A Framework for Predictable Actor-Critic Control

Anonymous

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 (Haarnoja et al. 2018; Schulman et al. 2017; Mnih et al. 2015). Such algorithms can perform at, and even surpass, human-level control in various domains. Examples include robotics (Haarnoja et al. 2018; Dey et al. 2021), traffic management (Ault and Sharon 2021; Ault, Hanna, and Sharon 2020), autonomous driving (Kendall et al. 2019; Sallab et al. 2017), and energy management (Gao 2014; Vandael et al. 2015).

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 (Wen et al. 2019; Sun, Zhan, and Tomizuka 2018; Kim, Park, and Sung 2020; Das et al. 2019; Raileanu et al. 2018) and for regulating an RL-agent's performance in safety-critical tasks (Biondi et al. 2020), 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 (Racanière et al. 2017).

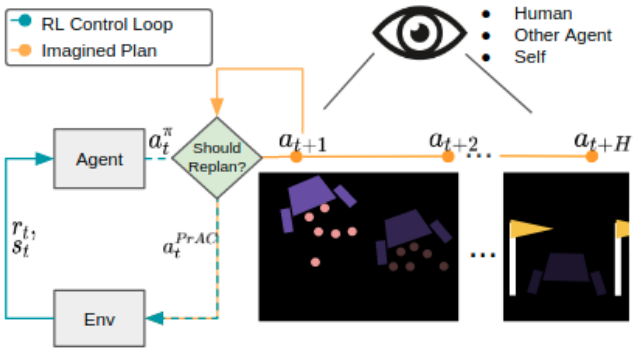

Figure 1: An overview of the PrAC framework. A predicted (imagined) plan is computed in order to provide predictable control. At each step the previously predicted plan is evaluated to determine whether replanning should be initiated at the expense of reduced predictability.

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* (Racanière et al. 2017). 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 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 LunaLander (Brockman et al. 2016), Walker, and Hop-

per (Coumans and Bai 2016–2021), 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 (Coumans and Bai 2016–2021), PrAC incurs substantial ($\sim$22%) performance degradation to produce an average of only 3 and 2 forecasted actions, respectively.

Preliminaries

This paper addresses control problems modeled as *Markov Decision Processes* (MDPs) (Puterman 2014), 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) (Haarnoja et al. 2018), 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 $V(s_0)$. An action-state (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 the optimal policy (π^*).

State-of-the-art RL algorithms (Haarnoja et al. 2018) use an actor-critic framework where an action-state value approximator (Q) is trained. Following the Policy-Gradient Theorem (Williams 2004), 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* (Kaiser et al. 2019; Xu et al. 2020) approach. Such a model can be used to produce imaginary trajectories (Racanière et al. 2017) 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 (Oh et al. 2015; Chiappa et al. 2017; Leibfried, Kushman, and Hofmann 2017), 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 (Talvitie 2014, 2015).

In order to reduce environment modeling errors, Ke et al. 2018 proposed building a latent-variable autoregressive model (Gulrajani et al. 2016) by leveraging recent ideas in variational inference (Zhang et al. 2018). Another recent approach for reducing modeling errors (Nagabandi et al. 2018) 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 (Foerster et al. 2016; Sukhbaatar, Fergus et al. 2016; Jiang and Lu 2018; Das et al. 2019).

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 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 an inaccurate, approximation 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) (Haarnoja et al. 2018). PrAC can be extended to other algorithms which learn and

store Q-values such as DQN (Mnih et al. 2015). 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, action-state 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 pre-planned (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 ϵ

1 Initialize: policy (π) parameters θ, action-value function (Q) parameters ψ, model approximation (P) parameters ϕ, empty replay buffer $\mathcal{D}$
2 **for** *each episode* **do**
3 reset environment: $s \leftarrow s_0$
4 $plan \leftarrow$REPLAN($s, plan :=$ empty array, $\pi_\theta, P_\phi, Q_\psi, \epsilon$)
5 **for** *each environment step* **do**
6 $a \leftarrow dequeue(plan)$
7 execute a and observe r, s'
8 store (s, a, r, s') in $\mathcal{D}$
9 $s \leftarrow s'$
10 $plan \leftarrow$REPLAN($s, plan, \pi_\theta, P_\phi, Q_\psi, \epsilon$) // *reuses as much of the old plan as possible*
11 **for** *each gradient step* **do**
12 $\psi = \psi - \lambda_Q \nabla_\psi [Q(s_t, a_t) - (r_t + \gamma(Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}, a_{t+1})))]$
13 $\theta = \theta - \lambda_\pi \nabla_\theta [\alpha \log \pi(a_t|s_t) - Q(s_t, a_t)]$
14 $\phi = \phi - \lambda_P \nabla_\phi [P(s_t, a_t) - s_{t+1}]^2$

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 sat-

Algorithm 2: Replan

Input : state s, current plan $plan$, policy π, action-value function Q, model approximation P, performance degradation tolerance ϵ

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 // critic says that following our last plan is ϵ-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 $plan'$

isfies:

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

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:

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

As a result, 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 Environments

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

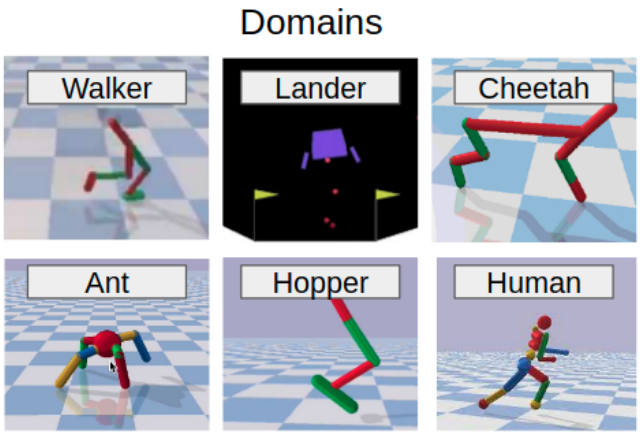

Figure 2: Snapshots from the domains used in the experimental section.

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 (Haarnoja et al. 2018). 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, for 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. For the PyBullet robotic domains, the action spaces are the amount of torque applied to each actuator. Humanoid is especially challenging because it has 17 actuators. Refer to Table 1 for the state and action space size for each environment we use.

Hyper-parameter settings

The reported PrAC implementation utilizes the open-source Stable Baselines (Hill et al. 2018) implementation of SAC.

Stable Baselines maintains a well-documented library of benchmark reinforcement-learning algorithms. The code for our experiments is available at [omitted for blind review].

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 (Raffin 2020). 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 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.

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).

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, $V^{PrAC}(s_0)$, represents the normalized performance of PrAC. In Lander, Hopper and Walker environments, we can see that the forecast decreases very little while providing a multistep action plan. Performance of other environments, namely Humanoid, Cheetah and Ant do slightly worse and experience some noticeable performance degradation. This is likely because they are harder environments to model (Table 1 shows the increased state space size for these environments, making an environment model harder to learn.) Nevertheless, all environments respect the theoretical bounds in expectation.

The hyper-parameter epsilon that we introduce 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 (Racanière et al. 2017) and (Kim, Park, and Sung 2020). We would expect a very small forecast with $\epsilon = 0$. On the other hand, a $\epsilon = \infty$ 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.

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, else 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: This is our 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 a learning curve for baseline Soft Actor-Critic.

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 additionally offers a 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 is too stubborn on keeping its old plan instead

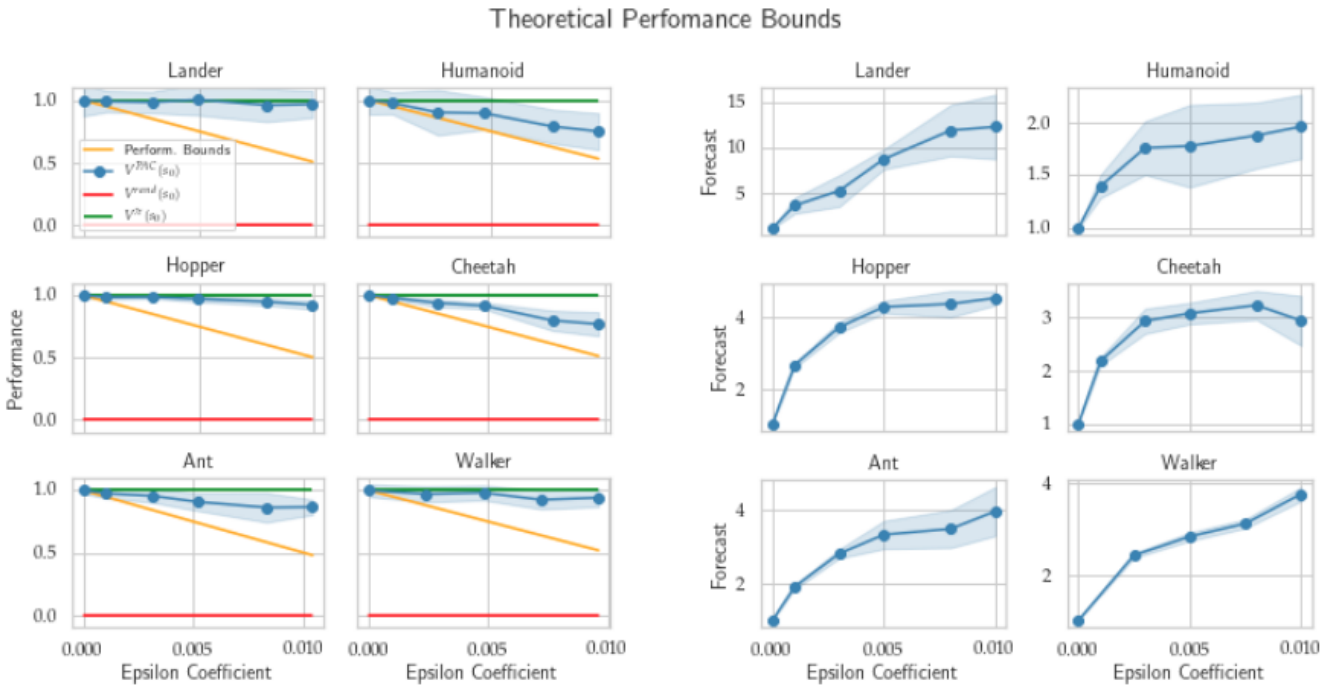

Figure 3: Left: normalized performance between random performance (red) and optimal performance (green) along with the theoretical performance bounds (yellow) and observed PrAC performance (blue) for different epsilon coefficient values. Right: average forecast as a function of the epsilon coefficient value. Shaded regions represent a 1 standard deviation over 20 runs per setting.

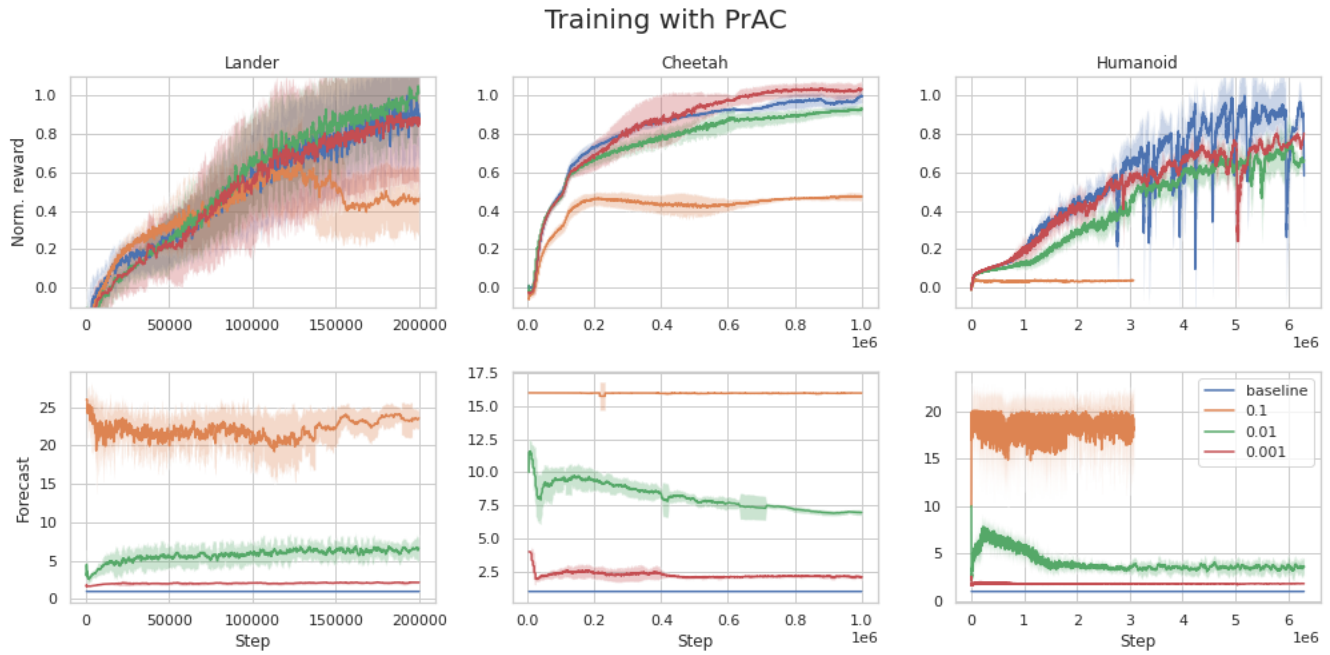

Figure 4: Top: training curves from representative environments. The y axis is the normalized episode reward. Bottom: The associated forecast for the environments. The baseline algorithm shows SAC without any modifications. Experiments run with 5 replications of each setting. Shown with 1 std. Mean and 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 after 3e6 steps as it fails to improve beyond random.

of replanning, so the episode reward never reaches 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 for this coefficient is impractical. In Cheetah, we can start to observe the trade-off at 0.01 and 0.001. For a small incurred reward penalty, we gain a forecast. The fact that the episode returns from PrAC(0.001) are slightly higher than the baseline for Cheetah is likely a random artifact.

With humanoid (the rightmost plot in Figure 4), we can see that a large epsilon coefficient totally 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 indeed learn 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 tradeoff occurs in both training-with-PrAC and training-then-PrAC. Next, we observe the effect of first training a baseline model to convergence and then applying PrAC on a static underlying policy. The results from these experiments are shown in the right-hand side of Figure 3 and in Figure 5. In Figure 3 and Figure 5, agents were trained to convergence without using PrAC's imagined plans. Only post-convergence were the imagined plans followed. At low values of ϵ, some domains see no loss in performance, while others experience very little. As ϵ increases, performance decreases across all environments. An opposite reaction applies to forecast lengths. As ϵ increases, forecasts improve. Across environments, our empirical findings show that an imagined plan can predict, on average, 5 steps into the future at a cost of a 25% performance degradation.

The forecast distributions shown in Figure 5 give us greater insight into the type of reliability we can expect out of a PrAC agent. We see that the forecast distributions are generally right skewed, although for four of the environments, the distribution is not right skewed on PrAC(0.1). We speculate that this is because these PrAC(0.1) has chosen to be completely stubborn in this case and not change plans for the sacrifice of totally degraded performance. Such a historical forecast distribution as presented here could help a supervising agent interpret the plans being generated by the PrAC agent to know how confident it can be in the plans generated.

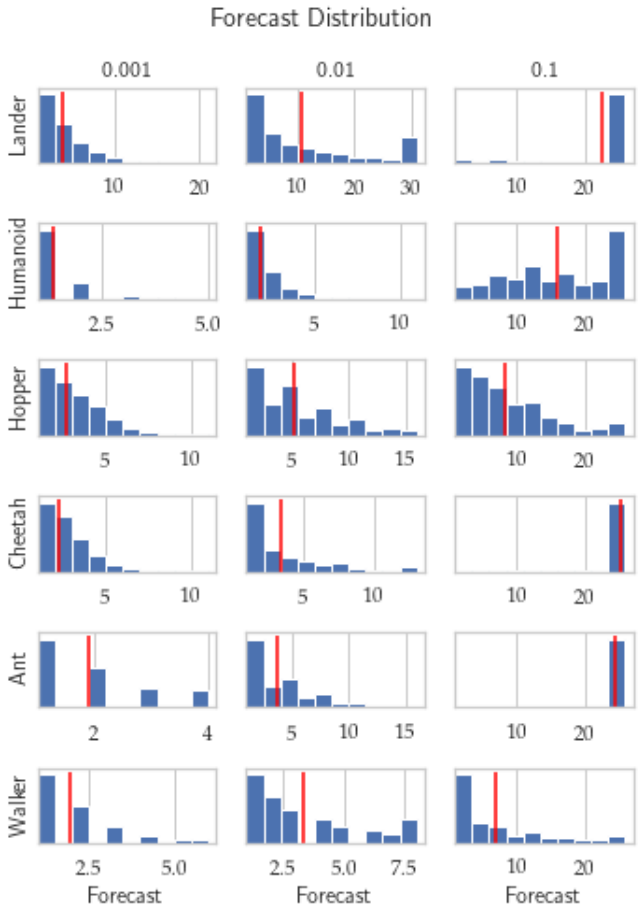

Figure 5: A histogram of the forecast length when applying PrAC after training the underlying policy. The mean forecast is shown with a red bar. The columns are separated by epsilon coefficient and the rows are separated by environment. The x axis is the forecast and the y axis is the normalized count. We run 10 replications for each plot. Each environment has had the 0.1 forecast capped to 25 for computational considerations.

5 Summary

Our research examines augmentations to RL algorithms that provide robust long-horizon planning. Doing so is of high importance in domains that require human supervision and in multiagent settings that require communicating behavior intentions. As a result, this research is expected to have a substantial impact on state-of-the-art deep RL technology for autonomous driving and similar safety critical / multiagent domains.

In this work, we have proposed a solution to give our intention (forecast) of what we plan to do in the future. Furthermore, we propose an easily adopted method to allow for reliability in this plan via action-value evaluations of our plan. We have proven a theoretical lower bound for the performance degradation for the PrAC policy. We have also shown empirically that PrAC respects these bounds in the expectation even under biases that result from model approximation. Finally, we have shown empirically the tradeoff between plan consistency and performance degradation. Reinforcement learning has great promise for real world applications but many of these applications are in safety critical and multiagent settings. By proposing a system for generating reliable future intention, we open up the door to applications where predictability is of high importance.

References

Ault, J.; Hanna, J.; and Sharon, G. 2020. 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.

Ault, J.; and Sharon, G. 2021. Reinforcement Learning Benchmarks for Traffic Signal Control. In *Proceedings of the 35th Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*.

Biondi, A.; Nesti, F.; Cicero, G.; Casini, D.; and Buttazzo, G. 2020. A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems. *IEEE Embedded Systems Letters*, 12(3): 78–82.

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

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

Coumans, E.; and Bai, Y. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

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

Dey, S.; Pendurkar, S.; Sharon, G.; and Hanna, J. 2021. A Joint Imitation-Reinforcement Learning Framework for Reduced Baseline Regret. In *Proceedings of the 34th International Conference on Intelligent Robots and Systems (IROS 2021)*.

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

Gao, J. 2014. Machine learning applications for data center optimization.

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

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

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

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

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

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

Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.; Lam, V.; Bewley, A.; and Shah, A. 2019. Learning to Drive in a Day. In *2019 International Conference on Robotics and Automation (ICRA)*, 8248–8254.

Kim, W.; Park, J.; and Sung, Y. 2020. Communication in Multi-Agent Reinforcement Learning: Intention Sharing. In *International Conference on Learning Representations*.

Leibfried, F.; Kushman, N.; and Hofmann, K. 2017. A Deep Learning Approach for Joint Video Frame and Reward Prediction in Atari Games. In *5th International Conference on Learning Representations (ICLR 2017)*, 1–17.

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

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

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-Conditional Video Prediction using Deep Networks in Atari Games. *Advances in Neural Information Processing Systems*, 28: 2863–2871.

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

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

Raffin, A. 2020. RL Baselines3 Zoo. https://github.com/DLR-RM/rl-baselines3-zoo.

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

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

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

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

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

Talvitie, E. 2014. Model Regularization for Stable Sample Rollouts. In *UAI*, 780–789.

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

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

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

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

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

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