Posterior Sampling for Deep Reinforcement Learning

Remo Sasso [1] **Michelangelo Conserva** [1] **Paulo Rauber** [1]

Abstract

Despite remarkable successes, deep reinforcement learning algorithms remain sample inefficient: they require an enormous amount of trial and error to find good policies. Model-based algorithms promise sample efficiency by building an environment model that can be used for planning. Posterior Sampling for Reinforcement Learning is such a model-based algorithm that has attracted significant interest due to its performance in the tabular setting. This paper introduces Posterior Sampling for Deep Reinforcement Learning (PSDRL), the first truly scalable approximation of Posterior Sampling for Reinforcement Learning that retains its model-based essence. PSDRL combines efficient uncertainty quantification over latent state space models with a specially tailored continual planning algorithm based on value-function approximation. Extensive experiments on the Atari benchmark show that PSDRL significantly outperforms previous state-of-the-art attempts at scaling up posterior sampling while being competitive with a state-of-the-art (model-based) reinforcement learning method, both in sample efficiency and computational efficiency.

1. Introduction

In a typical reinforcement learning problem, an agent interacts with an environment in a sequence of episodes by observing states and rewards and acting according to a policy that maps states to actions. A reinforcement learning algorithm seeks a policy that maximizes expected cumulative rewards. Because many problems in healthcare, robotics, logistics, finance, and advertising can be naturally formulated as problems of maximizing a measure of success through a sequence of decisions informed by data, the recent successes of reinforcement learning have attracted significant interest. In particular, the combination of reinforcement learning with artificial neural networks has led to the best computer agents that play games such as Chess and Go (Schrittwieser et al., 2020), Dota 2 (Berner et al., 2019), and StarCraft II (Vinyals et al., 2019).

Despite these notable successes, the corresponding reinforcement learning algorithms are remarkably *sample inefficient*: they require an enormous amount of trial-and-error to find good policies. In contrast with games and simulations, real-world applications are heavily constrained by the cost of trial-and-error and available data. As a consequence, sample inefficiency limits the applicability of reinforcement learning. This inefficiency is fundamentally linked with the trade-off between *exploring* an environment in order to learn about potentially better sources of reward and *exploiting* the well-known sources of reward. In the tabular reinforcement learning setting, where the number of states is finite, several (theoretically and often empirically) efficient exploration methods are well understood (Osband et al., 2013; Agrawal & Jia, 2017; Azar et al., 2017; Zanette & Brunskill, 2019; Russo, 2019; Ménard et al., 2021). However, there are no generally efficient methods that cope with the non-linear function approximation required in non-tabular settings.

Model-based reinforcement learning methods seek sample efficiency by building an environment model that enables predicting how actions affect the state of the environment and how the states of the environment relate to rewards. Because such a model can be used for planning (searching for a good policy without interacting with the environment), model-based methods have the potential to be substantially more sample efficient than model-free algorithms (Kaiser et al., 2019; Janner et al., 2019), which attempt to find good policies without building a model. Recent work has shown that learning models in latent state space can significantly reduce the computational cost of model-based reinforcement learning (Ha & Schmidhuber, 2018; Hafner et al., 2019b;a; Schrittwieser et al., 2020), allowing its application in environments with high-dimensional state spaces.

Posterior Sampling for Reinforcement Learning is a model-based algorithm that has attracted significant interest due to its strong (theoretical and empirical) performance in the tabular setting (Osband et al., 2013). This algorithm repre-

[1]School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom. Correspondence to: Remo Sasso <r.sasso@qmul.ac.uk>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

sents its knowledge about an environment by a distribution over environment models and repeats the following steps: a single model is drawn from the distribution over models; an optimal policy is found for this model; this policy is used to interact with the environment for one episode; and the resulting data are used to update the distribution over models. Intuitively, exploration decreases as knowledge increases.

This paper introduces Posterior Sampling for Deep Reinforcement Learning (PSDRL), the first truly scalable approximation of Posterior Sampling for Reinforcement Learning that retains its model-based essence. PSDRL encodes a high-dimensional state into a lower-dimensional latent state to enable predicting transitions in latent state space for any given action. PSDRL represents uncertainty through a Bayesian neural network that maintains a distribution over the parameters of this transition model. This enables sampling a model that can be used for planning, which is accomplished by value function approximation with an artificial neural network. This so-called value network retains information across sampled models, which is crucial for sample efficiency. Completing the algorithm, the data collected by the agent while acting greedily with respect to the value network is regularly used to update the latent state representations and the distribution over the parameters of the model.

Extensive experiments on the Atari benchmark (Bellemare et al., 2013) show that PSDRL significantly outperforms previous state-of-the-art attempts at scaling up posterior sampling (Bootstrapped DQN with randomized priors (Osband et al., 2018) and Successor Uncertainties (Janz et al., 2019)). They also show that PSDRL is competitive with a state-of-the-art (model-based) reinforcement learning method (DreamerV2 (Hafner et al., 2020)), both in sample efficiency and computational efficiency.

The remaining text is organized as follows. Section 2 relates our contributions to previous works. Section 3 describes PSDRL in technical detail. Section 4 presents the experimental protocol and its results. Finally, Section 5 summarizes our contributions and suggests future work.

2. Related works

Model-based reinforcement learning has historically underperformed in complex environments that require non-linear function approximation when compared with model-free reinforcement learning. However, two model-based methods have recently matched (or surpassed) the performance of model-free methods in the common Atari game playing benchmark (Schrittwieser et al. (2020) and Hafner et al. (2020)). This is an important development since planning has the potential to make model-based methods highly sample efficient (Kaiser et al., 2019; Janner et al., 2019).

Exploration methods that cope with the non-linear function approximation required in challenging environments can be based on one of four foundations (Badia et al., 2020): domain-specific knowledge, unsupervised policy learning, intrinsic motivation, or posterior sampling. *Domain-specific knowledge* methods typically combine human demonstrations, handcrafted features, and heuristics (Aytar et al., 2018; Ecoffet et al., 2021). Despite their potential to be highly sample efficient in their target environments, adapting these methods to new environments requires significant effort and expertise. *Unsupervised policy learning* methods encourage agents to acquire a diverse set of *skills* without receiving reward signals (Eysenbach et al., 2018). Given such limited feedback, identifying generally useful skills for efficient exploration through reuse and composition is a difficult task. *Intrinsic motivation* methods aim to encourage (re)visiting states with bonus rewards derived from ensembles of Q-functions (Chen et al., 2017; Bai et al., 2021; Tiapkin et al., 2022), visitation counts (Bellemare et al., 2016; Rashid et al., 2020), or episodic curiosity (Savinov et al., 2018; Badia et al., 2019). Although many of these exploration methods are certainly promising and often effective, our focus on posterior sampling is justified by the fact that Posterior Sampling for Reinforcement Learning is the simplest among the potentially scalable and principled methods that is capable of leveraging the strengths of both Bayesian methods and model-based reinforcement learning.

Posterior Sampling for Reinforcement Learning has been scaled up to non-tabular settings by Tziortziotis et al. (2013) and Fan & Ming (2021). Both works rely on state-action embeddings and a posterior distribution based on Bayesian linear regression. Tziortziotis et al. (2013) employ a fixed embedding and explore different approximate dynamic programming techniques (such as least-square policy iteration), which makes their algorithm limited to very simple environments. Fan & Ming (2021) learn state-action embeddings and use model predictive control (Camacho & Bordons, 2007) via the cross-entropy method (Botev et al., 2013) for planning instead of utilizing a value function, also limiting scalability relatively simple environments. The main obstacle in scaling up model-based posterior sampling to complex environments is the extreme computational and memory cost of searching for a good policy for a sampled model while retaining too little information from previous searches.

Randomized value function methods are the model-free counterparts of Posterior Sampling for Reinforcement Learning (Osband et al., 2016a; 2018; 2019). These methods (implicitly) represent their knowledge about the optimal value function by a distribution over value functions and repeat the following steps: a single value function is (implicitly) drawn from the distribution over value functions; a greedy policy is derived from this value function; this policy is used to interact with the environment during one (or more)

episodes; and the resulting data are used to (approximately) update the (implicit) distribution over value functions. In this context, Osband et al. (2016b) approximate randomized least-square value iteration. Engel et al. (2003; 2005) employ Gaussian processes trained via temporal-difference learning to induce a distribution over Q-functions. Similarly, Azizzadenesheli et al. (2018), O'Donoghue et al. (2018), and Janz et al. (2019) rely on Bayesian linear regression on top of learned state embeddings, while Flennerhag et al. (2020) approximate a distribution over temporal differences. However, explicitly maintaining and accurately updating a distribution that represents knowledge about the optimal value function is generally infeasible (Osband et al., 2019), and natural approximations of this approach may fail to realize its potential (Janz et al., 2019). This is partially due to the fact that randomized value function methods are often based on fitted value iteration, which provides unreliable and noisy learning targets (Kumar et al., 2019; 2020). In contrast with randomized value functions, model-based posterior sampling methods have the potential advantage of separating uncertainty about the environment from uncertainty about the optimal value function (or policy).

3. Posterior Sampling for Deep Reinforcement Learning

This section introduces the PSDRL algorithm, which approximates Posterior Sampling for Reinforcement Learning while retaining its model-based essence. Section 3.1 introduces our notation. Section 3.2 describes the latent state space transition model. Section 3.3 explains how uncertainty over these models is represented. Section 3.4 details planning based on sampled models. Finally, Section 3.5 justifies important implementation choices and reports potential pitfalls in scaling up model-based posterior sampling.

3.1. Preliminaries

We consider a deterministic sequential decision-making problem where, at a given time step t, an agent observes a state s_t from a state space $\mathcal{S}$ and chooses an action a_t from an action space $\mathcal{A}$ using a policy $\pi : \mathcal{S} \to \mathcal{A}$, which results in a next state $s_{t+1} = T(s_t, a_t)$ and reward $r_t = R(s_t, a_t)$. The goal of the agent is to find a policy π that maximizes the return $G(\pi) = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0, 1)$ is a discount factor. The action-value $Q_\pi(s, a)$ is the return from following policy π starting from state s and action a, while the value $V_\pi(s)$ is given by $V_\pi(s) = Q_\pi(s, \pi(s))$.

3.2. Transition model

Inspired by the recent remarkable successes of latent space models in model-based reinforcement learning (Ha & Schmidhuber, 2018; Schrittwieser et al., 2020; Hafner et al., 2019a; 2020), PSDRL employs a latent space transition model (Fig. 1) composed of the following components (subscripts denoting parameters): autoencoder (E_χ, D_χ); forward model f_θ; and termination model ω_η .

The *autoencoder* is convolutional (Masci et al., 2011). For a given state s_t, the encoder E_χ produces a latent state $z_t = E_\chi(s_t)$ that may be decoded by the decoder D_χ into a state estimate $\hat{s}_t = D_\chi(z_t)$. Encoding a state space into a lower dimensional latent state space enables more efficient planning. The *forward model* is an artificial neural network with a Gated Recurrent Unit (GRU, Bahdanau et al., 2015) that aggregates temporal information to deal with partial observability. The forward model receives a latent state z_t and an action a_t together with its hidden state h_t from a previous time step t and outputs a next latent state estimate $\hat{z}_{t+1}$, a next reward estimate $\hat{r}_t$, and a next hidden state h_{t+1}. The *termination model* ω_η receives a latent state z_t and outputs an estimate $\hat{\delta}_t \in [0, 1]$ of whether z_t corresponds to an absorbing state (end of a so-called episode).

The parameters of these three models are iteratively updated in a specific order (autoencoder, forward model, then termination model) using mini-batches stochastic gradient descent. A batch $\mathcal{B} = \{\{(s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1}, \delta_{i,t})\}_{t=0}^{L-1}\}_{i=0}^{B-1}$ is composed of B sequences of length L. Each sequence corresponds to an episode chosen at random from a replay buffer $\mathcal{D}$ with capacity C and starts at a time step chosen at random within the corresponding episode.

First, the autoencoder parameters χ are updated to minimize the reconstruction loss $\mathcal{L}_{\text{AE}}$ given by

$$\mathcal{L}_{\text{AE}}(\chi) = \frac{1}{BL} \sum_{i=0}^{B-1} \sum_{t=0}^{L-1} \| D_\chi(E_\chi(s_{i,t})) - s_{i,t} \|^2. \quad (1)$$

If we let $z_{i,t} = E_\chi(s_{i,t})$, the forward model parameters θ are then updated to minimize the loss $\mathcal{L}_{\text{F}}$ given by

$$\mathcal{L}_{\text{F}}(\theta) = \frac{1}{BL} \sum_{i=0}^{B-1} \sum_{t=0}^{L-1} \| \hat{z}_{i,t+1} - z_{i,t+1} \|^2 + (\hat{r}_{i,t} - r_{i,t})^2, \quad (2)$$

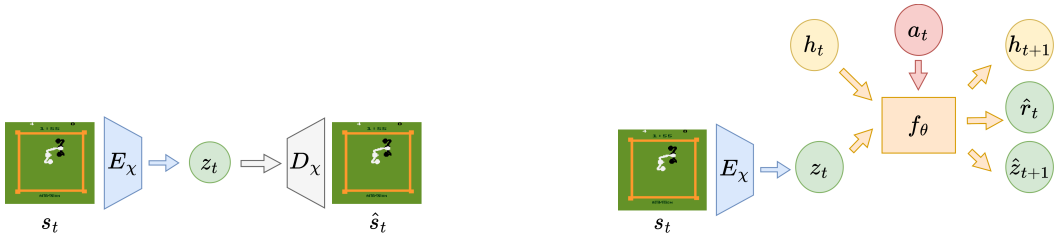
where $(\hat{z}_{i,t+1}, \hat{r}_{i,t}, h_{i,t+1}) = f_\theta(z_{i,t}, a_{i,t}, h_{i,t})$ and $h_{i,0} = 0$ for every i. Finally, the termination model parameters η are updated to minimize the loss $\mathcal{L}_{\text{T}}$ given by

$$\mathcal{L}_{\text{T}}(\eta) = \frac{1}{BL} \sum_{i=0}^{B-1} \sum_{t=0}^{L-1} (\omega_\eta(z_{i,t+1}) - \delta_{i,t})^2, \quad (3)$$

where $\delta_{i,t}$ indicates whether $s_{i,t+1}$ is an absorbing state.

3.3. Uncertainty model

PSDRL represents its uncertainty about the forward model using the neural-linear method (Snoek et al., 2015). This approach achieves remarkable success when combined with



(a) (b)

Figure 1. (a) The autoencoder learns latent state representations z_t for true environment states s_t through reconstruction. (b) The forward model learns to predict rewards r_t and next latent states z_{t+1} given the current latent states z_t, actions a_t, and previous hidden states h_t.

posterior sampling in the contextual bandits setting, especially when compared to other (often much more computationally expensive) implementations of Bayesian neural networks (Riquelme et al., 2018).

Linear forward model. In our context, the neural-linear approach models uncertainty over the parameters W of a forward model $g_W : \mathcal{X} \to \mathcal{Y}$ that ideally maps each possible vector $x = (z, a, h)$ to a vector $y = (z', r')$, where z is a latent state, a is an action, h is a hidden state, z' is the resulting latent state, and r' is the resulting reward.

First, suppose there is a known feature map $\phi : \mathcal{X} \to \mathbb{R}^k$ such that $y = g_{W^*}(x) = W^*\phi(x)$ for every input $x \in \mathcal{X}$ and some unknown matrix W^*. In words, suppose that the output of the (unknown) forward model g_{W^*} is the result of multiplying an (unknown) parameter matrix W^* by a (known) feature vector $\phi(x)$ that represents the input x. Under this assumption, Bayesian linear regression (Rasmussen, 2005) may be used to model uncertainty over each row w_j of the parameter matrix W^*, which is associated with predicting the j-th element y_j of the output vector y given the input x. Note that y_j is either an element of the next latent state z' or a reward r'.

In order to enable efficient inference, PSDRL supposes that the prior density for the vector w_j is given by $\mathcal{N}(w_j \mid 0, \sigma_j^2 I)$, where $\mathcal{N}(\cdot \mid \mu, \Sigma)$ denotes the multivariate normal density function with mean μ and covariance matrix Σ and I denotes the identity matrix. If the index j corresponds to a latent state element, we let $\sigma_j^2 = \sigma_S^2$ for a hyperparameter σ_S^2. If the index j corresponds to a reward, we let $\sigma_j^2 = \sigma_R^2$ for a hyperparameter σ_R^2.

Consider a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$ composed of all latent state transitions derived from the replay buffer $\mathcal{D}$. Let Φ denote a matrix whose i-th row is given by $\phi_i = \phi(x^{(i)})$, and let t_j denote the j-th vector of targets such that $t_j = (y_j^{(1)}, y_j^{(2)}, \ldots, y_j^{(N)})$. In order to account for potential modeling errors while still enabling efficient inference, PSDRL also supposes that the likelihood of the parameter vector w_j is given by $\mathcal{N}(t_j \mid \Phi w_j, \sigma^2 I)$, where the noise variance σ^2 is a hyperparameter. Under these assumptions, the posterior density for the vector w_j is given by $\mathcal{N}(w_j \mid \mu_j, \Sigma_j)$, where

$$\Sigma_j^{-1} = \sigma_j^{-2} I + \sigma^{-2}\Phi^T\Phi \quad \text{and} \quad \mu_j = \sigma^{-2}\Sigma_j\Phi^T t_j. \quad (4)$$

Let $\tilde{w}_j$ denote a parameter vector drawn from this posterior distribution, and let $\tilde{W}$ denote a matrix whose j-th row is given by $\tilde{w}_j$. For a given feature map ϕ, the forward model $g_{\tilde{W}}$ may be employed to predict the output $\hat{y} = g_{\tilde{W}}(x) = \tilde{W}\phi(x)$ for a given input x. In other words, $g_{\tilde{W}}$ corresponds to an environment model sampled from the posterior, as required by Posterior Sampling for Reinforcement Learning.

Feature map learning. The previous paragraphs have presupposed the existence of a known feature map $\phi : \mathcal{X} \to \mathbb{R}^k$ such that an output y could be predicted by $y = g_{W^*}(x) = W^*\phi(x)$ for every input $x \in \mathcal{X}$ and some unknown matrix W^*. In practice, the neural-linear approach derives this feature map from an (iteratively trained) forward model. More concretely, the architecture of an artificial neural network f_θ is chosen such that its prediction $\hat{y}$ for an input x is given by $\hat{y} = f_\theta(x) = W\phi_\theta(x)$, where ϕ_θ is a feature map subnetwork and the matrix W is contained in θ.

In summary, the neural-linear approach represents uncertainty solely over the parameters of the output layer of an artificial neural network whose output layer is a linear function of the last hidden layer while disregarding uncertainty about the parameters of earlier layers.

3.4. Planning

Posterior Sampling for Reinforcement Learning prescribes sampling a forward model $f_{\tilde{\theta}}$ from the corresponding posterior distribution and finding and following an optimal policy $\tilde{\pi}^*$ for this model until the end of an episode. Naturally, it is infeasible to find such an optimal policy in the non-tabular setting. This section describes how PSDRL efficiently searches for a policy for a sampled forward model.

Value function approximation. Consider once again a

batch $\mathcal{B}$ composed of B sequences of length L obtained from the replay buffer $\mathcal{D}$. Let $f_{\tilde{\theta}}$ denote a forward model sampled from the posterior such that $\tilde{\theta}$ is composed of a sampled matrix $\tilde{W}$ and parameters of a feature map subnetwork. PSDRL attempts to approximate the value function $V_{\tilde{\pi}^*}$ of an optimal policy $\tilde{\pi}^*$ for the sampled forward model $f_{\tilde{\theta}}$ through a continual procedure that updates the parameters of an artificial neural network V_ψ. The parameters ψ are updated through mini-batches stochastic gradient descent to minimize the loss $\mathcal{L}_V$ given by

$$\mathcal{L}_V(\psi) = \frac{1}{BL} \sum_{i=0}^{B-1} \sum_{t=0}^{L-1} \mathcal{L}_V^{(i,t)}(\psi), \tag{5}$$

where

$$\mathcal{L}_V^{(i,t)}(\psi) = \left(V_\psi(z_{i,t}, h_{i,t}) - \max_a \left[\hat{r}_{i,t}^{(a)} + \gamma \hat{v}_{i,t+1}^{(a)} \right] \right)^2,$$

$$(\hat{z}_{i,t+1}^{(a)}, \hat{r}_{i,t}^{(a)}, h_{i,t+1}^{(a)}) = f_{\tilde{\theta}}(z_{i,t}, a, h_{i,t}), \tag{6}$$

$$\hat{v}_{i,t+1}^{(a)} = \mathbb{1}[\omega_\eta(\hat{z}_{i,t+1}^{(a)}) < 0.5] V_{\psi'}(\hat{z}_{i,t+1}^{(a)}, h_{i,t+1}^{(a)}), \tag{7}$$

$z_{i,t} = E_\chi(s_{i,t})$, $h_{i,0} = 0$, $h_{i,t+1} = h_{i,t}^{(a_{i,t})}$, and ψ' are parameters stored from a previous iteration.

This approach is highly related to fitted value iteration (Munos & Szepesvári, 2008). In summary, the predicted value of each latent state is updated to become closer to the maximum (considering all actions) predicted next reward plus discounted value predicted for the next predicted latent state (treated as a constant). The current forward model $f_{\tilde{\theta}}$ is responsible for predicting next latent states and rewards.

Greedy policy. PSDRL derives a (greedy) policy $\tilde{\pi}$ from the value network V_ψ such that

$$\tilde{\pi}(z_t, h_t) = \operatorname*{argmax}_a \left[\hat{r}_t^{(a)} + \gamma \hat{v}_{t+1}^{(a)} \right], \tag{8}$$

where z_t is a latent state, h_t is a hidden state, $\hat{r}_t^{(a)}$ is the predicted reward after action a, and $\hat{v}_{t+1}^{(a)}$ is the predicted value for the predicted next latent state, which are obtained in analogy with Equations 6 and 7 (except ψ overrides ψ').

Algorithm 1 summarizes PSDRL (forward models are sampled every m time steps instead of episodically).

3.5. Rationale

This section discusses crucial implementation choices that enable PSDRL to scale up to complex environments.

The choice of a latent state space transition model (Sec. 3.2) enables more efficient planning in comparison with employing a (higher-dimensional) state space transition model. It is also justified by the recent success of latent state space models in model-based reinforcement learning . Similarly, the choice of the neural-linear approach for modeling uncertainty over forward models (Sec. 3.3) enables (relatively) inexpensive posterior sampling. Furthermore, the neural-linear approach has achieved remarkable empirical success with posterior sampling in the contextual bandits setting.

Algorithm 1 PSDRL

1: $\mathcal{D} \leftarrow$ empty (FIFO) buffer with capacity C
2: $s_0 \leftarrow$ initial state
3: $h_0 \leftarrow 0$
4: **for** each $t \in \{0, \ldots, T-1\}$ **do**
5: **if** $t \mod m = 0$ **then**
6: $\chi \leftarrow$ update for autoencoder loss $\mathcal{L}_{\text{AE}}$ (Eq. 1)
7: $\theta \leftarrow$ update for forward model loss $\mathcal{L}_{\text{F}}$ (Eq. 2)
8: $\eta \leftarrow$ update for termination model loss $\mathcal{L}_{\text{T}}$ (Eq. 3)
9: $\mu_j, \Sigma_j \leftarrow$ update posterior for each j (Eq. 4)
10: $\tilde{W} \leftarrow$ sample from posteriors
11: $f_{\tilde{\theta}} \leftarrow$ forward model derived from $\tilde{W}$ and feature map subnetwork ϕ_θ
12: $\psi \leftarrow$ update for value network loss $\mathcal{L}_V$ (Eq. 5) based on the sampled forward model $f_{\tilde{\theta}}$
13: **end if**
14: $z_t \leftarrow E_\chi(s_t)$
15: $a_t \leftarrow \tilde{\pi}(z_t, h_t)$ (Eq. 8)
16: $r_t, s_{t+1}, \delta \leftarrow$ outcome of action a_t
17: $h_{t+1} \leftarrow$ corresponding output from $f_{\tilde{\theta}}(z_t, a_t, h_t)$
18: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1}, \delta)\}$
19: **if** $\delta = 1$ **then**
20: $s_{t+1} \leftarrow$ initial state
21: $h_{t+1} \leftarrow 0$
22: **end if**
23: **end for**

However, the choice of planning algorithm (Sec. 3.4) is comparatively much more involved since it requires careful consideration of the strengths and weaknesses of the aforementioned components. In particular, the planning algorithm needs to avoid the following potential pitfalls.

Recency bias. The posterior update step (Alg. 1, line 9) implicitly requires recomputing all latent state transitions derived from the replay buffer $\mathcal{D}$. This is a consequence of the fact that the latent state representations are learned (as opposed to given), which precludes the typical Bayesian approach of using a posterior as the new prior when additional data is acquired. Therefore, although the forward model parameters θ are potentially influenced by all previous transitions (since they reflect all previous updates), the posterior over (output layer) parameters is strictly influenced by the transitions in the replay buffer. Because the replay buffer is necessarily limited in capacity, this introduces a bias that may severely impact the efficiency of the algorithm.

Previous works that adapt Posterior Sampling for Reinforcement Learning to non-tabular settings (Tziortziotis et al., 2013; Fan & Ming, 2021) sidestep this pitfall by employ-

ing fixed state representations or an unlimited replay buffer, which pose significant scalability challenges.

PSDRL addresses this recency bias by retaining information obtained from previous sampled models in a value network V_ψ through continual training. More concretely, the value network trained for a previous sampled model is used as the starting point for the current sampled model. Therefore, even if the entire replay buffer $\mathcal{D}$ is composed of relatively uninformative transitions (for instance, zero-reward transitions), the decisions of the agent are potentially influenced by previous planning results. This is notably distinct from the planning approaches that rely entirely on a current sampled model employed in previous works. In comparison with search methods, derivative-free optimization, and policy gradient methods, value-function approximation is naturally suited to aggregate information across sampled models, which justifies its choice. Importantly, this continual approach also improves planning efficiency when the current sampled model is similar to the previous sampled model. However, training a value function approximator across sampled models introduces another potential pitfall.

Status quo bias. Posterior Sampling for Reinforcement Learning prescribes sampling a forward model from the corresponding posterior distribution and following an optimal policy for this model for at least one episode. The fact that a posterior sampling agent may radically change its behavior based on a sampled model is crucial for efficient exploration. Therefore, employing a value network that aggregates information across sampled models as a starting point to train a value network for a current model may bias the agent in a way that is detrimental to exploration.

PSDRL addresses this status quo bias by choosing actions that maximize the one-step return, which combines the next reward predicted by the sampled model with the value predicted by the value network for the next (latent) state predicted by the sampled model (Eq. 8). For instance, instead of considering the one-step return, it would be possible to train a typical action-value network Q_ψ and choose the action $\operatorname{argmax}_a Q_\psi(z_t, h_t, a)$ for any given latent state z_t and hidden state z_t. However, unless Q_ψ changes significantly after a relatively brief training procedure (Alg. 1, line 12), the resulting policy (Alg. 1, line 15) could fail to incorporate sufficient feedback from the current sampled model. In order to further reduce the status quo bias, it would also be possible (but more expensive) to consider the k-step return of the greedy policy with respect to V_ψ (or Q_ψ).

In summary, there is a natural trade-off between recency bias and status quo bias. Section 4.3 reports an ablation study that conclusively shows that disregarding planning results from previous sampled models is not a suitable alternative under realistic computational constraints, whereas our choices lead to efficient exploration.

Table 1. Median and mean human-normalized score (Mnih et al., 2015), mean record-normalized score (Toromanoff et al., 2019), and mean clipped record-normalized score (Hafner et al., 2020).

Metric	PSDRL	Dv2	B+P	SU
Gamer Median	**23**%	18%	7%	0%
Gamer Mean	58%	**100**%	0%	21%
Record Mean	6%	**8**%	0%	0%
Clipped Record Mean	6%	**8**%	0%	0%

4. Experiments

4.1. Experimental protocol

Environments. We provide an experimental comparison between PSDRL and other algorithms on 55 Atari 2600 games that are commonly used in the literature (Mnih et al., 2015). We evaluate each algorithm on all 55 games using no full action space, no access to life information, no sticky actions, and an action repeat of four for three random seeds. In order to keep to a feasible computational budget, we restrict environment steps to 1M (4M frames).

Algorithms. Since previous model-based posterior sampling algorithms do not scale to complex environments, we make a comparison with two state-of-the-art randomized value function (RVF) algorithms: Bootstrapped DQN with randomized priors (B+P, Osband et al., 2018) and Successor Uncertainties (SU, Janz et al., 2019). Additionally, we make a comparison with the model-based algorithm DreamerV2 (Dv2, Hafner et al., 2020), which is the state-of-the-art single-GPU algorithm for the Atari benchmark.

Evaluation metrics. Traditionally, authors report the mean and median human-normalized scores across all games (gamer mean and gamer median) (Mnih et al., 2015). A human-normalized score of zero corresponds to a randomly acting agent, whereas a human-normalized score of one corresponds to a professional gamer. However, Toromanoff et al. (2019) convincingly argue that the gamer mean can be misleading due to outliers, whereas the gamer median can be misleading if almost half of the scores are zero. Therefore, they propose to normalize according to the human world records instead. Hafner et al. (2020) further propose the clipped record-normalized score. A record-normalized score of zero corresponds to a randomly acting agent, whereas a record-normalized score of one corresponds to a world record performance. Performances that exceed the world record receive a clipped record-normalized score of one. We report both traditional human-normalized scores and (clipped) record-normalized scores, which are computed using the average evaluation return across 1M environment steps, measured every 10k environment steps.

Hyperparameters. As the original hyperparameters for the

baseline algorithms were tuned for 200M frames, and sticky actions in the case of Dv2, we have tuned each baseline for the deterministic 4M frames setting to guarantee a fair comparison. Hyperparameters are obtained with an exhaustive grid search on six Atari games commonly used for this purpose (Munos et al., 2016; Janz et al., 2019): ASTERIX, ENDURO, FREEWAY, HERO, QBERT, and SEAQUEST.

Reproducibility. The source code for replicating all experiments is available as supplementary material. A description of the hyperparameter search procedure and other implementation decisions are available in Appendix D.

4.2. Main results

The results for each evaluation metric are summarized in Table 1. Note that although SU significantly outperforms B+P in terms of gamer mean, this is almost entirely due to its excellent performance on BERZERK, which is just one example of how this traditional metric is flawed. A more informative visualization comparing PSDRL to each baseline algorithm in terms of human-normalized score is presented in Figure 4 (Appendix C details this visualization). The raw game scores and corresponding learning curves for each individual game can be found in Appendices A and B, respectively. Remarkably, the results show that PSDRL is competitive with the state-of-the-art algorithm Dv2 in terms of sample efficiency while using a similar computational budget. Additionally, PSDRL substantially outperforms the state-of-the-art RVF baselines in almost all of the games.

Appendix F shows that PSDRL is able to decode latent state predictions obtained from sampled forward models close to perfectly for many games at the end of training, even when the corresponding state images have many details. Exceptionally, the fact that PSDRL and Dv2 receive downscaled 64×64 images (compared to 84×84 for B+P) as state representations makes it difficult for their agents to distinguish between the enemies depicted with small differences in color and shape in games like BEAMRIDER within 1M environment steps, which explains why they are outperformed by B+P in this case (see Table A). Therefore, the state representation and the model capacity of an autoencoder introduce an additional trade-off between sample efficiency and computational efficiency.

4.3. Ablation studies

This section details two ablation studies that we conducted to better understand the importance of posterior sampling and our continual value function approximation approach.

Posterior sampling. In order to investigate the significance of posterior sampling in PSDRL, we make a comparison with two ϵ-greedy approaches that select exploratory actions at random with probability ϵ. Concretely, these approaches employ the same architecture and hyperparameters as PSDRL but do not sample forward models from the corresponding posterior. Instead, these approaches always use the current forward model parameters θ to make predictions. In order to provide a fair comparison, we linearly anneal ϵ from 1 to 0.01 in two different schemes: an *exploitative* annealing scheme across 50k environment steps, and an *explorative* annealing scheme across 1M environment steps. The results in Figure 2 clearly show that the success of PSDRL is heavily dependent on the natural balancing between exploration and exploitation provided by posterior sampling. For instance, in the sparse-reward environment FREEWAY, posterior sampling allows the agent to learn the optimal policy rapidly, whereas the explorative agent is slowed down significantly and the exploitative agent settles on a sub-optimal policy.

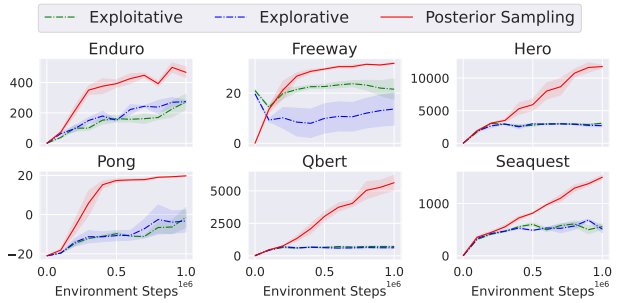


Figure 2. Average evaluation episode returns comparing posterior sampling to *exploitative* and *explorative* ϵ-greedy approaches.

Continual value network. Section 3.5 provides the rationale behind our continual approach that trains a value function approximator across sampled models instead of using a newly initialized value network for each sampled model. In order to provide a fair comparison between these two approaches, we quadruple the number of iterations used to train newly initialized value functions at every planning step (Alg 1, line 12). The results in Figure 3 show that,

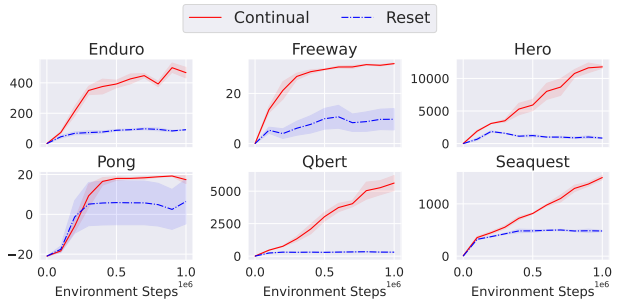


Figure 3. Average evaluation episode returns comparing planning continually with planning with newly initialized value functions.

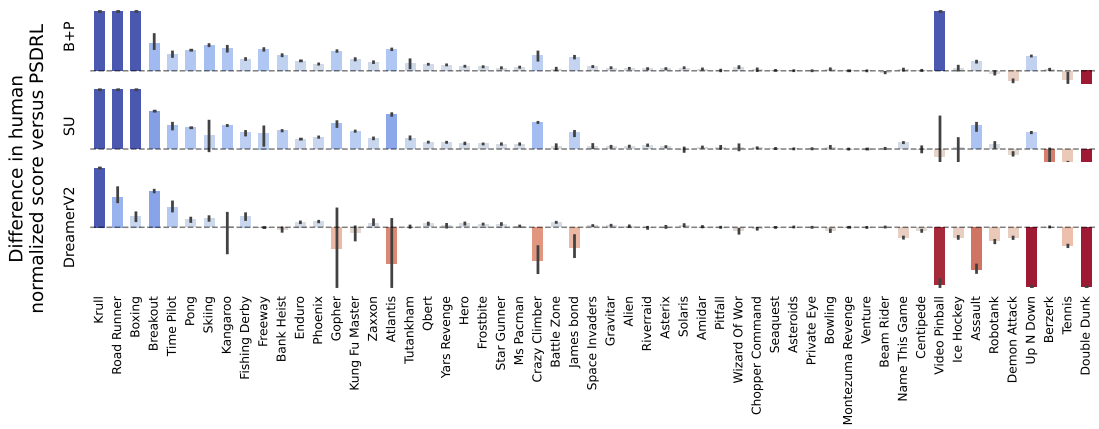


Figure 4. A comparison between PSDRL and B+P (top), SU (middle), and Dv2 (bottom) for all 55 Atari games. Blue indicates that PSDRL scores higher than the baseline, red indicates the opposite. Y-axis has been clipped to $[-2.5, 2.5]$ to facilitate visualization.

even when the total computational cost is increased by approximately 60%, using a newly initialized value network at every planning step is detrimental to sample efficiency. This clearly justifies biasing planning towards the optimal policy for previously sampled models.

5. Conclusion

We introduced Posterior Sampling for Deep Reinforcement Learning (PSDRL), the first truly scalable approximation of Posterior Sampling for Reinforcement Learning that retains its model-based essence. Although there are many other promising approaches towards efficient exploration, Posterior Sampling for Reinforcement Learning is the simplest among the potentially scalable and principled methods that is capable of leveraging the strengths of both Bayesian methods and model-based reinforcement learning.

In addition to an efficient architecture for representing uncertainty over latent state space transition models inspired by recent successes in model-based reinforcement learning and contextual multi-armed bandits, our technical contributions include a specially tailored continual planning algorithm based on value-function approximation that addresses two newly identified potential pitfalls in scaling up Posterior Sampling for Reinforcement Learning (recency bias and status quo bias). Our extensive experiments on the Atari benchmark show that PSDRL significantly outperforms previous state-of-the-art randomized value function approaches, its natural model-free counterparts, while being competitive with a state-of-the-art (model-based) reinforcement learning method in both sample efficiency and computational efficiency. Our approach is further validated by ablation studies that show the importance of retaining planning information across sampled models and the superiority of posterior sampling over a naive exploration method.

Similarly to most deep reinforcement learning algorithms, the main weakness of PSDRL is the cost of successfully dealing with traditionally *hard* environments (such as those with very sparse rewards), which are defined by having high visitation complexity or estimation complexity (Conserva & Rauber, 2022). Such environments naturally require large replay buffers (to counteract the recency bias), lower forward model sampling frequency (to encourage deep exploration (Osband et al., 2019)), and longer planning times (to obtain better policies). Some of these issues may be mitigated by employing smart replay buffers and strategies such as prioritized experience replay (Schaul et al., 2016). Employing a latent state space model architecture that allows incremental posterior updates is another promising approach, although it requires finding a careful balance between reliable uncertainty quantification and computational efficiency.

There are numerous possibilities for future work. The most important and demanding is to investigate the many feasible combinations of latent state space model architectures with Bayesian neural networks. Our experience indicates that it is crucial to tailor the corresponding planning algorithm while considering the strengths and weaknesses of the remaining components. Extending our approach to deal with more general non-deterministic environments is also especially important. Finally, significant insight into our approach may be gained from studying it from a theoretical perspective.

Acknowledgements

This research was financially supported by the Intelligent Games and Games Intelligence CDT (IGGI;EP/S022325/1) and utilized Queen Mary's Apocrita HPC facility (http://doi.org/10.5281/zenodo.438045).

References

Agrawal, S. and Jia, R. Optimistic posterior sampling for reinforcement learning: worst-case regret bounds. *Advances in Neural Information Processing Systems*, 30, 2017.

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and De Freitas, N. Playing hard exploration games by watching youtube. *Advances in Neural Information Processing Systems*, 31, 2018.

Azar, M. G., Osband, I., and Munos, R. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, volume 70, pp. 263–272. PMLR, 2017.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. Efficient Exploration Through Bayesian Deep Q-Networks. In *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9, 2018.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*, 2019.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Bahdanau, D., Cho, K. H., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Bai, C., Wang, L., Han, L., Hao, J., Garg, A., Liu, P., and Wang, Z. Principled exploration via optimistic bootstrapping and backward induction. In *International Conference on Machine Learning*, pp. 577–587. PMLR, 2021.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 29, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint*, 2019.

Botev, Z. I., Kroese, D. P., Rubinstein, R. Y., and L'Ecuyer, P. Chapter 3 - The Cross-Entropy Method for Optimization. In *Handbook of Statistics*, volume 31, pp. 35–59. Elsevier, 2013.

Camacho, E. F. and Bordons, C. *Model Predictive control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2007.

Chen, R. Y., Sidor, S., Abbeel, P., and Schulman, J. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

Conserva, M. and Rauber, P. Hardness in markov decision processes: Theory and practice. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Engel, Y., Mannor, S., and Meir, R. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *International Conference on Machine Learning*, pp. 154–161. PMLR, 2003.

Engel, Y., Mannor, S., and Meir, R. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, pp. 201–208. PMLR, 2005.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*, 2018.

Fan, Y. and Ming, Y. Model-based Reinforcement Learning for Continuous Control with Posterior Sampling. In *International Conference on Machine Learning*, pp. 3078–3087. PMLR, 2021.

Flennerhag, S., Wang, J. X., Sprechmann, P., Visin, F., Galashov, A., Kapturowski, S., Borsa, D. L., Heess, N., Barreto, A., and Pascanu, R. Temporal difference uncertainties as a signal for exploration. *arXiv preprint arXiv:2010.02255*, 2020.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems*, 31, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*, 2019a.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*, 2020.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Janz, D., Hron, J., Mazur, P., Hofmann, K., Hernández-Lobato, J. M., and Tschiatschek, S. Successor uncertainties: exploration and uncertainty in temporal difference learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model Based Reinforcement Learning for Atari. In *International Conference on Learning Representations*, 2019.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Kumar, A., Gupta, A., and Levine, S. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33, 2020.

Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pp. 52–59. Springer, 2011.

Ménard, P., Domingues, O. D., Shang, X., and Valko, M. UCB Momentum Q-learning: Correcting the bias without forgetting. In *International Conference on Machine Learning*, pp. 7609–7618. PMLR, 2021.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Munos, R. and Szepesvári, C. Finite-Time Bounds for Fitted Value Iteration. *Journal of Machine Learning Research*, 9(5):815–857, 2008.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 29, 2016.

Osband, I., Russo, D., and Van Roy, B. (More) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, 26, 2013.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped DQN. *Advances in Neural Information Processing Systems*, 29, 2016a.

Osband, I., Van Roy, B., and Wen, Z. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pp. 2377–2386. PMLR, 2016b.

Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Osband, I., Van Roy, B., Russo, D. J., and Wen, Z. Deep Exploration via Randomized Value Functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pp. 3839–3848. PMLR, 2018.

Rashid, T., Peng, B., Böhmer, W., and Whiteson, S. Optimistic Exploration even with a Pessimistic Initialisation. In *International Conference on Learning Representations*, 2020.

Rasmussen, C. E. *Gaussian processes for machine learning*. MIT press, 2005.

Riquelme, C., Tucker, G., and Snoek, J. Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling. In *International Conference on Learning Representations*, 2018.

Russo, D. Worst-case regret bounds for exploration via randomized value functions. *Advances in Neural Information Processing Systems*, 32, 2019.

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic Curiosity through Reachability. In *International Conference on Learning Representations*, 2018.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized Experience Replay. In *International Conference on Learning Representations*, 2016.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pp. 2171–2180. PMLR, 2015.

Tiapkin, D., Belomestny, D., Moulines, E., Naumov, A., Samsonov, S., Tang, Y., Valko, M., and Menard, P. From dirichlet to rubin: Optimistic exploration in rl without bonuses. In *International Conference on Machine Learning*. PMLR, 2022.

Toromanoff, M., Wirbel, E., and Moutarde, F. Is Deep Reinforcement Learning Really Superhuman on Atari? Leveling the playing field. *arXiv preprint arXiv:1908.04683*, 2019.

Tziortziotis, N., Dimitrakakis, C., and Blekas, K. Linear Bayesian reinforcement learning. In *International joint conference on artificial intelligence*, 2013.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Zanette, A. and Brunskill, E. Tighter problem-dependent regret bounds in reinforcement learning without domain knowledge using value function bounds. In *International Conference on Machine Learning*, pp. 7304–7312. PMLR, 2019.

A. Atari benchmark: average returns

Table 2. Average return for evaluation episodes in each individual game (1M environment steps, averaged across three seeds).

Game	PSDRL	B+P	SU	Dv2
Alien	**1199**	505	386	844
Amidar	**156**	36	19	117
Assault	662	461	147	**1589**
Asterix	**1235**	514	431	1090
Asteroids	**1141**	932	760	549
Atlantis	35273	20578	11174	**163070**
Bank Heist	596	125	18	**673**
Battle Zone	**10612**	8127	6007	3275
Beam Rider	779	**2091**	370	587
Berzerk	386	226	**33644**	339
Bowling	15	5	4	**36**
Boxing	**79**	6	-46	73
Breakout	**46**	8	0	2
Centipede	2594	2446	2888	**4067**
Chopper Command	899	502	589	**1289**
Crazy Climber	30392	17942	2086	**65140**
Demon Attack	410	1225	776	**1233**
Double Dunk	-16	-6	**-1**	-4
Enduro	**363**	0	0	178
Fishing Derby	**-60**	-86	-96	-85
Freeway	**28**	1	7	**28**
Frostbite	**929**	204	6	354
Gopher	2683	865	316	**5194**
Gravitar	**494**	92	208	222
Hero	**7965**	2202	768	3096
Ice Hockey	-12	-13	-13	**-6**
James bond	231	69	41	**470**
Kangaroo	**3180**	384	190	3063
Krull	**10802**	2235	903	7301
Kung Fu Master	17528	6647	412	**22249**
Montezuma Revenge	0	0	**7**	0
Ms Pacman	**1824**	812	429	1502
Name This Game	4037	3767	2473	**6534**
Phoenix	**3876**	1756	323	1981
Pitfall	-44	-63	-469	**-39**
Pong	**11**	-20	-21	0
Private Eye	68	-53	-793	**79**
Qbert	**4245**	560	337	2188
Riverraid	3858	2488	1235	**4174**
Road Runner	**22272**	528	984	11954
Robotank	3	5	2	**10**
Seaquest	1070	307	96	**1282**
Skiing	**-14980**	-28748	-22105	-19777
Solaris	1794	396	**2083**	866
Space Invaders	**511**	240	335	403
Star Gunner	**2542**	1349	531	1188
Tennis	-22	-17	-13	**-10**
Time Pilot	**4156**	2996	2390	2767
Tutankham	**94**	60	19	89
Up N Down	8596	1587	794	**90684**
Venture	0	3	0	**8**
Video Pinball	8857	1145	8641	**14973**
Wizard Of Wor	1459	848	1181	**2124**
Yars Revenge	**17038**	3657	1804	12144
Zaxxon	**4413**	1158	206	2671

B. Atari benchmark: learning curves

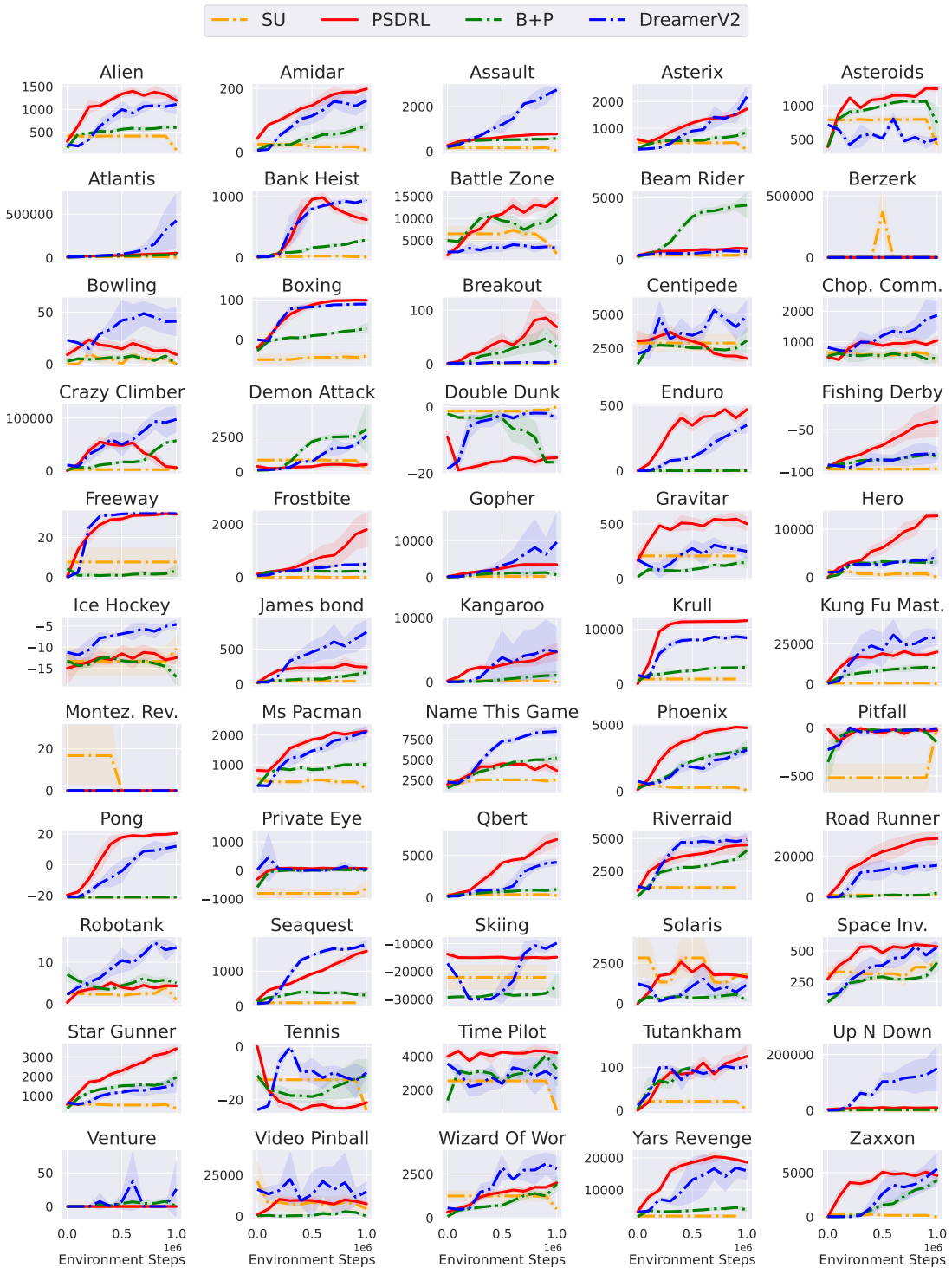


Figure 5. Returns for evaluation episodes in each individual game (1M environment steps, averaged across three seeds).

C. Normalized score visualizations

Figures 6 and 7 compare PSDRL with the baselines on 55 Atari games in terms of scores normalized with respect to a representative human performance or the human record performance, respectively. Concretely, the height of each bar corresponds to the difference between the normalized score of PSDRL and the normalized score of a given baseline. The vertical black lines represent 95% bootstrapped confidence intervals. Darker shades of blue correspond to a wider gap in favor of PSDRL, darker shades of red correspond to a wider gap in favor of a given baseline. In Figure 6, the height of each bar has been clipped to the interval $[-2.5, 2.5]$, since otherwise most bars would be difficult to see.

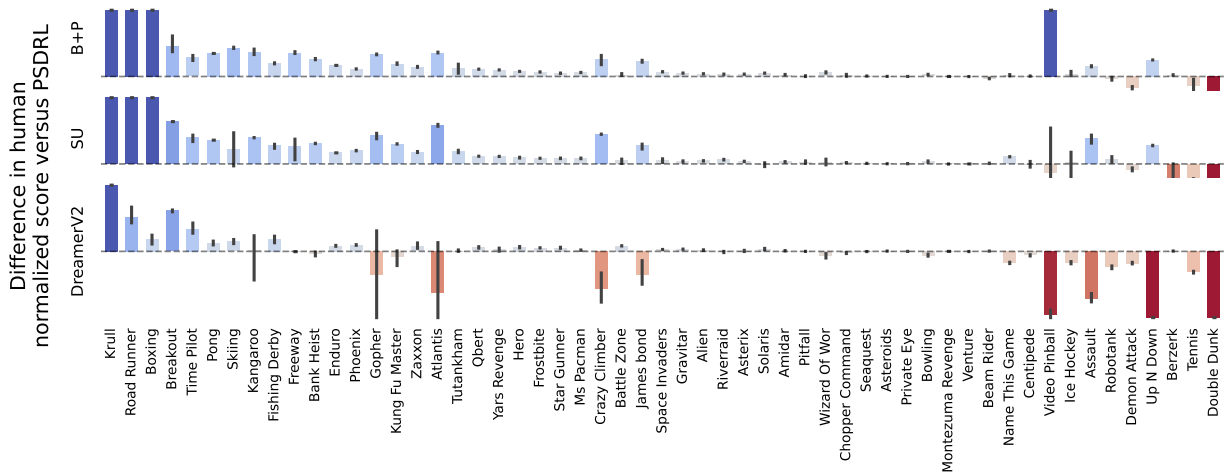


Figure 6. A comparison between PSDRL and B+P (top), SU (middle), and Dv2 (bottom) in terms of human-normalized score. Blue indicates that PSDRL scores higher than the baseline, red the opposite. Y-axis has been clipped to $[-2.5, 2.5]$ to facilitate visualization. This figure is identical to Figure 4 and is presented here to facilitate comparison with Figure 7.

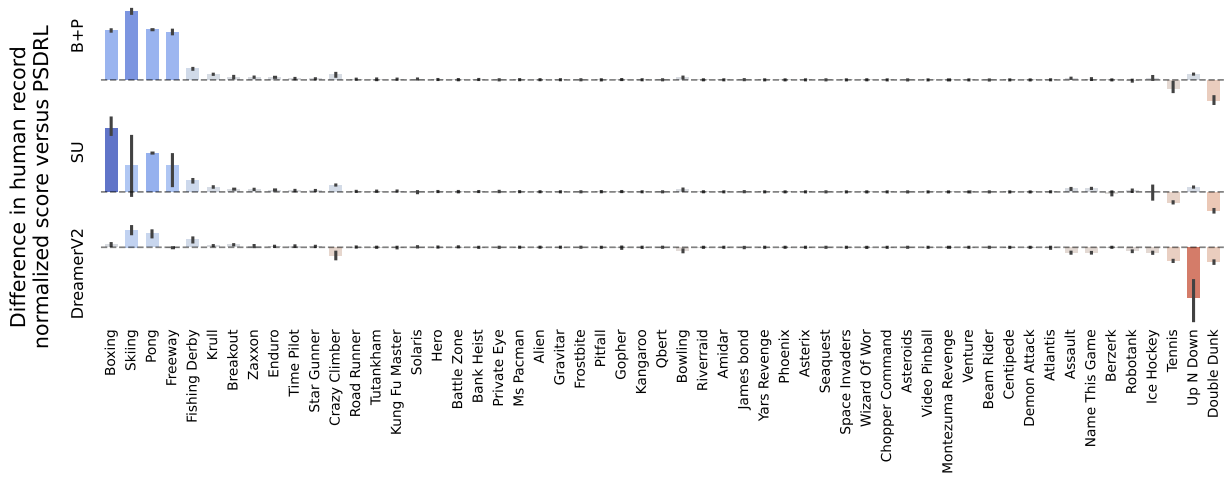


Figure 7. A comparison between PSDRL and B+P (top), SU (middle), and Dv2 (bottom) in terms of record-normalized score. Blue indicates that PSDRL scores higher than the baseline, red the opposite.

D. Hyperparameters and implementation details

Hyperparameters search was conducted through a grid search over a carefully selected subset of hyperparameters for each algorithm. The evaluation metric employed for this search (average return during evaluation episodes) is further averaged over results on six Atari games that are commonly used for this purpose (Munos et al., 2016; Janz et al., 2019): ASTERIX, ENDURO, FREEWAY, HERO, QBERT, and SEAQUEST. In order to guarantee a fair comparison, we chose hyperparameters that allow each algorithm to finish training for each game in under 10 hours.

D.1. Posterior sampling for deep reinforcement learning

The implementation for PSDRL can be found at `https://github.com/remosasso/PSDRL`. Table 3 presents the hyperparameters for PSDRL, the search sets used for grid search, and the resulting values used for the experiments. Additionally, note that we update the components more frequently ($m = 250$) during the first 100k environment steps in comparison with the remaining steps ($m = 1000$). See Figure 8 for a diagram that demonstrates how the components of PSDRL interact when interacting with the environment.

PSDRL receives states represented as 64×64 grayscale images and does not require frame stacking due to employing a recurrent forward model. The autoencoder model, forward model, termination model, and value network are trained with the Adam optimizer (Kingma & Ba, 2015) using a learning rate of 1e-4.

In our implementation, a batch $\mathcal{B}$ containing $B \times L$ transitions is sampled and used in slightly different ways by each component in a given training iteration. The autoencoder model performs B gradient updates with an *inner batch* of size L. The forward model, termination model, and value network perform L/l gradient updates with a batch of size B, where l denotes the horizon for truncated backpropagation through time. For the forward and termination models, $l = 4$. For the value network, $l = 1$. Therefore, according to the parameters in Table 3, the parameters of the value network are updated $\kappa L/l = 750$ times with batches of size $B = 125$ during each planning iteration (Alg. 1, line 12).

Instead of choosing actions strictly according to the greedy policy (Eq. 8), our implementation of PSDRL chooses actions according to the corresponding $\hat{\epsilon}$-greedy policy, where $\hat{\epsilon}$ is a very small value. This ensures that the agent does not waste all of its time caught in a loop between two states early in training even if the sampled model would justify this behavior.

We make use of an NVIDIA A100 GPU for training. PSDRL takes about 9 hours per 1 million environment steps per game.

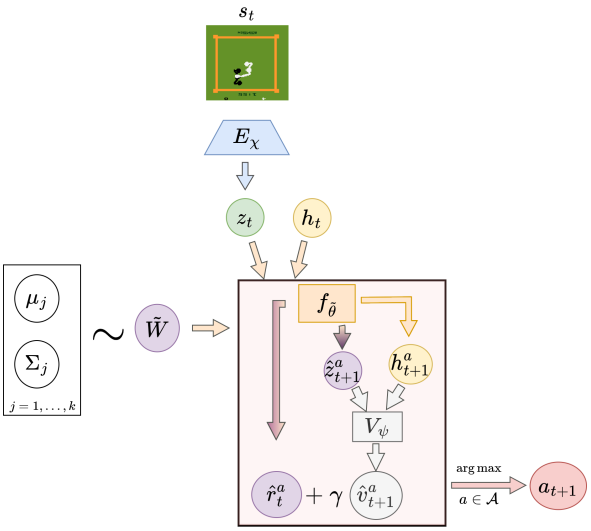


Figure 8. Diagram of the interactions between components in PSDRL when interacting with the environment. At timestep t, the agent encodes an observation s_t to latent state z_t. Given z_t and the previous hidden state h_t, the forward model $f_{\tilde{\theta}}$ with sampled parameters $\tilde{W}$ predicts the next latent state $\hat{z}^a_{t+1}$ and reward $\hat{r}^a_t$ for every action $a \in \mathcal{A}$. For each of the predicted next latent states, the value function V_ψ predicts a valuation $\hat{v}^a_{t+1}$. Finally, an action is yielded by computing: $a_{t+1} = \text{argmax}_a \left[\hat{r}^{(a)}_t + \gamma \hat{v}^{(a)}_{t+1}\right]$, and the corresponding hidden state h^a_{t+1} is carried over to the next timestep.

Table 3. Hyperparameters for PSDRL.

Name	Symbol	Search set	Value used
Bayesian linear regression			
Prior variance for latent state parameters	σ_S^2	$\{$1e1, 1e3, 1e5$\}$	1e3
Prior variance for reward parameters	σ_R^2	$\{$1e1, 1e3, 1e5$\}$	1e3
Noise variance	σ^2	—	1
Forward model			
Number of layers		—	5
Activation function		—	Tanh
Hidden units		—	2292
Learning rate		—	1e-4
Training iterations		—	3
Recurrent hidden units		—	756
Window update length		l	4
Terminal model			
Number of layers		—	4
Activation function		—	Tanh
Hidden units		—	1536
Learning rate		—	1e-4
Training iterations		—	3
Value network			
Number of layers		—	5
Activation function		—	Tanh
Hidden units		—	2292
Learning rate		—	1e-4
Training iterations	κ	—	3
Target update frequency		4	
Discount factor	γ	$\{$0.99, 0.999$\}$	0.99
Autoencoder model			
Number of encoder layers		—	4
Number of decoder layers		—	4
Activation function		—	ReLu
Encoded dimensions		—	1536
Learning rate		—	1e-4
Training iterations		—	3
Replay buffer			
Batch size	B	$\{$125, 250$\}$	125
Sequence length	L	—	250
Capacity	C	—	1e5
Environment interaction			
Update frequency	m	—	1000
Policy noise	$\hat{\epsilon}$	0.001	

D.2. Successor uncertainties

Via private communication, the authors of SU suggested that three hyperparameters should be tuned to improve the sample efficiency of their algorithm. These hyperparameters are reported in Table 4.

We make use of the implementation published by the authors at https://github.com/DavidJanz/successor_uncertainties_atari.

Table 4. Hyperparameters for SU.

Name	Search set	Value used
Likelihood prior	{1e-1, 1e-3, 1e-5}	1e-3
Training interval	{2, 4}	2
Batch size	{32, 64}	64

D.3. Bootstrapped DQN with randomized priors

Because B+P shares several components with SU, we selected a similar set of hyperparameters to tune for sample efficiency. These hyperparameters are reported in Table 5.

We make use of an implementation for Atari available at https://github.com/johannah/bootstrap_dqn.

Table 5. Hyperparameters for B+Q.

Name	Search set	Value used
Likelihood prior	{1,3,10}	1
Training interval	{2, 4}	2
Batch size	{32, 64}	64

D.4. DreamerV2

The authors of Dv2 mention that the training interval (originally 16) should be decreased (Hafner et al., 2020) to increase sample efficiency. Table 6 reports the corresponding values employed for grid search.

We make use of the implementation published by the authors at https://github.com/danijar/dreamerv2.

Table 6. Hyperparameters for Dv2.

Name	Search set	Value used
Training interval	{4,8,12}	8

E. Runtime comparison

Table 7 reports the amount of wall clock time required for 1M environment steps in 7 games for each of the algorithm implementations used in this paper. Note that this is the computational efficiency on an NVIDIA A100 GPU *after* tuning the baselines for data efficiency.

Table 7. Wall clock time for 1M environment steps for each of the algorithms after tuning.

Game	**SU**	**PSDRL**	**B+P**	**Dv2**
Freeway	5h3m $\pm$ 20m	8h53m $\pm$ 0m	8hr11m $\pm$ 24m	11h20m $\pm$ 31m
Qbert	6h28m $\pm$ 23m	7h39m $\pm$ 43m	9h43m $\pm$ 9m	10h52m $\pm$ 20m
Enduro	5h13m $\pm$ 25m	9h35m $\pm$ 15m	8h12m $\pm$ 32m	10h58m $\pm$ 3m
Asterix	4h54m $\pm$ 4m	7h44m $\pm$ 24m	9h7m $\pm$ 14m	10h18m $\pm$ 18m
Seaquest	5h30m $\pm$ 27 m	8h31m $\pm$ 2m	8h56m $\pm$ 15m	10h33m $\pm$ 7m
Pong	7h0m $\pm$ 34m	7h58m $\pm$ 42m	8h58m $\pm$ 6m	11h7m $\pm$ 35m
Hero	5h46m $\pm$ 19m	9h26m $\pm$ 2m	9h32m $\pm$ 28m	12h24m $\pm$ 26m
Average	**5h41m $\pm$ 64m**	**8h31m $\pm$ 44m**	**8h56m $\pm$ 32m**	**11h3m $\pm$ 37m**

F. Decoded sampled forward model predictions

Figures 9 and 10 provide a comparison between a true environment state image s_{t+1} and a decoded state image $\hat{s}_{t+1} = D_\chi(\hat{z}_{t+1})$ after training 1M environment steps, where $\hat{z}_{t+1}$ is the latent state predicted by a sampled forward model $f_{\tilde{\theta}}$ given a state image s_t encoded as $z_t = E_\chi(s_t)$, action a_t, and hidden state h_t. Figures 9 and 10 also show the corresponding error image $\hat{s}_{t+1} - s_{t+1}$. As discussed in Section 4.2, BEAM RIDER state images contain important details that are missed (see second and third error images in the last row of Fig. 10).

Additionally, using decoded predictions of sampled forward models we can illustrate the uncertainty quantification over environment models. The following video showcases decoded latent rollouts generated by different sampled forward models, where each rollout was initiated from the same starting state after 75k environment steps of training: https://gifyu.com/image/SIMGF. The video clearly demonstrates significant diversity in predictions made by distinct sampled models, resulting in disparate trajectories over time, consequently driving the exploration of the agent.

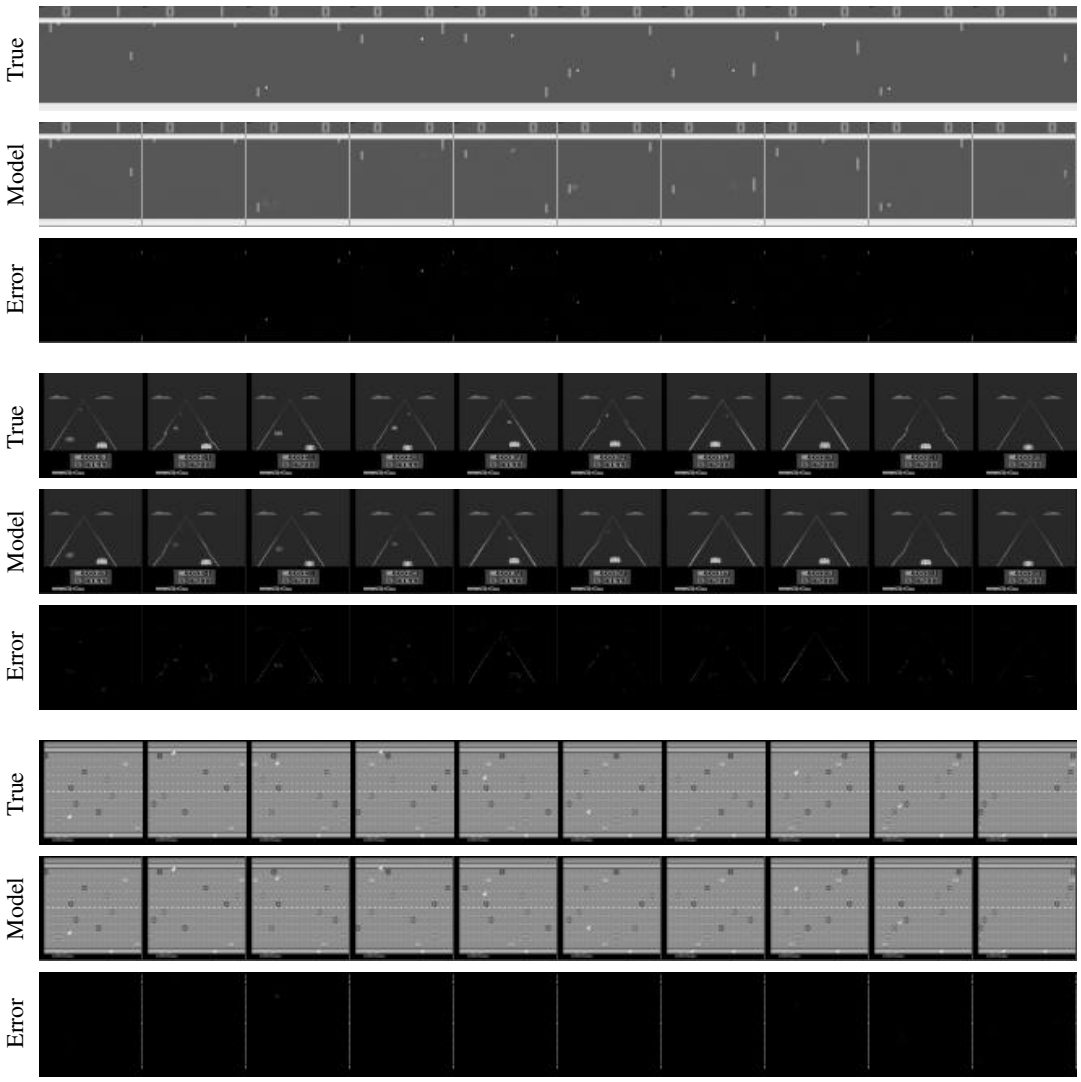


Figure 9. Decoded sampled forward model predictions compared to the true environment state for PONG, ENDURO, and FREEWAY.

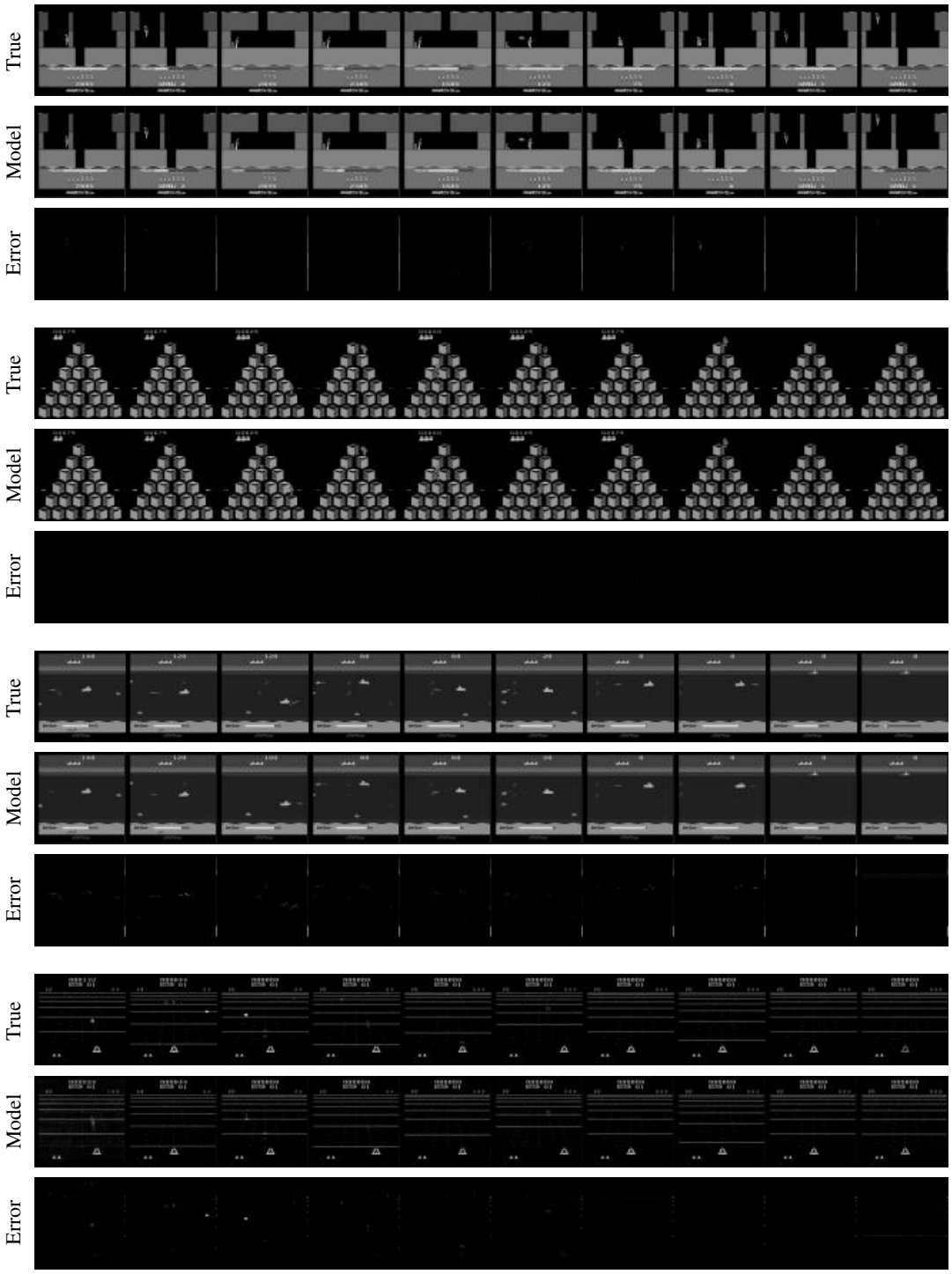


Figure 10. Decoded sampled forward model predictions compared to the true environment state for HERO, QBERT, SEAQUEST and BEAM RIDER.