logic Reinforcement Learning

Programmatic policies such as those in PIRL suffer from the fact that they have to be explicit in defining the agent's decision-making, which leads to either limited expressivity or vast and complicated program spaces to be searched. Logic programs offer a more succinct and elegant representation of this reasoning process. In particular, a logic program may be composed of such rules that a multi-step deduction over this logic program enriched with a symbolic description of the current state outputs the action to be taken. For instance, the rule "$a(X, Y) \leftarrow f_1(X), f_2(Y), aux(X, Y)$" could entail that "action a on objects X and Y should be taken if in the current state, X and Y have features f_1 and f_2, respectively, and are in auxiliary relation aux." Assigning weights to the rules further enables fuzzy deduction results and soft policies.

Neural Logic Reinforcement Learning (NLRL) [JL19] provides a method for search-

ing the space of logic programs in the above-presented form to find an optimal policy. This algorithm first generates a set of rules from the supplied templates under the specified restrictions on rule complexity to create an initial logic program with all rules having the maximal weight, 1. The initial logic program acting as a policy is then iteratively optimized by adjusting the rule weights through REINFORCE and a slightly modified ∂ILP procedure while the agent gathers experience.

Since NLRL learns a symbolic policy in a differentiable manner from the ground up, symbolic and sub-symbolic techniques are here truly intertwined. The structure of the agent's policy given by the predicates and the templates allowed thus directly affects the course of learning. This property of NLRL lets the designer use their background knowledge to aid the learning.

Neurally Guided Differentiable Logic Policies

One of the main disadvantages of NLRL is that it heavily relies on heuristics employed in the rule-generating phase. To alleviate this problem, Delfosse et al. [Del+23] propose a framework for Neurally Guided Differentiable Logic Policies (NUGDE) [Del+23] that incorporates the idea of neural guidance from PIRL into the NLRL scheme.

In the first step, NUDGE constructs the initial logic program from rules that are produced based on the provided templates and that most closely correspond to the behavior of the given, trained neural policy. The weights of the rules in this logic program are subsequently optimized via the AC RL method and ∂ILP.

NUDGE effectively addresses the issue of assembling a reasonable set of rules for the initial program in NLRL. The downside of NUDGE is that it requires, as PIRL, a neural policy to be trained in advance. However, unlike in PIRL, the neuro-symbolic learning process continues after the neural policy has been copied.

Reinforcement Learning with dNL Inductive Logic Programming

Payani and Fekri [PF20] take a fundamentally different approach to the rule-generation problem for logic-program policies with their dNL-ILP system [PF20].

For each action predicate, a dNL network is built that comprises one conjunctive layer followed by one disjunctive layer and receives state-feature and auxiliary atoms on input. Such a network may be seen as a formula in disjunctive normal form (DNF) or as a group of weighted rules for deriving a particular action within a logic program. Moreover, the weights in the conjunctive layer enable the agent to learn and pick which inputs are important for the respective rules online, which allows for bypassing the manual generation stage. The prepared set of dNL networks is thus directly trained through AC and gradient optimization for dNL-ILP.

Although this method neatly avoids any heuristics for rule synthesis, the designer

must define the auxiliary relations by hand or determine the skeletons thereof to be learned. The complexity of logical constructs that can be effectively induced from experience this way may be limited.

Symbolic Deep Reinforcement Learning

Despite their potential to concisely describe decision-making, logic programs only operate over discrete, finite domains of features and actions. One way to extend such policy representations to continuous state and action spaces is to adopt HRL frameworks like the meta-controller/controller model. Logics can then be used on the more abstract level of options and sub-symbolic function approximation may be employed to learn how to execute the individual options in continuous spaces. This principle is at the core of Symbolic Deep Reinforcement Learning (SDRL) [Lyu+19], which deals with episodic tasks.

For the meta-controller part, SDRL leverages traditional symbolic planning, with pre-conditions and effects of all options specified by a human expert. A planner is run at the beginning of each episode to find a sequence of options accumulating the highest average extrinsic reward per time step. Therefore, the values of options are defined in terms of average extrinsic rewards, and their estimates are learned via a value-based RL method applicable to this setting. The controller maintains an action-value ANN approximator and is trained with TD on intrinsic rewards to perform the options in the plan. During one episode, i.e., one plan execution, the controller tries to complete each option several times to gain enough experience to improve.

Lyu et al. [Lyu+19] have shown that SDRL is capable of solving even long-horizon tasks in continuous spaces, benefiting from high-level symbolic abstraction. The trained agent must, however, always plan their actions up front, which may easily get computationally intensive in complex environments. In addition, pre-loading the information on how options affect the agent-environment interaction and when they are available requires us to have prior knowledge of the environment's dynamics on the options level.

Hierarchical Reinforcement Learning using Inductive Logic Programming

Duo and Faramarz [DF21] propose another meta-controller/controller approach [DF21], mainly focusing on increasing sample efficiency.

The controller functions and is optimized on a similar basis as in SDRL. The meta-controller relies on a value-based RL technique to select and discover the most profitable options in the logic representation of the problem. Furthermore, it uses the observed transitions induced by the taken options to generate rules for an internal symbolic *transition model* and to update their weights with ∂ILP. This model enables the meta-controller to alternate between real experience and simulation while learning, reducing

the necessary interaction with the environment.

A shortcoming of this solution is that it employs only standard value-based optimization at the symbolic level. The transition model could be integrated into a more sophisticated inference procedure such as deduction over a logical program or symbolic planning.

Part II

Contribution

5 Aim

6 Approach

7 Experiments

8 Results

8.1 Symbolic Environment

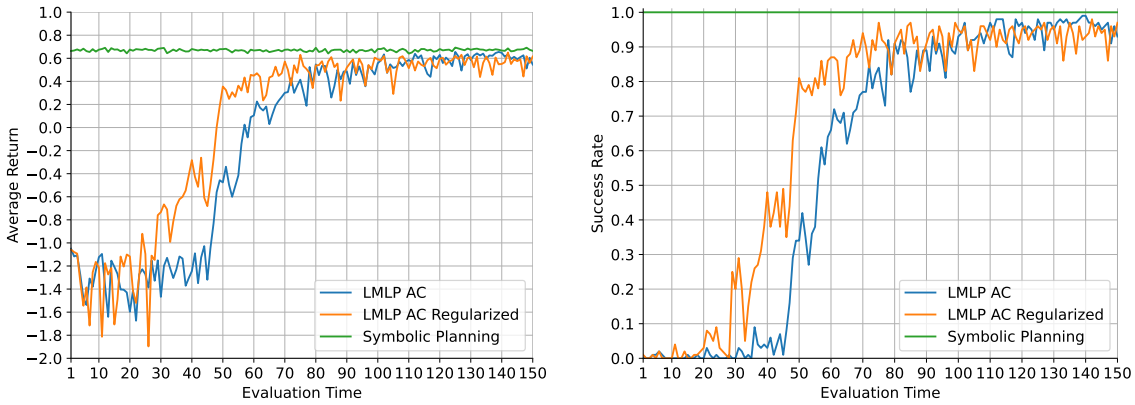
The final performance of the three considered solution methods is summarized in Table 8.1, which shows the *average* ± *standard deviation* of return and success rate over the 1 000 evaluation episodes. These results suggest that regularized and non-regularized versions of our LMLP AC agent are able to achieve similar average returns and success rates. We can thus assume that regularization does not significantly hinder convergence. However, their average returns and success rates are still lower than those of symbolic planning, which we aim to attain. We may also notice a higher variance in returns collected by LMLP AC agents which corresponds with the unsuccessful attempts to reach the desired goal state.

Table 8.1: Symbolic Environment: Final performance.

Method	Return	Success Rate
LMLP AC	0.571 ± 0.430	0.939 ± 0.239
LMLP AC Regularized	0.544 ± 0.486	0.920 ± 0.271
Symbolic Planning	0.681 ± 0.103	1.000 ± 0.000

To better compare all tested methods, we decided to track their learning progression by evaluating their current performance on a set of 100 episodes randomly generated after every 100 learning steps. Figure 8.1 depicts the average returns and success rates from this continual assessment. As expected, LMLP AC agents gradually converged to their best, final performance. We can also see that both experienced a critical period of sudden improvement. An interesting observation is that regularization helped the LMLP AC agent enter this major improvement stage earlier. We suppose that this phenomenon stems from more cautious weight updates the regularized agent makes, avoiding serious incorrect conclusions in the initial phases of learning.

In addition, the regularization managed to achieve its main objective – to reduce the complexity of the learned rules. Figure 8.2 proves this result for the final rules for action $move(b, a)$ learned by the two LMLP AC agents. These rules were obtained by



(a) Average return.

(b) Success rate.

Figure 8.1: Symbolic Environment: Average return and success rate progressions during learning.

dropping the input atoms with tanh-transformed weights less than 0.3. The weights of the remaining inputs are indicated in superscripts. The regularized agent effectively minimized the necessary inputs to the action's pre-conditions – $top(a)$ and $top(b)$ – and the fact that block a is in the right place – $on(a, table)$. Atom $\neg on(c, b)$ subsumed by $top(b)$ was left as an artifact.

$$\begin{aligned} move(b,a) \leftarrow\ & top(a)^{(0.920)}, \neg on(b,a)^{(0.563)}, \neg on(c,a)^{(0.998)}, \neg on(d,a)^{(0.934)}, \\ & top(b)^{(0.995)}, \neg on(a,b)^{(0.999)}, \neg on(c,b)^{(0.982)}, \neg on(d,b)^{(0.970)}, \\ & \neg on(a,c)^{(0.995)}, \neg on(d,c)^{(0.785)}, \\ & \neg on(a,d)^{(0.993)}, \neg on(b,d)^{(0.470)}, \\ & on(a,table)^{(0.998)}, on(d,table)^{(0.841)} \end{aligned}$$

(a) LMLP AC.

$$move(b,a) \leftarrow top(a)^{(1.000)}, top(b)^{(1.000)}, \neg on(c,b)^{(1.000)}, on(a,table)^{(1.000)}$$

(b) LMLP AC Regularized.

Figure 8.2: Symbolic Environment: Learned rules for action $move(b, a)$.

9 Discussion

10 Conclusions

Bibliography

[BM03] Barto, Andrew G. and Mahadevan, Sridhar. "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379. DOI: 10.1023/A:1025696116075.

[Del+23] Delfosse, Quentin et al. "Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Oh, A. et al. Vol. 36. Curran Associates, Inc., 2023, pp. 50838–50858. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/9f42f06a54ce3b709ad78d34c73e4363-Paper-Conference.pdf.

[DF21] Duo, Xu and Faramarz, Fekri. *Interpretable Model-based Hierarchical Reinforcement Learning using Inductive Logic Programming*. June 2021. arXiv: 2106.11417 [cs.LG].

[DH92] Dayan, Peter and Hinton, Geoffrey E. "Feudal Reinforcement Learning". In: *Proceedings of the 5th International Conference on Neural Information Processing Systems, NIPS'92, Denver, CO, USA, November 30 - December 3, 1992*. Ed. by Hanson, Stephen Jose, Cowan, Jack D., and Giles, C. Lee. Morgan Kaufmann, 1992, pp. 271–278. URL: https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

[Die00] Dietterich, T. G. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". In: *Journal of Artificial Intelligence Research* 13 (Nov. 2000), pp. 227–303. DOI: 10.1613/jair.639.

[Gla+22] Glanois, Claire et al. *A Survey on Interpretable Reinforcement Learning*. Feb. 2022. arXiv: 2112.13112 [cs.LG].

[HML22] Hutsebaut-Buysse, Matthias, Mets, Kevin, and Latré, Steven. "Hierarchical Reinforcement Learning: A Survey and Open Research Challenges". In:

Machine Learning and Knowledge Extraction 4.1 (2022), pp. 172–221. DOI: 10.3390/make4010009.

[JL19] Jiang, Zhengyao and Luo, Shan. "Neural Logic Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, CA, USA*. Ed. by Chaudhuri, Kamalika and Salakhutdinov, Ruslan. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3110–3119. URL: http://proceedings.mlr.press/v97/jiang19a.html.

[Kul+16] Kulkarni, Tejas D. et al. "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Barcelona, Spain,* ed. by Lee, Daniel D. et al. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 3682–3690. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/f442d33fa06832082290ad8544a8da27-Paper.pdf.

[Lyu+19] Lyu, Daoming et al. "SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 2970–2977. DOI: 10.1609/aaai.v33i01.33012970.

[PF20] Payani, Ali and Fekri, Faramarz. *Incorporating Relational Background Knowledge into Reinforcement Learning via Differentiable Inductive Logic Programming*. Mar. 2020. arXiv: 2003.10386 [cs.LG].

[Put94] Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. New York, NY, USA: John Wiley & Sons, 1994. ISBN: 9780471619772.

[Rie+20] Riegel, Ryan et al. *Logical Neural Networks*. June 2020. arXiv: 2006.13155 [cs.AI].

[SB18] Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. Second Edition. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 9780262039246.

[SPS99] Sutton, Richard S., Precup, Doina, and Singh, Satinder. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112.1 (1999), pp. 181–211. DOI: 10.1016/S0004-3702(99)00052-1.

[Ver+18] Verma, Abhinav et al. "Programmatically Interpretable Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Dy, Jennifer G. and Krause, Andreas. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5052–5061. URL: http://proceedings.mlr.press/v80/verma18a.html.

[Wil92] Williams, Ronald J. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. DOI: 10.1007/BF00992696.