Adversarial Curiosity for Model-Based Reinforcement Learning

Gleb Shevchuk
AA228

Abstract—**Curiosity can improve exploration and reduce sample inefficiency in reinforcement learning. Yet, while curiosity-based exploration has seen a surge in popularity, it is difficult to extend current formulations to model-based reinforcement learning. In response, we present an adversarial method for curiosity to improve model-based learning. By formulating action selection as a minimax problem between a learner and an adversarial teacher, we attempt to improve exploration and increase the final performance of model-based algorithms. We show preliminary progress by visualizing exploration and charting final performance on several simulated environments.**

I. Introduction

Reinforcement learning is plagued by sample inefficiency. While model-free reinforcement learning methods like Asynchronous Actor Critic [1] and Proximal Policy Optimization [2] have led to sustained improvement on common benchmarks, they usually require enormous amounts of data collection. If algorithms require millions or billions of examples to learn a task, it becomes impossible to roll these systems out in the real world.

One alternative to pure model-free reinforcement learning is to explicitly learn our environment, an approach known as model-based reinforcement learning. Once we have this model, we can either apply control methods like Model Predictive Control [3] and Linear Quadratic Regulator Control [4] or use model-free learning to learn a value function. The advantage is that once we learn a sufficient representation of our environment, we can decrease the amount of planning needed in the real environment.

However, model-based reinforcement learning faces a major challenge. If the dynamics model is not able to accurately represent the environment, error compounds and any learning that utilizes the dynamics model will become sub-optimal. It is generally accepted that this problem, termed model bias, leads model-based RL to perform asymptotically worse than model-free RL.

In this work, we build on two previous approaches, Neural Network Dynamics for Model-Based RL [5] and Predictability Minimization [6] to help alleviate this problem of model bias.

By taking inspiration from recent work in curiosity, we show how adversarial training can be used to improve exploration and increase asymptotic performance of model-based learning.

Algorithm 1 Adversarial Dynamics Learning

```
Require: θ_D, initial dynamics model parameters. θ_AD, initial adversarial model parameters. T, episode length.
 1: Initialize replay buffer $\mathcal{B}$ to capacity $\mathcal{N}$
 2: Initialize adversarial model $\hat{ad}$ with parameters $\theta_{AD}$ and dynamics model $\hat{d}$ with parameters $\theta_D$.
 3: while $\theta_D$ has not converged do
 4:     for t in T do
 5:         Get current state $s_t$
 6:         $a_t \leftarrow \hat{s}_{\theta_D}(s_t)$
 7:         $s_{t+1} \leftarrow$ Execute $a_t$
 8:         Store transition $(s_t, a_t, s_{t+1})$ in $\mathcal{B}$
 9:         Sample random minibatch of N transitions from $\mathcal{B}$
10:         $L \leftarrow \frac{1}{N}\sum((\hat{d}_{\theta_D}(s_t) - s_{t+1})^2)$
11:         Update $\theta_D$ by minimizing $L$
12:         Update $\theta_{AD}$ by maximizing $L$
13:     end for
14: end while
```

II. Motivation and Related Work

Recently, concepts like curiosity and surprise have been used to improve exploration in long-horizon planning tasks. Two recent works, Random Network Distillation (RND) [7] and Self Supervised Prediction [8], used curiosity as an intrinsic reward to improve performance across several RL benchmarks. However, by explicitly representing novelty as an intrinsic reward, these methods bind their user to a model-free paradigm. However, if we want to learn through curiosity in a model based setting, where we do not initially care about task-specific rewards, our options are more limited. We might take a more simple, tabular approach that keeps track of the states we visited [9], but this often fails to generalize in large, continuous state spaces. We might also try to empirically predict progress [10] or use an algorithm like R-MAX that internally deals with exploration [11], but these approaches are difficult to extend to continuous deep model based learning.

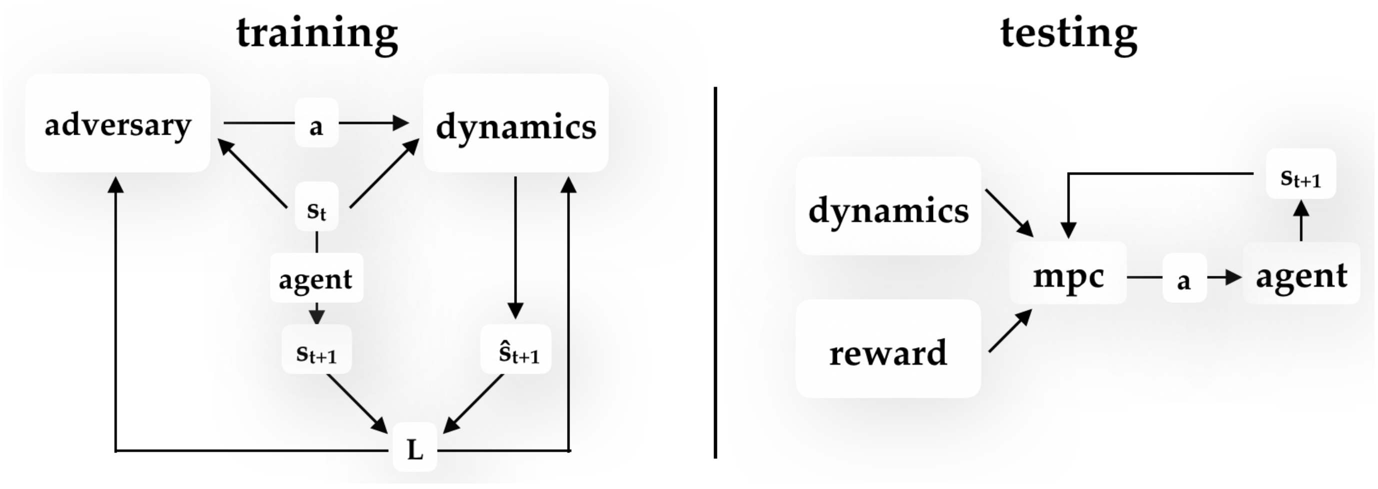

Fig. 1. A general overview of the method proposed. During training time, an adversary chooses which action to take at a given state, which is used to update dynamics.

III. Problem Definition

We consider a traditional reinforcement learning setting and formulate our task as a Markov Decision Process, or MDP, with known reward and unknown dynamics.

We seek to find an optimal policy π^* that maximizes the cumulative discounted reward $V^{\pi}(s)$. In order to learn this optimal policy with as few real samples as possible, we aim to find π^* using a minimal set of transition samples $\mathcal{B}$.

To find this optimal policy, we take a model-based approach. We introduce a dynamics function f parameterized by θ_f that, given current state s_t and current action a_t, predicts s_{t+1}. After collecting a dataset $\mathcal{B}$ of transition samples of the form $< s_t, a_t, s_{t+1} >$, we update θ_f with the objective of minimizing the mean-squared error loss $\frac{1}{N}\sum((\hat{d}_{\theta_D}(s_t) - s_{t+1})^2)$ across N new transition samples.

After we learn f, we apply a planning method and utilize f to produce a policy π. In this paper, we use Model Predictive Control (MPC) for its simplicity. At each time step, we sample a set of actions. For each of those actions, we roll out several random trajectories in the simulated environment represented by f over a horizon H, then return the action that leads to the highest expected reward. We repeat this process and replan at each time step.

In the next section, we show how we can use adversarial learning to improve this process.

IV. Approach

To incentivize exploration, we introduce an adversarial teacher t. The goal of t is to, given the current state s, produce an action a that maximizes dynamics loss L. This formulation can be viewed as a minimax game, where dynamics model f attempts to minimize L while adversarial teacher t attempts to maximize L.

Crucially, the same dynamics loss is used to update both the teacher and the dynamics model. In turn, while the objective of the dynamics model is to minimize its loss on the predicted next state, the objective of the adversarial model is to maximize this loss. This minimax formulation is adapted from [6]. The full algorithm for learning dynamics is shown in Algorithm 1.

The full approach is as follows. First, we use adversarial training to collect a dataset $\mathcal{B}$ from a real environment. The dynamics model and adversarial model are updated at each time step. Next, we train our dynamics model on this dataset. Next, given a reward function R and our trained dynamics f, we perform model predictive control over a given task as detailed above. Although several more complex procedures can be used, like alternating learning between our simulated and real environments, we stick to the simplest approach of only learning a dynamics model once and testing it

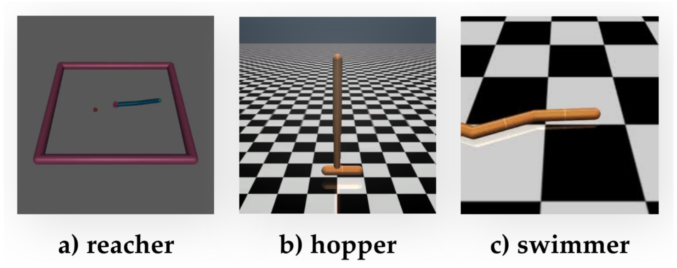

Fig. 2. MuJoCo environments used for visualization and testing.

once to best ascertain the efficacy of adversarial training. Both the dynamics model f and the adversarial teacher t are represented as fully-connected feedforward neural networks. This approach is summarized in Figure 1.

Because this approach uses the same loss for both the dynamics model and the adversarial model, implementing it simply involves adding an extra feedforward neural network.

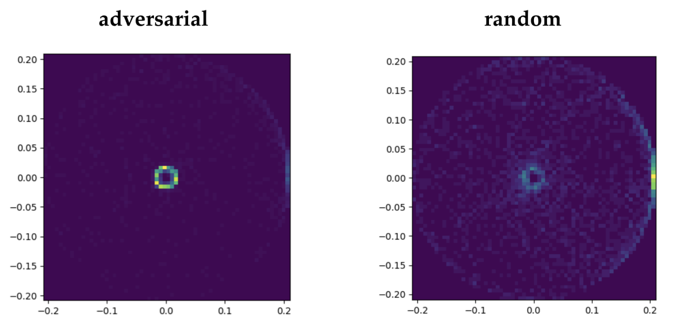

Fig. 3. An overhead heat map of the states visited by each method. Each state corresponds to the x-y position of the reacher's end effector.

V. Experiments

A. Visualizing Exploration

In order to first understand what types of policies an adversarial teacher can give us, we visualized them across several environments. Ideally, visualization should show that adversarial training leads us to visit novel states that a random policy would otherwise ignore. Initially, we tried to perform this visualization on two classic RL tasks, inverted pendulum and cartpole. However, we found that the set of reachable states was too small, meaning that most states could be reached in relatively few steps, to see any meaningful difference between the adversarial rollout and random rollout.

In turn, we switched to another RL environment, Reacher. Reacher is a 3D task built in Mujoco. In this task, the RL agent's objective is to actuate a 2-joint arm to reach points on a plane. An image of this environment is shown in Figure 2.

After training for 16 epochs on nearly 2,000 episodes, we visualized several rollouts of the adversarial and random policies. As shown in Figure 3, the adversarial agent ended up visiting states closest to its center, while the random agent visited states fairly uniformly. Intuitively, this might be because the points where the end effector touches the base of the reacher are often encountered when the agent applies large control, making it spin violently into itself.

This, in turn, might make state transition harder to predict, therefore incentivizing our adversary to reach those points. However, this kind of behavior is suboptimal, since most points we would like to reach probably will not appear near the center of the reacher. In turn, if our dynamics model is trained on these states, it does not actually learn more about states that lead to more success, which we care more about. Clearly, then, continuously training in an adversarial manner might harm learning progress if it incentivizes us to seek bad states. In turn, this realization led us to develop two versions of this method. We will detail these two approaches in the next section.

B. Performance

In order to ascertain whether this approach works across the full model-based pipeline, we seek to roll it out on several reinforcement learning tasks and measure the total reward that the method achieves.

Then, we compare it to a baseline that uses random action sampling to construct $\mathcal{B}$ and rolls out policies using MPC.

In our preliminary results, we test it on two simulated environments from OpenAI Gym, a popular RL benchmarking library. These two environments are built in MuJoCo, a physics simulator. In the first task, Hopper, the RL agent's task is to "hop" a simple leg as far as possible down a line. In the second task, Swimmer, the task is to swim a snake-like model as far as possible. These environments are visualized in Figure 2. Finally, inspired by what we saw in visualization, we test the performance of two variations of adversarial curiosity.

1) Continuously Using Adversarial Training: In the standard approach, we continually train the dynamics model adversarially and collect samples during training, and only use MPC rollout to test performance. While we do collect samples from MPC rollout, most of the dataset consists of samples from adversarial rollout.

We expect this approach to either work at the level of our baseline or below the level of our baseline, since our dynamics model will most likely overtrain to hard-to-predict states and not capture dynamics around state-actions that lead to better rewards.

2) Using Adversarial Training to Bootstrap: In our modified approach, we first train the dynamics model adversarially and collect transition samples. Then, we rollout the dynamics model using MPC in the real world. At each iteration of rollout, we append our MPC-given transition samples and use those to retrain the dynamics model.

Therefore, in this case, the adversarial model is only used initially to bootstrap the dynamics model before we start training it on data from MPC rollout.

This could be beneficial if we can successfully learn dynamics through adversarial training before rolling out with MPC. Afterwards, adding training data from MPC rollout allows us to focus on training dynamics in states that are beneficial for specific task performance.

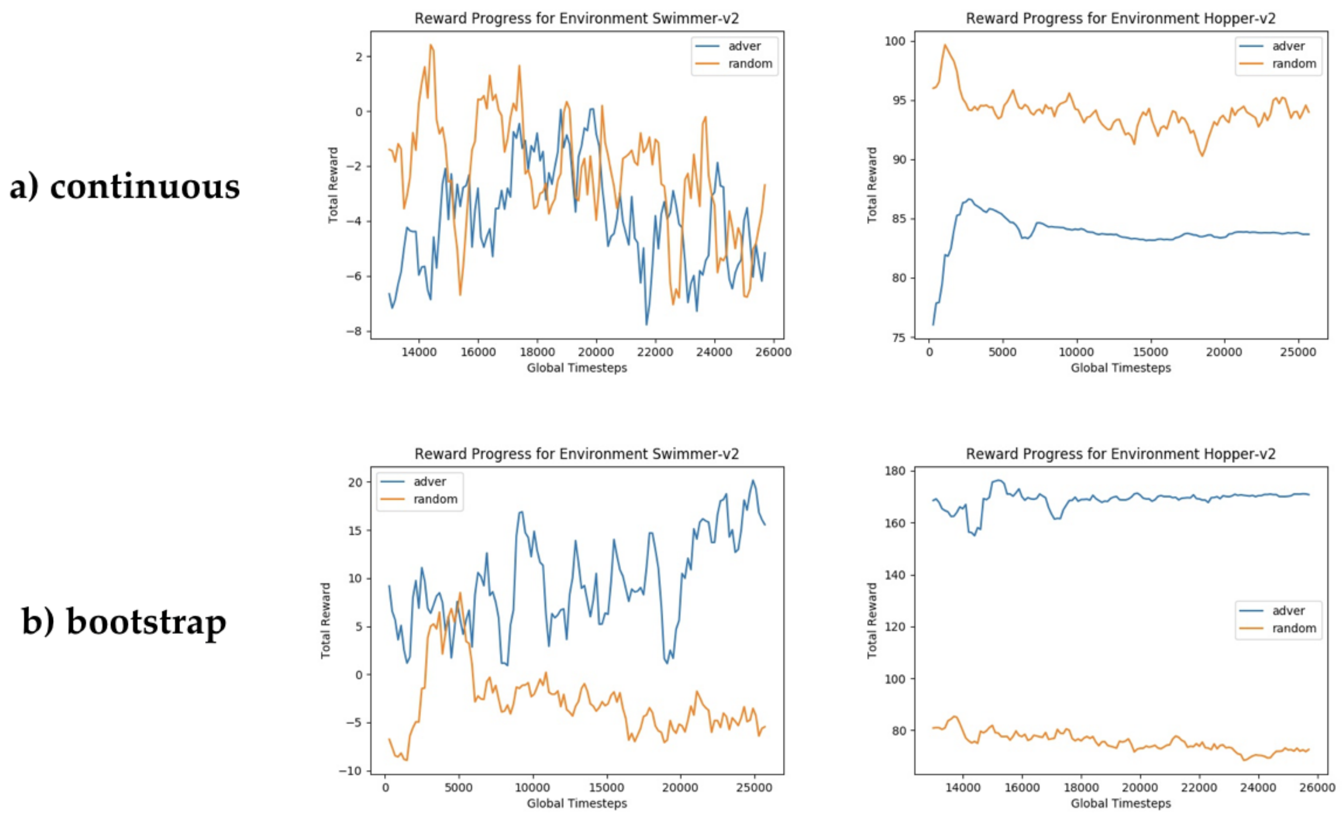

Fig. 4. Preliminary performance of rolled out policies trained using each method. The top row shows comparisons between random and adversarially trained policies by continuously training with the adversary, while the bottom row shows comparisons for the bootstrapped method.

We might expect this approach to perform either at the level of our baseline or slightly above our baseline, since adding adversarial samples might diversify our training dataset enough to learn a better dynamics function.

VI. Results

Due to time constraints, we were only able to visualize performance across the first 10k to 20k timesteps for one run of each algorithm on each environment. We included the hyperparameters used in Tables I,II, III, and IV. Total performance for our two methods is visualized in Figure 4.

A. Continually Using Adversarial Training

As seen in the first row, continually using adversarial training seems to either degrade or maintain baseline performance. In the Swimmer environment, our method and the baseline appeared to perform nearly identically. Lastly, in the Hopper environment, the random method performed substantially better than the adversarial method, although average total reward did not go up, indicating a flaw in our learning process.

Therefore, this continuous training approach performed overall worse than the baseline, as expected, most likely do to the dynamics model being majority trained on poor states reached through adversarial training.

B. Using Adversarial Training To Bootstrap

Using adversarial training to bootstrap learning seemed to perform better than the original approach. In the Swimmer environment, our method begins to outperform the baseline, but the baseline staying relatively constant is cause for concern. Then, in the Hopper environment, our method does outperform the baseline, but since average rewards remains constant for both, this result might be flawed.

To summarize, it seems like this bootstrap method can lead to better performance, as seen in our two preliminary examples. However, in all these cases, our testing size should leave much room for skepticism, meaning that although these results present a promising start, they by no means answer the efficacy of our method.

VII. Discussion

Although our preliminary results showed some hope for adversarial training, quantifying its efficacy requires training across many more environments and conditions.

A. Limitations

As discussed earlier, one limitation of adversarial curiosity is that it might lead the agent to visit "bad" states. This might make our dynamics model more robust, but this robustness might go unused if a successful policy only involves visiting a small set of specific states.

Furthermore, because our manner of updating the adversary simply involved using the same update as the one used for the dynamics model, we do not know if we can create the most effective adversary.

Another limitation is that training dynamics on a more diverse set of states makes it harder to fit a dynamics function, especially if our state observations are noisy, as would be the case in a real-world environment.

Furthermore, since model-based approaches are known to underperform their model-free counterparts, it is unclear whether the benefit provided by adversarial training, if any, is marginal at best.

Finally, in these preliminary tests, our reward functions had to have access to the full state space, which meant we had to learn the full state space. This means that if we want to represent larger state spaces, it again becomes more and more difficult to accurately learn a robust dynamics model. This could be prevented by first condensing the state space to a representation space, either through a tool like Principal Component Analysis or by learning a representation space using a method like Time Contrastive Networks [12].

B. Future Work

First and foremost, we hope to run this approach on significantly more simulated environments and across various training schemes and more explicitly compare performance to model-free methods. Our first planned extension is to replace MSE loss with an H-step loss which allows us to get dynamics error over several time steps. One possible training scheme is to randomly choose between an action selected by an adversary and an action selected with MPC with some probability ϵ, then learn the most effective ϵ across one or several environments. Then, we hope to extend this approach to image-based observations. Next, we hope to experiment with how the dynamics model and adversarial model can be updated differently, instead of relying on the same loss. Furthermore, we hope to apply this type of adversarial curiosity to model-free learning and directly compare it to Random Network Distillation [7] and Self Supervised Prediction [8].

References

[1] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[2] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[3] Basil Kouvaritakis and Mark Cannon. *Model predictive control*. Springer, 2016.

[4] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems*. Vol. 1. Wiley-Interscience New York, 1972.

[5] Anusha Nagabandi et al. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566.

[6] Jürgen Schmidhuber. "Learning factorial codes by predictability minimization". In: *Neural Computation* 4.6 (1992), pp. 863–879.

[7] Yuri Burda et al. *Exploration by Random Network Distillation*. 2018. arXiv: 1810.12894 `[cs.LG]`.

[8] Deepak Pathak et al. "Curiosity-Driven Exploration by Self-Supervised Prediction". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (July 2017). DOI: 10.1109/cvprw.2017.70. URL: http://dx.doi.org/10.1109/CVPRW.2017.70.

[9] Haoran Tang et al. "# Exploration: A study of count-based exploration for deep reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2753–2762.

[10] Manuel Lopes et al. "Exploration in Model-based Reinforcement Learning by Empirically Estimating Learning Progress". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 206–214. URL: http://papers.nips.cc/paper/4642-exploration-in-model-based-reinforcement-learning-by-empirically-estimating-learning-progress.pdf.

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

[12] Pierre Sermanet et al. "Time-Contrastive Networks: Self-Supervised Learning from Video". In: *arXiv preprint arXiv:1704.06888* (2017).

TABLE I
TRAINING HYPERPARAMETERS

Hyperparameter	**Value**
Replay buffer size	10e6
Episode length	100
Training steps	4096
Train batch size	512
Test rollouts per epoch	1
Observation scaling	10.0
Input normalization	True

TABLE II
MPC HYPERPARAMETERS

Hyperparameter	**Value**
MPC horizon	10
MPC simulated trajectories	5000

TABLE III
CONTINUOUS TRAINING HYPERPARAMETERS

Hyperparameter	**Value**
Number of updates	128
Training epochs per update	1
Testing epochs per update	1

TABLE IV
BOOTSTRAP TRAINING HYPERPARAMETERS

Hyperparameter	**Value**
Number of updates	1
Training epochs per update	128
Testing epochs per update	128