TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Deep Reinforcement Learning with Applications to Autonomous Driving

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Helmut Horvath
Matrikelnummer: 01526425

ausgeführt am Institut für Analysis und Scientific Computing
der Fakultät für Mathematik und Geoinformation
der Technischen Universität Wien

Betreuer: Assoz. Prof. Dipl.-Ing. Dr. techn. Clemens Heitzinger

Wien, 19. März 2024

Helmut Horvath

Clemens Heitzinger

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit, einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. März 2024

Helmut Horvath

Kurzfassung

Deep Reinforcement Learning hat sich seit der Veröffentlichung des DQN Algorithmus rasant weiterentwickelt. In dieser Arbeit untersuchen wir die Evolution von DQN hin zu populären off-policy, modellfreien Algorithmen für kontinuierliche Kontrollprobleme, indem wir die theoretischen Grundlagen kohärent aufarbeiten, Lücken zwischen Theorie und praktischen Algorithmen diskutieren und anschließend die Algorithmen an eigens im CARLA Simulator implementierten Kontrollproblemen des autonomen Fahrens vergleichen.

Abstract

Deep Reinforcement Learning has seen rapid development ever since the publication of the DQN algorithm. In this work we study the evolution from DQN to popular off-policy, model free algorithms for continuous control by working out a coherent theoretical basis, discussing the theory-praxis gap of practical algorithms and comparing the algorithms on customly implemented autonomous driving tasks within the CARLA simulator.

Acknowledgements

First, I want to thank Prof. Heitzinger for his guidance and his enthusiasm for reinforcement learning and artificial intelligence in general.

I would particularly like to thank Tobias Kietreiber for being a great collaborator in the autonomous driving project and for proofreading this thesis.

I want to thank my family and friends for always supporting me throughout my studies.

Last but not least, I want to thank the amazing technical support teams of CLIP and VSC for always addressing technical issues promptly and supporting my special use cases.

Contents

Introduction **1**

1 Theoretical Framework **3**

1.1 Markov Decision Process 3
1.2 Optimality Criteria 5
1.3 Existence of Optimal Policies 6
1.4 Theoretical Foundations of Q-learning 9
1.5 Distributional Operators 10
1.6 Reinforcement Learning Terminology 13
1.7 Multi-Objective RL 14

2 Algorithms and Implementation **16**

2.1 Q-Learning 16
2.2 Q-Learning with Function Approximation 18
2.2.1 Semi-Gradient method 18
2.2.2 Residual Method 19
2.2.3 Notes on Function Approximation 19
2.3 Deep Q-Learning 20
2.4 Double DQN 22
2.5 Deep Deterministic Policy Gradient 23
2.6 Issues of Policy Gradient Methods 25
2.7 Twin Delayed DDPG 26
2.8 Soft Actor Critic 27
2.8.1 Theory 27
2.8.2 Implementation 31
2.9 Quantile Regression DQN 33
2.9.1 Quantile Dynamic Programming 33
2.9.2 Quantile Q-learning 34
2.9.3 QR-DQN 36
2.10 Implicit Quantile Networks 36
2.11 Truncated Quantile SAC 39
2.11.1 Distributional SAC 39
2.11.2 TQC 39
2.12 Neural Network Architecture 40
2.12.1 Critic-only Networks 40
2.12.2 Actor-Critic Networks 42
2.12.3 Concrete Architecture 42

3 Environment for Autonomous Driving **44**
3.1 Problem Formulation and Scope . . . 44
3.2 Environment Setup . . . 45
3.2.1 Time Steps and Synchrony . . . 45
3.2.2 Episode Setup . . . 45
3.2.3 Terminal States and Truncation . . . 47
3.2.4 Action Space Design . . . 48
3.3 Reward Design . . . 48
3.3.1 Cruise Control Reward . . . 48
3.3.2 Adaptive Cruise Control Reward . . . 49
3.3.3 Lane Keeping Reward . . . 50
3.3.4 Combined Control Reward . . . 51
3.3.5 Highway Obstacle Avoidance Reward . . . 51
3.3.6 Combining Multiple Objectives . . . 52
3.4 State Space Design . . . 52
3.4.1 Vectorized Observations . . . 53
3.4.2 Camera Image Observations . . . 54
3.4.3 Preprocessing . . . 56
3.5 Implementation Details . . . 56
3.5.1 Simulation Setup . . . 56
3.5.2 Gymnasium Interface . . . 57
3.5.3 Data Retrieval . . . 58
3.5.4 Determinism . . . 59
3.5.5 Error Recovery . . . 59

4 Numerical Results **60**
4.1 Methods . . . 60
4.1.1 Data Generation . . . 60
4.1.2 Performance Metrics . . . 61
4.1.3 Statistical Significance . . . 61
4.2 Algorithm Comparison across Environments . . . 62
4.2.1 Cruise Control . . . 62
4.2.2 Adaptive Cruise Control . . . 62
4.2.3 Lane Keeping . . . 63
4.2.4 Lane Keeping from Pixels . . . 64
4.2.5 Combined Control . . . 65
4.2.6 Combined Control from Pixels . . . 66
4.2.7 Adaptive Combined Control . . . 67
4.2.8 Adaptive Combined Control from Pixels . . . 68
4.2.9 Obstacle Avoidance . . . 68
4.2.10 Navigation . . . 69
4.2.11 Adaptive Navigation . . . 71
4.3 Summary . . . 72
4.4 Own Publication . . . 73

4.5 Hyperparameters 74

Bibliography **77**

Introduction

This thesis originated from a project practicum in reinforcement learning that had the goal to use reinforcement learning to formulate and solve autonomous driving problems in a simulator (CARLA).

In Chapter 1, we formulate the underlying theory for the algorithms introduced in Chapter 2 as well as for the reinforcement learning problem formulation of the autonomous driving tasks in Chapter 3, particularly the reward design. Contrary to most of the reinforcement learning literature, we formulate the theory for continuous state and action spaces, motivated by the continuous nature of our control problems and adjust the model assumptions to serve as a coherent basis for the formulation of our autonomous driving tasks.

In Chapter 2, we introduce popular state-of-the-art deep reinforcement learning algorithms that turned out to be successful for solving our autonomous driving tasks. Although theoretical convergence guarantees for deep reinforcement learning algorithms are still a topic of ongoing research, our aim in this chapter is to give a clear motivation for the design choices of the algorithms, draw parallels between them and discuss theoretical issues. Our focus is on off-policy model free algorithms that fit into a theoretical Q-iteration framework and their distributional extensions.

Chapter 3 is devoted to the formulation of practically motivated autonomous driving tasks as reinforcement learning problems, especially the design of the state space and the reward functions which turned out to be vital for successful results.

Finally, in Chapter 4, we empirically compare the algorithms introduced in Chapter 2 for our own autonomous driving problems.

1 Theoretical Framework

1.1 Markov Decision Process

We base our definition mostly on [Put05, Sec. 2.3.2] and [Fei11, Sec. 3], however we allow the reward function to also depend on the successor state. The space of all probability measures over a measure space (X, Σ) is denoted by $\mathcal{P}(X)$.

Definition 1.1.1. A *Markov Decision Process (MDP)* is given by a tuple $(T, \mathcal{S}, \rho_0, \mathcal{A}, p, r)$, where

1. T is a set of *(time) steps.*
2. $\mathcal{S} = (S, \mathcal{B}(S))$ is a standard Borel space on a set of *states* S with σ-algebra $\mathcal{B}(S)$.
3. ρ_0 is a probability measure on $\mathcal{B}(S)$, the *initial state distribution.*
4. $\mathcal{A} = (A_s)_{s \in S}$ is family of *action sets*, where $A_s \in \mathcal{B}(A)$ and $(A, \mathcal{B}(A))$ is a standard Borel space, such that there exists a measurable function $\mu : S \to A$ with $\mu(s) \in A_s$. Let $G := \{(s, a) \mid s \in S, a \in A_s\}$.
5. $p : G \to \mathcal{P}(S),\ (s, a) \mapsto p(\cdot \mid s, a)$ is a *transition kernel* such that $p(X \mid \cdot, \cdot)$ is measurable with respect to $\mathcal{B}(G)$ for all $X \in \mathcal{B}(S)$.
6. $r : G \times S \to \mathbb{R}$ is a *reward function* satisfying
   a) $r(\cdot, \cdot, \cdot)$ is measurable with respect to $\mathcal{B}(G \times S)$.
   b) $r(s, \cdot, s')$ is integrable with respect to all $q \in \mathcal{P}(A_s)$ for all $s, s' \in S$.

A state is $s \in S$ is called *absorbing* if $p(\{s\} \mid s, a) = 1$ for all $a \in A_s$.

We only consider *discrete-time* MDPs, i.e., $T \subseteq \mathbb{N}$ with the following additional requirements: r is bounded and $r(s, a, s) = 0$ for all $a \in A$ for all absorbing states $s \in S$. Furthermore, in this work we assume $A_s = A$ for all $s \in S$ and we identify $\mathcal{A}$ as $(A, \mathcal{B}(A))$.

Definition 1.1.2. Let $H_n := (G)^n \times S$ denote the set of *histories* up to time n and let $H := \bigcup_{n=1}^{\infty} H_n$. For $h_n \in H_n$ let $h_n^n \in S$ denote the last element. A *policy* π is a mapping

$$\pi : H \to \bigcup_{s \in S} \mathcal{P}(A_s),\ h_n \mapsto \pi(\cdot, h_n)$$

such that $\pi(\cdot, h_n) \in \mathcal{P}(A_s)$ with $s = h_n^n$ and such that $h_n \mapsto \pi(Y, h_n)$ is measurable on H_n for every $Y \in \mathcal{B}(A_s)$. A policy is called *deterministic* if for each $h_n \in H$ the probability measure $\pi(\cdot, h_n)$ is concentrated at one point $a \in A_s$, otherwise it is called *stochastic*. We call a policy π *stationary* if $\operatorname{dom} \pi = S$. The set of all policies of an MDP is denoted by Π, the set of all stationary policies by Π_s and the set of all stationary, deterministic policies by Π_d.

The following theorem has been adapted to our continuous formulation from [LS20, p. 514, Sec. Probability Spaces] and [Put05, p. 30] and ensures the existence of a probability measure for infinitely long state-action sequences.

Theorem 1.1.3 (existence theorem)**.** *Let (T, $\mathcal{S}$, ρ_0, $\mathcal{A}$, p, r) be an MDP and let* (Ω, Σ) *be the* canonical probability space of the MDP*, where* $\Omega := G^\infty$ *is the Cartesian product endowed with the product* σ*-algebra* $\Sigma := \bigotimes \mathcal{B}(G)$*. For* $(s_1, a_1, s_2, a_2, \dots) =: \omega \in \Omega$ *we define random variables*

$$S_t(\omega) := s_t, \quad A_t(\omega) := a_t.$$

Then, for any policy π *and any measure* $\mu \in \mathcal{P}(S)$ *there exists a unique probability measure* $\mathbb{P}^\pi_\mu$ *over* (Ω, Σ) *satisfying*

1. $\mathbb{P}^\pi_\mu(S_0 \in X) = \mu(X)$ *for all* $X \in \mathcal{B}(S)$.
2. $\mathbb{P}^\pi_\mu(A_{t+1} \in A \mid S_0, A_0, \dots, S_t, A_t) = \pi(A \mid S_0, A_0, \dots, S_t, A_t)$ *for all* $A \in \mathcal{B}(A)$.
3. $\mathbb{P}^\pi_\mu(S_{t+1} \in X' \mid S_0, A_0, \dots, S_t, A_t) = p(X' \mid S_t, A_t)$ *for all* $X' \in \mathcal{B}(S)$.

Proof. The theorem directly follows from the Ionescu-Tulcea theorem. $\square$

Property 3 is known as *Markov property*, i.e., the only part in the history of the stochastic process that matters for the probability of the next state is the previous state and action.

Definition 1.1.4. In the above theorem, we call Ω *trajectory space* or *sample space* and $\omega \in \Omega$ a *trajectory* or *sample path*. Likewise, we also refer to the sequence of random variables $S_0, A_0, \dots$ as the trajectory.

Definition 1.1.5. An MDP is said to have an *indefinite horizon*, if for any policy π and starting state distribution μ

$$\mathbb{P}^\pi_\mu(\{\omega \in \Omega : \exists n\ \exists a \in S\ \forall t \geq n : \omega_t = a,\ a \text{ is absorbing}\}) = 1.$$

Otherwise, the MDP is said to have an *infinite horizon*. A trajectory ending in an absorbing state is called *episode*.

In the reinforcement learning literature, tasks modelled by indefinite horizon MDPs are called *episodic*, while infinite horizon tasks are called *continuing* [SB18, p. 70], absorbing states are usually called *terminal*. As soon as the agent environment interaction modelled by the MDP ends in a terminal state, the agent starts in a new initial state. We formulated all of our definitions from the beginning to give a unified perspective on episodic and continuing tasks as described in [SB18, Sec. 3.4].

1.2 Optimality Criteria

Definition 1.2.1. Let $\gamma \in [0,1)$ be a *discount factor* and $s \in S$. For a trajectory starting in $S_0 = s$ with $s \in S$ and $R_i := r(S_{i-1}, A_{i-1}, S_i)$ for $i \geq 1$ we define the random variable

$$G^\pi_\gamma(s) := \sum_{k=0}^{\infty} \gamma^k R_k.$$

For a trajectory with $S_0 = s$ and $A_0 = a$ with $s \in S$, $a \in A$ we define the random variable

$$Z^\pi_\gamma(s,a) := \sum_{k=0}^{\infty} \gamma^k R_k.$$

Both random variables are referred to as *return.*

Remark 1.2.2. It is common in the MDP literature to define $R_i := r(S_{i-1}, A_{i-1})$, however we consider it practical to give the reward as feedback depending on the outcome of the action, thus we allow dependence on the successor state.

Both random variables in Definition 1.2.1 are well-defined in case of a bounded reward function with $\forall s \in S\ \forall a \in A : r_l \leq r(s,a) \leq r_u$, as for every ω with $S_0(\omega) = s$ we have

$$r_l \cdot \frac{1}{1-\gamma} = \sum_{k=0}^{\infty} \gamma^k r_l \leq G^\pi_\gamma(s)(\omega) \leq \sum_{k=0}^{\infty} \gamma^k r_u = r_u \cdot \frac{1}{1-\gamma}$$

and analogously for Z^π. With regard to Theorem 1.1.3 we have $\mu = \delta_s$, where δ_s is the Dirac measure concentrated on the point s.

Definition 1.2.3. We write $\Omega_s := \{\omega' \mid (s,\omega') \in \Omega\}$ and $\Omega_{s,a} := \{\omega' \mid (s,a,\omega') \in \Omega\}$. We equip those sets with the respective Borel σ-algebras and the respective marginal measure of $\mathbb{P}^\pi_{\delta_s}$ (for $\Omega_{s,a}$ we require $\pi(\cdot|s) = \delta_a$) and denote those by $\mathbb{P}^\pi_s$ and $\mathbb{P}^\pi_{s,a}$.

Definition 1.2.4. The *value function* of a policy is defined as

$$v^\pi_\gamma : S \to \mathbb{R},\ s \mapsto \mathbb{E}\left[G^\pi_\gamma(s)\right],$$

the *action value function* of a policy is defined as

$$q^\pi_\gamma : S \times A \to \mathbb{R},\ (s,a) \mapsto \mathbb{E}\left[Z^\pi_\gamma(s,a)\right].$$

The value of a state $s \in S$ thus is the expected total discounted reward of a trajectory starting in s.

Definition 1.2.5. For a given $\gamma \in [0,1)$ a policy π^* is said to be *uniformly optimal* if

$$v^{\pi^*}_\gamma(s) \geq v^\pi_\gamma(s) \quad \forall \pi \in \Pi\ \forall s \in S.$$

The *optimal value function* is defined as $v^*_\gamma(s) := \sup_{\pi \in \Pi} v^\pi_\gamma(s)$. A policy π is said to be ε*-optimal* if $v^\pi(s) \geq v^\pi(s) - \varepsilon$ for all $s \in S$.

Definition 1.2.6. A policy π^* is said to be *Blackwell-optimal* if there exists a $\gamma \in [0, 1)$ such that π^* is uniformly optimal for every $\gamma' \in [\gamma, 1)$.

Definition 1.2.7. For $\gamma \in [0, 1)$, we call a policy π *weakly-optimal* if it maximizes the objective function

$$J_\gamma(\pi) := \mathbb{E}_{s\sim\rho_0} v_\gamma^\pi(s)$$

where ρ_0 is the initial state distribution of the MDP.

Since a policy π' is uniformly better than π if $v^{\pi'}(s) \geq v^\pi(s)\ \forall s \in S$ the induced order relation is only partial, while the weaker notion $J(\pi') \geq J(\pi)$ defines a total order.

1.3 Existence of Optimal Policies

The formulation of definitions and results in this section is inspired by the treatment of [Lig24], who abstracted various results of prior literature, however we adapt the treatment to our setting. In particular the reward function is dependent on s' as well, and we treat the case of stochastic policies as well.

Given Borel spaces $(X, \mathcal{B}(X))$ and $(Y, \mathcal{B}(Y))$ we denote the class of all measurable functions from X to Y by $U(X, Y)$. For any set S we denote the space of real valued bounded functions by $B(S)$. Furthermore, we denote the integral of a function f with regard to a measure μ as $\int f(x)\ \mu(dx)$.

Definition 1.3.1. Let S be the state set, A_s the action set of a state $s \in S$, r the reward function and p the transition kernel of an MDP. For $v \in U(S, \mathbb{R}) \cap B(S)$ the *Bellman optimality operator* is defined as

$$Tv(s) := \sup_{a\in A_s} \int_S r(s, a, s') + \gamma v(s')\ p(ds', s, a)$$

and for $\pi \in \Pi_s$ the *Bellman Operator*

$$T_\pi v(s) := \int_{A_s} \int_S r(s, a, s') + \gamma v(s')\ p(ds', s, a)\pi(da, s).$$

By T_a we denote the operator T_π with the Dirac measure $\pi(\cdot \mid s) = \delta_a$.

Lemma 1.3.2. *The operators T_π and T are both γ-contractive regarding the supremum norm $\|.\|$ on $B(S)$.*

Proof. For any $s \in S$ via the triangular inequality we have

$$|T_\pi f(s) - T_\pi g(s)| \leq \gamma \int_{A_s} \int_S |f(s') - g(s')|\ p(ds', s, a)\pi(s, da) \leq \gamma\|f - g\|. \qquad (1.1)$$

Thus, via the supremum over $s \in S$ we arrive at $\|T_\pi f - T_\pi g\| \leq \gamma\|f - g\|$. By Eq. (1.1) it follows that for all $s \in S$, any $f, g \in B(S)$ and $a \in A_s$ the inequalities

$$T_a f(s) \leq T_a g(s) + \gamma\|f - g\| \leq Tg(s) + \gamma\|f - g\|$$

hold. Via the supremum over $a \in A_s$ we arrive at $Tf(s) \leq Tg(s) + \gamma\|f - g\|$. The same argument can be repeated with the roles of f and g exchanged, thus a symmetry argument yields $|Tf(s) - Tg(s)| \leq \gamma\|f - g\|$ and taking the supremum does the rest. $\square$

The following lemma is common knowledge from mathematical analysis, a proof can also be found in [BDR23, Proposition 4.7]. Compared to Banach's fixed point theorem, completeness of the underlying space is not required.

Lemma 1.3.3. *Let (M, d) be a metric space and $T : M \to M$ be a γ-contractive mapping for $\gamma \in [0, 1)$ with fixed point u^*. Then u^* is the unique solution of $Tu = u$ and for any $u_0 \in M$ the sequence $(u_k)_{k\geq 0}$ defined by $u_{k+1} = Tu_k$ converges to u^* with respect to d, i.e., $\lim_{k\to\infty} d(u_k, u^*) = 0$.*

Theorem 1.3.4. *Let $D \subseteq U(S, \mathbb{R}) \cap B(S)$ be Bellman closed i.e., $T_\pi f \in D$ for $\pi \in \Pi_s$ and every $f \in D$. Then the value function v^π with respect to policy $\pi \in \Pi_s$ is the unique function in D that satisfies*

$$v^\pi = T_\pi v^\pi,$$

notably for every $f \in D$ we have $v^\pi = \lim_{k\to\infty} T_\pi^k f$.

Proof. We start by showing the equation

$$v^\pi(s) = \int_\Omega G(s)(\omega)\ \mathbb{P}_s(d\omega)$$

for v^π. With Fubini's theorem for the product measure we obtain:

$$\begin{aligned} &= \int_{A_s}\int_S\int_{\Omega_{s,a}} r(s,a,s') + \gamma G(s')(\omega')\ \mathbb{P}_{s,a}(d\omega')p(ds',s,a)\pi(da,s) \\ &= \int_{A_s}\int_S r(s,a,s') + \gamma v^\pi(s')\ p(ds',s,a)\pi(da,s) = T_\pi v(s). \end{aligned}$$

We have now shown that v^π is a fixed point of T_π. Lemma 1.3.3 concludes the proof. $\square$

Theorem 1.3.5. *Let $D \subseteq U(S, \mathbb{R}) \cap B(S)$ be a complete metric space with respect to the supremum norm such that the constant function $y(x) := \frac{\sup_{s,s' \in S, a \in A} r(s,a,s')}{1-\gamma} \in D$ and for every $f \in D$ we have $Tf \in D$. If for every $f \in D$ there exists $\lambda \in \Pi_d$ such that $Tf(s) = T_\lambda f(s)$ for every $s \in S$, then:*

1. *The optimal value function v^* is the unique solution of $v = Tv$.*
2. *There exists a stationary deterministic policy π such that $v^* = v^\pi$.*

Proof. By Banach's fixed point theorem the operator T has a unique fixed point x. We start by showing $v^* \leq x$. Since $v^* \leq y$ and $y \in D$, by Theorem 1.3.4 we have

$$v^\pi(s) = \int_{A_s}\int_S r(s,a,s') + \gamma v(s')\ p(ds',s,a)\pi(da,s)$$

$$\leq \sup_{a \in A_s} \int_S r(s,a,s') + \gamma v^*(s')\; p(ds', s, a)$$
$$\leq \sup_{a \in A_s} \int_S r(s,a,s') + \gamma y(s')\; p(ds', s, a) = Ty(s)$$

for any policy π. Taking the supremum over all $\pi \in \Pi$ yields $v^* \leq Ty$. By induction, it follows that $v^* \leq T^n y$ and since $T^n y \to x$ as $n \to \infty$ by Banach's theorem we arrive at $v^* \leq x$.

It remains to show that $x \leq v^*$. Per assumption there exists a $\lambda \in U(S, A) \cap G$ such that $T_\lambda x = Tx$. Since $Tx = x$ we have $T_\lambda x = x$ and thus $x = v^\lambda$ by Theorem 1.3.4. Thus, we arrive at

$$v^* \leq x \leq v^\lambda \leq v^*$$

and thus $v^*(s) = x(s) = v^\lambda(s)$ for all $s \in S$. □

Definition 1.3.6. The equation $v = T_\pi v$ is called *Bellman equation*, the equation $v = Tv$ is called *Bellman optimality equation.*

There are various choices of D studied in the literature, depending on the restrictions imposed on the MDP. For an overview see [Lig24] and [Fei11]. The choice $D = U(S, \mathbb{R}) \cap B(S)$ does not work in general as there are $f \in D$ such that $Tf \notin D$, see [Lig24, p. 8].

Theorem 1.3.7. *Let (T, $\mathcal{S}$, ρ_0, $\mathcal{A}$, p, r) be an MDP with the assumptions:*

1. *A_s is compact for all $s \in S$ and $s \mapsto A_s$ is upper semi-continuous.*
2. *$(s, a) \mapsto p(\cdot|s, a)$ is weakly continuous, i.e., if $s_n \to s$ and $a_n \to a$, then $p(\cdot|s_n, a_n)$ converges weakly to $q(\cdot|s, a)$.*
3. *$r(s, a, s')$ is a bounded upper semi-continuous function on $S \times A \times S$.*

Then there exists a stationary deterministic optimal policy.

Proof. Let D be the class of all bounded upper semi-continuous functions on S, which is a closed subset of the complete metric space $B(S)$ and thus complete. It can be shown that under the conditions imposed on the MDP $v^* \in D$, $Tf \in D$ for all $f \in D$ and that there exists $\lambda \in \Pi_d$ such that $Tf(s) = T_\lambda f(s)$ for every $s \in S$. For the latter we refer to the proof of the selection theorem in [Mai68]. Thus, Theorem 1.3.5 can be invoked. □

Even these compactness and continuity conditions do not hold it can still be shown, that ε-optimal policies exist [Fei11, Theorem 20]:

Theorem 1.3.8. *If the class of policies is extended to the class of universally measurable functions, then for any $\varepsilon > 0$ there exists a stationary, deterministic universally measurable ε-optimal policy.*

Theorem 1.3.9. *In an MDP with finite state and action spaces there exists a stationary deterministic optimal policy.*

Proof. Choose $D = B(S)$ and notice that all the requirements of Theorem 1.3.5 are met. □

1.4 Theoretical Foundations of Q-learning

The results in this section are adaptions of the results in Section 1.3 for the action value function to serve as a theoretical basis for Q-learning algorithms.

Definition 1.4.1. For $q \in U(G, \mathbb{R}) \cap B(G)$ the *Bellman optimality operator* is

$$Tq(s,a) := \int_S r(s,a,s') + \gamma \sup_{a \in A_{s'}} q(s',a')\ p(ds', s, a)$$

and for $\pi \in \Pi_s$ the *Bellman operator* is defined as

$$T_\pi q(s,a) := \int_S r(s,a,s') + \gamma \int_{A_{s'}} q(s',a')\ \pi(da', s') p(ds', s, a).$$

Note that the operators carry the same name and symbol as in the last section, but they operate on different spaces (in the last section value functions and in this section action value functions). The equation $q = Tq$ with $q \in B(G)$ is also referred to as *Bellman optimality equation* (for action value functions) as opposed the Bellman optimality equation for value functions in the last section.

Lemma 1.4.2. *The operators T_π and T are both γ-contractive regarding the supremum norm $\|.\|$ on $B(G)$.*

Proof. For T_π the proof is analogous to Lemma 1.3.2. For T we have

$$\begin{aligned} |Tq_1(s,a) - Tq_2(s,a)| &\leq \int_S \gamma |\sup_{a' \in \Gamma(s')} q_1(s',a') - \sup_{a' \in \Gamma(s')} q_2(s',a')|\ p(ds', s, a) \\ &\leq \int_S \gamma \sup_{a' \in \Gamma(s')} |q_1(s',a') - q_2(s',a')|\ p(ds', s, a) \\ &\leq \gamma \sup_{(s',a') \in G} |q_1(s',a') - q_2(s',a')| \end{aligned}$$

and the supremum norm again concludes the proof. □

Theorem 1.4.3. *Let $D \subseteq U(G, \mathbb{R}) \cap B(G)$ be Bellman closed i.e., $T_\pi f \in D$ for $\pi \in \Pi_s$ and every $f \in D$. Then the value function q^π with respect to policy $\pi \in \Pi_s$ is the unique function in D that satisfies*

$$q^\pi = T_\pi q^\pi,$$

notably for every $f \in D$ we have $q^\pi = \lim_{k \to \infty} T_\pi^k f$.

Proof. Analogous to the proof of Theorem 1.3.4. □

Theorem 1.4.4. *The Bellman optimality operator for q-functions admits a unique fixed point $q^* \in B(G)$ with*

$$v^*(s) = \sup_{a \in A_s} q^*(s,a)$$

where v^* is the optimal value function.

Proof. Since $B(G)$ is a complete metric space with the supremum norm, Banach's fixed point theorem guarantees the existence of a unique fixed point q^*, with $q^* = Tq^*$. Taking the supremum on both sides of the fixed point equation yields

$$\sup_{a\in A_s} q^*(s,a) = \sup_{a\in A_s} \int_S r(s,a,s') + \gamma \sup_{a\in A_{s'}} q^*(s',a')\ p(ds',s,a).$$

Now let $v(s) := \sup_{a\in A_s} q^*(s,a)$. Then we get

$$v(s) = \sup_{a\in A_s} \int_S r(s,a,s') + \gamma v(s')\ p(ds',s,a),$$

which is the Bellman optimality equation for v^*, hence $v^*(s) = v(s)$. □

Corollary 1.4.5. *If an optimal policy π^* exists, then $q^*(s,a) = q^{\pi^*}(s,a)$.*

Proof. Let π^* be an optimal policy, from the definition of q^{π^*} we obtain

$$q^{\pi^*}(s,a) = \int_S r(s,a,s') + \gamma v^{\pi^*}(s')\ p(ds',s,a)$$

and since π^* is an optimal policy we have $v^{\pi^*}(s) = v^*(s) = \sup_{a\in A_s} q^*(s,a)$, thus

$$\begin{aligned} &= \int_S r(s,a,s') + \gamma \sup_{a\in A_{s'}} q^*(s',a')\ p(ds',s,a) \\ &= q^*(s,a) \end{aligned}$$

where the last equality is due to $Tq^* = q^*$. □

1.5 Distributional Operators

The theory of this section is based on [BDR23, Chap. 4, 7], albeit we adapt the formulation to our MDP definition (which assumes a deterministic reward function that depends on the next state, which is different from the assumption in [BDR23, p. 15]) and remain in the continuous state and action space formulation.

Let $\eta(s)$ be the distribution of $G(s)$ and $\nu(s,a)$ the distribution of $Z(s,a)$, i.e., $\eta(s) = G(s)^{\#}\mathbb{P}_s$ and $\nu(s,a) = Z(s,a)^{\#}\mathbb{P}_{s,a}$ where $f^{\#}\mu$ denotes the push-forward measure of μ through a measurable function f.

Then $v(s) = \int_{\mathbb{R}} x\ \eta(s)(dx)$ is the *induced value function* and $q(s,a) = \int_{\mathbb{R}} x\ \nu(s,a)(dx)$ the *induced action value function.*

Definition 1.5.1. Let $\mathcal{P}(\mathbb{R})^S := \{\eta(s) : \eta(s) \in \mathcal{P}(\mathbb{R}), s \in S\}$ denote the space of *return-distribution functions* and similarly $\mathcal{P}(\mathbb{R})^G := \{\nu(s,a) : \eta(s,a) \in \mathcal{P}(\mathbb{R}), (s,a) \in G\}$.

Definition 1.5.2. For $\eta \in \mathcal{P}(\mathbb{R})^S$ and a stationary policy π we define the *distributional Bellman operator*

$$(\mathcal{T}_\pi \eta)(s)(B) := \int_{A_s} \int_S b^{\#}_{r(s,a,s'),\gamma} \eta(s')(B)\ p(ds',s,a)\pi(da,s),$$

where $b_{r,\gamma} : \mathbb{R} \to \mathbb{R},\ z \mapsto r + \gamma \cdot z$ and $B \in \mathcal{B}(\mathbb{R})$. Similarly, for $\nu \in \mathcal{P}(\mathbb{R})^G$ we define

$$(\mathcal{T}_\pi \nu)(s,a)(B) := \int_S \int_{A_{s'}} b^{\#}_{r(s,a,s'),\gamma} \nu(s',a')(B)\ \pi(da',s') p(ds',s,a).$$

Definition 1.5.3. A *greedy selection rule* is a function $\mathcal{P}(\mathbb{R})^G \to \Pi_s,\ \nu \mapsto \mathcal{G}(\nu)$ with

$$\mathcal{G}(\nu)(a,s) > 0 \Rightarrow Q(s,a) = \sup_{a' \in A_s} Q(s,a')$$

for the induced action value function.

Definition 1.5.4. For $\nu \in \mathcal{P}(\mathbb{R})^G$ we define the *distributional Bellman optimality operator* with greedy selection rule $\mathcal{G}$ as

$$(\mathcal{T}_{\mathcal{G}} \nu)(s,a)(B) := \int_S \int_{A_{s'}} b^{\#}_{r(s,a,s'),\gamma} \nu(s',a')(B)\ \mathcal{G}(\nu)(s')(da',s') p(ds',s,a).$$

Theorem 1.5.5 (Distributional Bellman equation)**.** *Let $\nu^\pi \in \mathcal{P}(\mathbb{R})^G$ be the return-distribution function of a stationary policy π. For any $(s,a) \in G$, the equality*

$$\nu^\pi(s,a) = (\mathcal{T}_\pi \nu^\pi)(s,a)$$

holds. An analogous result holds for $\eta \in \mathcal{P}(\mathbb{R})^S$ with the respective operator.

Proof. For $B \in \mathcal{B}(\mathbb{R})$ we have

$$\begin{aligned} \nu^\pi(s,a)(B) &= \mathbb{P}_{s,a}(Z^\pi(s,a) \in B) \\ &= \mathbb{P}_{s,a}(\{\omega \in \Omega \mid Z^\pi(s,a)(\omega) \in B\}). \end{aligned}$$

By the property of the product measure $\mathbb{P}_{s,a}$ we have

$$\begin{aligned} &= \int_S \int_{A_{s'}} \mathbb{P}_{s',a'}(\{\omega' \in \Omega' \mid b_{r(s,a,s'),\gamma} \circ Z(s',a')(\omega') \in B\})\ \pi(da',s')p(ds',s,a) \\ &= \int_S \int_{A_{s'}} \mathbb{P}_{s',a'}(Z(s',a')^{-1} \circ b^{-1}_{r(s,a,s'),\gamma}(B))\ \pi(da',s')p(ds',s,a) \\ &= \int_S \int_{A_{s'}} \nu(s',a')(b^{-1}_{r(s,a,s'),\gamma}(B))\ \pi(da',s')p(ds',s,a) \\ &= \int_S \int_{A_{s'}} b^{\#}_{r(s,a,s'),\gamma} \nu(s',a')(B)\ \pi(da',s')p(ds',s,a) = (\mathcal{T}_\pi \nu^\pi)(s,a)(B). \qquad \square \end{aligned}$$

Lemma 1.5.6. *Let $Z(s,a) \sim \nu(s,a)$ for all $(s,a) \in G$ be a collection of random variables on $\Omega_{s,a}$. Then the random variable $r(s,a,X') + \gamma Z(X',A')$ has the distribution $(\mathcal{T}_\pi \nu)(s,a)$.*

Proof. Let μ be a measure on $\Omega_{s,a}$ such that $\nu(s,a) = Z(s,a)^{\#}\mu$. Since

$$\begin{aligned} &\mathbb{P}(r(s,a,X') + Z(X',A') \in B) \\ &= \int_S \int_{A_{s'}} \mu(\{\omega' \in \Omega_{s,a} \mid r(s,a,s') + \gamma Z(s',a')(\omega') \in B\})\ \pi(da',s')p(ds',s,a), \end{aligned}$$

we arrive at the desired result by analogous manipulation as in the preceding proof. $\square$

Definition 1.5.7. The *p-Wasserstein distance* of $\nu \in \mathcal{P}(\mathbb{R})$ and $\nu' \in \mathcal{P}(\mathbb{R})$ is given by

$$w_p(\nu, \nu') := \left(\int_0^1 |F_\nu^{-1}(\tau) - F_{\nu'}^{-1}(\tau)|^p \, d\tau \right)^{\frac{1}{p}}$$

for $p \in [1, \infty)$ and by

$$w_\infty(\nu, \nu') = \sup_{\omega \in (0,1)} |F_\nu^{-1}(\tau) - F_{\nu'}^{-1}(\tau)|$$

for $p = \infty$, where $F_\nu^{-1}(\tau) = \inf_{z \in \mathbb{R}}\{z : F_\nu(z) \geq \tau\}$ is the generalized inverse of the cumulative distribution function F_ν of ν.

Definition 1.5.8. Let $X = S$ or $X = G$. For $p \in [1, \infty]$ the *supremum p-Wasserstein distance* between $\mu, \mu' \in \mathcal{P}(\mathbb{R})^X$ is given by

$$\overline{w}_p(\mu, \mu') = \sup_{x \in X} w_p(\mu(x), \mu'(x)).$$

Lemma 1.5.9. *Let $p \in [1, \infty]$ and X as in the previous definition. Then the supremum p-Wasserstein distance is a metric on $\mathcal{P}(\mathbb{R})^X$.*

Definition 1.5.10. A *coupling* between $\nu, \nu' \in \mathcal{P}(\mathbb{R})$ is a random vector $(Z, Z') \sim v$ on $\mathbb{R}^2$ such that $Z \sim \nu$ and $Z' \sim \nu'$. For the set of all couplings we write $\mathcal{C}(\nu, \nu')$.

Lemma 1.5.11. *Let $p \in [1, \infty)$. Then the equalities*

$$\begin{aligned} w_p(\nu, \nu') &= \min_{(Z,Z') \in \mathcal{C}(\nu,\nu')} \mathbb{E}[|Z - Z'|^p]^{\frac{1}{p}}, \\ w_\infty(\nu, \nu') &= \min_{(Z,Z') \in \mathcal{C}(\nu,\nu')} \inf\{z \in \mathbb{R} : v(|Z - Z'| > z) = 0\} \end{aligned}$$

hold.

Theorem 1.5.12. *The operator $\mathcal{T}_\pi$ on $\mathcal{P}(\mathbb{R})^G$ is γ-contractive with regard to $\overline{w}_p$ for $p \in [0, \infty]$, the same applies to the operator $\mathcal{T}_\pi$ on $\mathcal{P}(\mathbb{R})^S$.*

For the proof we follow [BDR23, p. 93].

Proof. Let $p \in [0, \infty)$. For each $(s, a) \in G$ let $Z(s, a) \sim \nu(s, a)$ and $Z'(s, a) \sim \nu'(s, a)$ with $\nu \in \mathcal{P}(\mathbb{R})^G$ and $\nu' \in \mathcal{P}(\mathbb{R})^G$. With Lemma 1.5.6 it follows that

$$(r(s, a, X') + \gamma Z(X', A'), r(s, a, X') + \gamma Z'(X', A'))$$

is a coupling between $(\mathcal{T}_\pi \nu)(s, a)$ and $(\mathcal{T}_\pi \nu')(s, a)$. Via Lemma 1.5.11 we have

$$\begin{aligned} w_p^p((\mathcal{T}_\pi \nu)(s, a), (\mathcal{T}_\pi \nu')(s, a)) &= \min_{(Z,Z') \in \mathcal{C}(\nu,\nu')} \gamma^p \mathbb{E}\left[|Z(X', A') - Z'(X', A')|^p\right] \\ &\leq \int_S \int_{A_{s'}} \gamma^p \mathbb{E}\left[|Z(s', a') - Z'(s', a')|^p\right] \, \pi(da', s') p(ds', s, a) \end{aligned}$$

$$\begin{aligned}&\leq \gamma^p \sup_{(s',a')\in G} \mathbb{E}\left[|Z(s',a') - Z'(s',a')|^p\right] \\ &= \gamma^p \overline{w}_p^p(\nu, \nu').\end{aligned}$$

Taking the p-th root on both sides and taking the supremum over $(s, a) \in G$ gives the desired contraction property. We omit the proof of the case $p = \infty$. □

The distributional Bellman operators $\mathcal{T}_\pi$ can be shown to be contractive more generally for probability metrics meeting certain requirements, for a characterization we refer to [BDR23, Theorem 4.25].

Theorem 1.5.13. *Let the reward function of the MDP be bounded and let $\nu_0 \in \mathcal{P}(\mathbb{R})^G$ be any initial distribution with bounded support. Then the sequence*

$$\nu_{k+1} = \mathcal{T}_\pi(\nu_k)$$

converges to ν^π with respect to $\overline{w}_p$ for any $p \in [0, \infty]$.

Proof. By assuming bounded rewards, all return values are bounded as well, thus all return distributions involved have bounded support which makes all Wasserstein distances finite. Since the space of return distributions with bounded support $\mathcal{P}(\mathbb{R})^G$ is closed under $\mathcal{T}_\pi$ and ν^π is part of that space, we can restrict our attention to that space. By Theorem 1.5.5 we already know that ν^π is fixed point of $\mathcal{T}_\pi$ and by Theorem 1.5.12 we know that the operator is a contraction, hence Lemma 1.3.3 does the rest. □

Theorem 1.5.14. *Let $\mathcal{T}^{\mathcal{G}}$ be the distributional Bellman optimality operator with a greedy selection rule $\mathcal{G}$, let the reward function of the MDP be bounded and suppose there exists a unique optimal policy π^*. Then, for any initial distribution $\nu_0 \in \mathcal{P}(\mathbb{R})^G$ with bounded support the sequence*

$$\nu_{k+1} = \mathcal{T}_\pi(\nu_k)$$

converges to ν^{π^*} with respect to $\overline{w}_p$ for any $p \in [0, \infty]$.

Remark 1.5.15. The distributional Bellman optimality operators are sadly not well-behaved in cases where multiple optimal policies exist, for counterexamples to convergence see [BDR23, Example 7.11]. This is due to the fact that in general the distributional optimality operators are not contractions anymore, see [BDR23, Proposition 7.7].

1.6 Reinforcement Learning Terminology

Reinforcement learning (RL) is the problem of finding (learning) an optimal policy (w.r.t. to a given optimality criterion suitable for the MDP) without direct access to the underlying transition kernel and reward function of the MDP.

In the classical *online* reinforcement learning setting, it is possible to sample transitions (s, a, r, s') by following a *behavior policy* π_b in order to be able to learn an optimal policy (referred to as *target policy*), while in the *offline* (also *batch*) reinforcement learning setting, a fixed dataset (batch) of transitions from an MDP is given without any possibility to interact with the MDP, i.e., the behavior policy used to collect the batch is not known (in the ideal case the batch contains transitions recorded from the environment interaction of an expert). Many state-of-the-art reinforcement learning algorithms are not purely online, but store past transitions in a batch and additionally to sampling the MDP also repeatedly sample from that batch, thus they are a middle ground between pure online and pure offline reinforcement learning, also referred to as the *growing batch* reinforcement learning problem (see [LGR12]).

Online algorithms that at each step estimate the (action)-value of the current behavior policy and improve the behavior policy towards the optimal policy are classified as *on-policy*, while algorithms that directly try to estimate the (action)-value of the optimal policy while following a behavior policy are classified as *off-policy* algorithms.

When trying to solve a reinforcement learning problem, there are two fundamental approaches:

1. *Model-free* reinforcement learning approaches directly try to learn an optimal policy without building a model of the transition kernel $p(s, a, s)$ or the reward function $r(s, a, s')$.

2. *Model-based* reinforcement learning approaches build a model of the transition kernel and reward function.

One possible advantage of model-based approaches is the possibility to query the model in order to plan ahead without querying the environment.

Algorithms that primarily learn an optimal (action)-value function are referred to as *value-based* methods, while algorithms that directly learn an optimal policy are referred to as *policy-based* methods. Policy-based methods that additionally learn the value function of the policy are commonly referred to as *actor-critic* algorithms, *actor* refers to the estimated policy and *critic* to the estimated value function. Algorithms that learn a return distribution are classified as *distributional*.

1.7 Multi-Objective RL

In many practical tasks there are multiple, often conflicting objectives the agent should optimize for simultaneously. Thus, it is natural to extend the MDP definition to the multi-objective setting by allowing reward vectors $\mathbf{r} \in \mathbb{R}^n$ instead of scalar rewards, where each vector component r_i represents the reward regarding objective i.

In a multi-objective MDP with n objectives the value function of a policy takes values in $\mathbb{R}^n$. Each objective i is to maximize the value function v_i^π by criteria as in Section 1.2.

Definition 1.7.1. Let v_i^π denote the value function with regard to objective i. We define

$$\pi >_i \pi' \quad :\Leftrightarrow \quad \forall s \in S : v_i^\pi(s) \geq v_i^{\pi'}(s) \land \exists s \in S : v_i^\pi(s) > v_i^{\pi'}(s).$$

Definition 1.7.2. Let $>_i$, $i \in \{1\ldots,n\}$ be as in the above definition. A policy π *Pareto-dominates* another policy π' if

$$\pi \succ \pi' :\Leftrightarrow \forall i : \pi \geq_i \pi' \wedge \exists j : \pi >_j \pi'.$$

A policy π is called *Pareto-optimal* if there is no policy that Pareto-dominates π. The set of all such policies is called *Pareto-front.*

Thus, unlike in the single objective case, there is no unique optimal value function, especially not without specifying how to prioritize (weight) between the different objectives. Furthermore, in the general case the existence of deterministic Pareto-optimal policies are lost. Multi-objective reinforcement learning is an entire field on its own, especially since there are applications where the weights of the objectives are not known during the learning phase, which requires learning the Pareto-front, for an overview of problem formulations we refer to [Roi+13].

A way to prioritize between the different objectives is by making use of a *scalarization* function (also called *utility* function) $f_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}$, where $\mathbf{w} \in \mathbb{R}^n$ is a parameter vector that weights the individual objectives. If the scalarization function $f_{\mathbf{w}}$ is linear, by the linearity of expectation we have

$$f_{\mathbf{w}}(\mathbf{v}^\pi) = \mathbf{w} \cdot \mathbf{v}^\pi = \mathbf{w} \cdot \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathbf{R}_k\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathbf{w} \cdot \mathbf{R}_k\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k f_{\mathbf{w}}(\mathbf{R}_k)\right].$$

Thus, it is possible to treat the multi-objective MDP the same way as a single objective MDP with the scalar reward $f_{\mathbf{w}}(\mathbf{r})$.

Theorem 1.7.3. *Let π^* be an optimal policy of the single objective MDP via the scalarization $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$, where $w_i > 0$ for all $i \in \{1,\ldots,n\}$. Then π^* is Pareto-optimal in each $s \in S$.*

Proof. Let $s \in S$ be fixed but arbitrary. Since π^* is optimal, we have for any $\pi \in \Pi$ that

$$\sum_{i=1}^{n} w_i \cdot (v^{\pi^*}(s) - v^{\pi}(s)) \geq 0.$$

Assume that π^* is not Pareto-optimal in s. Then there exists $\pi' \in \Pi$ with $v_i^{\pi'}(s) \geq v_i^{\pi^*}(s)$ for all $i \in \{1,\ldots,n\}$ and $v_j^{\pi'}(s) > v_j^{\pi^*}(s)$ for some $j \in \{1,\ldots,n\}$. Sine $w_i > 0$ for each i we obtain

$$\sum_{i=1}^{n} w_i \cdot (v^{\pi^*}(s) - v^{\pi'}(s)) < 0,$$

which is a contradiction. $\square$

Thus, regular reinforcement learning methods are sufficient to obtain Pareto-optimal policies if the reward vectors are linearly scalarized. The question, whether all policies on the Pareto-front can be obtained via linear scalarization depends on the convexity of the value function spaces, see [LHY23, Sec. 4.1].

2 Algorithms and Implementation

In this chapter we generally denote the space of all stationary policies by Π.

2.1 Q-Learning

Let S and A be finite. Then the action value function can be represented by a table with $|S|$ rows and $|A|$ columns where each entry represents $Q(s, a)$. The idea of Q-learning is to iteratively improve a Q-table estimate of $Q^*(s, a)$ by updating the values via the *temporal-difference* error $\delta_t := \hat{T}Q(s, a) - Q(s, a)$ where $\hat{T}$ denotes the *empirical Bellman optimality operator*

$$\hat{T}Q(s, a) := r(s, a, s') + \gamma \max_{a' \in A_{s'}} Q(s', a'),$$

see Algorithm 2.1.

Algorithm 2.1 Tabular Q-learning

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: approximation of optimal action value function $Q^*(\cdot, \cdot)$
Parameters: learning rate α, exploration rate ϵ, optional decay schedules for α, ε

```
1  **Initialize** $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
2  Sample initial state $s$ from environment
3  **for** number of training steps **do**
4      Optionally update $\epsilon$ or $\alpha$ according to decay schedules
5      $a \leftarrow \begin{cases} \operatorname{argmax}_a Q(s, a) & \text{with probability } 1 - \epsilon \text{, breaking ties randomly} \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
6      Execute action $a$ in environment, observe next state $s'$ and reward $r$
7      $y \leftarrow \begin{cases} r & \text{if } s' \text{ is } \textit{terminal} \\ r + \gamma \max_{a'} Q(s', a') & \text{otherwise} \end{cases}$
8      $Q(s, a) \leftarrow Q(s, a) + \alpha(y - Q(s, a))$
9      **if** $s'$ is *terminal* or *truncation condition* is true **then**
10         Sample new initial state $s$ from environment
11     **else** $s \leftarrow s'$
12 **end**
```

The case-handling in line 7 is usually avoided in the literature [SB18, Sec. 6.8], where instead it is required to initialize the Q-table with the constraint $Q(s_t, a) = 0$ whenever

s_t is a terminal state, which is equivalent, but in our opinion an unpractical formulation, since in practice we usually do not have a list of terminal states available, which practically means to initialize the table with zeros or do the case handling as in our formulation.

The truncation condition in line 9 is in practice usually a time limit, which is met when the number of steps since the last initial state exceeds a certain threshold. This makes especially sense in infinite-horizon problems, where the agent might be possibly forever stuck in areas of the state space (even under optimal behavior), e.g., in a game like chess that could possibly go on forever.

In order to be able to precisely formulate the asymptotic convergence guarantees of the algorithm we need some definitions first. We mostly follow the notations of [RA21], who give a more formal treatment of the original result and proof from [WD92].

Definition 2.1.1. The *learning-trajectory space* of an MDP M is the measurable space

$$(\Omega_M, \mathcal{F}_M) := \left((G \times S)^{\mathbb{N}_0}, \bigotimes_{t \in \mathbb{N}_0} 2^{(G \times S)} \right).$$

Let $\Delta(\Omega_M, \mathcal{F}_M)$ denote the set of all probability measures $\mathbb{P}$ on $(\Omega_M, \mathcal{F}_M)$ satisfying

$$\mathbb{P}(S_t = s' \mid S_0, A_0, S'_0, \dots S_t, A_t) = p(s' \mid S_t, A_t)$$

almost surely for any $s' \in S$ and $t \in \mathbb{N}_0$, where $p(\cdot \mid \cdot, \cdot)$ is the transition kernel of M.

The environment interaction of the algorithm generates what we refer to as a *learning-trajectory*, a sequence of transitions $((S_i, A_i, S'_i))_{i \in \mathbb{N}_0}$ which we identify as

$$\omega = (S_0, A_0, S'_0, S_1, A_1, S'_1 \dots) \in \Omega_M.$$

Definition 2.1.2. We denote the occurrences of state-action pair $(s, a) \in G$ along a learning-trajectory $\omega \in \Omega_M$ by

$$O_{(s,a)}(\omega) := \{t \in \mathbb{N}_0 : (S_t, A_t)(\omega) = (s, a).\}$$

Definition 2.1.3. The *Q-learning iterates* on a finite MDP M with discount rate γ, a learning rate sequence $\alpha = (\alpha_t)_{t \in \mathbb{N}_0}$ in $[0, 1]$ and a learning-trajectory $\omega \in \Omega_M$ form the sequence $(Q^\alpha_t(\omega))_{t \in \mathbb{N}_0}$ in $\ell^\infty(G)$ defined recursively by $Q^\alpha_0(\omega) = \mathbf{0}$ and

$$Q^\alpha_{t+1}(\omega)(s, a) := \begin{cases} Q^\alpha_t(\omega)(s, a), & \text{if } t \notin O_{(s,a)}(\omega) \\ Q^\alpha_t(\omega)(s, a) + \alpha_t(r(s, a, S'_t(\omega)) \\ \qquad + \gamma \max_{a \in A_s} Q(\omega)(S'_t(\omega), a)), & \text{otherwise} \end{cases}$$

Theorem 2.1.4. *Let $\mathbb{P} \in \Delta(\Omega_M, \mathcal{F}_M)$ and $(\alpha_t)_{t \in \mathbb{N}_0}$ in $[0, 1]$ be such that the* Robbins-Monroe *conditions are satisfied, i.e., for all $(s, a) \in G$ and $\mathbb{P}$-almost all $\omega \in \Omega_M$*

$$\sum_{t \in O_{(s,a)}(\omega)} \alpha_t = \infty \quad \text{and} \quad \sum_{t \in O_{(s,a)}(\omega)} \alpha_t^2 < \infty. \tag{2.1}$$

Then the Q-learning iterates $(Q^\alpha_t(\omega))_{t \in \mathbb{N}_0}$ converge uniformly to the optimal action value function Q^*_M for $\mathbb{P}$-almost all $\omega \in \Omega_M$.

The condition $\sum_{t\in O_{(s,a)}(\omega)} \alpha_t = \infty$ implicitly requires $O_{(s,a)}(\omega)$ to be infinite for all state-action pairs (s,a), which means that the algorithm needs to interact with the environment in a way (which induces the measure $\mathbb{P}$) to visit every state-action pair infinitely many times, which is the reason for the requirement of the random action case in line 5 of Algorithm 2.1. In practice, to get a good convergence rate to the approximately optimal Q-function it is essential to find a good balance between exploring random actions (to potentially learn something new) and exploiting the best known actions, thus the exploration rate is usually set to a constant between 0.01 and 0.05 or annealed to such a constant starting from an initially higher exploration rate.

Since state-action pairs (s_t, a) where s_t is a terminal state are never updated and already have their correct Q-value initialized with zero, the Robbins-Monroe conditions for such pairs are irrelevant for the asymptotic convergence guarantee.

Note, that it is not required that $S'_i(\omega) = S_{i+1}(\omega)$, thus the truncation condition in the algorithm does not affect convergence as long as it does not make states inaccessible.

2.2 Q-Learning with Function Approximation

Tabular Q-learning is practically only feasible for MDPs with a few states. One solution to deal with big discrete or even continuous state spaces is to approximate Q^* in some parametrized function space $\mathcal{F}(\theta) \subseteq \mathbb{R}^{S\times A}$, where $\theta \subseteq \mathbb{R}^n$ denotes the parameter vector that parametrizes the functions in $\mathcal{F}$. Such a function space could in case of *non-linear function approximation* be the space of all functions realized by a neural network with weights w or in case of *linear function approximation* a linear space

$$\left\{ Q \in \mathbb{R}^{S\times A} : Q(s,a,\theta) = \sum_{i=1}^{n} \theta_i \phi_i(s,a) \right\},$$

where $\phi : S \times A \to \mathbb{R}^n$ is a function that assigns each state-action pair a *feature vector*.

2.2.1 Semi-Gradient method

In order to realize Q-learning as in the tabular case, the parameters of the action value function $Q(s,a,\theta)$ are gradually updated to minimize the loss

$$\begin{aligned} L(\theta) &= \frac{1}{2}\left(\mathcal{T}Q(s,a) - Q(s,a,\theta)\right)^2 \\ &\approx \frac{1}{2}\left(\hat{T}Q(s,a) - Q(s,a,\theta)\right)^2. \end{aligned}$$

Note that we omitted the dependency on θ of $\hat{T}Q(s,a,\theta)$, thus when minimizing the loss via stochastic gradient descent we calculate a *semi-gradient*

$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha \nabla L(\theta_t) \\ &= \theta_t + \alpha\left(\mathbb{E}\left[\hat{T}Q(s,a,\theta_t)\right] - Q(s,a,\theta_t)\right)\nabla Q(s,a,\theta_t) \end{aligned}$$

$$\approx \theta_t + \alpha \left(\hat{T}Q(s,a,\theta_t) - Q(s,a,\theta_t)\right) \nabla Q(s,a,\theta_t)$$

by ignoring this dependency. In the linear function approximation case $\nabla Q(s,a,\theta_t) = \phi(s,a)$. If we set $\phi : S \times A \to \mathbb{R}^{S\times A}$ such that $\phi_i(s,a) := 1_{(s,a)}(i)$ is the indicator function of index pair (s,a), i.e., $Q(s,a,\theta) = \theta_{(s,a)}$ the update reduces to the regular tabular Q-learning update

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha(\hat{T}Q_t(s,a) - Q_t(s,a)).$$

Unfortunately, in general even with linear function approximation there are circumstances where Q-learning has been shown to diverge, this is typically referred to in the literature as *deadly-triad* as experience showed that instability and divergence issues arise as soon as one tries to combine off-policy learning with bootstrapping target estimates and function approximation, see [SB18, Sec. 11.2, 11.3]. This can be mostly attributed to the fact that one update to the parameters changes the Q-values for multiple states, which can have the advantage of *generalization* if it improves the values for multiple other states as well, but can also be responsible for regressions in multiple other states and an overall unstable or even divergent training process. With neural networks in particular it has been empirically observed that the training process is often unstable and very sample inefficient, see [Rie05]. That changed with the publication of DQN in [Mni+15] described in Section 2.3.

2.2.2 Residual Method

Instead of calculating the semi-gradient of the temporal difference error, one could try to work with the true gradient as well to find

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \nabla L(\theta_t)\\ &= \theta_t + \alpha \left(\mathbb{E}\left[\hat{T}Q(s,a,\theta_t)\right] - Q(s,a,\theta_t)\right)\\ &\qquad \left(\nabla Q(s,a,\theta_t) - \gamma \mathbb{E}\left[\nabla \max_{a' \in A_{s'}} Q(s',a',\theta_t)\right]\right).\end{aligned}$$

This leads to a *double-sampling* issue, since we can not use one single sample (s,a,s',r) to get an unbiased estimate of the product if we would take the same sample for estimating both expectations. In the literature various algorithms have been proposed that are often referred to as *residual-gradient* or *full-gradient* methods, but despite convergence guarantees those have not been successful in practice due to slower convergence and convergence to bad action value functions according to [SB18, p. 273] and [WU22, p. 3].

2.2.3 Notes on Function Approximation

Using function approximation introduces a number of theoretical challenges:

1. In a function class parametrizing value functions that does not contain the optimal value function, there is no general notion of the best representable value function due

to the fact that the standard notion of optimality induces a partial order. This issue does not apply for the weaker notion of optimality we defined in Definition 1.2.7, however. For a deeper discussion of this issue we refer to [Nai+19]. The Q-iteration may still converge to a fixed point in that function space, however theoretical results in the function approximation setting are sparse.

2. Consider a linear value function approximation

$$V(s) = V(\phi(s), \theta),$$

where $\phi : S \to \Phi \subseteq \mathbb{R}^e$ maps states to features and $\theta \subseteq \mathbb{R}^p$ denotes a vector of adaptable parameters. If ϕ is not injective, the resulting feature space MDP becomes partially observable, see [Has12, Sec. 7.2.1.2], thus leading to a loss of most theoretical convergence guarantees. This especially affects the most straightforward function approximation techniques such as state aggregation.

2.3 Deep Q-Learning

Compared to semi-gradient Q-learning, DQN introduces the following changes:

1. A second Q network initialized with the same parameters referred to as *target network* is used to calculate the empirical Bellman operator. The target network is a copy of the primary network that is updated every C steps, thus the loss function remains constant for C steps instead of changing every step, which stabilizes training.

2. Instead of a mean squared error loss, a Huber loss

$$\ell_\kappa(x) := \begin{cases} \frac{1}{2}x^2, & \text{for } |x| \leq \kappa, \\ \kappa \cdot (|x| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases}$$

with $\kappa = 1$ is used. This has been ambiguously described in the paragraph before algorithm 1 in [Mni+15] where it is supposedly talked about clipping the gradient of the squared loss to $[-1, 1]$ which we would like to point out is not equivalent to the Huber loss with $\kappa = 1$ since the absolute value of the multiplied factor ∇Q could be greater than one, subsequent publications of affiliated authors mention the Huber loss explicitly, see [Dab+18a, Footnote 4] or [CC21, Sec. 5.4].

3. Experience replay: Instead of learning purely online, samples (s, a, s', r) are stored in an experience replay buffer and every update is formed by uniformly sampling a batch of transitions from that experience replay buffer. This brings a couple of advantages:
   - Multiple samples are used for each parameter update, which improves the gradient calculation
   - Most optimization algorithms assume that samples are independently and identically distributed which would not be given with sequential data

- Higher data-efficiency by reusing samples multiple times for updates
- Breaking correlation between the samples, as learning from batches of consecutive samples is inefficient, e.g., a change of certain maximization actions might shift the agent to produce samples mostly from a certain subset of the state space and overfit on that data.

See Algorithm 2.2 for the pseudocode of DQN.

Algorithm 2.2 DQN

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: approximation $Q_\theta(\cdot,\cdot)$ of optimal action value function $Q^*(\cdot,\cdot)$
Parameters: learning rate α, exploration rate ε, target update parameters C and τ, batch size n, optional decay schedules for ε or α, replay buffer capacity, neural network architecture

```
1  **Initialize** $Q_\theta : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ with random weights $\theta$, let $Q_\theta(s,a) := Q_\theta(s)[a]$
2  **Initialize** $Q_{\theta^-} : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ with weights $\theta^- = \theta$
3  **Initialize** replay-buffer $D$ of given capacity
4  **Initialize** stochastic gradient optimizer opt
5  Sample initial state $s$ from environment
6  **for** number of training steps **do**
7      Optionally update $\epsilon$ or $\alpha$ according to decay schedules
8      $a \leftarrow \begin{cases} \text{argmax}_a Q_\theta(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
9      Execute action $a$ in environment, observe next state $s'$ and reward $r$
10     Store $(s,a,r,s')$ in $D$
11     Sample random mini-batch of transitions $(s_i,a_i,r_i,s'_i)_{i\leq n}$ from $D$
12     $y_i \leftarrow \begin{cases} r_i & \text{if } s'_i \text{ is } \textit{terminal} \\ r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i,a') & \text{otherwise} \end{cases}$  for $i \in \{1,\dots,n\}$
13     $L(\theta) \leftarrow \frac{1}{n}\sum_{i=1}^n \ell_1(y_i - Q_\theta(s_i,a_i))$
14     $\theta \leftarrow \theta - \text{opt.step}(\alpha, \nabla L(\theta))$
15     Every $C$ steps $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$
16     **if** $s'$ is *terminal* or *truncation condition* is true **then**
17         Sample new initial state $s$ from environment.
18     **else** $s \leftarrow s'$.
19 **end**
```

In this algorithm and all those that follow we denote the change to parameters θ done by one step of a stochastic gradient descent algorithm with regard to an objective function f and step size η as $\text{opt.step}(\eta, \nabla f(\theta))$. The most naive form of stochastic gradient descent would mean $\text{opt.step}(\eta, \nabla f(\theta)) = \eta \cdot \nabla f(\theta)$, but state-of-the-art algorithms such as *Adam* introduced in [KB15] keep track of additional variables across different steps such as moment vectors, hence we symbolically initialize such an optimization algorithm in the beginning of the pseudocode and repeatedly call the step method. This is also motivated by practice: For example in PyTorch the gradient is calculated via `f.backward()` and the parameters are update via `optimizer.step()`.

Compared to the original proposal of DQN in [Mni+15] we include an additional soft-update parameter $0 \leq \tau \leq 1$ which allows for a soft update of the target network parameters in line 15 originally proposed in [Lil+16, p. 4]. The original DQN algorithm is a special case with $\tau = 1$ which corresponds to a hard copy of the parameters. Usually in practice either $C > 1$ and $\tau = 1$ or $C = 1$ and $\tau > 1$.

Despite the great empirical success of DQN, its theoretical properties are still a subject of ongoing research. [CCM23] showed for a purely online version of DQN for linear function approximation that the introduction of a target network together with truncation warrants convergence to the optimal action value function, where truncation refers to the truncation effect the Huber loss has implicitly on the temporal difference. In their adaptation they suggest instead to truncate the target via

$$r_i + \gamma \max_{a'} \lceil Q(s', a', \theta^-) \rceil,$$

where $\lceil x \rceil = x$ if $|x| \leq r$, $\lceil x \rceil = -r$ if $x < -r$ and $\lceil x \rceil = r$ if $x > r$, where $r = \frac{1}{1-\gamma}$. In a similar spirit [CMS20] show the convergence of a two-timescale algorithm inspired by DQN where the slower timescale can be seen as a modified target network.

Convergence guarantees for tabular Q-learning with experience replay have been established in [SS21].

2.4 Double DQN

In [Has10] it was pointed out that the maximum operation in the Q-learning target $\max_{a'} Q_t(s', a')$ is an unbiased estimate for $\mathbb{E}\left[\max_{a'} Q_t(s', a')\right]$ but a biased estimate for $\max_{a'} \mathbb{E}\left[Q_t(s', a')\right]$, which we seek to approximate in the first place ($(Q_t(s', a'))_{a' \in A_{s'}}$ is treated as a collection of random variables here and the expectation should be thought over all possible runs of the same experiment) which leads to overestimation. In the tabular case the author proposed double Q-learning which does not have unbiased estimates either, but overestimation is eliminated and underestimation may occur. This method learns two independent action value estimates Q_a and Q_b but when calculating the target for Q_a it takes the maximum over Q_b and vice versa. A minimal adaption for DQN was proposed in [HGS16] by replacing line 12 in Algorithm 2.2 with

$$y_i \leftarrow \begin{cases} r_i, & \text{if } s_i' \text{ is } \textit{terminal}, \\ r_i + \gamma Q_{\theta^-}(s_i', \operatorname{argmax}_a Q_\theta(s, a)), & \text{otherwise}, \end{cases}$$

which does not make DQN more computationally demanding and has been empirically shown to retain advantages from double Q-learning. Later approaches in the continuous domain however did not find that minimal change to be sufficient and use two action value estimates, see Section 2.7.

2.5 Deep Deterministic Policy Gradient

If action spaces are not discrete then Q-learning cannot be directly used in practice, since the maximization operation $\max_{a \in A} Q(s,a)$ is not tractable with neural networks as the function approximation class.

Starting from here, whenever we refer to a continuous action space, we always assume the action set to be a real interval $A = [A_{\min}, A_{\max}]$.

One popular way to deal with this issue is the introduction of a deterministic policy parametrization

$$\mu_\phi : S \to A, \ s \mapsto \mu_\phi(s),$$

that jointly serves as an approximation to the optimal policy (by Theorem 1.3.7 an optimal deterministic policy exists) as well as a maximizer for $Q_\theta(s, \cdot)$ via

$$\max_{a \in A} Q_\theta(s,a) = Q_\theta(s, \mu_\phi(s)).$$

This allows us to define an objective function

$$J(\phi) := \mathbb{E}_{s \sim D} \left[Q_\theta(s, \mu_\phi(s)) \right] \tag{2.2}$$

which we seek to maximize.

Building upon DQN by replacing the empirical Bellman optimality operator with the Bellman operator of policy μ and also parametrizing a deterministic policy μ with a neural network which is used for calculating the empirical Bellman target one arrives at the Deep Deterministic Policy Gradient algorithm (DDPG) as proposed in [Lil+16], see Algorithm 2.3.

Albeit the authors of DDPG used another theoretical background (deterministic policy gradient theorem, see Theorem 2.6.1) for motivating their algorithm design, we choose to view DDPG as a Q-iteration algorithm instead for reasons we discuss in Section 2.6.

While ε-greedy exploration would work as well in theory by sampling a random action from $[A_{\min}, A_{\max}]$, the authors who proposed Deep Deterministic Policy Gradient (DDPG) choose to rather implement exploration by adding an action noise modelled as Ornstein-Uhlenbeck process to the greedy action $\mu_\phi(s)$. Subsequent work however showed no benefit of the Ornstein-Uhlenbeck process, see [FHM18, Sec. 6.1], thus we follow [FHM18] by making the behavior policy follow

$$\operatorname{clip}_A(\mu_\phi(s) + \varepsilon), \quad \varepsilon \sim \mathcal{N}(0, \sigma), \quad \operatorname{clip}_A(x) := \max(\min(x, A_{\max}), A_{\min}).$$

Compared to ε-greedy exploration this might have the benefit of more targeted exploration during later stages of training when the current greedy action is already near the optimal action, with the disadvantage of biasing exploration to a neighborhood of an action that has an inaccurately high value estimate in the beginning of training.

To give an idealized theoretical motivation (leaving all function approximation issues aside) for the algorithm we can view the algorithm as a successive sequence of policy evaluation

$$Q_{\theta_{k+1}} = \mathcal{T}^\mu Q_{\theta_k}$$

Algorithm 2.3 DDPG

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: deterministic $\mu_\phi(\cdot)$ approximating optimal deterministic policy $\mu^*(\cdot)$
Parameters: learning rates η_c, η_a, action noise σ, target update parameters C and τ, batch size n, optional decay schedules for learning rates, replay buffer capacity, neural network architectures

```
 1 Initialize $\mu_\phi : \mathcal{S} \to \mathcal{A}$ with random weights $\phi$
 2 Initialize $\mu_{\phi^-} : \mathcal{S} \to \mathcal{A}$ with weights $\phi^- = \phi$
 3 Initialize $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with random weights $\theta$
 4 Initialize $Q_{\theta^-} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with weights $\theta^- = \theta$
 5 Initialize replay-buffer $D$ of given capacity
 6 Initialize stochastic gradient descent optimizers $\mathrm{opt}_\theta$, $\mathrm{opt}_\phi$
 7 Sample initial state $s$ from environment
 8 for number of training steps do
 9     Optionally update $\eta_c$, $\eta_a$ or $\sigma$ according to decay schedules
10     Choose action $a = \mathrm{clip}_{\mathcal{A}}(\mu_\phi(s) + \varepsilon)$ with $\varepsilon \sim \mathcal{N}(0, \sigma)$
11     Execute action $a$ in environment, observe next state $s'$ and reward $r$
12     Store $(s, a, r, s')$ in $D$
13     Sample random mini-batch of transitions $(s_i, a_i, r_i, s_i')_{i \leq n}$ from $D$
14     $y_i \leftarrow \begin{cases} r_i & \text{if } s_i' \text{ is } \textit{terminal} \\ r_i + \gamma Q_{\theta^-}(s_i', \mu_{\phi^-}(s_i')) & \text{otherwise} \end{cases} \quad \text{for } i \in \{1, \dots, n\}$
15     $L(\theta) \leftarrow \frac{1}{n} \sum_{i=1}^n (y_i - Q_\theta(s_i, a_i))^2$
16     $J(\phi) \leftarrow \frac{1}{n} \sum_{i=1}^n Q_\theta(s_i, \mu_\phi(s_i))$
17     $\theta \leftarrow \theta - \mathrm{opt}_\theta . \mathrm{step}(\eta_c, \nabla L(\theta))$
18     $\phi \leftarrow \phi + \mathrm{opt}_\phi . \mathrm{step}(\eta_a, \nabla J(\phi))$
19     Every $C$ steps $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$
20     Every $C$ steps $\phi^- \leftarrow \tau\phi + (1 - \tau)\phi^-$
21     if $s'$ is terminal or truncation condition is true then
22         Sample new initial state $s$ from environment
23     else $s \leftarrow s'$
24 end
```

and policy improvement. This viewpoint has been inspired by the convergence arguments the authors of Soft Actor Critic made in [Haa+18], which we discuss in Section 2.8.

Lemma 2.5.1 (Policy Improvement). *Let μ' be such, that $Q^\mu(s, \mu'(s)) \geq Q^\mu(s, \mu(s))$ for all $s \in S$. Then $Q^{\mu'}(s, a) \geq Q^\mu(s, a)$ for all $(s, a) \in G$.*

Proof. By the Bellmen equation for the deterministic policy μ we have

$$Q^\mu(s, a) = \int_S r(s, a, s') + \gamma Q^\mu(s', \mu(s'))\, p(ds', s, a)$$

$$\leq \int_S r(s,a,s') + \gamma Q^{\mu}(s', \mu'(s'))\ p(ds', s, a) = \mathcal{T}^{\mu'}(Q^{\mu}).$$

Repeated application of the Bellman evaluation operator on the right-hand side yields the desired result by Theorem 1.4.3. $\square$

Theorem 2.5.2 (Policy Iteration). *Let M be an MDP with bounded rewards and compact state and action spaces. Let Q_0 be any function and μ_0 be any deterministic policy. Iterative application of policy evaluation ($Q_{k+1} = \mathcal{T}^{\mu_k} Q_k$) and policy improvement (μ_{k+1} such that $Q_k(s, \mu_{k+1}(s)) \geq Q_k(s, \mu_k(s))$ for all $(s,a) \in G$) leads to convergence $\mu_k \to \mu^*$ and $Q_k \to Q^{\mu *}$ as $k \to \infty$.*

Proof. By the previous lemma the sequence of Q-iterates is monotonically increasing and bounded from above since the rewards are bounded. Thus, the sequence of Q-iterates is converging point wise to a limit function $Q^{\mu'}$. Since $Q^{\mu'}(s, \mu'(s)) \geq Q^{\mu'}(s, \mu(s))$ for any policy μ the same iterative argument as in the last lemma yields $Q^{\mu'}(s,a) \geq Q^{\mu}(s,a)$ for all $(s,a) \in G$ and thus $\mu' = \mu^*$. $\square$

2.6 Issues of Policy Gradient Methods

In the treatment so far we have focused on solving reinforcement learning problems via Q-iteration. Another approach to solve reinforcement learning problems are *policy gradient methods*, where the idea is to parametrize a policy $\pi_\phi(a,s)$ and directly search for the optimal policy by maximizing an objective such as

$$J_{\rho_0}(\phi) = \mathbb{E}_{s \sim \rho_0}[v_\gamma^{\pi_\phi}(s)].$$

There are various *policy gradient theorems* in the literature such as the following for deterministic policies from [Sil+14].

Theorem 2.6.1. *Let M be an MDP such that the density function $p(s \mid s, a)$ of the Markov kernel exists and is continuous and continuously differentiable w.r.t to a and assume the same for the reward function r. Let μ_ϕ be a deterministic policy parametrized in ϕ such that $\mu_\phi(s)$ is continuously differentiable w.r.t. to ϕ. Then*

$$\nabla_\phi J_{\rho_0}(\phi) = \int_S d^{\mu_\phi}(s) \nabla_\phi \mu_\phi(s) \frac{\partial}{\partial a} Q^{\mu}(s,a) \mid_{a=\mu_\phi(s)}\ ds \tag{2.3}$$

$$= \mathbb{E}_{s \sim d^{\mu_\phi}} \left[\nabla_\phi \mu_\phi(s) \frac{\partial}{\partial a} Q^{\mu}(s,a) \mid_{a=\mu_\phi(s)} \right], \tag{2.4}$$

where d^{μ_ϕ} denotes the density of the discounted state distribution *given as*

$$d^{\mu_\phi}(s) = \int_S \sum_{t=0}^{\infty} \gamma^t \rho_0(s) p^{\mu_\phi}(s \to s', t, \mu_0) ds',$$

where $p(s \to s', t, \mu_0)$ is defined recursively as

$$p(s \to s, 0, \mu_\phi) = 1,$$

$$
\begin{aligned}
p(s \to s', 1, \mu_\phi) &= p(s' \mid s, \mu_\phi(s)), \\
&\vdots \\
p(s \to s', t+1, \mu_\phi) &= \int_S p(s \to x, t, \mu_\phi) p(x \to s', 1, \mu_\phi) \; dx.
\end{aligned}
$$

Usually policy gradient theorems give the gradient as an expectation as in Eq. (2.4) which in theory makes it possible to estimate the policy gradient via Monte Carlo sampling. As pointed out in [NT20] usually state-of-the-art algorithms implement the policy gradient incorrectly since those methods fail to sample from the discounted state distribution, since it is practically not directly accessible. This is acknowledged by the original authors of DDPG, arguing that this is commonly done in practice in policy gradient implementations, see [Lil+16, Footnote 2]. In the case of DDPG the policy gradient is approximated as

$$
\frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial a} Q(s_i, a) \mid_{a=\mu(s_i)}, \nabla_\phi \mu_\phi(s_i) \quad s_i \sim D, \; i \in \{1, \dots, n\}
$$

where the samples come from the replay buffer and the authors cite Theorem 2.6.1 as their theoretical justification. We would like to point out, that this exactly estimates the gradient of the objective we defined in Eq. (2.2) with a different theoretical motivation.

As a result of that theory-practice gap there exists an active line of research that tries to address this issue, see [CVM23, Sec. 3]. On the other hand, in [Wu+22] the authors show that by using an experience replay buffer for the biased policy gradient estimate, this biased estimate is an unbiased estimator of $\nabla J_{\rho_0'}(\theta)$ where ρ_0' is different from ρ_0 and that a maximizer of $J_{\rho_0'}(\theta)$ is also a maximizer of $J_{\rho_0}(\theta)$. The authors argue that this explains the strong empirical performance of off-policy policy gradient methods.

In [TTM19, Sec. 3] it is pointed out, that when pure policy gradient methods are used it can not be ensured that the optimal policy is reached by policy gradient methods even when an optimal policy is parametrized by the function approximation class, due to gradient descent being prone to converge to local maxima instead of global ones.

Even if a global optimizer π_θ for $J_{\rho_0}(\theta)$ is found, it is not necessarily a uniformly optimal policy in the sense of Definition 1.2.5 due to the weighting by ρ_0, even though a uniformly optimal policy is clearly an optimizer of $J_{\rho_0}(\theta)$ due to

$$
\forall \pi \; \forall s : v_{\pi^*}(s) \geq v_\pi(s) \Rightarrow \forall \pi : \int_S \rho_0(s) v_{\pi^*}(s) \; ds \geq \int_S \rho_0(s) v_\pi(s) \; ds.
$$

Many of the modern deep reinforcement learning actor-critic algorithms often portrayed in the literature as policy-gradient methods can actually more accurately be interpreted as Q-iteration algorithms, see [GSP19, Sec. 4.1, 5.1].

2.7 Twin Delayed DDPG

Twin Delayed DDPG (TD3) was introduced in [FHM18] and directly improves upon DDPG by introducing three changes that help to reduce the error induced by function approximation:

1. In order to address overestimation bias that is caused by updating the policy parameters with the objective of maximizing an approximate action value function, the authors propose *clipped double Q-learning.* The idea is to parametrize two Q-functions (initialized with different initial weights) with separate target networks each and update each Q-function by clipping the Q-value in the target in the loss by the minimum over both value functions

   $$r + \gamma \min_{j \in \{1,2\}} Q_{\theta_j}(s, a).$$

   This adopts the idea of double Q-learning, convergence of that update in the tabular case is shown in [FHM18, Appendix A].

2. Alternating between policy evaluation and policy improvement steps generates a constantly moving optimization target for the policy. Since the value estimates of the current policy may not be good enough after one step of policy evaluation, the variance of value estimates is high, leading to poor policy updates. To counter that, the authors propose to delay policy improvement steps, i.e., to only perform a policy improvement step every d steps, where d is a new hyperparameter. Similarly, the target networks are also only updated softly every d steps.

3. Deterministic policies are prone to overfit on narrow peaks in the value estimate, hence the authors propose a regularization strategy they call *target policy smoothing.* Under the assumption that similar actions should have similar values, they propose fitting the value to a small area around the target action

   $$r + \gamma \mathbb{E}_{\epsilon \sim \mathrm{clip}_{[-c,c]}(\mathcal{N}(0,\sigma))} \left[Q_{\theta^-}(s', \mu_\phi(s') + \epsilon) \right].$$

The full pseudocode is given in Algorithm 2.4.

2.8 Soft Actor Critic

Soft actor critic (SAC) was originally proposed in [Haa+18] and later refined in [Haa+19]. In the following we will describe the latter algorithm design.

2.8.1 Theory

Soft actor critic tries to solve a *maximum entropy reinforcement learning* problem, where the objective is to find a maximum entropy stochastic policy with maximal value. Intuitively such methods try to perform as good as possible while at the same time as randomly as possible, which should greatly improve the exploration during learning.

Definition 2.8.1. The *entropy* of a random variable X with probability density function f is defined as $\mathcal{H}(X) = \mathbb{E}[-\log(f(X))]$.

Algorithm 2.4 TD3

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: deterministic $\mu_\phi(\cdot)$ approximating optimal deterministic policy $\mu^*(\cdot)$
Parameters: learning rates η_c, η_a, action noise σ, policy noise σ_t, policy noise clip c, policy delay d, target update parameters C and τ, batch size n, optional decay schedules for learning rates, replay buffer capacity, neural network architectures

```
 1 Initialize $\mu_\phi : \mathcal{S} \to \mathcal{A}$ with random weights $\phi$
 2 Initialize $\mu_{\phi^-} : \mathcal{S} \to \mathcal{A}$ with weights $\phi^- = \phi$
 3 Initialize $Q_{\theta_1}, Q_{\theta_2} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with random weights $\theta_1$ and $\theta_2$
 4 Initialize $Q_{\theta_1^-}, Q_{\theta_2^-} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with weights $\theta_1^- = \theta_1$ and $\theta_2^- = \theta_2$
 5 Initialize replay-buffer $D$ of given capacity
 6 Initialize stochastic gradient descent optimizers $\mathrm{opt}_{\theta_1}$, $\mathrm{opt}_{\theta_2}$, $\mathrm{opt}_\phi$
 7 Sample initial state $s$ from environment
 8 for number of training steps do
 9     Optionally update $\eta_c$, $\eta_a$, $\eta_\alpha$ according to decay schedules
10     Choose action $a = \mathrm{clip}_\mathcal{A}(\mu_\phi(s) + \varepsilon)$ with $\varepsilon \sim \mathcal{N}(0, \sigma)$
11     Execute action $a$ in environment, observe next state $s'$ and reward $r$
12     Store $(s, a, r, s')$ in $D$
13     Sample random mini-batch of transitions $(s_i, a_i, r_i, s'_i)_{i \leq n}$ from $D$
14     $a'_i \leftarrow \mathrm{clip}_\mathcal{A}(\mu_{\phi^-}(s') + \mathrm{clip}_{[-c,c]}(\varepsilon))$ with $\varepsilon \sim \mathcal{N}(0, \sigma)$ for $i \in \{1, \dots, n\}$
15     $y_i \leftarrow \begin{cases} r_i & \text{if } s'_i \text{ is } \textit{terminal} \\ r_i + \gamma \min_{j \in \{1,2\}} Q_{\theta_j^-}(s'_i, a'_i) & \text{otherwise} \end{cases}$ for $i \in \{1, \dots, n\}$
16     $L(\theta_j) \leftarrow \frac{1}{n} \sum_{i=1}^n (y_i - Q_{\theta_j}(s_i, a_i))^2$ for $j \in \{1, 2\}$
17     $L(\phi) \leftarrow -\frac{1}{n} \sum_{i=1}^n Q_{\theta_1}(s_i, \mu_\phi(s_i))$
18     $\theta_j \leftarrow \theta_j - \mathrm{opt}_{\theta_j}.\mathrm{step}(\eta_c, \nabla L(\theta_j))$ for $j \in \{1, 2\}$
19     Every $d$ steps do
20         $\phi \leftarrow \phi - \mathrm{opt}_\phi.\mathrm{step}(\eta_a, \nabla L(\phi))$
21         Every $C$ steps $\theta_j^- \leftarrow \tau\theta_j + (1 - \tau)\theta_j^-$ for $j \in \{1, 2\}$
22         Every $C$ steps $\phi^- \leftarrow \tau\phi + (1 - \tau)\phi^-$
23     if $s'$ is terminal or truncation condition is true then
24         Sample new initial state $s$ from environment
25     else $s \leftarrow s'$
26 end
```

Maximum entropy learning augments the reward function with an entropy term

$$\tilde{r}(S, A, S') := r(S, A, S') + \alpha\mathcal{H}(\pi(\cdot, S')),$$

where $\alpha \in [0, 1]$ is a *temperature* parameter that determines the relative importance of the entropy term. In the limit $\alpha \to 0$ the classical reinforcement learning objective is recovered.

The action value function $\tilde{q}$ with regard to the entropy augmented reward is referred to as *soft action value function.* We can thus derive the *Soft Bellman Operator* via

$$\tilde{T}_\pi \tilde{q}(s,a) := \int_S \tilde{r}(s,a,s') + \gamma \int_{A_{s'}} \tilde{q}(s',a')\ \pi(da',s')p(ds',s,a) \tag{2.5}$$

$$= \int_S r(s,a,s') + \gamma \int_{A_{s'}} \tilde{q}(s',a') - \frac{\alpha}{\gamma}\log(f_\pi(a',s'))\ \pi(da',s')p(ds',s,a), \tag{2.6}$$

where f_π denotes the probability density function of the policy π.

Corollary 2.8.2 (Soft Policy Evaluation)**.** *If both the reward function and the entropy of π are bounded, then for any mapping $Q_0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the sequence*

$$Q_{k+1} = \tilde{T}_\pi Q_k$$

converges to the soft action value function $\tilde{Q}^\pi(s,a)$ of policy π.

Proof. Direct consequence of Eq. (2.5) and Theorem 1.4.3. □

In order to give an objective for policy improvement we need a definition first.

Definition 2.8.3. For distributions P and Q with probability density functions p and q the *Kullback-Leibler divergence* is defined as

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x)\ \log\left(\frac{p(x)}{q(x)}\right)\ \mathrm{d}x.$$

In the following we will also write $D_{\mathrm{KL}}(p \parallel q)$ for the same expression.

The Kullback-Leibler divergence is a metric for the similarity of distributions. For policy improvement the idea is to make the policy similar to a distribution induced by the Q-function, giving higher probabilities around actions of higher value. Let $\pi(\cdot|s)$ denote the density of the policy denoted with the same symbol, then the objective for an improved policy π' given a policy π for each $s \in S$ is

$$\pi'(\cdot|s) = \mathrm{argmin}_{\pi' \in \Pi} D_{\mathrm{KL}}\left(\pi'(\cdot|s)\ \middle\|\ \frac{\exp(\alpha' Q^\pi(s,\cdot))}{Z^\pi(s)}\right), \quad \alpha' := \frac{\gamma}{\alpha}. \tag{2.7}$$

The choice to fit the policy to the probability density function of the *softmax policy* $\mathrm{sm}_{\alpha'}(Q)(a,s) := \frac{\exp(\alpha' Q(s,a))}{\int_A \exp(\alpha' Q(s,a))da}$ was not motivated in the original paper, however [SPC19] showed the following theorem (that is not formulated in the maximum entropy setting).

Theorem 2.8.4. *Let $\mathcal{T}_{\mathrm{soft}} Q(s,a) := \mathcal{T}_{\mathrm{sm}_\tau(Q)} Q(s,a)$ be the* softmax Bellman operator *and let $\mathcal{T}^k Q_0$ and $\mathcal{T}^k_{\mathrm{soft}} Q_0$ denote the k-th iteration of the respective operator over some initial Q_0. Then $\mathcal{T}_{\mathrm{soft}}$ converges to $\mathcal{T}$ exponentially fast in terms of τ, i.e., the upper bound of $\mathcal{T}^k Q_0 - \mathcal{T}^k_{\mathrm{soft}} Q_0$ decays exponentially fast as a function of τ.*

A neat side effect of the softmax policy density is that it helps to calculate the logarithm in the KL divergence as can be seen in the proof of the following lemma.

Lemma 2.8.5 (Soft Policy Improvement)**.** *Let π be any policy and π' as in Eq.* (2.7)*. Then $Q^{\pi'}(s,a) \geq Q^{\pi}(s,a)$ for all $(s,a) \in G$.*

Proof. By definition of the Kullback-Leibler divergence and logarithm calculus we get

$$\pi'(\cdot|s) = \operatorname{argmin}_{\pi' \in \Pi} \mathbb{E}_{a \sim \pi'} \left[\log(\pi'(a|s)) - \alpha' Q^{\pi}(s,a) + \log(Z^{\pi}(s)) \right]. \tag{2.8}$$

Since $\pi \in \Pi$ as well, we obtain

$$\begin{aligned} &\mathbb{E}_{a \sim \pi'} \left[\log(\pi'(a|s)) - \alpha' Q^{\pi}(s,a) + \log(Z^{\pi}(s)) \right] \\ \leq\ &\mathbb{E}_{a \sim \pi} \left[\log(\pi(a|s)) - \alpha' Q^{\pi}(s,a) + \log(Z^{\pi}(s)) \right]. \end{aligned}$$

Multiplication by $-\frac{1}{\alpha'}$ and subtraction of the $Z^{\pi}(s)$ term on both sides yields

$$\mathbb{E}_{a \sim \pi} \left[Q^{\pi}(s,a) - \frac{\alpha}{\gamma} \log(\pi(a|s)) \right] \leq \mathbb{E}_{a \sim \pi'} \left[Q^{\pi}(s,a) - \frac{\alpha}{\gamma} \log(\pi'(a|s)) \right].$$

With the Bellman equation we obtain

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s,a,s') + \gamma \mathbb{E}_{a \sim \pi} \left[Q^{\pi}(s,a) - \frac{\alpha}{\gamma} \log(\pi(a|s)) \right] \right] \\ &\leq \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s,a,s') + \gamma \mathbb{E}_{a \sim \pi'} \left[Q^{\pi}(s,a) - \frac{\alpha}{\gamma} \log(\pi'(a|s)) \right] \right] = \tilde{T}_{\pi'}(Q^{\pi}(s,a)). \end{aligned}$$

Recursive application of the soft Bellman operator on the right-hand side yields $Q^{\pi}(s,a) \leq Q^{\pi'}(s,a)$ by Corollary 2.8.2. □

Theorem 2.8.6 (Policy Iteration)**.** *Let M be an MDP with bounded rewards, compact state space and action spaces and let Π be a function class of stationary policies that admits a uniform bound on entropy term. Let Q_0 be any function and $\pi_0 \in \Pi$ be any policy. Iterative application of soft policy evaluation ($Q_{k+1} = \tilde{\mathcal{T}}^{\pi_k} Q_k$) and soft policy improvement (π_{k+1} is the minimizer as in Eq.* (2.7) *for all $s \in S$) leads to convergence $\pi_k \to \pi^*$ and $Q_k \to Q^{\pi *}$.*

Proof. Analogous to Theorem 2.5.2. □

For meeting the requirement of bounded entropy we state the following theorem.

Theorem 2.8.7. *For a continuous random variable Z with density f supported on $B := [a_{n_i}, b_{n_i}]^{|n|}$ the entropy is bounded $|\mathcal{H}(Z)| \leq |\log(\mathrm{vol}(B))|$.*

Proof. Since $x \mapsto |\log(x)|$ is concave, we can apply Jensen's inequality to find

$$\begin{aligned} |\mathcal{H}(Z)| &= \left| \int_B -\log(f(x)) dF(x) \right| = \left| \int_B \log\left(\frac{1}{f(x)}\right) dF(x) \right| \leq \int_B \left| \log\left(\frac{1}{f(x)}\right) \right| dF(x) \\ &\leq \left| \log\left(\int_B \frac{1}{f(x)} dF(x) \right) \right| = \left| \log\left(\int_B \frac{1}{f(x)} f(x) dx \right) \right| = |\log(\mathrm{vol}(B))|. \end{aligned}$$

□

2.8.2 Implementation

In the following we assume, that the action space is given by $[a_{n_i}, b_{n_i}]^{|n|}$. In practice $s \mapsto \pi_\phi(\cdot, s)$ is realized as a neural network that predicts mean $\mu \in \mathbb{R}^n$ and variance $\Sigma \in \mathbb{R}^n$ of a multivariate Gaussian distribution, the policy then samples an action from that distribution and applies a squash function such as $s : x_i \mapsto \frac{b_i - a_i}{2} \tanh(x_i) + a_i + 1$ which ensures the action bounds, see [Haa+18, Appendix C] for details.

The soft Q-evaluation objective is analogously defined as for DDPG via

$$L(\theta) := \mathbb{E}_{(s,a)\sim D}\left[\frac{1}{2}(\tilde{T}^\pi Q_{\theta^-}(s,a) - Q_\theta(s,a))^2\right],$$

where the empirical soft Bellman evaluation operator $\tilde{T}^\pi$ given a transition (s, a, r, s') and a sample $a' \sim \pi(\cdot|s')$ is defined as

$$\tilde{T}^\pi Q(s,a) := \begin{cases} r, & \text{if } s' \text{ is terminal,} \\ r + \gamma\left(Q(s', a') - \frac{\alpha}{\gamma}\log \pi(a'|s')\right), & \text{otherwise.} \end{cases}$$

Since the $Z^\pi(s)$ term in Eq. (2.8) does not depend on the parameters of the new policy π nor on a the term can be ignored for the optimization, thus a theoretical loss function for policy improvement could be given as

$$L(\phi) := \mathbb{E}_{s\sim D, a\sim\pi'_\phi}\left[\frac{\alpha}{\gamma}\log(\pi'_\phi(a|s)) - Q^\pi(s,a)\right],$$

where the multiplication of both terms with the constant $\frac{\gamma}{\alpha}$ does not alter the objective either. In order to be able to estimate the gradient of that objective the authors of SAC propose to apply a reparametrization trick: Instead of sampling $a \sim \mathcal{N}(\mu(s), \Sigma(s))$, the action is obtained via ("$\cdot$" denotes element-wise multiplication)

$$a = f(s, \epsilon) := \mu(s) + \epsilon \cdot \Sigma(s), \qquad \epsilon \sim \mathcal{N}(0, 1),$$

which is also distributed according to $\mathcal{N}(\mu, \Sigma)$. Thus, the objective can be expressed as

$$L(\phi) = \mathbb{E}_{s\sim D, \epsilon\sim\mathcal{N}(0,1)}\left[\frac{\alpha}{\gamma}\log(\pi'_\phi(f_\phi(s,\epsilon)|s)) - Q^\pi(s, f_\phi(s,\epsilon))\right].$$

Remark 2.8.8. Modern deep learning frameworks such as PyTorch do not require to manually implement the reparametrization trick, in PyTorch for example many distributions provide a method `rsample()` that implicitly tracks gradients.

So far we have described everything needed to implement SAC with a constant temperature α. In Algorithm 2.5 we generally write α instead of $\frac{\alpha}{\gamma}$ without loss of generality. In the refined version of SAC as given in Algorithm 2.5 the authors further incorporate clipped double Q-learning as in TD3 to reduce overestimation bias and additionally propose a way to automatically tune the entropy temperature.

Algorithm 2.5 SAC

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: stochastic $\pi_\phi(\cdot \mid \cdot)$ approximating optimal policy $\pi^*(\cdot \mid \cdot)$
Parameters: learning rates η_c, η_a, η_α, entropy temperature α, target entropy $\overline{H}$, target update parameters C and τ, batch size n, optional decay schedules for learning rates, replay buffer capacity, neural network architectures

```
1  Initialize $\pi_\phi : \mathcal{S} \to \mathcal{P}(A)$ with random weights $\phi$
2  Initialize $Q_{\theta_1}, Q_{\theta_2} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with random weights $\theta_1$ and $\theta_2$
3  Initialize $Q_{\theta_1^-}, Q_{\theta_2^-} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with weights $\theta_1^- = \theta_1$ and $\theta_2^- = \theta_2$
4  Initialize replay-buffer $D$ of given capacity
5  Initialize stochastic gradient descent optimizers $\mathrm{opt}_{\theta_1}$, $\mathrm{opt}_{\theta_2}$, $\mathrm{opt}_\phi$, $\mathrm{opt}_\alpha$
6  Sample initial state $s$ from environment
7  for number of training steps do
8      Optionally update $\eta_c$, $\eta_a$ according to decay schedules
9      Sample $a \sim \pi_\phi(s)$
10     Execute action $a$ in environment, observe next state $s'$ and reward $r$
11     Store $(s, a, r, s')$ in $D$
12     Sample random mini-batch of transitions $(s_i, a_i, r_i, s_i')_{i \le n}$ from $D$
13     Sample actions $a_i' \sim \pi_\phi(s_i')$ with densities $\pi_\phi(a_i'|s_i')$ for $i \in \{1, \dots, n\}$
14     $y_i \leftarrow \begin{cases} r_i & \text{if } s_i' \text{ is } \textit{terminal}\text{, otherwise} \\ r_i + \gamma \left( \min_{j \in \{1,2\}} Q_{\theta_j^-}(s_i', a_i') - \alpha \log \pi_\phi(a_i'|s_i') \right) \end{cases}$ for $i \in \{1, \dots, n\}$
15     $L(\theta_j) \leftarrow \frac{1}{n} \sum_{i=1}^n (y_i - Q_{\theta_j}(s_i, a_i))^2$ for $j \in \{1, 2\}$
16     Sample actions $\tilde{a}_i \sim \pi_\phi(s_i)$ with densities $\pi_\phi(\tilde{a}_i|s_i)$ for $i \in \{1, \dots, n\}$
17     $L(\phi) \leftarrow \frac{1}{n} \sum_{i=1}^n \left( \alpha \log \pi_\phi(\tilde{a}_i|s_i) - \min_{j \in \{1,2\}} Q_{\theta_j}(s_i, \tilde{a}_i) \right)$
18     $L(\alpha) \leftarrow -\frac{1}{n} \sum_{i=1}^n \alpha(\pi_\phi(\tilde{a}_i|s_i) + \overline{H})$
19     $\theta_j \leftarrow \theta_j - \mathrm{opt}_{\theta_j}.\mathrm{step}(\eta_c, \nabla L(\theta_j))$ for $j \in \{1, 2\}$
20     $\phi \leftarrow \phi - \mathrm{opt}_\phi.\mathrm{step}(\eta_a, \nabla L(\phi))$
21     $\alpha \leftarrow \alpha - \mathrm{opt}_\alpha.\mathrm{step}(\eta_\alpha, \nabla L(\alpha))$
22     Every $C$ steps $\theta_j^- \leftarrow \tau\theta_j + (1 - \tau)\theta_j^-$ for $j \in \{1, 2\}$
23     if $s'$ is terminal or truncation condition is true then
24         Sample new initial state $s$ from environment.
25     else $s \leftarrow s'$.
26 end
```

For automatic entropy tuning the authors introduce, given a minimum average target entropy $\overline{H}$, an additional constraint on the reinforcement learning problem essentially demanding that the average entropy of the policy is bigger than $\overline{H}$. Since their definition is only well posed for finite horizon problems, we are not giving details here and refer to [Haa+19, Sec. 5]. The loss function they optimize for can be stated as

$$L(\alpha) := \mathbb{E}_{s \sim D, a \sim \pi(\cdot, s)} \left[\alpha(-\log(\pi(a|s)) - \overline{H}) \right],$$

which we see as a heuristic that works well empirically. To give a motivation, the objective is equivalent to $\alpha(\mathbb{E}_{s\sim D}[\mathcal{H}(\pi(\cdot,s))] - \overline{H})$, which means if the average entropy of the current policy is bigger than the target entropy the objective is to make α smaller and vice versa, always giving more weight to the entropy when it is too low thus encouraging more exploration. To make that process fit in the theoretical picture of soft Q-iteration, the step of modifying the temperature could be seen as third component after soft policy evaluation and soft policy improvement, although we note that the influence on convergence even in an idealized theoretical setting from that viewpoint remains open since the proof of Lemma 2.8.5 is broken by varying α.

2.9 Quantile Regression DQN

Distributional reinforcement learning via quantile regression was introduced in [Dab+18a] and showed stronger empirical performance than the preceding approach to approximate the return distribution via categorical distributions as introduced in [BDM17] that started the field of distributional reinforcement learning. Approximating the quantile function of the return distribution $Z(s,a)$ at m discrete points has the advantage that the support of the function is always $[0,1]$ compared to a direct approximation of the return distribution at m points which depending on the pre-specified support boundaries $[V_{\min}, V_{\max}]$ might not always allow for efficient approximation, especially since the actual domain of the return distribution could greatly vary between different state-action pairs. The results of this section are from [Row+23], [BDR23, Chap. 5] and the original paper [Dab+18a].

2.9.1 Quantile Dynamic Programming

Let $\theta_i : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ parametrize the τ_i quantile of the return distribution $Z(s,a)$, for $i\in\{0,\dots,m\}$ such that τ_i are equally spaced in $[0,1]$. Then the return distribution can be approximated via

$$Z_\theta(s,a) := \sum_{i=1}^{m} \frac{1}{m}\delta_{\theta_i(s,a)},$$

where δ_z denotes the Dirac measure and the action value function is approximated via

$$Q_\theta(s,a) = \frac{1}{m}\sum_{i=1}^{m}\theta_i(s,a).$$

Definition 2.9.1. Let $m\in\mathbb{N}$. The *m-quantile representation* is defined as

$$\mathcal{F}_{Q,m} = \left\{\frac{1}{m}\sum_{i=1}^{m}\delta_{\theta_i} : \theta_i\in\mathbb{R}\right\}.$$

Clearly $\mathcal{F}_{Q,m}\subseteq\mathcal{P}(\mathbb{R})^G$, however this space is not closed under the application of distributional Bellman operators, hence an additional projection step is needed.

Definition 2.9.2. For $\nu \in \mathcal{P}(\mathbb{R})$ and $\lambda \in [0,1]$ a *projection operator* is a mapping $\Pi_Q^\lambda : \mathcal{P}(\mathbb{R}) \to \mathcal{F}_{Q,m}$ such that the 1-Wasserstein distance $w_1(\Pi_Q^\lambda(\nu), \nu)$ is minimal.

Lemma 2.9.3. *Let $\nu \in \mathcal{P}(\mathbb{R})$ and $\lambda \in [0,1]$. Then a projection operator is given by $\Pi_Q(\nu) = \frac{1}{m}\sum_{i=1}^m \theta_i$ with*

$$\theta_i = (1-\lambda)F_\nu^{-1}\left(\frac{2i-1}{2m}\right) + \lambda \overline{F}_\nu^{-1}\left(\frac{2i-1}{2m}\right) \quad i = 1, \dots, m,$$

where $F_\nu^{-1}(\tau) = \inf\{y : F_\nu(y) \geq \tau\}$ denotes the generalized inverse and $\overline{F}_\nu^{-1}(\tau) := \inf\{y : F_\nu(y) > \tau\}$.

Proof. We refer to the proof of [BDR23, Proposition 5.15], which can be adapted to match the generalized projection formulation in [Row+23, Formula (12)]. □

The choice of the parameter λ only matters in cases where $F_\nu^{-1}(x) \neq \overline{F}_\nu^{-1}(x)$, i.e., in cases where the quantile does not have a unique preimage, thus τ interpolates between the left and the right end of the constant interval.

Theorem 2.9.4. *Let $\lambda \in [0,1]^{G\times m}$, let $\tau_i := \frac{2i-1}{2m}$ as in the previous lemma and let*

$$\Pi^\lambda \nu(s,a) := \frac{1}{m}\sum_{i=1}^m \delta_{(1-\lambda(s,a,i))F^{-1}_{\nu(s,a)}(\tau_i) + \lambda(s,a,i)\overline{F}^{-1}_{\nu(s,a)}(\tau_i)}.$$

Then $\Pi^\lambda \mathcal{T}^\pi : (\mathcal{F}_{Q,m})^G \to (\mathcal{F}_{Q,m})^G$ is a γ-contraction with respect to $\overline{w}_\infty$. Thus, $\Pi^\lambda \mathcal{T}^\pi$ has a unique fixed point ν_λ^π in $\mathcal{F}_{Q,m}^G$ and for any initial $\nu_0 \in \mathcal{F}_{Q,m}^G$ the sequence $\nu_{k+1} = \Pi^\lambda \mathcal{T}^\pi \nu_k$ converges to the fixed point $\overline{w}_\infty(\nu_k, \nu_\lambda^\pi) \to 0$ as $k \to \infty$.

Proof. Since $\mathcal{T}^\pi$ is a contraction in $\overline{w}_\infty$ by Theorem 1.5.12 it suffices to show that Π^λ is a non expansion in $\overline{w}_\infty$. For the calculation we refer to [Row+23, Appendix A.1]. The rest follows in the usual way from Banach's fixed point theorem. □

2.9.2 Quantile Q-learning

In order to learn unbiased estimates of quantiles, *quantile regression* may be used. Let $\nu \in \mathcal{P}(\mathbb{R})$, $\tau \in (0,1)$ be a quantile value and $\theta \in \mathbb{R}$ the associated quantile i.e., $F(q) \geq \tau$ for the cumulative distribution F. Then θ is the minimizer of the loss

$$L(v) = \mathbb{E}_{Z\sim\nu}\left[(\tau 1_{\{Z-v>0\}} + (1-\tau)1_{\{Z-v<0\}})|Z-v|\right].$$

With $\rho_\tau(u) := u(\tau - 1_{\{u<0\}})$ the loss can be compactly rewritten as $\mathbb{E}_{Z\sim\nu}[\rho_\tau(Z-v)]$. Now let $Z(s,a) = \sum_{t=0}^\infty \gamma^t R_t \sim \nu$ and let $\theta_i(s,a)$ denote the respective τ_i-quantile with $\tau_i = \frac{2i-1}{2m}$ for $i \in \{1,\dots,m\}$. Then with Lemma 1.5.6 the loss can be written as

$$\begin{aligned} L(\theta_i(s,a)) &= \mathbb{E}_{Z\sim\nu}\left[\rho_{\tau_i}\left(Z(s,a) - \theta_i(s,a)\right)\right] \\ &= \mathbb{E}_{Z\sim\nu}\left[\rho_{\tau_i}\left(r(s,a,S') + \gamma Z(S',A') - \theta_i(s,a)\right)\right], \ A' \sim \pi(S'), \ S' \sim p(\cdot, s, a) \end{aligned}$$

$$\approx \frac{1}{m}\sum_{j=1}^{m} \rho_{\tau_i}\left(r(s,a,S') + \gamma\theta_j(S',A') - \theta_i(s,a)\right).$$

In case of a distributional optimality operator $A' \sim \pi(S')$ is exchanged for $A' \sim \mathcal{G}(\nu)$ for a greedy selection rule $\mathcal{G}$. Now everything is in place to state the quantile regression adaption of DQN, as in the next section, but before we want to shortly derive the tabular version of quantile Q-learning: Taking the negative gradient in the last line yields

$$\frac{1}{m}\sum_{j=1}^{m}\left(\tau_i - 1_{\{r(s,a,S')+\gamma\theta_j(S',A')-\theta_i(s,a)>0\}}\right)$$

and thus motivating a tabular update rule of the form

$$\theta_i^{k+1}(s,a) = \theta_i^k(s,a) + \frac{\alpha}{m}\sum_{j=1}^{m}\left(\tau_i - 1_{\{r(s,a,S')+\gamma\theta_j^k(S',A')-\theta_i^k(s,a)>0\}}\right)$$

for all $i \in \{1,\dots,m\}$ and with learning rate α. In the case of quantile temporal difference learning (which is interested in estimating the value-distribution compared to the action value distribution setting we formulated here), the convergence in the tabular setting has been established in [Row+23] (the operators for the value function case are defined analogously):

Theorem 2.9.5. *Let $\theta^0 \in \mathbb{R}^{S\times m}$ be arbitrary and $(\theta^k)_{k=0}^{\infty}$ be the sequence defined by*

$$\theta_i(s)^{k+1} = \theta_i(s)^k + \frac{\alpha_k}{m}\sum_{j=1}^{m}\left(\tau_i - 1_{\{r(s,a,S')+\gamma\theta_j^k(S')-\theta_i^k(s)>0\}}\right),$$

with non-negative step sizes satisfying the condition

$$\sum_{k=0}^{\infty}\alpha_k = \infty, \quad \alpha_k = o(1/\log(k)).$$

Then $(\theta^k)_{k=0}^{\infty}$ converges almost surely to the fixed points of the projected distributional Bellman operators $\{\Pi^\lambda \mathcal{T}^\pi : \lambda \in [0,1]^{S\times m}\}$, i.e.,

$$\inf_{\lambda\in[0,1]^{S\times m}} |\theta^k - \hat{\theta}_\lambda^\pi|_\infty \to 0$$

with probability 1, where $\hat{\theta}_\lambda^\pi$ denotes the fixed point of $\Pi^\lambda\mathcal{T}^\pi$.

The above theorem only establishes convergence of quantile temporal difference learning, i.e., policy evaluation. In the control case, even with the assumption that a unique optimal policy exists, no proof is known at the time of this thesis, but for categorical Q-learning convergence has been shown, by using the mean-preserving nature of the categorical Bellman operator, see [BDR23, Table 5.1, p. 217, Corollary 7.10] and [Row+18, Theorem 2].

2.9.3 QR-DQN

QR-DQN builds upon DQN, by making the following changes:

1. Change the net architecture to approximate the quantile-function via a neural net $\theta : S \times A \to \mathbb{R}^m$ with weights ϕ instead of the action value function.

2. The Huber loss is replaced by a quantile Huber loss

$$\rho_\tau^\kappa(u) := |\tau - 1_{\{u<0\}}|\mathcal{L}_\kappa(u),$$

   where $\mathcal{L}_\kappa(u)$ is the Huber loss. For $\kappa = 0$ this is the quantile regression loss as before. The authors of QR-DQN empirically determined that $\tau = 1$ works best and argue for the modification of the quantile regression loss with the Huber loss for the continuity around 0. The overall loss for a single sample transition (s, a, s', r) with a greedy action a' in s' becomes

$$L(\phi) = \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{1}{m} \rho_{\tau_i}^\kappa \left(r(s, a, s') + \gamma\theta_j(s', a') - \theta_i(s, a, \phi) \right).$$

3. The action selection works analogously to DQN by recovering the action value function $Q(s, a, \phi) = \frac{1}{m} \sum_{i=1}^{m} \theta_i(s, a, \phi)$.

The full pseudocode is given in Algorithm 2.6.

Empirically, QR-DQN shows strong performance, however there are two theoretical issues we wish to highlight:

1. The distributional Bellman optimality operators do not necessarily converge to the action value distribution of an optimal policy, not even in an idealized tabular setting as mentioned in Remark 1.5.15.

2. There is no constraint in the neural network architecture that enforces the monotonicity of the quantiles. To address this, there exist multiple approaches in follow-up work, see the works referenced in [Row+23, Sec. 7, Paragraph Quantiles in reinforcement learning].

2.10 Implicit Quantile Networks

While QR-DQN only learns a fixed number of quantiles, in [Dab+18b] an extension of QR-DQN was proposed that aims to learn the whole return distribution. In order to do so, the neural network architecture is changed to learn a quantile function

$$\theta : S \times A \times [0, 1] \to \mathbb{R},$$

such that $\theta(s, a, \tau)$ is the τ-quantile of the return distribution of (s, a), where τ is any quantile fraction of $[0, 1]$. The loss function is identical to the QR-DQN loss, but instead of the fixed τ-fractions, for the target m' quantile-fractions $\tau' \sim \mathcal{U}([0, 1])$ are sampled

Algorithm 2.6 QR-DQN

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: approximation $\theta_\phi(\cdot,\cdot)$ of action value distribution $\nu^{\pi^*}(\cdot,\cdot)$
Parameters: number of quantiles m, learning rate α, exploration rate ε, target update parameters C and τ, batch size n, optional decay schedules for ε or α, replay buffer capacity, neural network architecture

```
1  Initialize $\theta_\phi : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|\cdot m}$ with random weights $\phi$, let $\theta_\phi^k(s,a) := \theta_\phi(s)[a][k]$
2  Initialize $\theta_{\phi^-} : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|\cdot m}$ with weights $\phi^- = \phi$
3  Initialize replay-buffer $D$ of given capacity
4  Initialize stochastic gradient descent optimizer opt
5  Sample initial state $s$ from environment
6  for number of training steps do
7      Optionally update $\epsilon$ or $\alpha$ according to decay schedules
8      $a \leftarrow \begin{cases} \operatorname{argmax}_a \frac{1}{m}\sum_{i=1}^m \theta_\phi^i(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
9      Execute action $a$ in environment, observe next state $s'$ and reward $r$
10     Store $(s,a,r,s')$ in $D$
11     Sample random mini-batch of transitions $(s_i,a_i,r_i,s'_i)_{i\leq n}$ from $D$
12     $a'_i \leftarrow \operatorname{argmax}_{a'} \frac{1}{m}\sum_{i=1}^m \theta_\phi^i(s_i,a')$ for $i \in \{1,\dots,n\}$
13     $y_{ik} \leftarrow \begin{cases} r_i & \text{if } s'_i \text{ is } \textit{terminal} \\ r_i + \gamma\theta_{\phi^-}^k(s'_i,a'_i) & \text{otherwise} \end{cases}$ for $i \in \{1,\dots,n\}$
14     $L(\phi) \leftarrow \frac{1}{n}\sum_{i=1}^n \sum_{j=1}^m \frac{1}{m}\sum_{k=1}^m \rho_{\tau_j}^\kappa\left(y_{ik} - \theta_\phi^j(s,a)\right)$
15     $\phi \leftarrow \phi - \operatorname{opt.step}(\alpha, \nabla L(\phi))$
16     Every $C$ steps $\phi^- \leftarrow \tau\phi + (1-\tau)\phi^-$
17     if $s'$ is terminal or truncation condition is true then
18         Sample new initial state $s$ from environment
19     else $s \leftarrow s'$
20 end
```

while for the current quantiles m quantiles $\tau \sim \mathcal{U}([0,1])$ are sampled, where m and m' are hyperparameters. The full loss for a single sample transition (s,a,s',r) with a greedy action a' in s' is given by

$$L(\phi) = \sum_{i=1}^{m}\sum_{j=1}^{m'} \frac{1}{m'}\rho_{\tau_i}^\kappa\left(r(s,a,s') + \gamma\theta(s',a',\tau'_j) - \theta(s,a,\tau_i,\phi)\right).$$

For the greedy action selection, yet another batch of $\tilde{m}$ quantile fractions $\tilde{\tau} \sim \mathcal{U}([0,1])$ is sampled in order to recover the action value function via

$$Q(s,a) = \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} \theta(s,a,\tilde{\tau}).$$

The full pseudocode is given in Algorithm 2.7.

Algorithm 2.7 IQN

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: approximation $\theta_\phi(\cdot,\cdot)$ of action value distribution $\nu^{\pi^*}(\cdot,\cdot)$
Parameters: number of quantile samples m, m', $\tilde{m}$, learning rate α, exploration rate ε, target update parameters C and τ, batch size n, optional decay schedules for ε or α, replay buffer capacity, neural network architecture

```
 1 Initialize $\theta_\phi : \mathcal{S} \times [0,1] \to \mathbb{R}^{|\mathcal{A}|}$ with random weights $\phi$, let $\theta_\phi(s,a,\tau_i) := \theta_\phi(s,\tau_i)[a]$
 2 Initialize $\theta_{\phi^-} : \mathcal{S} \times [0,1] \to \mathbb{R}^{|\mathcal{A}|}$ with weights $\phi^- = \phi$
 3 Initialize replay-buffer $D$ of given capacity
 4 Initialize gradient descent optimizer opt
 5 Sample initial state $s$ from environment
 6 for number of training steps do
 7     Optionally update $\epsilon$ or $\alpha$ according to decay schedules
 8     Sample $\tau_i \sim \mathcal{U}([0,1])$ for $i \in \{1,\dots,\tilde{m}\}$
 9     $a \leftarrow \begin{cases} \text{argmax}_a \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} \theta_\phi(s,a,\tau_i) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
10     Execute action $a$ in environment, observe next state $s'$ and reward $r$
11     Store $(s,a,r,s')$ in $D$
12     Sample random mini-batch of transitions $(s_i,a_i,r_i,s'_i)_{i\le n}$ from $D$
13     $a'_i \leftarrow \text{argmax}_{a'} \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} \theta^i_\phi(s_i,a')$ for $i \in \{1,\dots,n\}$
14     Sample $\tau'_j \sim \mathcal{U}([0,1])$ for $j \in \{1,\dots,m'\}$ and $\tau_j \sim \mathcal{U}([0,1])$ for $j \in \{1,\dots,m\}$
15     $y_{ik} \leftarrow \begin{cases} r_i & \text{if } s'_i \text{ is } \textit{terminal} \\ r_i + \gamma\theta_{\phi^-}(s'_i,a'_i,\tau'_k) & \text{otherwise} \end{cases}$ for $i \in \{1,\dots,n\}$
16     $L(\phi) \leftarrow \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{m'}\sum_{k=1}^{m'}\rho^\kappa_{\tau_j}\left(y_{ik} - \theta_\phi(s,a,\tau_j)\right)$
17     $\phi \leftarrow \text{opt}.\text{step}(\alpha, \nabla L(\phi))$
18     Every $C$ steps $\phi^- \leftarrow \tau\phi + (1-\tau)\phi^-$
19     if $s'$ is terminal or truncation condition is true then
20         Sample new initial state $s$ from environment
21     else $s \leftarrow s'$
22 end
```

2.11 Truncated Quantile SAC

The pioneering work of distributional reinforcement learning has originally been published based on modifications of the DQN algorithm, however the distributional adaptions straight forwardly carry over to actor-critic algorithms such as DDPG and SAC by replacing the value critic with a distributional critic. In the literature the distributional adaptions were mostly published in combination with other changes as in [Bar+18] for DDPG and in [Kuz+20] for SAC. In the following we present a distributional adaption of SAC as originally proposed in [Kuz+20] together with a distributional method to reduce overestimation bias proposed in the same paper.

2.11.1 Distributional SAC

The action value network of SAC is replaced by a quantile network. Analogously as for the action value function, a soft distributional Bellman operator can be defined as

$$(\tilde{\mathcal{T}}_\pi \nu)(s,a)(B) := \int_S \int_{A_{s'}} b^{\#}_{r(s,a,s')-\alpha\log(\pi(s',a')),\gamma}\nu(s',a')(B)\ \pi(da',s')p(ds',s,a).$$

The contraction properties and the subsequent fixed point theory for the distributional policy evaluation case carry over for this modified operator. From the quantile regression perspective this leads to the critic loss

$$L(\phi) = \sum_{i=1}^{m}\sum_{j=1}^{m}\frac{1}{m}\rho^{\kappa}_{\hat{\tau}_i}\left(r(s,a,s') + \gamma\left(\theta_j(s',a') - \alpha\log(\pi(a',s'))\right) - \theta_i(s,a,\phi)\right)$$

for a single sample transition (s,a,s',r) with $a' \sim \pi(s')$. Double Q-learning as used by SAC to reduce the overestimation bias can be adapted by recovering the action values and determining the index $k(s',a')$ of the quantile network that has the minimum value

$$k(s',a') = \operatorname{argmin}_k \frac{1}{m}\sum_{i=1}^{m}\theta^i_{\psi_k^-}(s',a').$$

The policy improvement step works identically as before by recovering the action value function $Q(s,a)$ from the quantiles.

2.11.2 TQC

In [Kuz+20] an alternative approach to clipped double Q-learning was introduced for reducing the overestimation bias by making use of the action value distribution. In an ensemble of n distributional critics that approximate the quantile function at m locations as in QR-DQN, the quantiles as predicted by the target networks with parameters ψ_l^- are pooled in a set

$$\mathcal{Z}(s',a') := \{\theta^j_{\psi_l^-}(s',a') : j \in \{1,\dots,m\}, l \in \{1,\dots,n\}\}.$$

Let $z_j(s', a')$ for $j \in \{1, \dots, kn\}$ be the kn smallest entries of $\mathcal{Z}$ in ascending order, where $k := (m - d)$ and d is a hyperparameter that controls how many quantiles from each network are dropped. Then the targets are calculated via

$$y_j(s, a) := r(s, a, s') + \gamma(z_j(s', a') - \alpha \log \pi(a'|s')),$$

resulting in critic losses

$$L(\psi_l) = \sum_{i=1}^{m} \sum_{j=1}^{kn} \frac{1}{kn} \rho^{\kappa}_{\tau_i} \left(y_j - \theta^i_{\psi_l}(s, a) \right), \quad l \in \{1, \dots, n\}.$$

On the actor side the authors propose to average over the Q-networks as opposed to SAC which takes the minimum over all Q-networks. The full pseudocode of TQC is given in Algorithm 2.8.

2.12 Neural Network Architecture

In this section we give a detailed description of how the neural networks are realized we use for our experiments in the last chapter.

2.12.1 Critic-only Networks

When the elements of the state space are low dimensional vectors, the action value function is parametrized by a fully connected neural network $f : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ via

$$Q(s, a) = f(s)^a.$$

In the case of high dimensional observations such as images, a convolutional neural network (CNN) $\Psi : S \to \mathbb{R}^d$ is used to implicitly learn an embedding into a d-dimensional feature space $\mathbb{R}^d$. In that case the action value function is parametrized via

$$Q(s, a) = f(\Psi(s))^a,$$

with a fully connected $f : \mathbb{R}^d \to \mathbb{R}^{\dim \mathcal{A}}$. If furthermore observations are a mix between images s_1 and low dimensional vectors s_2, i.e., $s = (s_1, s_2)$, $s_1 \in S_1$, $s_2 \in S_2$ we concatenate the image embedding and the vector and apply f on top

$$Q(s, a) = f(\Psi(s_1), s_2)^a,$$

with $f : \mathbb{R}^d \times S_2 \to \mathbb{R}^{\dim \mathcal{A}}$ adapted accordingly. Since the feature embedding network is integrated in the action value parametrization, learning a good embedding works implicitly via backpropagation to the convolutional layers as part of Q-learning.

For QR-DQN the neural network architecture works analogously, in case of IQN the authors propose to adapt the architecture for the quantile input as

$$\theta(s, a, \tau) = f((\Psi(s_1), s_2) \odot \Phi(\tau))^a,$$

Algorithm 2.8 TQC

Input: environment interface with specification of spaces $\mathcal{S}$ and $\mathcal{A}$.
Output: stochastic $\pi_\phi(\cdot \mid \cdot)$ approximating optimal policy $\pi^*(\cdot \mid \cdot)$
Parameters: number of quantiles m, number of critics n, number of quantiles to drop d, learning rates η_c, η_a, η_α, entropy temperature α, target entropy $\overline{H}$, target update parameters C and τ, batch size b, optional decay schedules for learning rates, replay buffer capacity, neural network architectures

```
1  **Initialize** $\pi_\phi : \mathcal{S} \to \mathcal{P}(A)$ with random weights $\phi$
2  **Initialize** $\theta_{\psi_1}, \ldots, \theta_{\psi_n} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ with random weights $\psi_1, \ldots, \psi_n$
3  **Initialize** $\theta_{\psi_1^-}, \ldots, \theta_{\psi_n^-} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ with weights $\psi_1^- = \psi_1, \ldots, \psi_n^- = \psi_n$
4  **Initialize** replay-buffer $D$ of given capacity
5  **Initialize** stochastic gradient descent optimizers $\mathrm{opt}_{\psi_1}$, $\mathrm{opt}_{\psi_2}$, $\mathrm{opt}_\phi$, $\mathrm{opt}_\alpha$
6  Sample initial state $s$ from environment
7  **for** number of training steps **do**
8      Optionally update $\eta_c$, $\eta_a$ according to decay schedules
9      Sample $a \sim \pi_\phi(s)$
10     Execute action $a$ in environment, observe next state $s'$ and reward $r$
11     Store $(s, a, r, s')$ in $D$
12     Sample random mini-batch of transitions $(s_i, a_i, r_i, s_i')_{i \le b}$ from $D$
13     Sample actions $a_i' \sim \pi_\phi(s_i')$ with densities $\pi_\phi(a_i'|s_i')$ for $i \in \{1, \ldots, b\}$
14     Enumerate $z_k \in \mathcal{Z}(s_i', a_i') := \{\theta^k_{\psi_l^-}(s_i', a_i') : k \in \{1, \ldots, m\}, l \in \{1, \ldots, n\}\}$ in
       ascending order for $i \in \{1, \ldots, b\}$
15     $y_{ik} \leftarrow \begin{cases} r_i & \text{if } s_i' \text{ is } \textit{terminal}\text{, otherwise} \\ r_i + \gamma\left(z_k - \alpha \log \pi_\phi(a_i'|s_i')\right) \end{cases}$ for $i \in \{1, \ldots, b\}$
16     $L(\psi_l) = \frac{1}{b} \sum_{i=1}^{b} \sum_{j=1}^{m} \sum_{k=1}^{dn} \frac{1}{dn} \rho^\kappa_{\tau_i}\left(y_{ik} - \theta^j_{\psi_l}(s, a)\right)$ for $l \in \{1, \ldots, n\}$
17     Sample actions $\tilde{a}_i \sim \pi_\phi(s_i)$ with densities $\pi_\phi(\tilde{a}_i|s_i)$ for $i \in \{1, \ldots, b\}$
18     $L(\phi) \leftarrow \frac{1}{b} \sum_{i=1}^{b} \left(\alpha \log \pi_\phi(\tilde{a}_i|s_i) - \frac{1}{mn} \sum_{k,l=1}^{m,n} \theta^k_{\psi_l}(s_i, \tilde{a}_i)\right)$
19     $L(\alpha) \leftarrow -\frac{1}{b} \sum_{i=1}^{b} \alpha(\pi_\phi(\tilde{a}_i|s_i) + \overline{H})$
20     $\psi_l \leftarrow \psi_l - \mathrm{opt}_{\psi_l}.\mathrm{step}(\eta_c, \nabla L(\psi_l))$ for $l \in \{1, \ldots, n\}$
21     $\phi \leftarrow \phi - \mathrm{opt}_\phi.\mathrm{step}(\eta_a, \nabla L(\phi))$
22     $\alpha \leftarrow \alpha - \mathrm{opt}_\alpha.\mathrm{step}(\eta_\alpha, \nabla L(\alpha))$
23     Every $C$ steps $\psi_l^- \leftarrow \tau\psi_l + (1 - \tau)\psi_l^-$ for $l \in \{1, \ldots, n\}$
24     **if** $s'$ is *terminal* or *truncation condition* is true **then**
25         Sample new initial state $s$ from environment
26     **else** $s \leftarrow s'$
27 **end**
```

where $\odot$ denotes the element-wise product of vectors and $\Phi : [0, 1] \to \mathbb{R}^{d'}$ with $d' :=$

$d + \dim S_2$ is given by

$$\Phi(\tau)^j := \mathrm{ReLU}\left(\sum_{i=0}^{d'-1} \cos(\pi i \tau) w^{ij} + b^j\right),$$

where w^{ij} denotes the weights and b^j the bias of a linear layer.

2.12.2 Actor-Critic Networks

In the vector case policies (actor networks) are parametrized by fully connected feed forward networks $f : \mathcal{S} \to \Theta$ where Θ is a set of parameters generating the policy, e.g., in the deterministic policy case $\Theta = \mathcal{A}$, i.e., $f_\mu : \mathcal{S} \to \mathcal{A}$ or in the case of parametrizing normal distributions $f_\pi : \mathcal{S} \to \Theta = \{(\hat{\mu}, \hat{\sigma}) : \hat{\mu}, \hat{\sigma} \in \mathbb{R}^{\dim \mathcal{A}}\}$, thus for these two examples the policies are obtained via

$$\mu(s) = f_\mu(s), \quad \pi(\cdot|s) = \mathcal{N}(f_\pi(s)).$$

The critics are parametrized with fully connected networks $f_Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$$Q(s, a) = f_Q(s, a).$$

In the case of high dimensional observations (let $s = (s_1, s_2)$, $s_1 \in S_1$, $s_2 \in S_2$ be again a mixture between image states s_1 and vector states s_2) a convolutional feature extractor $\Psi : S_1 \to \mathbb{R}^d$ as in the previous section is used for both the actor and the critic, thus for actors as above we obtain

$$\mu(s) = f_\mu(\Psi(s_1), s_2), \quad \pi(\cdot|s) = \mathcal{N}(f_\pi(\Psi(s_1), s_2)),$$

with $f_\mu, f_\pi : \mathbb{R}^d \times S_2 \to \Theta$ and similarly for the critic

$$Q(s, a) = f_Q(\Psi(s_1), s_2, a),$$

with $f_Q : \mathbb{R}^d \times S_2 \times \mathcal{A} \to \mathbb{R}$. To save computation time, the feature extractor layers can be shared between the actor and critic networks by disabling backpropagation of the gradients to the feature extractor layers when updating the actor network.

2.12.3 Concrete Architecture

We realize a fully connected network $f : \mathbb{R}^n \to \mathbb{R}^m$ with a *number of hidden linear layers* d with a *number of hidden nodes* w each followed by a nonlinear activation function $\sigma : \mathbb{R}^w \to \mathbb{R}^w$ as

$$f(x) = f_{d+1} \circ \sigma \circ f_d \circ \cdots \circ \sigma \circ f_1 \circ \sigma \circ f_0(x),$$

where $f_0 : \mathbb{R}^n \to \mathbb{R}^w,\ x \mapsto A_0 x + b_0$, with a weight matrix $A_0 \in \mathbb{R}^{w \times n}$ and a bias weight vector $b_0 \in \mathbb{R}^w$ and analogously $f_i : \mathbb{R}^w \to \mathbb{R}^w$ for $i \in \{1, \ldots, d\}$ and $f_{d+1} : \mathbb{R}^w \to \mathbb{R}^m$.

For the feature extractor we follow the architecture of [Mni+15] for the convolutional layers design choices

$$\Psi(x) = f_e \circ \ell \circ \sigma \circ \Psi_d \circ \cdots \circ \sigma \circ \Psi_1 \circ \sigma \circ \Psi_0(x).$$

Each convolution layer (in the following the drop the layer index for readability) is determined by the *number of output channels* $o \in \mathbb{N}$, the *kernel size* $\underline{k} = (k_1, k_2) \in \mathbb{N}^2$ and the *stride* $\underline{s} = (s_1, s_2) \in \mathbb{N}^2$: For a stack of input images $I \in \mathbb{R}^{c \times h \times w}$ and $i \leq o$, $j \leq (h - k_1)/s_1 + 1$, $k \leq (w - k_2)/s_2 + 1$ the $[i, j, k]$ component of the output is given by

$$\Psi_{o,\underline{k},\underline{s}}(I)[i,j,k] = b_i + \sum_{l=0}^{c-1} \sum_{m=0}^{h/s_1} \sum_{n=0}^{w/s_2} K_{il}[j,k] I[l, j + m + s_1, k + n + s_2],$$

where $K_{il} \in \mathbb{R}^{k_1 \times k_2}$ is the convolutional kernel for output channel i and input channel l and b_i a bias weight for output channel i. Finally, $\ell : \mathbb{R}^{c \times h \times w} \to \mathbb{R}^{chw}$ flattens the stacked images into a vector and $f_e : \mathbb{R}^{chw} \to \mathbb{R}^e$ is a linear layer as above.

It is now possible to describe the actual realization of the architecture by giving the relevant hyperparameters:

- For the fully connected network, given input and output dimension the architecture above is uniquely determined by d and w.
- The feature extractor above, given the input dimensions is uniquely determined by the output dimension e and for each convolutional layer the specification of the parameters c, k and s. We always specify k and s by a single integer per convolutional layer that is used for both components and separate the value of each parameter for each convolutional layer by a comma, see Table 4.1.

We generally always use $\sigma = \text{ReLU}$ for the activation function.

3 Environment for Autonomous Driving

The design and implementation of the autonomous driving tasks described in this chapter has been partly done in collaboration with Tobias Kietreiber [Kie23].

3.1 Problem Formulation and Scope

For our treatment of autonomous driving we define the following control problems:

1. *Cruise control*: control the brake and throttle of the car in order to reach and maintain a given target velocity.
2. *Adaptive cruise control*: same as cruise control but additionally maintain a safe distance to other vehicles in front of the car.
3. *Lane keeping*: control the steering angle of the car in order to follow a given lane.
4. *Combined control*: combination of cruise control and lane keeping, i.e., the agent controls brake, throttle and steering angle in order to meet the cruise control and the lane keeping objectives.
5. *Adaptive combined control*: combination of adaptive cruise control and lane keeping.
6. *Obstacle avoidance*: control the steering angle of a vehicle at different velocities (the agent has no control of), in order to follow the current lane and change lane in case of an obstacle.
7. *Navigation:* combined control, but the agent has to follow navigation instructions additionally (left, right, straight, follow lane, change lane left, change lane right), thus junctions come in as an additional challenge. When reaching the end of a route the agent is supposed to stop.
8. *Adaptive Navigation:* same task as navigation, but with other traffic participants.

We do not consider the control of vehicles with a manual gearbox, since we believe that there is no point in explicitly exposing gear shift control to an autonomous agent, as automatic gear shift control has long been existent in non-autonomous vehicles. Furthermore, electric vehicles usually do not expose gear shift control.

Moreover, we do not comply with traffic lights or traffic signs in the scope of this work for the following reasons:

- Traffic lights are a challenge on their own to detect from camera input, requiring a sufficiently high resolution camera image. While this is no challenge in terms of the reinforcement learning formulation it is a computational challenge we do not want to pursue.

- Traffic signs such as stop signs, while easier to detect visually, often are not visible anymore for the agent at the desired stop point, thus each state of the agent would have to include a record back to the point in time the agent passed the sign or alternatively the state would need to be augmented with such information, which is a challenge in terms of the reinforcement learning formulation of the problem we consider out of scope.

3.2 Environment Setup

We use the CARLA open-source simulator for autonomous driving research introduced in [Dos+17] to set up our control problems. From the CARLA carpool we select a Tesla model 3 as the vehicle to be controlled, but nothing in our setup is tailored to that choice, so picking any other vehicle for training will work as well.

3.2.1 Time Steps and Synchrony

We choose the time between two environment steps to be 0.1 seconds of simulated time, to make a trade-off between reaction speed to environmental changes and computational effort. This also seems reasonable, as it is half the reaction time of a human, for more technical reasons for this choice see Section 3.5.

The agent environment interaction is *synchronous* in all of our experiments, i.e., when the agent sets an action, exactly 0.1 seconds of simulated time pass and no time passes in the simulator (simulation is frozen) until the agent sets the next action, thus the runtime performance of the agent does not have any influence on the Markov property of the environment. If the agent were to learn to control the car in reality it would be essential to ensure it is performant enough to keep the time between actions as close as possible to the specified length, or augment the state space with information about the time that has passed since the last action was taken.

3.2.2 Episode Setup

Cruise control problems are implemented in the CARLA map `Town06` as depicted in Fig. 3.1, as that map features long straight roads. Lane keeping, combined control and obstacle avoidance problems are all set up on the eight-shaped highway in `Town04` as depicted in Fig. 3.2, as the highway road geometry allows training straight road segments, left and right curves without the need to handle junctions (there are junctions, but we always opt to remain on the highway in the same direction). Navigation scenarios are implemented in `Town07`, a rural town with many junctions and with roads that partly lack lane markings, see Fig. 3.3.

Cruise Control

At the start of each episode the vehicle is placed on one end of the very long roads, as the vehicle approaches the other end of the road the target velocity is set to zero. The maximal episode time (as stated in the next section) is chosen in a way that the agent will not be able to collide at the other end of the road, even when going with full throttle all the time.

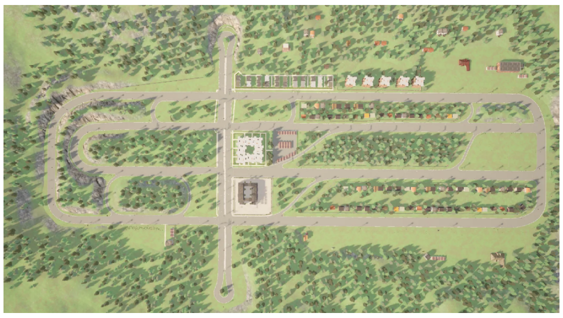

Figure 3.1: Aerial perspective of `Town06` [dev23].

During an episode the target velocities change randomly. For adaptive cruise control another vehicle is spawned ahead of the vehicle with a random probability in a random distance at the beginning of each episode with the throttle being randomly changed during the episode.

Lane Keeping, Combined Control and Obstacle Avoidance

At the start of each episode the vehicle is spawned at one randomly selected position out of 6 fixed positions that are roughly evenly spread across the highway, all in the same lane with the vehicle always facing in the same direction. This ensures that the agent also gets to explore curves early as opposed to at a later point after it learned to drive straight fist.

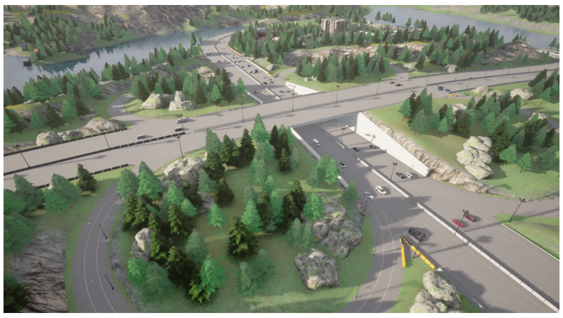

Figure 3.2: Partial aerial perspective of `Town04` showing part of the eight shaped highway, the bridge forms the center of the eight [dev23].

In case of lane keeping and obstacle avoidance scenarios, the throttle of the car is changed randomly to challenge the agent to control the steering at different velocities, in case of the combined control scenarios the target velocity is changed randomly during an episode.

The adaptive combined scenario similarly as the adaptive cruise control scenario features a front vehicle that appears and disappears randomly during the episodes in random distance and is controlled by a CARLA autopilot with velocities that randomly change. In case of the obstacle avoidance scenario vehicles are spawned randomly in front of the vehicle that do not move at all. After the car has passed the obstacle, the obstacle is removed from the road.

Navigation

At the start of each episode the vehicle is placed on a random starting position on the map, sampling from a list of possible spawn positions happens uniformly.

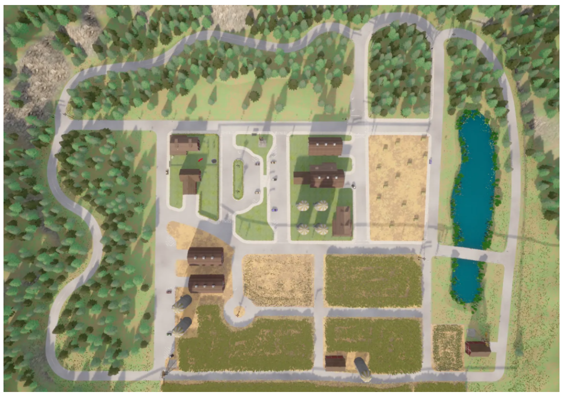

Figure 3.3: Aerial perspective of `Town07` [dev23].

The target velocity for the navigation scenario is set to 50 km/h, upon reaching the destination of a route, the target velocity is adjusted to 0 km/h. In the adaptive navigation scenario, traffic is generated on the map by CARLA traffic manager. The agent is not aware of the route, it only gets the next navigation instruction when approaching a junction.

3.2.3 Terminal States and Truncation

We consider a state terminal if any of the following conditions apply:

1. the car exceeds a maximal offset o_m from the center of the current lane, or
2. the car collides with another object (no matter if static or dynamic)

For all scenarios we specify $o_m = 1.62$, except for cruise-control where no maximal offset is specified due to a fixed steering angle and for obstacle avoidance where $o_m = 2$ in order to be able to change lane without termination (the road width is 4 meters).

In case of cruise control the episode is truncated after 280 time steps due to the limited length of the straight road, while all other environments have a default time limit of 3000 steps, to allow making a full round in `Town04` within one episode even at lower velocities (a full round with 50 km/h takes over 2000 time steps).

3.2.4 Action Space Design

The CARLA simulator API accepts throttle action values from the interval $\mathcal{A}_t := [0,1]$, likewise brake action values from $\mathcal{A}_b := [0,1]$. We unify brake and throttle control for the agent in the sense that negative throttle values encode the corresponding brake value

$$\mathcal{A}_c := [-1,1] \to \mathcal{A}_t \times \mathcal{A}_b,\ a \mapsto \begin{cases} (a,0), & \text{if } a \geq 0, \\ (0,a), & \text{if } a \leq 0. \end{cases}$$

When we refer to throttle control below, we thus always refer to the unified action space $\mathcal{A}_c$. Similarly, the simulator API accepts steering values from the interval $\mathcal{A}_s := [-1,1]$, where -1 corresponds to the maximum steering to the left and 1 to the maximum steering angle to the right.

For algorithms that require a discrete action space we discretize the action spaces. Doing so uniformly while still allowing fine-grained control would result into many actions, thus we propose a relative control paradigm, where the action a the agent selects gets added to the absolute control value c by

$$c = \min(\max(c+a,-1),1).$$

This requires the agent to be aware of the absolute control value c, which is not a limitation in our case, since we provide that anyway in the state space to account for the Markov property. With relative control approach the agent is able to set control values that are missing from the discretization at the expense, that the desired value might only be realized after multiple time steps.

3.3 Reward Design

In the following sections we propose our reward functions for the respective control problems, where we follow the convention that the range of the reward function should lie in the interval $[-1,1]$.

The reward functions for the navigation tasks are identical with the combined control rewards, thus there is no section dedicated to them.

3.3.1 Cruise Control Reward

For cruise control we define the reward function

$$r_c(v,v_t) = 1 + \begin{cases} -1, & \text{if } v < 0.5 \text{ and } v_t > 0 \text{ or terminal}, \\ -\max\left(\frac{|v-v_t|}{153}, 0.85\right), & \text{if } 0.5 \leq v < v_t + 0.5, \\ -\max\left(\frac{|v-v_t|}{153}, 0.85\right) - 0.15, & \text{if } v_t + 0.5 \leq v, \end{cases}$$

where v is the current velocity of the car in km/h and v_t is a given target velocity. This is motivated by the following ideas:

1. Standing around is maximally penalized independent of the delta velocity from the target velocity, reinforcing the agent to start moving.

2. The reward per se is designed to be of penalizing nature, with the idea being that every velocity apart from the target velocity is bad, the linear feedback should guide the agent to minimize the velocity difference. The constant in the denominator is chosen such to enable dense feedback across a range that includes most common legal speed limits.

3. The additive constant 1 would not matter for the optimal policy if the agent were not able to enter terminal states. However, by radical throttle changes the agent is able to make the vehicle spin slightly and since the steering angle is fixed to a zero angle the car would bump into the roadside shortly after, terminating the episode and thus obtaining a higher (negative) return.

4. If the agent surpasses the target velocity the same delta velocity gets a lower reward (-0.15) to indicate that going slower is better than going faster.

3.3.2 Adaptive Cruise Control Reward

For adaptive cruise control we define the reward function

$$r_{ac}(v, v_t, d) = \begin{cases} 0.15 - \frac{d_m(v) - d}{20 d_m(v)}, & \text{if } d < d_m(v) \text{ or terminal,} \\ 0, & \text{if } v < 0.5 \text{ and } v_t > 0 \text{ and } d > 5, \\ 1 - \max\left(\frac{|v - v_t|}{153}, 0.85\right), & \text{if } 0.5 \leq v < v_t + 0.5, \\ 1 - \max\left(\frac{|v - v_t|}{153}, 0.85\right) - 0.15, & \text{if } v_t + 0.5 \leq v, \end{cases}$$

where d is the distance in meters to a front vehicle in the same lane and $d_m(v) = \max\left(\frac{v}{2}, 2.5\right)$ is the minimum safety distance the vehicle should keep at velocity v. This is motivated by the following ideas:

1. The reward needs to be (at least partly) positive as when the agent has no possibility to get positive reward, ending an episode by crashing into a front car would get an attractive option to maximize the return.

2. As long as the agent keeps at least the minimum distance, the usual cruise control reward applies (with the important exception of case 2).

3. As soon as the distance to a front vehicle is lower than the minimum safe distance, the cruise control reward gets irrelevant and reward less than the worst cruise control reward (beneath the target velocity) is given.

4. In the second case it is important to make $d > c$ with some threshold $c > 2.5$ mandatory, otherwise in situations where the agent is stuck behind a vehicle that is not moving there would be a conflict between the punishment for standing around and the punishment for getting dangerously near to a front vehicle (as handled by the first case).

   This could also be resolved by giving a negative reward for unsafe distances, however we design the reward to be in $[0, 1]$ for the easy linear combination with the lane keeping reward.

3.3.3 Lane Keeping Reward

For lane keeping we define the reward function

$$r_l(o, s, s_p) = \begin{cases} 0, & \text{if } o > o_m \text{ or terminal}, \\ \left(1 - \sqrt{\frac{|o|}{o_m}}\right) \cdot (1 - \min(|s - s_p|, 0.9)), & \text{if } o \leq o_m, \end{cases}$$

where o is the normal offset of the center of the vehicle from the center of the lane in meters, o_m is a given maximum offset parameter, s the current steering angle and s_p the previous steering angle. This is motivated by the following ideas:

1. The agent needs to be able to reach positive reward for the same reason as in the previous section: otherwise it would be attractive to end the episode by entering a terminal state by driving out of the lane.

2. The maximum offset parameter is a safety parameter, exceeding the specified offset would for example mean leaving the lane, which would lead to termination of the training episode, thus the maximum offset allows a normalization of the observed offset, the objective is to keep the offset at a minimum.

3. Within the maximum offset range, the agent gets the feedback that a lower offset is better, the square root increases small offset values allowing stronger feedback in the small offset area where we aim to be.

4. The agent should adjust an appropriate steering angle right away and refrain from doing large corrections, thus changing the steering angle is always punished relative to the last steering angle by the additive term

   $$-\left(1 - \sqrt{\frac{|o|}{o_m}}\right) \cdot (1 - \min(|s - s_p|, 0.9)).$$

   This enforces continuity and thus smoother control. The reason we scale the punishment term with the offset term is to introduce a dependency on the offset: bigger offset scales down the punishment term as bigger steering deviations are reasonable in such situations.

3.3.4 Combined Control Reward

For the agent to control both throttle and steering we propose the reward

$$r_{cc}(v, v_t, o, s, s_p) = r_c(v, v_t) - 1 + r_l(o, s, s_p)$$

and analogously the combined reward for adaptive control

$$r_{acc}(v, v_t, d, o, s, s_p) = r_{ac}(v, v_t, d) - 1 + r_l(o, s, s_p),$$

where all the variables and reward functions are as defined in the previous sections. This is motivated by the following ideas

1. To end up within a $[-1, 1]$ interval for the combined adaptive reward, we subtract 1 from the cruise control rewards, as in the combined control scenario the agent has the possibility to get positive reward.

   While adding a positive constant to the reward does not alter the objective due to linearity of expectation and the geometric series, the same can't be generally said for a negative constant due to the first point outlined in the previous sections, however in our case this does not cause problems since the agent at all times has realistic chances to obtain a positive reward.

   To remain within the reward range $[-1, 1]$ it would also be a possibility to just define $r_{cclk} = \frac{1}{2}r_{acc} + \frac{1}{2}r_{lk}$, which would work as well, empirically our proposed variant worked out better.

2. The individual rewards are within $[-1, 0]$ and $[0, 1]$, thus the sum is within $[-1, 1]$. Since the case of terminal states is handled in the individual rewards, we always get a well-defined terminal reward of -1.

3.3.5 Highway Obstacle Avoidance Reward

We give the obstacle avoidance reward by

$$r_{lc}(v, d, o, s, s_p, l_c) = \frac{r'_{ac}(v, d, l_c) + r'_l(o, s, s_p, l_c)}{2},$$

where l_c is a variable indicating whether a lane change happened and whether it was to the left or to the right lane and

$$r'_{ac}(v, d, l_c) = \begin{cases} 0, & \text{if } l_c \text{ is left or right,} \\ r_{ac}(v, v, d), & \text{otherwise,} \end{cases} \tag{3.1}$$

$$r'_l(o, s, s_p, l_c) = \begin{cases} -0.5, & \text{if } l_c \text{ is left,} \\ -1, & \text{if } l_c \text{ is right,} \\ r_l(o, s, s_p) & \text{otherwise,} \end{cases} \tag{3.2}$$

are modified adaptive cruise control and lane keeping rewards. This is motivated by the following ideas:

1. Lane changes should only happen when necessary, hence all lane changes are punished, but a change to the left is preferred.
2. Since the agent only has to control the steering in this scenario, we set $v_t = v$ in the combined adaptive reward function with the consequence that the cruise control reward is always 1 as long as the agent maintains a safe distance.
3. While combination of the cruise control and lane keeping rewards analogous as for the combined adaptive scenarios would work, empirically the proposed variant works better, which could be explained by the fact that the agent does not have to go through multiple negative reward steps in order to change lane.

3.3.6 Combining Multiple Objectives

As outlined in Section 1.7 we implicitly transform the multi objective MDP into a single objective MDP via linear scalarization by specifying a scalar reward function that is a linear combination of single objective reward functions. By decomposing the combined value function

$$V_{1+2}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t (w_1 r_1(s) + w_2 r_2(s))\right) = w_1 V_1(s) + w_2 V_2(s)$$

into a weighted sum of single objective values we can argue, that the above combined reward indeed encourages the agent to meet our design goals.

Assuming that the agent is not too short-sighted by a low discounting factor in the objective, it is apparent that a policy that leads into a terminal state is never optimal in any of the objectives and thus not Pareto optimal.

As a consequence even if the cruise control objective were picked for primary optimization it is not optimal for the agent to pursue a target velocity where the vehicle hits a terminal condition.

This means any Pareto-optimal policy would be a wise trade-off between lane keeping and cruise control objectives: The agent will sacrifice lane keeping reward to some extent in order to reach a higher cruise-control objective or vice versa.

Standing around in the center of the lane would maximize the lane keeping objective, however this will not be a Pareto-optimal policy in any realistic settings (under the assumption that the cruise control reward is weighted accordingly), since it is always possible to drive the vehicle with non-zero velocity in such a way to have almost optimal lane keeping objective.

3.4 State Space Design

Except for the most simple cruise control scenario, all the proposed control problems per se require complex sensory input, e.g., camera image or Lidar data. This could be factored out of the reinforcement learning problem, e.g., by feeding the sensory inputs into a separately trained neural network that calculates a vector representation of the state instead of treating the sensor inputs as the state (which we refer to as *end-to-end* learning).

We provide gymnasium environment implementations where the observations are encoded as vectors (for problems 1 to 5 as in Section 3.1) as well as environment implementations where observations are a mix of vector and image data (for problems 2–8).

3.4.1 Vectorized Observations

We make the assumption here, that such vector states could be obtained in practice by concatenating scalar data such as current speed with predictions from a neural net that extracts information such as offset from lane center out of a camera image.

Cruise Control

The observation space is a linear subspace in $\mathbb{R}^3$ with vectors as outlined in the following table.

Dim	Observation	Value range
1	Velocity in km/h	$[0, 200]$
2	Throttle	$[-1, 1]$
3	Target velocity in km/h	$[0, 130]$

The throttle of the car is necessary to have the Markov property as the current velocity of the vehicle would not be sufficient information (for example there is a difference if we currently go with 30 km/h after deceleration from a higher velocity or if we were accelerating).

Adaptive Cruise Control

The previous observation space is extended to include information about a potential front vehicle in the current lane. The distance to a front vehicle could be predicted with the help of a neural net from a front camera image, or from radar resp. Lidar data.

Dim	Observation	Value range
1	Velocity in km/h	$[0, 200]$
2	Throttle	$[-1, 1]$
3	Target velocity in km/h	$[0, 130]$
4	Distance to front vehicle in m	$[0, 100]$
5	Relative velocity to front vehicle in km/h	$[0, 130]$

We confine the distance to a front vehicle to 100 meters, if no front vehicle is visible we treat it as 100 meters as well as if a front vehicle is present but farther away than

100 meters. The relative velocity is required for the Markov property as there is an important difference between approaching another car that is standing still or one that is only driving slightly slower. Alternatively one could drop that state and instead stack multiple previous observations, which would likely be the way to go unless one obtains such information with a radar sensor or the like where the relative velocity is already encoded in the measurements.

Lane Keeping

The state space is designed to feature sufficient information about the position of the vehicle in the current lane.

Dim	Observation	Value range
1	Offset from lane center in m	$[-1.68, 1.68]$
2	Angle between vehicle long axis and lane tangent in deg/90	$[-1, 1]$
3	Curvature of surrounding road segment	$[-\infty, \infty]$
4	Steering control value	$[-1, 1]$
5	Velocity in km/h	$[0, 200]$
6	Side acceleration in m/s^2	$[-50, 50]$

We assume that the first 3 dimensions could be obtained from a front camera image, e.g., with the help of a neural network. The surrounding road segment is defined by 3 points: the nearest midpoint of the current lane, one midpoint 20 meters ahead of that point and one midpoint 20 meters behind that point. We want to highlight that the current steering value is important for the Markov property as the current optimal steering action depends on the last steering position. We found the side acceleration of the vehicle also gives additional information for optimal control, although it may not be strictly necessary and could be replaced by a longer history of previous steering values.

Combined Control

Concatenation of the lane keeping and cruise control observation vectors.

Adaptive Combined Control

Concatenation of the lane keeping and adaptive cruise control observation vectors.

3.4.2 Camera Image Observations

A stack of camera images alone would not be sufficient to have the Markov property of our control problem, as the agent would not have any information of the current throttle or steering values, hence we handle observations as tuples of camera image and vector data.

Lane Keeping and Obstacle Avoidance

The observations consist of a camera image (camera mounted in front of vehicle with a field of view of 55 degrees) together with a vector as listed in the table below.

Dim	Observation	Value range
1	Steering control value	$[-1, 1]$
2	Velocity in km/h	$[0, 200]$
3	Side acceleration in m/s^2	$[-50, 50]$

The camera image is converted to grayscale and scaled down to 84×84 pixels. While the agent would have some information about the velocity from a stack of images, we supply the exact velocity, since that is information any car is able to provide anyway.

Combined Control Problems

Same camera setting as above, the additional vector observations are as in the table below.

Dim	Observation	Value range
1	Throttle	$[-1, 1]$
2	Steering control value	$[-1, 1]$
3	Velocity in km/h	$[0, 200]$
4	Target velocity in km/h	$[0, 130]$
5	Side acceleration in m/s^2	$[-50, 50]$

For the adaptive problems one can not expect that one observation contains all the information necessary (e.g., relative velocity to front vehicle), thus it is important to stack the last n observations into one observation for the agent, with $n = 3$ usually.

Navigation Problems

Camera settings are different: bigger field of view and the camera is mounted at the very front of the car in order to give the agent a broader view at junctions.

Dim	Observation	Value range
1	Navigation instruction	$\{0, 1, 2, 3, 4\}$
2	Distance to target in waypoints	$[0, 20]$
3	Throttle	$[-1, 1]$
4	Steering control value	$[-1, 1]$
5	Velocity in km/h	$[0, 200]$
6	Side acceleration in m/s^2	$[-50, 50]$

The navigational instructions (left, right, straight, follow lane, change lane left, change lane right) are encoded as integers in the vector state, as well as the distance to a possible stop point at the end of a route.

In case of the adaptive scenario stacking of multiple prior observations is necessary.

3.4.3 Preprocessing

In order to make the learning problems with camera input computationally more tractable, we follow a similar approach as [Mni+15] by down sampling the images to 84×84 dimensions and converting the camera to grayscale, which effectively reduces the 3 channels to 1. For stacked observations in the adaptive scenarios with image observations, we stack the last 3 preprocessed imaged along the channel axis, as well as the last 3 vector observations by concatenating those vectors.

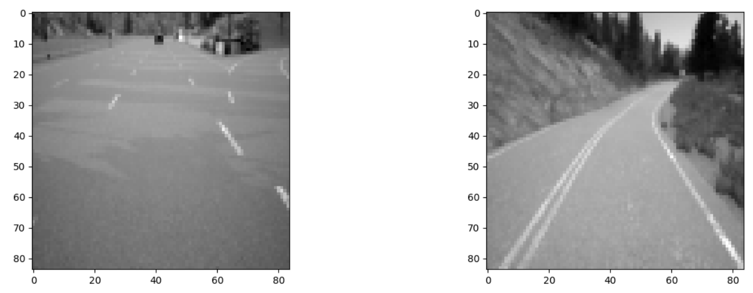

Figure 3.4: Preprocessed samples of camera images. Left sample is from the combined adaptive scenario, right on is from the navigation scenario.

3.5 Implementation Details

The CARLA simulator consists of a client server architecture: the server itself is based on the computer game engine Unreal Engine and runs the simulation while the client is responsible for setting up and controlling *actors* in the simulation as well as controlling the simulated world conditions. Interaction between the server and clients happens through network sockets.

CARLA offers an extensive Python API for the implementation of clients, which we make use of to implement *gymnasium environments* (see Section 3.5.2) for our proposed control problems.

3.5.1 Simulation Setup

Since CARLA is based on the gaming engine Unreal Engine the default simulation mode has a variable time step, meaning that the simulated time that passes between two simulation steps corresponds to the time it took to compute the simulation step, while the simulator does as many simulation steps as it is able to compute in a given real time frame, independently of client inputs.

We configure the simulator to have a *fixed time step* of 0.1 seconds, i.e., the simulated time that passes between simulation steps is always 0.1 seconds and set the simulator to *synchronous* mode, i.e., the simulator only does on simulation step when it is told to do so by the client and remains frozen otherwise.

This not only has the advantage of making the implementation of exactly evenly spaced time steps possible but also has additional technical advantages:

- Depending on the computational power it is possible to make the simulation faster than real time, e.g., if 100 steps of 0.1 seconds length can be computed within 1 second, then the simulation is 10 times faster than real time.
- All observation data can be obtained from the exact same moment in the simulation.
- Physics becomes deterministic making it possible to reach the exact same state with the same sequence of actions, given the same starting position.

In case of the vectorized observation environments we could greatly speed up the simulation (up to 30 times faster than real time) by turning off the rendering of the simulator (`settings.no_rendering=True`) as the state representation does not require rendering of the scene, while for the environments that depend on a camera sensor the simulation can be sped up with a lower graphics setting (`-quality-level=Low`) of the simulator at the expense of less detailed renderings (up to 6 times faster than real time compared to 3 times faster than real time with the highest graphic detail).

3.5.2 Gymnasium Interface

We provide all control problems as *gymnasium environments*, that means our control problems follow an API as proposed in [Tow+23]:

- `env = gymnasium.make(<NAME>)`: create the environment `env` with name `<NAME>`.
- `obs = env.reset(seed)`: obtain initial state `obs` of the environment, use the optional `seed` argument to seed the random number generators.
  In this method all actors including the vehicle the agent controls are destroyed (if they exist) and spawned at their initial positions (which are uniformly randomly sampled), new target velocities, navigation instructions etc. are sampled depending on the control problem.
- `obs, rew, terminated, truncated, info = env.step(action)`: perform one time step in the environment with `action`, return observation `obs`, a float reward `rew`, boolean `terminated` that classifies the observation as terminal, boolean `truncated` that is true when truncation conditions are met such as the maximum number of time steps for one episode and a dictionary `info` that provides additional info about the current step.

In this method the throttle and steering control value of the vehicle is applied, one simulation step is executed and all the observation data are collected.

As soon as a terminal state is encountered or a truncated signal is obtained, the `reset()` method is called.

Gymnasium provides dedicated data structures for action and observation spaces of which we make use of the following:

- `Box`: represents the Cartesian product of n closed intervals in $\mathbb{R}^n$, which we use for the action spaces of the continuous control variants of the environments, as well as the observation spaces of all vectorized observation environments. RGB images are also represented in box spaces (interval $[0, 255]$ and shape $(3, x, y)$ where x and y are the dimensions of the image in pixels).
- `Discrete`: represents a finite set of integers, we use it for discrete action spaces.
- `Dict`: composite space that is a Python dictionary of other spaces, which we use for the observation spaces of our environments that have pairs of images and vectors as observations.

3.5.3 Data Retrieval

Velocity and acceleration as well as location coordinate and direction vector information of the vehicle are directly retrievable as properties of the vehicle via the Python API.

The offset of the vehicle from the lane center is calculated with the help of *waypoints* that can be queried via the Python API: waypoints are a pair of 3-tuples, the first 3-tuple encodes coordinates of a point and the second pair encodes a normed direction vector attached to the coordinate point. The nearest lane center waypoint at each location is retrievable via the API, allowing the calculation of the offset as the normal distance between the location coordinates of the vehicle mass center and the waypoint coordinates. Likewise, the angle between the direction vector of the vehicle and the direction vector of the waypoint can be obtained, and the curvature can be calculated by using front and back waypoints to the nearest waypoint.

To calculate the distance between vehicles, we attach an *obstacle detector sensor* to the front of the car, that allows to read out the distance between the vehicle and another traffic participant if there is any within reach of the sensor.

In order to terminate episodes as soon as collisions happen there is a *collision sensor* available via the Python API that can be attached to the vehicle that registers collisions.

For camera image observations an *RGB camera sensor* is attached to the front of the vehicle that reports one camera image at each time step of the simulation. In order not to miss any camera images we wait until the camera image has been received by the client before making the next simulation step.

For the adaptive combined control and the adaptive navigation scenarios we make use of CARLA's traffic manager module, that allows the simulation of other traffic participants which are moved around by the traffic manager module.

For the navigation instructions of the navigation scenarios we make use of CARLA's route planner: we randomly sample a target location on the map and invoke the route planner which returns a list of pairs (waypoint, topological instruction) describing the shortest path from the current location to the target location. Using the waypoints we calculate the lane center offset of the car relative to the route and the current topological instruction is added to the state space.

3.5.4 Determinism

The physics of the simulator are deterministic in synchronous mode with fixed time steps. RGB camera output is not exactly deterministic with some slight noise differences even in the same exact physical state of the simulation. Likewise, sensors such as the collision sensor sometimes report a collision only one step later than it occurred, that might however be a race condition between the client and the server (the collision sensor does not report information at every simulation step, thus it does not make sense to wait for the sensor's report after every time step, which could lead to the situation that the sensor report only arrives after the next simulation step already has been initiated by the client).

The traffic manager module also allows setting a seed to make traffic situations reproducible, however traffic situations are only perfectly reproducible when the map is freshly loaded which is too runtime expensive to practically make use of, as reloading a map can take up to 2 minutes.

3.5.5 Error Recovery

Since the CARLA simulator and all of its components are a complex piece of software, crashes of the simulator, the python API or lockups where the client indefinitely hangs on a function call occur occasionally. In order to deal with this, we wrote a wrapper around the environment that runs our whole CARLA environment python code in an isolated sub-process. Timeouts are imposed for each call to the sub-process and if the sub-process encounters an error (e.g., the CARLA server crashed), gets killed or locks up, we close the sub-process including the simulator process, create a new sub-process, start the episode with exactly the same random seed and re-apply all the same actions that have been taken so far in the current episode in order to obtain the same environment state. To the agent none of this is noticeable. The physics determinism of the simulator thus makes silent error recovery possible, except for the adaptive navigation scenario.

4 Numerical Results

4.1 Methods

4.1.1 Data Generation

Since all of our problems are infinite-horizon problems, the agent-environment interaction is always truncated after an environment-specific number of steps, unless the agent encountered a terminal state before. In the following we refer to a trajectory that is ended by truncation or termination during training as a *training episode*, although in the first case it is not an episode by definition. Analogously, we use the term *evaluation episode*. To collect data for algorithm evaluation, we use the following protocol:

1. Training: number of r independent training runs with fixed number of training steps with different random seeds (for both environment and agent), each agent is saved periodically for evaluation (every i environment steps), the sum of rewards of each training episode is logged, as well as length of the training episode, whether the training episode was terminated or truncated and the computation time needed.

2. Evaluation: For each run we evaluate each saved agent by generating n evaluation-episodes with fixed environment seeds $\{s_1, \ldots, s_n\}$ by following the deterministic policy of the saved agent. We log the same data for each evaluation-episode as with training episodes, except for the computation time of the evaluation-episode.

In order to analyze performance of the algorithms we provide plots with regard to some metric over training with regard to either step or time units. This provides the following additional insights compared to tables of peak or final performances:

1. Stability of training: of particular interest is, whether the agent regresses in-between or almost monotonically improves performance over the course of training.

2. Sample efficiency: environment interaction is often expensive in terms of time, hence reaching higher performance with fewer samples is desirable. More sample efficient algorithms are often more computationally expensive however, so depending on how expensive the environment is to sample from an additional runtime comparison may draw a different picture.

In earlier literature it was common to use the number of episodes on the x-axis instead of the number of steps, but the latter is better for comparison as the number of episodes an agent has experienced is generally not a good indicator of how much experience it gathered (if the initial policy leads to long training episodes the agent will gather a lot more experience compared to an initial policy that will lead to immediate termination).

4.1.2 Performance Metrics

We consider the following performance metrics:

1. *Cumulative reward*: sum of all rewards of a training- or evaluation-episode. This is a common metric in the literature, despite the possible discrepancy with the discounted return the agent optimizes for.

2. *Average reward per step*: sum of rewards divided by the length of the training- or evaluation-episode.

3. *Termination rate*: number of times the agent encountered a terminal state divided by the number of evaluation trails. Depending on the problem termination can mean success or failure, in our domains it means failure.

One way to get an impression of the agent performance during training is to plot the respective performance metric of training episodes directly, which we refer to as *training curve*. As opposed to a *learning curve*, where we plot metrics with regard to evaluation-episodes, the training curve shows the impact of exploration during training, thus it is to be expected that the training curve is less stable and the obtained performance is lower than what the agent is capable of with the same experience when purely exploiting the learned policy. Another aspect to take into consideration is the fact, that in some environments the situations encountered in different training episodes are vastly different in terms of what performance the agent is able to reach even if following an optimal policy, thus momentary regression in the training curve does not necessarily equate to regression of the policy. Performance during training (averaged over the last k training episodes) as primary performance metric in reinforcement learning research papers has been proposed in [Mac+18, Sec. 4.2] for reducing the computational burden of doing evaluations in computationally expensive environments.

In the case of plotting the learning curve we usually do multiple evaluation-episodes per saved agent, thus we plot the mean over the performances obtained with regard to the above performance metrics.

4.1.3 Statistical Significance

In order to add statistical significance to the algorithm evaluation, multiple independent training runs are performed. We try to follow the recommendations given in [CSO19] and in [Aga+21].

In case of the training curve calculating statistics over multiple runs is usually not possible at the same point of training experience of the agent, since the experience the agent has at the start of a training episode depends on the length of all previous episodes, which usually varies between runs. Therefore, we treat the training curve of each individual run as a piece wise constant function over the number of steps and average over the runs at each step.

4.2 Algorithm Comparison across Environments

All algorithms use the stable-baselines3 codebase [Raf+21], that already contained peer reviewed implementations of all algorithms in this thesis except for IQN. We implemented the latter on top of the stable-baselines3 codebase, as well as diverse adaptions for the existing algorithms such as double DQN.

For hyperparameter tuning initially Optuna [Aki+19] was used, but in the end we manually came up with a selection of hyperparameters that work across all our domains, mostly with adaptions for sensitive and algorithm specific parameters, see Section 4.5.

The computational results presented were obtained using the CLIP cluster[1] and the VSC-5 cluster[2].

4.2.1 Cruise Control

The first challenge for the agent to overcome in this domain is picking up speed since it requires multiple successive positive throttle values for the vehicle to actually gain speed, which can be hard to explore in the beginning when the policy behaves mostly randomly. This is especially a noticeable problem for DDPG and TD3, see Fig. 4.1.

We would a priori not have expected the agent to be able to hit terminal states by going off the road, since the steering angle is held fixed, but it turned out that this happens when rapid acceleration changes happen (e.g., the agent suddenly applies maximum throttle). In the real physical world such situations might occur as well when the vehicle is sufficiently powerful. Applying steering corrections in our cruise-control experiment is out of scope, so avoiding this situation poses an additional challenge for the agent to overcome, since per the reward signal the incentive of the agent is to reach the target velocity as fast as possible, which stands in conflict to avoiding rapid acceleration changes.

The distributional algorithm variants except IQN are significantly more sample efficient in this domain as can be seen from SAC vs TQC and DQN vs QRDQN, the effect on final performance is minimal, however. While both QRDQN is more expensive in computational time than DQN, QRDQN manages to stay more performant overall. This computational time aspect does not carry over to TQC vs SAC.

4.2.2 Adaptive Cruise Control

Intuitively, the additional challenge of keeping a safe distance to a front vehicle would seem to be a more complex task that is more sample expensive to learn compared to the previous task. However, the learning curves reveal that this is not the case, see Fig. 4.2. In case of TD3 and DDPG, the additional challenge even leads to better convergence properties. Another side effect of the secondary task is, that the agents seems to be more gentle with acceleration, as terminal conditions are rarely encountered after convergence. Otherwise, the relative performance between algorithms remained approximately the same.

[1]`https://clip.science`
[2]`https://vsc.ac.at//systems/vsc-5/`

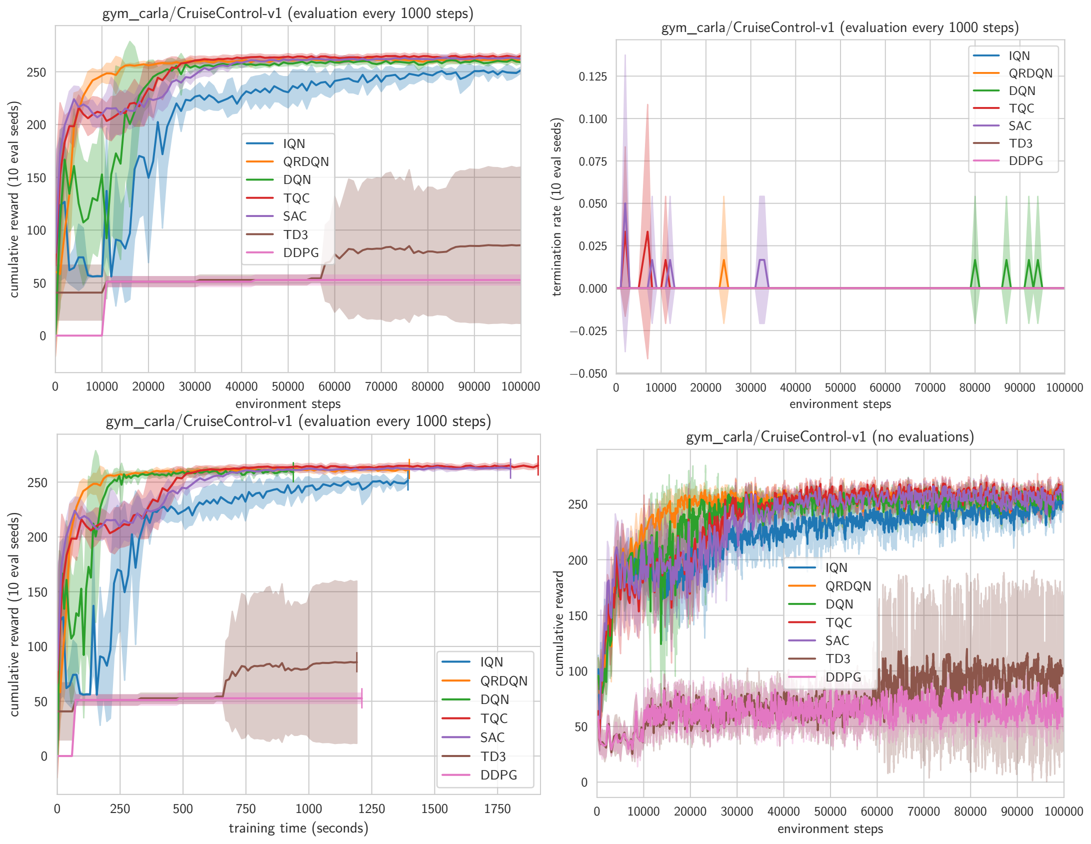

Figure 4.1: Cruise control learning and training curves.

4.2.3 Lane Keeping

For lane-keeping, policies that remain within the lane are comparatively easy to explore, since the velocity is not controlled by the agent and bad actions soon lead to termination by driving out of the lane. The main challenge for the agents is to learn the different curves at different velocities. Interestingly, the neural networks seem to generalize well, as can be seen on the sudden very steep ascent of the learning curves in Fig. 4.3, which means the agent does not slowly learn the curves one after another, but figures out a policy that works for all curves within a very short time.

All algorithms manage to learn good policies, as a finial performance around 2500 corresponds to smooth control keeping the offset to the lane center minimal. The termination rate plot, however uncovers the weakness of DQN, QRDQN and IQN to learn a safe policy, i.e., the agent should not hit a terminal conditions by driving out of the lane, while the actor critic methods manage to learn safe policies.

In this problem, IQN is clearly outperforming QRDQN while the latter is performing almost identically as DQN, which is exactly opposed to the observed performance in the Cruise control domain. It can be seen that the relative performance between algorithms using different means of exploration by the training curve is indeed inconclusive, as the

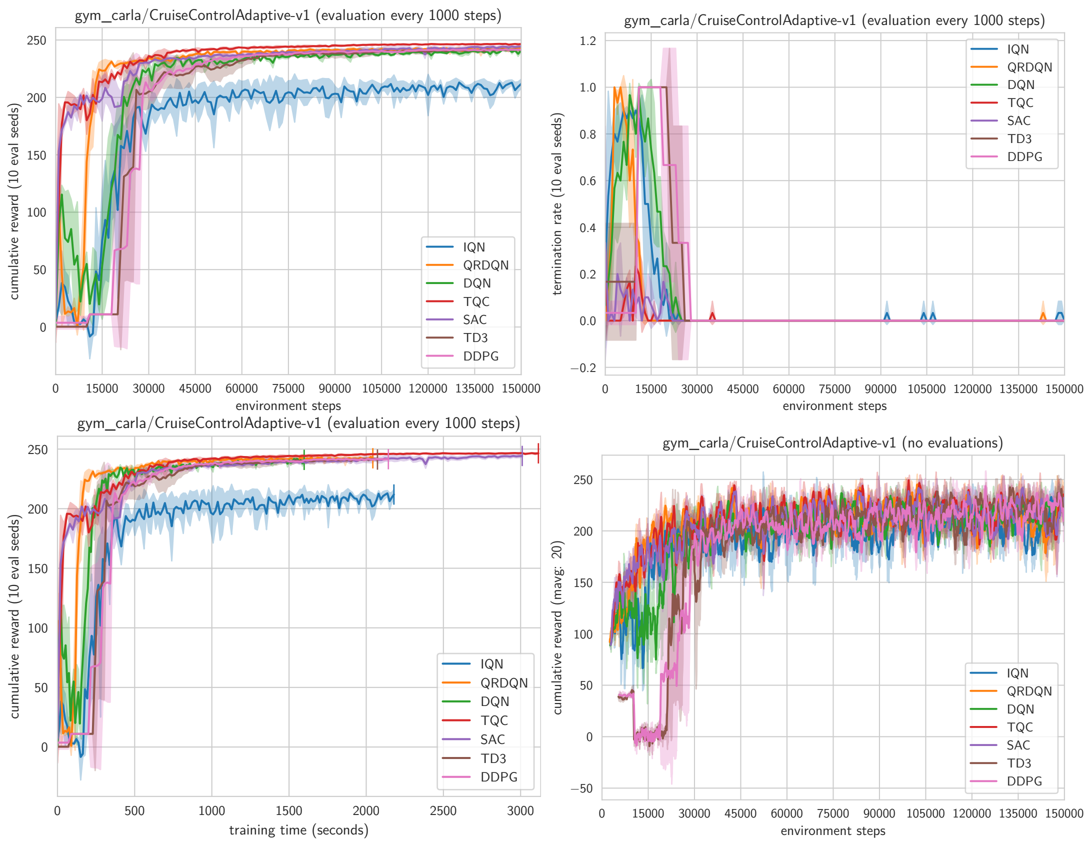

Figure 4.2: Adaptive Cruise control learning and training curves.

relative performance in the learning curve differs significantly from the one obtained in the learning curve. While this is to be expected for DDPG and TD3 which do not anneal the exploration rate, it can be seen that for maximum-entropy methods a higher level of stochasticity remains compared to the final exploration rate of ε-greedy exploration.

4.2.4 Lane Keeping from Pixels

With image based observations, the results look similar as for the vector representations, with the difference that reaching the same performance level takes more training steps which is to be attributed to the fact, that the agent implicitly needs to learn to extract the relevant features from the camera images in the CNN part of the neural network architecture. The final performance of each agent as can bee seen in Fig. 4.4 is similar to the performance without learning from pixels as in Fig. 4.3.

The IQN agent is considerably faster in learning to avoid terminal conditions in comparison to DQN and QRDQN, which struggle more with the pixel based observations in this regard, as is especially visible in the training curve, where IQN is able to consistently reach a high return despite exploration while QRDQN and DQN regularly fail keep the

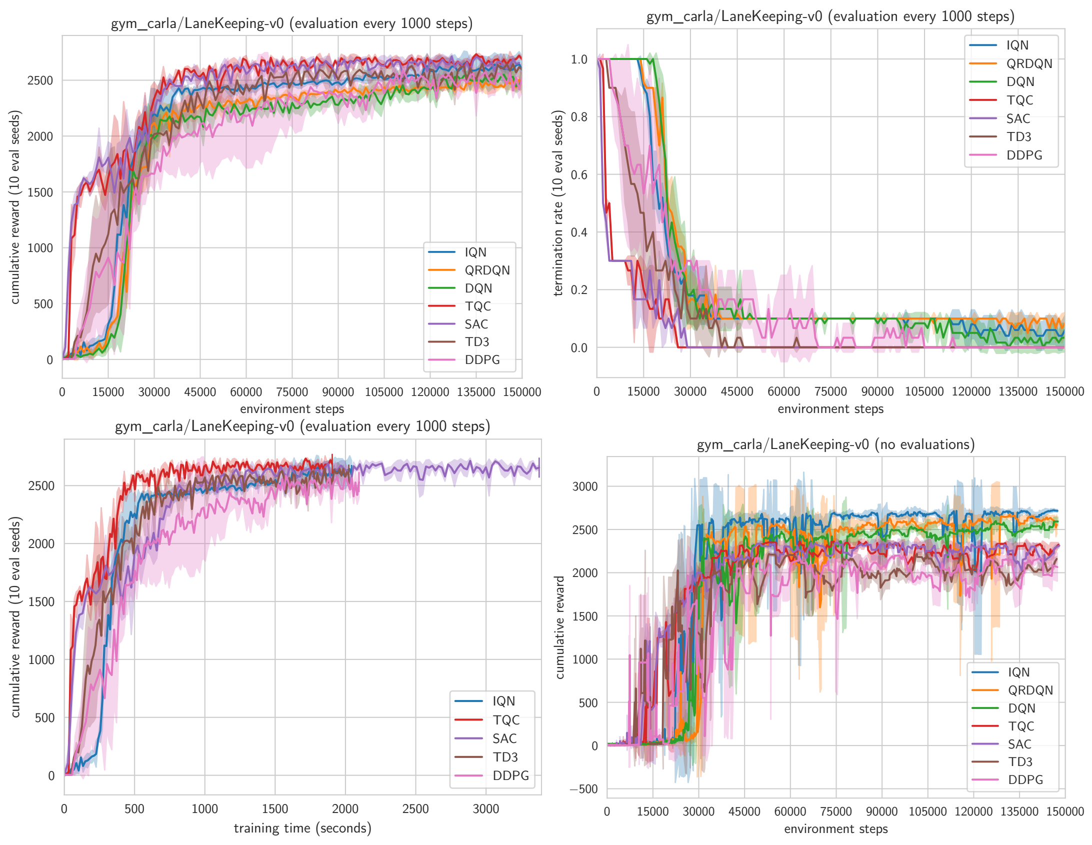

Figure 4.3: Lane keeping learning and training curves.

lane.

4.2.5 Combined Control

As expected, combined control is a lot more expensive to learn compared to the single objective tasks but at the advantage of being able to avoid terminal situations that would become unavoidable when just stacking individually trained agents. This becomes especially apparent in situations where the target velocity of the vehicle is too high to make a curve, which challenges the agent to slow down in advance.

A look at the plot of the termination rate plot uncovers a variety in the learning process between the different algorithms, see Fig. 4.5. While DQN, QRDN and IQN have high failure rates in the beginning and slowly learn to decrease their risk of failure, SAC and TQC quickly learn to avoid terminal situations by keeping the lane early. This changes throughout the training process, as the agent not only needs to keep the lane, but also needs to get as near as possible to a specified target velocity and with higher velocities comes higher risk of failure. TQC manages to minimize the risk of failure compared to SAC in the end, but both learn very smooth control policies that meet the objectives of both tasks well.

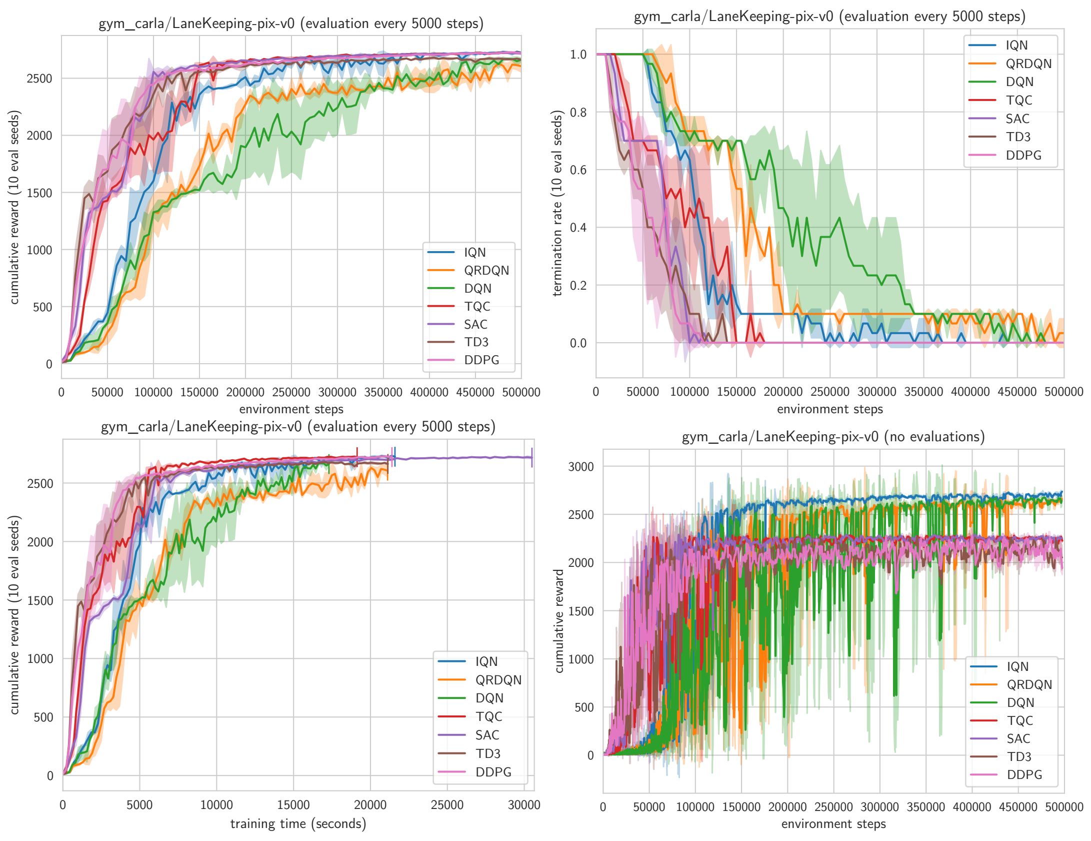

Figure 4.4: Lane-keeping from pixels learning and training curves.

The other algorithms do not reach the same level as SAC and TQC: the policies of IQN and QRDQN perform well overall but fail to stay within the lane eventually in some evaluation runs.

IQN is outperforming QRDQN, while DQN fails to learn good policies due to overestimation bias, which DDQN resolves.

Compared to the original publications, we needed to set the stability parameter for IQN much higher.

4.2.6 Combined Control from Pixels

Contrary to the lane-keeping domain, directly learning from pixels for combined control is even more sample efficient compared to directly learning from vector representations, at least in case of SAC and TQC, but also the other algorithms do not experience any noticeable drop in performance, see Fig. 4.6. In fact DQN is even able to learn good policies, which was not the case when directly learning from the vector representations as in Fig. 4.5.

The termination rate throughout training also suggests there is a difference in the learning dynamics compared to the vectorized environment, since this time around SAC

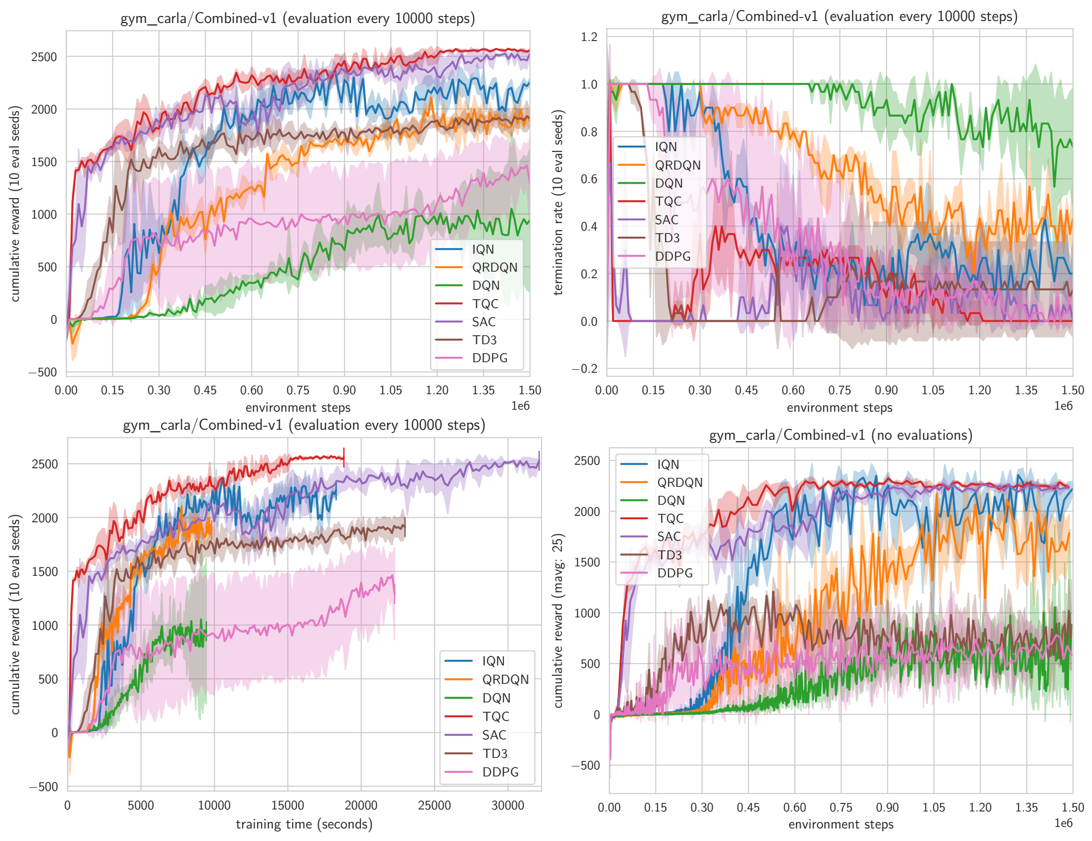

Figure 4.5: Combined control learning and training curves.

and TQC also slowly monotonically decrease their termination rate. This could be explained by the fact, that it takes longer for the agent to figure out the relevant features for reliably staying within the lane from the images, which makes the rapid improvement in terms of termination rate as seen in the vectorized environment impossible.

4.2.7 Adaptive Combined Control

When comparing the obtained returns from the learning curves in this domain with the ones in the previous section, one has to consider, that in the adaptive scenario an optimal policy may not be able to reach the same return if a front vehicle is there that makes reaching the target velocity impossible, in which case the agent is challenged to get as near as possible to the target velocity while keeping a safe distance at the same time.

Overestimation bias seems to be a considerable challenge in this problem domain, as DQN diverged regardless of the hyperparameters, while DDQN was able to learn good policies, see Fig. 4.7. TQC which was proposed primarily to overcome overestimation bias is also able to outperform SAC by a considerable margin which comes from the fact that it is able to avoid terminal conditions reliably, which all the other algorithms are not able to achieve.

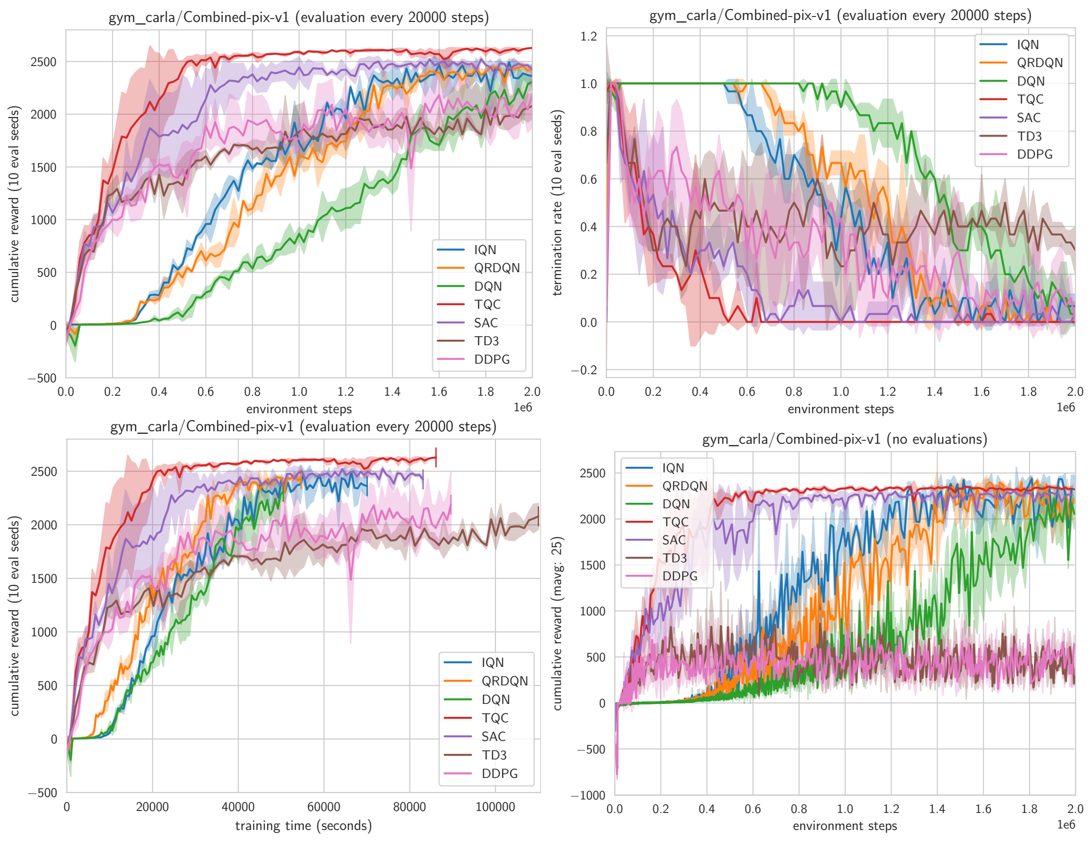

Figure 4.6: Combined control from pixels learning and training curves.

4.2.8 Adaptive Combined Control from Pixels

Except for TQC also in this domain the sample efficiency and overall performance is higher compared to directly learning from vectorized observations, see Fig. 4.8. TQC was not able to consistently maintain its advantage over the other algorithms with regard to avoiding terminal conditions, despite reaching minimal termination rate after around 1.3 million training steps, which per se demonstrates that a perfect result is indeed possible.

While the variance between runs of IQN in the vectorized setting, IQN was consistently more sample efficient compared to QRDN and DDQN, however the benefit did not translate in terms of runtime performance.

4.2.9 Obstacle Avoidance

Although the problem setting is very similar to the lane-keeping from pixels problem, the challenge of changing lane before getting dangerously near a front vehicle makes this problem considerably harder.

TQC and SAC both outperform all the other algorithms, but still both algorithms struggle in the end to completely avoid terminal conditions, see Fig. 4.9. Video inspection

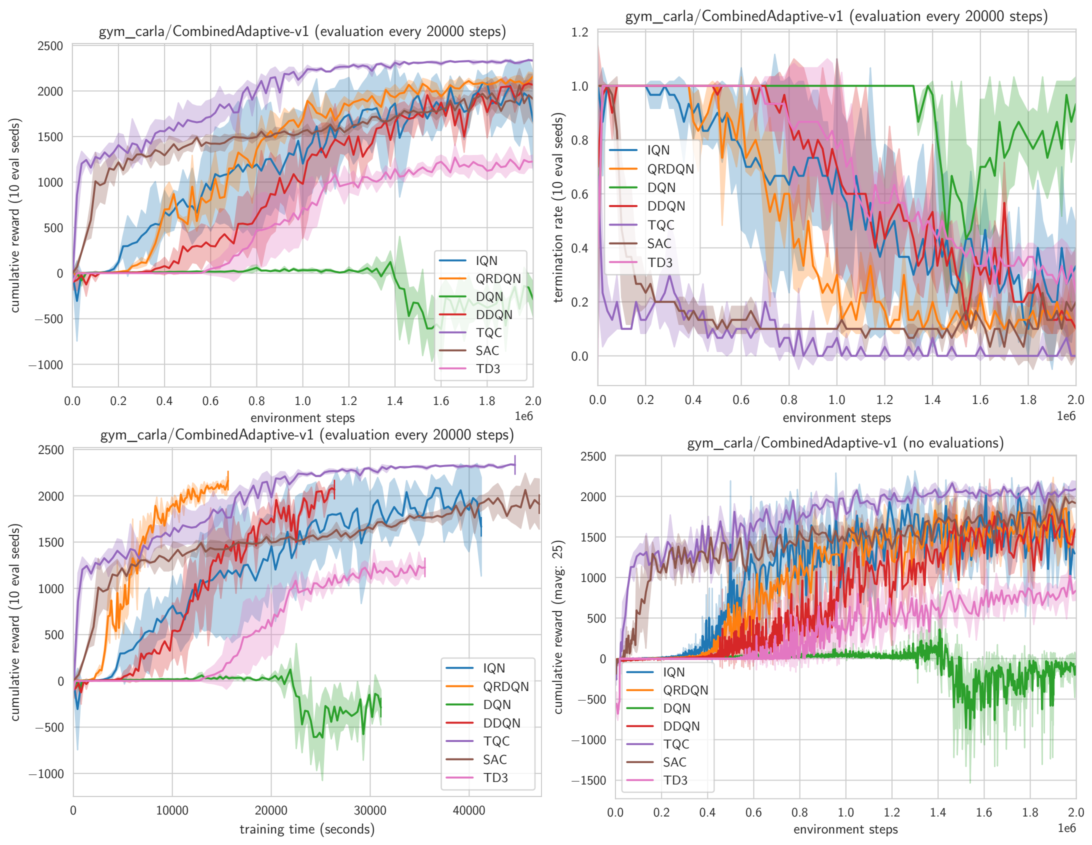

Figure 4.7: Adaptive combined control learning and training curves.

reveals two reasons for that:

1. In some situations, there is some space next to the very left lane that is separated by a solid line. Sometimes the agent tries to overtake using that space rather than choosing the lane to the right which leads to immediate termination.

2. During evaluation in some situations the control car has a high velocity while an obstacle appears suddenly after a curve, in which case the agent is sometimes not able to avoid the obstacle anymore despite making an attempt to do so.

4.2.10 Navigation

Since the navigation task has a goal-oriented nature, we use a different evaluation protocol: We run 30 evaluation runs, each time with a different target location for which the agent gets the route instructions during the episode. When the agent reaches the target point and manages to stop there within a certain radius, the episode ends with a success, all other cases are treated as failure. Since the agent learns to reach the target point faster over the course of training (the agent manages to stay closer to the target velocity even

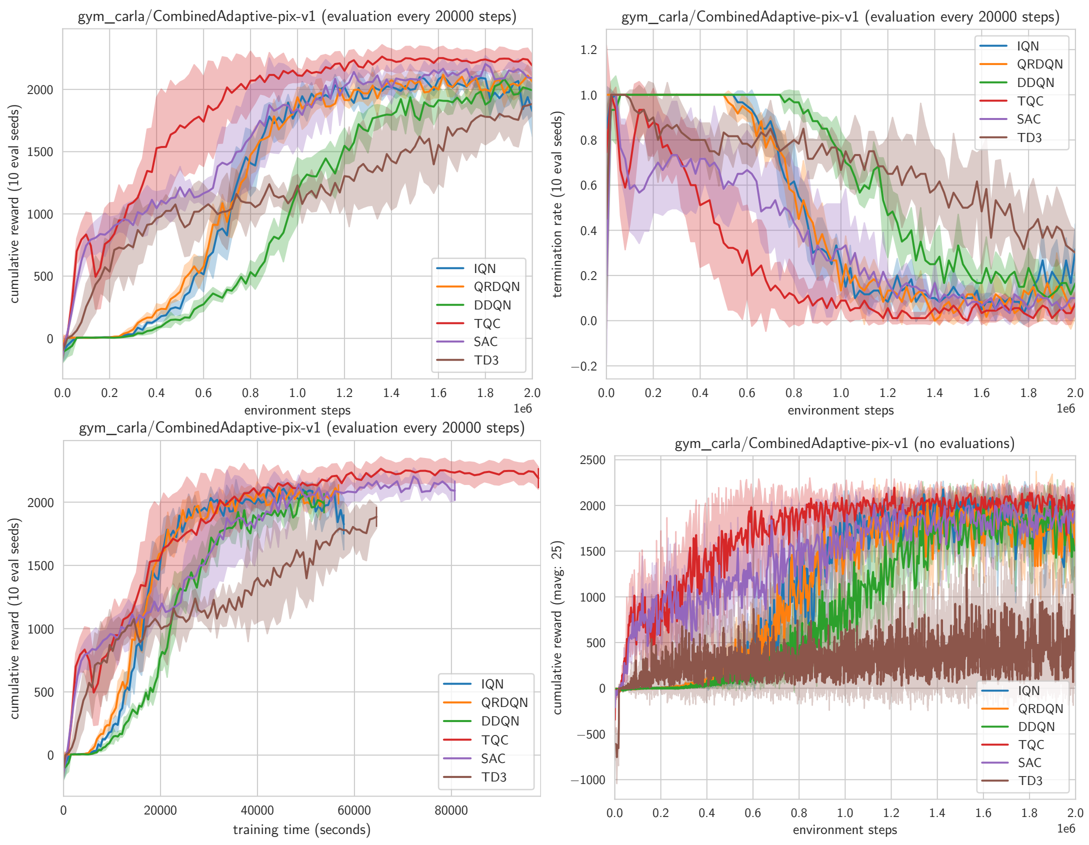

Figure 4.8: Adaptive combined control from pixels learning and training curves.

in junctions, while earlier in the training it slowed down considerable before junctions), the cumulative reward is not a good indicator of driving quality, which is why we opt for the average cumulative reward, besides the success rate. The average cumulative reward gives us an impression how well the control works with regard to lane keeping and adaptive cruise-control objectives.

Video inspection of evaluation runs reveals that for the top performing agents all the terminal conditions are encountered with hard left and right turns in junctions as the agent missed the opportunity to make the turn early enough, in which case it still tries to somehow make the curve but ends up in the opposite lane during the attempt and thus the episode is terminated.

While IQN performed inferior compared to QRDQN in the beginning, after a certain point of training experience IQN began to significantly outperform QRDQN. While there is no visible benefit of TQC over SAC in terms of the failure rate, TQC is significantly more stable and performant in terms of reward per time steps, i.e., in terms of how well the agent controls throttle and steering.

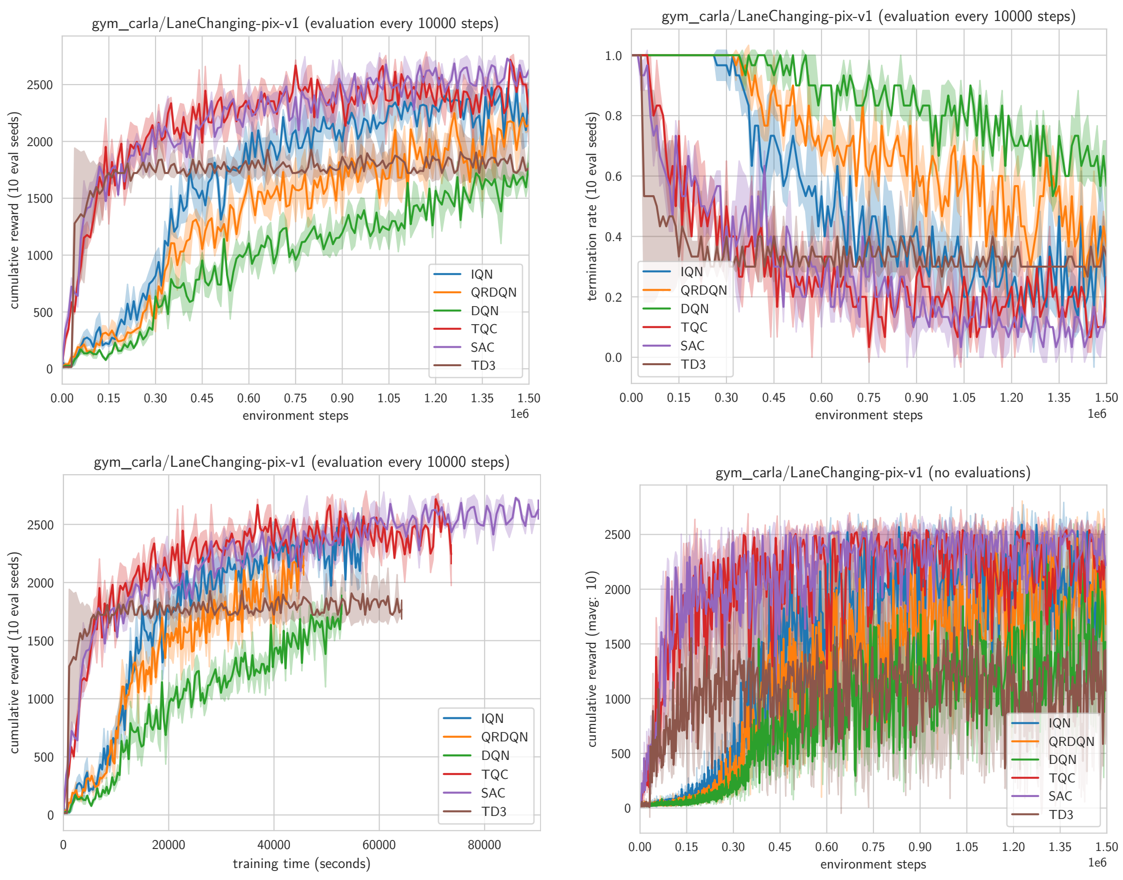

Figure 4.9: Obstacle avoidance learning and training curves.

4.2.11 Adaptive Navigation

The results as seen in Fig. 4.11 mostly match those of the navigation task Fig. 4.10 as seen in without traffic in terms of relative performance of the algorithms compared to each other.

Unfortunately none of the agents was able to learn a safe policy as we still obtain around 40% failure rate in the evaluations, and it does not appear like if longer training would help to counter this, as the SAC agent for example is stuck at that performance level for around 2.5 million steps without any noticeable change. Upon video inspection of the evaluation runs it turns out that the agents are mostly learning to drive well but are often unable to resolve traffic situations in junctions:

- Most of the terminations can be attributed to the agent trying to illegally surpass a front vehicle in junctions.

- Retrying the same traffic situation with the same trained agent multiple times often leads to different outcomes, since the traffic participants and the images are not

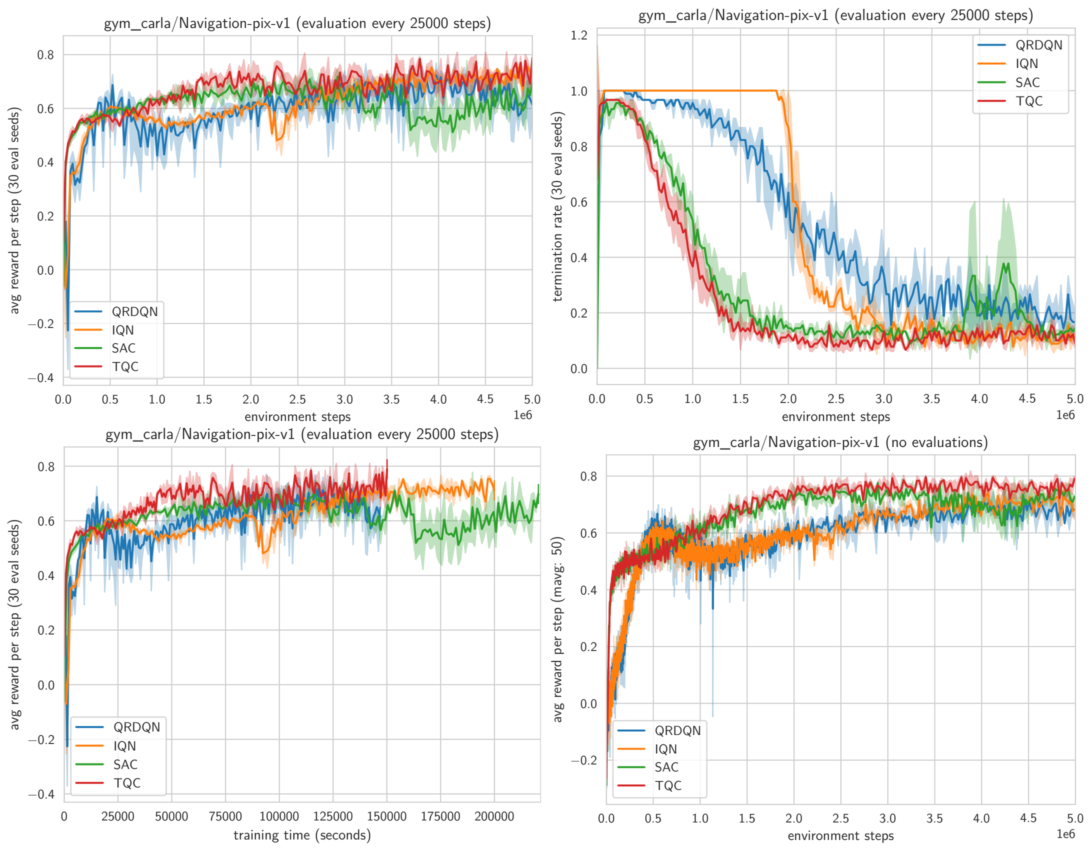

Figure 4.10: Navigation learning and training curves.

100% deterministic but to the human eye the situations up until shortly before the agent starts to make a sequence of bad decisions look almost identical.

4.3 Summary

Generally for the problems we designed, the evolution of deep reinforcement learning algorithms could be well understood and many of the improvements claimed in their original papers also apply to our problems, albeit not always as drastically as for the MuJoCo tasks and Atari games used as benchmark in the original publications. In the easier single objective tasks all algorithms obtained similarly good policies in the end, although the weakness of DDPG compared to the other algorithms was already apparent for the cruise control problem and despite significant improvements of TD3 over DDPG, there was still a very large variance between runs for the former.

Although in theory the problems with discretized action space should be easier to solve, the actor-critic algorithms for the continuous control problem generally were more sample efficient and produced better policies in the end, albeit at significantly higher run time cost due to at least two additional networks to update.

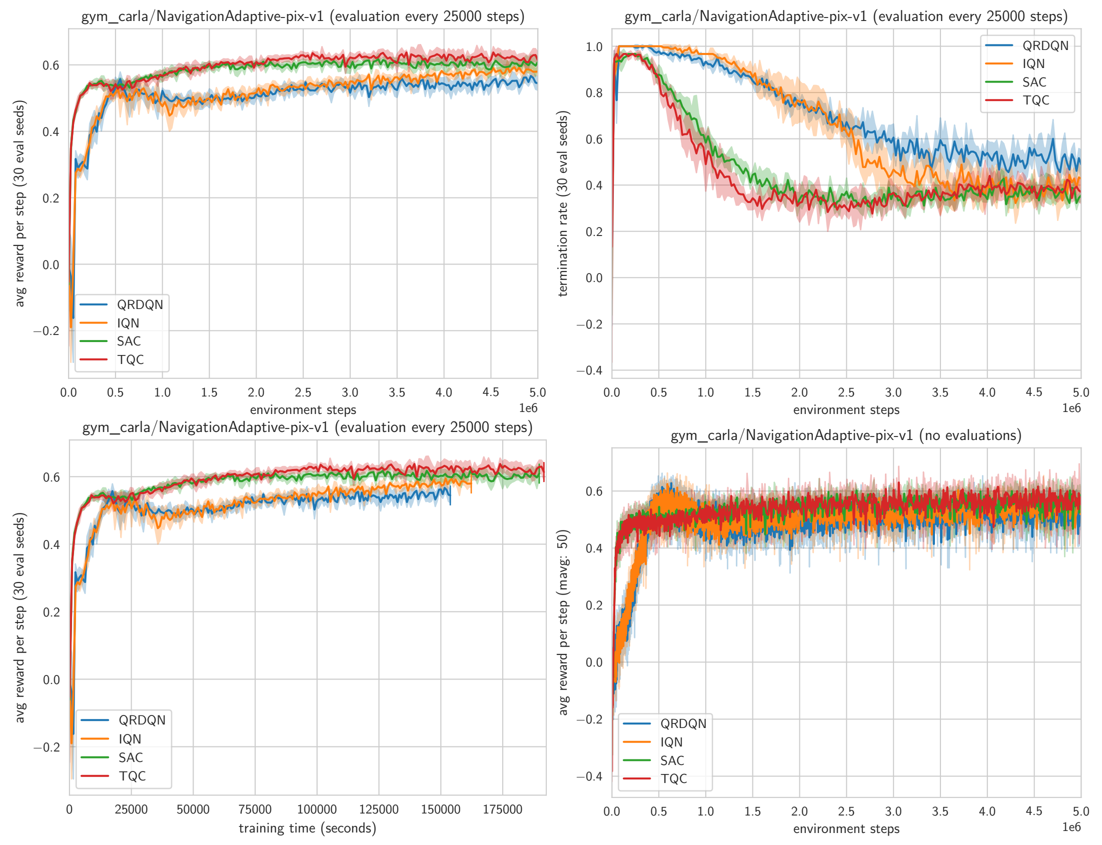

Figure 4.11: Adaptive navigation learning and training curves.

End to end learning from camera inputs was surprisingly not less sample efficient for the more complicated problems, although the cost of additionally learning the feature extractor component was very apparent for the easier problems.

The distributional variants of the algorithms were in the first place more sample efficient and less sensitive to overestimation bias generally leading to better convergence properties.

Videos of trained agents in action can be found here: `https://www.youtube.com/playlist?list=PLQQfP_m6pmBr8oK-YOTZkIuFzHni7j-3n`.

4.4 Own Publication

Parallel work using parts of the autonomous driving reinforcement learning stack developed in this thesis form the basis for [Lor+24].

4.5 Hyperparameters

Hyperparameters have been chosen such to work across as many environments as possible, thus a conservative choice has often been made e.g., generally a lower optimizer learning rate, that leads to the same final performance albeit at a little slower learning speed. Parameters that are brittle however can be seen in rows where many choices are made across environments and where a common choice did not come without significant disadvantages in the learning speed. All hyperparameters for each algorithm-environment combination are compactly presented in Table 4.1. Empty entries in each column inherit the value from the first non-empty entry to the left. Hyperparameters not mentioned for an algorithm are inherited from the top next row that mentions the parameter. We always make use of the Adam optimizer, see [KB15].

Parameter (Common)	CC	CCA	LK	LK-PIX	CB	CB-PIX	CBA	CBA-PIX	OA	NAV	NAVA
training length	10^5	$1.5 \cdot 10^5$	10^5	$5 \cdot 10^5$	$1.5 \cdot 10^6$	$2 \cdot 10^6$			$1.5 \cdot 10^6$	$5 \cdot 10^6$	
discount factor γ	0.95										
buffer size	10^6										
extractor dimension e	–			256	–	256	–	256	256		
extractor channels o	–			$32, 64, 64$	–	$32, 64, 64$	–	$32, 64, 64$	$32, 64, 64$		
extractor kernels $\underline{k}$	–			$8, 4, 3$	–	$8, 4, 3$	–	$8, 4, 3$	$8, 4, 3$		
extractor strides $\underline{s}$	–			$4, 2, 1$	–	$4, 2, 1$	–	$4, 2, 1$	$4, 2, 1$		
hidden units w of f_Q	256									512	
hidden layers d of f_Q	2										
optimizer	Adam										
optimizer learning rate	10^{-4}										
Parameter (DQN)											
learning starts	256										
mini-batch size	256				32						
target network update C	10^3		$3.5 \cdot 10^3$		$3 \cdot 10^4$		$5 \cdot 10^5$		$5 \cdot 10^3$	$5 \cdot 10^5$	
target network update τ	1										
exploration rate ε	0.05										
exploration decay fraction	0.2				0.25						
Parameter (QR-DQN)											
number of quantiles	128										

	CC	CCA	LK	LK-PIX	CB	CB-PIX	CBA	CBA-PIX	OA	NAV	NAVA
Parameter (IQN)											
number of quantiles m	64										
number of quantiles m'	64										
number of quantiles $\tilde{m}$	32										
Parameter (SAC)											
mini-batch size	256										
target network update C	1										
target network update τ	0.005										
target entropy	$-\dim \mathcal{A}$										
hidden units f_π	256									512	
hidden layers f_π	2										
Parameter (DDPG)											
learning starts	10^4		$2 \cdot 10^3$	$3 \cdot 10^3$		$1.2 \cdot 10^4$	$1.8 \cdot 10^4$		$3 \cdot 10^4$		
action noise	0.1				0.15						
Parameter (TD3)											
policy delay	2										
target policy noise	0.2										
target noise clip	0.5										
Parameter (TQC)											
number of critics n	2										
number of quantiles m	25										
quantile to drop d	2										

Table 4.1: Hyperparameters.

Bibliography

[Aga+21] Rishabh Agarwal et al. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 29304–29320. URL: `https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf`.

[Aki+19] Takuya Akiba et al. "Optuna: A Next-Generation Hyperparameter Optimization Framework". In: *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 2623–2631.

[Bar+18] Gabriel Barth-Maron et al. "Distributional Policy Gradients". In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=SyZipzbCb`.

[BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 449–458. URL: `https://proceedings.mlr.press/v70/bellemare17a.html`.

[BDR23] Marc G. Bellemare, Will Dabney, and Mark Rowland. *Distributional Reinforcement Learning*. `http://www.distributional-rl.org`. MIT Press, 2023.

[CC21] Johan Samir Obando Ceron and Pablo Samuel Castro. "Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 1373–1383. URL: `https://proceedings.mlr.press/v139/ceron21a.html`.

[CCM23] Zaiwei Chen, John-Paul Clarke, and Siva Theja Maguluri. "Target Network and Truncation Overcome the Deadly Triad in Q-Learning". In: *SIAM Journal on Mathematics of Data Science* 5.4 (2023), pp. 1078–1101. eprint: `https://doi.org/10.1137/22M1499261`. URL: `https://arxiv.org/pdf/2203.02628.pdf`.

[CMS20] Diogo Carvalho, Francisco S. Melo, and Pedro Santos. "A new convergent variant of Q-learning with linear function approximation". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 19412–19421. URL: `https://`

proceedings.neurips.cc/paper_files/paper/2020/file/e1696007be4eefb81b1a1d39ce48681b-Paper.pdf.

[CSO19] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. “A Hitchhiker’s Guide to Statistical Comparisons of Reinforcement Learning Algorithms”. In: *Reproducibility in Machine Learning, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryx0N3IaIV.

[CVM23] Fengdi Che, Gautham Vasan, and A. Rupam Mahmood. “Correcting discount-factor mismatch in on-policy policy gradient methods”. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 4218–4240. URL: https://proceedings.mlr.press/v202/che23a.html.

[Dab+18a] Will Dabney et al. “Distributional reinforcement learning with quantile regression”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI’18/IAAI’18/EAAI’18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8. URL: https://aaai.org/papers/11791-distributional-reinforcement-learning-with-quantile-regression/.

[Dab+18b] Will Dabney et al. “Implicit Quantile Networks for Distributional Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1096–1105. URL: https://proceedings.mlr.press/v80/dabney18a.html.

[dev23] CARLA developers. *CARLA Documentation*. 2023. URL: https://carla.readthedocs.io/en/0.9.15 (visited on 02/19/2024).

[Dos+17] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[Fei11] Eugene A. Feinberg. “Total Expected Discounted Reward MDPS: Existence of Optimal Policies”. In: (2011). eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470400531.eorms0906. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7862cf86f3fee05fc62b5f01aaee66a48044d623.

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1587–1596. URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

[GSP19] Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. “A Theory of Regularized Markov Decision Processes”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2160–2169. URL: `https://proceedings.mlr.press/v97/geist19a.html`.

[Haa+18] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1861–1870. URL: `https://proceedings.mlr.press/v80/haarnoja18b.html`.

[Haa+19] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications.* 2019. arXiv: `1812.05905 [cs.LG]`.

[Has10] Hado Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: `https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf`.

[Has12] Hado van Hasselt. “Reinforcement Learning in Continuous State and Action Spaces”. In: *Reinforcement Learning: State-of-the-Art*. Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 207–251. ISBN: 978-3-642-27645-3. URL: `https://doi.org/10.1007/978-3-642-27645-3_7`.

[HGS16] Hado van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double Q-Learning”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI’16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100. URL: `https://aaai.org/papers/10295-deep-reinforcement-learning-with-double-q-learning/`.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: `http://arxiv.org/abs/1412.6980`.

[Kie23] Tobias Kietreiber. “Combining maximum entropy reinforcement learning with distributional Q-value approximation methods: At the example of autonomous driving”. MA thesis. Technische Universität Wien, 2023. URL: `https://repositum.tuwien.at/handle/20.500.12708/177687`.

[Kuz+20] Arsenii Kuznetsov et al. “Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research.

PMLR, 2020, pp. 5556–5566. URL: https://proceedings.mlr.press/v119/kuznetsov20a.html.

[LGR12] Sascha Lange, Thomas Gabel, and Martin Riedmiller. "Batch Reinforcement Learning". In: *Reinforcement Learning: State-of-the-Art.* Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 45–73. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_2. URL: https://doi.org/10.1007/978-3-642-27645-3_2.

[LHY23] Haoye Lu, Daniel Herman, and Yaoliang Yu. "Multi-Objective Reinforcement Learning: Convexity, Stationarity and Pareto Optimality". In: *The Eleventh International Conference on Learning Representations.* 2023. URL: https://openreview.net/forum?id=TjEzIsyEsQ6.

[Lig24] Bar Light. *The Principle of Optimality in Dynamic Programming: A Pedagogical Note.* 2024. arXiv: 2302.08467 [math.OC].

[Lil+16] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.* Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: https://arxiv.org/abs/1509.02971.

[Lor+24] Pierrick Lorang, Helmut Horvath, Tobias Kietreiber, Patrik Zips, Clemens Heitzinger, and Matthias Scheutz. "Adapting to the "Open World": The Utility of Hybrid Hierarchical Reinforcement Learning and Symbolic Planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA).* 2024, to appear.

[LS20] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms.* Cambridge University Press, 2020. URL: https://tor-lattimore.com/downloads/book/book.pdf.

[Mac+18] Marlos C. Machado et al. "Revisiting the arcade learning environment: evaluation protocols and open problems for general agents". In: *J. Artif. Int. Res.* 61.1 (Jan. 2018), pp. 523–562. ISSN: 1076-9757. URL: https://www.jair.org/index.php/jair/article/view/11182/26388.

[Mai68] Ashok Maitra. "Discounted Dynamic Programming on Compact Metric Spaces". In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 30.2 (1968), pp. 211–216. URL: http://library.isical.ac.in:8080/jspui/bitstream/10263/742/1/68.02.pdf (visited on 02/25/2024).

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf.

[Nai+19] Abhishek Naik et al. “Discounted Reinforcement Learning is Not an Optimization Problem”. In: *NeurIPS Optimization Foundations for Reinforcement Learning Workshop.* 2019. URL: https://optrl2019.github.io/assets/accepted_papers/66.pdf.

[NT20] Chris Nota and Philip S. Thomas. “Is the Policy Gradient a Gradient?” In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems.* AAMAS ’20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 939–947. ISBN: 9781450375184. URL: https://arxiv.org/pdf/1906.07073.pdf.

[Put05] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley series in probability and statistics. Hoboken, NJ: John Wiley & Sons Inc., 2005.

[RA21] Matthew T. Regehr and Alex Ayoub. *An Elementary Proof that Q-learning Converges Almost Surely.* 2021. arXiv: 2108.02827 [cs.LG].

[Raf+21] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[Rie05] Martin Riedmiller. “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method”. In: *Machine Learning: ECML 2005.* Ed. by João Gama et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328. ISBN: 978-3-540-31692-3.

[Roi+13] Diederik M. Roijers et al. “A survey of multi-objective sequential decision-making”. In: *J. Artif. Int. Res.* 48.1 (Oct. 2013), pp. 67–113. ISSN: 1076-9757. URL: https://arxiv.org/ftp/arxiv/papers/1402/1402.0590.pdf.

[Row+18] Mark Rowland et al. “An Analysis of Categorical Distributional Reinforcement Learning”. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics.* Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. PMLR, 2018, pp. 29–37. URL: https://proceedings.mlr.press/v84/rowland18a.html.

[Row+23] Mark Rowland et al. *An Analysis of Quantile Temporal-Difference Learning.* 2023. arXiv: 2301.04462 [cs.LG].

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* Second Edition. Cambridge, MA: MIT press, 2018. URL: http://incompleteideas.net/book/RLbook2020.pdf.

[Sil+14] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning.* Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Beijing, China: PMLR, 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

[SPC19] Zhao Song, Ron Parr, and Lawrence Carin. “Revisiting the Softmax Bellman Operator: New Benefits and New Perspective”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5916–5925. URL: `https://proceedings.mlr.press/v97/song19c.html`.

[SS21] Liran Szlak and Ohad Shamir. *Convergence Results For Q-Learning With Experience Replay*. 2021. arXiv: `2112.04213 [cs.LG]`.

[Tow+23] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: `10.5281/zenodo.8127026`. URL: `https://zenodo.org/record/8127025` (visited on 07/08/2023).

[TTM19] Chen Tessler, Guy Tennenholtz, and Shie Mannor. “Distributional Policy Optimization: An Alternative Approach for Continuous Control”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper_files/paper/2019/file/72da7fd6d1302c0a159f6436d01e9eb0-Paper.pdf`.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. URL: `https://doi.org/10.1007/BF00992698`.

[Wu+22] Shuang Wu et al. “Understanding Policy Gradient Algorithms: A Sensitivity-Based Approach”. In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 24131–24149. URL: `https://proceedings.mlr.press/v162/wu22i.html`.

[WU22] Zhikang T. Wang and Masahito Ueda. “Convergent and Efficient Deep Q Learning Algorithm”. In: *International Conference on Learning Representations*. 2022. URL: `https://openreview.net/forum?id=OJm3HZuj4r7`.