Real-Time Reinforcement Learning

Simon Ramstedt
Mila, Element AI,
Université de Montréal
simonramstedt@gmail.com

Christopher Pal
Mila, Element AI,
Polytechnique Montréal
christopher.pal@polymtl.ca

Abstract

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 paper, we introduce a new framework, in which states and actions evolve simultaneously and 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. Code and videos can be found at github.com/rmst/rtrl.

Reinforcement Learning, has led to great successes in games (Tesauro, 1994; Mnih et al., 2015; Silver et al., 2017) and is starting to be applied successfully to real-world robotic control (Schulman et al., 2015; Hwangbo et al., 2019).

The theoretical underpinning for most methods in Reinforcement Learning is the Markov Decision Process (MDP) (Bellman, 1957). While the MDP is well-suited to describe turn-based decision problems such as board games, it is ill-suited for real-time applications in which the environment's state continues to evolve while the agent selects an action (Travnik et al., 2018). Nevertheless, MDPs 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).

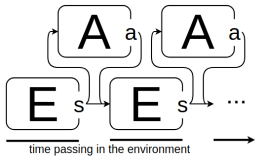

Figure 1: Turn-based interaction

Instead of relying on such tricks, we propose an augmented decision making framework - Real-Time Reinforcement Learning (RTRL) - in which an agent is allowed exactly one time-step to select an action. This keeps it conceptually simple and opens up a number of new algorithmic possibilities.

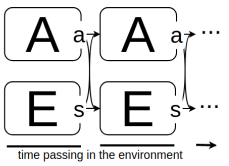

Figure 2: Real-time interaction

We 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.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

1 Background

In Reinforcement Learning the universe is split-up into agent and environment. The agent is represented by a policy – a state-conditioned action distribution, while the environment is represented by Markov Decision Process (Definition 1). Traditionally, the agent-environment interaction has been governed by the MDP framework. Here, even though we use MDPs, we strictly use them to represent the environment. The agent-environment interaction is instead described by different types of Markov Reward Processes (MRP), with the *TBMRP* (Definition 2) behaving like the traditional interaction scheme.

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 $\mu : S \to \mathbb{R}$,
(4) transition distribution $p : S \times S \times A \to \mathbb{R}$, (5) reward function $r : S \times A \to \mathbb{R}$.

An agent-environment system can be condensed into a Markov Reward Process $(S, \mu, \kappa, \bar{r})$ consisting of a Markov process (S, μ, κ) and a state-reward function $\bar{r}$. The Markov process induces a sequence of states $(s_t)_{t\in\mathbb{N}}$ and, together with $\bar{r}$, a sequence of rewards $(r_t)_{t\in\mathbb{N}} = (\bar{r}(s_t))_{t\in\mathbb{N}}$.

As usual, the 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.

1.1 Turn-Based Reinforcement Learning

Usually considered part of the standard Reinforcement Learning framework is the turn-based scheme in which agent and environment interact. We call this interaction scheme Turn-Based Markov Reward Process.

Definition 2. *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 π, such that*

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

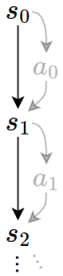

Figure 3: *TBMRP*

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 1.

2 Real-Time Reinforcement Learning

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 and 4.

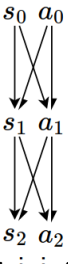

Figure 4: *RTMRP*

Definition 3. *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 π, such that*

$$\boldsymbol{\kappa}(s_{t+1},a_{t+1}|s_t,a_t) = p(s_{t+1}|s_t,a_t)\, \boldsymbol{\pi}(a_{t+1}|s_t,a_t) \quad \textit{and} \quad \bar{\boldsymbol{r}}(s_t,a_t) = r(s_t,a_t). \quad (2)$$

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.

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

2.1 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_π	(e.g. time required for a forward pass through 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.

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

It is possible to express real-time interaction within the standard, turn-based framework, which allows us to reconnect the real-time framework to the vast body of work in RL. Specifically, we are trying to find an augmented environment $\mathit{RTMDP}(E)$ 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" the action to the next state by augmenting the transition function. The *RTMDP* has two types of actions, (1) the actions emitted by the policy $\boldsymbol{a}_t$ and (2) the action component of the state $\boldsymbol{x}_t = (s_t, a_t)$, where $a_t = \boldsymbol{a}_{t-1}$ (with probabilty one).

Definition 4. *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) state space $\boldsymbol{X} = S \times A$*, (2) action space is* A*,*
(3) initial state distribution $\boldsymbol{\mu}(s_0, a_0) = \mu(s_0)\ \delta(a_0 - c)$*,*
(4) transition distribution $\boldsymbol{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t) = \boldsymbol{p}(s_{t+1}, a_{t+1}|s_t, a_t, \boldsymbol{a}_t) = p(s_{t+1}|s_t, a_t)\ \delta(a_{t+1} - \boldsymbol{a}_t)$
(5) reward function $\boldsymbol{r}(s_t, a_t, \boldsymbol{a}_t) = r(s_t, a_t)$*.* *(tap to see code)*

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.*

$$\mathit{RTMRP}(E, \boldsymbol{\pi}) = \mathit{TBMRP}(\mathit{RTMDP}(E), \boldsymbol{\pi}). \tag{3}$$

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.

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

It is also possible to define an augmentation $\mathit{TBMDP}(E)$ that allows us to express turn-based environments (e.g. Chess, Go) within the real-time framework (Definition 5 in the Appendix). 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.

Theorem 2. *A policy* $\boldsymbol{\pi}(a'|s, b, a) = \pi(a'|s)$ *interacting with* $\mathit{TBMDP}(E)$ *in real-time, gives rise to a Markov Reward Process that contains (Def. 8) the MRP resulting from* π *interacting with* E *in the conventional, turn-based manner, i.e.*

$$\mathit{TBMRP}(E, \pi) \propto \mathit{RTMRP}(\mathit{TBMDP}(E), \boldsymbol{\pi}) \tag{4}$$

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.

3 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 which is recursively defined as follows.

$$q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{a_{t+1}\sim\pi(\cdot|s_{t+1})}[q^\pi(s_{t+1}, a_{t+1})]] \tag{5}$$

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

In $\mathit{RTMDP}(E)$ the equation above becomes

$$\boldsymbol{q^\pi}(\boldsymbol{x}_t,\boldsymbol{a}_t)=\boldsymbol{q^\pi}(s_t,a_t,\boldsymbol{a}_t)=r(s_t,a_t)+\mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{\boldsymbol{a}_{t+1}\sim\boldsymbol{\pi}(\cdot|s_{t+1},\boldsymbol{a}_t)}[\boldsymbol{q^\pi}(s_{t+1},\boldsymbol{a}_t,\boldsymbol{a}_{t+1})]] \tag{6}$$

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 5.1.

3.1 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. Its recursive definition

$$v^\pi(s_t) = \mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[v^\pi(s_{t+1})]] \tag{7}$$

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 change the action distribution to allow for off-policy learning since we have no way of resampling the next state.

However, when going from E to $\mathit{RTMDP}(E)$, we have

$$\boldsymbol{v^\pi}(\boldsymbol{x}_t) = \boldsymbol{v^\pi}(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}(\cdot|s_t,a_t)}[\boldsymbol{v^\pi}(s_{t+1},\boldsymbol{a}_t)]]. \tag{8}$$

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 8 is the same as Equation 5 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 8.

4 Real-Time Actor Critic (RTAC)

Actor-Critic algorithms (Konda & 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 π (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

$$L^{\text{SAC}}_{E,\pi} = \mathbb{E}_{s_t \sim D} D_{\text{KL}}(\pi(\cdot|s_t) || \exp(\tfrac{1}{\alpha} q(s_t, \cdot))/Z(s_t))^2 \tag{9}$$

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 5 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 8.

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

$$L^{\textit{RTAC}}_{\textit{RTMDP}(E),\boldsymbol{\pi}} = \mathbb{E}_{(s_t,a_t)\sim D} \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)} D_{\textit{KL}}(\boldsymbol{\pi}(\cdot|s_t,a_t) || \exp(\tfrac{1}{\alpha}\gamma \boldsymbol{v}(s_{t+1},\cdot))/Z(s_{t+1})) \tag{10}$$

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

$$\nabla_{\boldsymbol{\pi}} L^{\textit{RTAC}}_{\textit{RTMDP}(E),\boldsymbol{\pi}} = \nabla_{\boldsymbol{\pi}} L^{\textit{SAC}}_{\textit{RTMDP}(E),\boldsymbol{\pi}} \tag{11}$$

Note that we need an extra γ 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 3.1 to fit an augmented version of Equation 8, specifically

$$\boldsymbol{v}_{\text{target}} = r(s_t,a_t) + \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[\mathbb{E}_{\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot|s_t,a_t)}[\bar{\boldsymbol{v}}_{\bar{\theta}}((s_{t+1},\boldsymbol{a}_t)) - \alpha \log(\boldsymbol{\pi}(\boldsymbol{a}_t|s_t,a_t))]]. \tag{12}$$

Therefore, for the value loss, we have

$$L^{\text{RTAC}}_{\textit{RTMDP}(E),\boldsymbol{v}} = \mathbb{E}_{(\boldsymbol{x}_t,\boldsymbol{r}_t,s_{t+1})\sim D}[(\boldsymbol{v}(\boldsymbol{x}_t) - \boldsymbol{v}_{\text{target}})^2] \tag{13}$$

4.1 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. Merging makes it necessary to trade off between the value function and policy loss. Therefore, we introduce an additional hyper-parameter β.

$$L(\theta) = \beta L^{\text{RTAC}}_{\textit{RTMDP}(E),\boldsymbol{\pi}_\theta} + (1-\beta) L^{\text{RTAC}}_{\textit{RTMDP}(E),\boldsymbol{v}_\theta} \tag{14}$$

Merging actor and critic could speed up learning and even improve generalization, but could also lead to greater instability. In Section 5, we compare RTAC with both merged and separate actor and critic networks.

4.2 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. 14
    $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$
(tap to see code)

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 π 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 (β). If v and π are separate, Pop-Art 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

[2] α is a temperature hyperparameter. For $\alpha \to 0$, the maximum entropy objective reduces to the traditional objective. To compare with the hyperparameters table we have $\alpha = \frac{\text{entropy scale}}{\text{reward scale}}$.

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 12.

5 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 α (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).

Implementation To have a fair comparison we also use output normalization in SAC which improves performance on all tasks (see Figure 9 in Appendix A 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 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 $0.1\times$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.

5.1 SAC struggles in $\mathit{RTMDP}(E)$ as predicted

When comparing the return trends of SAC in turn-based environments E against SAC in real-time environments $\mathit{RTMDP}(E)$, the performance of SAC deteriorates. This confirms our hypothesis from Section 3.

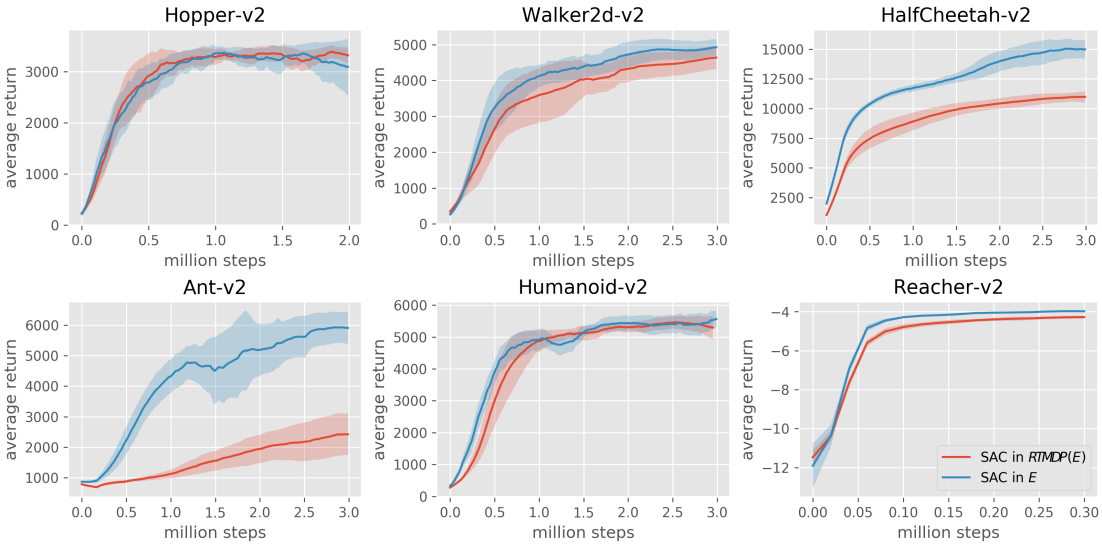

Figure 5: Return trends for SAC in turn-based environments E and real-time environments $\mathit{RTMDP}(E)$. Mean and 95% confidence interval are computed over eight training runs per environment.

5.2 RTAC is able to cope with real-time environments

Figure 6 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 $\mathit{RTMDP}(E)$. 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 $\mathit{RTMDP}(E)$, 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 11). 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 12).

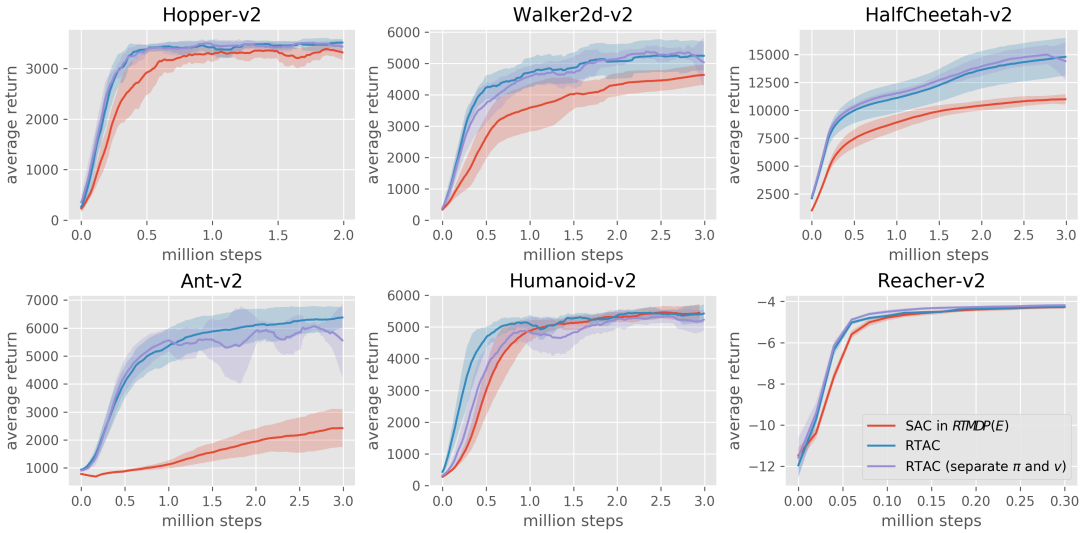

Figure 6: Comparison between RTAC and SAC in RTMDP versions of the benchmark environments. Mean and 95% confidence interval are computed over eight training runs per environment.

5.3 Autonomous Driving Task

In addition to the Mujoco benchmark, we have also tested RTAC and SAC on an autonomous driving task using the Avenue simulator. Avenue is a game-engine-based simulator where the agent controls a car. In the task shown here, the agent has stay on the road and possibly steer around pedestrians. The observations are single image (256x64 grayscale pixels) and the car's velocity. The actions are continuous and two dimensional, representing steering angle and gas-brake. 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 or another car. In addition, episodes are terminated when leaving the road or colliding with any objects or pedestrians.

Figure 7: Left: Agent's view in `RaceSolo-v2`. Right: Passenger view in `CityPedestrians-v1`.

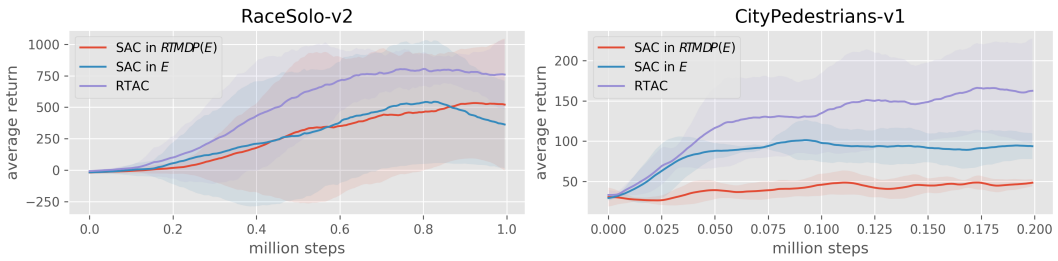

Figure 8: Comparison between RTAC and SAC in RTMDP versions of the autonomous driving tasks. We can see that RTAC under real-time constraints outperforms SAC even without real-time constraints. Mean and 95% confidence interval are computed over four training runs per environment.

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.

6 Related work

Firoiu et al. (2018) applies a multi-step action delay to level the playing field between humans and artificial agents on the ALE (Atari) benchmark However, it does not address the problems arising from the turn-based MDP framework or recognizes the significance and consequences of the one-step action delay. Travnik et al. (2018) noticed that the traditional MDP framework is ill-suited for real-time problems. Other than our paper however, no rigorous framework is proposed as an alternative, nor do they provide any theoretical analysis. 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.

7 Discussion

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.

Acknowledgments

We would like to thank Cyril Ibrahim for for building and helping us with the Avenue simulator. Thank you to Craig Quiter and Sherjil Ozair for insightful discussions about agent-environment interaction and timing. Thank you to Alex Piche, Scott Fujimoto, Bhairav Metha and Jhelum Chakravorty, who read and gave feedback on drafts of this paper. Thank you to Jose Gallego, Olexa Bilaniuk and many others Mila that helped us through countless discussions. This work was completed during a part-time internship at Element AI. The Open Philanthropy Project for provided the financial support for this work.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

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

Travnik, Jaden B., Mathewson, Kory W., Sutton, Richard S., and Pilarski, Patrick M. Reactive reinforcement learning in asynchronous environments. *Frontiers in Robotics and AI*, 5:79, 2018. ISSN 2296-9144. doi: 10.3389/frobt.2018.00079. URL `https://www.frontiersin.org/article/10.3389/frobt.2018.00079`.

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

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