DYNA-AIL : ADVERSARIAL IMITATION LEARNING BY PLANNING

Vaibhav Saxena, Srinivasan Sivanandan & Pulkit Mathur [*]
Department of Computer Science
University of Toronto
Toronto, ON M5S, Canada
{vaibhav,srinivasan,pulkit}@cs.toronto.edu

ABSTRACT

Adversarial methods for imitation learning have been shown to perform well on various control tasks. However, they require a large number of environment interactions for convergence. In this paper, we propose an end-to-end differentiable adversarial imitation learning algorithm in a Dyna-like framework for switching between model-based planning and model-free learning from expert data. Our results on both discrete and continuous environments show that our approach of using model-based planning along with model-free learning converges to an optimal policy with fewer number of environment interactions in comparison to state-of-the-art imitation learning methods.

1 INTRODUCTION

Imitation learning refers to learning a policy from expert's state-action-state trajectories without access to reinforcement signal from the environment, or interactions with an expert. While model-free imitation learning methods can be used to learn complex policies in high-dimensional state spaces, they incur a huge cost in the number of interactions required with the environment. Use of a world model reduces the number of such interaction trials drastically, but requires the dynamics model to be very accurate, as a small bias in the model can lead to a strong bias in the policy.

Dyna (Sutton, 1991) based methods use a combined approach wherein the policy is learned by alternating between model-based and model-free methods. Bansal et al. (2017) incorporated the advantages of learning a world model, by learning a probabilistic dynamics model which later acts as a prior for model-free optimization.

Ho & Ermon (2016) proposed the Generative Adversarial Imitation Learning (GAIL) algorithm for performing imitation learning in a model-free setup using a GAN-like architecture, where the generator acts as the policy and the discriminator function judges whether an action given to it is from an expert or from the imitating policy. In this method, the agent learns the expert behaviour by an on-policy update on the state transitions sampled from the environment. Baram et al. (2017) proposed MGAIL, an end-to-end differentiable version of GAIL where they use a forward-model to propagate the gradient of the discriminator D w.r.t. the state s through to the policy for a gradient update. However, we note that MGAIL still samples state transitions from the environment while unrolling trajectories, and uses a re-parameterization of environment state transitions so as to calculate the gradients.

In many practical learning situations, interactions with the environment are quite expensive. In this paper, we propose Dyna Adversarial Imitation Learning (Dyna-AIL), a method for adversarial imitation learning which involves learning the expert policy by switching between world-model based planning and model-free reactive execution.

2 DYNA - ADVERSARIAL IMITATION LEARNING

In this work, we present an algorithm (Algorithm 1) for using the Dyna framework with adversarial imitation learning methods to obtain improvement over environment sampling efficiency. Inspired by the work of Baram et al. (2017), we use a forward-model through which the gradient of discriminator

[*]equal contribution.

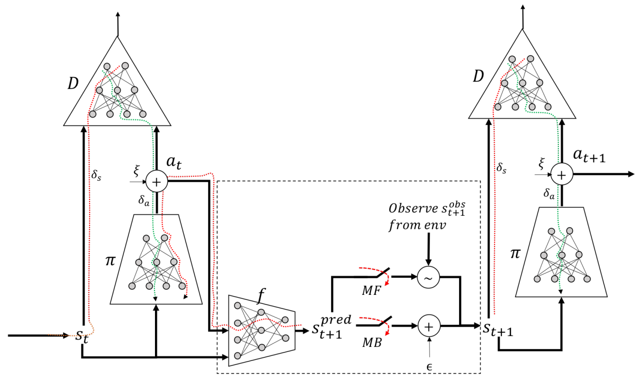


Figure 1: Computation graph of our proposed Dyna-AIL method showing the switch between model-based (MB) planning and model-free (MF) learning by execution.

D w.r.t. the state s can be used for the policy update. The details of MGAIL and its comparison to GAIL are included in Appendix A.

Model setup Our proposed architecture contains a discriminator and policy network setup similar to MGAIL. Additionally, at every timestep, we reparameterize the output of the forward-model depending on a MF/MB switch as,

$$s_{t+1} = \begin{cases} s_{t+1}^{pred} + \text{stop\_gradient}(s_{t+1}^{obs} - s_{t+1}^{pred}) & \text{if Model-Free} \\ s_{t+1}^{pred} + \epsilon & \text{if Model-Based} \end{cases} \tag{1}$$

where s_{t+1}^{pred} denotes the next state predicted by the forward-model, s_{t+1}^{obs} denotes the next state sampled from the environment, and ϵ represents a standard Gaussian noise. Our entire computation graph is illustrated in Fig. 1. To better represent the environment by a neural network function approximator, we designed the output of our forward-model as the change in the state values $f(s,a) = \delta s_{pred}$ such that $s'_{pred} = s + f(s,a)$.

Training objective Our model aims is to learn a discriminator function D and a policy function π which respectively maximize and minimize the expression

$$\mathbb{E}_{\pi}[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))]. \tag{2}$$

We train our forward-model by minimizing the squared loss between the state transition predictions from the forward-model and the observed state transitions as

$$L_f = \sum_{(s,a,s') \in B} \frac{1}{2} ||f(s,a) - (s' - s)||^2. \tag{3}$$

Algorithm We use a deep neural network to parameterize D and π, and use a set of expert trajectories to compute the expectation w.r.t. π_E. Now, in every iteration, we alternate between a learning phase and a planning phase. During the learning phase, we sample trajectories using π on the *environment* to make transitions, and take a gradient step on D using (4) in order to maximize (2). Then we use an on-policy gradient update on π using (5) so as to minimize (2). In every iteration, we train the forward-model by minimizing the squared loss in (3) using state transitions (s, a, s') sampled from trajectories stored in the experience replay buffer B.

During the planning phase, we sample trajectories using π on the *forward-model* for state transitions, and take a gradient step on π using (6). By having a planning step in every iteration, our approach optimizes the number of environment interactions required to imitate an expert policy.

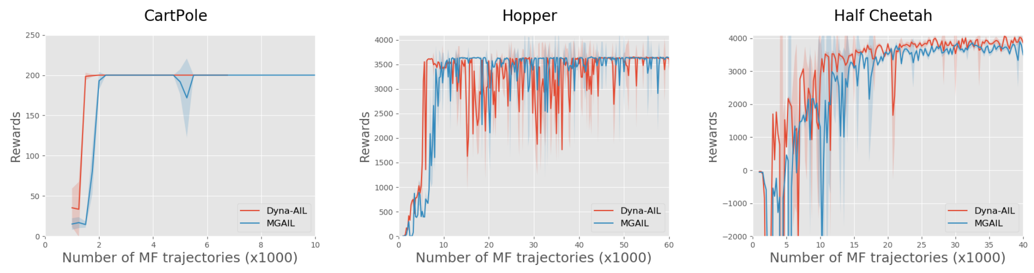
In our experiments, we use the multi-step computation graph as shown in Fig. 1 with gradient updates as given in (5) and (6). However, we also discuss our algorithm's performance using natural policy gradient update with TRPO rule, in Section 4, for comparison with (Ho & Ermon, 2016).

3 Experiments and Results

We evaluate our proposed algorithm on one discrete control task (CartPole (Barto et al., 1983)), and two continuous control tasks (Hopper, HalfCheetah) modeled by the MuJoCo (Todorov et al., 2012)

Algorithm 1 Dyna - Adversarial Imitation Learning

```
1:  **Input:** Expert trajectories $\tau_E$, experience buffer $B$, initial parameters for policy ($\pi$) and discriminator ($D$) $\theta_g, \theta_d$
2:  **repeat**
3:      Sample trajectories $\tau_i$ using policy $\pi(a|s;\theta_g)$ on environment
4:      Store trajectories $\tau_i$ into $B$                                  ▷ for experience replay
5:      Update discriminator ($D$) parameters $\theta_d$ with gradient
            $$\hat{\mathbb{E}}_{\tau_i}[\nabla_{\theta_d} \log(D_{\theta_d}(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_{\theta_d} \log(1 - D_{\theta_d}(s,a))] \quad (4)$$
6:      Update policy ($\pi$) parameters $\theta_g$ with gradient
            $$\nabla_{\theta_g} \mathbb{E}_{\tau_i}\left[\sum_{t=0} \gamma^t \log(D(s_t, a_t))\right] \quad (5)$$
7:      Train forward-model $f$ using $(s, a, s')$ from $B$
8:      Sample trajectories $\tau_j$ using policy $\pi(a|s;\theta_g)$ on forward model $f$
9:      Update policy ($\pi$) parameters $\theta_g$ with gradient                ▷ model-based planning
            $$\nabla_{\theta_g} \mathbb{E}_{\tau_j}\left[\sum_{t=0} \gamma^t \log(D(s_t, a_t))\right] \quad (6)$$
10: **until** convergence
```



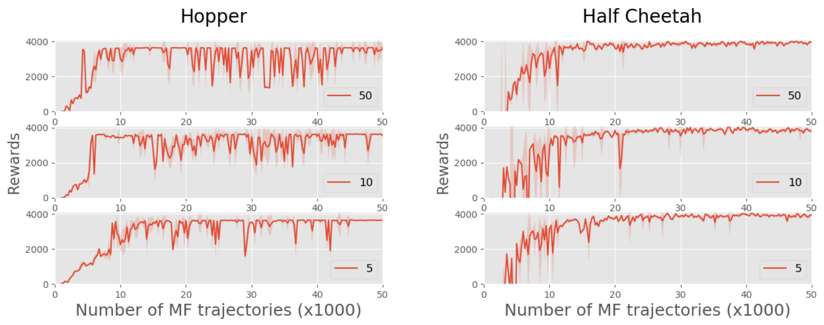
(a) Cartpole environment. (b) Hopper environment. (c) HalfCheetah environment.

Figure 2: Results of our experiments showing the rewards against number of model-free trajectories used while training Dyna-AIL vs MGAIL on discrete (CartPole) and continuous (Hopper, HalfCheetah) control tasks. (Trajectory length while planning $T_p = 10$.) We observe that Dyna-AIL converged to an optimal policy using lesser number of model-free trajectories as compared to MGAIL.

physics simulator, using the cost function defined in the OpenAI Gym (Brockman et al., 2016). We use the Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) algorithm to train our expert policies. For each of the tasks, we generated expert datasets with a number of trajectories ($n_T = 50$), where each trajectory: $\tau = s_0, a_0, s_1, ...s_N, a_N$ is of length N = 1000.

The discriminator and policy networks are each designed with two hidden layers (200 and 100 units for discriminator, and 100 and 50 units for policy) with ReLU non-linearity and trained using the Adam (Kingma & Ba, 2014) optimizer (similar to the architectures used in (Baram et al., 2017)). As noted in (Baram et al., 2017), the forward-model structure is crucial for the stability of the network's learning process. Since the state and action inputs are from different distributions, we embed them into a shared space using state and action encoder networks. Then, we combine the embeddings through a Hadamard product and use that as input for the state transition model. We also model the environment as a n$^{\text{th}}$ order MDP, with a recurrent connection from the previous states, using a GRU layer as a part of the state encoder (as previously used in (Baram et al., 2017)).

In every iteration of Algorithm 1, we perform the updates in (4) and (5) using a m-step ($m = 50$) stochastic gradient descent with trajectories sampled from the environment. Later in the iteration, we perform planning in (6) using a p-step ($p = 50$) stochastic gradient descent with trajectories



(a) Hopper environment. (b) HalfCheetah environment.

Figure 3: Results of our experiments showing the rewards against number of model-free trajectories used while training Dyna-AIL, for different lengths of trajectories ($T_p = 5, 10, 50$) sampled from the forward-model.

sampled from the forward-model. To ensure the stability of the planner, we restrict the unrolling of the trajectory in planning phase to a fixed number of steps (T_p). In Section 4, we discuss the performance of our algorithm by varying the trajectory lengths T_p.

After every iteration, the policy learned in the algorithm was evaluated using 10 episodes by acting on the environment. Fig. 2 shows the learning curve comparison between our algorithm and MGAIL. From the experimental results, we observe that our algorithm learns the optimal policy with fewer number of interactions with the environment, as compared to MGAIL. However, in the case of Hopper environment, our algorithm has a high variance in performance, which could be attributed to the bias introduced by the forward-model used for sampling trajectories in the planning phase. In Section 4, we discuss a few possible solutions for overcoming the instability issues.

4 Discussion

To evaluate the stability of our Dyna planner w.r.t the planning phase trajectory length T_p, we performed experiments with different values of T_p on Hopper and HalfCheetah environments. Fig. 3 shows the learning curves for $T_p = 5, 10, 50$. We observe that with the Hopper environment, the learning curve has a higher variance with $T_p = 50$ as compared to with lower values of T_p. However, with the HalfCheetah environment, learning curves for all T_p values converge with very low variance.

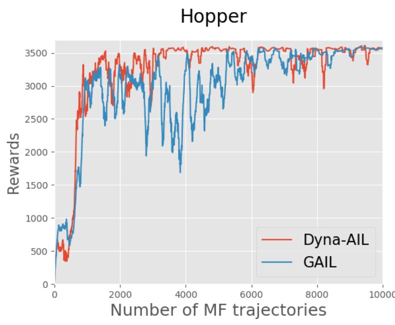


Figure 4: Results of our experiments showing the rewards against number of model-free trajectories used while training Dyna-AIL vs GAIL (using TRPO gradient step) on Hopper task.

Even though MGAIL addresses the problem of high variance gradients by utilizing the forward-model gradients, they still use a small step size due to the gradient update as per Heess et al. (2015). GAIL copes with the high variance in gradients by performing the policy update using a natural policy gradient step with TRPO update rule. The KL divergence constraint enables us to use a large step size for policy update without worrying about the noisy policy gradients. Hence, we evaluated Dyna-AIL on Hopper environment without forward-model gradients using the TRPO update rule, as per Ho & Ermon (2016). Fig. 4 shows the performance of the Dyna-AIL algorithm with TRPO update compared with the results of GAIL. The TRPO update ensures that the policy updates do not diverge by adjusting the step size in every gradient update and hence reduces the variance in the Hopper environment.

In this paper, we proposed a framework to train policies via imitation learning by switching between model-based planning and model-free execution. The results show that this framework reduces the number of environment interactions required to learn a good policy while imitating an expert. However, we still perform only one planning step for every learning step as there is no direct metric to evaluate the quality of the planning step to stop planning and learn from the environment. Wu et al. (2018) propose an adaptive Dyna-Q framework by integrating a switcher that automatically determines whether to use a real or simulated experience for Q-learning based on a quality metric. A possible future direction is to have an adaptive switching mechanism in our algorithm that could query the environment only when the model is uncertain about a state transition.

Acknowledgements We thank Jimmy Ba and Tingwu Wang for helpful discussions. We also thank the Department of Computer Science at the University of Toronto for providing the resources to complete this project.

REFERENCES

Somil Bansal et al. MBMF: model-based priors for model-free reinforcement learning. *CoRR*, abs/1709.03153, 2017. URL `http://arxiv.org/abs/1709.03153`.

Nir Baram et al. End-to-end differentiable adversarial imitation learning. In *Proceedings of the 34th ICML*, volume 70, pp. 390–399. PMLR, 06–11 Aug 2017. URL `http://proceedings.mlr.press/v70/baram17a.html`.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):834–846, 1983.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Nicolas Heess, Greg Wayne, David Silver, Timothy P. Lillicrap, Yuval Tassa, and Tom Erez. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015. URL `http://arxiv.org/abs/1510.09142`.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL `http://arxiv.org/abs/1606.03476`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL `http://arxiv.org/abs/1502.05477`.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4): 160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL `http://doi.acm.org/10.1145/122344.122377`.

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

Yuexin Wu, Xiujun Li, Jingjing Liu, Jianfeng Gao, and Yiming Yang. Switch-based active deep dyna-q: Efficient adaptive planning for task-completion dialogue policy learning. *CoRR*, abs/1811.07550, 2018. URL `http://arxiv.org/abs/1811.07550`.

A Background

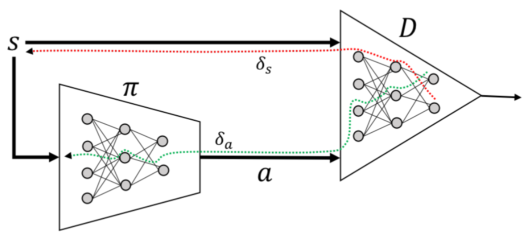


Figure 5: Computation graph of GAIL showing how gradient w.r.t. state (δ_s) is disregarded during back propagation.

A.1 Generative Adversarial Imitation Learning

Ho & Ermon (2016) proposed the Generative Adversarial Imitation Learning (GAIL) architecture for learning policies from expert data using Generative Adversarial Networks (Goodfellow et al., 2014). GAIL is a model-free approach wherein, the adversarial two-player zero-sum game can be represented as follows:

$$\arg\min_{\pi} \arg\max_{D\in(0,1)} \mathbb{E}_{\pi}[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] \tag{7}$$

A discriminator function D tries to maximize the objective in (7), while a policy function π tries to minimize it. In a neural network implementation, GAIL alternates between an Adam gradient (Kingma & Ba, 2014) step on the parameters of D, and a Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) gradient step on the parameters of π which minimizes the cost $c(s,a) = \log D(s,a)$ for the policy. Fig. 5 shows how the gradients are passed from the discriminator down to the policy network. Since the state transitions are taken from the environment, the gradient w.r.t. state s (δ_s) does not pass through to the policy, or in other words, it is ignored and not used for the policy update.

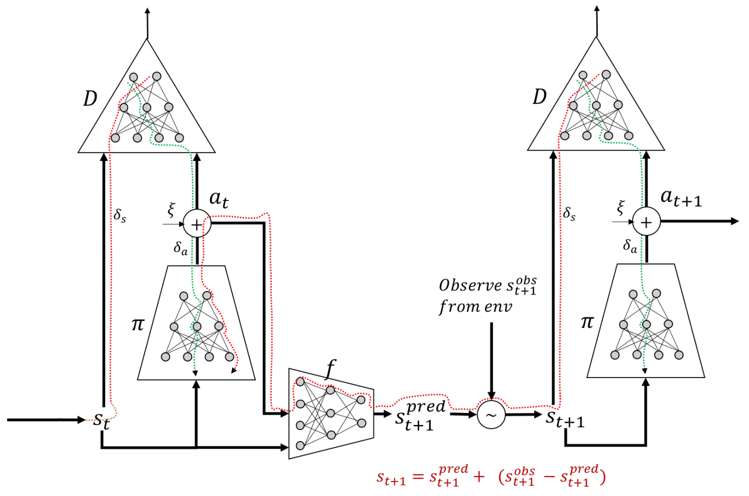


Figure 6: Computation graph of the MGAIL algorithm showing the end-to-end gradient computation over multiple time steps with the re-parameterization trick

A.2 End-to-end differentiable Adversarial Imitation Learning (MGAIL)

Baram et al. (2017) proposed an end-to-end differentiable version of GAIL where they use a forward-model through which the gradient of discriminator D w.r.t. the state s can be used for the policy update. They argue that since REINFORCE policy gradients suffer from high variance, it is difficult to work with them even after using variance reduction techniques. They also state that REINFORCE gradients are not same as the exact gradient of D, due to the independence assumption of state transitions w.r.t. policy parameters taken in REINFORCE.

So, if we assume that the state transition probabilities are dependent on the policy parameters, i.e. use exact gradients through the incorporation of a forward-model, the policy will receive better signals while learning. They build an end-to-end differentiable computation graph that spans over multiple time-steps as given by Heess et al. (2015) in their formulation, which for a transition (s, a, s') is given as:

$$J_s = \mathbb{E}_{p(a|s)}\mathbb{E}_{p(s'|s,a)}\left[D_s + D_a\pi_s + \gamma J'_{s'}(f_s + f_a\pi_s)\right], \tag{8}$$

$$J_\theta = \mathbb{E}_{p(a|s)}\mathbb{E}_{p(s'|s,a)}\left[D_a\pi_\theta + \gamma(J'_{s'}f_a\pi_\theta + J'_\theta)\right]. \tag{9}$$

The gradient with respect to an entire trajectory is obtained by applying (8) and (9) recursively over the trajectory, and is illustrated in Fig. 6. We note that MGAIL still uses state transitions from the environment while unrolling trajectories, probably because of an inadequacy in the forward-model capacity which would otherwise lead to noisy state transitions, and use a re-parameterization of environment state transitions so as to calculate gradients.

A.3 COMPARING THE ADVERSARIAL IMITATION LEARNING ALGORITHMS

Baram et al. (2017) added a forward-model in their algorithm so as to pass the gradient of discriminator D w.r.t. state s down to the policy network. However, they still sampled trajectories from the environment for computing the loss for the policy as well as the discriminator. In order to compute the gradients of the discriminator w.r.t. states, they re-parameterize the observed state s_{t+1}^{obs} as $s_{t+1}^{pred} + \nu$, where $\nu = s_{t+1}^{obs} - s_{t+1}^{pred}$ as shown in Fig. 6. Hence, MGAIL, like GAIL, is a model-free algorithm w.r.t. the state transitions.