Université de Montréal

# Real-Time Reinforcement Learning

**par Simon Ramstedt**

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

September, 2019

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

**Real-Time Reinforcement Learning**

présenté par:

**Simon Ramstedt**

a été évalué par un jury composé des personnes suivantes:

| | |
|---|---|
| **Philippe Langlais**, | président-rapporteur |
| **Christopher Pal**, | directeur de recherche |
| **Alain Tapp**, | membre du jury |

Mémoire accepté le: ............................

# Résumé

Les processus de décision markovien (MDP), le cadre mathématiques sous-jacent à la plupart des algorithmes de l'apprentissage par renforcement (RL) est souvent utilisé d'une manière qui suppose, à tort, que l'état de l'environnement d'un agent ne change pas pendant la sélection des actions. Puisque les systèmes RL basés sur les MDP classiques commencent à être appliqués dans les situations critiques pour la sécurité du monde réel, ce décalage entre les hypothèses sous-jacentes aux MDP classiques et la réalité du calcul en temps réel peut entraîner des résultats indésirables. Dans cette thèse, nous introduirons un nouveau cadre dans lequel les états et les actions évoluent simultanément, nous montrerons comment il est lié à la formulation MDP classique. Nous analyserons des algorithmes existants selon la nouvelle formulation en temps réel et montrerons pourquoi ils sont inférieurs, lorsqu'ils sont utilisés en temps réel. Par la suite, nous utiliserons ces perspectives pour créer un nouveau algorithme Real-Time Actor Critic qui est supérieur au Soft Actor Critic contrôle continu de l'état de l'art actuel, aussi bien en temps réel qu'en temps non réel.

**mots-clés:** apprentissage profond, apprentissage par renforcement

# Summary

Markov Decision Processes (MDPs), the mathematical framework underlying most algorithms in Reinforcement Learning (RL), are often used in a way that wrongfully assumes that the state of an agent's environment does not change during action selection. As RL systems based on MDPs begin to find application in real-world safety critical situations, this mismatch between the assumptions underlying classical MDPs and the reality of real-time computation may lead to undesirable outcomes. In this thesis, we introduce a new framework, in which states and actions evolve simultaneously, we show how it is related to the classical MDP formulation. We analyze existing algorithms under the new real-time formulation and show why they are suboptimal when used in real-time. We then use those insights to create a new algorithm, Real-Time Actor Critic (RTAC) that outperforms the existing state-of-the-art continuous control algorithm Soft Actor Critic both in real-time and non-real-time settings.

**Keywords:** deep learning, reinforcement learning

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| PDF | Probability Density Function |
| CNN | Convolutional Neural Networks |
| RL | Reinforcement Learning |
| RTRL | Real-Time Reinforcement Learning |
| MDP | Markov Decision Process |
| MRP | Markov Reward Process |
| TBMRP | Turn-based Markov Reward Process |
| RTMRP | Real-time Markov Reward Proces |
| TBMRP | Turn-based Markov Decision Process |
| RTMRP | Real-time Markov Decision Proces |
| SAC | Soft Actor-Critic |
| RTAC | Real-Time Actor-Critic |
| TD3 | Twin Delayed Deep Deterministic Policy Gradients |

# Acknowledgments

I am very grateful for the support and encouragement from my family during my past two years in Montreal – Danke!

Thank you to Qiao Huang for her love and support.

Thank you to my advisor Chris Pal for his guidance, support and the incredible freedom and resources he made available to me.

Thank you to Alex Piché and Cyril Ibrahim for great discussions and for being a great coworkers.

Thank you to Craig Quiter for patiently discussing with me about simulation environments and having gotten me to think more deeply agent-environment interaction.

Thank you to Sherjil Ozair for encouraging and reviewing early drafts of our paper.

Thank you to Jhelum Chakravorty and Scott Fujimoto for reviewing and discussing our paper with me.

Thank you to Jose Gallego, Olexa Bilaniuk and many others Mila that helped me through countless discussions online and offline.

Thank you to ElementAI for providing a great research environment and great resources.

And finally thank you to the Open Philanthropy Project for providing the financial support for my Master's degree.

# 1 Introduction

Even today, most machines are still hand-programmed by humans. These machines, however, are limited by the finite capacity of humans to understand their environment's dynamics and our poor ability of communicating knowledge via computer code. This is hardly surprising. We have evolved to communicate mainly with other humans that are able understand abstract and vague concepts.

With the rise of Machine Learning in recent years, hand-programmed systems are getting replaced in areas such as computer vision or language understanding. These areas are often the ones in which humans are intuitively good at but are not amenable to traditional scientific and mathematical analysis. In those areas, instead of programming rules, we provide training data that is used by learning systems to find rules automatically, with the hope that the rules will generalize to new data outside of the training set. While these supervised learning systems are less limited than hand-programmed machines, they still require large amounts of training data and sometimes learn spurious rules that do not generalize.

Reinforcement Learning systems can close that generalization gap as they interact with their environment and are able to continuously incorporate new training data. Furthermore, rather than requiring large amounts of training data it allows humans to specify goals that can be encoded in a reward function. Unfortunately, few Reinforcement Learning systems are currently operating in the real world. Even though, there have been promising examples [Mnih et al., 2015, Silver et al., 2017], most of them were in simulated environments that allowed for huge amounts of environment interactions. As algorithms get more data efficient, they often become more complex, too, involving multiple function approximators that are being optimized with respect to each other [Janner et al., 2019].

In this thesis we take a look at the foundations of Reinforcement Learning and find that there is room for improvement. We propose a new agent-environment interaction scheme that improves real-world applicability and has the potential to make future Reinforcement Learning algorithms simpler.

## 1.1 Reinforcement Learning

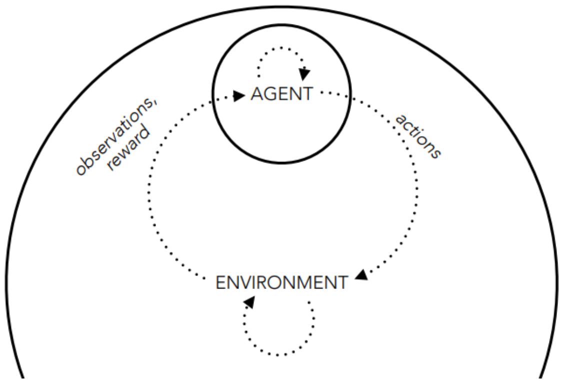


**Figure 1.1:** The agent-environment system (curtesy of Joe Marino [2019])

Reinforcement Learning splits up the world into agent and environment. The agent observes the environment and receives rewards while it influences the environment with its actions.

Time plays a central role. At each timestep an agent observes new data and selects a new action in order to maximize rewards in the future. The stream of observations, actions and rewards is divided into discrete timesteps so it can be handled by digital computers. For mathematical convenience, we introduce the notion of a state. A state contains all information from past observations about the agent's future observations and rewards, i.e. states are markovian. A markovian state can be constructed for any type of observation by concatenating all past observations.

With a well defined state space and action space we can represent the agent by a policy, the environment by a Markov Decision Process and the agent-environment system by a Markov Reward Process. While a policy is simply a distribution over actions conditioned on a state, the Markov Decision Process [Bellman, 1957] defines the action's effects on the state and determines the evolution of states. Again:

$$\begin{aligned}\text{Agent} &\rightarrow \text{Policy},\\ \text{Environment} &\rightarrow \text{Markov Decision Process (MDP)},\\ \text{Agent-Environment System} &\rightarrow \text{Markov Reward Process (MRP)}.\end{aligned}$$

---

**Definition 1.** *A Markov Decision Process (MDP) is characterized by a tuple with*

| | |
|---|---|
| *(1) state space* | $S$, |
| *(2) action space* | $A$, |
| *(3) initial state distribution* | $\mu : S \rightarrow \mathbb{R}$, |
| *(4) transition distribution* | $p : S \times S \times A \rightarrow \mathbb{R}$, |
| *(5) reward function* | $r : S \times A \rightarrow \mathbb{R}$. |

## 1.2 The Agent-Environment System

An agent-environment system can be condensed into a Markov Reward Process $(S, \mu, \kappa, \bar{r})$ which consists of a Markov process $(S, \mu, \kappa)$ and a state-reward function $\bar{r}$.

Until now, the standard way to condense an agent-environment system was to marginalize out the action. Here, we refer to this procedure as the Turn-Based Markov Reward Process (*TBMRP*). Usually the *TBMRP* remains unnamed because it is considered part of the standard RL and MDP framework. We call it *TBMRP* to contrast it with the new Real-Time Markov Reward Process (*RTMRP*) that is going to be introduced in the next chapter.

**Definition 2.** *A Turn-Based Markov Reward Process* $(S, \mu, \kappa, \bar{r}) = \mathit{TBMRP}(E, \pi)$ *combines a Markov Decision Process* $E = (S, A, \mu, p, r)$ *with a policy* $\pi$*, such that*

| | | |
|---|---|---|
| *(1) state space* | $S$ | *(is the same as in* $E$*),* |
| *(2) initial distribution* | $\mu$ | *(is the same as in* $E$*),* |
| *(3) transition kernel* | $\kappa(s_{t+1}\|s_t) = \int_A p(s_{t+1}\|s_t, a)\pi(a\|s_t)\ da,$ | |
| *(4) state-reward function* | $\bar{r}(s_t) = \int_A r(s_t, a)\pi(a\|s_t)\ da.$ | |

The Markov process induces a sequence of states $(s_t)_{t\in\mathbb{N}}$ and, together with $\bar{r}$, a sequence of rewards $(r_t)_{t\in\mathbb{N}} = (\bar{r}(s_t))_{t\in\mathbb{N}}$.

**Definition 3.** *The probability density for a sequence of states* $(s_0, ..., s_t)$ *in a Markov Reward Process* $\Omega = (S, \mu, \kappa, \bar{r})$ *can be recursively defined as*

$$p_\Omega^t(s_0, ..., s_t) = \kappa(s_t|s_{t-1})\ p_\Omega^{t-1}(s_0, ..., s_{t-1}) \quad \Big|\ \textit{with } p_\Omega^0 = \mu \tag{1.1}$$

The Reinforcement Learning objective is to find a policy that maximizes the expected sum of rewards. In practice, rewards can be discounted and augmented to guarantee convergence, reduce variance and encourage exploration. However, when evaluating the performance of an agent, we will always use the undiscounted sum of rewards.

## Value Functions on a Markov Reward Process

One of the most important functions in Reinforcement Learning is the value function. The value function measures the expected future reward for each state and can thus be used to guide a learning agent towards higher value states. State-value functions are a property of a Markov Reward Process (they are fully defined by it). There are multiple flavours of value functions, some of which are defined below.

**Definition 4.** *The $n$-step value function $v_\Omega^n$ of a MRP $\Omega = (S, \mu, \kappa, \bar{r})$ is the expected sum over the $n$ next rewards, i.e.*

$$v_\Omega^n(s_t) = \bar{r}(s_t) + \int_S \kappa(s_{t+1}|s_t) v_\Omega^{n-1}(s_{t+1})\ ds_{t+1} \quad \Big|\ \textit{with } v_\Omega^1 = \bar{r} \tag{1.2}$$

$$= \mathbb{E}[\sum_{i=0}^{n} \bar{r}(s_i) | s_{i>0} \sim \Omega, s_0 = s]. \tag{1.3}$$

Besides being useful in multiple other ways, the $n$-step value function is one way of ensuring the finiteness of the sum of rewards. Another way, that is usually preferred, is to exponentially discount rewards as follows.

**Definition 5.** *The $\gamma$-discounted value function $v_\Omega$ of a MRP $\Omega = (S, \mu, \kappa, \bar{r})$ is the expected infinite discounted sum over future rewards, i.e.*

$$v_\Omega(s_t) = \bar{r}(s_t) + \gamma \int_S \kappa(s_{t+1}|s_t) v_\Omega(s_{t+1})\ ds_{t+1} \tag{1.4}$$

$$= \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i\ \bar{r}(s_i) | s_{i>0} \sim \Omega, s_0 = s] \tag{1.5}$$

### Action-Value Functions

Apart from state-value functions there are also action-value functions which additionally condition on an action, i.e. they measure the expected future reward for each state-action pair. However, since the Markov Reward Process, in its most basic form, only models the state evolution (e.g. the actions are marginalized out in the *TBMRP*), the action-value function needs access to the Markov Decision Process and the policy. Furthermore, it assumes that agent and environment interact in a turn-based manner (as in a *TBMRP*).

**Definition 6.** *The $\gamma$-discounted action-value function $q_E^\pi$ in an environment $E = (S, A, \mu, p, r)$ with a policy $\pi$ is the expected infinite discounted sum over future rewards while in $s_t$, applying $a_t$ and then evolving according to*

$$q_E^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \int_{S \times A} \pi(a_{t+1}|s_{t+1})\ p(s_{t+1}|s_t, a_t) q_E^\pi(s_{t+1}, a_{t+1})\ d(s_{t+1}, a_{t+1}) \tag{1.6}$$

$$= r(s, a) + \gamma \int_S p(s'|s, a)\ v_{TBMRP(E,\pi)}(s')\ ds' \tag{1.7}$$

Even though action-value is the basis of many successful Reinforcement Learning algorithms such as Q-learning [Watkins and Dayan, 1992] or DQN [Mnih et al., 2015], there are some problems. One problem that we will encounter and solve in the next chapter is that the action-value function is not defined for non-*TBMRP* interaction schemes. Another problem is that it is fundamentally timestep dependent and it is undefined in the limit of infinitely small timesteps [Tallec et al., 2019].

## Transforming and Comparing Markov Reward Processes

For the proof of Theorem 2, we need to manipulate and compare two Markov Reward Processes that run at different timescales. Below, we introduce the necessary definitions for that. This section can be skipped if the reader is not interested the proof.

**Definition 7.** *The $n$-step transition function of a MRP $\Omega = (S, \mu, \kappa, \bar{r})$ is*

$$\kappa^n(s_{t+n}|s_t) = \int_S \kappa(s_{t+n}|s_{t+n-1})\kappa^{n-1}(s_{t+n-1}|s_t)\ ds_{t+n-1}. \quad \Big|\ \textit{with } \kappa^1 = \kappa \tag{1.8}$$

**Definition 8.** *$\boldsymbol{\Omega} = (Z, \nu, \boldsymbol{\kappa}, \bar{\boldsymbol{r}})$ is a sub-MRP of $\Psi = (Z, \nu, \sigma, \bar{\rho})$ if its states are sub-sampled with interval $n \in \mathbb{N}$ and rewards are summed over each interval, i.e. for almost all $z$*

$$\boldsymbol{\kappa}(z'|z) = \kappa^n(z'|z) \quad \textit{and} \quad \bar{\boldsymbol{r}}(z) = v^n_\Psi(z). \tag{1.9}$$

**Definition 9.** *A MRP $\Omega = (S, \mu, \kappa, \bar{r})$ is a reduction of $\boldsymbol{\Omega} = (Z, \nu, \boldsymbol{\kappa}, \bar{\boldsymbol{r}})$ if there is a state transformation $f : \boldsymbol{Z} \to S$ that neither affects the evolution of states nor the rewards, i.e.*

| | |
|---|---|
| *(1) state space* | $S = \{f(z) : z \in Z\}$, |
| *(2) initial distribution* | $\mu(s) = \int_{f^{-1}(s)} \nu(z)dz$, |
| *(3) transition kernel* | $\kappa(s_{t+1}\|s) = \int_{f^{-1}(s_{t+1})} \boldsymbol{\kappa}(z'\|z)\ dz'$ *for almost all* $z \in f^{-1}(s)$, |
| *(4) state-reward function* | $r(s) = \bar{\boldsymbol{r}}(z)$ *for almost all* $z \in f^{-1}(s)$. |

**Definition 10.** *A MRP $\Psi$ contains another MRP $\Omega$ (we write $\Omega \propto \Psi$) if $\Psi$ works at a higher frequency and has a richer state that $\Psi$ but behaves otherwise identically. More precisely,*

$$\Omega \propto \Psi \iff \Omega \textit{ is a reduction (Def. 9) of a sub-MRP (Def. 8) of } \Psi. \tag{1.10}$$

**Reaction Time in Reinforcement Learning**

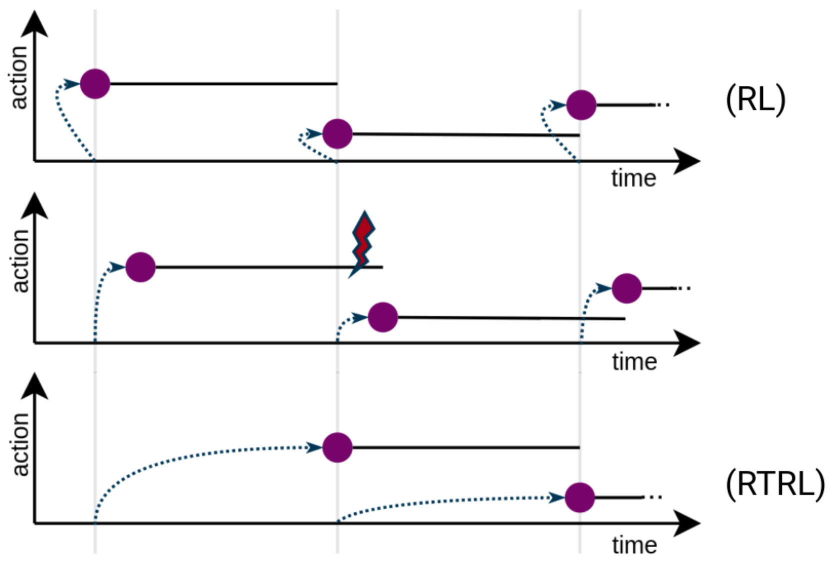


**Figure 1.2:** The action selection process from observations to actions. The vertical grey bars mark observation times and the horizontal, black bars mark the time in which a particular action influences the next observation.

In this thesis we propose a new interaction scheme (RTRL) in which the agent is granted one full timestep to react to an observation. This stands in contrast to the traditional Reinforcement Learning framework (RL) in which the agent has to react instantaneously.

In Figure 1.2 we explore different action selection times. The top graph shows the traditional framework. In the graph in the middle the agent selects an action within a fraction of a timestep. Here, as in the traditional framework, each action affects the immediately following observation. However, it is unclear if and how each action affects the observation after that (thus the red lightning bolt).

The bottom graph shows our proposed framework in which the action only affects the next observation but one. This is conceptually and mathematically much more convenient and still covers the most important use case of back-to-back action selection, as we explain in Section 2.3.1.

# 2 Real-Time Reinforcement Learning

**Authors:** Simon Ramstedt, Christopher Pal

This chapter presents work that has been accepted at the Neural Information Processing Systems (NeurIPS 2019) Confenference.

**Contribution:** We point out problems with the current agent-environment interaction scheme in Reinforcement Learning and propose a new real-time interaction scheme. Furthermore we demonstrate the usefulness of said scheme by creating a new state-of-the-art continuous control algorithm on its basis.

**Affiliation**

- Simon Ramstedt, Mila, University of Montreal
- Christopher Pal, Mila, Polytechnique Montreal

In contrast to the paper, all proofs and information from the paper's appendix are given in the main text and marked with a gray background. Some additional definitions were already given in this thesis' introduction.

## 2.1 Introduction

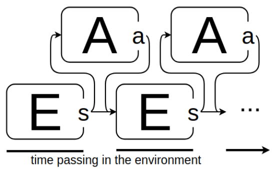


**Figure 2.1:** Turn-based interaction

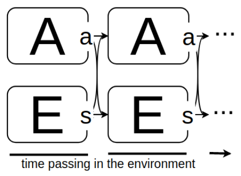


**Figure 2.2:** Real-time interaction

Deep Reinforcement Learning, that is Reinforcement Learning with deep neural networks as the underlying function approximators, has led to great successes in games [Tesauro, 1994, Mnih et al., 2015, Silver et al., 2017] and in real-world robotic control [Schulman et al., 2015, Hwangbo et al., 2019].

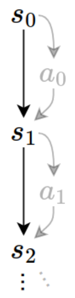
At the same time, all of these methods rely on Turn-Based Markov Reward Processes as underlying interaction framework. Turn-Based Markov Reward Processes are a perfect fit for turn-based decision problems such as board games but less suited for real-time applications in which the environment's state continues to evolve while the agent selects an action. Nevertheless algorithms based on the Turn-Based Markov Reward Process have been used for these real-time problems using what are essentially tricks, e.g. pausing a simulated environment during action selection or ensuring that the time required for action selection is negligible [Hwangbo et al., 2017].

Instead of relying on such tricks, we propose an augmented decision making framework - Real-Time Reinforcement Learning (RTRL) - in which an agent is allowed to take exactly one time-step to select an action. Surprisingly RTRL allows us to greatly simplify many RL procedures such as actor-critic learning and transition model rollouts. We will leverage RTRL to create Real-Time Actor-Critic (RTAC), our new actor-critic algorithm based on Soft Actor-Critic (SAC) [Haarnoja et al., 2018a] that is better suited for real-time environments. We then show experimentally that RTAC outperforms SAC in both real-time and non-real-time settings.

## 2.2 Background

### 2.2.1 Turn-Based Reinforcement Learning

Recall the turn-based agent-environment interaction scheme:



$$\begin{tikzcd}[row sep=small] s_0 \arrow[d] \arrow[dr, bend left] & \\ & a_0 \arrow[dl, bend left] \\ s_1 \arrow[d] \arrow[dr, bend left] & \\ & a_1 \arrow[dl, bend left] \\ s_2 & \end{tikzcd}$$

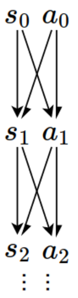
**Figure 2.3:** *TBMRP*

**Definition 11.** *A Turn-Based Markov Reward Process* $(S, \mu, \kappa, \bar{r}) =$ *TBMRP*$(E, \pi)$ *combines a Markov Decision Process* $E = (S, A, \mu, p, r)$ *with a policy* $\pi$*, such that*

$$\kappa(s_{t+1}|s_t)=\int_A p(s_{t+1}|s_t,a)\pi(a|s_t)\; da \quad \textit{and} \quad \bar{r}(s_t)=\int_A r(s_t,a)\pi(a|s_t)\; da. \tag{2.1}$$

We say the interaction is turn-based, because an action selected in a certain state is paired up again with that same state to induce the next state, i.e. the environment's state did not change during the action selection process. This is illustrated in Figure 2.3.

## 2.3 Real-Time Reinforcement Learning



$$\begin{tikzcd}[row sep=large] s_0 \arrow[d] \arrow[dr] & a_0 \arrow[d] \arrow[dl] \\ s_1 \arrow[d] \arrow[dr] & a_1 \arrow[d] \arrow[dl] \\ s_2 & a_2 \end{tikzcd}$$

**Figure 2.4:** *RTMRP*

In contrast to the conventional, turn-based interaction scheme, we propose an alternative, real-time interaction framework in which states and actions evolve simultaneously. Here, agent and environment step in unison to produce new state-action pairs $\boldsymbol{x}_{t+1} = (s_{t+1}, a_{t+1})$ from old state-action pairs $\boldsymbol{x}_t = (s_t, a_t)$ as illustrated in the Figures 2.2 and 2.4.

**Definition 12.** *A Real-Time Markov Reward Process* $(\boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\kappa}, \bar{\boldsymbol{r}}) =$ *RTMRP*$(E, \boldsymbol{\pi})$ *combines a Markov Decision Process* $E = (S, A, \mu, p, r)$ *with a policy* $\pi$*, such that*

$$\boldsymbol{\kappa}(s_{t+1},a_{t+1} \mid s_t,a_t)=p(s_{t+1}|s_t,a_t)\;\boldsymbol{\pi}(a_{t+1}|s_t,a_t) \quad \textit{and} \quad \bar{\boldsymbol{r}}(s_t,a_t)=r(s_t,a_t). \tag{2.2}$$

*The state space* $\boldsymbol{X} = S \times A$ *and* $a_0$ *can be set to some fixed value, i.e.* $\boldsymbol{\mu}(s_0, a_0) = \mu(s_0)\ \delta(a_0 - c)$.[1]

Note that we introduced a new policy $\boldsymbol{\pi}$ that takes state-action pairs instead of just states. That is because the state $(s, a)$ of the RTMRP is now a state-action pair and $s$ alone is not a sufficient statistic of the future of the process anymore.

### 2.3.1 Why is the real-time framework sensible?

Consider the following two time spans:

| | | |
|---|---|---|
| timestep size | $t_s$ | (the time between two observations) |
| action selection time | $t_\pi$ | (e.g. time for a forward pass of the policy network) |

The real-time framework deals with the special case in which $t_s = t_\pi$. In that case an action $a_t$ does not affect the next state $s_{t+1}$, which opens up a number of new algorithmic possibilities. We think $t_s = t_\pi$ is the right assumption because it leads to *back-to-back action selection.* That is, immediately upon finishing to compute an action the next observation is sampled. This should always be the goal, no matter how little time is required to compute an action. It allows the agent to update its actions the quickest, e.g. if we can compute an action in 1ms we should do so 1000 times per second.

### 2.3.2 Real-time interaction can be expressed within the turn-based framework

At first glance, the RTMRP looks quite different from the conventional MRP. However, it is possible to express real-time interaction within the MRP framework, which allows us to reconnect the real-time framework to the vast body of work that has been done using to MRP framework. Specifically, we are trying to find an augmented environment *RTMDP*$(E)$ that behaves the same with turn-based interaction as would $E$ with real-time interaction.

In the real-time framework the agent communicates its action to the environment via the state. However, in the turn-based framework only the environment can directly influence the state. We therefore need to deterministically "pass through"

[1] $\delta$ is the Dirac delta distribution. If $y \sim \delta(\cdot - x)$ then $y = x$ with probability 1.

the action to the next state by augmenting the transition function. Reusing existing variable names if equal, we define the can Real-Time Markov Decision Process (RTMDP). We will denote the actions of the RTMDP with $\boldsymbol{a}_t$ and after they are passed through into the next state they become $a_{t+1}$.

**Definition 13.** *A Real-Time Markov Decision Process* $(\boldsymbol{X}, A, \boldsymbol{\mu}, \boldsymbol{p}, \boldsymbol{r}) = \mathit{RTMDP}(E)$ *augments another Markov Decision Process* $E = (S, A, \mu, p, r)$*, such that (1) the state space* $\boldsymbol{X} = S \times A$*, (2) the action space is* $A$*,*
*(3) the initial state distribution is* $\boldsymbol{\mu}(s_0, a_0) = \mu(s_0)\ \delta(a_0 - c)$*,*
*(4) the transition distribution is* $\boldsymbol{p}(s_{t+1}, a_{t+1} | s_t, a_t, \boldsymbol{a}_t) = p(s_{t+1}|s_t, a_t)\ \delta(a_{t+1} - \boldsymbol{a}_t)$*, and*
*(5) the reward function is* $\boldsymbol{r}(s_t, a_t, \boldsymbol{a}_t) = r(s_t, a_t)$*.*

**Theorem 1.** *A policy* $\boldsymbol{\pi} : A \times \boldsymbol{X} \to \mathbb{R}$ *interacting with* $\mathit{RTMDP}(E)$ *in the conventional, turn-based manner gives rise to the same Markov Reward Process as* $\boldsymbol{\pi}$ *interacting with* $E$ *in real-time, i.e.*

$$\mathit{RTMRP}(E, \boldsymbol{\pi}) = \mathit{TBMRP}(\mathit{RTMDP}(E), \boldsymbol{\pi}) \tag{2.3}$$

*Proof.* For any environment $E = (S, A, \mu, p, r)$ we want to show that the two above MRPs are the same. Per Definition 11 and 13 for $\mathit{TBMRP}(\mathit{RTMDP}(E), \boldsymbol{\pi})$ we have

| | |
|---|---|
| (1) state space | $S \times A$, |
| (2) initial distribution | $\mu(s)\delta(a - c)$, |
| (3) transition kernel | $\int_A p(s_{t+1}\|s_t, a_t)\delta(a_{t+1} - \boldsymbol{a})\ \boldsymbol{\pi}(\boldsymbol{a}\|s_t, a_t)\ d\boldsymbol{a}$, |
| (4) state-reward function | $\int_A r(s, a)\ \boldsymbol{\pi}(\boldsymbol{a}\|s_t, a_t)\ d\boldsymbol{a}$. |

The transition kernel, using the definition of the Dirac delta function $\delta$, can be simplified to

$$p(s_{t+1}|s_t, a_t) \int_A \delta(a_{t+1} - \boldsymbol{a})\ \boldsymbol{\pi}(\boldsymbol{a}|s_t, a_t)\ d\boldsymbol{a} = p(s_{t+1}|s_t, a_t)\ \boldsymbol{\pi}(a_{t+1}|s_t, a_t). \tag{2.4}$$

The state-reward function can be simplified to

$$r(s_t, a_t) \int_A \pi(\boldsymbol{a}|\boldsymbol{x})\ d\boldsymbol{a} = r(s_t, a_t). \tag{2.5}$$

It should now be easy to see how the elements above match *RTMRP*$(E, \boldsymbol{\pi})$, Definition 12. □

Interestingly, the RTMDP is equivalent to a 1-step constant delay MDP (Walsh et al. [2008]). However, we believe the different intuitions behind both of them warrant the different names: The constant delay MDP is trying to model external action and observation delays whereas the RTMDP is modelling the time it takes to select an action. The connection makes sense, though: In a framework where the action selection is assumed to be instantaneous, we can apply a delay to account for the fact that the action selection was not instantaneous after all.

Below we provide a listing of the code used in creating an environment wrapper class for *TBMRP* using the OpenAI gym [Brockman et al., 2016] framework.

```
import gym
from gym.spaces import Tuple

class RealTimeWrapper(gym.Wrapper):
  def __init__(self, env):
    super().__init__(env)
    self.observation_space = Tuple((env.observation_space,
                                    env.action_space))
    self.initial_action = env.action_space.sample()

  def reset(self):
    self.action = self.initial_action
    return super().reset(), self.action

  def step(self, action):
    observation, reward, done, info = super().step(self.action)
    self.action = action
    return (observation, self.action), reward, done, info
```

### 2.3.3 Turn-based interaction can be expressed within the real-time framework

It is also possible to define an augmentation *TBMDP*$(E)$ that allows us to express turn-based environments (e.g. Chess, Go) within the real-time framework (Definition 14). By assigning separate time steps to agent and environment, we can allow the agent to act while the environment pauses. More specifically, we add a binary variable $b$ to the state to keep track of whether it is the environment's or the agent's turn. While $b$ inverts at every time step, the underlying environment only advances every other time step.

**Definition 14.** *A Turn-Based Markov Decision Process* $(Z, A, \nu, q, \rho) = \textit{TBMDP}(E)$ *augments another Markov Decision Process* $E = (S, A, \mu, p, r)$*, such that*

$$\begin{aligned}
&\textit{(1) state space} && Z = S \times \{0,1\}, \\
&\textit{(2) action space} && A, \\
&\textit{(3) initial state distribution} && \nu(\, s_0, b_0\,) = \mu(s_0)\ \delta(b_0), \\
&\textit{(4) transition distribution} && q(\, s_{t+1}, b_{t+1} \mid s_t, b_t\,, a_t) \\
& && = \begin{cases} \delta(s_{t+1} - s_t)\ \delta(b_{t+1} - 1) & \textit{if } b_t = 0 \\ p(s_{t+1}|s_t, a_t)\ \delta(b_{t+1}) & \textit{if } b_t = 1 \end{cases} \\
&\textit{(5) reward function} && \rho(\, s, b\,, a) = r(s,a)\ b.
\end{aligned}$$

**Theorem 2.** *A policy* $\boldsymbol{\pi}(a'|s,b,a) = \pi(a'|s)$ *interacting with* *TBMDP*$(E)$ *in real-time, gives rise to a Markov Reward Process that contains (Def. 10) the MRP resulting from* $\pi$ *interacting with* $E$ *in the conventional, turn-based manner, i.e.*

$$\textit{TBMRP}(E, \pi) \propto \textit{RTMRP}(\textit{TBMDP}(E), \boldsymbol{\pi}) \tag{2.6}$$

*Proof.* Given MDP $E = (S, A, \mu, p, r)$, we have $\Psi = (Z, \nu, \sigma, \bar{\rho}) = \mathit{RTMRP}(\mathit{TBMDP}(E), \boldsymbol{\pi})$ with

$$\text{(1) state space} \qquad Z = S \times \{0,1\} \times A, \tag{2.7}$$

$$\text{(2) initial distribution} \qquad \nu(\, s, b, a \,) = \mu(s)\ \delta(b)\ \delta(a - c), \tag{2.8}$$

$$\text{(3) transition kernel} \qquad \sigma(\, s_{t+1}, b_{t+1}, a_{t+1} \,|\, s_t, b_t, a_t \,) \tag{2.9}$$

$$= \begin{cases} \delta(s_{t+1} - s_t)\ \delta(b_{t+1} - 1)\ \pi(a_{t+1}|s_t) & \text{if } b_t = 0 \\ p(s_{t+1}|s_t, a_t) \quad \delta(b_{t+1}) \quad \pi(a_{t+1}|s_t) & \text{if } b_t = 1 \end{cases}, \tag{2.10}$$

$$\text{(4) state-reward function} \qquad \bar{\rho}(\, s, b, a \,) = r(s, a)\ b. \tag{2.11}$$

We can construct $\boldsymbol{\Omega} = (Z, \nu, \boldsymbol{\kappa}, \bar{\boldsymbol{r}})$, a sub-MRP with interval $n = 2$. Since we always skip the step in which $b = 1$ we only have to define the transition kernel for $b_t = 0$, i.e.

$$\boldsymbol{\kappa}(z_{t+1}|z_t) = \sigma^2(\, s_{t+1}, b_{t+1}, a_{t+1} \,|\, s_t, b_t, a_t \,) \tag{2.12}$$

$$= \int_{S\times A} \sigma(\, s_{t+1}, b_{t+1}, a_{t+1} \,|\, s', 1, a' \,)\ \sigma(\, s', 1, a' \,|\, s_t, 0, a_t \,)\ d(s', a') \tag{2.13}$$

$$= \int_{S\times A} p(s_{t+1}|s', a')\ \delta(b_{t+1})\ \pi(a_{t+1}|s')\ \delta(s' - s_t)\ \pi(a'|s_t)\ d(s', a') \tag{2.14}$$

$$= \int_{A} p(s_{t+1}|s_t, a')\ \delta(b_{t+1})\ \pi(a'|s_t)\ da'. \tag{2.15}$$

For the state-reward function we have (again only considering $b = 0$)

$$\bar{\boldsymbol{r}}(\, s, b, a \,) = v_\Psi^2(\, s, b, a \,) \tag{2.16}$$

$$= \underbrace{\bar{\rho}(\, s, 0, a \,)}_{=0} + \int_{S\times A} \bar{\rho}(\, s', 1, a' \,)\ \sigma(\, s', 1, a' \,|\, s, 0, a \,)\ d(s', a') \tag{2.17}$$

$$= \int_{S\times A} r(s', a')\ \delta(s' - s)\ \pi(a'|s)\ d(s', a') \tag{2.18}$$

$$= \int_{A} r(s, a')\ \pi(a'|s)\ da'. \tag{2.19}$$

The sub-MRP $\boldsymbol{\Omega}$ is already very similar to $\mathit{TBMRP}(E)$ except for having a larger state-space. To get rid of the $b$ and $a$ state components, we reduce $\boldsymbol{\Omega}$ with a state transformation $f(s,b,a) = s$. The reduced MRP has

(1) state space $$\{f(z) : z \in Z\} = S, \tag{2.20}$$

(2) initial distribution $$\int_{f^{-1}(s)} \nu(z)dz = \int_{\{s\}\times\{0,1\}\times A} \mu(s)\delta(b)\delta(a-c)\ d(s,b,a) = \mu(s), \tag{2.21}$$

(3) transition kernel $$\int_{f^{-1}(s_{t+1})} \boldsymbol{\kappa}(z'|z)\ dz' \text{ for almost all } z \in f^{-1}(s_t) \tag{2.22}$$

$$= \int_{\{s_{t+1}\}\times\{0,1\}\times A} \boldsymbol{\kappa}(z'|z)\ dz' \text{ for almost all } z \in \{s_t\} \times \{0,1\} \times A \tag{2.23}$$

$$= \int_A p(s_{t+1}|s_t, a')\ \pi(a'|s_t)\ da', \tag{2.24}$$

(4) state-reward function $$\bar{\boldsymbol{r}}(z) \text{ for almost all } z \in f^{-1}(s) \tag{2.25}$$

$$= \int_A r(s,a')\ \pi(a'|s)\ da', \tag{2.26}$$

which is exactly *TBMRP*$(E)$. □

As a result, not only can we use conventional algorithms in the real-time framework but we can use algorithms built on the real-time framework for all turn-based problems.

## 2.4 Reinforcement Learning in Real-Time Markov Decision Processes

Having established the RTMDP as a compatibility layer between conventional RL and RTRL, we can now look how existing theory changes when moving from an environment $E$ to $\mathit{RTMDP}(E)$.

Since most RL methods assume that the environment's dynamics are completely unknown, they will not be able to make use of the fact the we precisely know part of the dynamics of RTMDP. Specifically they will have to learn from data, the effects of the "feed-through" mechanism which could lead to much slower learning and worse performance when applied to an environment $\mathit{RTMDP}(E)$ instead of $E$. This could especially hurt the performance of off-policy algorithms which have been among the most successful RL methods to date [Mnih et al., 2015, Haarnoja et al., 2018a] since they can leverage old experience collected under different policies. Most off-policy methods make use of the action-value function.

**Definition 15.** *The action value function $q_E^\pi$ for an environment $E = (S, A, \mu, p, r)$ and a policy $\pi$ can be recursively defined as*

$$q_E^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim p(\cdot|s_t, a_t)}[\mathbb{E}_{a_{t+1} \sim \pi(\cdot|s_{t+1})}[q_E^\pi(s_{t+1}, a_{t+1})]] \tag{2.27}$$

When this identity is used to train an action-value estimator, the transition $s_t, a_t, s_{t+1}$ can be sampled from a replay memory containing off-policy experience while the next action $a_{t+1}$ is sampled from the policy $\pi$.

**Lemma 1.** *In a Real-Time Markov Decision Process for the action-value function we have*

$$\begin{aligned} &q^{\boldsymbol{\pi}}_{RTMDP(E)}(s_t, a_t, \boldsymbol{a}_t) \\ &= r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim p(\cdot|s_t, a_t)}[\mathbb{E}_{\boldsymbol{a}_{t+1} \sim \boldsymbol{\pi}(\cdot| s_{t+1}, \boldsymbol{a}_t)}[q^{\boldsymbol{\pi}}_{RTMDP(E)}(s_{t+1}, \boldsymbol{a}_t, \boldsymbol{a}_{t+1})]] \end{aligned} \tag{2.28}$$

*Proof.* After starting with the definition of the action-value function for an environment $(\boldsymbol{X}, A, \boldsymbol{\mu}, \boldsymbol{p}, \boldsymbol{r}) = \mathit{RTMDP}(E)$ with $E = (S, A, \mu, p, r)$, we separate the transition distribution $\boldsymbol{p}$ into its two constituents $p$ and $\delta$ and then, integrate over the Dirac delta.

$$q^{\boldsymbol{\pi}}_{RTMDP(E)}(\boldsymbol{x}_t,\boldsymbol{a}_t) = q^{\boldsymbol{\pi}}_{RTMDP(E)}(\,s_t,a_t\,,\boldsymbol{a}_t) \tag{2.29}$$

$$= \boldsymbol{r}(\,s_t,a_t\,,\boldsymbol{a}_t)+\mathbb{E}_{s_{t+1},a_{t+1}\sim\boldsymbol{p}(\cdot|\,s_t,a_t\,,\boldsymbol{a}_t)}\underbrace{[\mathbb{E}_{\boldsymbol{a}_{t+1}\sim\boldsymbol{\pi}(\cdot|\,s_{t+1},a_{t+1}\,)}[q^{\boldsymbol{\pi}}_{RTMDP(E)}(\,s_{t+1},a_{t+1}\,,\boldsymbol{a}_{t+1})]]} \tag{2.30}$$

$$= r(s_t,a_t) \quad + \quad \int_S p(s_{t+1}|s_t,a_t) \quad \int_A \delta(a_{t+1}-\boldsymbol{a}_t) \qquad\qquad \dots \qquad da_{t+1}\; ds_{t+1} \tag{2.31}$$

$$= r(s_t,a_t)+\int_S p(s_{t+1}|s_t,a_t)\;\; \mathbb{E}_{\boldsymbol{a}_{t+1}\sim\boldsymbol{\pi}(\cdot|\,s_{t+1},\boldsymbol{a}_t)}[q^{\boldsymbol{\pi}}_{RTMDP(E)}(\,s_{t+1},\boldsymbol{a}_t\,,\boldsymbol{a}_{t+1})]\;\; ds_{t+1} \tag{2.32}$$

$\square$

Notice that the action $\boldsymbol{a}_t$ does not affect the reward nor the next state. The only thing that $\boldsymbol{a}_t$ does affect is $a_{t+1}$ which, in turn, only in the next time step will affect $r(s_{t+1}, a_{t+1})$ and $s_{t+2}$. To learn the effect of an action on $E$ (specifically the future rewards), we now have to perform two updates where previously we only had to perform one. We will investigate the effect of this experimentally in Section 2.7.1.

## 2.4.1 Learning the state-value function off-policy

The state-value function can usually not be used in the same way as the action-value function for off-policy learning.

**Definition 16.** *The state-value function $v^\pi_E$ for an environment $E = (S, A, \mu, p, r)$ and a policy $\pi$ is*

$$v^\pi_E(s_t) = \mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[v^\pi_E(s_{t+1})]] \tag{2.33}$$

The definition shows that the expectation over the action is taken *before* the expectation over the next state. When using this identity to train a state-value estimator, we cannot simply change the action distribution to allow for off-policy learning since we have no way of resampling the next state.

**Lemma 2.** *In a Real-Time Markov Decision Process for the state-value function* *we have*

$$v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t, a_t)}[v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_{t+1}, \boldsymbol{a}_t)]]. \tag{2.34}$$

*Proof.* We follow the same procedure as for Lemma 1.

$$v^{\boldsymbol{\pi}}_{RTMDP(E)}(\boldsymbol{x}_t) = v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_t,a_t) \tag{2.35}$$

$$= \mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t,a_t)}[\boldsymbol{r}(s_t,a_t,\boldsymbol{a}_t)+\mathbb{E}_{s_{t+1},a_{t+1}\sim\boldsymbol{p}(\cdot|s_t,a_t,\boldsymbol{a}_t)}[v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_{t+1},a_{t+1})]] \tag{2.36}$$

$$= r(s_t,a_t)+\mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t,a_t)}[\int_S p(s_{t+1}|s_t,a_t) \int_A \delta(a_{t+1}-\boldsymbol{a}_t)\ v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_{t+1},a_{t+1})\ da_{t+1}\ ds_{t+1}] \tag{2.37}$$

$$= r(s_t,a_t)+\int_S p(s_{t+1}|s_t,a_t)\ \ \mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t,a_t)}[v^{\boldsymbol{\pi}}_{RTMDP(E)}(s_{t+1},\boldsymbol{a}_t)]\ \ ds_{t+1} \tag{2.38}$$

□

Here, $s_t, a_t, s_{t+1}$ are always a valid transition no matter what action $\boldsymbol{a}_t$ is selected. Therefore in RTMDPs, we can use the value function for off-policy learning. In fact Equation 2.34 is the same as Equation 2.27 except for the policy inputs. This is suggesting that we can use the state-value function where previously the action-value function was used without having to learn the dynamics of the RTMDP from data since they have already been applied to Equation 2.34.

### 2.4.2 Partial simulation

The off-policy learning procedure described in the previous section can be applied more generally. Whenever parts of the agent-environment system are known and (temporarily) independent of the remaining system, they can be used to generate synthetic experience. More precisely, transitions with a start state $s = (w, z)$ can be generated according to the true transition kernel $\kappa(s'|s)$ by simulating the known part of the transition $(w \to w')$ and using a stored sample for the unknown part of the transition $(z \to z')$. This is only possible if the transition kernel factorizes as $\kappa(w', z'|s) = \kappa_{\text{known}}(w'|s)\ \kappa_{\text{unknown}}(z'|s)$. Hindsight Experience Replay

[Andrychowicz et al., 2017] can be seen as another example of partial simulation. There, the goal part of the state evolves independently of the rest which allows for changing the goal in hindsight. In the next section, we use the same partial simulation principle to compute the gradient of the policy loss.

## 2.5 Real-Time Actor-Critic (RTAC)

Actor-Critic algorithms [Konda and Tsitsiklis, 2000] formulate the RL problem as bi-level optimization where the critic evaluates the actor as accurately as possible while the actor tries to improve its evaluation by the critic. Silver et al. [2014] showed that it is possible to reparameterize the actor evaluation and directly compute the pathwise derivative from the critic with respect to the actor parameters and thus telling the actor how to improve. Heess et al. [2015] extended that to stochastic policies and Haarnoja et al. [2018a] further extended it to the maximum entropy objective to create Soft Actor-Critic (SAC) which RTAC is going to be based on and compared against.

In SAC a parameterized policy $\pi$ (the actor) is optimized to minimize the KL-divergence between itself and the exponential of an (approximate) action-value function $q$ (the critic) normalized by $Z$ (where $Z$ is unknown but irrelevant to the gradient) giving rise to the policy loss

$$L^{\text{SAC}}_{E,\pi} = \mathbb{E}_{s_t \sim D} D_{\text{KL}}(\pi(\cdot|s_t) || \exp(\tfrac{1}{\alpha} q(s_t, \cdot))/Z(s_t)) \tag{2.39}$$

where $D$ is a uniform distribution over a buffer of past states, actions and rewards. The action-value function itself is optimized to fit Equation 2.27 presented in the previous section (augmented with an entropy term). We can thus expect SAC to perform worse in RTMDPs.

In order to create an algorithm better suited for the real-time setting we propose to use a state-value function approximator $\boldsymbol{v}$ as the critic instead, optimized to fit Equation 2.34.

**Proposition 1.** *The following policy loss based on the state-value function*

$$L^{RTAC}_{RTMDP(E),\boldsymbol{\pi}} = \mathbb{E}_{(s_t,a_t)\sim D}\mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)} D_{KL}(\boldsymbol{\pi}(\cdot|s_t,a_t)|| \exp(\tfrac{1}{\alpha}\gamma\boldsymbol{v}(s_{t+1},\cdot))/Z(s_{t+1})) \tag{2.40}$$

*has the same policy gradient as* $L^{SAC}_{RTMDP(E),\boldsymbol{\pi}}$*, i.e.*

$$\nabla_{\boldsymbol{\pi}} L^{RTAC}_{RTMDP(E),\boldsymbol{\pi}} = \nabla_{\boldsymbol{\pi}} L^{SAC}_{RTMDP(E),\boldsymbol{\pi}} \tag{2.41}$$

*Proof.* As shown in Haarnoja et al. [2018a], Equation 2.39 can be reparameterized to obtain the policy gradient, which, applied in a RTMDP, yields

$$\nabla_{\boldsymbol{\pi}} L^{\mathrm{SAC}}_{RTMDP(E),\boldsymbol{\pi}} = \mathbb{E}_{\boldsymbol{x}_t,\epsilon}[\nabla_{\boldsymbol{\pi}}(\log \boldsymbol{\pi}(\boldsymbol{h}_{\boldsymbol{\pi}}(\boldsymbol{x}_t,\epsilon),\boldsymbol{x}_t) - \tfrac{1}{\alpha}\nabla_{\boldsymbol{\pi}} q(\boldsymbol{x}_t, \boldsymbol{h}_{\boldsymbol{\pi}}(\boldsymbol{x}_t,\epsilon))] \tag{2.42}$$

and reparameterizing Equation 2.40 yields

$$\nabla_{\boldsymbol{\pi}} L^{\mathrm{RTAC}}_{RTMDP(E),\boldsymbol{\pi}} = \mathbb{E}_{\boldsymbol{x}_t,\epsilon}[\nabla_{\boldsymbol{\pi}}(\log \boldsymbol{\pi}(\boldsymbol{h}_{\boldsymbol{\pi}}(\boldsymbol{x}_t,\epsilon),\boldsymbol{x}_t) - \tfrac{1}{\alpha}\gamma\nabla_{\boldsymbol{\pi}}\mathbb{E}_{s_{t+1}\sim p(\cdot|\boldsymbol{x}_t)}[\boldsymbol{v}(s_{t+1},\boldsymbol{h}_{\boldsymbol{\pi}}(\boldsymbol{x}_t,\epsilon))]] \tag{2.43}$$

where $\boldsymbol{h}_{\boldsymbol{\pi}}$ is a function mapping from state and noise to an action distributed according to $\boldsymbol{\pi}$. This leaves us to show that

$$\nabla_{\boldsymbol{a}_t} q(\boldsymbol{x}_t,\boldsymbol{a}_t) = \underbrace{\nabla_{\boldsymbol{a}_t}\boldsymbol{r}(\boldsymbol{x}_t, a_t)}_{=0} + \nabla_{\boldsymbol{a}_t}\gamma\mathbb{E}_{\boldsymbol{x}_{t+1}\sim \boldsymbol{p}(\cdot|\boldsymbol{x}_t,\boldsymbol{a}_t)}[\boldsymbol{v}(\boldsymbol{x}_{t+1})] \tag{2.44}$$

$$= \gamma\nabla_{\boldsymbol{a}_t}\mathbb{E}_{s_{t+1}\sim p(\cdot|\boldsymbol{x}_t)}[\boldsymbol{v}(s_{t+1},\boldsymbol{a}_t)] \tag{2.45}$$

which follows from the definition of the soft action-value function and simplifying quantities defined in the RTMDP. □

Note that we need an extra $\gamma$ in the exponential to account for the discounting of the value function. The value function itself is trained off-policy according to the procedure described in Section 2.4.1 to fit an augmented version of Equation 2.34, specifically

$$\boldsymbol{v}_{\mathrm{target}} = r(s_t,a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t,a_t)}[\bar{\boldsymbol{v}}_{\bar{\theta}}((s_{t+1},\boldsymbol{a}_t)) - \alpha\log(\boldsymbol{\pi}(\boldsymbol{a}_t|s_t,a_t))]]. \tag{2.46}$$

Therefore, for the value loss, we have

$$L^{\text{RTAC}}_{RTMDP(E),\boldsymbol{v}} = \mathbb{E}_{(\boldsymbol{x}_t,\boldsymbol{r}_t,s_{t+1})\sim D}[(\boldsymbol{v}(\boldsymbol{x}_t) - \boldsymbol{v}_{\text{target}})^2] \tag{2.47}$$

To trade off between the value function and policy loss, we introduce an additional hyper-parameter $\beta$.

$$L(\theta) = \beta L^{\text{RTAC}}_{RTMDP(E),\boldsymbol{\pi}_\theta} + (1-\beta) L^{\text{RTAC}}_{RTMDP(E),\boldsymbol{v}_\theta} \tag{2.48}$$

### 2.5.1 Merging Actor and Critic

Using the state-value function as the critic has another advantage: When evaluated at the same time step, the critic does not depend on the actor's output anymore and we are therefore able to use a single neural network to represent both the actor and the critic. This could speed up learning and even improve generalization, but could also lead to greater instability. In Section 2.7, we compare RTAC with both merged and separate actor and critic networks.

### 2.5.2 Stabilizing learning

**Algorithm 1:** Real-Time Actor-Critic

```
Initialize parameter vectors $\theta, \bar{\theta}$
for each iteration do
    for each environment step do
        $a_{t+1} \sim \pi(\cdot|s_t, a_t)$
        $s_{t+1} \sim p(\cdot|s_t, a_t)$
        $D \leftarrow D \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    for each gradient step do
        $\theta \leftarrow \theta + \lambda \nabla_\theta L(\theta)$    Eqn. 2.48
        $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$
```

Actor-Critic algorithms are known to be unstable during training. We use a number of techniques that help make training more stable. Most notably we use Pop-Art output normalization [van Hasselt et al., 2016] to normalize the value targets. This is necessary if $v$ and $\pi$ are represented using an overlapping set of parameters. Since the scale of the error gradients of the value loss is highly non-stationary it is hard to find a good trade-off between policy and value loss ($\beta$). If $v$ and $\pi$ are separate, PopArt matters less, but still improves performance both in SAC as well as in RTAC.

Another difficulty are the recursive value function targets. Since we try to maximize the value function, overestimation errors in the value function approximator are amplified and recursively used as target values in the following optimization steps. As introduced by Fujimoto et al. [2018] and like SAC, we will use two value function approximators and take their minimum when computing the target values to reduce value overestimation, i.e. $\bar{\boldsymbol{v}}_{\bar{\theta}}(\cdot) = \min_{i \in \{1,2\}} \ \boldsymbol{v}_{\bar{\theta},i}(\cdot)$.

Lastly, to further stabilize the recursive value function estimation, we use target networks that slowly track the weights of the network [Mnih et al., 2015, Lillicrap et al., 2015], i.e. $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$. The tracking weights $\bar{\theta}$ are then used to compute $\boldsymbol{v}_{\text{target}}$ in Equation 2.46.

## 2.6 Related work

While Firoiu et al. [2018] apply a multi-step action delay to level the playing field between humans and artificial agents, it does not address the issue of turn-based interaction and the significance and consequences of the one-step delay. Similar to RTAC, NAF [Gu et al., 2016] is able to do continuous control with a single neural network. However it is requiring the action-value function to be quadratic in the action (and thus possible to optimize in closed form). This assumption is quite restrictive and could not outperform more general methods such as DDPG. In SVG(1) [Heess et al., 2015] a differentiable transition model is used to compute the path-wise derivative of the value function one time step after the action selection. This is similar to what RTAC is doing when using value function to compute the policy gradient. However in RTAC, we use the actual differentiable dynamics of the RTMDP, i.e. "passing through" the action to the next state, and therefore we do not need to approximate the transition dynamics. At the same time, transitions for the underlying environment are not modelled at all and instead sampled which is only possible because the actions $\boldsymbol{a}_t$ in a RTMDP only start to influence the underlying environment at the next time step.

## 2.7 Experiments

We compare RTAC to Soft Actor-Critic from Haarnoja et al. [2018a] on the standard OpenAI Gym continuous control benchmark suite [Brockman et al., 2016]. Our SAC agents include both a action-value and a state-value function and use a fixed entropy scale $\alpha$ (as in Haarnoja et al. [2018a] and not in Haarnoja et al. [2018b] although performance is comparable). For a comparison to other algorithms such as DDPG, PPO and TD3 also see Haarnoja et al. [2018a,b].

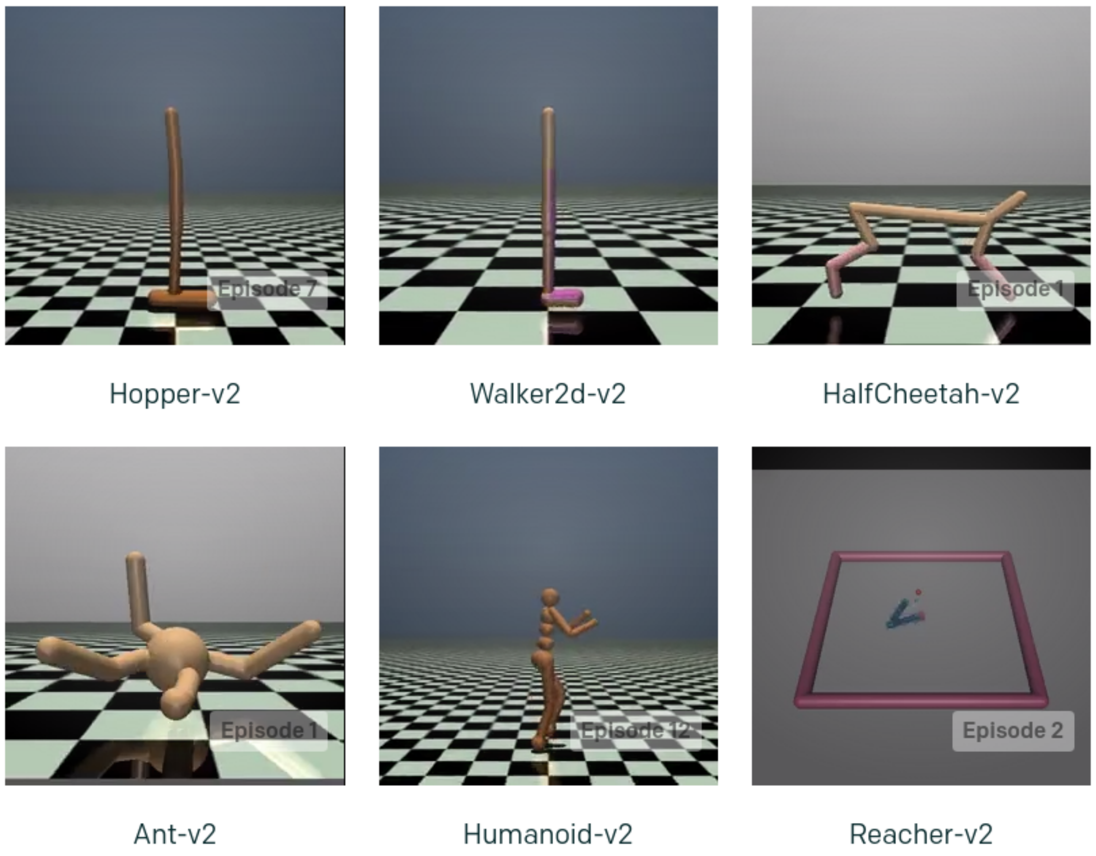


**Figure 2.5:** A collection six, representative MuJoCo tasks from the OpenAI Gym continuous control benchmark suite.

**Implementation** To have a fair comparison we also use output normalization in SAC which improves performance on all tasks (see Figure 2.10 in Appendix 2.8 for a comparison between normalized and unnormalized SAC). The performance of our SAC implementation in the non-real-time environments matches Haarnoja et al.

[2018a,b] almost exactly. Both SAC and RTAC are performing a single optimizer step at every time step in the environment except for the first 10000 time steps. The hyper-parameters used can be found in Table 2.1.

**Figures** All figures show return trends over several runs. For each run, the test return is computed each 20000 time steps as the average return over 100000 time steps using a deterministic policy. For each run the test returns are then smoothed with window size 0.1×number of test returns per run. The return trends show the mean over all runs of the smoothed test returns whereas the shaded region is the 95% confidence interval assuming independently, normally distributed data points with unknown mean and variance.

### 2.7.1 SAC struggles in *RTMDP*$(E)$ as predicted

When comparing the return trends of SAC in turn-based environments $E$ against SAC in real-time environments *RTMDP*$(E)$, the performance of SAC deteriorates. This confirms our hypothesis from Section 2.4.

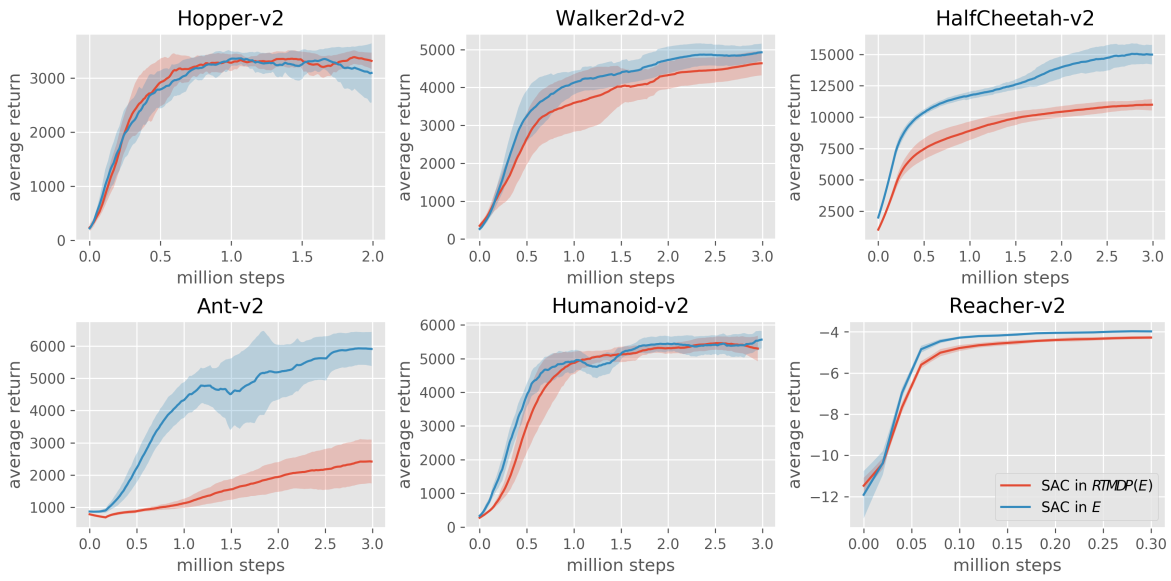


**Figure 2.6:** Return trends for SAC in turn-based environments $E$ and real-time environments *RTMDP*$(E)$. Mean and 95% confidence interval are computed over eight training runs per environment.

### 2.7.2 RTAC is able to cope with real-time environments

Figure 2.7 shows a comparison between RTAC and SAC in real-time versions of the benchmark environments. We can see that RTAC learns much faster and achieves higher returns than SAC in $RTMDP(E)$. This makes sense as it does not have to learn from data the "pass-through" behavior of the RTMDP. We show RTAC with separate neural networks for the policy and value components showing that a big part of RTAC's advantage over SAC is its value function update. However, the fact that policy and value function networks can be merged further improves RTAC's performance as the plots suggest. Note that RTAC is always in $RTMDP(E)$, therefore we do not explicitly state it again.

RTAC is even outperforming SAC in $E$ (when SAC is allowed to act without real-time constraints) in four out of six environments including the two hardest - Ant and Humanoid - with largest state and action space (Figure 2.12). We theorize this is possible due to the merged actor and critic networks used in RTAC. It is important to note however, that for RTAC with merged actor and critic networks output normalization is critical (Figure 2.13).

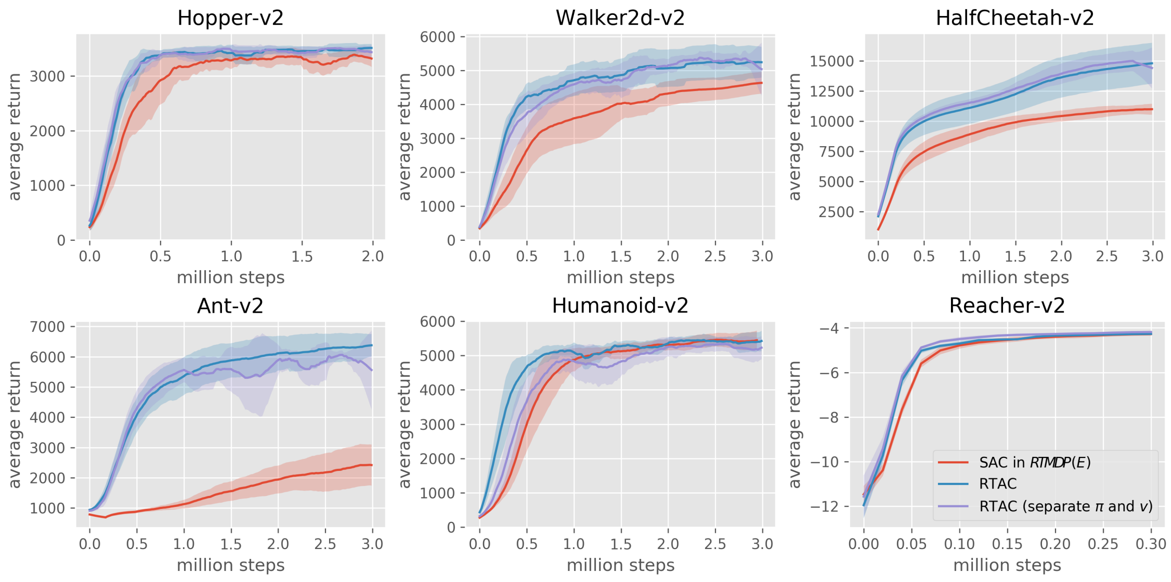


**Figure 2.7:** Comparison between RTAC and SAC in RTMDP versions of the benchmark environments. Mean and 95% confidence interval are computed over eight training runs per environment.

### 2.7.3 Autonomous Driving Task

In addition to the Mujoco benchmark, we have also tested RTAC and SAC on an autonomous driving task using a game-engine-based simulator where the agent controls a car that has to steer around pedestrians. The observations are single image (256x64 RGB pixels) and the car's velocity. The actions are continuous and three dimensional where the first dimension is the steering angle and the others are for accelerating and breaking, respectively. The agent is rewarded proportionally to the car's velocity in the direction of the road and negatively rewarded when making contact with a pedestrian. In addition, episodes are terminated when leaving the road or colliding with any objects or pedestrians. We will provide a citation to a document with more details about this simulator after anonymous peer review.

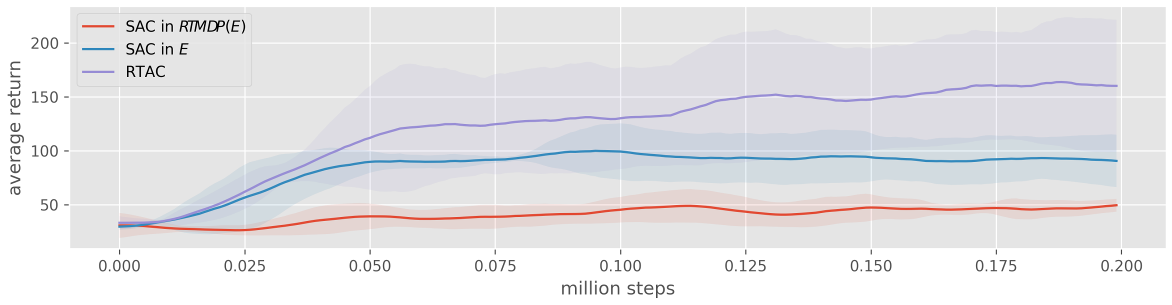


**Figure 2.8:** Comparison between RTAC and SAC in RTMDP versions of the autonomous driving task. We can see that RTAC under real-time constraints outperforms SAC even without real-time constraints. Mean and 95% confidence interval are computed over four training runs per environment.

The hyperparameters used for the autonomous driving task are largely the same as for the OpenAI Gym tasks, however we used a higher reward scale (20) and lower learning rate (0.0001). We used convolutional neural networks with four layers of convolutions with filter sizes $(8, 4, 4, 4)$, strides $(2, 2, 2, 1)$ and 32 channels at each layer. The convolutional layers are followed by two fully connected layers with 256 units each.

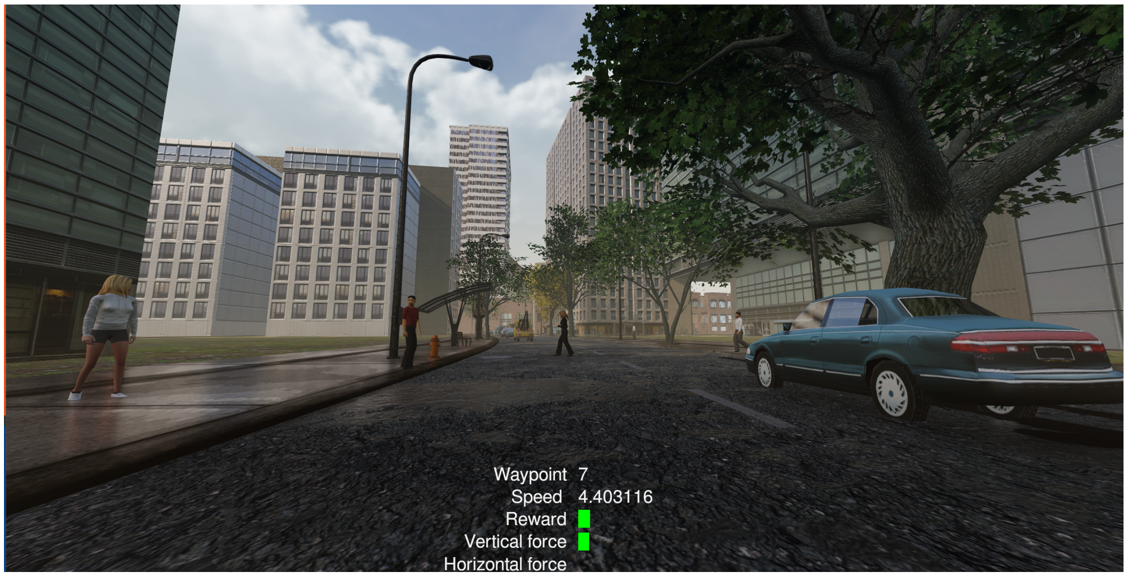


**Figure 2.9:** A screenshot from the driving simulator used for the experiments in Figure 2.8.

## 2.8 Additional Experiments

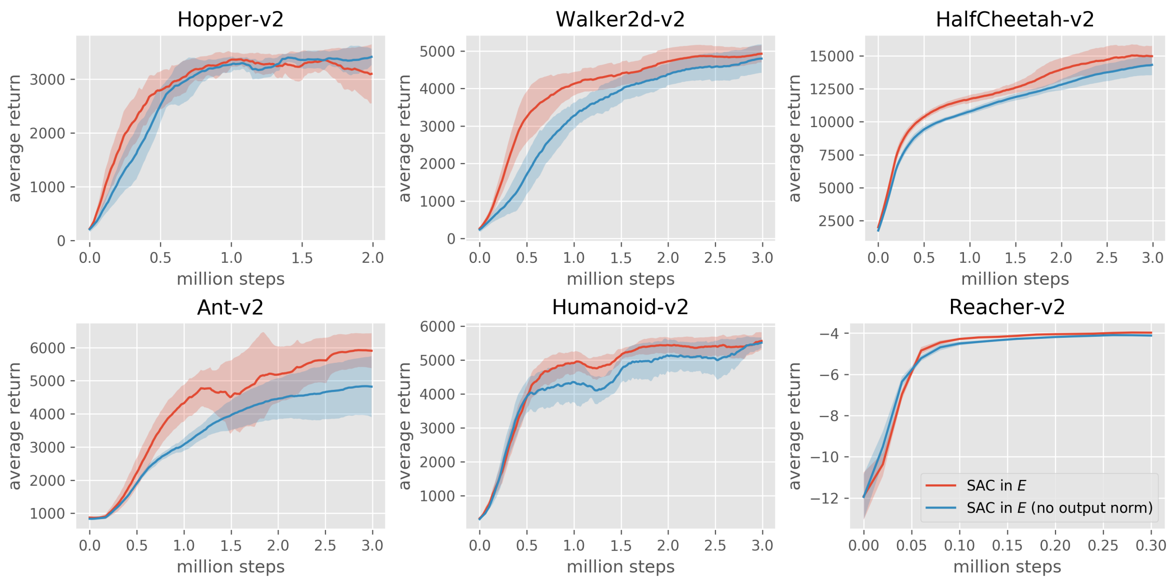


**Figure 2.10:** SAC with and without output normalization. SAC in $E$ (no output norm) corresponds to the canonical version presented in Haarnoja et al. [2018a]. Mean and 95% confidence interval are computed over eight training runs per environment.

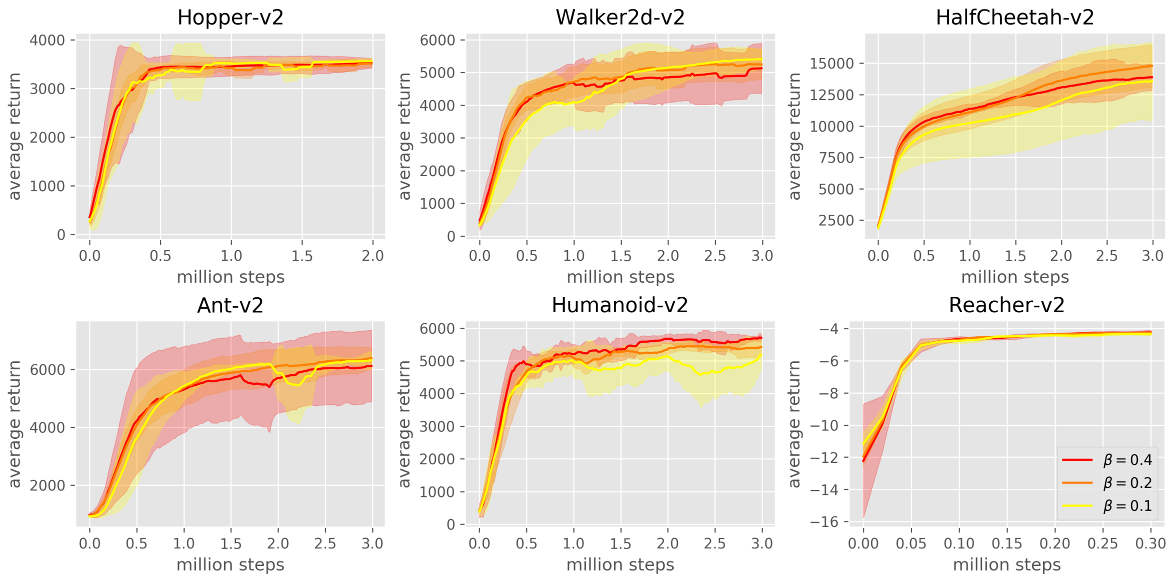


**Figure 2.11:** Comparison between different actor loss scales ($\beta$). Mean and 95% confidence interval are computed over four training runs per environment.

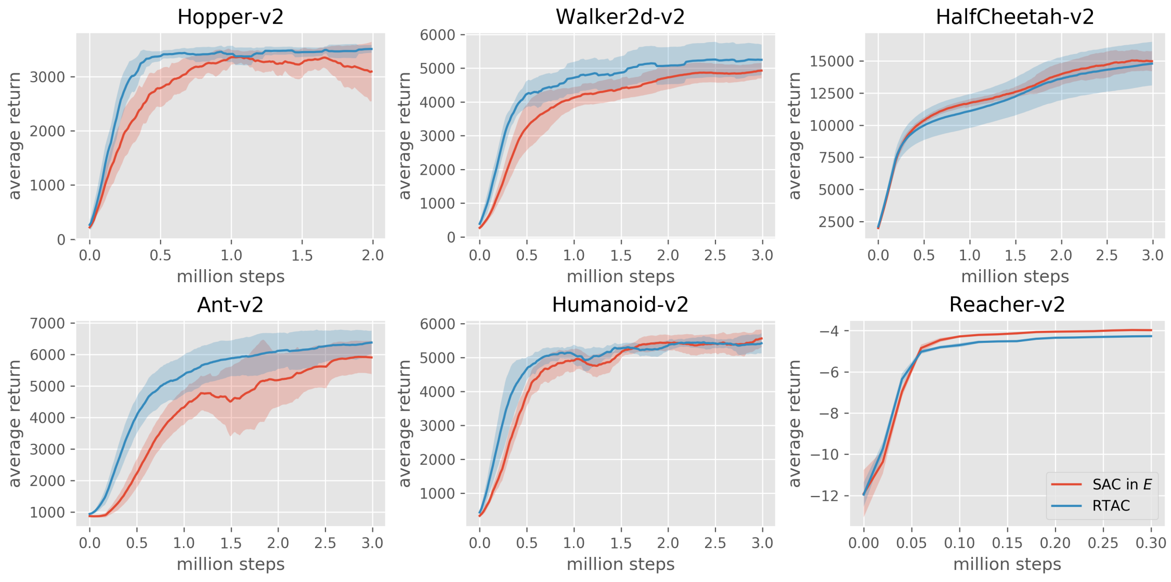


**Figure 2.12:** Comparison between RTAC (real-time) and SAC in $E$ (turn-based). Mean and 95% confidence interval are computed over eight training runs per environment.

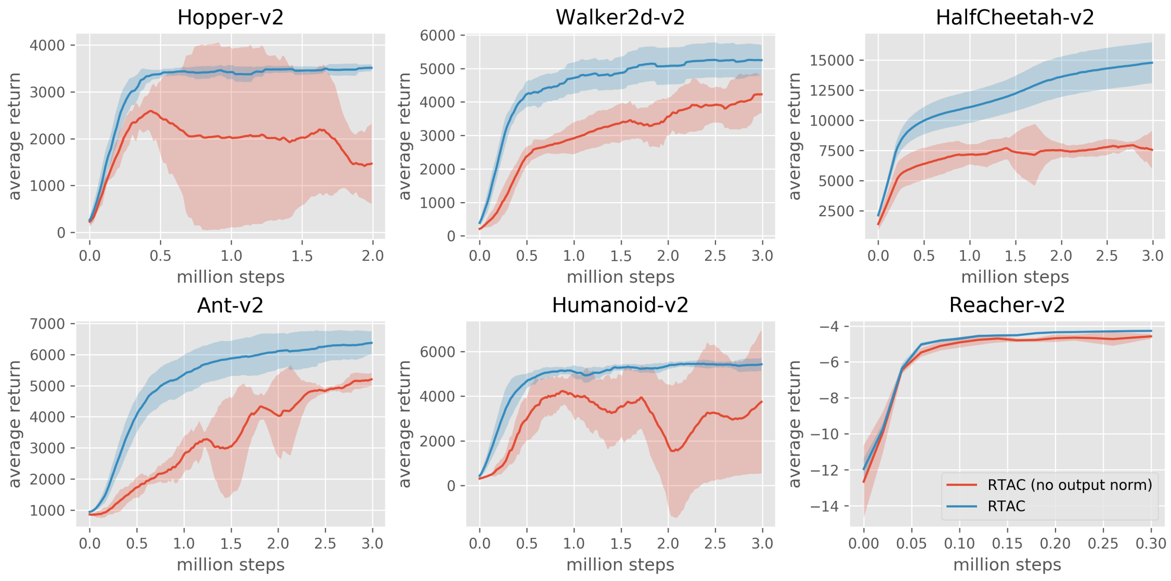


**Figure 2.13:** RTAC with and without output normalization. Mean and 95% confidence interval are computed over eight and four training runs per environment, respectively.

## 2.9 Hyper-parameters

**Table 2.1:** Hyper-parameters

| Name | RTAC | SAC |
|---|---|---|
| optimizer | Adam | Adam [Kingma and Ba, 2014] |
| learning rate | 0.003 | 0.003 |
| discount ($\gamma$) | 0.99 | 0.99 |
| hidden layers | 2 | 2 |
| units per layer | 256 | 256 |
| samples per minibatch | 256 | 256 |
| target smoothing coefficient ($\tau$) | 0.005 | 0.005 |
| gradient steps / environment steps | 1 | 1 |
| reward scale | 5 | 5 |
| entropy scale ($\alpha$) | 1 | 1 |
| actor loss scale ($\beta$) | 0.2 | - |
| PopArt alpha | 0.0003 | - |

# 3 Conclusion

We have introduced a new framework for Reinforcement Learning, RTRL, in which agent and environment step in unison to create a sequence of state-action pairs. We connected RTRL to the conventional Reinforcement Learning framework through the RTMDP and investigated its effects in theory and practice. We predicted and confirmed experimentally that conventional off-policy algorithms would perform worse in real-time environments and then proposed a new actor-critic algorithm, RTAC, that not only avoids the problems of conventional off-policy methods with real-time interaction but also allows us to merge actor and critic which comes with an additional gain in performance. We showed that RTAC outperforms SAC on both a standard, low dimensional continuous control benchmark, as well as a high dimensional autonomous driving task.

# Bibliography

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Vlad Firoiu, Tina Ju, and Joshua B. Tenenbaum. At human speed: Deep reinforcement learning with action delay. *CoRR*, abs/1810.07286, 2018.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2 (4):2096–2103, 2017.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

Joe Marino. An inference perspective on model-based reinforcement learning (slides). URL: https://joelouismarino.github.io/files/papers/2019/variational_rl/icml_workshop_presentation.pdf, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al.

Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Corentin Tallec, Léonard Blier, and Yann Ollivier. Making deep q-learning methods robust to time discretization. *arXiv preprint arXiv:1901.09732*, 2019.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18:83–105, 2008.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.