A Framework for Predictable Actor-Critic Control

Josiah Coad
Texas A&M University
josiah@coad.net

James Ault
Texas A&M University
jault@tamu.edu

Jeff Hykin
Texas A&M University
jeff.hykin@tamu.edu

Guni Sharon
Texas A&M University
guni@tamu.edu

Abstract

Reinforcement learning (RL) algorithms commonly provide a one-action plan per time step. Doing this allows the RL agent to quickly adapt and respond to stochastic environments yet it restricts the ability to predict the agent's future behavior. This paper proposes an actor-critic framework that predicts and follows an n-step plan. Committing to the next n actions presents a trade-off between behavior predictability and reduced performance. In order to balance this trade-off, a dynamic plan-following criteria is proposed for determining when it is too costly to follow the preplanned actions and a replanning procedure should be initiated instead. Performance degradation bounds are presented for the proposed criteria when assuming access to accurate state-action values. Experimental results, using several robotics domains, suggest that the performance bounds are also satisfied in the general (approximation) case on expectancy. Additionally, the experimental section presents a study of the predictability versus performance degradation trade-off and demonstrates the benefits of applying the proposed plan-following criteria.

1 Introduction

Deep reinforcement-learning (RL) algorithms are considered state of the art for solving Markov decision processes [12, 28, 20]. Such algorithms can perform at, and even surpass, human-level control in various domains. Examples include robotics [12, 8], traffic management [1, 2], autonomous driving [17, 27], and energy management [10, 33].

Common RL algorithms train a policy which, given the current state of the environment, returns a single action to be applied at the current time step. While allowing the RL agent flexibility to quickly react to changes in stochastic environments, such a one-step planning horizon limits the ability to predict the agent's behavior. This is a limiting characteristic of common RL algorithms, as predictability was shown to be beneficial in multiagent domains [34, 30, 18, 7, 26] and for regulating an RL-agent's performance in safety-critical tasks [3], e.g. a human operator overseeing an autonomous driving controller. Additionally, it is shown to be helpful for an agent to learn a stable policy by relying on its own predicted plan [24].

In this paper we take a first step toward reconciling state-of-the-art RL with n-step planning while bounding the potential performance degradation. The proposed approach, denoted predictable actor-critic (PrAC), trains an environment model approximator which is used to produce and store *imaginary plans* [24]. At each time step, PrAC uses a state-action (Q) value approximator to evaluate the performance degradation affiliated with following the (previously computed) imaginary plan. If the expected future discounted reward is reduced by more than some threshold, ϵ, a replanning

Deep Reinforcement Learning Workshop at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

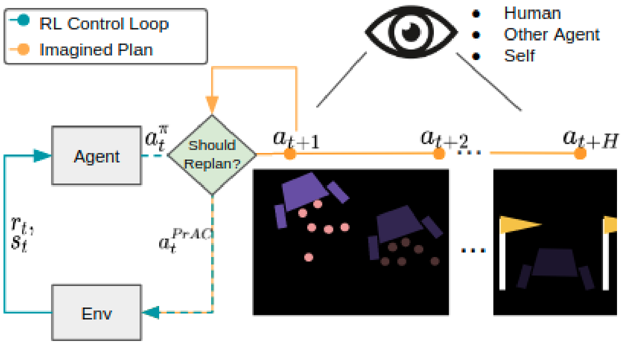

Figure 1: An overview of the PrAC framework. A predicted (imagined) plan is computed in order to provide predictable control. At each step the previously predicted plan is evaluated to determine whether replanning should be initiated at the expense of reduced predictability.

operation is performed and a new (presumably better) plan is followed. The proposed approach is illustrated in Figure 1. We show that, under simplifying assumptions, applying PrAC on top of a static policy would not degrade the expected sum of discounted rewards by more than $\frac{\epsilon}{1-\gamma}$, where γ is the domain discount factor.

Our experimental results show that in some domains, such as LunarLander [4], Walker, and Hopper [6], PrAC results in little or no performance degradation while predicting an average of 12, 5, and 4 actions into the future, respectively. Meanwhile for other domains, such as Cheetah and Humanoid [6], PrAC incurs substantial ($\sim$22%) performance degradation to produce an average of only 3 and 2 forecasted actions, respectively.

1.1 Preliminaries

This paper addresses control problems modeled as *Markov Decision Processes* (MDPs) [23], each with state space, $\mathcal{S}$, action space, $\mathcal{A}$, transition probabilities, P, reward function, R, and discount factor, γ. An agent is assumed to start from state s_0 and select action a_0 according to a *policy*, π. The policy is commonly assumed to be stochastic (soft policy) [12], i.e., mapping states to a distribution over actions. Given a soft policy, an action is selected by sampling the affiliated distribution, $a_t \sim \pi(s_t)$. Based on the chosen action and transition probability, $P(s_{t+1}|s_t, a_t)$, the agent receives a reward, $r_t \sim R(s_t, a_t, s_{t+1})$, from the environment and observes the next state, s_{t+1}. A transition is a tuple of the form (s_t, a_t, r_t, s_{t+1}). A set of consecutive transitions generates a (stochastic) trajectory, $\tau = (s_0, a_0, r_0, s_1, s_1, a_1, r_1, s_2, \ldots)$. A solution for a given MDP is a policy, $\pi^* = \arg\max_\pi V^\pi(s_0)$, where $V^\pi(s) = \mathbb{E}_{\tau\sim\pi|s_0=s}[\sum_{t=0}^{\infty} \gamma^t r_t]$ is called the state value function. In this paper, we quantify an algorithm's *performance* as $V(s_0)$. A state-action (Q) value is defined over a policy, π, as $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi|s_0=s,a_0=a}[\sum_{t=0}^{\infty} \gamma^t r_t]$. Q^* is the Q value obtained under an optimal policy π^*.

State-of-the-art RL algorithms [12] use an actor-critic framework where a state-action value approximator (Q) is trained. Following the Policy-Gradient Theorem [35], the approximated Q values are used to compute and apply a policy gradient step with respect to the expected sum of discounted rewards.

2 Related work

This paper proposes extending the common actor-critic framework to compute and follow n-step plans. In order to do so, we suggest training a model approximator for the underlying state transition probabilities (P). Training a model of the environment, i.e., approximating $P(s'|s,a)$ based on observed transitions, is commonly done as part of a *model-based RL* [15, 36] approach. Such a model can be used to produce imaginary trajectories [24] that improve training sample efficiency in many domains. The imagined trajectory is a simulated future trajectory based on the learned model and current policy. We differentiate between the imaginary trajectory, which is a set of predicted

(s, a, r, s') transitions, and the imaginary plan, which is the affiliated sequence of actions in the imaginary trajectory.

For computing the imaginary trajectory, Racaniere et al. 2017 trained an environment model with a recurrent architecture. The proposed environment model extended on the notion of action-conditional next-step predictors [22, 5, 19], which predicts the next state, and potentially the reward, received after following a given action at a given state (or history of states). It is important to note that modeling errors can compound over long trajectories resulting in an unlikely imaginary trajectory and an affiliated risky imaginary plan. Such modeling errors were shown to be common in complex domains [31, 32].

In order to reduce environment modeling errors, Ke et al. 2018 proposed building a latent-variable autoregressive model [11] by leveraging recent ideas in variational inference [37]. Another recent approach for reducing modeling errors [21] suggests training a prediction model, $\hat{f}_\phi(s_t, a_t)$, that outputs not the next state, s_{t+1}, but the resulting change to the current state, $s_{t+a} - s_t$. The affiliated ($l2$) loss function is defined as $L_\phi(s_t, a_t, s_{t+1}) = \|(s_{t+1} - s_t) - \hat{f}_\phi(s_t, a_t)\|^2$; see Eq2 in Nagabandi et al.

Kim et al. 2020 proposed to utilize imaginary plans towards intention sharing in multiagent RL scenarios. It was demonstrated that broadcasting agents' future intentions (the imagined plans) improve coordination and overall performance over sharing just the current and past states of agents [9, 29, 14, 7].

3 Predictable actor-critic control

As stated above, Racaniere et al. 2017 used imagined trajectories to provide context to model-based algorithms. However, using those trajectories as a planned control sequence presents a challenge. Following the noisy predictions of imaginary trajectories may reduce final performance or prevent policy convergence altogether. This noise or variance in the imagined trajectory is determined by the two components which produce it: the agent's policy and the model of the environment. When the policy is suboptimal or the model is inaccurate, approximation of the trajectory may fail to correctly predict the future. Even with an optimal policy and fully known model, a stochastic environment could push the agent off a predicted trajectory. As such, new predictions must be produced when the current one is evaluated as insufficient.

Perpetually changing the predicted (imaginary) trajectory, however, does not lend well to predicting which actions an agent will take in the future. In our approach, Predictable Actor-Critic (PrAC), we suggest a method of merging these two goals in a generalized framework by reviewing the current plan at each new state and preserving planned steps for as long as some performance degradation threshold is met. A predictable control plan is therefore a trade-off between maximizing the length of predicted plans and minimizing performance degradation. We utilize values learned by the critic component of a Q-learning actor-critic RL algorithm to evaluate trajectories imagined through an approximated model.

Algorithm 1 presents the PrAC framework when applied on top of Soft Actor-Critic (SAC) [12]. PrAC can be extended to other algorithms which learn and store Q-values such as DQN [20]. However, studying the impact of PrAC in such cases is beyond the scope of this paper. We note that SAC might seem incompatible with PrAC as it does not converge on exact Q values but some combination of the Q values and a bias towards high entropy in the Q-value distribution. However, if the temperature parameter (α) in SAC is reduced over time to zero, as proposed by Haarnoja et al. 2018, then the learned Q-values are eventually unbiased.

PrAC receives, as input, a performance degradation tolerance value, ϵ. In line 1, PrAC starts by initializing three function approximators, a policy (π_θ), an action-value function (Q_ψ), and a transition function (P_ϕ). At the beginning of each episode a new plan is imagined (Line 4). The initial plan is a simple rollout of the current policy (π) on top of the current model approximation (P). Next, for each environment step, a first-in-first-out action is retrieved from the imagined plan queue (Line 6) and executed (Line 7). Once the next state is observed, a replan operation is initiated, where the current imaginary plan is reevaluated and adjusted (Line 10). Finally, in Lines 11–14, the policy, state-action approximator, and model approximator are updated using gradient descent. As presented, the policy (π) and state-action approximator (Q) updates follow the SAC, entropy-adjusted loss function. As such, these include a learning rate parameter, λ_Q and λ_π, respectively, as well as the temperature parameter, α. We note that these update rules are not inherent to PrAC and thus refer the reader to Haarnoja et al. 2018 which provides full details regarding these parameters and suggested value assignments. The gradient step update for the model approximator (P) follows the model loss function proposed in Nagabandi et al. 2018 along with a learning rate parameter, λ_P.

Algorithm 2 details how an imaginary plan is evaluated and updated within the *Replan* procedure. As input, it takes the current state of the environment (s), the current imagined plan ($plan$), the policy (π), state-action function (Q), environment model (P), and performance degradation tolerance parameter (ϵ). Each action in the current imagined plan, starting from the earliest planned action and moving forwards in time, is checked against the plan-following criteria. This criteria determines whether an action and its succeeding plan, should still be considered as the projected plan. The proposed criteria compares the state-action value of the preplanned (imaginary) action against the state-action value of an action drawn from the behavior policy (π). For an action to remain in the imaginary (predicted) plan its value must be within ϵ of the value that a newly recalculated imagined plan would have taken. The tolerance parameter ϵ bounds the acceptable degradation in performance for facilitating a predictable control plan. The criteria is checked for each action in Lines 2-4; if passed, the plan is preserved in Line 6. The imaginary plan validation process continues as the following state is sampled from the approximated environment model. It should be noted that the model may predict different state transitions than it predicted when the plan was originally formed. So long as the planed action meets the criteria, no adjustment is necessary. However, once an action is found in violation of the criteria then the future plan from that point onward should be recomputed. The preservation of the plan up to this action provides the predictable portion of the plan.

In Algorithm 2, the imagination core function from Racaniere et al. 2017 is used in Lines 8-11. An action is first sampled from the policy at the final state in the current imaginary plan. Then the state transition is predicted by the environment model given both the state and action. This process could be repeated to produce an arbitrary length imaginary plan.

Line 8 of Algorithm 2 presents a method for slowly building up the length of the imaginary plan (1 step beyond the current plan length per step) as we can consistently follow the plan that we have. However, if we break from our current plan, this logic resets our plan length. This approach is preferable for computational reasons. In cases where computational resources are sufficient, one can set a constant n-step prediction horizon. This would always result in a constant length imaginary plan, even when the model approximation is not reliable enough to allow following lengthy plans. In such cases, the plan following criteria will often truncate the imaginary plan.

Algorithm 1: Predictable Actor-Critic (PrAC)

```
Input   : Suboptimality tolerance $\epsilon$
Initialize: policy ($\pi$) parameters $\theta$, action-value function ($Q$) parameters $\psi$, model approximation
  ($P$) parameters $\phi$, empty replay buffer $\mathcal{D}$
for each episode do
    reset environment: $s \leftarrow s_0$
    $plan \leftarrow$REPLAN($s, plan :=$ empty array, $\pi_\theta, P_\phi, Q_\psi, \epsilon$)
    for each environment step do
        $a \leftarrow dequeue(plan)$
        execute $a$ and observe $r, s'$
        store $(s, a, r, s')$ in $\mathcal{D}$
        $s \leftarrow s'$
        $plan \leftarrow$REPLAN($s, plan, \pi_\theta, P_\phi, Q_\psi, \epsilon$) // reuses as much of the old plan as possible
    for each gradient step do
        $\psi = \psi - \lambda_Q \nabla_\psi [Q(s_t, a_t) - (r_t + \gamma(Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}, a_{t+1})))]$
        $\theta = \theta - \lambda_\pi \nabla_\theta [\alpha \log \pi(a_t|s_t) - Q(s_t, a_t)]$
        $\phi = \phi - \lambda_P \nabla_\phi [P(s_t, a_t) - s_{t+1}]^2$
```

Algorithm 2: Replan

```
Input  : state $s$, current plan $plan$, policy $\pi$, action-value function $Q$, model approximation $P$,
         performance degradation tolerance $\epsilon$
Output : a new plan $plan'$
Initialize an empty array $plan'$
for $t$ in $[0, ..., length(plan) - 1]$ do
    $a \leftarrow plan[t]$
    if $Q(s, a) + \epsilon < Q(s, \pi(\cdot|s))$ then
        break // Following the last plan is $\epsilon$-suboptimal. Abort.
    append $a$ to $plan'$
    $s \sim P(\cdot|s, a)$
for 2 iterations do
    $a \sim \pi(\cdot|s)$
    append $a$ to $plan'$
    $s \sim P(\cdot|s, a)$
return $plan'$
```

3.1 Performance bounds

We show that, under simplifying assumptions, the proposed plan-following criterion bounds the performance degradation incurred by PrAC.

Lemma 1. *Assuming known state-action (Q) values for a policy π, and a discount factor, $0 < \gamma < 1$, applying PrAC on top of π would not degrade the sum of discounted rewards by more than $\frac{\epsilon}{1-\gamma}$.*

Proof. At every state, s, PrAC applies action, a^p, which satisfies:

$$V^{\pi}(s) \leq Q^{\pi}(s, a^p) + \epsilon \tag{1}$$

By definition:

$$Q^{\pi}(s, a^p) = E_{s' \sim P(s'|s, a^p)} \left[R(s, a^p) + \gamma V^{\pi}(s') \right] \tag{2}$$

Combining 1 and 2 yields:

$$V^{\pi}(s) \leq E_{s' \sim P(s'|s, a^p)} \left[R(s, a^p) + \gamma V^{\pi}(s') \right] + \epsilon \tag{3}$$

Equation 3 results in a recursive definition over successive state values $V^{\pi}(s)$ and $V^{\pi}(s')$. Considering the PrAC state-value definition, $V^p(s) = E_{\tau \sim PrAC|s_0=s} \sum_t \gamma^t R(s_t, a^p)$, and expanding the recursive term results in:

$$V^{\pi}(s) \leq V^p(s) + \sum_t \gamma^t \epsilon \tag{4}$$

Assuming $\gamma < 1$, the performance degradation of PrAC can be bounded by $\sum_{t=0}^{\infty} \gamma^t \epsilon = \frac{\epsilon}{1-\gamma}$. □

In many realistic scenarios, it is not reasonable to assume that Q^{π} is known but that it is only approximated. Nonetheless, we present empirical results showing that this bound also holds, in expectancy, for cases where Q^{π} is only approximated, as is the case in actor-critic algorithms.

4 Experimental study

The experimental study is designed towards the following objectives:

1. Investigate to what extent does the theoretical performance bound for PrAC hold when Q-values are approximated.
2. Present the trade-off between performance degradation and predicted plan length in several domains, both for the case when we use PrAC during training and the case where we train the baseline policy and then apply PrAC.

Table 1: State and action space size for each domain in the experimental study.

Environment	Lander	Hopper	Walker	Cheetah	Ant	Humanoid
State Dim.	8	15	22	26	28	44
Action Dim.	2	3	6	6	8	17

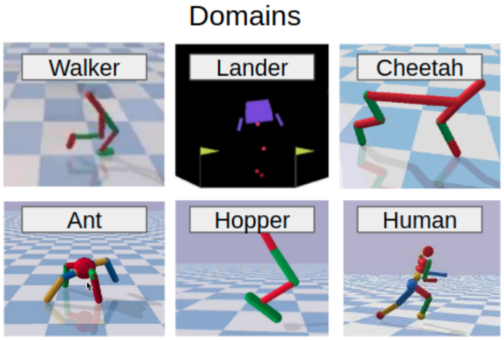

Figure 2: Snapshots from the domains used in the experimental study.

4.1 Domains

Results for six continuous-action domains are presented in this section. These include: HumanoidBulletEnv-v0, HopperBulletEnv-v0, HalfCheetahBulletEnv-v0, Walker2DBulletEnv-v0, and AntBulletEnv-v0. These domains were selected to be similar to those used to study SAC [12]. Additionally, we present results for the LunarLanderContinuous-v2. A snapshot from all six domains can be seen in Figure 2.

The domains included here have state spaces which are vectors of high-level features (i.e. not images). For the PyBullet robotic domains, the state space consists of physics descriptors for the agent. For example, in the Ant domain, the state space consists of the position and orientation of the torso and the joint angles, the velocity of the agent, and the external forces applied to each of the links at the center of mass.

The action spaces in all environments are continuous. In the PyBullet robotic domains, the action spaces are the magnitude of torque applied to each actuator. The state and action space sizes for each domain is given in Table 1.

4.2 Hyper-parameter settings

The reported PrAC implementation utilizes the open-source Stable Baselines [13] implementation of SAC. Stable Baselines maintains a well-documented library of benchmark reinforcement-learning algorithms. The code for our experiments is available at github.com/josiahcoad/AFRL.

In our experiments many hyper-parameters were constant between the domains and were selected to match the tuned parameters made available in the Stable Baselines Zoo [25]. These were as follows: A 3e-4 learning rate for both the actor (λ_π) and the critic (λ_Q), a 1e6 buffer size, 100 random actions before starting learning, a minibatch size for each gradient update of 256, a soft polyak update coefficient (τ) of 0.005, a discount factor (γ) of 0.98, and a training frequency of every step with one gradient step per training and a target network update frequency of every step. The entropy coefficient (α) in SAC, equivalent to the inverse of reward scale in the original SAC paper, was dynamically adjusted using the Stable Baselines implementation.

The Humanoid and Cheetah domains required some parameters to be adjusted from other domains. In the Humanoid domain a batch size of 64 was used and the number of random acts pre-training was raised to 1,000. In the Cheetah domain the number of random acts pre-training was raised to 10,000, learning rate (λ_π, λ_Q) set to 7.3e-4, a 3e-5 buffer size, $\tau = 0.02$, and an update frequency of once every 8 steps with 8 gradient steps. The environment model for all domains uses a small network

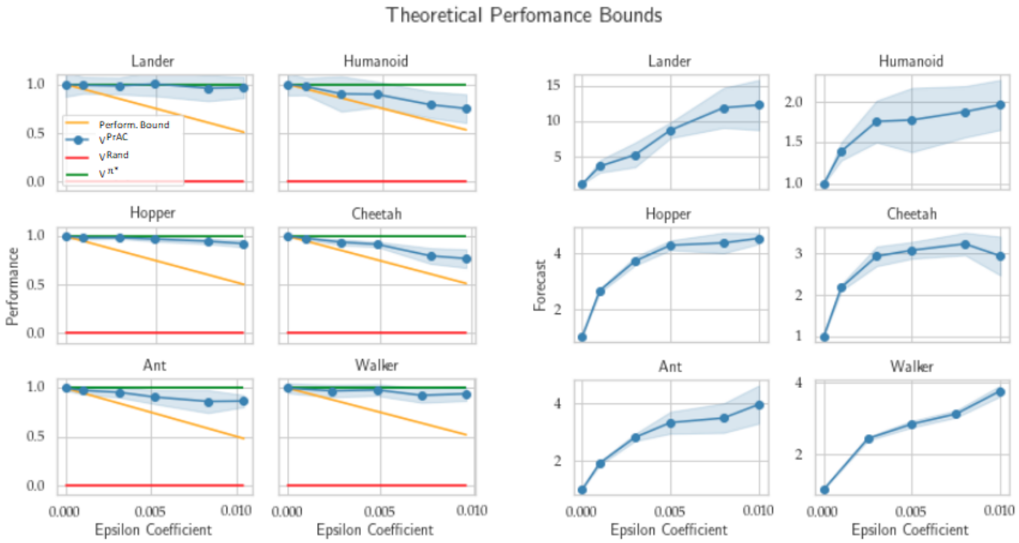

Figure 3: Left: Normalized performance between random performance (red) and optimal performance (green) along with the theoretical performance bounds (yellow) and observed train-then-PrAC performance (blue) for different epsilon coefficient values. Right: The average forecast length in number of steps as a function of the epsilon coefficient value. Shaded regions represent 1 standard deviation over 20 random seeds per setting.

with 4 hidden layers, each with 64 nodes and relu activation functions. The Adam optimizer with learning rate (λ_P) of 1e-4 was used to optimize parameters.

4.3 Computing resources

Modest compute was necessary to conduct our experiments. Amazon EC2 type t3.large instances were used. This type contains 2 vCPUs and 2 GB of memory. Experiments were run on the Amazon Linux 2 operating system and require the following open-source Python packages: NumPy (1.19.5), Gym (0.18.0), Torch (1.8.1), PyBullet (3.1.4) and Stable Baselines3 (1.1.0).

4.4 Performance bounds

We start by investigating the performance degradation introduced by PrAC as a function of the tolerance parameter ϵ. The theoretical analysis in Section 3 relies on several limiting assumptions. However, we find that in practice this bound still holds in empirical evaluation. In order to present a clearer trend over several domains, we report the normalized performance for PrAC where the normalized performance is defined as follows.

Definition 1 (Normalized performance)**.**

$$NP(PrAC) = \frac{V^{PrAC}(s_0) - V^{rand}(s_0)}{V^{\pi^*}(s_0) - V^{rand}(s_0)}$$

V^{rand} *is the expected sum of discounted returns following a random policy.*
V^{π^*} *is the expected sum of discounted returns following a Q-greedy policy, i.e.,* $\pi^*(s) = \arg\max_a Q(s,a)$.

In the left-hand side of Figure 3, the theoretical bound is plotted in yellow as a function of the ϵ coefficient on the horizontal axis with $\gamma = 0.98$. The lower-bound formula derived from Section 3 is used here. The bound decreases as ϵ increases. On the vertical axis, the performance of a random baseline (red) and the SAC policy π value (green) is plotted. The blue line, V^{PrAC}, represents the normalized performance of PrAC. In Lander, Hopper and Walker environments, we observe that the forecast decreases very little while providing a multistep action plan. In the remaining environments,

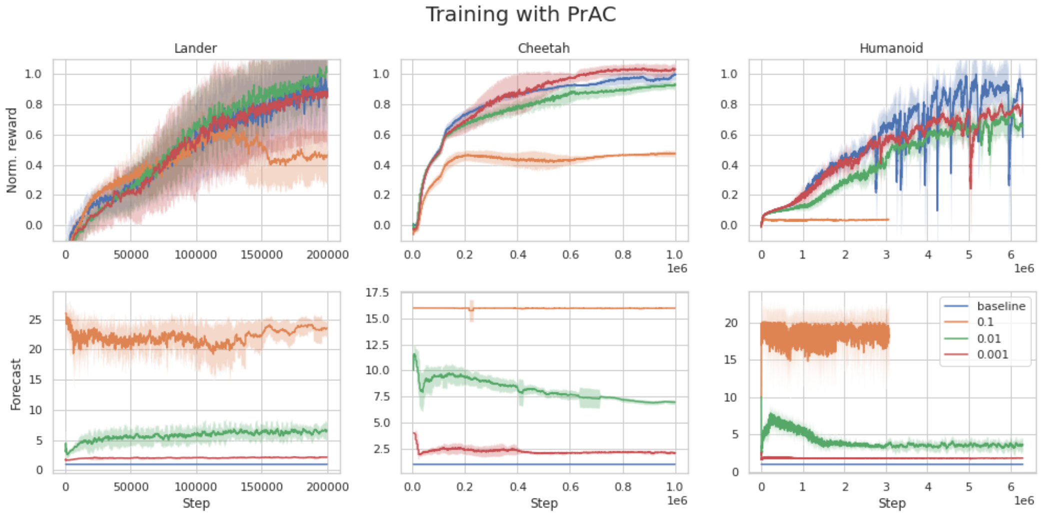

Figure 4: Top: training curves from representative environments. The y axis is the normalized episode reward. Bottom: The associated forecast for the environments. The baseline algorithm shows SAC without any modifications. Experiments run with random seeds for each setting. Shown with 1 std. Mean and std aggregated over rolling window of 20 episodes. The forecast of PrAC(0.1) for cheetah and humanoid has been capped at 16 and 20, respectively, for computational considerations. Humanoid-0.1 is stopped early as it fails to improve beyond random performance.

Humanoid, Cheetah and Ant, performance degrades significantly at higher coefficients. We speculate these environments are more difficult to model due to an increased state space size. Nevertheless, all environments respect the theoretical bounds in expectation.

The hyper-parameter, epsilon, introduced with PrAC has the intuition that $\epsilon = 0$ reduces to a plan generator with no additional care given to retaining the current plan, e.g. the type of plan generators in [24] and [18]. We should expect a very small forecast with $\epsilon = 0$. On the other hand, $\epsilon = \infty$ is a plan generator which never deviates from the current plan and thus would have a high forecast and low performance. The epsilon coefficient equal to $\epsilon/(V^{\pi}(s_0) - V^{rand}(s_0))$ is used to generalize the epsilon hyper-parameter better across domains.

4.5 Predictability-performance trade-off

The design of PrAC leads to a trade-off between agent performance and plan reliability. We seek to characterize this relationship through a series of experiments that showcase both use cases for PrAC. PrAC can be applied both during training and after training. This makes PrAC a versatile addition to any RL agent which learns Q-values. If predictability is desired during training, then training-*with*-PrAC should be considered, otherwise training-*then*-PrAC can be used to apply PrAC to an already fully trained agent. We analyze and report results on both training-*with*-PrAC and training-*then*-PrAC.

Forecast: A measure (scalar) of predictability or how far into the future PrAC provides a reliable plan. This value is computed per time step and in hindsight. When applying action a_t, forecast$_t$ is defined as $f_t = t + 1 - i$, where a_t was originally determined during the $Replan$ procedure (Algorithm 2) at time step i. The forecast value of a full episode (of length T) is defined as $\frac{1}{T}\sum_{t=0}^{T} f_t$.

In Figure 4 the learning curves of 3 representative domains are presented when PrAC's imagined plans are used while the agent is learning. Learning curves for decreasing values of ϵ are shown against the learning curve of the baseline Soft Actor-Critic algorithm.

In the LunarLander domain (the leftmost plot in Figure 4), values smaller than 0.01 achieve the domain goal of a 200 point reward. The rate of reaching this reward is similar to the baseline for each as well. However, PrAC offers a plan forecast of about 5 during this training which provides additional predictability. When the epsilon coefficient is increased to 0.1, forecast length is increased dramatically; but the agent does not replan appropriately, so the episode reward fails to reach the final baseline episode reward.

A similar trend is observed in the Cheetah domain (the center plot in Figure 4). When the ϵ coefficient is large, e.g. 0.1, episode reward is greatly reduced with an improvement in forecasting. Using PrAC with this coefficient is impractical. In this domain, we start to observe the trade-off at epsilon coefficients of 0.01 and 0.001. For a small incurred reward penalty, a forecast is possible. We speculate returns with $epsilon = 0.001$ are slightly higher than the baseline due to randomization in the training process.

With humanoid (the rightmost plot in Figure 4), a large epsilon coefficient obstructs the policy from learning. The episode reward stays consistent with that of a random agent. However, for smaller values of the epsilon coefficient, the agent does improve and obtains a final episode reward within 75% of the baseline agent learned policy.

Overall, we observe a trend across the studied environments—decreased performance yields improved forecasting. This trade-off occurs in both training-with-PrAC and training-then-PrAC.

5 Summary

In this work, we have proposed an easily adopted method to allowing for plan forecasting via action-value evaluations. We prove a lower bound for the performance degradation incurred for following a forecasted plan, under some assumptions. Empirically, we found that the method respects these bounds in expectation when biases are introduced resulting from model approximation errors. Our empirical study also shows a trade-off between plan consistency and performance degradation in a number of environments. Our study suggests the proposed method, Predictable Actor-Critic (PrAC), could be used to produce n-step plans from one-step reinforcement learning methods with bounded degradation.

References

[1] J. Ault and G. Sharon. Reinforcement learning benchmarks for traffic signal control. In *Proceedings of the 35th Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*, December 2021.

[2] J. Ault, J. Hanna, and G. Sharon. Learning an interpretable traffic signal control policy. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2020)*. International Foundation for Autonomous Agents and Multiagent Systems, May 2020.

[3] A. Biondi, F. Nesti, G. Cicero, D. Casini, and G. Buttazzo. A safe, secure, and predictable software architecture for deep learning in safety-critical systems. *IEEE Embedded Systems Letters*, 12(3):78–82, 2020. doi: 10.1109/LES.2019.2953253.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

[6] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[7] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[8] S. Dey, S. Pendurkar, G. Sharon, and J. Hanna. A joint imitation-reinforcement learning framework for reduced baseline regret. In *Proceedings of the 34th International Conference on Intelligent Robots and Systems (IROS 2021)*, September 2021.

[9] J. N. Foerster, Y. M. Assael, N. De Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[10] J. Gao. Machine learning applications for data center optimization. 2014.

[11] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[13] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[14] J. Jiang and Z. Lu. Learning attentional communication for multi-agent cooperation. *arXiv preprint arXiv:1805.07733*, 2018.

[15] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374*, 2019.

[16] N. R. Ke, A. Singh, A. Touati, A. Goyal, Y. Bengio, D. Parikh, and D. Batra. Modeling the long term future in model-based reinforcement learning. In *International Conference on Learning Representations*, 2018.

[17] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, May 2019. doi: 10.1109/ICRA.2019.8793742.

[18] W. Kim, J. Park, and Y. Sung. Communication in multi-agent reinforcement learning: Intention sharing. In *International Conference on Learning Representations*, 2020.

[19] F. Leibfried, N. Kushman, and K. Hofmann. A deep learning approach for joint video frame and reward prediction in atari games. In *5th International Conference on Learning Representations (ICLR 2017)*, pages 1–17, 2017.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[21] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[22] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. *Advances in Neural Information Processing Systems*, 28:2863–2871, 2015.

[23] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[24] S. Racanière, T. Weber, D. P. Reichert, L. Buesing, A. Guez, D. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5694–5705, 2017.

[25] A. Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

[26] R. Raileanu, E. Denton, A. Szlam, and R. Fergus. Modeling others using oneself in multi-agent reinforcement learning. In *International conference on machine learning*, pages 4257–4266. PMLR, 2018.

[27] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] S. Sukhbaatar, R. Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

[30] L. Sun, W. Zhan, and M. Tomizuka. Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2111–2117. IEEE, 2018.

[31] E. Talvitie. Model regularization for stable sample rollouts. In *UAI*, pages 780–789, 2014.

[32] E. Talvitie. Agnostic system identification for monte carlo planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2986–2992, 2015.

[33] S. Vandael, B. Claessens, D. Ernst, T. Holvoet, and G. Deconinck. Reinforcement learning of heuristic ev fleet charging in a day-ahead electricity market. *IEEE Transactions on Smart Grid*, 6(4):1795–1805, 2015.

[34] Y. Wen, Y. Yang, R. Luo, J. Wang, and W. Pan. Probabilistic recursive reasoning for multi-agent reinforcement learning. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[35] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

[36] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International Conference on Machine Learning*, pages 10607–10616. PMLR, 2020.

[37] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.