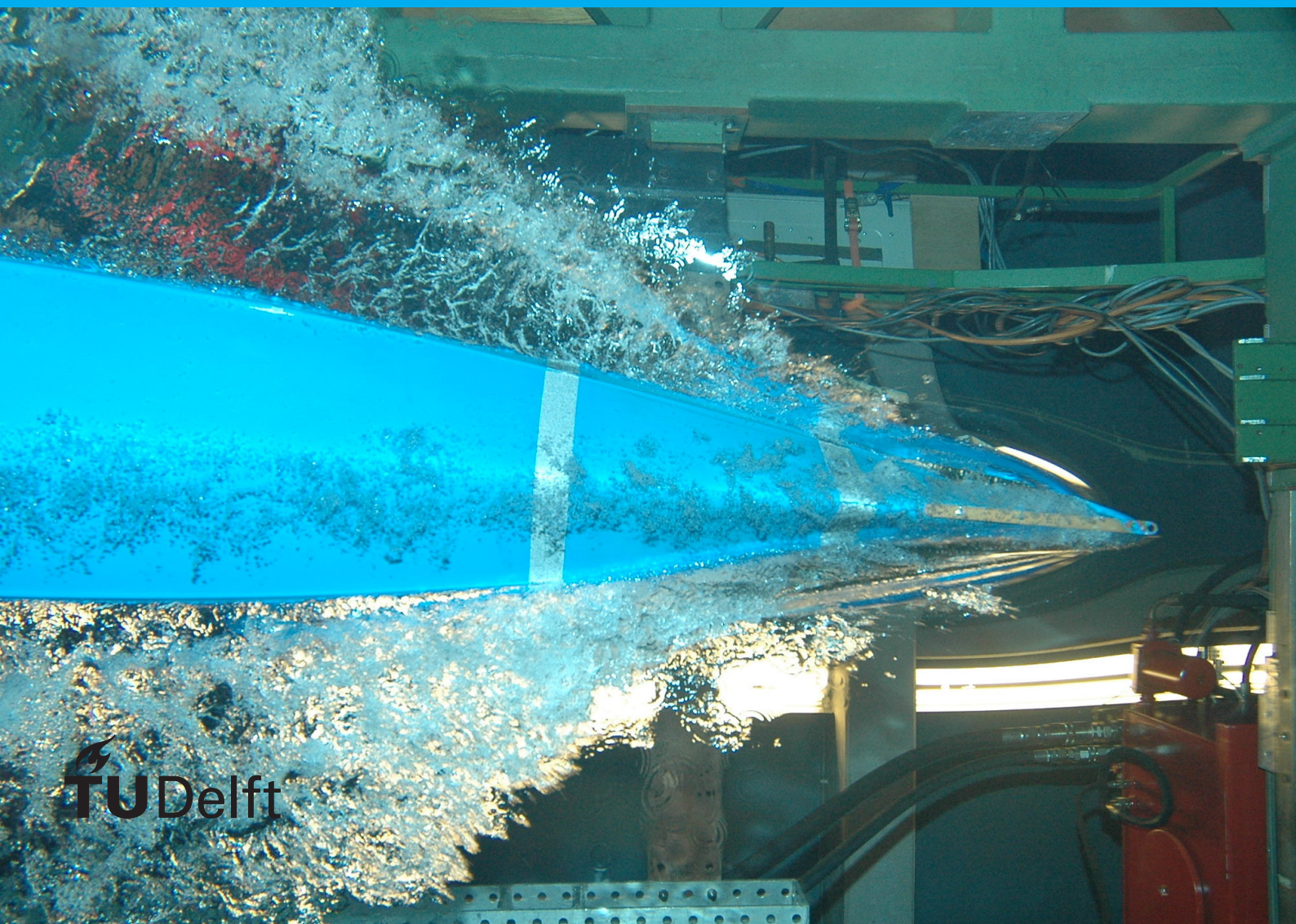
REAL Reinforcement Learning

Planning with adversarial models

Daniele Foffano



TU Delft

REAL Reinforcement Learning

Planning with adversarial models

by

Daniele Foffano

to obtain the degree of Master of Science in Computer Science, with a specialization in Data Science and Technology at the EEMCS faculty of Delft University of Technology,
to be defended publicly on Friday January 14th, 2022 at 10:00 AM.

Student number:	5040140	
Project duration:	March 26, 2021 – January 14, 2022	
Thesis committee:	Dr. F. A. Oliehoek,	TU Delft, supervisor
	Dr. J. C. van Gemert,	TU Delft, committee member
	J. He,	TU Delft, daily supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report presents the work of my Master's thesis project on Robust Ensemble Adversarial Model-Based Reinforcement. The research work was conducted at the Interactive Intelligence Group at TU Delft, under the supervision of Dr. Frans A. Oliehoek and Jinke He (my daily supervisor).

First, I would like to thank Jinke for his guidance and availability. He has always been there when I was in need of new inputs: without our weekly meetings and his knowledge, this project would not have been possible. I am also grateful to Frans for sharing his knowledge on the vast field of Reinforcement Learning, and for all the essential questions and feedback provided during our meetings. Finally, I am grateful to Dr. Jan van Gemert for agreeing to be part of the thesis committee and for asking insightful questions when I presented him my thesis project.

Last but not least, I would like to thank my family, my girlfriend and my dearest Incapriati friends for the support provided during these challenging and stimulating years at TU Delft.

Daniele Foffano
Delft, January 2022

Contents

1

Scientific Article

Robust Ensemble Adversarial Model-Based Reinforcement Learning

Daniele Foffano
Department of Intelligent Systems
TU Delft
D.Foffano@student.tudelft.nl

Jinke He
Department of Intelligent Systems
TU Delft
J.He-4@tudelft.nl

Frans A. Oliehoek
Department of Intelligent Systems
TU Delft
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 Netoworks) 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 experimentally show that finding a maximin strategy for this game results in a policy robust to model errors leading to better performance when compared to assuming the learned dynamics to be correct.

KEYWORDS

Reinforcement Learning, Robust MDPs

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[1], 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[2]. Ensembles have been employed also to steer the agent exploration towards more uncertain areas of the 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

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

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

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 expected 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 $s \leftarrow \varepsilon$
for iteration $\in$ {1,2,...,K} **do**
  for interaction $\in$ {1,2,...,N} **do**
    Generate action: $a \leftarrow \pi(s)$
    Generate reward, next state: $r, s' \leftarrow \varepsilon(a)$
    $m, d \leftarrow$ UPDATEMODEL(s, a, r, s')
    Update current state: $s \leftarrow s'$
  end for
  for planning step $\in$ {1,2,...,P} **do**
    Generate state, action $\bar{s}, \bar{a} \leftarrow d$
    Generate reward, next state $\bar{r}, \bar{s}' \leftarrow m(\bar{s}, \bar{a})$
    $\pi \leftarrow$ UPDATEPOLICY($\bar{s}, \bar{a}, \bar{r}, \bar{s}'$)
  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 $\mathbb{P}_s \subseteq \mathbb{R}_+^{|S| \times |A|}$ 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 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 $M_\psi = \{M_{\psi_1}, \ldots, M_{\psi_N}\}$, parameterised by $\psi = \{\psi_1, \ldots, \psi_N\}$. 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 $\xi_\omega : S \times A \to \{1, \ldots, N\}$ 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., $\xi : S \to \mathbb{M}$).

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_i 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}, D)
    ◇ Update the main and adversarial policies
    $\pi_\theta, \xi_\omega \leftarrow$ IMPROVE$(\pi_\theta, \xi_\omega, M_\psi)$
end for

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 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 $\epsilon = 1$.

4 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

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 $\xi_\omega(s, a)$

Parameter	**Value**
Buffer size $\|B\|$	1000
# environment samples	100
# model samples	5000
Model rollout length L	min(100, 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 K	1000	
# model samples	20000	
Model rollout length L	min(300, TF)	min(200, 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.

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.

4.1 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 2 leads to the worst outcome for the main player. This choice can be intuitively explained by looking at the single transition probabilities: despite all three models predicting with a high probability that the agent would fall into a hole (worst possible outcome), model 2 is the only one that does not give the agent an opportunity to get closer to the goal tile.

When examining the average return, we can see that all the instances of REAL behave quite similarly, and they all except $\epsilon = 0.9$ improve on the baseline in early iterations, converging faster to the optimal policy. When $\epsilon = 0.9$, our agent behaves more similarly to the baseline. This is expected, since a (almost) random choice of the models in the ensemble should converge in expectation to the average of the models (which is the baseline). The results are reported in Figure 2. In Figure 2b, we compare the $\epsilon = 0.3$ agent to the plain Model-Based baseline, since it has a more stable learning curve when compared to the others. We can also observe that the learning curves contains several spikes, despite having a low variance as if the agent was forgetting the information learned in the past: increasing the number of samples collected to improve the policy might help mitigating this problem. A summary of the algorithm hyperparameters is provided in Table 1.

4.2 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

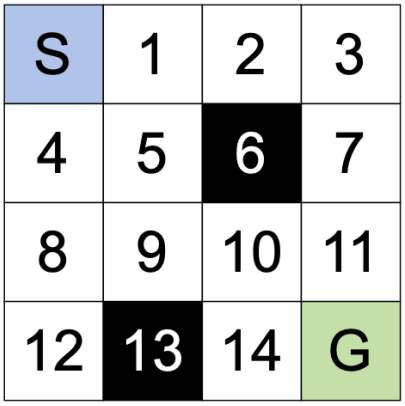
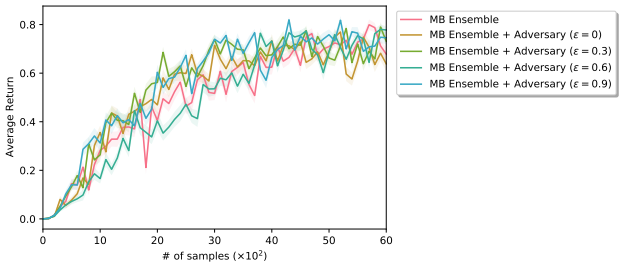
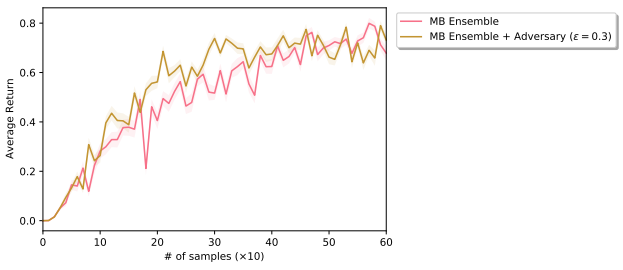


Figure 1: An illustration of the 4x4 map used in our experiments with the Frozen Lake environment. The S represents the starting point, while the G is the goal state. The black tiles represent holes in the gridworld which will determine the failure of the task for the agent. Tiles that are not black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

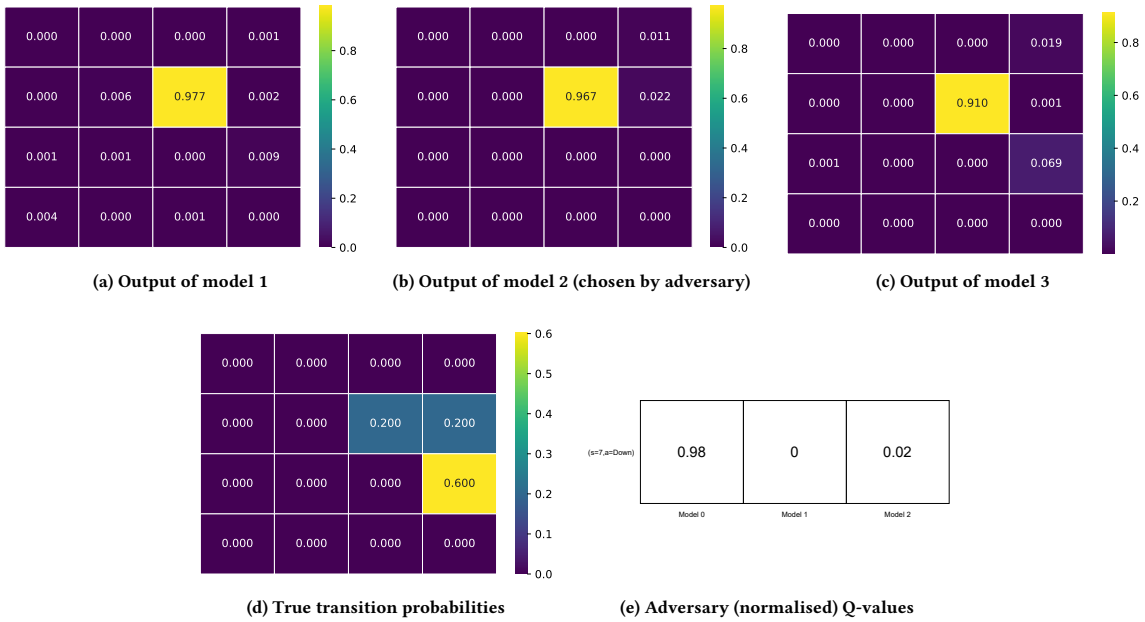


(a)



(b)

Figure 2: Frozen Lake environment. Comparison of REAL instances with $\epsilon \in \{0, 0.3, 0.6, 0.9\}$ (figure a) and comparison between REAL with $\epsilon = 0.3$ and the Model-Based baseline (figure b). Bold lines represent the average return over 10 runs, shaded areas evidence the performance for one standard deviation over and below the average.



(a) Output of model 1 **(b) Output of model 2 (chosen by adversary)** **(c) Output of model 3**

(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 state enumeration starts from 0.

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 Table 2. 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., $s' = \frac{1}{N}\sum_{i=1}^{N} M_{\psi_i}(s, a)$). 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.

4.3 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.

5 RELATED WORK

The prediction accuracy of the model is of critical importance in MBRL algorithms since planning relies on it. Using a model to

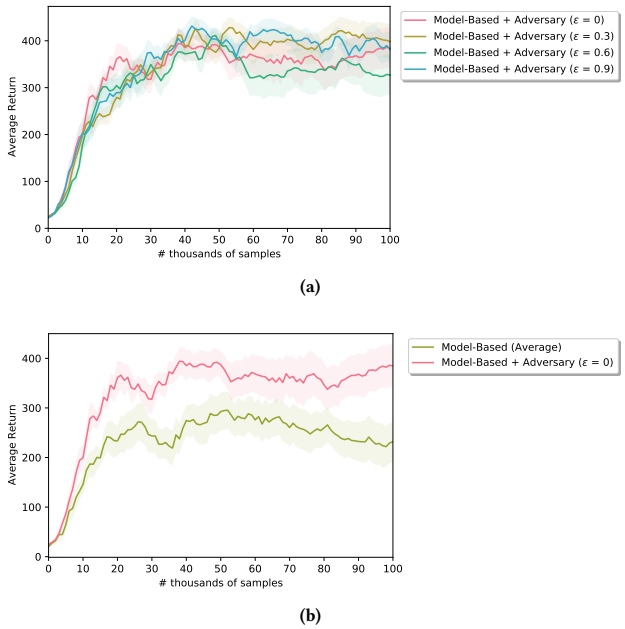


Figure 4: Cartpole environment. Comparison of REAL instances with $\epsilon \in \{0, 0.3, 0.6, 0.9\}$ (figure a) and comparison between REAL with $\epsilon = 0$ and the Model-Based baseline (figure b).

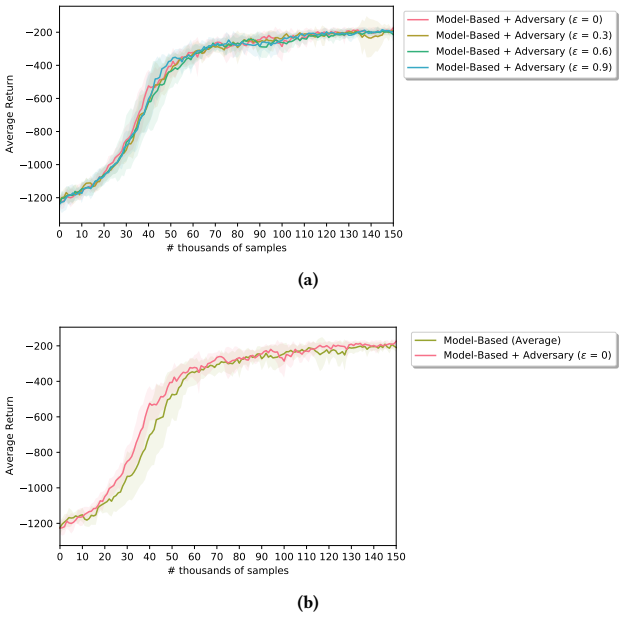


Figure 5: Comparison of the instances of REAL with $\epsilon \in \{0.3, 0.6, 0.9\}$ in the Pendulum environment (figure a). We can observe that for $\epsilon = 0$, the agent learns more efficiently at first and we compare it to the Model-Based baseline in figure b.

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 mitigate the effects of the model inaccuracy, reducing the impact of compounding errors on the policy improvement. In fact, because of the recursive nature of MBRL approaches, a single prediction error will generate another slightly bigger error at

the next step and so on, leading to a failure in the learning process. [44] proposes a regularization technique to make the model more robust to compounding errors. This method, called *hallucinated replay*, consists in adding samples from the model to the training data set. More specifically, once a trajectory is generated from the environment, there is a chance that it will be replaced by an hallucinated duplicate: that is, one or multiple subsequent observations along the trajectory are substituted by the ones generated by the model for that same transition. With this approach, the model will be trained with some samples that are generated by itself, while the others will still come from the environment, giving the model a way to adjust its predictions. A different approach is to change the plan backwards [14]: when planning forward, the value of a real state might be updated taking into account misleading values of hallucinated states. However, in backward planning, after observing a transition $(s_t, a_t, r_{t+1}, s_{t+1})$ the model will be used to generate the transitions $(\hat{s}_{t-n}, \hat{a}_{t-n}, \dots, \hat{r}_t)$ that led to it and to update

$$Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1}) \leftarrow Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1}) + \\ \alpha(\hat{r}_{t-n} + \gamma \max_a Q(\hat{s}_{t-n}, a) - Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1})).$$

This way, the algorithm will update values for states generated by the model by using real and generated transitions. The damage from the compounding error is therefore reduced, since it would mislead the value of hallucinated states that will never be encountered in the environment.

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.

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 have meaningful differences in what they learn, to grant the adversarial agent meaningful choices that can 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 Bayesian NNs).

REFERENCES

[1] Andreae, J. H. (1966). *Learning machines: a unified view.* Standard Telecommunications Laboratories.

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

[3] Boney, R., Di Palo, N., Berglund, M., Ilin, A., Kannala, J., Rasmus, A., and Valpola, H. (2019). Regularizing trajectory optimization with denoising autoencoders. In *Advances in Neural Information Processing Systems*, pages 2859–2869.

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

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

[6] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

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

[8] Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2013). Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423.

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

[10] Givan, R., Leach, S., and Dean, T. (2000). Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1-2):71–109.

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

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

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

[14] Jafferjee, T., Imani, E., Talvitie, E., White, M., and Bowling, M. (2020). Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*.

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

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

[17] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR.

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

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

[20] Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE.

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

[22] Korenkevych, D., Mahmood, A. R., Vasan, G., and Bergstra, J. (2019). Autoregressive policies for continuous control deep reinforcement learning. *arXiv preprint arXiv:1903.11524*.

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

[24] Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

[25] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[26] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

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

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

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

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

[31] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR.

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

[33] Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2016). Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

[34] Rajeswaran, A., Mordatch, I., and Kumar, V. (2020). A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, pages 7953–7963. PMLR.

[35] Rashidinejad, P., Zhu, B., Ma, C., Jiao, J., and Russell, S. (2021). Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *arXiv preprint arXiv:2103.12021*.

[36] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

[37] Shyam, P., Jaśkowski, W., and Gomez, F. (2019). Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788. PMLR.

[38] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[39] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[40] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[41] Smit, J., Ponnambalam, C. T., Spaan, M. T., and Oliehoek, F. A. (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] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

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

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

[45] van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable approximations to mdp homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963*.

[46] Van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, pages 14322–14333.

[47] Von Stackelberg, H. (2010). *Market structure and equilibrium*. Springer Science & Business Media.

[48] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[49] Wiesemann, W., Kuhn, D., and Rustem, B. (2013). Robust Markov decision processes. *Mathematics of Operations Research*, 38(1):153–183.

2

Appendix

2.1. Introduction

When human beings learn, they interact with the world. We do it through our own life experience, by combining knowledge from other people, making mistakes and many more ways. In Reinforcement Learning (RL), an agent interacts with the environment to learn its dynamics by retrieving experience samples [89]. As humans, we know the consequences of throwing a ball to a wall: we can easily predict which direction it will bounce towards, even if we saw that happening only once. Similarly, when playing video-games, even if the environment does not equal our world (e.g., in pixel-art games) we are still able to generalize our knowledge to understand the dynamics of the game. Sequential decision-making problems like this can be modelled as Markov Decision Processes: a way to solve these problems is using RL algorithms if the dynamics of the environment are not known in advance in their integrity (e.g., video-games). RL agents will need several interactions with the environment to learn its dynamics and, more generally, how to behave to accomplish one or more tasks.

RL algorithms use interactions with the environment to compute a policy determining the agent's actions to accomplish a predefined task. In the literature, we can observe two classes of methods to exploit the interplay between our agent and the real world: Model-Free Reinforcement Learning (MFRL) and Model-Based Reinforcement Learning (MBRL) algorithms. The former learn directly from data retrieved from the environment. The latter build an approximate model of the environment which will generate new simulated experience to adjust and refine their behaviour. In this paper, we will deal with the latter methods: MBRL results to be more appealing in theory, since it requires fewer interactions with the environment, which in many practical applications are often scarce or costly to retrieve. However, they may not always achieve the best solution.

MBRL methods struggle to make their way in real-world scenarios due to some flaws in their decision-making process. When dealing with complex environments, the dynamics are often estimated with powerful function approximators, such as Neural Networks. A drawback of these methods is that they are not guaranteed to converge to an optimal solution, and they might not accurately resemble the true environment dynamics. As a consequence, the planning process is not guaranteed to result in an optimal policy. This leads to several problems. For example, the policy might exploit inaccuracies of the learned model to retrieve high rewards [10, 97] 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. Another major issue in MBRL algorithms is the possibility of the model dynamics to diverge from the ones of the environment [97] due to errors in the model predictions being propagated and summing up along a trajectory, ending up in a state that hardly resembles any state from the environment. These two phenomena are known in the literature respectively as model exploitation and compounding errors. Nonetheless, Model-Based methods are very appealing with respect to Model-Free approaches since using a model to approximate the environment has the potential to lead to better sample efficiency: instead of retrieving samples from the real world, they can be generated by the model.

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 maximise the expected reward while the other wants to minimise it by choosing the worst possible model in

the ensemble. This way, the player maximising the expected reward will have to take into account the adversary actions by solving a maximin optimisation 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?

2.2. Background

The environment in which RL agents act can be formalized as a Markov Decision Process [4].

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, $P : S \times A \rightarrow S$ is the transition probability function, $R : S \times A \rightarrow \mathbb{R}$ is the reward function and γ is the discount rate.

When interacting with the environment, at each time step $t = 1 \dots T$ 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 expected 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

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \tag{2.1}$$

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

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a]. \tag{2.2}$$

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

$$\pi^* = \arg\max_\pi q_\pi(s, a), \tag{2.3}$$

for all $s \in S$ and $a \in A$

Several algorithms have been proposed to solve an MDP and compute an optimal policy [9, 17, 44, 57, 70, 77, 86, 103]. We will briefly introduce the two classes of RL algorithm: Model Free Reinforcement Learning (MFRL) and Model Based Reinforcement Learning (MBRL). The difference between the two categories of models is that the latter builds an internal model approximating the dynamics of the MDP to solve. In figure 2.1 it is presented a taxonomy of some of the most known RL algorithms.

2.2.1. Deep Reinforcement Learning

Tabular RL methods allows the usage of exact planning algorithms that guarantee the convergence to an optimal policy for an MDP, keeping track of the visited state-action pairs in a table. However, these methods do not scale well to large problems where it is unfeasible to explore and store all the possible pairs. This problem led researchers to find solutions using function approximation. In fact, it is possible to estimate several functions involved in RL problems, such as the Q-(or state-) value function, the MDP dynamics (reward and transition probability functions) or the policy. Employing function approximators allows for a more efficient

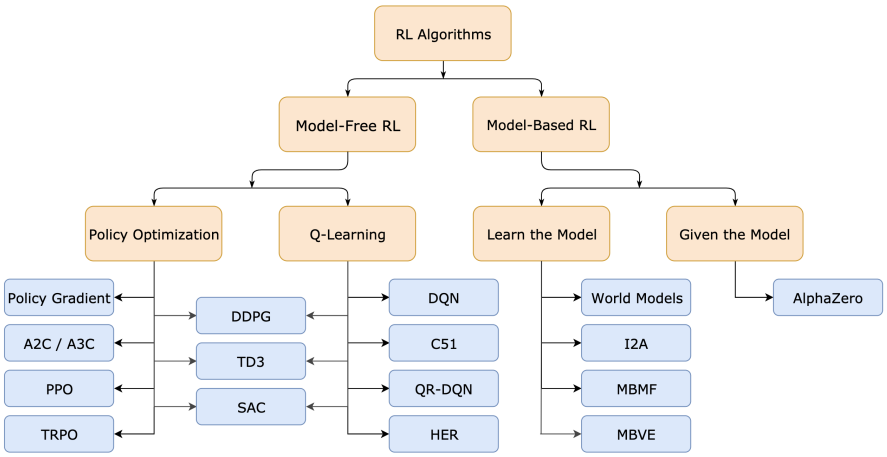


Figure 2.1: A (non-exhaustive) taxonomy of Reinforcement Learning algorithms, from [2].

representation of the state-action space. More specifically, significant accomplishments have been achieved using Deep Neural Networks (DNNs). DNNs are non-linear function approximators that can be used to represent any smooth function, but they are not guaranteed to converge to a local optimum. In the following sections, when considering approximated functions we will represent them with a subscript indicating their parameters (e.g., π_θ is a policy parameterised by parameters θ). When not specified, the reader can assume that the function approximator employed is a Deep Neural Network.

2.2.2. Model-Free Reinforcement Learning

MFRL algorithms aim to solve an MDP by directly approximating the value function, without making use of the transition and reward function of the MDP. It can be seen as a trial-and-error process where the agent explores the environment gathering more experience to understand which states have the highest value. An example of MFRL algorithm is Q-Learning [104], which stores the values of each state-action pair and, after taking action a_t in state s_t (according to the current estimate of Q) and observing r_t and s_{t+1}, updates them according to the Bellman equation

$$Q^{new}(s_t,a_t) = \underbrace{Q(s_t,a_t)}_{\text{old estimate}} + \overbrace{\alpha(\underbrace{r_t + \gamma \max_a Q(s_{t+1},a_{t+1})}_{\text{target estimate using } r_t} - \underbrace{Q(s_t,a_t)}_{\text{old estimate}})}^{\text{update factor}}, \tag{2.4}$$

The Q-Learning algorithm is guaranteed to converge to the optimal solution Q^* if every state-action value is constantly updated. However, this solution is impractical in large domains (e.g., where the state space is continuous) due to the large amount of memory required to store all the Q-values.

This inability of scaling to large domains was addressed with Neural Fitted Q-Learning (NFQ) [74] by approximating the Q-value function with Neural Networks. This way, instead of a look-up table, the agent uses a function to determine the value of each state-action pair, saving a considerable amount of space. Nonetheless, NFQ requires to retrain a brand new neural network at each iteration, which is a computationally expensive procedure for complex architectures as Convolutional Neural Networks [45]. Also, the deep neural network used in NFQ introduces new instabilities: the algorithm is not guaranteed to converge to an optimal solution, due to the possibility of divergence in NN architectures, and the max operator (as in equation 2.4) might lead to overestimates of the values [96], due to a phenomenon called *moving target*. Deep Q-Learning (DQN) [57] proposes an improvement over NFQ by implementing memory replay: the transitions samples are stored in a buffer, and the NN is updated through regression steps over mini batches drawn from it, reducing the variance in the network outputs. DQN also introduces a target network to select the best action and its weights are updated only once every few iterations: this way, the target is "moving slower" and the bias of the values estimate is reduced. [96] propose Double DQN (DDQN), where two different networks are used to estimate the values and to compute the next best action: this way the target estimate from equation 2.4 becomes

$$r_t + \gamma Q_{\theta^-}(s_{t+1}, max_a Q_\theta(s_{t+1},a)), \tag{2.5}$$

reducing the overestimation of state-action values.

Other Model-Free approaches that proved to be effective are policy gradient and actor-critic methods. In policy gradient methods, the policy is approximated by a function with parameters θ (e.g., with a neural network). The expected return of the policy π_θ can then be written as

$$V^{\pi_\theta}(s_0) = \int_S \rho^{\pi_\theta}(s) \int_A \pi_\theta(s,a) R'(s,a) da ds \tag{2.6}$$

$$R'(s,a) = \int_{s' \in S} T(s'|s,a) R(s,a,s'), \tag{2.7}$$

where $\rho^{\pi_\theta}(s)$ is the discounted state distribution

$$\rho^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi_\theta\}. \tag{2.8}$$

The Policy Gradient Theorem (PGT) states that the gradient of the value function w.r.t. the parameters θ can be expressed as follows

$$\nabla_\theta V^{\pi_\theta}(s_0) = \int_S \rho^{\pi_\theta}(s) \int_A \nabla_\theta \pi_\theta(s,a) Q^{\pi_\theta}(s,a) da ds. \tag{2.9}$$

Through this theorem, the policy parameters can be updated by using the experience retrieved from the real world, as $\theta \leftarrow \theta + \alpha \nabla_\theta V^{\pi_\theta}$, where α is the learning rate. To compute equation 2.9, it is required an estimate of the state-action value function Q, since the algorithm does not have access to it. A common approach is to use actor-critic methods, where the actor is the policy (i.e., the one taking decisions) and the value function is the critic (i.e., analyzing whether the decisions are good). Both the policy and the value function are usually approximated by some parametric model. The former is updated using the PGT according to the feedback of the critic which, in turn, is updated following the directions of the actor.

2.2.3. Model-Based Reinforcement Learning

In MBRL, we build a parameterized 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 enviornment (which might be difficult or expensive to get in some-real world scenarions):

Definition 4. 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 [87]. A general template for a MBRL algorithm is described in Algorithm 1 [97]: the procedure UPDATEMODEL fits the parameterized model to the data retrieved exploring the environment, while UPDATEAGENT procedure solves the given MDP.

However, model-learning is not straightforward: most of the times a ground-truth model will not be available to estimate the real dynamics, and 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 that the algorithms can exploit model inaccuracies (a phenomenon known as model exploitation), computing a policy that performs incredibly well on the model but sub-optimally (or even terribly) in the environment.

2.2.4. Robust Reinforcement Learning

Using complex function approximators like DNNs has led to remarkable advances in the field of Reinforcement Learning. Employing Neural Networks, algorithms can achieve a (super-) human level of ability in performing certain tasks. However, a critical drawback of these powerful architectures is their dependency on data: a huge amount of samples is required to obtain significant results. This is a major bottleneck in a field like Reinforcement Learning, where collecting new samples from a simulator can be highly time expensive, while collecting them from the real world could even be dangerous. Model-Free RL approaches usually rely on the former: learning a policy from a simulator which should then be deployed in real-world scenarios. However, if the algorithms are not robust to modeling errors, the gap between reality and simulators often makes this transfer unsafe. In previous chapters we discussed how Model-Based RL methods try to mitigate

Algorithm 1 Model-Based Reinforcement Learning

Input: state sample procedure d
Input: model m
Input: policy π
Input: environment ε
Get initial state $s \leftarrow \varepsilon$
for iteration $\in$ {1,2,...,K} **do**
  for interaction $\in$ {1,2,...,N} **do**
    Generate action: $a \leftarrow \pi(s)$
    Generate reward, next state: $r, s' \leftarrow \varepsilon(a)$
    $m, d \leftarrow$ UPDATEMODEL(s, a, r, s')
    Update current state: $s \leftarrow s'$
  end for
  for planning step $\in$ {1,2,...,P} **do**
    Generate state, action $\bar{s}, \bar{a} \leftarrow d$
    Generate reward, next state $\bar{r}, \bar{s}' \leftarrow m(\bar{s}, \bar{a})$
    $\pi \leftarrow$ UPDATEAGENT$(\bar{s}, \bar{a}, \bar{r}, \bar{s}')$
  end for
end for

this data request: using some real world samples to learn an approximated model of the dyanamics of the MDP allows to generate more artificial data to use during planning. Still, MBRL approaches are not free from problems. Using a learned dynamics model further exacerbates the gap between reality and the experience provided to the planning agent.

When interacting with the true MDP we can rely on optimistic exploration, knowing that our (Q-)value function and policy estimates will eventually converge to an optimal solution. However, when using function approximation to model the dynamics of the MDP, this is no longer guaranteed since we are planning on an imperfect model. Therefore, MBRL algorithms must be robust to model errors to achieve good performance.

Robust Markov Decision Processes (RMDPs) address the dynamics uncertainty by following a robust optimisation 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 maximise the worst-case expected reward, based on the adversarial choice of $P \in \mathbb{P}$. Following this game-theoretical approach the optimisation problem in (3) can be redefined as

$$\pi^* = \arg\max_{\pi \in \Pi} \min_{P \in \mathbb{P}} v_{\pi,P}(s), \tag{2.10}$$

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^*. Through this formulation, it is possible to take into account the modelling imperfections as adversarial attacks that meddle with each transition output. In the literature, this approach has been used both for Model-Based and Model-Free algorithms to improve their robustness to approximation errors. In [52], the authors combine Optimism in the Face of Uncertainty (OFU) with the Robust MDPs framework, detecting adversarial inputs in an online manner and computing a provably optimal minimax policy. However, their approach requires to sample all the state-action space, which is unfeasible for large domains.In [71], the authors present Ensemble Policy Optimization (EPOpt). In this Model-Based approach, using an ensemble of simulators, the agent learns a robust policy by planning on the worst ϵ percentage of trajectories experienced. In this work, the algorithm does not learn the dynamics of the simulated environment, but a distribution on its parameters (e.g.: gravity, friction, mass): this is feasible for a small parameter spaces, but will require more computational power for more complex environments. A Model-Free approach is followed by [69], 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 minimise the expected reward, while the protagonist aims to maximise the expected reward (as usual in RL settings) by learning a policy robust to the adversarial inputs. In [72], the authors propose a Model-Based RL formulation of a two-player game. More specifically, they cast the problem as a Stackelberg game [102]: an asymmetric game where a specific order of the players is imposed. In this approach, the pol-

icy player wants to maximise the expected reward within the current approximated model, while the model player wants to minimise the prediction error under the state distribution induced by the policy.

2.3. Model Accuracy in MBRL

The prediction accuracy of the model is of critical importance in MBRL algorithms since planning relies on it. A poor model may lead to faults in planning or to an overestimate of the expected reward which will, in turn, compromise the accuracy of the policy computed by the algorithm [10, 33, 48]. We will first cover a straightforward approach to improve the accuracy when learning: choosing the appropriate model. Then we will go through techniques to reduce the dimensionality of the domain through abstraction and representation learning, and to explore the environment more efficiently.

2.3.1. Model Choice

Model-Based Reinforcement Learning algorithms have always been compelling due to their data efficiency. 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 [46]. For example, the Dyna architecture [87] relies on the model built during the learning phase to generate new experience and adjust the learned policy during planning. With this approach, the required amount of data requested is greatly reduced and it is still possible to use Model-Free techniques in the planning section to have a better asymptotic performances. Therefore, choosing the appropriate model to improve the algorithm performance might seem a simple task, but it is not always straightforward to accomplish.

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) [20, 29, 39, 41, 42, 61], but 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) [16, 33, 37, 59]. Parametric models can rely on more flexibility, allowing them to represent more complex functions, while non-parametric models are more efficient where few data samples are available, as is the case for GP.

PILCO [19] is an example of MBRL algorithm using a non-parametric model. In fact, PILCO uses Gaussian Processes to implement a probabilistic dynamics model of the environment reducing model bias and improving sample efficiency. Two main issues with this solution are the sensitiveness to noise in the states and the ability to deal only with fairly simple environments, due to the curse of dimensionality affecting Gaussian Processes. The former problem was addressed by [55], where the authors extended the algorithm to Partially Observable environments while keeping the same sample efficiency, while the latter was addressed by [16], by employing Neural Networks ensembles to model the uncertainty, achieving the state of the art in high dimensional domains.

[75] show that the good planning performance obtained when acting with the learned approximated dynamics might not match the performance in the real environment. This is because the exploration policy (used to collect samples) usually differs from the policy computed during planning, leading to a state distribution shift. Because of this, the model will end up underestimating the cost of less-explored regions of the domain and the optimized policy will lean towards them, causing poor performance of the algorithm. [75] formally guarantee that by using Data Aggregation (DAgger) [76] when learning the model, either the algorithm computes a good policy or it does not exist an accurate model for this domain (and must be found in another class of models). During the learning phase, at each iteration n of the algorithm, DAgger proceeds by:

1. With probability $\frac{1}{2}$, collecting new trajectories using the current policy π_n. Otherwise, transitions are sampled from the exploration distribution.

2. Aggregating the newly acquired data into the training dataset.

3. Fitting the model to the training dataset.

4. Solving the model, computing the new policy π_{n+1}.

This way, the agent will not focus solely on the state distribution arising from the optimized policy. However, since DAgger does not guarantee the improvement of the policy (i.e., it might be that $E_{\pi_n}[G_t] \geq E_{\pi_{n+1}}[G_t]$),

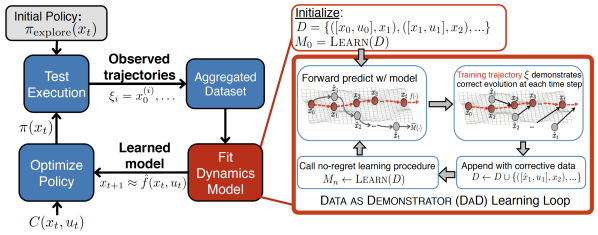


Figure 2.2: From [99]. On the right it is illustrated the Data as Demonstrator process: the predicted transitions are adjusted by using the ones observed in the real environment and then added to the dataset. The model can therefore adjust its predictions.

the algorithm has to keep track of the best policy found so far and accordingly updates it at each iteration. A scheme illustrating this procedure is provided in Figure 2.2.

The DAgger approach can suffer from the compounding errors resulting from multi-step rollouts (used to sample trajectories). [99] explore a way to improve the learning process of models using multi-step predictions, leveraging Data as Demonstrator [100] and DAgger. With single-step predictions, given a state s and an action a, the model fits the environment dynamics in a supervised learning fashion: minimizing the loss between the output $f(s_t, a_t) = \tilde{s}_{t+1}$ of the model and the true value s_{t+1} from the environment. Differently, when using multi-step predictions, samples are collected from the environment along N trajectories ξ_n, then minimizing the loss [1]

$$L = \sum_{t=1}^{T} \sum_{i=t}^{T} ||s_i - \tilde{s}_{i|t}||_2^2, \tag{2.11}$$

Where $\tilde{s}_{i|t}$ is the output of the model after i roll-out steps starting from s_t, following the trajectory ξ_n. However, this loss is very hard to optimize and, when performing roll-outs, the output of the model is recursively fed as input to the next step, so the prediction errors can compound and cause serious inaccuracies [1, 98, 100]. To improve the robustness to compunding errors of the multi-step model, [99] propose to use the data from the environment as a guide for the algorithm (Figure 2.2). This means that, after generating the roll-out steps, each artificial data sample $(\tilde{s}_t, a_t, \tilde{s}_{t+1})$ will be adjusted by substituting true data (previously sampled from the environment) to the output state. The new triplet $(\tilde{s}_t, a_t, s_{t+1})$ will then be added to the dataset on which the model will be trained, to correct compounding errors.

The experiments conducted by [99] prove that the addition of DaD enables the model to learn more stable multi-step dynamics. In fact, when using a random exploration policy, the DAgger approach rarely learned a stable policy, as opposed to the DAgger+DaD algorithm. Moreover, when DAgger and DaD are combined, a good performance is obtained even without adding samples to the training set from the exploration distribution (while this is a requirement for the DAgger only performance to be guaranteed). However, in noisy environments, combining the two algorithms did not bring substantial improvements with respect to the DAgger baseline, due to the variation of the sampled trajectories limiting the learning ability of the algorithm.

Recently, Model-Based Reinforcement Learning was employed in the Atari games domain [37]. The authors proposed Simulated Policy Learning (SimPLe), summarized in Algorithm 2. Using deep convolutional model, the algorithm is based on video prediction and was able to outperform the learning speed of Model-Free state-of-the-art methods. The SimPLe architecture was trained on the Atari games domain, a very challenging environment due to the high dimensional state space. After initializing the model by training it on trajectories collected through a random policy, the dataset is subsequently extended through further interactions with the environment, and the model is accordingly updated. When planning, the policy is improved using Proximal Policy Optimization (PPO) [80]. Since PPO performs rollouts using the model, to reduce the impact of compounding errors the authors reduced their length. As it will be discussed in Section 2.4.2, reducing the planning horizon can help preventing overfitting the data: in SimPLe, the authors not only fine-tuned the γ parameter, but also truncated the roll-out at a certain depth. This technique may not allow the algorithm to access rewards after a certain time step, preventing it from getting a complete experience along the trajectory: to solve this problem, after stopping the roll-out, the authors added to the trajectory evaluation the value estimate of the final state. However, the final scores were still favoring the Model-Free approach and the performance of the algorithm varied substantially for different runs on the same game. SimPLe also uses a Data as Demonstrator approach to mitigate the effect of compounding errors when performing roll-outs, due to imperfections of the model.

Algorithm 2 SimPLe

Input: Empty dataset D
Input: Random policy π
Input: Initialized model parameters θ for model M_θ
Input: number of iterations N
Input: procedure TRAIN_MODEL to fit the model to the dataset
Input: procedure TRAIN_POLICY to update the policy using the model

for n $\in \{0, ..., N\}$ **do**
    $\diamond$ Collect new observations from the environment
    $D \leftarrow$ COLLECT(π_n)
    $\diamond$ Update model using the extended dataset
    $\theta \leftarrow$ TRAIN_MODEL($M_{\theta,D}$)
    $\diamond$ Update policy using the updated model
    $\pi \leftarrow$ TRAIN_POLICY(π, M_θ)
end for

While in SimPLe (as in many other MBRL architectures) the model predicts the next observed state, MuZero [79] predicts three different quantities: the (expected) value function, the immediate reward and the policy (i.e., the next action to take). The intuition behind MuZero internal structure is that the dynamics model simulates an MDP where the states do not represent actual states of the environment: they can be seen as an inner layer of a Neural Network, since they only have the purpose of correctly approximate the three aforementioned functions. An advantage of this approach is that the model is not constrained to capture a specific state semantic nor to accurately learn all the environment dynamics, but only the most relevant features to predict the policy and state values. MuZero uses Monte Carlo Tree Search (MCTS) [13], which computes the policy to follow and the expected value at each iteration by repeating three steps (the search process) for every simulation:

- Selection: starting from the root node of the internal model s_0, at each simulation the algorithm performs an n-steps roll-out, until reaching a leaf node s_n. The roll-out is performed by choosing actions that minimizes an Upper Confidence Bound (UCB)
- Expansion: after reaching the leaf node, the reward, expected value and state s_n are predicted using the model and stored for future iterations. Then, s_n is expanded by creating a number of successors equal to the number of possible actions (which are the edges connecting them to s_n).
- Backup: the expected value of the simulated trajectory is propagated back to the ancestors of s_n, updating the (approximated) state-action value function.

The authors proposed also a variation of the MuZero algorithm, called MuZero Reanalyze, that re-experiences past trajectories by performing the search operation using the most updated model, improving the quality of the policy. Also, the algorithm parameters were fine-tuned to increase sample reuse and reduce overfitting. The MuZero algorithm (and its variation) proved to achieve better scores than state-of-the-art MFRL algorithms in the challenging domain of Atari Games.

However, as shown by [5], having a great accuracy on the model does not guarantee efficiency when planning. The authors propose Generative Adversarial Tree Search (GATS), an algorithm combining Generative Adversarial Networks (for the model) and Monte Carlo Tree Search (for planning). A downside of the MCTS algorithm is the heavy computational cost to perform hundreds of rollout steps. The authors hypothesized that by having an accurate model, the number of steps could have been reduced: their results proved that short rollouts can fail even when used in combination with perfect models, therefore requiring more data (i.e., rollouts) to improve the accuracy. The authors further claimed and empirically tested the fact that, with short rollouts, negative actions can be avoided but the information is not efficiently propagated, slowing the learning process.

2.3.2. Abstraction and Representation Learning

In real-world scenarios the amount and complexity of states and actions greatly increase. While deep (MF)RL algorithms become sample hungry, requiring more data to accurately estimate the value function and an

optimal policy, MBRL offers an appealing alternative thanks to the learned model. However, as explained in previous sections, MBRL do not always learn a good representation of the environment: these algorithms are usually forced to explore a very restricted part of the domain, due to its complexity, which might not be enough to fully understand the true dynamics. This can be seen as an instance of the *curse of dimensionality* [8].

An effective solution to this problem is representation learning (i.e., dimensionality reduction), where agents learn how to act in an environment simpler than, but equivalent to, the real one. States and actions can be mapped to a different domain but with dynamics similar to the ground truth environment, so that the learning process can be simplified while not reducing the quality of the policy computed (which can be "translated" back to the original domain). A renown algorithm in ML is Principal Component Analysis (PCA) [66], which has also been applied in the RL setting [63]. However, the most common (and successful) approach to learn abstract representations resides in Neural Networks, by using parameterized functions:

1. $f_\theta^{\text{enc}}(s_t) : S \to Z$, encoding states from the original space and mapping them to the abstract domain Z;
2. $A_\theta(a_t) : A \to \overline{A}$, encoding actions from the original space and mapping them to the abstract domain $\overline{A}$;
3. $R_\theta(r_t) : R \to \overline{R}$, encoding actions from the original space and mapping them to the abstract domain $\overline{R}$;
4. $\overline{T}_\theta(z_t, \overline{a}_t) : Z \times \overline{A} \to Z$, approximating the transition function of the abstract domain;
5. $f_\theta^{\text{dec}}(z_{t+1}) : Z \to S$, decoding abstract states and mapping them to the original state-space.

An important aspect of representation learning is the loss function. Since the agent plans in the abstract space, the abstract dynamics should be guaranteed to be close to the original ones. Therefore, when taking an action a_t in the abstract state z_t, the resulting state z_{t+1} should be very close to the encoding of the ground-truth state $f_\theta^{\text{enc}}(s_{t+1})$. One approach would be to implement pixel reconstruction, using a loss that simply penalizes states that different from the encoding of the true next state [105]. However, these kind of losses are computationally expensive due to the reconstruction process. In the most recent literature the focus has moved on contrastive losses. A contrastive loss measures the similarity with other observations, instead of computing the difference between single data points. Some works focus solely on the prediction of the state following a transition, but this can result in incorrect mappings where states are considered to be the same (in the latent space) despite having a very different value. [94] propose a loss to learn equivariant mappings, where the dynamics of the abstraction are equivalent to the ones from the environment. To avoid an erroneous state mapping, they included the reward function into the loss, to better distinguish states. However, in case of sparse rewards the learned representation could collapse to a trivial solution (all states mapped to the same one). To prevent this, they added a contrastive loss computing the distance between the predicted state and the encoding of a set S' of other states sampled along the trajectory. Given N ground truth sample transitions in the form (s_{t+1}, s_t, a_t, r_t) and the corresponding latent-space approximations $(z_{t+1}, z_t, \overline{a}_t, \overline{r}_t)$, the loss can be expressed as follows

$$L = \frac{1}{N}\sum_{n=1}^{N}\left[\underbrace{d\left(z_{n+1}, \overline{T}_\theta(z_n, \overline{a}_t)\right)}_{\text{Distance for (predicted) next state}} + \overbrace{d\left(r_n, \overline{r}_n\right)}^{\text{Distance for (predicted) reward}} + \underbrace{\sum_{s\in S'} d_-\left(f_\theta^{\text{enc}}(s), \overline{T}_\theta(z_n, \overline{a}_t)\right)}_{\text{Contrastive loss to avoid embedding collapse}}\right], \tag{2.12}$$

where $d_-(z, z') = \max(0, \epsilon - d(z, z'))$ is a negative distance function. The contrastive loss grows if the predicted state is similar to the others sampled states: since the algorithm is minimizing L, it will lean towards learning embeddings that do not oversimplify the original domain.

The authors finally proved that when minimizing 2.12, the abstraction resulted in an MDP *homomorphism*, meaning that the dynamics in the embedded MDP are equivalent to the true ones. More formally

Definition 5. A (deterministic) MDP homomorphism from a deterministic MDP $M = \langle S, A, T, R\rangle$ to an MDP $\overline{M} = \langle Z, \overline{A}, \overline{T}, \overline{R}\rangle$ is a tuple $h = \langle f_\theta^{\text{enc}}, A_\theta\rangle$ with

- $f_\theta^{\text{enc}} : S \to Z$ being the state embedding function
- $A_\theta : A \to \overline{A}$ being the action embedding function

such that the two following identities hold

$$\overline{T}(f_\theta^{\text{enc}}(s), A_\theta(a)) = Z(s') \qquad \forall_{s,s' \in S, a \in A} \tag{2.13}$$

$$R_\theta(f_\theta^{\text{enc}}(s), A_\theta(a)) = R(s,a) \qquad \forall_{s \in S, a \in A} \tag{2.14}$$

where $s' = T(s,a)$.

A critical property of MDP homomorphisms is that an optimal policy for the abstract MDP $\overline{M}$ is also optimal in the original MDP M. Therefore, if the loss presented in equation 2.12 is minimized and the abstract space is small, exact planning algorithms (like Value Iteration) can be employed so that the computed policy is guaranteed to be optimal. However, as the authors of the paper state, their paper only covers the case for deterministic MDPs: the application of this algorithm to stochastic environments is still unexplored.

2.3.3. Exploration

Another way 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.

All the previous approaches make use of greedy exploration. That is, they select the policy π that greedily maximizes the expected return over the model uncertainty, which is an optimal strategy only in a few cases, as the Linear Quadratic Regulators [54]. Provably optimal exploration strategies have been implemented as variations of Thompson Sampling [92], where a set of actions is sampled from the posterior distribution over the MDP. A first approach of this technique was proposed by [85], with the Posterior Sampling for Reinforcement Learning (PSRL) algorithm. Another common heuristic is the Optimism-in-the-Face-of-Uncertainty (OFU), where the agent assigns to each action an optimistic estimate of the expected reward, and select the option with the highest future value. This way, if the action value is overestimated, the agent can fine-tune its expectations and learn from the experience gained. An example of algorithm using this approach is R-Max [11], described in Algorithm 3 (from [73]). In R-Max, the agent maintains an optimistic estimate of a state-action pair until it is experienced a certain amount of times m. After m times, the estimated value for the state-action pair is set to its expected valued, considering the experience gained in the past.

Thompson Sampling (TS) has been proven to reach better statistical efficiency, while also having a lower regret bound [65]. However, TS can be applied successfully only when the posterior distributions is tractable, which is a big limitation when dealing with complex domains. On the other hand, tabular OFU algorithms like R-Max do not scale well either to more complex domains, where the number of states and actions dramatically increase.

[18] propose an OFU algorithm to reduce from optimistic exploration (which is generally intractable) to greedy exploration. They propose a variation of Upper Confidence Reinforcement Learning (UCRL) [34]. UCRL optimizes the following criterion in a tabular MDP setting:

$$\pi_t^{UCRL} = \arg\max_{\pi \in \Pi} \max_{\tilde{f} \in M_t} J(\tilde{f}, \pi). \tag{2.15}$$

The UCRL algorithm is intractable, since it jointly optimizes over policies Π and over the set of statistically-plausible models $M_t = \{\tilde{f} \text{ s.t. } |\tilde{f}(s,a) - \mu_t(s,a)| \leq \beta_t \sigma_t(s,a) \forall s, a \in S \times A\}$, where μ and σ are the expected value and variance of the models' dynamics at time t. Therefore, [18] propose Hallucinated-UCRL (H-UCRL), where the policy is obtained as follows:

$$\pi_t^{H-UCRL} = \arg\max_{\pi \in \Pi} \max_{\eta(\cdot) \in [-1,1]^p} J(\tilde{f}, \pi) = \mu_{t-1}(s,a) + \beta_{t-1} \Sigma_{t-1}(s,a) \eta(s,a), \tag{2.16}$$

which is tractable when choosing Lipschitz-continuous bounded functions $\eta(\cdot)$. The η function allows to select the best policy by controlling the degree of confidence on the output of the dynamics (i.e., decides the most appropriate amount of variance in the output of the function). Through η, the model is able to optimistically choose the dynamics output while still taking into account the confidence interval. Also, this way the uncertainty of the model is propagated step by step (as represented in Figure 2.3) and not over the whole trajectory.

[81] introduce Model-Based Active eXploration (MAX) to solve both the model accuracy and model exploitation problem. MAX is 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. To do so, the authors compute the exploration policy that maximizes the following utility function

Algorithm 3 R-MAX

Input: S, A, γ, m, ϵ_1, $R_{\max}$

$\overline{S} \leftarrow S \cup \{z\}$ where z is a fictitious state
for $(s,a) \in \overline{S} \times A$ **do**
  $n(s,a) \leftarrow 0$
  $r(s,a) \leftarrow 0$
  $\tilde{Q}(s,a) \leftarrow \frac{R_{\max}}{(1-\gamma)}$
  $\tilde{R}(s,a) \leftarrow R_{\max}$
  for $s' \in S$ **do**
    $n(s,a,s') \leftarrow 0$
    $\tilde{T}(s,a,s') \leftarrow 0$
  end for
    $n(s,a,z) \leftarrow 0$
    $\tilde{T}(s,a,z) \leftarrow 1$
end for
for $t = 1,2,\ldots,3$ **do**
  Observe state s
  Execute action $a := \arg\max_{a' \in A} \tilde{Q}(s,a')$
  Observe immediate reward r and next state s'
  if $n(s,a) < m$ **then**
    $n(s,a) \leftarrow n(s,a) + 1$
    $r(s,a) \leftarrow r(s,a) + r$
    $n(s,a,s') \leftarrow n(s,a,s') + 1$
    if $n(s,a) < m$ **then**
      $\tilde{R}(s,a) \leftarrow \frac{r(s,a)}{m}$
      for $s'' \in \overline{S}$ **do** $\tilde{T}(s,a,s') \leftarrow \frac{n(s,a,s')}{m}$
      $\tilde{Q} \leftarrow$ Solve MDP $(\overline{S}, A, \tilde{R}, \tilde{T}, \gamma, \epsilon_1)$ using Value Iteration
    endif
  endif
end for

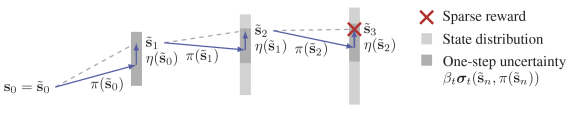


Figure 2.3: From [18]. At each step of the trajectory, the following state is chosen within an uncertainty region (dark grey) of the domain (light grey) using η.

$$u(s,a) = JRD(p_1(s'|s,a), p_2(s'|s,a), \dots, p_N(s'|s,a)) \tag{2.17}$$

$$= H_2\left(\frac{1}{N}\sum_i^N p_i(s'|s,a)\right) - \frac{1}{N}\sum_i^N H_2(p_i(s'|s,a)) \tag{2.18}$$

$$H_\alpha(X) = \frac{1}{1-\alpha}\ln\int p(x)^\alpha dx \tag{2.19}$$

where N is the number of models implemented in the ensemble, each of which approximates a transition probability p_i, JRD is the Jensen-Rényi Divergence and $H_\alpha(X)$ is the Rény entropy of a random variable X. The JRD express the disagreement of the models on the transitions, therefore the policy maximising this utility function will be prone to explore the uncertain regions of the model, to collect new information. The JRD is used so that there is a tractable solution for the continuous state space: the authors chose to model the ensemble as a mixture of Gaussians, and with $\alpha = 2$ equation 18 has a closed-form solution.

2.4. Model exploitation in MBRL algorithms

The model learned by MBRL algorithms is what makes these methods more appealing than model-free solutions and, unfortunately, also what makes them less reliable. [37] show that MBRL can learn environment dynamics in a much more efficient way than model-free methods. The experiments set was the Atari video games domain [7]. Despite the higher data-efficiency, their method still obtained lower scores than the state-of-the-art model-free methods when trained with more iterations. Also, as reported by the authors themselves, the scores obtained by the model-based architecture were not consistent over multiple runs on the same game. The instability of MBRL methods resides in the inaccuracies of the model used [97].

Model inaccuracies are defined as differences between its dynamics and the ones of the real environment, which the model tries to approximate. The dynamics of the model are said to diverge when the model may lead to a state distribution other than the environment one. This prediction mistakes will then sum up if the output will in turn be the input to the model to generate a trajectory, a phenomenon known as *compounding errors*.

According to the recent literature [97], the model accuracy is affected by the so-called *deadly triad* [88]. The term *deadly triad* indicates the combination of *bootstrapping*, *function approximation* and *off-policy learning* in the same model. These three components can be identified in MBRL algorithms as follows: the parameterized model equals *function approximation*, since it is basically a regression task on the environment dynamics. *Bootstrapping*, which means improving the policy using estimates of the future rewards and not only real data, is applied when planning (i.e., the third for-loop in algorithm 1). *Off-policy learning* in MBRL methods is another consequence of using a parameterized model. In fact, during planning, MBRL agents use a policy improved according to the model: this policy does not reflect the one used to generate new real data and this is the definition of off-policy learning.

The problem of model inaccuracies exploitation is well-known in the literature [10, 36, 51] and it is closely related to the problem of model accuracy. The agent might exploit these model faults to compute very high expected reward policies in the simulated environment, which will nonetheless obtain low rewards in the real one due to different dynamics. A good illustration of this problem is presented by [10], which is provided to the reader in the rightmost image of figure 2.6. In a complex environment like Half-cheetah the expectation of the agent's utility is very different from the real one (bottom-right plot).

In the next sections we will cover how the planning direction, horizon length and regularization can influence the convergence of the algorithm to an optimal policy. Afterwards, we will see how some of these techniques have been applied in the literature to prevent compounding errors. Finally, we will cover some of the most recent approaches to counter the objective mismatch in MBRL algorithms.

2.4.1. Planning Direction

When planning, the optimization objective is to maximize (or minimize) the policy expected reward (or cost). The planning algorithm should suit the approximated model [58]: some require the model to be differentiable [19, 24, 56, 93], others do not [26, 49, 78, 82, 83, 87]. In addition to the algorithm itself, the planning result can be improved by appropriately choosing the planning direction.

Two planning directions can be distinguished: forward and backward. In forward planning [30, 31, 38, 87], the algorithm predicts which state s_{t+1} will follow from taking an action a_t in a certain state s_t, updating the

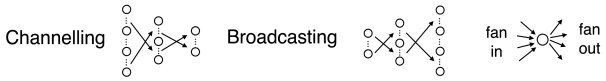


Figure 2.4: From [15]. A graphic representation of Channelling, Broadcasting, Fan-in and Fan-out definitions.

current state expected value. Backward planning works the opposite way, predicting which action-state pair (s_{t-1}, a_{t-1}) generated a certain state s_t, updating the previous state expected value.

[15] explore the implications of planning direction in a credit assignment setting, based on the environment structure. In credit assignment, the objective is to assign the correct reward to each state (or state-action pair) through the delayed rewards obtained for each transition. Forward models, once a transition (s, a, s', r) has happened, compute the TD-error between the experienced and expected value of the state s and propagate the new value estimate to the states preceding s in the trajectory. This can be a non-trivial task due to the complexity of searching for a predecessor of s that has to be updated. Also, due to model errors, the states previously generated by the model might not correspond to actual states from the environment: incorrectly updating these imaginary states can lead the model to overestimate the reward and to a sub-optimal policy. Both these problems can be addressed by using backward models, since they predict and update predecessors states. If the Q-Value function is represented by a look-up table and these states do not exist in the environment, the update will not influence the value of real states. If instead the value function is approximated by a parameterized model (e.g., a NN), the parameters update will affect all the state-action pair values, but the overestimation is expected to be reduced. In fact, experiments performed by the authors showed that in a stochastic setting, the impact from model errors was lower in backward models. This can be related to counterfactual learning, where the agent uses experience to predict *what would have happened if* it had taken another action in the same state. Backward models can be interpreted in the same way: by propagating back the reward experienced in the present, the agent can reconsider actions taken in the past. However, for highly stochastic rewards backward models are still unable to accurately capture the environment dynamics.

[15] also studied the effectiveness of backward and forward planning in different kinds of environments. They characterized the environments according to two features: *fan-in*, the number of states leading to a certain state, and *fan-out*, the number of states that can be reached from a certain state. Using these two quantities, we can define *channelling* (high fan-in, low fan-out) and *broadcasting* (low fan-in, high fan-out) environments, as represented in Figure 2.4. Backward planning was more effective in the former, where many previous states can be updated at once (high fan-in) and with lower variance (low fan-out). In contrast, forward planning achieved lower loss scores in broadcasting environments.

2.4.2. Horizon Length

When planning in a MDP, we define *evaluation* and *planning* horizon, respectively represented by γ_{eval} and γ. The former is used to evaluate the policy by discounting future rewards (see equation 2.1), while the second determines the impact of future rewards when computing the optimal policy. While γ_{eval} is usually defined by the problem, γ is a parameter of the planning algorithm and can be fine-tuned to improve the performance of the policy. Greater values of γ will result in accurate policies and increase the computational cost, while smaller values will lighten the computational expenses but will result in sub-optimal policies [40, 43]. Therefore, when choosing an appropriate value for the planning horizon we incur in a trade-off between accuracy and computational cost.

[35] study this trade-off and show that the planning horizon controls the policy overfitting to the problem. In supervised learning settings, overfitting occurs when the complexity of the model is too high and does not generalize well to unseen data. This results in low loss values in the training dataset but poor performances when evaluating the model with the test dataset. Limiting the model complexity is a way to reduce overfitting. [35] state and prove the following theorem

Theorem 1. *For any fixed state space S, action space A, and reward function R, define*

$$\Pi_{R,\gamma} = \{\pi : \exists\ P \text{ s.t. } \pi \text{ is optimal in } \langle S, A, P, R, \gamma \rangle\}$$

Then the following claims hold:

1. $|\Pi_{R,0}| = 1$.
2. $\forall \gamma, \gamma' : 0 \leq \gamma \leq \gamma' < 1, \Pi_{R,\gamma} \subseteq \Pi_{R,\gamma'}$.

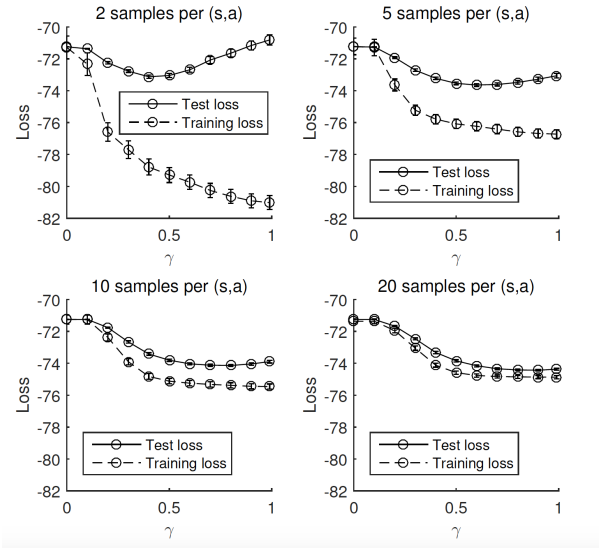


Figure 2.5: From [35]. Here the loss is represented as a function of γ, for training and testing. It can be observed a trend very similar to the one of supervised learning: the training loss is constantly decreasing as we increase γ, while the test loss is U-shaped (the model approximates a larger number of functions, and therefore overfit). The best value of γ is therefore at the bottom of the U-shaped test loss. When we increase the number of sample, the best γ value increases, meaning that the model can afford to fit a larger set of functions.

3. $\exists \gamma \in [0,1), |\Pi_{R,\gamma}| \geq |A|^{|S|-2}$
   if $\exists\, s, s' \in S, \max_{a \in A} R(s,a) > \max_{a' \in A} R(s',a')$.

The first claim ensures that, if there is a single optimal action (i.e., no ties in the reward obtained for each state), the optimal policy is unique for $\gamma = 0$. The second claim states that the optimal policies obtained for a certain value of γ can be obtained also with a higher value γ'. The third claim sets a lower bound of $|A|^{|S|-2}$ for the size of the policy class $\Pi_{R,\gamma}$, if the maximum reward is not available in every state. As a whole, this theorem shows that increasing the value of γ will also increase the size of the policy class (second claim), up to *at least* $|A|^{|S|-2}$ (third claim), which is slightly smaller than the total possible policies $|A|^{|S|}$. As the number of available policies can be matched with the concept of complexity of the model for supervised learning (i.e., the number of functions that can be represented), γ can be used to balance the model overfitting. Figure 2.5 explicitly represents this phenomenon. The authors also proved that the value loss $||V^*_{\pi^*} - V^*_{\tilde{\pi}^*}||_\infty$ is upper bounded, with probability at least $1-\delta$, by

$$||V^*_{\pi^*} - V^*_{\tilde{\pi}^*}||_\infty \leq \frac{\gamma_{eval} - \gamma}{(1-\gamma_{eval})(1-\gamma)} R_{max} + \frac{2R_{max}}{(1-\gamma)^2} \sqrt{\frac{1}{2n} \log \frac{2|S||A||\Pi_{R,\gamma}|}{\delta}}, \tag{2.20}$$

where $V^*_{\pi^*}$ is the true optimal value and $V^*_{\tilde{\pi}^*}$ is the optimal value obtained after planning and computing the optimal policy $\tilde{\pi}^*$ in the approximated model. The second term depends on the complexity $|\Pi_{R,\gamma}|$, which will increase with γ, while making the second term larger while the first one will tend to zero. This effect can be mitigated by increasing the number of available samples n, which allow greater values for γ: again, this resembles the classical machine learning algorithm behaviour, where having more data available will allow more complex functions to be used.

2.4.3. Regularization

To prevent model exploitation, regularization can be applied to act in ways similar to what it has observed in the past. However, this solution brings a trade-off with the exploration of the agent: when penalizing unfamiliar trajectories, the agent might settle for highly sub-optimal solutions. For example, KL-divergence has been widely used to regularize the objective function of MBRL algorithms [6, 46, 51, 67, 68]. The Kullback–Leibler (KL) divergence $D_{KL}(p||q)$ is a measure quantifying the difference between probability distributions defined as

$$D_{KL}(p||q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right).$$

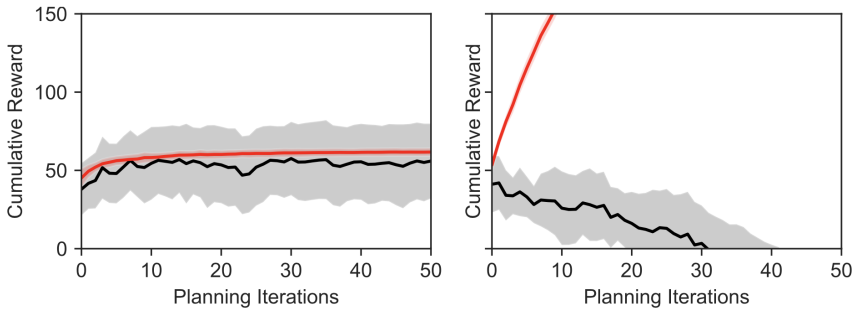


Figure 2.6: Experiments results from [10]. The red line represents the agent expected utility in the approximated environment, while the black one is the utility achieved in the real one. In the leftmost figure, the objective function has been penalized with DAE.

The KL is a natural expression of the distance between the actions generated by the policy when exploring the model and the ones observed when exploring the environment. Using this as a penalization term will lead the algorithm to prefer policies generating data that is likely to happen in the environment (i.e., that were observed in previous exploration). However, as pointed out by [10], when using KL divergence as a constraint for the action distribution, the algorithm will be less prone to experience unfamiliar states, which might harm performances in high-dimensional environments.

Denoising Autoencoders (DAE) [101] have also been employed to perform trajectory optimization, penalizing the objective function [10]. By using DAE, the authors approximate the *derivative of* the probability distribution of observing a given trajectory given the past experience. In fact, when the trajectories are far from the training distribution, the error value computed by the DAE will be larger, indicating that the dynamics are less accurate in that area due to the scarcity of samples in the training dataset. This way, the model will explore trajectories that are more familiar and will not have any constraints on the states. In leftmost image of figure 2.6, we can see that adding Denoising Autoencoders regularization significantly improves the algorithm performances in the environment. However, as stated by the authors, this method penalizes exploration for the algorithm which will lean towards trajectories experienced in the past. This might prevent the algorithm to improve its performances after a certain training time.

2.4.4. Preventing Compounding Errors

In Model-Based Reinforcement Learning, we can observe compounding errors in two different steps of the algorithms during planning: when approximating the value function (due to bootstrapping) and when generating trajectories with the learned model (due to its imperfections). Because of the recursive nature of MBRL approaches, a single error will generate another slightly bigger error in the next step and so on, leading to a failure in the learning process.

When using bootstrapping, the Q-value of each state-action pair is updated using equation 2.4. With this formula, we are updating the old value based on the difference between our previous estimate of the state action value pair and a new target estimate. This last new estimate is more accurate since it is computed using the observed reward r_t. Bootstrapping happens when we use an estimate to update another estimate (i.e, when using $\max_a Q(s_{t+1}, a_{t+1})$).

A possible approach to overcome compounding errors in the value function is using multi-step returns in the target estimate. This way, the model dynamics will diverge less [95], as we can see in figure 2.7. This is because when using n-step returns, the target estimate in equation 2.4 is computed as

$$\sum_{i=0}^{n} \gamma^i r_i + \gamma^{n+1} \max_a Q(s_{t+1}, a_{t+1}). \tag{2.21}$$

Note that when $n = \infty$, this becomes the definition of expected return (equation 2.1), as the second term will tend to 0. So, in the case of infinite-step returns we update our value estimate towards the true expected return, which would be ideal. However, in many cases we must settle for a finite value of n due to the environment complexity and this leads to a reduction in the divergence of the models instead of eliminating it completely.

When generating trajectories (rollouts) with the learned model, we also experience compounding errors: since the approximation is not exact, the model can generate transitions that do not exist in the environment creating *hallucinated* states [91, 97]. In turn, these non-existing states will be used as a starting point to

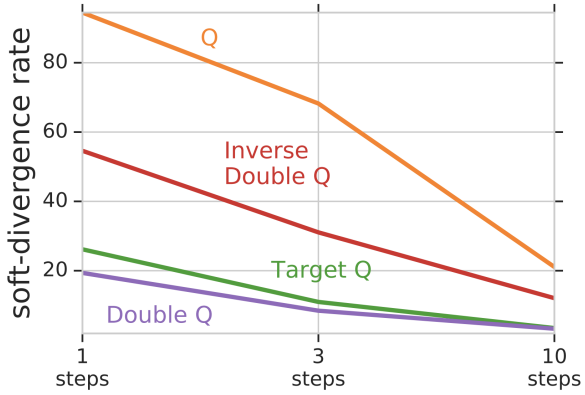


Figure 2.7: Experiments results from [95]. When increasing the multi-step returns size, all models experience a divergence rate reduction.

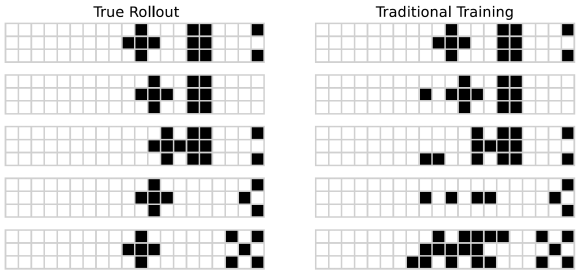


Figure 2.8: From [91]. In this game, the ball (cross) should hit the wall and bounce back. On the right image is represented the true sequence of frames. On the left is the sequence of states generated by the model: it can be seen how just one wrong pixel in the second frame can cause a disastrous chain of errors ending up in a completely different (and non-existing) state.

generate the subsequent transitions, compounding and misguiding the whole trajectory. A graphical example of this phenomenon is provided in figure 2.8.

[91] proposes a regularization technique to make the model more robust to compounding errors. This method, called *hallucinated replay*, consists in adding samples from the model to the training data set. More specifically, once a trajectory is generated from the environment, there is a chance that it will be replaced by an hallucinated duplicate: that is, one or multiple subsequent observations along the trajectory are substituted by the ones generated by the model for that same transition. With this approach, the model will be trained with some samples that are generated by itself, while the others will still come from the environment, giving the model a way to adjust its predictions.

However, increasing model prediction accuracy and robustness is not enough to completely overcome compounding errors in Dyna architectures. [31] study how the model mistakes compromise learning even when there is an effort to minimize them. Using longer rollouts can bring benefits since the model will experience unexplored states, but even if the model imperfections are reduced, errors will catastrophically compound as the rollout length grows.

A different approach is to change the planning direction [33]: when planning forward, the value of a real state might be updated taking into account misleading values of hallucinated states. However, in backward planning, after observing a transition $(s_t, a_t, r_{t+1}, s_{t+1})$ the model will be used to generate the transitions $(\hat{s}_{t-n}, \hat{a}_{t-n}, \dots, \hat{r}_t)$ that led to it and to update[1]

$$Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1}) \leftarrow Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1}) + \alpha(\hat{r}_{t-n} + \gamma \max_a Q(\hat{s}_{t-n}, a) - Q(\hat{s}_{t-n-1}, \hat{a}_{t-n-1})). \tag{2.22}$$

This way, the algorithm will update values for states generated by the model by using real and generated transitions. The damage from the compounding error is therefore reduced, since it would mislead the value of hallucinated states that will never be encountered in the environment.

Several works have studied the effects of modelling the *uncertainty* of the model and including it in the

[1] Equation 2.22 represents a single-step update: to obtain the equation for the multi-step approach, equation 2.21 can be adapted to the backward setting and plugged in equation 2.22.

planning and policy evaluation phases of the algorithm, to make learning and planning more robust to compounding errors [16, 19, 21, 27, 28, 48]. Probabilistic models naturally model their uncertainty: since they approximate a probability distribution function, by definition, they express the uncertainty of their output. Then, during planning, it will be possible to estimate how far off the approximated trajectory is when compared to executing the same actions in the environment.

A common planning method is to model the choice of each action along a trajectory as a function of the current state, that is modelling a policy $\pi : s_t \rightarrow a_t$. Alternatively, using a Model Predictive Control (MPC) approach, the algorithm optimizes with respect to the sequence of actions taken: given a set of actions $\{a_t, \dots, a_{t+n}\}$ and a starting state s_t, the algorithm computes a probability distribution over the trajectories $s_{t:t+n}$. At each time step, the algorithm applies the action maximising the expected reward. Since MBRL methods use the model to predict the evolution of states after n actions (i.e., over a horizon of length n), the uncertainty of a single prediction must be propagated over the same horizon during planning, when estimating the expected reward. Depending on the method used for state representation, there are several approaches in the literature to approximate uncertainty propagation. Deterministic methods take the mean prediction without relying on uncertainty estimation, particle methods generate Monte Carlo samples along the given trajectory, while parametric methods include Gaussian (Mixture) Models. Particle methods proved to be competitive with respect to parametric methods [47], both in terms of accuracy and computational cost while also having the advantage of not making assumptions on the modelled distribution.

Algorithm 4 PETS

Input: Dataset D initialized with data from a random exploration policy
Input: The horizon length H
for round $k \in \{1, 2, ..., K\}$ **do**
  Train a Probabilistic Ensemble dynamics model $\tilde{f}$ on D
  Initialize P starting states (particles) s_τ^p to the same initial state s_0, where $p \in \{1, 2, ..., P\}$
  for time step $t \in \{0, 1, ..., H\}$ **do**
    for sample $i \in \{1, 2, ..., N\}$ **do**
      Sample actions $a_{t:t+T} \sim \text{CEM}(\cdot)$
      Propagate state particles s_τ^p using TS and the ensemble $\tilde{f}$, along the sampled trajectory $a_{t:t+T}$
      Evaluate the trajectory according to the expected value $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(s_\tau^p, a_\tau)$
      Update CEM$(\cdot)$ distribution
    end for
    Select the optimal trajectory $a^*_{t:t+T}$ among the N samples and select the first action a_t^*
    $s_{t+1} = \tilde{f}(s_t, a_t^*)$
    $D \leftarrow D \cup (s_t, a_t^*, s_{t+1})$
  end for
end for

An example of architecture combining particle-based propagation and ensembles of parametric models to model uncertainty is PETS [16], summarized in Algorithm 4. The model incorporates uncertainty by using an ensemble of bootstrapped models. By using bootstrap aggregation (i.e., training each model on a sampled subset of the original training dataset), the variance of the model can be greatly reduced while also diminishing overfitting [12]. The authors train N models f_{θ_n}, where θ_n are the parameters for the nth model, that are then aggregated in a final model f such that $f = \frac{1}{N} \sum_{n=1}^{N} f_{\theta_n}$. To generate a trajectory, the PETS algorithm uses Trajectory Sampling. That is, at each time step t, the transition from a state s is carried out by one of the N bootstrapped models chosen according to a function $b(s, t)$. The model choice for Trajectory Sampling (TS) can therefore be expressed as a function of time, depending on the implementation. The authors explore two different TS strategies, namely TS1 and TS-∞. With TS1, the models are uniformly resampled at each time step, while in TS-∞ each trajectory is carried out by the same model. An advantage of the second approach is that the model can take into account two different kinds of uncertainty, *aleatoric* and *epistemic*, without fine-tuning additional hyperparameters. The former is the variance of the data itself (e.g., noise from the generating process), while the latter is the uncertainty of each model about the real dynamics functions (due to scarce data) and it is given by the disagreement between the models (Figure 2.9). The PETS algorithm was able to reach a performance similar to the one of state-of-the-art Model-Free methods, proving that taking uncertainty into account during learning and planning is a critical factor for MBRL methods.

[48] use model ensembles to prevent overfitting the model during planning: during policy validation the

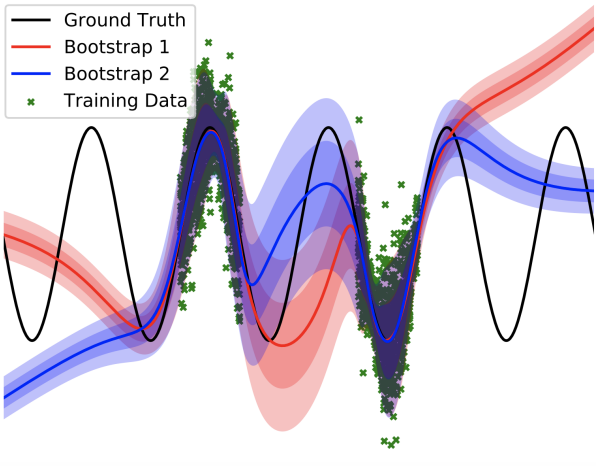


Figure 2.9: From [16]. We can observe how the predictions of the two models diverge in areas where less data is available (epistemic uncertainty). The variance of the prediction (red and blue shaded areas) models aleatoric uncertainty.

algorithm leverages the different models to evaluate the outcome of a certain policy on a diversified set of futures. In fact, Kurutach et al. take into account the ratio of models ϕ_k in which the policy π_θ improves

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}[\hat{\eta}(\theta_{new};\phi_k) > \hat{\eta}(\theta_{old};\phi_k)],$$

where K is the number of models used by the ensemble and $\hat{\eta}(\theta;\phi)$ is the function computing the performance of the policy π_θ on model ϕ. This ratio is used in the algorithm to evaluate when the policy has converged to a (local) optimum.

2.4.5. Objective Mismatch

MBRL algorithms consists of the following steps (cyclically repeated): gaining experience samples with the current policy, adjusting the environment model using the experience collected and improving the policy through planning with the approximated model. Due to this modular structure, in MBRL we can distinguish between *model accuracy* and *planning performance* [50]. The former defines how close the model is to the environment itself, while the latter quantifies the agent's performance in the environment. These two optimization problems can be addressed separately but, as shown by [50], it will negatively affect performances as the environment becomes more complex, creating an *objective mismatch*.

The MBRL approach assumes a strong correlation between the two losses used for the model fitting and planning steps: it is expected that when improving the model, so will the optimal policy computation. This assumption is inherited from System-Identification (SI) theory, where a model is fitted to a set of sampled trajectories from the environment and then employed to fulfill a certain task. However, SI approaches also rely on three more assumptions that grant a positive result from the optimization of two different losses, which do not hold true for MBRL [50]:

1. The presence of virtually infinite data: this is explicitly in contrast with MBRL where algorithms try to be data efficient (i.e., make use of less data from the environment and still be very accurate).

2. The collected trajectories cover the entire state-action space: this does not hold for MBRL, since models focus on specific tasks, and therefore will favour exploration of regions useful to its accomplishment.

3. The model resulting from SI approach is global and generalizable: this is prevented by the fact that MBRL models are usually biased towards the task they have to accomplish.

[50] experimentally show that the two losses optimized in MBRL algorithms are actually uncorrelated, originating the *objective mismatch* phenomenon. To confirm this assumption, the authors studied the impact of model accuracy on the final performance. By performing an adversarial attack on the dynamics model, they lowered the experienced rewards by a large amount when observing a drop in the validation likelihood. The attacked model resulted having a mildly increased accuracy with respect to the non-attacked model, but

computed a notably under-performing policy. Thus, the higher model accuracy does not imply an improvement in performance.

The objective mismatch has also been tackled with differentiable architectures [3, 64, 84]: this way, the planning loss can be backpropagated through the model as well. However, these solutions are not always possible to implement (e.g., due to intractability), and other methods rely on implementing heuristics to optimize the model dynamics with respect to the faced task.

[25] propose Value Aware Model Learning (VAML), a technique to link learning and planning in MBRL. The idea is that the policy optimization process should impact the approximation of the model. When just considering model estimation, the cost to optimize could be expressed as

$$c = \left| \int \left[P(dx'|x,a) - \hat{P}(x'|x,a) \right] V(x') dx' \right|,$$

where the distance between the transitions computed by the true model P and its approximation $\hat{P}$ is weighted by the value V of that same transition. A first problem with this cost function is that we do not know either the true value or probability function. Moreover, the cost function is a pointwise measure (since it is computed for a fixed value of x and a), while it would be best to consider a continuous function in the space $X \times A$. The authors propose a new loss to estimate the model while taking into account also the value function V of the task, given a dataset of samples $D_n = \{(X_i, A_i, X_i')\}_{i=1}^n$:

$$c_{2,n}^2(\hat{P}) = \frac{1}{n} \sum_{(X_i,A_i) \in D_n} \sup_{V \in F} \left| V(X_i') - \int \hat{P}(x'|X_i,A_i) V(x') dx' \right|^2,$$

where $\hat{P}$ is the transition probability function approximated by the model, assumed to be an exponential with features $\phi' : X \times A \times X \to \mathbb{R}^d$ and weights w. So $\hat{P} = \hat{P}_w$ such that

$$\hat{P}_w = \frac{\exp(\phi'(x'|x,a)^T w)}{\int \exp(\phi'(x''|x,a)^T w) dx''}. \tag{2.23}$$

The authors proved that the gradient of this cost function is, with respect to the parameters vector w

$$\nabla_w c_{2,n}^2(\hat{P}_w) = \frac{2B^2}{n} \sum_{i=1}^n \left[E_{X' \sim \hat{P}_w(|X_i,A_i)}[\phi(X')] - \phi(X_i') \right]^T \cdot \mathbf{Cov}_{X' \sim \hat{P}_w(|X_i,A_i)}(\phi(X'), \phi'(X'|X_i,A_i)), \tag{2.24}$$

where $\phi : X \to R^p$ is the feature mapping for the value function $V_\theta(x) = \phi^T(x)\theta$ and $\|\theta\|_2 \leq B$ (the features extracted with ϕ are assumed to be different from ϕ' ones). The first term of the gradient computes the difference between the value experienced and its expectation, whereas with with a regression approach we would have the difference between transition probabilities. The second term of the gradient weights the difference using the covariance between the transition probability and the state value.

Despite bringing awareness of the task in the model, by introducing a "value term" both in the cost function and its gradient, this approach also introduces new technical difficulties: the normalizing factor in equation 2.23 will not have a closed form in most of the cases, and computing the gradient of this loss is not trivial. The first problem can be solved by using Monte Carlo methods or with variational inference [53], while for the second the authors propose to estimate the gradient by using a few samples from the probability distribution, as in contrastive divergence learning [14].

[22] introduce Gradient Aware Model-Based Policy Search (GAMPS), mainly exploring the batch setting. When considering the batch setting, the agent learns the model by using only a dataset consisting of previously collected trajectories (following a known behavioural policy π_b) that is not extended through additional interactions with the environment. Policy-search algorithms then learn a parameterized policy π_θ, that maximizes the expected return. The gradient of the expected return, computed w.r.t. parameters θ, is defined by the Policy Gradient Theorem [89, 90] (see equation 2.9). Since in batch policy optimization the gradient is computed for a policy π_θ different from the behavioural policy π_b, the mismatch between the distributions is addressed by using importance sampling, re-weighting the experienced transitions according to the probability of observing them under policy π_θ. The importance weight for the transitions happening in trajectory τ from time t' to t'' is defined as

$$\rho_{\pi_\theta,\pi_b}(\tau_{t':t''}) = \prod_{t=t'}^{t''} \frac{\pi_\theta(a_t,s_t)}{\pi_b(a_t,s_t)}. \tag{2.25}$$

The gradient of the expected return, computed w.r.t. parameters θ, is defined by the Policy Gradient Theorem [89, 90]. The authors propose an improvement over the Policy Gradient Theorem (PGT), called Model-Value-based Gradient (MVG) by approximating the action-value function $Q(s,a)$ through the model $\hat{p}$

$$\nabla_\theta^{\text{MVG}} J(\theta) = \frac{1}{1-\gamma}\int_S\int_A \delta_\mu^{\pi,p}(s,a)\nabla_\theta \log\pi(a|s) \times Q^{\pi,\hat{p}}(s,a)\mathrm{d}s\mathrm{d}a. \tag{2.26}$$

This way, the variance is reduced with respect to the PGT, where $Q(s,a)$ is approximated using samples from the environment, and the model bias is limited to the Q-function estimate since the model is not used to sample transitions in $\delta_\mu^{\pi,p}(s,a)$. The authors proved that not all the state-action pairs are critical to compute a good estimate of the gradient: the agent should prioritize those that are likely to be experienced when following the policy from pairs with greater gradient-magnitude. In fact, the magnitude $||\nabla_\theta \log\pi(a|s)||$ can be seen as quantifying how much can the policy be improved in that area of the state-action space.

In GAMPS, presented in Algorithm 5, the agent can evaluate which dynamics of the environment are critical to learn with respect to the task and which are not: when learning the model, transition probabilities are re-weighted using the "awareness" of the environment given by the current agent's policy. More formally, after generating a dataset of transitions τ^i following a behaviour policy π_b, the model $\hat{p}$ is computed in the following way

$$\hat{p} = \arg\max_{\bar{p}\in\mathscr{P}} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\omega_t^i \bar{p}(x_{t+1}^i|x_t^i,a_t^i)$$

$$\omega_t^i = \gamma_t \rho_{\pi/\pi_b}(\tau_{0:t}^i)\sum_{l=0}^{t}||\nabla_\theta \log\pi(a_l^i|s_l^i)||_q,$$

where $\mathscr{P}$ is the class of models approximating the environment dynamics, π is the currently estimated policy and T^i is the length of the ith trajectory τ^i. The transitions are re-weighted using ω_t^i, which is defined by three components: the discount factor γ^t makes later transitions less important, $\rho_{\pi/\pi_b}(\tau_{0:t})$ is larger for transitions occurring often under current policy π and $\sum_{l=0}^{t}||\nabla_\theta \log\pi(a_l|s_l)||_q$ will favor the transitions with highest accumulated reward under the current policy.

Algorithm 5 Model-Value-based Policy Search

Input: Trajectory dataset D
Input: behavior policy π_b
Input: initialized parameters θ_0
Input: step size α

for k $\in \{0,...,K-1\}$ **do**
  $\diamond$ Learn model $\hat{p}$ using experience samples from D
  $\omega_t^i \leftarrow \gamma_t \rho_{\pi_{\theta_k}/\pi_b}(\tau_{0:t}^i)\sum_{l=0}^{t}||\nabla_\theta \log\pi_{\theta_k}(a_l^i|s_l^i)||_q$
  $\hat{p_k} \leftarrow \arg\max_{\bar{p}\in\mathscr{P}} \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\omega_t^i \bar{p}(x_{t+1}^i|x_t^i,a_t^i)$
  $\diamond$ Approximate the Q-value function using the model $\hat{p}_k$
  Generate M trajectories for each (s,a) using $\hat{p}_k$
  $\hat{Q}_k(s,a) \leftarrow \frac{1}{M}\sum_{j=1}^{N}\sum_{t=0}^{T_j-1}\gamma^t r(s_t^j,a_t^j)$
  $\diamond$ Improve the parameterized policy
  $\hat{\nabla} J(\theta_k) \leftarrow \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\gamma_t \rho_{\pi_{\theta_k}/\pi_b}(\tau_{0:t}^i)\nabla_\theta \log\pi_{\theta_k}(a_t^i|s_t^i) \times \hat{Q}_k(s_t^i,a_t^i)$
  $\theta_{k+1} \leftarrow \theta_k + \alpha_k \hat{\nabla}_\theta J(\theta_k)$
end for

While the previous approaches rely on the reward labels retrieved from the environment, [60] approach the objective mismatch problem in a self-supervised MBRL setting (i.e., without reward labels). In this setting, an MDP is defined as $M = \langle S, A, p, G, \gamma\rangle$, where G is the set of goal states s_g. Given a set of trajectories $[\tau_1,\ldots,\tau_N]$ where $\tau = [(s_0,a_0),(s_1,a_1),\ldots,(s_T)]$ and a distance measure $C : S\times S \to \mathbb{R}$, the MBRL agent must minimize the (expected) cost $C(s_T,s_g)$[2]. The intuition behind Nair et al. work can be described as follows.

[2]The approach and intuition can be easily extended to the reward-based setting.

In MBRL, when optimizing the model, the errors are uniformly distributed across the whole dynamics domain: this can lead to catastrophic performance, since the algorithm can exploit mistakes in certain areas of the domain during planning. However, the model does not have to be equally accurate on all transitions: the problem can be reduced to have a high accuracy in areas of the domain that are critical to accomplish the task, while dynamics unrelated to the problem can be approximated with lower precision. In fact, the authors state and prove the following theorem

Theorem 2. *Let a policy π be used to select N action sequences $a^i_{1:T}$ when interacting with the model. Let $c^*_i = E_{p,a^i_{1:T}}[C(s_T, s_g)]$ be the expected cost of the ith action sequence under the true dynamics of the system (unknown to the agent), and $\hat{c}^*_i = E_{p_\theta,a^i_{1:T}}[C(\hat{s}_T, s_g)]$ the expected cost under the learned model. Then the policy will remain ϵ-optimal, that is*

$$c^*_{i'} \leq c^*_1 + \epsilon, \quad i' = \arg\min_i \hat{c}_i, \tag{2.27}$$

if the following two conditions are met:

1. *the model prediction error on the best action sequence $a^1_{1:T}$ is bounded such that $|c^*_1 - \hat{c}_1| < \epsilon$,*
2. *the errors of sub-optimal actions sequences $a^i_{1:T}$ are bounded by $|c^*_i - \hat{c}_1| < (c^*_i - c^*_1) - \epsilon$.*

These two conditions imply that the model error should be low for good (ϵ-optimal) trajectories, but that a higher error is acceptable for the others. So, while usually the model learning redistributes the model error uniformly (along all trajectories), the authors propose to stir the model to have accurate predictions along relevant trajectories. With an accurate cost function $C(s_T, s_g)$, weighting transitions in the training loss by the inverse cost would encourage the model to focus on low-cost trajectories (that would carry a greater weight). However, this approach would not be as effective for high-dimensional states (e.g., video prediction), since the cost function would be very sparse. Therefore, the authors introduce goal-aware prediction (GAP). That is, a technique through which the model reconstructs the difference between the goal and the next state $p((s_g - s_{t-1})|s_t, s_g, a_t)$, by setting the final state of a trajectory as its goal. This way the model only needs to learn components useful for the task, since it is approximating only the difference between states, and it will predict more accurately states that are closer to the goal.

2.5. Method

In previous sections we went through the challenges brought by introducing a model of the dynamics of an MDP. In tabular methods like R-MAX, 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 state-action spaces, we need to rely on function approximation to represent transition probabilities and value functions. However, powerful function approximators like Neural Networks are not guaranteed to converge to the correct probability distribution and the RL agents might not converge to an optimal policy. Robust Markov Decision Processes grant the possibility to deal with the uncertainty in the MDPs dynamics and reward function by casting the decision problem as a two-player game. With RMDPs, we can mitigate the impact of the incorrect dynamics on the policy learned by the agent, which will be more robust to the model errors and closer to the optimal policy.

2.5.1. Robust Markov Decision Processes

In Robust Markov Decision Processes (RMDPs), the transition probability P is considered to be an element of the convex set $\mathbb{P}$, called *uncertainty set*. We can define different structures for the uncertainty set: s-rectangular, (s,a)-rectangular and nonrectangular.

Definition 6. We define s-rectangular uncertainty set as the Cartesian product of independent subsets $\mathbb{P}_s \subseteq \mathbb{R}^{|S|\times|A|}_+$ for each $s \in S$

$$\mathbb{P} = \bigtimes_{s\in S} \mathbb{P}_s \tag{2.28}$$

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 $(s,a) \in S \times A$

$$\mathbb{P} = \bigtimes_{(s,a)\in S\times A} \mathbb{P}_{(s,a)} \tag{2.29}$$

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 [106]. (s,a)-rectangular uncertainty sets are a specific instance of s-rectangular sets [23, 106], where an optimal robust policy can be computed with value iteration and it exists an optimal policy that is Markovian, stationary and deterministic [32, 62]. Uncertainty sets that are neither s- nor (s,a)-rectangular are said to be nonrectangular, for which exact robust policy evaluation and improvement is intractable.

In the RMDPs framework, we assume each transition probability to be chosen adversarially from the uncertainty set $\mathbb{P}$. So, an optimal policy can be obtained by solving

$$\pi^* = \arg\max_{\pi \in \Pi} \min_{P \subset \mathbb{P}} v_{\pi,P} \tag{2.30}$$

where $v_{\pi,P}$ is the value function under policy π and following the adversarial transitions P.

2.5.2. Robust Ensemble AdversariaL (REAL) MBRL

When using function approximation to represent the environment dynamics, we cannot guarantee the convergence of the model to the true distribution. In this section, we propose Robust Ensemble AdversariaL (REAL) MBRL, a Model-Based algorithm leveraging the RMDPs framework to compute a policy more robust to modeling errors. We will introduce two methods: the first approach will use a greedy adversary to choose the worst transitions, while the second algorithm will have an ϵ-random adversarial agent to increase the adversarial exploration.[3]

In REAL, we approximate the environment dynamics using an ensemble of N models $M_\psi = \{M_{\psi_1}, \dots, M_{\psi_N}\}$ where $p(s'|s,a) = \frac{1}{N}\sum_{i=1}^{N} M_{\psi_i}(s,a)$. By leveraging equation 2.30 and definition 7, we can cast the policy improvement problem as a two-player game. In fact, we can define a (s,a)-rectangular uncertainty set on the ensemble of models so that $\mathbb{M} = \bigtimes_{(s,a)\in S\times A} \mathbb{M}_{(s,a)}$. The main player following a policy π will have as objective to maximise the expected return under the current dynamics $M \subset \mathbb{M}$, which is

$$\arg\max_{\pi} v_{\pi,M}(s), \qquad \forall s \in S. \tag{2.31}$$

The adversary will choose the dynamics to minimise the expected return of the main player, so

$$\min_{M \subset \mathbb{M}} v_{\pi,M}(s), \qquad \forall s \in S. \tag{2.32}$$

By representing the behaviour of the adversary as a policy $\xi : S \times A \to \mathbb{M}$, we obtain the equivalent definition of the adversarial optimization problem

$$\min_{\xi} v_{\pi,\xi}(s), \qquad \forall s \in S. \tag{2.33}$$

By combining equations 2.30, 2.31 and 2.33 we obtain the two-player game

$$\pi^* = \arg\max_{\pi} \min_{\xi} v_{\pi,\xi}(s) \qquad \forall s \in S. \tag{2.34}$$

To compute the optimal policy, our reinforcement learning agent continuously iterates between a model-learning and policy improvement phase. At each iteration t, when performing model-learning, we fit our approximated model to samples gathered in the true environment by using the current policy π_t. We then improve the policies π_t and ξ_t by generating new transition samples with the approximated dynamics, resulting in the new policies π_{t+1} and ξ_{t+1}. In algorithm 6, we provide a generic pseudocode for the algorithm. The IMPROVE function uses the main and adversarial policy to perform rollouts in the approximated model, where each transition is performed by the model chosen by the adversary. The gathered samples are then used to improve both policies.

To encourage adversarial exploration, we propose a version of our algorithm involving an ϵ-random adversary. The ϵ-random adversary will select a random action with probability ϵ. With probability $(1-\epsilon)$, the adversary will exploit what it has learned so far and choose the action according to the policy ξ_ω. When $\epsilon = 0$, we obtain the REAL algorithm with a greedy adversary always choosing the action according to the policy.

[3] Notice that the former case is a specific case for the latter, where $\epsilon = 0$.

Algorithm 6 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_i for adversary ξ_ω
Input: number of iterations K

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

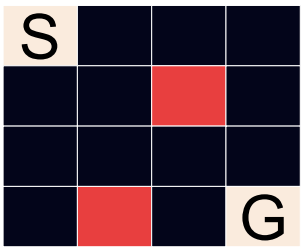


Figure 2.10: An illustration of the 4x4 map used in our experiments with the Frozen Lake environment. The S represents the starting point, while the G is the goal state. The red tiles represent holes in the gridworld which will determine the failure of the task for the agent. Finally, black tiles represent the "frozen" tiles, on which the agent can safely step.

2.6. Experiments and results

In this section, we will describe the experiments performed to evaluate our REAL algorithm. We focused on three environments, namely: Frozen Lake, Cartpole and Pendulum. All these environments are part of the OpenAI Gym suite. Frozen Lake is a grid-world environment, where the agent has to learn how to reach the goal state by moving towards the four cardinal directions. In Cartpole, a pole is connected to a cart moving on a frictionless track: the pole is set upright at the beginning of each episode, and the agent has to learn how to keep it balanced and prevent it from falling down. In the Pendulum environment, the agent can learn how to swing a pendulum up (and keep it upright) by applying the correct amount of force to it.

2.6.1. Frozen Lake

In the Frozen Lake environment, the agent moves the character on a gridworld. The character can move towards the four cardinal directions: up, down, right and left. The objective is to reach the goal cell, situated on the opposite side of the map. Some cells of the gridworld are "holes" that when entered will result in the termination (and failure) of the episode. An illustration of the environment is presented in Figure 2.10. The transition function of the environment is stochastic and upon taking an action a, the agent might slip and move in one of the perpendicular directions with a non-zero probability p, hence the name Frozen Lake[4]. In our experiments, the probability to move in one of the perpendicular directions is 0.2 and the probability to move in the correct one is 0.6.

The low complexity of this task allows us to properly evaluate the efficacy of our algorithm, by examining in detail the moves of both the main player and the adversary. To increase the interpretability of our method, we are using Q-learning to plan and compute the optimal policy. Through a tabular planning method we can observe and evaluate the choices of our agents, since we have access to the value estimate for each state-action pair.

[4]The dynamics can also be set to be deterministic (i.e., $p = 0$).

Parameter	Value
Buffer size $\|B\|$	1000
# environment samples per iteration	100
# model episodes per iteration	5000
Model rollout length L	min(100, termination function)
Discount rate 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 2.1: Algorithm hyperparameters for the Frozen Lake environment

Algorithm details

In the Frozen Lake environment, we approximate the dynamics using an ensemble M_ψ where $|M_\psi| = 3$. We use a probabilistic model, estimating the probability over the next state $p(s'|s,a)$. In other words, we define the dynamics learning problem as a classification task, where the output class is the index of the next state s'. Since the state-action space is small, we can afford to use tabular methods to perform planning and improve the policies π_θ and ξ_ω. For our experiments, we used Q-Learning and we improved the policies by following the update rule

$$Q^{\text{new}}(s_t,a_t) = Q(s_t,a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1},a_t) - Q(s_t,a_t)),$$

where α is the learning rate, γ is the discount factor and s_t,a_t,r_t are the state, action and reward gathered at time t. Note that for the adversarial agent we would substitute s_t with $s_t^{\text{adv}} = (s_t,a_t)$ and a_t with $a_t^{\text{adv}} \in [1,\dots,|M_\psi|]$. When gathering samples from the true environment we performed ϵ-greedy exploration with a fixed ϵ_π.[5] In table 2.1 we provide a more detailed insight on the algorithm hyperparameters.

Results

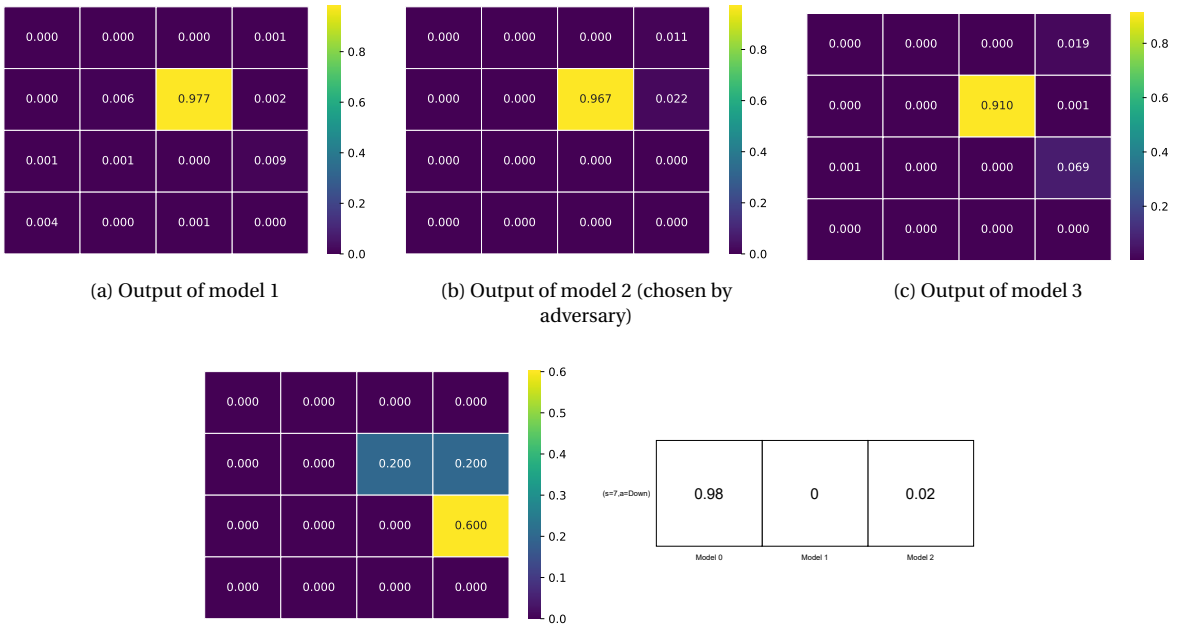
Since the Frozen Lake environment has a small state-action space, and we are using tabular methods to compute the optimal policies, we can inspect what is being learned by the algorithm more in detail. First, we focus on what the model and the adversary are learning. In figure 2.11a, 2.11b and 2.11c are presented 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 2.11d, we can see that they are far from correctly representing the correct transition. When examining the Q-values of the adversary (Figure 2.11e), we observe that at this stage, it has learned that choosing model 2 leads to the worst outcome for the main player. This choice can be intuitively explained by looking at the single transition probabilities. Despite all three models predicting with a high probability that the agent would fall into a hole (worst possible outcome), model 2 is the only one that does not allow the agent to get closer to the goal tile.

By examining the average return of our algorithm, we can observe that for different values of ϵ the learning process is quite similar, except for $\epsilon = 0.6$, where the average return is significantly lower up until 4k samples. We chose to compare the $\epsilon = 0.3$ agent to the plain Model-Based baseline since it has a more stable learning curve when compared to the others. The results are reported in Figure 2.12 and Figure 2.13. We can see that, in the first few iterations, the presence of an adversary enables the agent to learn a policy that is more robust to model errors and achieves better average results. Since a faster improvement of the dynamics model could cause the increased average return, we inspect the average cosine similarity of the ensemble predicted distributions. We compared it for each instance of REAL to the plain model-based ensemble in Figure 2.14. We can see that all models improve at the same pace, and the adversary is the only discriminant between the methods helping to compute a more robust policy.

2.6.2. Pendulum

Through this experiment, we aim to show that our algorithm can scale to more complex environments with continuous observation and action spaces.

[5] We named it ϵ_π to distinguish it from the adversarial ϵ.



(a) Output of model 1

(b) Output of model 2 (chosen by adversary)

(c) Output of model 3

(d) True transition probabilities

(e) Adversary (normalised) Q-values

Figure 2.11: Predicted distribution $p(s'|s = 7, a =$ "Down") according to each model in the ensemble and Q-values for the adversarial agent, after collecting 900 samples from the environment. Note that state enumeration starts from 0.

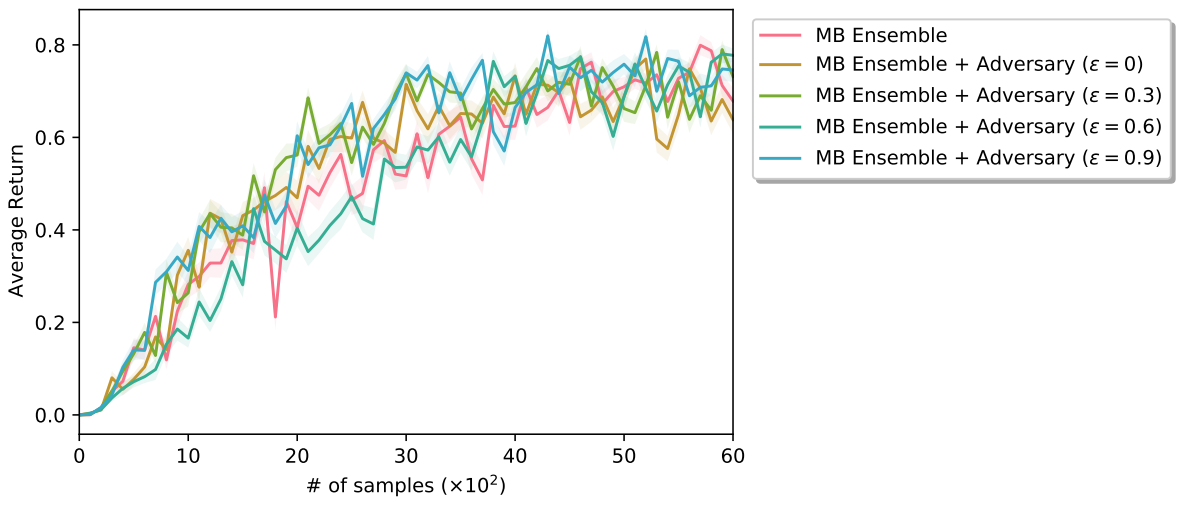


Figure 2.12: Comparison of REAL with $\epsilon \in \{0, 0.3, 0.6, 0.9\}$ in the Frozen Lake environment. Bold lines represent the average return over 10 runs, shaded areas evidence the standard error.

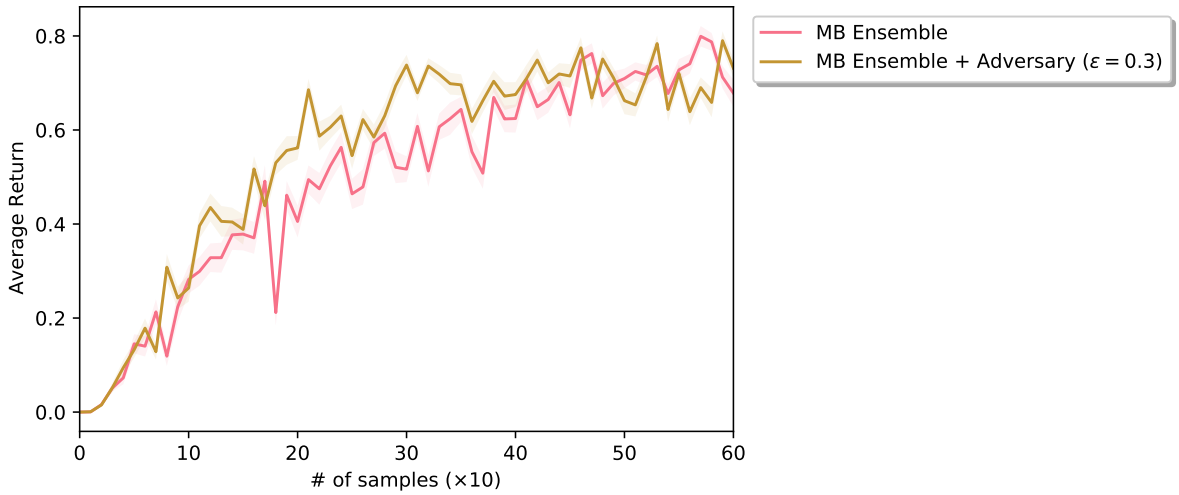


Figure 2.13: Comparison between REAL with $\epsilon = 0.3$ and the vanilla Model-Based baseline in the Frozen Lake environment.

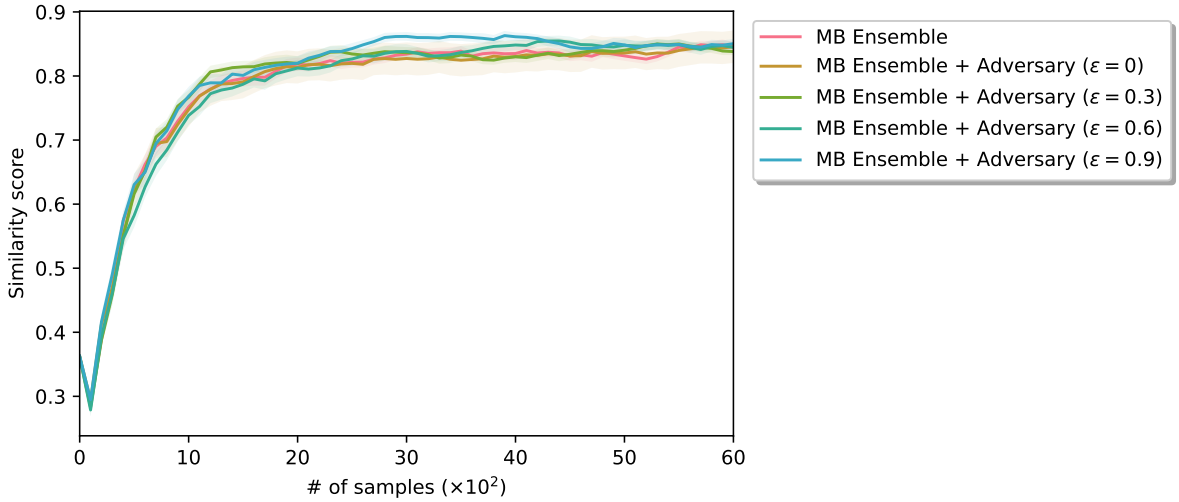


Figure 2.14: Average cosine similarity between the outputs of the models and the true probability vectors in the Frozen Lake environment.

In this 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. In the Pendulum environment, transitions are deterministic, observations $o \in \mathbb{R}^3$ and actions $a \in \mathbb{R}$. Since this environment has a continuous state-action space, it is not feasible to employ a tabular planning algorithm as we did in Frozen Lake: we decided to compute the optimal policy using Proximal Policy Optimization (PPO), a policy gradient algorithm. Policy Gradient methods model the policy directly, through a parameterised function π_θ, and optimize it according to the Policy Gradient Theorem (2.9). PPO imposes a KL-divergence constraint to limit the size of the policy updates. 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. The pseudocode of the algorithm is presented in Algorithm 7.

Algorithm 7 Robust Ensemble AdversariaL (REAL) MBRL

Input: Empty dataset D
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_i for adversary ξ_ω
Input: number of iterations K

for k $\in \{0, ..., K\}$ **do**
  $\diamond$ Collect new observations from the environment
  $D \leftarrow$ COLLECT(π_θ)
  $\diamond$ Update models using the extended dataset
  $\psi_i \leftarrow$ TRAIN_MODEL(M_{ψ_i}, D)
  $\diamond$ Update adversary using "inverse" REINFORCE
  $L = \frac{1}{T} \sum_t^T G_t ln(\xi_\omega(M_{i,t}|s_t))$
  $\omega \leftarrow \omega + \alpha \nabla_\omega L$
  $\diamond$ Update π_θ carrying out transitions with an adversarially chosen model
  $\pi_\theta \leftarrow$ PPO($\pi_\theta, M_\psi, \xi_\omega$)
end for

Algorithm details

For the Pendulum environment we approximate the domain dynamics using an ensemble M_ψ where $|M_\psi| = 3$. The model is deterministic, so it directly approximates the output state of the transition. 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. A more detailed view of the algorithm hyperparameters is presented in Table 2.2.

Results

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 2.15, we can see the average performance of each agent over 10 different runs. 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 2.16 we compare the $\epsilon = 0$ REAL agent with the Model-Based baseline. We can

Parameter	Value	
	Cartpole	Pendulum
Buffer size $\|B\|$	1500	
# environment samples K	1000	
# model samples	20000	
Model rollout length L	min(300, TF)	min(200, 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 2.2: Algorithm hyperparameters for the Cartpole and Pendulum environments. TF stands for Termination Function.

see that the adversarial choice of the model enables for more efficient early planning.

2.6.3. Cartpole

In the Cartpole environment, a pole is attached to a joint on a cart. The goal is to keep the pole upright and prevent it from falling down by moving the cart along the cart track. The action space is discrete: the agent can apply a force of $+1$ or -1 to the cart, moving it right or left. The observation space is continuous, where $o \in \mathbb{R}^4$. As for the Pendulum environment, we used a simplified version of PPO to improve the policy and REINFORCE to train the adversary agent.

Algorithm details

For the Cartpole environment, we use the same approach as we do for the Pendulum one. We use a deterministic model where $|M_\psi| = 3$ to approximate the state transitions, PPO to improve the policy of the main player and REINFORCE for the adversarial policy. When gathering new transition samples with the learned model, we use a fixed horizon of length L to reduce the effect of compounding errors on the predicted trajectory. To mitigate the negative impact of using a fixed horizon on the planning step, the starting state of model rollouts is sampled from the buffer B with probability δ. In table 2.2, we provide a more in-depth view of the hyperparameters.

Results

We evaluate the policy robustness to model errors by examining the average return of different algorithm instances over multiple runs. We consider instances of the REAL agent with $\epsilon \in \{0.3, 0.6, 0.9\}$. The performance of the algorithm on the Cartpole environment is presented in Figure 2.17. We can observe that using a ϵ-random adversary with $\epsilon = 0$ results in a policy outperforming the other instances in the first iterations. When compared to the vanilla Model-Based agent, our method achieves a consistently better performance, as presented in Figure 2.18.

2.7. Conclusions

Model-Based Reinforcement Learning (MBRL) algorithms solve sequential decision-making problems, usually formalized as Markov Decision Processes, using a model of the environment dynamics to compute the optimal policy. MBRL algorithms have been an appealing alternative to Model-Free methods due to their potential sample efficiency but have been successful only in simple domains. Having a model to generate new samples for planning reduces the number of interactions with the environment, but it brings new challenges to the research community. When dealing with large state-action spaces, algorithms rely on complex function approximators that are not guaranteed to converge to the correct dynamics. As a result, the convergence of the policy to an optimal solution is also not guaranteed. In this thesis, we investigated a new approach leveraging model imprecisions to learn a policy more robust to approximation errors.

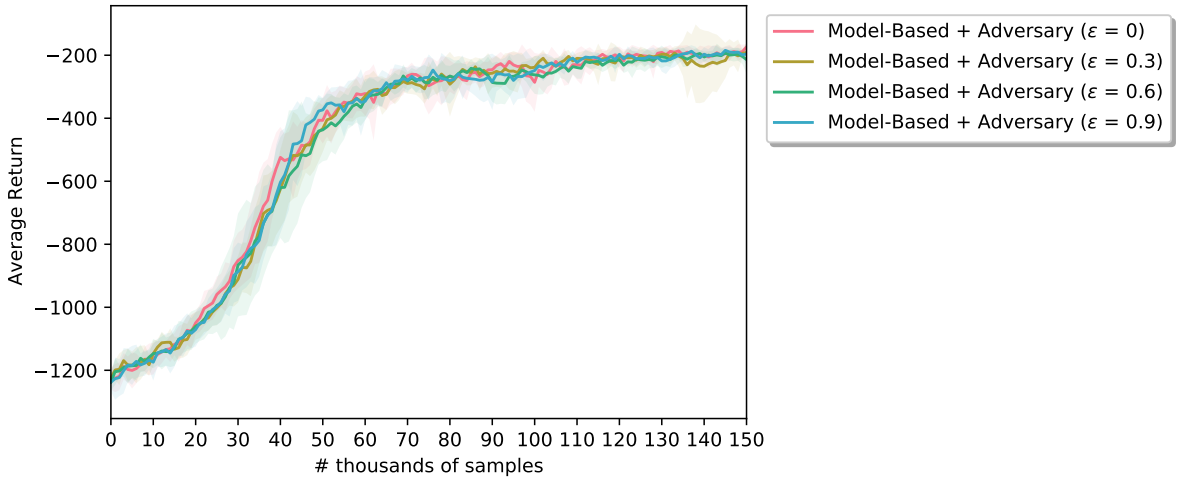


Figure 2.15: Comparison of the instances of REAL with $\epsilon \in \{0.3, 0.6, 0.9\}$ in the Pendulum environment. We can observe that for $\epsilon = 0$, the agent learns more efficiently at first, and then converges to the same average return as the other instances. Bold lines represent the average return over 10 runs, shaded areas evidence the standard error.

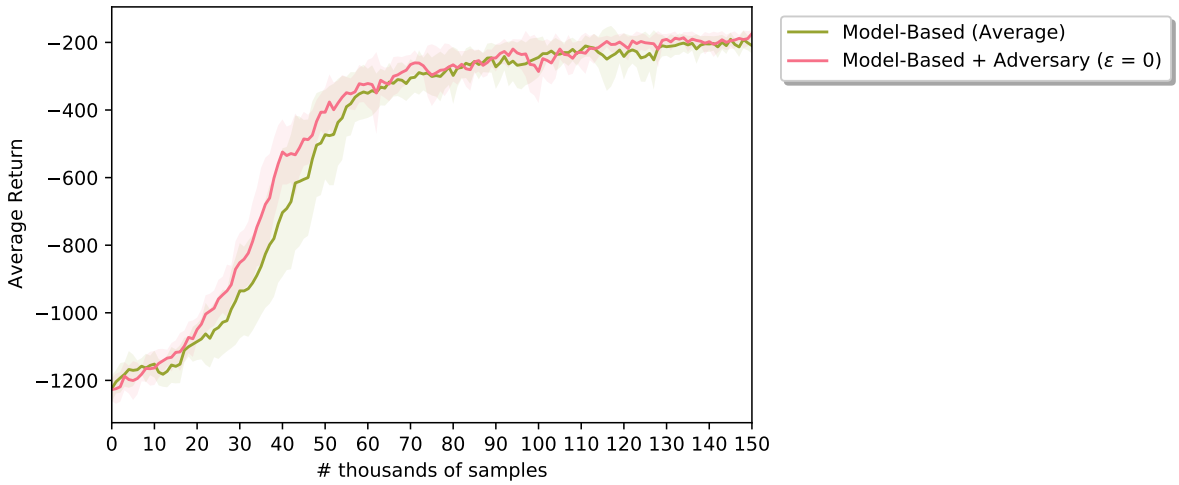


Figure 2.16: Comparison of the Model-Based algorithm and the REAL instance with $\epsilon = 0$ for the Pendulum environment.

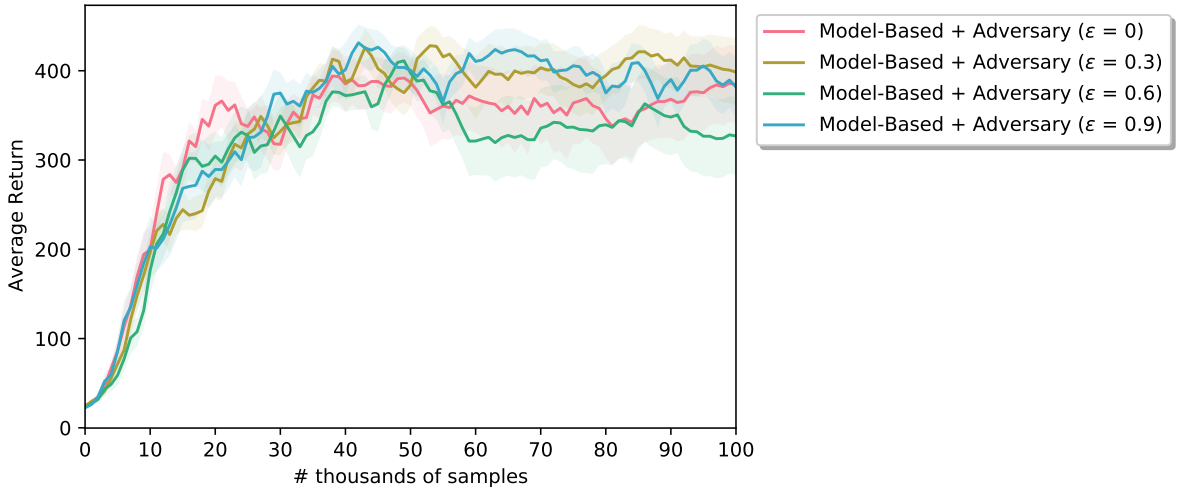


Figure 2.17: Comparison of the instances of REAL with $\epsilon \in \{0.3, 0.6, 0.9\}$ in the Cartpole environment. We can observe that for $\epsilon = 0$, the agent learns more efficiently at first.

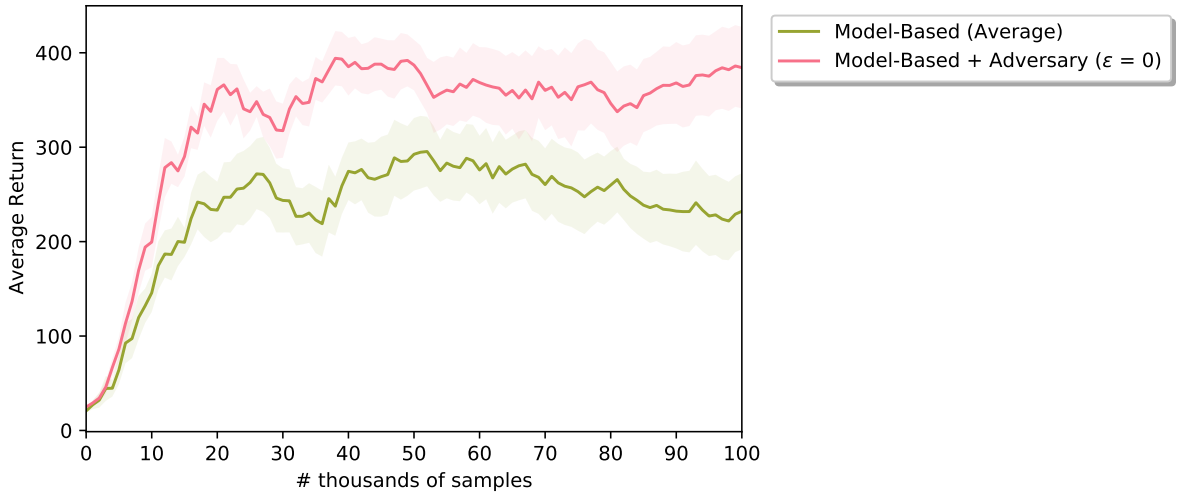


Figure 2.18: Comparison of the Model-Based algorithm and the REAL instance with $\epsilon = 0$ for the Cartpole environment.

Before diving into the details of our method, to understand the consequences of an imperfect model, we went through two important problems of MBRL methods and some of the most recent techniques addressing them. Model errors prevent MBRL from being a feasible solution in real applications with very complex environments also because prediction mistakes can be exploited by the algorithm during planning, overestimating the expected reward of the computed policy and leading to sub-optimal results when executed in the real environment. In Section 2.3 we discuss the problem of model accuracy and some of the techniques commonly employed to increase it. Since in MBRL algorithms the model is used to generate new experience for planning and improving the policy, it is desirable for its predictions to be as close as possible to the real environment dynamics. A first way to improve the prediction accuracy is to choose a model that can appropriately approximate the environment (Section 2.3.1). However, the dynamics domain may be very complex to represent and the samples provided to the model might not be enough to cover it entirely (curse of dimensionality): this would lead to sub-optimal performance when acting in *unobserved* areas of the domain, despite the model being very accurate in predicting *observed* dynamics. This is why researchers have been exploring other techniques to improve the domain approximation in the Model-Based setting. For instance, with abstraction and representation learning (Section 2.3.2), the domain space dimension is reduced so that it is easier to learn with a smaller amount of data (an approach similar to dimensionality reduction). Also, by choosing an appropriate exploration strategy (Section 2.3.3), the algorithm can collect samples more efficiently and represent a wider area of the domain. In Section 2.4, we discussed how faults in dynamics approximation can lead to disastrous results when acting (i.e., model exploitation) and how these effects can be mitigated. Choosing a different planning direction or horizon length (sections 2.4.1 and 2.4.2) and regularization (Section 2.4.3) have been proved to be successful to reduce the impact of model errors on planning (Section 2.4.4). With Section 2.4.5 we covered the problem of *objective mismatch*, which has only recently catch researchers attention: this problem originates from the fact that the two losses used in MBRL (one for learning the model, and one for planning) are independently optimized, while they are in fact closely related.

Based on the previous literature, we decided to approach the problem of learning a policy robust to model errors from a game-theoretical point of view. By applying the Robust MDPs theory to Ensemble-Based MBRL, we derived Robust Ensemble AdversariaL (REAL) MBRL, a formulation of the problem where the adversarial player should choose the worst model in the ensemble for carrying out each transition. We also extended this approach with an ϵ-random adversary, exploring the action space more than the greedy one.

To test our algorithm, in section 2.6 we evaluated it on different environments from the OpenAI gym suite. In every task, REAL was able to outperform the vanilla (Ensemble) Model-Based agent. In the Gridworld environment, due to the small state-action space, we were able to evaluate and interpret the decisions taken by the adversarial agent, proving that it was learning to contrast the main player. Through the Gridworld environment, we were able to show that the model accuracy improves at (more or less) the same rate for every instance of the algorithm, including the vanilla Model-Based one: this means that the only feature distinguishing them and leading to policy robustness is the adversarial agent.

Bibliography

[1] Abbeel, P., Ganapathi, V., and Ng, A. (2005). Learning vehicular dynamics, with application to modeling helicopters. *Advances in Neural Information Processing Systems*, 18:1–8.

[2] Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning.

[3] Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable mpc for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 31:8289–8300.

[4] Andreae, J. H. (1966). *Learning machines: a unified view*. Standard Telecommunications Laboratories.

[5] Azizzadenesheli, K., Yang, B., Liu, W., Lipton, Z. C., and Anandkumar, A. (2018). Surprising negative results for generative adversarial tree search. *arXiv preprint arXiv:1806.05780*.

[6] Bagnell, J. A. and Schneider, J. (2003). Covariant policy search.

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

[8] Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

[9] Berteskas, D. (1987). Dynamic programming: deterministic and stochastic models.

[10] Boney, R., Di Palo, N., Berglund, M., Ilin, A., Kannala, J., Rasmus, A., and Valpola, H. (2019). Regularizing trajectory optimization with denoising autoencoders. In *Advances in Neural Information Processing Systems*, pages 2859–2869.

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

[12] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[13] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

[14] Carreira-Perpinan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. In *Aistats*, volume 10, pages 33–40. Citeseer.

[15] Chelu, V., Precup, D., and van Hasselt, H. P. (2020). Forethought and hindsight in credit assignment. *Advances in Neural Information Processing Systems*, 33.

[16] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

[17] Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

[18] Curi, S., Berkenkamp, F., and Krause, A. (2020). Efficient model-based reinforcement learning through optimistic policy search and planning. *Advances in Neural Information Processing Systems*, 33.

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

[20] Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2013). Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423.

[21] Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., and Udluft, S. (2016). Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*.

[22] D'Oro, P., Metelli, A. M., Tirinzoni, A., Papini, M., and Restelli, M. (2020). Gradient-aware model-based policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3801–3808.

[23] Epstein, L. G. and Schneider, M. (2003). Recursive multiple-priors. *Journal of Economic Theory*, 113(1):1–31.

[24] Fairbank, M. and Alonso, E. (2012). Value-gradient learning. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[25] Farahmand, A.-m., Barreto, A., and Nikovski, D. (2017). Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494.

[26] François-Lavet, V., Bengio, Y., Precup, D., and Pineau, J. (2019). Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3582–3589.

[27] Fröhlich, F., Theis, F. J., and Hasenauer, J. (2014). Uncertainty analysis for non-identifiable dynamical systems: Profile likelihoods, bootstrapping and more. In *International Conference on Computational Methods in Systems Biology*, pages 61–72. Springer.

[28] Gal, Y., McAllister, R., and Rasmussen, C. E. (2016). Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 25.

[29] Grancharova, A., Kocijan, J., and Johansen, T. A. (2008). Explicit stochastic predictive control of combustion plants based on gaussian process models. *Automatica*, 44(6):1621–1631.

[30] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

[31] Holland, G. Z., Talvitie, E. J., and Bowling, M. (2018). The effect of planning shape on dyna-style planning in high-dimensional state spaces. *arXiv preprint arXiv:1806.01825*.

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

[33] Jafferjee, T., Imani, E., Talvitie, E., White, M., and Bowling, M. (2020). Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*.

[34] Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(4).

[35] Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer.

[36] Jie, T. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008.

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

[38] Kalweit, G. and Boedecker, J. (2017). Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206.

[39] Kamthe, S. and Deisenroth, M. (2018). Data-efficient reinforcement learning with probabilistic model predictive control. In *International Conference on Artificial Intelligence and Statistics*, pages 1701–1710. PMLR.

[40] Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3):193–208.

[41] Ko, J., Klein, D. J., Fox, D., and Haehnel, D. (2007). Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proceedings 2007 ieee international conference on robotics and automation*, pages 742–747. IEEE.

[42] Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE.

[43] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

[44] Kocsis, L., Szepesvári, C., and Willemson, J. (2006). Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1.

[45] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

[46] Kumar, V., Todorov, E., and Levine, S. (2016). Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE.

[47] Kupcsik, A., Deisenroth, M., Peters, J., and Neumann, G. (2013). Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 27.

[48] Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

[49] Lai, M. (2015). Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*.

[50] Lambert, N., Amos, B., Yadan, O., and Calandra, R. (2020). Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*.

[51] Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

[52] Lim, S. H., Xu, H., and Mannor, S. (2013). Reinforcement learning in robust markov decision processes. *Advances in Neural Information Processing Systems*, 26:701–709.

[53] MacKay, D. J. and Mac Kay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

[54] Mania, H., Tu, S., and Recht, B. (2019). Certainty equivalence is efficient for linear quadratic control. In *Advances in Neural Information Processing Systems*, pages 10154–10164.

[55] McAllister, R. and Rasmussen, C. E. (2017). Data-efficient reinforcement learning in continuous state-action gaussian-pomdps. In *Advances in Neural Information Processing Systems*, pages 2040–2049.

[56] Mishra, N., Abbeel, P., and Mordatch, I. (2017). Prediction and control with temporal segment models. *arXiv preprint arXiv:1703.04070*.

[57] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[58] Moerland, T. M., Broekens, J., and Jonker, C. M. (2020). Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

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

[60] Nair, S., Savarese, S., and Finn, C. (2020). Goal-aware prediction: Learning to model what matters. *arXiv preprint arXiv:2007.07170*.

[61] Nguyen-Tuong, D., Peters, J., and Seeger, M. (2008). Local gaussian process regression for real time online model learning. *Advances in neural information processing systems*, 21:1193–1200.

[62] Nilim, A. and El Ghaoui, L. (2005). Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798.

[63] Nouri, A. and Littman, M. L. (2010). Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning*, 81(1):85–98.

[64] Okada, M., Rigazio, L., and Aoshima, T. (2017). Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*.

[65] Osband, I. and Van Roy, B. (2017). Why is posterior sampling better than optimism for reinforcement learning? In *International Conference on Machine Learning*, pages 2701–2710. PMLR.

[66] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[67] Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*, volume 10, pages 1607–1612. Atlanta.

[68] Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

[69] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR.

[70] Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137.

[71] Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2016). Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

[72] Rajeswaran, A., Mordatch, I., and Kumar, V. (2020). A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, pages 7953–7963. PMLR.

[73] Rao, K., Whiteson, S., et al. (2012). V-max: tempered optimism for better pac reinforcement learning. In *AAMAS*, pages 375–382.

[74] Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

[75] Ross, S. and Bagnell, J. A. (2012). Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*.

[76] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

[77] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

[78] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

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

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

[81] Shyam, P., Jaśkowski, W., and Gomez, F. (2019). Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788. PMLR.

[82] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[83] Silver, D., Sutton, R. S., and Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pages 968–975.

[84] Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks. *arXiv preprint arXiv:1804.00645*.

[85] Strens, M. (2000). A bayesian framework for reinforcement learning. In *ICML*, volume 2000, pages 943–950.

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

[87] Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier.

[88] Sutton, R. S. (1995). On the virtues of linear learning and trajectory distributions. In *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference*, page 85.

[89] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[90] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

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

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

[93] Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE.

[94] van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable approximations to mdp homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963*.

[95] Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*.

[96] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[97] Van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, pages 14322–14333.

[98] Venkatraman, A., Boots, B., Hebert, M., and Bagnell, J. A. (2014). Data as demonstrator with applications to system identification. In *ALR Workshop, NIPS*.

[99] Venkatraman, A., Capobianco, R., Pinto, L., Hebert, M., Nardi, D., and Bagnell, J. A. (2016). Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics*, pages 703–713. Springer.

[100] Venkatraman, A., Hebert, M., and Bagnell, J. A. (2015). Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[101] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

[102] Von Stackelberg, H. (2010). *Market structure and equilibrium*. Springer Science & Business Media.

[103] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[104] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[105] Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *arXiv preprint arXiv:1506.07365*.

[106] Wiesemann, W., Kuhn, D., and Rustem, B. (2013). Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183.