Robust Ensemble Adversarial Model-Based Reinforcement Learning

Daniele Foffano
TU Delft
Delft, Netherlands
D.Foffano@student.tudelft.nl

Jinke He
TU Delft
Delft, Netherlands
J.He-4@tudelft.nl

Frans A. Oliehoek
TU Delft
Delft, Netherlands
F.A.Oliehoek@tudelft.nl

ABSTRACT

Model-Based Reinforcement Learning (MBRL) algorithms solve sequential decision-making problems, usually formalised as Markov Decision Processes, using a model of the environment dynamics to compute the optimal policy. When dealing with complex environments, the environment dynamics are frequently approximated with function approximators (such as Neural Networks) that are not guaranteed to converge to an optimal solution. As a consequence, the planning process using samples generated by an imperfect model is also not guaranteed to converge to the optimal policy. In fact, the mismatch between source and target dynamics distribution can result in compounding errors, leading to poor algorithm performance during testing. To mitigate this, we combine the Robust Markov Decision Processes (RMDPs) framework and an ensemble of models to take into account the uncertainty in the approximation of the dynamics. With RMDPs, we can study the uncertainty problem as a two-player stochastic game where Player 1 aims to maximize the expected return and Player 2 wants to minimize it. Using an ensemble of models, Player 2 can choose the worst model to carry out the transitions when performing rollout for the policy improvement. We present Robust Ensemble AdversariaL (REAL) MBRL, an ensemble-based algorithm leveraging the use of an Adversarial agent to compute a policy more robust to model errors. We propose two adversarial approaches: one with a greedy adversary and one with an ϵ-random adversary. We experimentally show that finding a maximin strategy for this two-player game results in a policy robust to model errors leading to better performance when compared to assuming the learned dynamics to be correct.[1]

KEYWORDS

Reinforcement Learning, Robust MDPs, Adversarial attacks

1 INTRODUCTION

In recent years, Deep Reinforcement Learning algorithms have seen notable growth in the literature. These methods have accomplished remarkable results in several fields, ranging from games [26, 38–40] to robotics [17, 22, 23]. In the literature, we can observe two classes of Deep RL methods: Model-Free and Model-Based. Model-Free Reinforcement Learning (MFRL) algorithms learn policies by interacting directly with the real world. Model-Based Reinforcement Learning (MBRL) methods build an approximate model of the source MDP dynamics[2], learning the optimal policy using simulated data. A well-established approach in MBRL is to implement Dyna-like architectures [43], where the agent iteratively cycles between improving the policy with artificial data and fitting the model to samples collected with the improved policy. MFRL methods have long learning times [16] and, thus, require more environment samples than Model-Based algorithms to learn the optimal policy. However, the environment approximation in MBRL methods carries along several challenges. In tabular methods like R-MAX [4], where the agent optimistically explores the environment and adjust the expected value for each transition, we can guarantee the convergence of the algorithm. When dealing with complex environments, the dynamics model is often implemented using powerful function approximators, such as Neural Networks, that are not guaranteed to converge to the correct probability distribution and the RL agents might not converge to an optimal policy. While in offline RL the fixed dataset might limit the learning of the environment dynamics, leading to an overestimate of the value function [5, 35], in online RL the overestimation of the value function could be caused by the uncertainty of the estimates learned by the agent. In fact, despite being able to visit every state of the domain infinitely often (ideally, in complex domains this is unfeasible), the function approximators might not have enough capacity to learn the true dynamics. This is especially true in MBRL, where we aim to approximate the environment transition and reward functions. Therefore, an optimistic approach might fail due to the approximation errors introduced by the estimated functions and a pessimistic one might lead to a more robust policy: by retaining multiple estimates, we can maximize the expected reward according to the worst one.

The mismatch between the approximated and real-world dynamics distribution might also lead to diverging model dynamics [46] due to errors in the model predictions being propagated and compounding along a trajectory, ending up in a state that hardly resembles any state from the environment. Also, the algorithm could mislead these differences in the distribution to retrieve high rewards [3, 46] when planning in poorly approximated areas of the domain: the algorithm will get higher values only in the simulated environment, while in the real world they will obtain poorer results.

The source-target distribution mismatch is a well-known issue in the RL literature. Recently, ensembles of models have been employed to learn policies more robust to approximation errors. Leveraging the model uncertainty using an ensemble of models reduces dynamics overfitting, leading to a more robust policy [6, 24]. In [33], the agent learns a robust policy by training on the trajectories producing the worst ϵ percentile of returns, generated by an ensemble of source MDPs[3]. Ensembles have been employed also to steer the agent exploration towards more uncertain areas of the

[1] The code is available at: https://github.com/danielefoffano/REAL-MBRL

[2] We will often use the terms "source MDP", "environment" and "real-world" interchangeably.

[3] Note that, in this case, the ensemble is made of source (not target) MDPs.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2022), Cruz, Hayes, da Silva, Santos (eds.), May 9-10, 2022, Online, https://ala2022.github.io/. 2022.

domain. In [37], the agent leverages the models' disagreement to compute exploration policies that each round will prefer to visit unknown areas of the environment, reducing the uncertainty.

Adversarial methods have been used in Deep Learning literature to train robust classifiers [11]. Both in control and RL theory, the robustness of the learned policy can be improved against an adversary. Robust MDPs are a generalisation of the exact MDP framework to a setting where transition probabilities are uncertain. The idea is rooted in stochastic zero-sum games, where a player facing an adversary can compute a maximin strategy to maximize the minimum gain [10, 13]. Using this game-theoretic formulation, [30] uses minimax dynamic programming to solve MDPs with adversarially chosen transitions. Robust Reinforcement Learning (RRL) was introduced in [27], where input disturbance and modelling errors are considered as adversarial perturbation and captured through a Model-Free Actor-Disturber-Critic architecture. This approach has been later extended in [31], using Deep Neural Networks as function approximators. [34] propose a game theoretical framework for MBRL between a policy player and a model player: the former wants to maximize the rewards collected with the learned model, while the latter aims to minimize the prediction error of the model under the improved policy.

Based on the Robust MDPs framework, we propose Robust Ensemble AdversariaL (REAL) MBRL. We map the model uncertainty problem as a two-player game, using an ensemble of models. One player wants to maximize the expected reward while the other wants to minimize it by choosing the worst possible model in the ensemble. This way, the player maximizing the expected reward will have to take into account the adversary actions by solving a maximin optimization problem. This leads to a more robust policy when compared to assuming all transitions of the model to be correct. Additionally, we study the effect of different ϵ-greedy adversaries on the resulting policy. We focus on the following research questions:

- Is the resulting policy more robust to model errors?
- Is the adversary learning meaningful information?
- Is the adversary helping the policy to be more robust to model errors, or are there other factors influencing the final result?

2 BACKGROUND

We represent the environment as a Markov Decision Process [1].

Definition 1. A Markov Decision Process (MDP) M is a 5-tuple $\langle S, A, P, R, \gamma \rangle$ where S is the state space our agent can be into, A is the set of actions our agent can perform in every state, $P : S \times A \times S \rightarrow [0, 1]$ is the transition probability function, $R : S \rightarrow \mathbb{R}$ is the reward function and γ is the discount rate.

When interacting with the environment, at each time step $t = 0 \dots T$ the agent will be in a certain state s_t and it will perform an action a_t: as a consequence, it will retrieve the reward r_t and the new state s_{t+1}. Therefore, we think about the experience of the agent as a sequence of interactions $\{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_T, a_T, r_T)\}$, called trajectory. The agent acts according to a policy $\pi : S \rightarrow A$ which, depending on the current state, will compute which action to perform.

Definition 2. We define the discounted return of the agent as the discounted sum of the rewards obtained in a trajectory $\{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_T, a_T, r_T)\}$ under a policy π

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}. \tag{1}$$

Definition 3. The state-action value function $q_\pi : S \times A \rightarrow \mathbb{R}$ under a policy π is defined as the expected return starting from a state s and taking action a while following the policy π

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a]. \tag{2}$$

Definition 4. The state-value function $v_\pi : S \rightarrow \mathbb{R}$ under a policy π is defined as the expected return starting from a state s and following the policy π

$$v_\pi(s) = E_\pi[G_t | s_t = s]. \tag{3}$$

RL techniques aim to maximize the expected reward retrieved by the agent under a policy π. This can be achieved by computing the optimal policy: the policy under which our agent will obtain the highest cumulative reward.

Definition 5. We define the optimal policy π^* as the policy that maximizes the expected return

$$\pi^* = \arg\max_\pi q_\pi(s, a), \tag{4}$$

for all $s \in S$ and $a \in A$.
Equivalently,

$$\pi^* = \arg\max_\pi v_\pi(s), \tag{5}$$

for all $s \in S$.

Model-Based Reinforcement Learning. Model-Based RL algorithms build a parameterised model M_θ approximating the dynamics of the real environment M: R_θ and P_θ will respectively be the reward and transition probability functions of M_θ. The main advantage of this approach is that the model can be used to plan, without requiring more samples from the real environment (which might be difficult or expensive to get in some-real world scenarios):

Definition 6. We define *planning* as the improvement of the agent behaviour (policy) by using experience sampled from a model of the real environment.

Through an iterative sequence of exploration of the environment and planning with the model fitted to the real experience, MBRL algorithms improve both their model and policy [43]. A general template for a MBRL algorithm is described in Algorithm 1 [46]: the procedure UPDATEMODEL fits the parameterised model to the data retrieved exploring the environment, while UPDATEPOLICY procedure solves the given MDP.

However, model learning is not straightforward: usually, a ground-truth model will not be available to estimate the real dynamics. Thus, the learning agent will have to rely only on samples from the environment. The model-based approach brings new challenges to RL researchers: the most important is the mismatch between source and target dynamics distribution. The algorithms can exploit model inaccuracies (a phenomenon known as model exploitation),

computing a policy that performs incredibly well on the target domain but sub-optimally (or even terribly) in the source domain.

Algorithm 1 Model-Based Reinforcement Learning

```
Input: state sample procedure d
Input: model m
Input: policy π
Input: environment ε
Get initial state s ← ε
for iteration ∈ {1,2,...,K} do
    for interaction ∈ {1,2,...,N} do
        Generate action: a ← π(s)
        Generate reward, next state: r, s' ← ε(a)
        m, d ← UPDATEMODEL(s, a, r, s')
        Update current state: s ← s'
    end for
    for planning step ∈ {1,2,...,L} do
        Generate state, action s̄, ā ← d
        Generate reward, next state r̄, s̄' ← m(s̄, ā)
        π ← UPDATEPOLICY(s̄, ā, r̄, s̄')
    end for
end for
```

Robust Markov Decision Processes. An MDP formulation of a Sequential Decision Making problem is convenient for several reasons. Most importantly, it allows a tractable model for very complex applications. Several algorithms have been proposed to solve an MDP and compute an optimal policy [2, 7, 21, 25, 32, 36, 42, 48]. However, the transition probability function P is unknown in many applications and the algorithms compute the optimal policy relying on a function estimated from noisy samples of the environment dynamics. When generating a trajectory, since the function approximation is not exact, the model can generate transitions that do not exist in the environment leading to *hallucinated* states [44, 46]. In turn, these non-existing states will be used as a starting point to generate the subsequent transitions, compounding and misguiding the whole trajectory.

Robust Markov Decision Processes (RMDPs) address the dynamics uncertainty by following a robust optimization approach. The uncertainty in the model transition probability P is considered as an adversarial selection from a convex set $\mathbb{P}$, called *uncertainty set*. The agent aims to maximize the worst-case expected reward, based on the adversarial choice of $P \in \mathbb{P}$. Following this game-theoretical approach the optimization problem in (5) can be redefined as

$$\pi^* = \arg\max_{\pi \in \Pi} \min_{P \in \mathbb{P}} v_{\pi,P}(s), \tag{6}$$

where $v_{\pi,P}(s)$ is the state-value function under policy π and following transitions as specified by P. The set of stationary Markovian Policies Π contains an optimal policy π^*. Uncertainty can be defined in different ways, two examples from the literature are *(s,a)-rectangular* and *s-rectangular* uncertainty sets.

Definition 7. We define (s,a)-rectangular uncertainty set as the Cartesian product of independent subsets $\mathbb{P}_{(s,a)} \subseteq \mathbb{R}_+^{|S|}$ for each $(s,a) \in S \times A$

$$\mathbb{P} = \bigtimes_{(s,a) \in S \times A} \mathbb{P}_{(s,a)}, \tag{7}$$

where $P(\cdot|s,a) \in \mathbb{P}_{(s,a)}$.

For (s,a)-rectangular uncertainty sets, an optimal robust policy can be computed with value iteration and it exists an optimal policy that is Markovian, stationary and deterministic [13, 30].

(s,a)-rectangular sets can be generalised to s-rectangular uncertainty sets [9, 49], where the adversarial agent is able to see only the state.

Definition 8. We define s-rectangular uncertainty set as the Cartesian product of independent subsets $\mathbb{P}_s \subseteq \mathbb{R}_+^{|S|}$ for each $s \in S$

$$\mathbb{P} = \bigtimes_{s \in S} \mathbb{P}_s \tag{8}$$

With s-rectangular uncertainty sets, a robust optimal policy can be computed and it exists an optimal policy that is Markovian and stationary, but that is not guaranteed to be deterministic [49].

3 RELATED WORK

The prediction accuracy of the model is of critical importance in MBRL algorithms since planning relies on it. Using a model to generate new data, through which it will be possible to plan further without interacting with the environment, can be critical for applications where there is limited data availability [23].

In the literature, we can observe two kinds of models that will approximate the environment dynamics: non-parametric and parametric. The non-parametric approach has seen successful applications in low-dimensional environments with Gaussian Processes (GP) [8, 12, 18–20, 29]. However, these models make assumptions on the underlying system distribution that will reduce the accuracy in complex domains. On the other hand, the parametric approach has seen a wild growth in popularity in recent years thanks to the constant improvements of functions approximators, such as Neural Networks (NN) [6, 14, 15, 28, 45]. Parametric models have more flexibility, allowing them to represent more complex functions, while non-parametric models are more efficient when few data samples are available. When using powerful approximators like Neural Networks, however, there is no guarantee that the learned transition function will converge to an optimal solution. Often, MBRL algorithms must settle for a sub-optimal representation of the environment dynamics, but a poor model may lead to faults in planning or an overestimate of the expected reward that will, in turn, compromise the accuracy of the policy computed by the algorithm [3, 14, 24]. Algorithms like R-MAX [4] overcome this problem by leveraging a tabular representation of the environment transitions and optimistically exploring the state-action space, estimating the probabilities similarly to Monte-Carlo methods. However, tabular representations do not scale well to complex domains and it would be unfeasible to use them in most real-world applications.

A first approach to improve the model accuracy is to choose the appropriate exploration strategy: if the model lacks data to compute a reliable estimate of the real dynamics, the algorithm will take advantage of it during planning, converging to a policy that is optimal for the model but inadequate to act in the environment. [37] introduce Model-Based Active eXploration (MAX), an active

exploration algorithm that leverages on the models' ensemble disagreement (i.e., uncertainty) to compute exploration policies that each round will visit unknown areas of the environment, solving the disagreement.

Ensembles of models have been widely used in the recent literature to improve the approximation of the dynamics in MBRL methods. In [6], the authors propose PETS, a MBRL architecture combining particle-based propagation and ensembles of parametric models to take into account the *aleatoric* (the variance of the data itself) and *epistemic* uncertainty (uncertainty of the dynamics model) of the learned dynamics. [24] use model ensembles to prevent overfitting the model during planning: during policy validation, the algorithm leverages the different models to evaluate the outcome of the policy on a diversified set of futures.

Other approaches, closer to our method, develop a game-theoretical framework to compute more robust policies. In [41], the authors propose a pessimism principle for offline RL, to maximize the expected reward according to the worst estimate of the value function. In [33], the authors propose EPOpt-ϵ, an algorithm combining model ensembles and adversarial training to learn policies robust to model errors. In EPOpt-ϵ, the model approximates a distribution over the true environment parameters (e.g., mass, ground friction, joint damping). This distribution is then used to sample multiple instances of the environment (i.e., the ensemble): the policy will be improved over the worst ϵ percentile of the rollouts performed using the ensemble. While in this work the EPOpt-ϵ agent acts pessimistically w.r.t. the distribution on the parameters that regulate the environment physics, our REAL agent acts pessimistically w.r.t. the estimate of the transition probability function. In [34], the authors propose a Model-Based RL formulation of a two-player game. More specifically, they cast the problem as a Stackelberg game [47]: an asymmetric game where a specific order of the players is imposed. In this approach, the policy player wants to maximize the expected reward within the current approximated model, while the model player wants to minimize the prediction error under the state distribution induced by the policy. A Model-Free approach is followed by [31], where policy learning is formulated as a zero-sum game between a protagonist player and an adversary: the adversary corrupts the transitions experienced by the protagonist, trying to minimize the expected reward, while the protagonist aims to maximize the expected reward (as usual in RL settings) by learning a policy robust to the adversarial inputs. Compared to the previous literature, we also cast the MBRL problem as a two-player game, following the Robust Markov Decision Processes (RMDPs) Framework. Unlike previous methods, we leverage an actual second player that, through its policy, adversarially picks the transitions at each timestep t by choosing the model of the ensemble that will carry out the transition.

4 METHOD

In this section we outline our method, Robust Ensemble AdversariaL (REAL) MBRL.

We approximate the environment dynamics with an ensemble M_ψ composed of N models (deep neural networks), thus $M_\psi = \{M_{\psi_1}, \dots, M_{\psi_N}\}$, parameterised by $\psi = \{\psi_1, \dots, \psi_N\}$. We define a (s,a)-rectangular uncertainty set $\mathbb{M}$ on the ensemble models such that

$$\mathbb{M} = \bigtimes_{(s,a)\in S\times A} \mathbb{M}_{(s,a)},$$

where $\mathbb{M}_{(s,a)} \subseteq \mathbb{R}_+^N$.

We then consider two players:

- Player 1, following a policy π, whose objective is to maximize the expected return of the actions taken when interacting with the environment

$$\max_\pi v_\pi(s), \qquad \forall s \in S,$$

where $v_\pi(s)$ is defined as in 4.

- Player 2, whose objective is to minimize the expected return for Player 1 by choosing the worst model to perform each transition

$$\min_{M\in\mathbb{M}} v_{\pi,M}(s), \qquad \forall s \in S,$$

where $v_{\pi,M}(s)$ is equivalent to $v_\pi(s)$ but the transitions are carried out following the dynamics of model M. By defining the policy for Player 2 as $\xi : S \times A \to \mathbb{M}$, we obtain the equivalent formulation

$$\min_{\xi_\omega} v_{\pi,\xi}(s), \qquad \forall s \in S,$$

where

$$v_{\pi,\xi}(s) = R(s) + \gamma \sum_{s'\in S} P_{\pi,\xi}(s'|s) V_{\pi,\xi}(s')$$

and

$$P_{\pi,\xi}(s'|s) = \sum_{a,M} P_{a,M}(s'|s,a) p_{\pi,s}(a) p_{\xi,(s,a)}(M).$$

$p_{\pi,s}(a)$ and $p_{\xi,(s,a)}(M)$ are the probabilities of taking actions a and M under policies π and ξ.

This leads to a two-player zero-sum game, where the adversary (Player 2) selects the worst possible model in the ensemble. Player 1 can find an optimal policy by solving the following maximin problem:

$$\pi^*(s) = \arg\max_\pi \min_\xi v_{\pi,\xi}(s) \qquad \forall s \in S. \tag{9}$$

In practice, to solve the maximin problem, we implemented the policies of both players as Deep Neural Networks $\pi_\theta : S \to A$ and $\xi_\omega : S \times A \to \{1, \dots, N\}$ parameterised by θ and ω. Our algorithm iteratively repeats two steps in a Dyna-style fashion:

(1) Fitting the models in the ensemble to the samples collected from the environment using policy π_θ, using the MSE loss function.
(2) Improving the policies of Player 1 and 2.

The pseudocode is provided in Algorithm 2. The algorithm can easily be adapted to work with s-rectangular uncertainty sets by making the adversarial policy dependent only on the state s (i.e., $\xi : S \to \mathbb{M}$).

We also study the effects of an ϵ-greedy adversary on the optimal policy π_θ^*, to encourage the exploration of more adversarial actions. The ϵ-greedy adversary will select a random action (i.e., a random model index) to carry out the transition with probability ϵ. With probability $(1-\epsilon)$ the action will be chosen according to the adversarial policy ξ_ω. The pseudocode for the ϵ-greedy adversarial policy

Algorithm 2 Robust Ensemble AdversariaL (REAL) MBRL - General sketch

Input: Empty dataset buffer B
Input: Random policy π_θ
Input: Initialized model parameters ψ_i
for ensemble M_ψ=$\{M_{\psi_1}, \ldots, M_{\psi_N}\}$
Input: Initialized adversary parameters ω for adversary ξ_ω
Input: number of iterations K

for k $\in \{0, ..., K\}$ **do**
    ◇ Collect new observations from the environment
    $B \leftarrow$ COLLECT(π_θ)
    ◇ Update models using the gathered samples
    $\psi_i \leftarrow$ TRAIN_MODEL(M_{ψ_i}, B)
    ◇ Update the main and adversarial policies
    $\pi_\theta, \xi_\omega \leftarrow$ IMPROVE$(\pi_\theta, \xi_\omega, M_\psi)$
end for

is given in Algorithm 3. Both the REAL and the vanilla (without adversary) ensemble MBRL algorithm are specific instances of the ϵ-greedy REAL, respectively with $\epsilon = 0$ and $\epsilon = 1$.

Algorithm 3 ϵ-greedy Adversarial policy $\xi_{\epsilon,\omega}$

Input: ϵ s.t.: $0 \leq \epsilon \leq 1$
Input: Adversary policy ξ_ω
Input: Current state s
Input: Action a taken by Player 1
Input: Size of ensemble N

Function $\xi_{\epsilon,\omega}$:
    ◇ Sample from uniform distribution
    $p \leftarrow$ Unif$(0, 1)$
    ◇ Apply ϵ-greedy policy
    if $p < \epsilon$:
        return Random$(0, N-1)$
    else:
        return $\xi_\omega(s, a)$

5 EXPERIMENTS AND RESULTS

In this section we empirically evaluate our method and discuss the obtained results. Through our experiments, we want to show that, by playing against an adversary choosing adversarial transitions, the main player can compute a policy that is more robust to model errors. We evaluate the behaviour of our agent on three environments of the OpenAI Gym suite: Frozen Lake, Cartpole and Pendulum. Frozen Lake is a grid-world environment where the agent has to reach the goal tile from a starting point without falling in holes spread across the field. In Cartpole, a pole is attached to a joint on a cart: the agent must prevent the pole from falling down by moving the cart along the track. Finally, in the Pendulum environment the goal of the agent is to learn how to keep a pendulum upright by applying the right amount of force on the joint. As a baseline, we use the "vanilla Model-Based" version of our algorithm (i.e., without the adversary). For the Frozen Lake environment we used a probabilistic model, while for the Cartpole and Pendulum environments the model was deterministic. As a baseline we used a version of Algorithm 2 without the adversary, where the model transitions are carried out using the average of the ensemble outputs (i.e., $s' = \frac{1}{N}\sum_{i=1}^{N} M_{\psi_i}(s, a)$).

5.1 Frozen Lake

The Frozen Lake environment has discrete state and action spaces: we chose to work on a 4x4 grid world, and the actions are Up, Down, Right and Left. An illustration of the environment is provided in Figure 1. The low dimensionality of the state-action space allows us to examine more in detail the behaviour of our agent and to zoom-in on the actual contribution of the adversary. This is why we employ Q-Learning to improve the policies of the main player and of the adversary: a tabular method gives us the chance to easily visualise the values of each state-action pair and interpret the decisions taken by the two players. First, we focus on what the model and the adversary are learning. Figures 3a, 3b and 3c show the transition probabilities predicted by the ensemble models when the agent is in state 7 and takes action "Down", after gathering 900 samples from the environment. By comparing them with the true probability, represented in Figure 3d, we can see that they are far from correctly representing the correct transition. When examining the Q-values learning from the adversary (Figure 3e), we observe that at this stage it has learned that choosing model 0 leads to the worst outcome for the main player. This choice can be intuitively explained by looking at the single transition probabilities: model 0 is the one with the highest probability to send the agent into one of the failure terminal states (hole on the ground on cell 6).

When examining the average return, we can see that all the instances of REAL behave quite similarly. This is expected, since the environment dynamics can be easily learned, with the models of the ensemble quickly converging to a good approximation. The results are reported in Figure 2. A summary of the algorithm hyperparameters is provided in Appendix A, in Table 1.

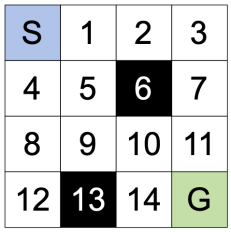


Figure 1: An illustration of the 4x4 map used in our experiments with the Frozen Lake environment. The S represents the starting point, while the G is the goal state. The black tiles represent holes in the gridworld which will determine the failure of the task for the agent. Tiles that are not black are "frozen" tiles, on which the agent can safely step.

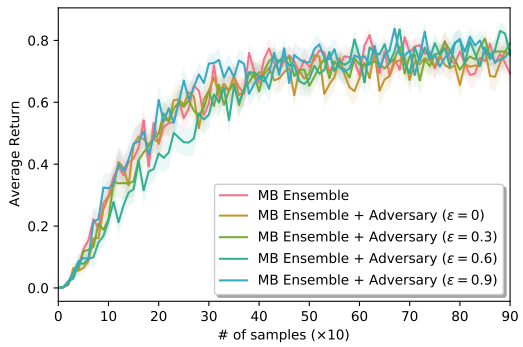
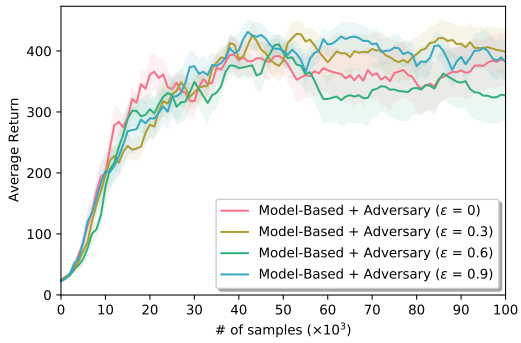


Figure 2: Frozen Lake environment. Comparison of REAL instances with $\epsilon \in \{0, 0.3, 0.6, 0.9\}$. Bold lines represent the average return over 10 runs, shaded areas evidence the performance for one standard error over and below the average.

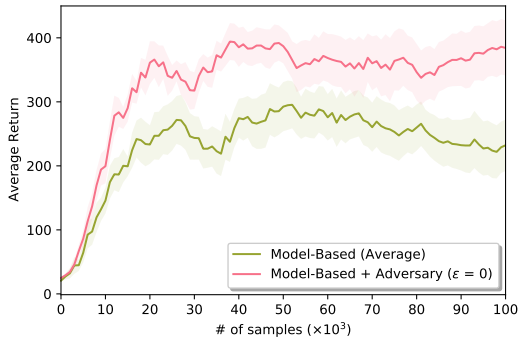
5.2 Cartpole

With the Cartpole environment, we want to test how our agent performs with continuous observations. The action space is still discrete, the agent can move the cart in the left or right direction. Since the state space is continuous, we cannot rely on tabular planning methods: we improve the two players' policies through Proximal Policy Optimization (PPO), a state-of-the-art policy gradient algorithm. The algorithm is based on an actor-critic architecture to compute an optimal policy: the critic (i.e., the value function) evaluates the decisions made by the actor (i.e., the policy), which in turn leverages this feedback to improve its behaviour. The adversary agent is also trained using a policy gradient method called REINFORCE, which computes Monte-Carlo estimates of the expected return to update the policy parameters. While state-of-the-art implementations of PPO rely on multiple actors, collecting independent samples from multiple instances of the environment, we implemented a simpler version of PPO, using just one actor. This way, we reduced the number of hyperparameters to fine-tune while still being able to achieve an optimal behaviour. When performing rollouts with the ensemble of models, we use a fixed horizon length L. This way, we reduce the impact of the compounding errors when planning, but we also decrease the amount of states visited by the actor. To overcome this limitation, with probability δ the rollout will start from a state sampled from the buffer, and with probability $1 - \delta$ from the environment starting state. This way, we can gather samples from different areas of the domain, converging faster to the optimal policy.

Results are presented in Figure 4a and a detailed illustration of the algorithm hyperparameters is available in Appendix A, in Table 2. In figure 4b we compare REAL with $\epsilon = 0$ to the Model-Based baseline since it outperforms the other instances in the first few iterations (and then converges to more or less the same value). We can see that the adversarially trained agent heavily outperform the ensemble-based algorithm, which results to be more vulnerable to the models approximation mistakes.



(a)

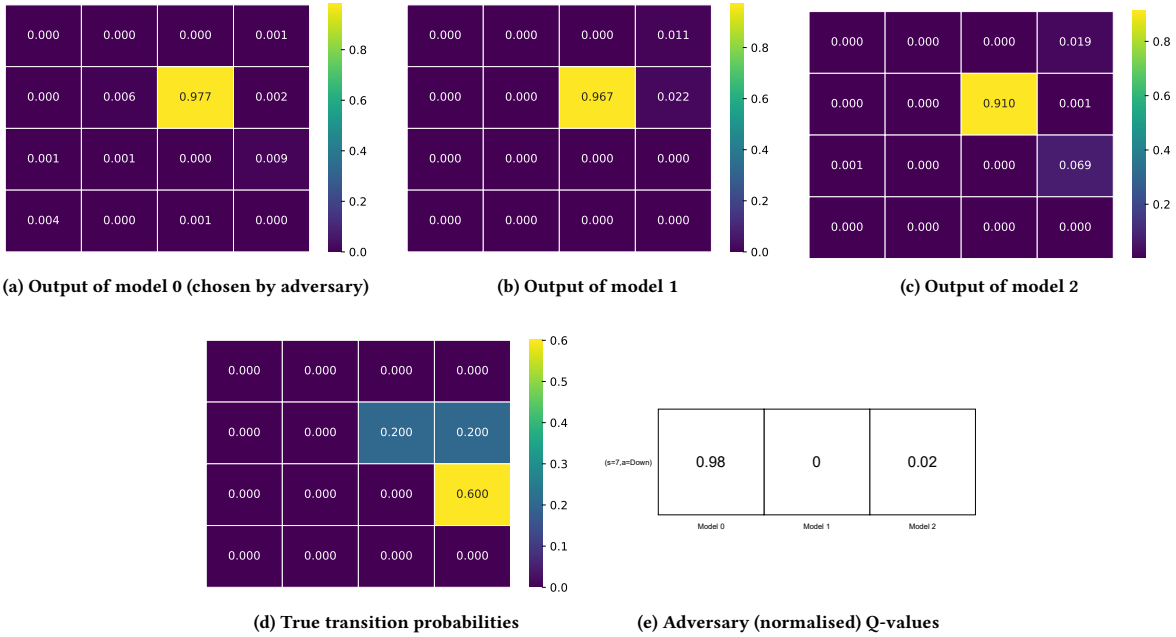


(b)

Figure 4: Cartpole environment. Comparison of REAL instances with $\epsilon \in \{0, 0.3, 0.6, 0.9\}$ (figure a) and comparison between REAL with $\epsilon = 0$ and the Model-Based baseline (figure b).

5.3 Pendulum

With the Pendulum environment, we extend REAL to environments with continuous action spaces. As we did for the Cartpole environment, we use PPO to improve the policies since it would be unfeasible to use tabular planning methods. To evaluate the robustness to model errors of our algorithm, we analyse the performance of different instances of our agent and compare it with the plain Model-Based approach, where the output of the model ensemble M_ψ is the average of the outputs of each component M_{ψ_n}. We consider instances of the REAL agent with an ϵ random adversary where $\epsilon \in \{0.3, 0.6, 0.9\}$. In figure 5a, we can see the average performance of each agent over 10 different runs. Despite behaving all very similarly, we observe that the agent playing against an adversary with $\epsilon = 0$ has an improved performance in the early stages of the training process, later converging to the same performance as the other agents. In figure 5b we compare the $\epsilon = 0$ REAL agent with the Model-Based baseline. We can see that the adversarial choice of the model enables for slightly more efficient early planning. A summary of the algorithm hyperparameters is provided in Appendix A, in Table 2.



(a) Output of model 0 (chosen by adversary)

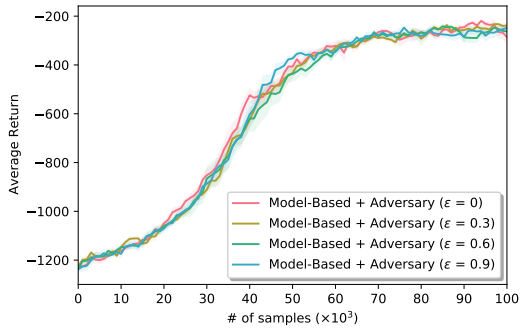
(b) Output of model 1

(c) Output of model 2

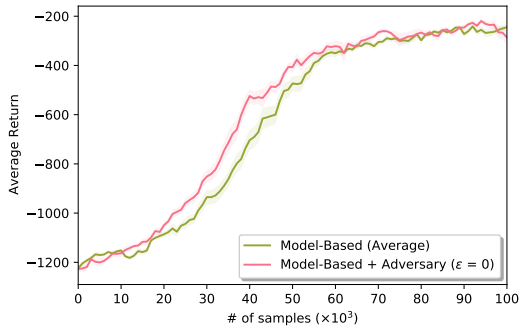
(d) True transition probabilities

(e) Adversary (normalised) Q-values

Figure 3: True and predicted distributions $p(s'|s = 7, a =$ "Down"$)$ according to each model in the ensemble and Q-values for the adversarial agent, after collecting 900 samples from the environment. Note that environment state enumeration starts from 0.



(a)



(b)

Figure 5: Comparison of the instances of REAL with $\epsilon \in \{0.3, 0.6, 0.9\}$ in the Pendulum environment (figure a). We can observe that for $\epsilon = 0$, the agent learns more efficiently at first and we compare it to the Model-Based baseline in figure b.

6 CONCLUSIONS AND FUTURE WORK

With our work we present Robust Ensemble AdversariaL (REAL) MBRL, an ensemble-based algorithm leveraging the use of an Adversarial agent to compute a policy more robust to model errors. We propose two adversarial approaches: one with a greedy adversary, always exploiting what it has learned, and one with an ϵ-random adversary, exploring other actions with probability ϵ.

When using powerful function approximators to estimate the environment dynamics, we are not guaranteed that the learned transition probabilities will converge to the correct distribution and therefore we cannot guarantee that the policy improvement will converge to an optimal solution either. Through the Robust Markov Decision Processes (RMDPs) framework, we cast the Model-Based Reinforcement Learning problem as a two-player game where the adversary can pick which model in the ensemble will carry out the transition at time t. The optimal policy will be the solution to the resulting maximin optimization problem.

With our experiments, we empirically show that:

- The policy learned by the agent is more robust to the model errors and can achieve a better return than the "single-player" ensemble-based MBRL approach,
- The adversary is learning meaningful information about which model can better interfere with the main player objective,
- The improvement over the single-player baseline depends only on the presence of the adversary,
- Our approach can scale to continuous state-action spaces.

For future work, it would be interesting to study the performance of our algorithm on more complex environments that can increase

the challenges for both the main and adversary player. Another research direction would be studying ways to ensure that the networks in the ensemble of models have meaningful differences in what they learn, to grant the adversarial agent more opportunities to interfere with the learning process of the main player. Finally, since we used deterministic models for the Cartpole and Pendulum environments, future work could involve extending the algorithm with probabilistic models (such as Gaussian MLPs).

REFERENCES

[1] John Hugh Andreae. 1966. *Learning machines: a unified view.* Standard Telecommunications Laboratories.

[2] D Berteskas. 1987. Dynamic programming: deterministic and stochastic models.

[3] Rinu Boney, Norman Di Palo, Mathias Berglund, Alexander Ilin, Juho Kannala, Antti Rasmus, and Harri Valpola. 2019. Regularizing trajectory optimization with denoising autoencoders. In *Advances in Neural Information Processing Systems*. 2859–2869.

[4] Ronen I Brafman and Moshe Tennenholtz. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3, Oct (2002), 213–231.

[5] Jacob Buckman, Carles Gelada, and Marc G Bellemare. 2020. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint arXiv:2009.06799* (2020).

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

[7] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

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

[9] Larry G Epstein and Martin Schneider. 2003. Recursive multiple-priors. *Journal of Economic Theory* 113, 1 (2003), 1–31.

[10] Robert Givan, Sonia Leach, and Thomas Dean. 2000. Bounded-parameter Markov decision processes. *Artificial Intelligence* 122, 1-2 (2000), 71–109.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[12] Alexandra Grancharova, Juš Kocijan, and Tor A Johansen. 2008. Explicit stochastic predictive control of combustion plants based on Gaussian process models. *Automatica* 44, 6 (2008), 1621–1631.

[13] Garud N Iyengar. 2005. Robust dynamic programming. *Mathematics of Operations Research* 30, 2 (2005), 257–280.

[14] Taher Jafferjee, Ehsan Imani, Erin Talvitie, Martha White, and Micheal Bowling. 2020. Hallucinating Value: A Pitfall of Dyna-style Planning with Imperfect Environment Models. *arXiv preprint arXiv:2006.04363* (2020).

[15] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. 2019. Model-based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374* (2019).

[16] Sham Machandranath Kakade. 2003. *On the sample complexity of reinforcement learning.* University of London, University College London (United Kingdom).

[17] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. 2018. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*. PMLR, 651–673.

[18] Sanket Kamthe and Marc Deisenroth. 2018. Data-efficient reinforcement learning with probabilistic model predictive control. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1701–1710.

[19] Jonathan Ko, Daniel J Klein, Dieter Fox, and Dirk Haehnel. 2007. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proceedings 2007 ieee international conference on robotics and automation*. IEEE, 742–747.

[20] Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. 2004. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, Vol. 3. IEEE, 2214–2219.

[21] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. 2006. Improved Monte-Carlo search. *Univ. Tartu, Estonia, Tech. Rep* 1 (2006).

[22] Dmytro Korenkevych, A Rupam Mahmood, Gautham Vasan, and James Bergstra. 2019. Autoregressive policies for continuous control deep reinforcement learning. *arXiv preprint arXiv:1903.11524* (2019).

[23] Vikash Kumar, Emanuel Todorov, and Sergey Levine. 2016. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 378–383.

[24] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. 2018. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592* (2018).

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[26] 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. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[27] Jun Morimoto and Kenji Doya. 2005. Robust reinforcement learning. *Neural computation* 17, 2 (2005), 335–359.

[28] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7559–7566.

[29] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. 2008. Local Gaussian process regression for real time online model learning. *Advances in neural information processing systems* 21 (2008), 1193–1200.

[30] Arnab Nilim and Laurent El Ghaoui. 2005. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research* 53, 5 (2005), 780–798.

[31] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. 2017. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2817–2826.

[32] Martin L Puterman and Moon Chirl Shin. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24, 11 (1978), 1127–1137.

[33] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. 2016. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283* (2016).

[34] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. 2020. A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*. PMLR, 7953–7963.

[35] Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. 2021. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *arXiv preprint arXiv:2103.12021* (2021).

[36] Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

[37] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. 2019. Model-based active exploration. In *International Conference on Machine Learning*. PMLR, 5779–5788.

[38] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[41] Jordi Smit, Canmanie T Ponnambalam, Matthijs TJ Spaan, and Frans A Oliehoek. 2021. PEBL: Pessimistic Ensembles for Offline Deep Reinforcement Learning. In *Robust and Reliable Autonomy in the Wild Workshop at the 30th International Joint Conference of Artificial Intelligence*.

[42] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.

[43] Richard S Sutton. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*. Elsevier, 216–224.

[44] Erik Talvitie. 2014. Model Regularization for Stable Sample Rollouts.. In *UAI*. 780–789.

[45] Elise van der Pol, Thomas Kipf, Frans A Oliehoek, and Max Welling. 2020. Plannable approximations to MDP homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963* (2020).

[46] Hado P Van Hasselt, Matteo Hessel, and John Aslanides. 2019. When to use parametric models in reinforcement learning?. In *Advances in Neural Information Processing Systems*. 14322–14333.

[47] Heinrich Von Stackelberg. 2010. *Market structure and equilibrium*. Springer Science & Business Media.

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

[49] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. 2013. Robust Markov decision processes. *Mathematics of Operations Research* 38, 1 (2013), 153–183.

Appendix A

A.1 Algorithm hyperparameters

For completeness, we include the hyperparameters used in the experiments presented in section 5.

Parameter	**Value**
Buffer size $\|B\|$	1000
# environment samples per iteration	100
# model samples per iteration	5000
Model rollout length L	$\min(100, \text{TF})$
Discount factor γ	0.99
Ensemble size N	3
Learning rate α	0.1
Exploration probability ϵ_π	0.1
Models layers	1x16
Starting state probability δ	0.5

Table 1: Algorithm hyperparameters for the Frozen Lake environment. TF stands for Termination Function.

Parameter	Value	
	Cartpole	Pendulum
Buffer size $\|B\|$	1500	
# environment samples per iteration	1000	
# model samples per iteration	20000	
Model rollout length L	$\min(300, \text{TF})$	$\min(200, \text{TF})$
Discount factor γ	0.99	
Ensemble size N	3	
Policy layers	3x32	
Models layers	3x256	
Reward net. layers	1x64	
Critic layers	2x128	
Adversary layers	2x32	
Starting state probability δ	0.5	

Table 2: Algorithm hyperparameters for the Cartpole and Pendulum environments. TF stands for Termination Function.