POSTERIOR SAMPLING: MAKE REINFORCEMENT LEARNING SAMPLE EFFICIENT AGAIN

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning thrives on leveraging structure in data, and many breakthroughs (e.g. convolutional networks) have been made by designing algorithms which exploit the underlying structure of a distribution. Reinforcement Learning agents interact with worlds that are similarly full of structure. For example, no sequence of actions an agent takes will ever cause the laws of physics to change, therefore an agent which learns to generalize such laws through time and space will have an advantage. Sample efficient reinforcement learning can be accomplished when assuming that the world has structure and designing learning algorithms which exploit this assumption without knowing the actual structure beforehand. Posterior Sampling for Reinforcement Learning (PSRL) (Strens, 2000) is such a method which assumes structure in the world and exploits it for learning. A PSLR learning agent first samples models of the environment which conform to both prior assumptions on the world's structure and past observations and then interacts with the true environment using a policy guided by the sampled model of the environment. While PSRL delivers theoretical Bayesian regret bounds, there are many open issues which must be addressed before PSRL can be applied to current benchmark continuous reinforcement reinforcement tasks. In this work, we identify these issues and find practical solutions to them leading to a novel algorithm we call Neural-PSRL[1]. We validate the algorithm's effectiveness by achieving state-of-the-art results in the HalfCheetah-v3 and Hopper-v3 domains.

1 INTRODUCTION

In Reinforcement Learning (RL) an agent interacts with an environment by taking a sequence of actions, making observations and collecting rewards along the way. The goal in classic Reinforcement Learning is to find an agent that is able to maximize its accumulated rewards over time. This task is challenging since the agent is interacting with an unknown environment, and the agent must balance efficiently exploring the environment while collecting rewards, a trade-off known as the exploration exploitation trade-off.

Strategies for solving this problem fall into two broad classes: model-based RL and model-free RL. In the model-based case, a model of the true environment is learned, and actions are selected based on this model. Some examples of model-based methods include Dyna (Sutton, 1990), Monte Carlo Tree Search (MCTS) and PILCO (Deisenroth & Rasmussen, 2011). In the model-free case, no model of the environment is learned, instead generally a value function or Q-function is learned directly from interactions with the environment. Some examples of model free include Q-Learning (Watkins & Dayan, 1992), DQN (Mnih et al., 2015), Actor Critic Methods (Mnih et al., 2016) and Soft Actor Critic (Haarnoja et al., 2018).

Model-free Reinforcement Learning has delivered impressive results on many tasks, including tasks with high dimensional state spaces. Nevertheless, such algorithms are not sample efficient, requiring millions of observations to achieve human level performance (Hessel et al., 2018; Jin et al., 2018).

Viewed from a certain light, the millions of observations required shouldn't be a surprise since if the agent interacts with a discrete Markov Decision Process (MDP), the expected regret (i.e. the difference between the reward under an agent learned by a RL algorithm and the reward under an

[1] See supplementary material for videos and code which will be published

optimal agent) of an arbitrary unknown environment is lower bounded by $O(\sqrt{SAT})$ (Osband & Van Roy, 2016b; Jaksch et al., 2010), where S is the number of possible states, A is the number of possible actions and T is the number of interactions with the environment. Therefore, as the number of possible states and number of possible actions grow, so does the regret. Transferring this to the continuous state space case would imply an exponential growth in regret as the dimensionality of the state and action spaces increases. Making weak assumptions on the problem's structure could improve this bound vis-à-vis the *tabula rasa* setting, i.e. settings where only information about the state and action spaces are known. Examples of such weak assumptions include assuming that transition and reward dynamics can be modeled with a parameterizable model (Osband & Van Roy, 2014a) or that the conditional reward and transition probabilities factor (Osband & Van Roy, 2014b). Since agents generally interact with worlds that have structure (for example, the same physical laws apply everywhere) such structure assumptions are justified.

A framework to integrate such assumptions comes from Osband & Van Roy (2014a), where they assume the true environment is a sample from a know prior over all possible environments. This prior can then be chosen so that all environments sampled have structure, even if the underlying structure isn't yet known. For example, if we want to sample environments with linear structure, we can assume there is a linear mapping from the previous state-action pair to the next state and sample linear weight matrices. By assuming knowledge of such a prior, one can assume that the true environment is likely to have desirable properties. We note that restricting the possible environments to reduce regret is not a new strategy, for example the Linear Quadratic Regulator Bemporad et al. (2002) is able to find an optimal policy when assuming linear dynamics. Imposing a prior over all possible environments is an extremely general and adaptable extension of this idea.

By assuming that the true environment is a sample from a prior over all environments, we can use posterior sampling for reinforcement learning (PSRL), a variant of Thompson Sampling (Thompson, 1933) for Markov Decision Processes, to find a policy. It's known that the expected regret of the PSRL learning rule (Strens, 2000; Osband et al., 2013) is upper bounded by $\tilde{O}(\sqrt{d_K d_E T})$ (Osband & Van Roy, 2014a). Here d_K is the Kolmogorov dimension and d_E is the Eluder dimension (Russo & Van Roy, 2013) of the space of possible environments that can be sampled from the prior. In this way, the regret is now bounded by the complexity of the prior we assume the true environment to be sampled from, and not the dimensionality of the state and action spaces. A corollary is then if we know the true environment to be highly structured then the regret can be bounded by choosing a prior which accurately reflects the structure.

Equivalent to Thompson Sampling, PSRL happens in three alternating steps. First, an environment is sampled from a posterior constructed from the prior and all previous environment observations. Next, the optimal policy for this sampled environment is found. Since the sampled environment is fully known, this step can be done with computing power alone and does not require environment interactions. Finally, the policy found in the last step interacts with the true environment for an episode, and the resulting observations are saved.

While this method seems very appealing from a theoretical perspective, to the best of our knowledge it hasn't been applied to benchmark continuous reinforcement learning tasks. This could very well be because two of the three PSRL steps are themselves very challenging machine learning problems. In order to sample environments from a posterior, one must come up with a generative model of environments, and generative modeling is very much a very active research area in Machine Learning. Further, finding a policy which is optimal with respect to a sampled MDP is computationally complex (Sidford et al., 2018), and this computational overhead must also be addressed in order to apply PSRL to challenging reinforcement learning tasks.

In this paper, we address both challenges and propose methods for learning a generative model of MDPs, and for finding an optimal policy for an MDPs sampled from this generative model. We refer to this suite of solutions as Neural-PSRL. Our method is then applied to the benchmark HalfCheetah-v3 and Hopper-v3 MuJoCo continuous control tasks (Todorov et al., 2012) and match the performance of current sample-efficient methods in both domains. Our work demonstrates that PSRL is more than just a theoretically appealing method, but is a framework which can leverage assumptions on a problem's structure to achieve state-of-the-art results.

2 NOTATION, DEFINITIONS AND ASSUMPTIONS

In the following section, we largely use the notation from Osband et al. (2013). For simplicity we assume interactions with the environment occur in episodes of fixed length $\tau \in \mathbb{N}$. Environments where an episode can end early, e.g. Hopper-v3 (Erez et al., 2012), can easily be adapted to the fixed length setting by adding an absorbing state. Such an environment is formalized by a Markov Decision Processes $M = (\mathcal{S}, \mathcal{A}, R^M, P^M, \tau, \rho)$ where $\mathcal{S}$ and $\mathcal{A}$ are vector-valued state and action spaces, $R^M(\cdot \mid s, a)$ is a reward distribution over the real numbers $\mathbb{R}$, and $P^M(\cdot \mid s, a)$ is the transition probability over $\mathcal{S}$ conditioned on state s and action a. Finally, $\tau \in \mathbb{N}$ is the episode length and ρ is an initial state distribution over $\mathcal{S}$.

A policy $\mu(\cdot \mid s, i)$ is a probability measure over the action space conditioned on a state $s \in \mathcal{S}$ and the current step in the episode $i \in \{1, \dots, \tau\}$, meaning when the environment is in state s and is about to take the i-th action in this episode, the policy determines with what probability an action $a \in \mathcal{A}$ is then taken.

Given a policy μ and an MDP $M = (\mathcal{S}, \mathcal{A}, R^M, P^M, \tau, \rho)$, we can recursively define the value function of policy μ at time $i \in \{1, \dots, \tau\}$ starting with $V^M_{\mu,\tau+1}(s) := 0$ and working backwards from $i = \tau$ to $i = 1$ via

$$V^M_{\mu,i}(s) := \mathbb{E}_{a \sim \mu(\cdot \mid s,i)} \Big[R^M(s,a) + \mathbb{E}_{s' \sim P^M(\cdot \mid s,a)} [V^M_{\mu,i+1}(s')] \Big].$$

Intuitively, the value function is the expected future reward under policy μ; at the end of an episode no more reward will come, thus $V^M_{\mu,\tau}(s) := 0$, and for every previous step the expected future value is the reward at the current step plus all future rewards.

A policy μ is said to be the optimal policy for MDP M if $V^M_{\mu,1}(s) = \max_{\mu'} V^M_{\mu',1}(s)$ for all $s \in \mathcal{S}$. Generally the optimal policy need not be unique; for the sake of simplicity we will not consider the whole set of optimal policies but use μ^M to denote some arbitrary policy from the set of policies which are optimal for M.

The regret of a policy μ under a MDP M is the difference between the reward that will on average be collected by policy μ vs. the optimal policy μ^M. Formally, the regret is defined as

$$\Delta^M_\mu := \mathbb{E}_{s \sim \rho}[V^M_{\mu^M,1}(s) - V^M_{\mu,1}(s)].$$

Lower regret policies are generally desirable, since such policies collect on average more rewards.

Interaction with the MDP M happens in episodes of length τ, and at the beginning of the k-th episode, we wish to use past observations to find a policy μ which will collect both rewards and new knowledge about the environment. A sequence of conditional distributions over policies $(\pi_k)_{k \in \mathbb{N}} = \pi_0, \pi_1, \dots$ where each π_k is conditioned on all past observations from the past k episodes is called a learning rule or learning agent. We denote with H_k all observations from the first k episodes. Although other evaluation criteria are possible, We evaluate a learning rule on an MDP M by it's sum of expected regrets

$$\text{Regret}(T, \pi, M) := \sum_{k=1}^{\lfloor T/\tau \rfloor} \mathbb{E}_{\mu \sim \pi_k(H_k)}[\Delta^M_\mu] \tag{1}$$

The rest of this paper will be devoted to motivating, developing and evaluating such a learning rule.

3 NEEDLE IN A d-DIMENSIONAL HAYSTACK

The fundamental problem with reinforcement learning in higher dimensions without any prior assumptions can be illustrated with the following extremely simple example adapted from Russo & Van Roy (2013) for the continuous case. Figure 1 is a visual representation of the example for the $d = 2$ case. Assume we have a family of deterministic, finite time horizon MDPs

$$(M_\vartheta)_{\vartheta \in [0,1]^d} = \{(\mathcal{S}, \mathcal{A}, P, R_\vartheta, \tau, \rho) \mid \vartheta \in [0,1]^d\}$$

where $\mathcal{S} = \{s_0\}$, $\mathcal{A} = [0,1]^d$, $\rho(\{s_0\}) = 1$, $\tau = 1$ (so the episode only has length 1). Further assume $R_\vartheta(s,a) = \max(1 - L\|a - \vartheta\|_2), 0)$. In this simple example, an agent takes an action by exploring a point in the d-dimensional hypercube. If the distance between the point explored and ϑ is smaller than $1/L$, a strict positive reward is collected. If not, a reward of zero is collected and the agent doesn't learn much useful information about the environment.

Even if an agent knows everything about this environment except for ϑ, the agent's best strategy will be some variant of grid search, which will take $O(L^d)$ episodes before a non-zero reward is found. Regret in the sense of Eq. 1 for any agent will therefore grow exponentially with d and polynomially with L, and higher dimensional problems will require billions of episodes for a reasonably good policy to be found.

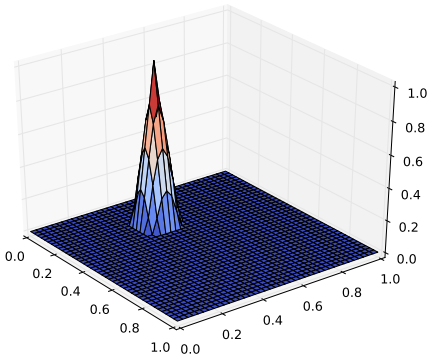

Figure 1: Reward function $R_\vartheta(s_0, \cdot)$. Any agent will struggle to learn the relevant information of this function (the location of the point) Samples from other areas yield little relevant information.

Thankfully, the world we live in (and consequently most reinforcement learning tasks we wish to solve) has a more generalizable structure than the needle in a haystack. For example, the same laws of physics apply everywhere, independent of time and space. Therefore physical laws learned in one area of the state space can be applied to other parts of the state space.

We can therefore safely make many prior assumptions about our environment (an example for such an assumption is that the world is generalizable) before the agent executes even one interaction. Nearly all successful deep-learning RL algorithms implicitly make such prior assumption by assuming that true transition dynamics (in the model-based case) and Q-and value-functions (in the model-free case) can be modeled with a neural network of limited capacity.

3.1 Accurate prior assumptions bound regret

Now that we have established that no prior assumptions results in (almost) unbounded regret, one asks oneself if the converse is true: can one use prior assumptions to derive regret bounds in the continuous state and action space MDP setting? In Russo & Van Roy (2013), such bounds can be derived for the infinite-arm bandit setting, and the proof concept is expanded upon in Osband & Van Roy (2014a) to bound the Bayesian regret for finite time horizon continuous Markov Decision Processes.

Theorem 1 (Regret for PSRL in parameterized MDPs, from Osband & Van Roy (2014a)) *Fix real state and action spaces $\mathcal{S}$ and $\mathcal{A}$ and two function families $\mathcal{R}$ and $\mathcal{P}$ of bounded mappings from $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Let M^* be an MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, reward distribution equal to $R^* + \varepsilon_\mathcal{R}$ where $R^* \in \mathcal{R}$ and $\varepsilon_\mathcal{R}$ a sub-Gaussian random variable with variance $\sigma_\mathcal{R}$ and similarly for transition probability equal to $P^* + \varepsilon_\mathcal{P}$ with $P^* \in \mathcal{P}$. If ϕ is the distribution of M^* and $K^* = K^{M^*}$ is a global Lipschitz constant for the future-value function, then*

$$\mathbb{E}[\text{Regret}(T, \pi^{PS}, M^*)] = \tilde{\mathcal{O}}(\sigma_\mathcal{R} \sqrt{d_K(\mathcal{R}) d_E(\mathcal{R}) T} + \mathbb{E}[K^*] \sigma_\mathcal{P} \sqrt{d_K(\mathcal{P}) d_E(\mathcal{P}) T}) \tag{2}$$

where $\tilde{\mathcal{O}}$ ignores terms logarithmic in T and d_K and d_E are the Kolmogorov and Eluder dimensions.

The main takeaway from this theorem is that the expected regret of the PSRL learning rule π^{PS} (defined in subsection 4.1) is bounded by the complexity of the classes of functions from which reward and transition dynamics are drawn. This result shouldn't be a surprise, if we know that the reward and transition distributions are drawn from a simple class of functions (e.g. linear functions) then the problem will be easier than if for example we only know that the environment dynamics are merely Lipschitz continuous. Work on analogous bounds for the infinite-time horizon MDP case is inconclusive, Abbasi-Yadkori & Szepesvári (2015) claims to have found such bounds, but Osband & Van Roy (2016a) points out a fault in their proof.

4 Method

Although the PSRL method is well understood in theory, the method assumes one can sample MDPs from a posterior distribution and find an optimal policy for said sampled MDPs, both non-trivial problems. Our contribution is to demonstrate Neural-PSRL, a suite of methods which address these

challenges in the continuous setting, thus opening a novel path to solve continuous reinforcement problems. The effectiveness of this approach is experimentally demonstrated in section 6

4.1 Posterior Sampling for Reinforcement Learning

The learning rule which achieves the desirable regret bounds in Eq. 2 is the Posterior Sampling for Reinforcement Learning rule π^{PS}, an extension of Thompson Sampling to Markov Decision Processes first proposed by Strens (2000). The learning rule assumes access to a prior over MDPs ϕ, and that the true MDP M^* is a sample from said prior. It is exactly this prior into which the practitioner can integrate prior knowledge about the environment. For example, by choosing a prior which when sampled returns for a transition dynamics model a neural network of limited capacity, the practitioner assumes the true transition dynamics are simple enough to be modeled by a neural network of limited capacity. Clearly, if the practitioner selects a prior from which the true MDP M^* cannot be sampled, the PSRL learning rule π^{PS} is not guaranteed to find a good policy.

Algorithm 1 Posterior Sampling for Reinforcement Learning (PSRL)

INPUT: Prior distribution ϕ for M^* , $t = 1$, $H_t = \{\}$
for episodes $k = 1, 2, \ldots$ **do**
  sample $M_k \sim \phi(\cdot \mid H_t)$
  compute $\mu_k = \mu^{M_k}$
  for timestep $j = 1, \ldots, \tau$ **do**
    perform action $a_t \sim \mu_k(s_t)$
    observe r_t and s_{t+1}
    $H_{t+1} \leftarrow H_t \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    $t \leftarrow t + 1$
  end for
end for

The PSRL algorithm is outlined in Algorithm 1. At its heart, it is analogous to Thompson sampling in that given past observations H_t, an environment M_k is sampled which fits both the prior assumptions and past observations H_t. In the next step a policy μ_k which is optimal for M_k is found and executed, and the observations are saved to the history. Since the MDP setting is more expressive than the simple bandit setting in which Thompson Sampling was developed, both these steps become more challenging. On the one hand, reward and transition dynamics are more complex than returns from a stateless multi-armed bandit which necessitates a more complex prior ϕ from which one can sample MDPs conditioned on past observations. This is nothing more than a generative model from which one can sample different transition and reward dynamics. In addition, while the optimal strategy given a known bandit is obvious (simply pull the arm that gives the highest expected return), discovering the optimal policy given a known MDP is often non-trivial (Azar et al., 2013). In the following two subsections, we will propose solutions to both these problems.

4.2 Generative Modeling of Posterior $\phi(\cdot \mid H_t)$

Constructing models from which complex distributions can be sampled is an active area of research, with many proposed methods including Variational Auto-Encoders (Kingma & Welling, 2014), Restricted Boltzmann Machines (Hinton, 2002) and Generative Adversarial Networks (Goodfellow et al., 2014). Generative modeling has already found applications in reinforcement learning (Deisenroth & Rasmussen, 2011; Ha & Schmidhuber, 2018; Chua et al., 2018).

Fortunately, in a meta-study of deep Baysian networks for Thompson sampling (Riquelme et al., 2018), it was shown that a relatively simple Bayesian linear regression model similar to Snoek et al. (2015) generally performs well on a broad range of reinforcement learning tasks. In that model, a neural network is trained to predict the next state (or reward), and the last layer of the network is then replaced with a Bayesian linear regression model; by sampling a new weight matrix from the Bayesian linear regression model one samples a new transition (or reward) model.

We use this model as a basis for our model, in which the predicted next state $\hat{s}_{t+1}$ is given by $\hat{s}_{t+1} := \mathbf{W} z_{(s_t, a_t)} + s_t$ where $z_{(s_t, a_t)}$ is the output of the last hidden layer of a network trained to predict $s_{t+1} - s_t$ given the previous state action pair (s_t, a_t) as input. Further, $\mathbf{W}$ is a weight

matrix sampled via Bayesian linear regression to regress $z_{(s_t,a_t)}$ to $s_{t+1} - s_t$ for all state-action-observation sets in our history. We predict not the next state but the difference to the previous state, as suggested in Deisenroth et al. (2013). Thus, dynamics models P^{M_k} (and by extension whole MDPs M_k) are sampled conditioned on past observations H_k by sampling only the last layer of a next-state prediction network. Note that in contrast to Chua et al. (2018), we sample deterministic next-state predictions, which allows us learn a policy with gradient based methods without the need to sample multiple trajectories (see below).

In a final fine-tuning step, we compose multiple copies of the sampled network from above to predict the n-next states and then backpropagate the errors through the composed-together network. This allows the network to not just predict the next state, but accurately predict the next n-states despite errors in next-state predictions. To both increase the generalization ability of the network, and integrate assumptions about the structure of the true environment, we use weight decay with the AdamW optimization algorithm.

4.3 Policy Finding

Given a known MDP M_k, the expected returns of a proposed policy μ can be evaluated via brute force Monte Carlo sampling, so in theory finding a (near) optimal policy is possible via trial and error. Unfortunately, in Azar et al. (2013) it's shown that even in the setting described in Kakade et al. (2003) with an oracle which returns a sample of the next observation conditioned on arbitrary state action pairs, finding an ϵ-optimal policy with probability $1 - \delta$ has a complexity of

$$\mathcal{O}\left[\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^3\epsilon^2}\right]$$

We are not aware of any analogous bounds for the continuous state-action space setting, but already this result would indicate that finding μ^{M_k} entails a high computational complexity. See Sidford et al. (2018) for an analogous bound for the finite-horizon case.

Fortunately, we have access not just to samples from the transition function but also to the gradients thereof, suggesting we should make use of an established policy gradient method (Silver et al., 2014; Sutton & Barto, 2018), offering a practical method to deal with large continuous actions spaces. One such method that has a proven track record is Soft Actor Critic (Haarnoja et al., 2018).

Soft Actor Critic learns a stochastic policy μ where actions sampled from the policy achieve a high reward with respect to the value function (here called the critic). Further, the entropy of sampled actions is enforced to match a pre-defined target, thereby ensuring the policy's actions are a bit random and explore nicely. SAC already brings a large bag of tricks to finding an optimal policy. For example the update uses a target value network $V_{\theta'}$ where θ' is the exponential moving average of the value function weights θ (Mnih et al., 2015). Second, they two value functions to minimize the positive bias that degrades performance of value based methods (Hasselt, 2010; Fujimoto et al., 2018). These two value functions are neural networks parameterized with parameters θ_1 and θ_2 and trained independently with standard next-step predictions provided by M_k (in contrast to Mnih et al. (2015) which uses a replay buffer) and Bellmann updates. To update the policy, the minimum of the two value functions is used.

We modify SAC in a few key points. First, assume that executing n steps of the policy μ starting in state $s_0 \in \mathcal{S}$ leads to state $s_1, \ldots, s_n \in \mathcal{S}$. With TD(0) updates (i.e. $V_\mu(s_i) \leftarrow (1-\lambda)V_\mu(s_i) + \lambda(r_i + \gamma V_\mu(s_{i+1}))$) it will take exponentially long in n for values from $V_\mu(s_n)$ update $V_\mu(s_0)$ (Arjona-Medina et al., 2019). We therefore elect to predict the next T states and update according to $V_\mu(s_i) \leftarrow (1-\lambda)V_\mu(s_0) + \lambda(\sum_{t=0}^{T-1} \gamma^t r_t + \gamma^T V_\mu(s_{i+1}))$

This is a multi player game (the policy, value function and transition + reward dynamics interact together) where the gradients from the value function and environment dynamics are used to update the policy. Such a setting is strongly reminiscent of Generative Adversarial Networks (Goodfellow et al., 2014). In this context it has been shown that low-variance gradient estimates are conducive for the multiple players to quickly reach a steady equilibrium (Seward et al., 2018). As networks grow deeper, errors in gradient estimation compound. Therefore, although we use a large rollout to update the value function, we elect to obtain policy gradient with a short, i.e. one-step rollout.

Policy gradient methods for continuous control problems can converge to locally optimal policies (Tessler et al., 2019). We observed this to be an issue in our experiments, with different runs under

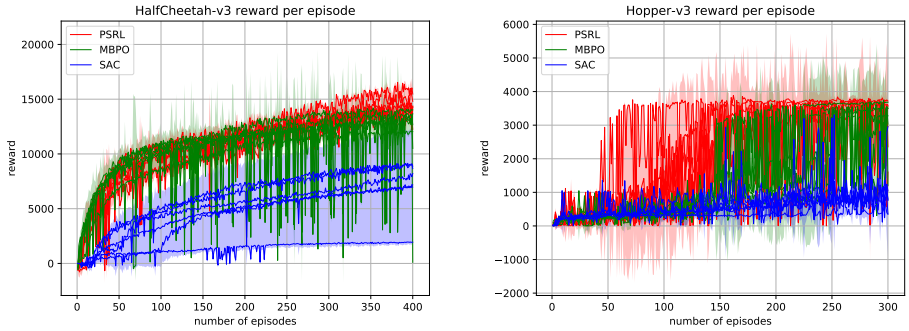

Figure 2: Training curves for Neural-PSRL and both model-free SAC (Haarnoja et al., 2018) and model-based (Janner et al., 2019) baselines on continuous control benchmarks Hopper-v3 and HalfCheetah-v3 Each environments is evaluated with the canonical 1000-step variant of the task. Thin curves indicate individual training curves, thick curves indicate the mean of the training runs and the shaded area is the standard deviation among trials. We see that in such a sample-constrained environment, the model based methods outperform the model-free SAC. Our method matches the performance of the current state-of-the-art model based method.

the same conditions converging to very different policies with very different average episode returns. This is a well documented issue in reinforcement learning (Henderson et al., 2018). We found using an ensemble of policies was a helpful heuristic to tackle this issue. There, multiple policies are initialized, and the value function is updated by performing separate rollouts with each policy and as a reward estimate the maximum of the reward estimates from all rollouts.

With this, we were able to reliably and find policies μ^{M_k} for sampled MDPs μ_k. Our method is summarized in Algorithm 2 in the Appendix.

5 Related Work

At this point, we'd like to showcase two similar methods which both solve continuous reinforcement learning tasks by learning models of the true environment and interacting with said environment guided by the learned world model. In the first method by Chua et al. (2018) called Probabilistic Ensembles with Trajectory Sampling (PETS) they first train an ensemble of models of the world using all past observations, then alternate between sampling action sequences $a_{t:t+T}$ from a cross-entropy method (Botev et al., 2013), predicting trajectories given action sequence $a_{t:t+T}$ and updating the CEM distribution from which action sequences are sampled. Once a satisfactory action sequence has been found, the first action in the sequence is performed.

An open question from Chua et al. (2018) is how long trajectories should be. An overly long trajectories will be inaccurate, while an overly short trajectories won't capture long-term consequences of an action. In Janner et al. (2019) this issue is addressed and a method for optimal Trajectory length is developed. Like in (Chua et al., 2018) they learn an ensemble of probabilistic neural networks, and sample trajectories from these probabilistic neural networks. The difference is that the actions for the trajectory samples are from a policy, and the policy is learned with SAC (Haarnoja et al., 2018) applied to the sampled trajectories.

6 Experiments

The goal of our experimental evaluation is to demonstrate the applicability of Neural-PSRL to sample efficient learning in benchmark reinforcement learning tasks. We evaluate Neural-PSRL on the popular and widely studied MuJoCo continuous control tasks (Todorov et al., 2012) of HalfCheetah-v3 and Hopper-v3 (Erez et al., 2012), comparing our method to Model Based Policy Optimization (MBPO) (Janner et al., 2019). To the best of our knowledge, MBPO is the current state-of-the-art

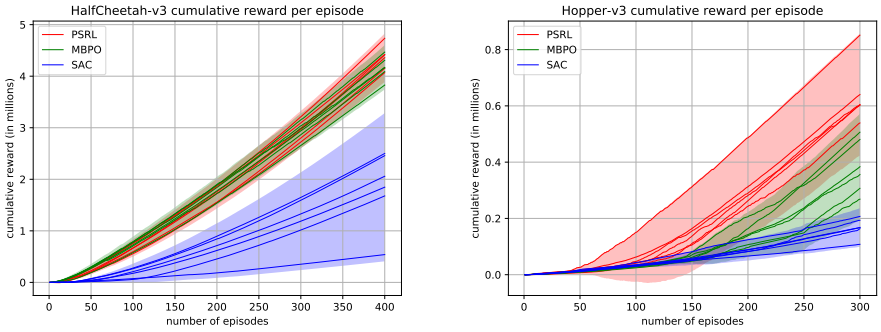

Figure 3: Cumulative training curves of runs from figure 2. To maximize cumulative reward (or equivalently minimize regret), learning must quickly learn a good policy and constantly improve it.

Table 1: Mean and 95% confidence bounds of training reward for DDPG Lillicrap et al. (2016), ACKTR Wu et al. (2017), TRPO Schulman et al. (2015), PPO Schulman et al. (2017), SAC Haarnoja et al. (2018), MBPO Janner et al. (2019) and Neural-PSRL when trained to convergence. Numbers for first four methods are from Henderson et al. (2018), see Appendix section A.2 for details.

	DDPG	ACKTR	TRPO	PPO	SAC	MBPO	Neural-PSRL
HalfCheetah	5037 (3664, 6574)	3888 (2288, 5131)	1254 (999, 1464)	3043 (1920, 4165)	11820 (2636, 21005)	12543 (11299, 13787)	13885 (11171, 16599)
Hopper	1632 (607, 2370)	2546 (1875, 3217)	2965 (2854, 3076)	2715 (2589, 2847)	3030 (2009, 4050)	3123 (2301, 3944)	3347 (2883, 3811)

on these tasks when the number of environment evaluations allowed is restricted ($\leq 500K$ steps). Since MBPO clearly outperforms by a wide margin other reinforcement learning methods such as PPO (Schulman et al., 2017) and PETS (Chua et al., 2018), we compare only to the model-based MBPO and the model-free SAC. Like in MBPO, we use the canonical 1000-step horizon with early termination versions of both tasks, and assume knowledge of the termination criteria, and for the sake of simplicity also assume knowledge of the reward distribution instead of learning it.

Since the goal set out Eq. 1 was to minimize the sum of regrets over all episodes, we report a closely related metric: the cumulative reward over episodes since the cumulative reward after n episodes is n times the expected reward of the unknown optimal policy minus the cumulative regret. In contrast to many experimental evaluations, we report the reward with respect to the number of episodes, not the number of environment interactions. When reporting cumulative reward with respect to the number of environment interactions, it is impossible to distinguish between undesirable policies that achieve a high per-interaction reward but often terminate early and more desirable policies which achieve similarly high per-interaction reward and rarely terminate early. We report the reward received during learning, not the reward that would have been received in an offline evaluation of the policy.

Since PSRL can be split into separate tasks (building a model of the world, sampling from the model with an eye to optimal exploration-exploitation trade-off, finding an optimal policy for sampled models) we leave the questions of optimizing the individual tasks to specific domains to future work, and restrict ourselves to demonstrating that Neural-PSRL works.

7 Next Steps

Neural-PSRL shows that the highly-adaptable PSRL framework is an important tool in the Reinforcement Learner's toolbox, and indeed our experiments confirm that PSRL can match state-of-the art performance with a relatively small number of samples. Still, our experiments are not yet a fair head-to-head comparison with other methods. For example, we employ the the ensemble of policies trick which could also improve other off-policy policy-gradient based methods such as MBPO, SAC, and PPO. In addition, the effect of an expanded Neural-PSRL which also learns the reward dynamics has not yet been explored and is something we would like to investigate.

REFERENCES

Yasin Abbasi-Yadkori and Csaba Szepesvári. Bayesian optimal control of smoothly parameterized systems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing*, 2019.

Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J Kappen. Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3): 325–349, 2013.

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre LEcuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pp. 35–59. Elsevier, 2013.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and systems VII*, pp. 73, 2012.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pp. 2450–2462. 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.

Hado V. Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems 23*, pp. 2613–2621. 2010.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems 31*, 2019.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems 31*, pp. 4863–4873. 2018.

Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Ian Osband and Benjamin Van Roy. Model-based reinforcement learning and the eluder dimension. In *Advances in Neural Information Processing Systems 27*, pp. 1466–1474, 2014a.

Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored mdps. In *Advances in Neural Information Processing Systems*, pp. 604–612, 2014b.

Ian Osband and Benjamin Van Roy. Posterior sampling for reinforcement learning without episodes. *arXiv preprint arXiv:1608.02731*, 2016a.

Ian Osband and Benjamin Van Roy. On lower bounds for regret in reinforcement learning. *arXiv preprint arXiv:1608.02732*, 2016b.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pp. 3003–3011, 2013.

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *International Conference on Learning Representations*, 2018.

Daniel Russo and Benjamin Van Roy. Eluder dimension and the sample complexity of optimistic exploration. In *Advances in Neural Information Processing Systems 26*, pp. 2256–2264. 2013.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Calvin Seward, Thomas Unterthiner, Urs Bergmann, Nikolay Jetchev, and Sepp Hochreiter. First order generative adversarial networks. *International Conference on Machine Learning*, 2018.

Aaron Sidford, Mengdi Wang, Xian Wu, Lin Yang, and Yinyu Ye. Near-optimal time and sample complexities for solving markov decision processes with a generative model. In *Advances in Neural Information Processing Systems 31*, pp. 5186–5196. 2018.

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

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180, 2015.

Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, pp. 943–950, 2000.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pp. 216–224. Elsevier, 1990.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *arXiv preprint arXiv:1905.09855*, 2019.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

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

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5279–5288. 2017.

A APPENDIX

A.1 METHOD

Algorithm 2 The policy ensemble algorithm we use to find a near optimal policy μ_M for an MDP M. The $\mathcal{H}_{\vartheta_k}(a)$ measures the entropy of the sampled actions a, see Haarnoja et al. (2018) for details.

Input: MDP $(\mathcal{S}, \mathcal{A}, P^M, R^M, \gamma)$, previous states S, rollout length T, learning rates $\lambda_P, \lambda_V, \lambda_M$
Initialize: ensemble policy weights $\vartheta_1, \dots \vartheta_n$
Initialize: value function weights θ_1, θ_2
set $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2$
for each iteration **do**
  sample a state s_0 uniformly from S
  // Update the value functions
  for $j = 1, 2$ **do**
    for $k = 1, \dots, n$ **do**
      $\hat{v}_k \leftarrow \sum_{t=0}^{T} \gamma^t R^M(s_t, a_t) + \gamma^T \hat{V}_{\theta'_j}(s_T) \qquad a_t \sim \mu_{\vartheta_k}(s_t)$ and $s_{t+1} \sim P^M(\cdot \mid s_t, a_t)$
    end for
  end for
  $\theta_j \leftarrow \theta_j - \lambda_V \hat{\nabla}_{\theta_j} (\max_k \hat{v}_k - \hat{V}_{\theta'_j})^2$
  $\theta'_j \leftarrow (1 - \lambda_M)\theta'_j + \lambda_M \theta_j$ *// θ'_j is the moving average of θ'_j*
  // Update the policy functions
  for $k = 1, \dots, n$ **do**
    $\vartheta_k \leftarrow \vartheta_k + \lambda_P \hat{\nabla}_{\vartheta_k} R^M(s, a) + \min_j \hat{V}_{\theta'_j}(s') + \mathcal{H}_{\vartheta_k}(a) \qquad a \sim \mu_{\vartheta_k}(s), s' \sim P^M(\cdot \mid s, a)$
  end for
end for

A.2 EXPERIMENTAL SETTING

Since the goal of our paper was to demonstrate the effectiveness of PSRL for benchmark continuous state-space problems, we wished to avoid excessive complexity. Therefore, in all our experiments we assume knowledge of the reward could easily be learned from past interactions. Further, in keeping with Janner et al. (2019) we assume knowledge of the terminal conditions. Lastly, we use the trick outlined in Chua et al. (2018) where in HalfCheetah, all states s_i which represent angles are transformed to $[\sin(s_i), \cos(s_i)]$.

The reported soft actor critic benchmark results are from experiments provided by the authors. Their codebase `https://github.com/haarnoja/sac` provides the following link `https://drive.google.com/open?id=1I0NUrAzU7wwJQiX_MSmr1LvshjDZ4gSh` which provides five runs of SAC on each benchmark discussed in Haarnoja et al. (2018).

For the results in table 1, we ran Neural-PSRL and MBPO for 400 epochs (so 400K steps), and used the 3000 epoch (so 3M step) runs discussed above for SAC. For each run we averaged the last 50 (for Neural-PSRL and MBPO) and last 100 (for MBPO) returns, and take the average and standard deviation of these five values to report both mean and 95% confidence intervals.

The exact parameters used to learn the HalfCheetah policy:

```
 python3 -u move_main.py \
--env_id="HalfCheetah-v3" \
--gym_server_ip=<IP of MuJoCo Server> \
--gym_server_port=$port \
--env_ctrl_cost_weight=0.1 \
--env_forward_reward_weight=1.0 \
--env_survival_reward=0.0 \
--num_epochs=400 \
--checkpoint_dir="cheetah_logs/${run_id}/checkpoints" \
--log_dir="cheetah_logs/${run_id}/logs" \
```

```
--sample_dir="cheetah_logs/${run_id}/samples" \
--num_random_episodes=5 \
--is_fixed_episode_length=1 \
--max_episode_length=1000 \
--nl_num_layers=4 \
--nl_num_hidden=512 \
--nl_batch_size=256 \
--nl_lr=.0002 \
--nl_sigma=0.5 \
--nl_alpha=1.0 \
--nl_reg_scale=0.000001 \
--nl_base_num_update=8000 \
--nl_epoch_num_update=5 \
--nl_finetune_num_update=10000 \
--nl_finetune_epoch_num_update=2 \
--value_num_layers=3 \
--value_num_hidden=256 \
--value_weight_decay=0.000001 \
--policy_num_layers=3 \
--policy_num_hidden=256 \
--policy_weight_decay=0.000001 \
--a2c_batch_size=128 \
--a2c_base_num_update=2000 \
--a2c_epoch_num_update=2 \
--a2c_rollout_length=4 \
--a2c_gamma=.99 \
--a2c_policy_lr=.0003 \
--a2c_value_lr=.0003 \
--a2c_max_grad_norm=2000. \
--a2c_reward_scale=1. \
--a2c_action_pen=0. \
--a2c_val_update_eps=0.005 \
--a2c_target_entropy_mult=0.5 \
--a2c_value_push_down=0.0 \
--a2c_max_reward=20.0 \
--a2c_value_activation="lrelu" \
--a2c_policy_activation="elu" \
--a2c_num_ensemble_policies=3 \
--num_online_update=10 \
--render=0 \
--save_folder=$save_folder
```

The exact parameters used to learn the Hopper policy:

```
 python3 -u move_main.py \
--env_id="Hopper-v3" \
--gym_server_ip="<IP of MuJoCo Server> \
--gym_server_port=$port \
--env_ctrl_cost_weight=0.001 \
--env_forward_reward_weight=1.0 \
--env_survival_reward=1.0 \
--num_epochs=800 \
--checkpoint_dir="cheetah_logs/${run_id}/checkpoints" \
--log_dir="cheetah_logs/${run_id}/logs" \
--sample_dir="cheetah_logs/${run_id}/samples" \
--num_random_steps=5000 \
--is_fixed_episode_length=0 \
--max_episode_length=1000 \
--nl_num_layers=4 \
```

```
--nl_num_hidden=512 \
--nl_batch_size=256 \
--nl_lr=.0002 \
--nl_sigma=0.5 \
--nl_alpha=1.0 \
--nl_reg_scale=0.000001 \
--nl_base_num_update=8000 \
--nl_epoch_num_update=5 \
--nl_finetune_num_update=10000 \
--nl_finetune_epoch_num_update=2 \
--value_num_layers=3 \
--value_num_hidden=256 \
--value_weight_decay=0.000001 \
--policy_num_layers=3 \
--policy_num_hidden=256 \
--policy_weight_decay=0.000001 \
--a2c_batch_size=128 \
--a2c_base_num_update=8000 \
--a2c_epoch_num_update=2 \
--a2c_rollout_length=6 \
--a2c_gamma=.995 \
--a2c_policy_lr=.00005 \
--a2c_value_lr=.0003 \
--a2c_max_grad_norm=2000. \
--a2c_reward_scale=1. \
--a2c_action_pen=0. \
--a2c_val_update_eps=0.01 \
--a2c_target_entropy_mult=1.0 \
--a2c_value_push_down=0.0 \
--a2c_max_reward=5.0 \
--a2c_value_activation="lrelu" \
--a2c_policy_activation="elu" \
--a2c_num_ensemble_policies=1 \
--num_online_update=10 \
--render=1 \
--save_folder=$save_folder
```