End-to-End Differentiable Adversarial Imitation Learning

Nir Baram [1] Oron Anschel [1] Itai Caspi [1] Shie Mannor [1]

Abstract

Generative Adversarial Networks (GANs) have been successfully applied to the problem of *policy imitation* in a model-free setup. However, the computation graph of GANs, that include a stochastic policy as the generative model, is no longer differentiable end-to-end, which requires the use of high-variance gradient estimation. In this paper, we introduce the Model-based Generative Adversarial Imitation Learning (MGAIL) algorithm. We show how to use a forward model to make the computation fully differentiable, which enables training policies using the exact gradient of the discriminator. The resulting algorithm trains competent policies using relatively fewer expert samples and interactions with the environment. We test it on both discrete and continuous action domains and report results that surpass the state-of-the-art.

1. Introduction

Learning a policy from scratch is often difficult. However, in many problems, there exists an expert policy that achieves satisfactory performance. We are interested in the scenario of imitating an expert. Imitation is needed for several reasons: *Automation* (in case the expert is human), *distillation* (e.g., if the expert is too expensive to run in real-time (Rusu et al., 2015)), and *initialization* (using an expert policy as an initial solution). In our setting, we assume that trajectories $\{s_0, a_0, s_1, ...\}_{i=0}^{N}$ of an expert policy π_E are given. Our goal is to train a new policy π which imitates π_E without access to the original reward signal r_E that was used by the expert.

There are two main approaches to solve imitation problems. The first, known as Behavioral Cloning (BC), directly learns the conditional distribution of actions over states $p(a|s)$ in a supervised learning fashion (Pomerleau, 1991). By providing constant supervision (*dense* reward signal in Reinforcement Learning (RL) terminology), BC overcomes fundamental difficulties of RL such as the credit assignment problem (Sutton, 1984). However, BC has its downsides as well. Contrary to temporal difference methods (Sutton, 1988) that integrate information over time, BC methods are trained using single time-step state-action pairs $\{s_t, a_t\}$. Therefore, an agent trained using BC is unaware of how his choice of actions affects the future state distribution, which makes it susceptible to compounding errors (Ross & Bagnell, 2010; Ross et al., 2011). On top of that, the sample complexity of BC methods is usually high, requiring a significant amount of expert data that could be expensive to obtain.

The second approach to imitation is comprised of two stages. First, recovering a reward signal under which the expert is uniquely optimal (often called inverse RL, for instance see Ng, Russell, et al.):

$$\mathbb{E}\big[\sum_t \gamma^t \hat{r}(s_t, a_t)|\pi_E\big] \geq \mathbb{E}\big[\sum_t \gamma^t \hat{r}(s_t, a_t)|\pi\big] \quad \forall\pi. \tag{1}$$

After reconstructing a reward signal $\hat{r}$, the second step is to train a policy that maximizes the discounted cumulative expected reward: $\mathbb{E}_\pi R = \mathbb{E}_\pi\big[\sum_{t=0}^{T} \gamma^t \hat{r}_t\big]$. The problem with this approach stems from the fact that restoring an informative reward signal, solely based on state-action observations, is an ill-posed problem (Ziebart et al., 2008). A different strategy could be to recover a sparser reward signal (a more well-defined problem) and enrich it by hand. However, this requires extensive domain knowledge (Dorigo & Colombetti, 1998).

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) is a recent method for training generative models. It uses a second neural network (D) to guide the generative model (G) towards producing patterns similar to those of the expert (see illustration in Figure 1).

The elegance of GANs has made it popular among a variety of problems other than creating generative models, such as image captioning (Mirza & Osindero, 2014) and video prediction (Mathieu et al., 2015). More recently, a work named Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016), has successfully applied the ideas of GANs to imitate an expert in a model-free setup. It showed

[1]Technion Institute of Technology, Israel. Correspondence to: Nir Baram <nirb@campus.technion.ac.il>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

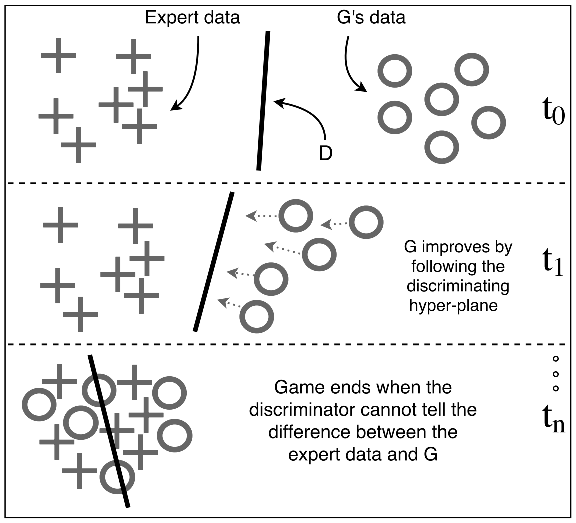

Expert data
G's data
D
t_0
t_1
G improves by following the discriminating hyper-plane
Game ends when the discriminator cannot tell the difference between the expert data and G
t_n

Figure 1. **Illustration of GANs.** The generative model follows the discriminating hyper-plane defined by the discriminator. Eventually, G will produce patterns similar to the expert patterns

that this type of learning could alleviate problems such as sample complexity or covariate shifts (Kanamori & Shimodaira, 2003), traditionally coupled with imitation learning.

The disadvantage of the model-free approach comes to light when training stochastic policies. The presence of stochastic elements breaks the flow of information (gradients) from one neural network to the other, thus prohibiting the use of backpropagation. In this situation, a standard solution is to use gradient estimation (Williams, 1992). This tends to suffer from high variance, resulting in a need for larger sample sizes as well as variance reduction methods.

In this work, we present a model-based imitation learning algorithm (MGAIL), in which information propagates fluently from the guiding neural network (D) to the generative model G, which in our case represents the policy π we wish to train. This is achieved by *(a)* learning a forward model that approximates the environment's dynamics, and *(b)* building an end-to-end differentiable computation graph that spans over multiple time-steps. The gradient in such a graph carries information from future states to earlier time-steps, helping the policy to account for compounding errors. This leads to better policies that require fewer expert samples and interactions with the environment.

2. Background

In this section, we review the mathematical formulation of Markov Decision Processes, as well as previous approaches to imitation learning. Lastly, we present GANs in detail.

2.1. Markov Decision Process

Consider an infinite-horizon discounted Markov decision process (MDP), defined by the tuple $(S, A, P, r, \rho_0, \gamma)$, where S is a set of states, A is a set of actions, $P : S \times A \times S \rightarrow [0,1]$ is the transition probability distribution, $r : (S \times A) \rightarrow \mathbb{R}$ is the reward function, $\rho_0 : S \rightarrow [0,1]$ is the distribution over initial states, and $\gamma \in (0,1)$ is the discount factor. Let π denote a stochastic policy $\pi : S \times A \rightarrow [0,1]$, $R(\pi)$ denote its expected discounted reward: $\mathbb{E}_\pi R = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t \hat{r}_t \right]$, and τ denote a trajectory of states and actions $\tau = \{s_0, a_0, s_1, a_1, ...\}$.

2.2. Imitation Learning

Learning control policies directly from expert demonstrations, has been proven very useful in practice, and has led to satisfying performance in a wide range of applications (Ross et al., 2011). A common approach to imitation learning is to train a policy π to minimize some loss function $l(s, \pi(s))$, under the discounted state distribution encountered by the expert: $d_\pi(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t p(s_t)$. This is possible using any standard supervised learning algorithm: $\pi = \operatorname{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi}[l(s, \pi(s))]$, where Π denotes the class of all possible policies. However, the policy's prediction affects the future state distribution, which violates the i.i.d assumption made by most SL algorithms. A slight deviation in the learner's behavior may lead it to a different state distribution than the one encountered by the expert, resulting in compounding errors.

To overcome this issue, Ross & Bagnell (2010) introduced the Forward Training (FT) algorithm that trains a non-stationary policy iteratively over time (one policy π_t for each time-step). At time t, π_t is trained to mimic π_E on the state distribution induced by the previously trained policies $\pi_0, \pi_1, ...\pi_{t-1}$. This way, π_t is trained on the actual state distribution it will encounter at inference. However, the FT algorithm is impractical when the time horizon T is large (or undefined), since it needs to train a policy at each time-step, and cannot be stopped before completion. The Stochastic Mixing Iterative Learning (SMILe) algorithm, proposed by the same authors, solves this problem by training a stochastic stationary policy over several iterations. SMILe starts with an initial policy π_0 that blindly follows the expert's action choice. At iteration t, a policy $\hat{\pi_t}$ is trained to mimic the expert under the trajectory distribution induced by π_{t-1}, and then updates:

$$\pi_t = \pi_{t-1} + \alpha(1-\alpha)^{t-1}(\hat{\pi_t} - \pi_0).$$

Overall, both the FT algorithm and SMILe gradually modify the policy from following the expert's policy to the learned one.

2.3. Generative Adversarial Networks

GANs learn a generative model using a two-player zero-sum game:

$$\underset{G}{\operatorname{argmin}} \underset{D\in(0,1)}{\operatorname{argmax}} \quad \mathbb{E}_{x\backsim p_E}[\log D(x)]+ \\ \mathbb{E}_{z\backsim p_z}\big[\log\big(1-D(G(z))\big)\big], \tag{2}$$

where p_z is some noise distribution. In this game, player G produces patterns (denoted as x), and the second one (D) judges their authenticity. It does so by solving a binary classification problem where G's patterns are labeled as 0, and expert patterns are labeled as 1. At the point when D (the judge) can no longer discriminate between the two distributions, the game ends since G has learned to mimic the expert.

The two players are modeled by neural networks (with parameters θ_d, θ_g respectively), therefore, their combination creates an end-to-end differentiable computation graph. For this reason, G can train by generating patterns, feeding it to D, and minimize the probability that D assigns to them:

$$l(z,\theta_g) = \log\big(1 - D(G_{\theta_g}(z))\big),$$

The benefit of GANs is that it relieves us from the need to define a loss function or to handle complex models such as RBM's and DBN's (Lee et al., 2009). Instead, GANs rely on basic ideas (binary classification), and basic algorithms (backpropagation). The judge D trains to solve a binary classification problem by ascending at the following gradient:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\Big[\log D_{\theta_d}\big(x^{(i)}\big) + \log\Big(1 - D_{\theta_d}\big(G(z^{(i)})\big)\Big)\Big],$$

interchangeably while G descends at the following direction:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\Big(1 - D\big(G_{\theta_g}(z^{(i)})\big)\Big).$$

Ho & Ermon (2016) (GAIL) proposed to apply GANs to an expert policy imitation task in a model-free approach. GAIL draws a similar objective function like GANs, except that here p_E stands for the expert's joint distribution over state-action tuples:

$$\underset{\pi}{\operatorname{argmin}} \underset{D\in(0,1)}{\operatorname{argmax}} \mathbb{E}_{\pi}[\log D(s,a)]+ \\ \mathbb{E}_{\pi_E}[\log(1-D(s,a))] - \lambda H(\pi), \tag{3}$$

where $H(\lambda) \triangleq \mathbb{E}_{\pi}[-\log\pi(a|s)]$ is the entropy.

The new game defined by Eq. 3 can no longer be solved using the standard tools mentioned above because player G (i.e., the policy π) is now *stochastic*. Following this modification, the exact form of the first term in Eq. 3 is given by $\mathbb{E}_{s\sim\rho_\pi(s)}\mathbb{E}_{a\sim\pi(\cdot|s)}[\log D(s,a)]$, instead of the following expression if π was deterministic: $\mathbb{E}_{s\sim\rho}[\log D(s,\pi(s))]$. The resulting game depends on the stochastic properties of the policy. So, assuming that $\pi = \pi_\theta$, it is no longer clear how to differentiate Eq. 3 w.r.t. θ. A standard solution is to use score function methods (Fu, 2006), of which REINFORCE is a special case (Williams, 1992), to obtain an unbiased gradient estimation:

$$\nabla_\theta\mathbb{E}_\pi[\log D(s,a)] \cong \hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log\pi_\theta(a|s)Q(s,a)], \tag{4}$$

where $Q(\hat{s},\hat{a})$ is the score function of the gradient:

$$Q(\hat{s},\hat{a}) = \hat{\mathbb{E}}_{\tau_i}[\log D(s,a)\,|\,s_0=\hat{s}, a_0=\hat{a}]. \tag{5}$$

Although unbiased, REINFORCE gradients tend to suffer high variance, which makes it hard to work with even after applying variance reduction techniques (Ranganath et al., 2014; Mnih & Gregor, 2014). In the case of GANs, the difference between using the exact gradient and REINFORCE can be explained in the following way: with REINFORCE, G asks D whether the pattern it generates are authentic or not. D in return provides a brief Yes/No answer. On the other hand, using the exact gradient, G gets access to the internal decision making logic of D. Thus it is better able to understand the changes needed to fool D. Such information is present in the Jacobian of D.

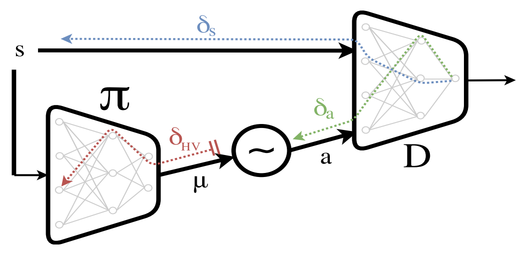
In this work, we show how a forward model utilizes the Jacobian of D when training π, *without* resorting to high-variance gradient estimations. The challenge of this approach is that it requires learning a differentiable approximation to the environment's dynamics. Errors in the forward model introduce a bias to the policy gradient which impairs the ability of π to learn robust and competent policies. We share our insights regarding how to train forward models, and in subsection 3.5 present an architecture that was found empirically adequate in modeling complex dynamics.

3. Algorithm

We start this section by analyzing the characteristics of the discriminator. Then, we explain how a forward model can alleviate problems that arise when using GANs for policy imitation. Afterward, we present our model-based adversarial imitation algorithm. We conclude this section by presenting a forward model architecture that was found empirically adequate.

3.1. The discriminator network

The discriminator network is trained to predict the conditional distribution: $D(s,a) = p(y|s,a)$ where $y \in \{\pi_E, \pi\}$. Put in words, $D(s,a)$ represents the likelihood ratio that the pair $\{s,a\}$ is generated by π rather than by



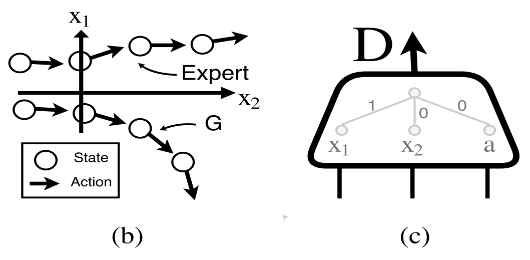


Figure 2. **(a)** Block-diagram of the model-free approach: given a state s, the policy outputs μ which is fed to a stochastic sampling unit. An action a is sampled, and together with s are presented to the discriminator network. In the backward phase, the error message δ_a is *blocked* at the stochastic sampling unit. From there, a high-variance gradient estimation is used (δ_{HV}). Meanwhile, the error message δ_s is flushed. **(b)** Discarding δ_s can be disastrous as shown in the following example. Assume some $\{s, a\}$ pairs produced by the expert and G. Let $s = (x_1, x_2)$, and $a \in \mathbb{R}$. **(c)** Assuming the expert data lies in the upper half-space ($x_1 > 0$) and the policy emits trajectories in the lower half-space ($x_1 < 0$). Perfect discrimination can be achieved by applying the following rule: $sign(1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot a)$. Differentiating w.r.t the three inputs give: $\frac{\partial D}{\partial x_1} = 1, \frac{\partial D}{\partial x_2} = 0, \frac{\partial D}{\partial a} = 0$. Discarding the partial derivatives w.r.t. x_1, x_2 (the state), will result in zero information gradients.

π_E. Using Bayes rule and the law of total probability we can write that:

$$D(s,a) = p(\pi|s,a) = \frac{p(s,a|\pi)p(\pi)}{p(s,a)} = \frac{p(s,a|\pi)p(\pi)}{p(s,a|\pi)p(\pi) + p(s,a|\pi_E)p(\pi_E)} = \frac{p(s,a|\pi)}{p(s,a|\pi) + p(s,a|\pi_E)}. \tag{6}$$

The last equality is correct since the discriminator is trained on an even distribution of expert/generator examples, therefore: $p(\pi) = p(\pi_E) = \frac{1}{2}$. Re-arranging and factoring the joint distribution we can rewrite $D(s, a)$ as:

$$D(s,a) = \frac{1}{\frac{p(s,a|\pi)+p(s,a|\pi_E)}{p(s,a|\pi)}} = \frac{1}{1 + \frac{p(s,a|\pi_E)}{p(s,a|\pi)}} = \frac{1}{1 + \frac{p(a|s,\pi_E)}{p(a|s,\pi)} \cdot \frac{p(s|\pi_E)}{p(s|\pi)}} . \tag{7}$$

Next let us define $\varphi(s, a)$, and $\psi(s)$ to be:

$$\varphi(s,a) = \frac{p(a|s,\pi_E)}{p(a|s,\pi)}, \psi(s) = \frac{p(s|\pi_E)}{p(s|\pi)},$$

and attain the final expression for $D(s, a)$:

$$D(s,a) = \frac{1}{1 + \varphi(s,a) \cdot \psi(s)} . \tag{8}$$

Inspecting the derived expression we see that $\varphi(s, a)$ represents a *policy likelihood ratio*, and $\psi(s)$ represents a *state distribution likelihood ratio*. This interpretation suggests that the discriminator makes its decisions by answering two questions. The first relates to the *state distribution*: what is the likelihood of encountering state *s* under the distribution induced by π_E vs. the one induced by π? And the second question relates to the *behavior*: given a state *s*, how likely is action *a* under π_E vs. π?

We reach the conclusion that effective learning requires the learner to be mindful of two effects. First, how its choice of actions stands against the expert? And second, how it affects the future state distribution? The desired change in states is given by $\psi_s \equiv \partial\psi/\partial s$. A careful inspection of the partial derivatives of D reveals that this information is present in the Jacobian of the discriminator:

$$\nabla_a D = -\frac{\varphi_a(s,a)\psi(s)}{(1+\varphi(s,a)\psi(s))^2}, \qquad \nabla_s D = -\frac{\varphi_s(s,a)\psi(s) + \varphi(s,a)\psi_s(s)}{(1+\varphi(s,a)\psi(s))^2}, \tag{9}$$

which increases the motivation to use it over other high-variance estimations. Next, we show how using a forward model, we can build a policy gradient directly from the Jacobian of the discriminator (i.e., $\nabla_a D$ and $\nabla_s D$).

3.2. Backpropagating through stochastic units

We are interested in training stochastic policies. Stochasticity is important for Policy Gradient (PG) methods since it encourages exploration. However, it poses a challenge for policies modeled by neural networks, considering it is unclear how to backpropagate through the stochastic elements. This problem plays a major role in algorithms that build differentiable computation graphs where gradients flow from one component to another, as in the case of deep actor-critic methods (Lillicrap et al., 2015), and *GANs*. In the following, we show how to estimate the gradients of continuous stochastic elements (for continuous action domains), and categorical stochastic elements (for the discrete case).

3.2.1. Continuous Action distributions

In the case of continuous action policies we use a mathematical tool known as ”re-parametrization” (Kingma &

Welling, 2013; Rezende et al., 2014), which enables computing the derivatives of stochastic models. Assume a stochastic policy with a Gaussian distribution[1], where the mean and variance are given by some deterministic functions μ_θ and σ_θ, respectively: $\pi_\theta(a|s) \sim \mathcal{N}(\mu_\theta(s), \sigma^2_\theta(s))$. It is possible to re-write π as $\pi_\theta(a|s) = \mu_\theta(s) + \xi\sigma_\theta(s)$, where $\xi \sim \mathcal{N}(0,1)$. In this way, we are able to get a Monte-Carlo estimator of the derivative of the expected value of $D(s,a)$ with respect to θ:

$$\nabla_\theta \mathbb{E}_{\pi(a|s)} D(s,a) = \mathbb{E}_{\rho(\xi)} \nabla_a D(a,s) \nabla_\theta \pi_\theta(a|s) \cong \frac{1}{M}\sum_{i=1}^{M} \nabla_a D(s,a) \nabla_\theta \pi_\theta(a|s)\Big|_{\xi=\xi_i}. \tag{10}$$

3.2.2. Categorical Action distributions

For the case of discrete action domains, we suggest to follow the idea of categorical re-parametrization with Gumbel-Softmax (Maddison et al., 2016; Jang et al., 2016). This approach relies on the Gumbel-Max trick (Gumbel & Lieblein, 1954); a method to draw samples from a categorical distribution with class probabilities $\pi(a_1|s), \pi(a_2|s), ...\pi(a_N|s)$:

$$a_{\text{argmax}} = \underset{i}{\text{argmax}}\,[g_i + \log \pi(a_i|s)],$$

where $g_i \sim \mathit{Gumbel}(0,1)$. Gumbel-Softmax provides a differentiable approximation of the *hard* sampling procedure in the Gumbel-Max trick, by replacing the *argmax* operation with a *softmax*:

$$a_{\text{softmax}} = \frac{\exp\left[\frac{1}{\tau}(g_i + \log \pi(a_i|s))\right]}{\sum_{j=1}^{k} \exp\left[\frac{1}{\tau}(g_j + \log \pi(a_i|s))\right]},$$

where τ is a "temperature" hyper-parameter that trades bias with variance. When τ approaches zero, the *softmax* operator acts like *argmax* ($a_{\text{softmax}} \approx a_{\text{argmax}}$) resulting in low bias. However, the variance of the gradient $\nabla_\theta a_{\text{softmax}}$ increases. Alternatively, when τ is set to a large value, the *softmax* operator creates a smoothing effect. This leads to low variance gradients, but at the cost of a high bias ($a_{\text{softmax}} \neq a_{\text{argmax}}$).

We use a_{softmax}, that is not necessarily "one-hot", to interact with the environment, which expects a single ("pure") action. We solve this by applying *argmax* over a_{softmax}, but use the **continuous approximation** in the backward pass by using the estimation: $\nabla_\theta a_{\text{argmax}} \approx \nabla_\theta a_{\text{softmax}}$.

[1]A general version of the re-parametrization trick for other distributions such as beta or gamma was recently proposed by Ruiz et al. (2016)

3.3. Backpropagating through a Forward model

So far we showed the changes necessary to use the exact partial derivative $\nabla_a D$. Incorporating the use of $\nabla_s D$ as well is a more involved and constitutes the **crux** of this work. To understand why, we can look at the block diagram of the model-free approach in Figure 2. There, s is treated as fixed (it is given as an input), therefore $\nabla_s D$ is discarded. On the contrary, in the model-based approach, s_t can be written as a function of the previous state and action: $s_t = f(s_{t-1}, a_{t-1})$, where f is the forward model. This way, using the law of total derivatives, we get that:

$$\nabla_\theta D(s_t, a_t)\Big|_{s=s_t, a=a_t} = \frac{\partial D}{\partial a}\frac{\partial a}{\partial \theta}\Big|_{a=a_t} + \frac{\partial D}{\partial s}\frac{\partial s}{\partial \theta}\Big|_{s=s_t} = \frac{\partial D}{\partial a}\frac{\partial a}{\partial \theta}\Big|_{a=a_t} + \frac{\partial D}{\partial s}\left(\frac{\partial f}{\partial s}\frac{\partial s}{\partial \theta}\Big|_{s=s_{t-1}} + \frac{\partial f}{\partial a}\frac{\partial a}{\partial \theta}\Big|_{a=a_{t-1}}\right). \tag{11}$$

Considering that a_{t-1} is a function of θ, we understand that by creating a computation graph that spans over more than a single time-step, we can *link* between $\nabla_s D$ and the policy. Put in words, during the backward pass, the error message regarding deviations of future states (ψ_s), propagates back in time and influences the actions of the policy in earlier times. Figure 3 summarizes this idea.

3.4. MGAIL Algorithm

We showed that a good approach for imitation requires: *(a)* to use a model, and *(b)* to process multi-step transitions. This setup was previously suggested by Shalev-Shwartz et al. (2016) and Heess et al. (2015), who built a multi-step computation graph for describing the familiar policy gradient objective, which in our case is given by: $J(\theta) = \mathbb{E}\Big[\sum_{t=0}\gamma^t D(s_t,a_t)\Big|\theta\Big]$. To show how to differentiate $J(\theta)$ over a trajectory of (s,a,s') transitions, we rely on the results of Heess et al. (2015):

$$J_s = \mathbb{E}_{p(a|s)}\mathbb{E}_{p(s'|s,a)}\Big[D_s + D_a\pi_s + \gamma J'_{s'}(f_s + f_a\pi_s)\Big], \tag{12}$$

$$J_\theta = \mathbb{E}_{p(a|s)}\mathbb{E}_{p(s'|s,a)}\Big[D_a\pi_\theta + \gamma(J'_{s'}f_a\pi_\theta + J'_\theta)\Big]. \tag{13}$$

The final policy gradient $\nabla_\theta J$ is calculated by applying Eq. 12 and 13 recursively, starting from $t = T$ all the way down to $t = 0$. The full algorithm is presented in Algorithm 1.

3.5. Forward Model Structure

The forward model prediction accuracy plays a crucial role in the stability of the learning process. However, learning

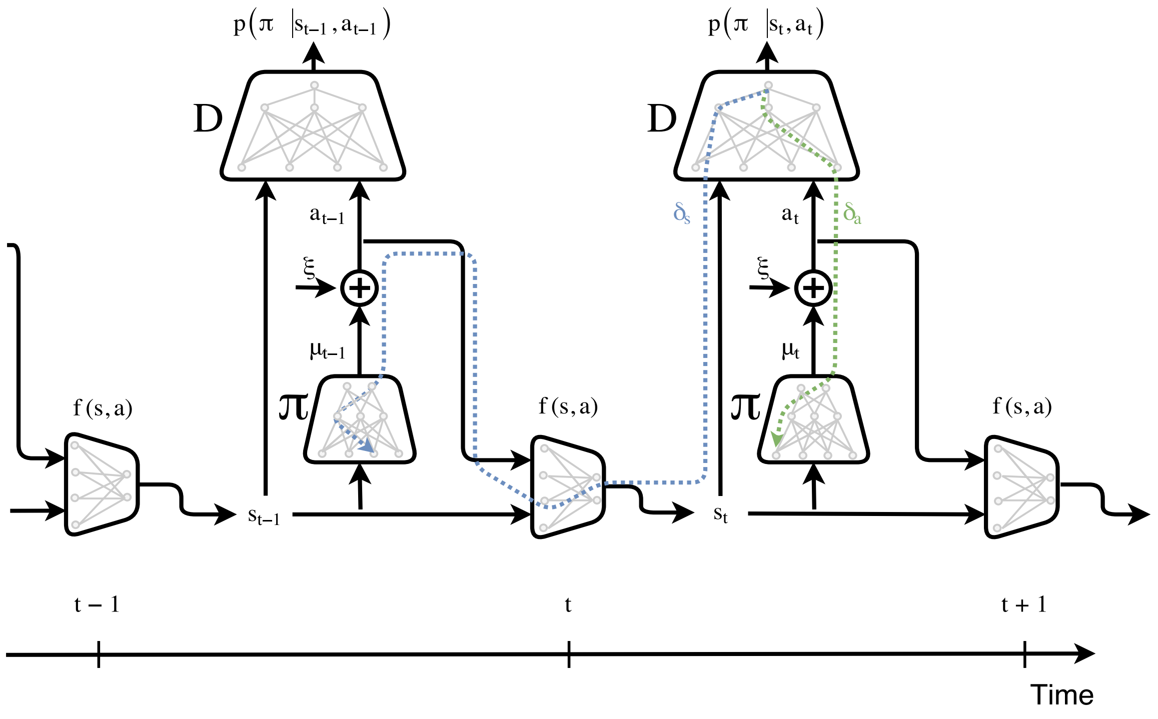


Figure 3. **Block diagram of model-based adversarial imitation learning.** This diagram describes the computation graph for training the policy (i.e. G). The discriminator network D is fixed at this stage and is trained separately. At time t of the *forward pass*, π outputs a distribution over actions: $\mu_t = \pi(s_t)$, from which an action a_t is sampled. For example, in the continuous case, this is done using the re-parametrization trick: $a_t = \mu_t + \xi \cdot \sigma$, where $\xi \sim \mathcal{N}(0, 1)$. The next state $s_{t+1} = f(s_t, a_t)$ is computed using the forward model (which is also trained separately), and the entire process repeats for time $t + 1$. In the *backward pass*, the gradient of π is comprised of *a.)* the error message δ_a (Green) that propagates fluently through the differentiable approximation of the sampling process. And *b.)* the error message δ_s (Blue) of future time-steps, that propagate back through the differentiable forward model.

Algorithm 1 **Model-based Generative Adversarial Imitation Learning**

```
 1: Input: Expert trajectories $\tau_E$, experience buffer $\mathcal{B}$, initial policy and discriminator parameters $\theta_g$, $\theta_d$
 2: for $trajectory = 0$ to $\infty$ do
 3:     for $t = 0$ to $T$ do
 4:         Act on environment: $a = \pi(s, \xi; \theta_g)$
 5:         Push $(s, a, s')$ into $\mathcal{B}$
 6:     end for
 7:     train forward model $f$ using $\mathcal{B}$
 8:     train discriminator model $D_{\theta_d}$ using $\mathcal{B}$
 9:     set: $j'_s = 0, j'_{\theta_g} = 0$
10:     for $t = T$ down to $0$ do
11:         $j_{\theta_g} = [D_a \pi_{\theta_g} + \gamma(j'_{s'} f_a \pi_{\theta_g} + j'_{\theta_g})]\big|_\xi$
12:         $j_s = [D_s + D_a \pi_s + \gamma j'_{s'}(f_s + f_a \pi_{\theta_g})]\big|_\xi$
13:     end for
14:     Apply gradient update using $j^0_{\theta_g}$
15: end for
```

an accurate forward model is a challenging problem by itself. We found that the performance of the forward model can be improved by considering the following two aspects of its functionality. First, the forward model should learn to use the action as an operator over the state space. Actions and states are sampled from entirely different distributions, so it would be preferable to first represent both in a shared space. Therefore, we first encode the state and action with two separate neural networks and then combine them to form a single vector. We found empirically that using a Hadamard product to combine the encoded state and action achieves the best performance. Additionally, predicting the next state based on the current state alone requires the environment to be representable as a first order MDP. Instead, we can assume the environment to be representable as an n'th order MDP and use multiple previous states to predict the next state. To model the multi-step dependencies, we use a recurrent connection from the previous state by incorporating a GRU layer (Cho et al., 2014) as part of the state encoder. Introducing these two modifications (see Figure 4), we found the complete model to achieve better

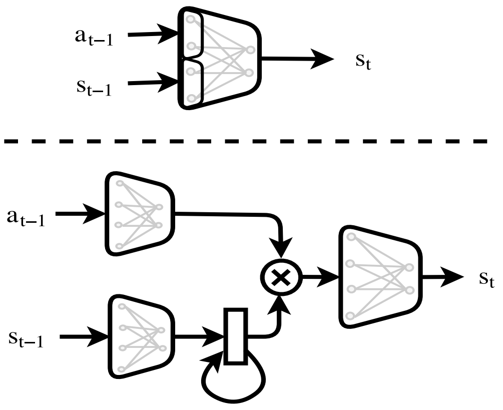


Figure 4. **Comparison between a vanilla forward model (Top) and an advanced forward model (Bottom).** In the vanilla forward model, the state and action vectors are concatenated and then passed through a two-layer neural network. In the advanced forward model, the action and state are passed independently through two neural networks and are then combined using Hadamard product. The result is then passed through another neural network to form the output.

and more stable results compared to using a vanilla feed-forward neural network as the forward model, as seen in Figure 5.

4. Experiments

We evaluate the proposed algorithm on three discrete control tasks (Cartpole, Mountain-Car, Acrobot), and five continuous control tasks (Hopper, Walker, Half-Cheetah, Ant, and Humanoid) modeled by the MuJoCo physics simulator (Todorov et al., 2012). These tasks involve complex second order dynamics and direct torque control. We use the Trust Region Policy Optimization (TRPO) algorithm (Schulman et al., 2015) to train expert policies. For each task, we produce datasets with a different number of trajectories, where each trajectory: $\tau = \{s_0, s_1, ...s_N, a_N\}$ is of length $N = 1000$.

The discriminator and policy neural networks are built from two hidden layers with Relu non-linearity and are trained using the ADAM optimizer (Kingma & Ba, 2014). Table 1 presents the total reward over a period of N steps, measured using three different algorithms: BC, GAIL, and MGAIL. The results for BC and GAIL are as reported in Ho & Ermon (2016). Our algorithm achieves the highest reward for most environments while exhibiting performance comparable to the expert over all of them (a Wilcoxon signed-rank test indicates superior performance with p-value < 0.05). We also compared the performance of MGAIL when using a basic forward model, versus using the more advanced model as described in Section 3. Figure 5 shows that better and more stable results are obtained when using the advanced forward model.

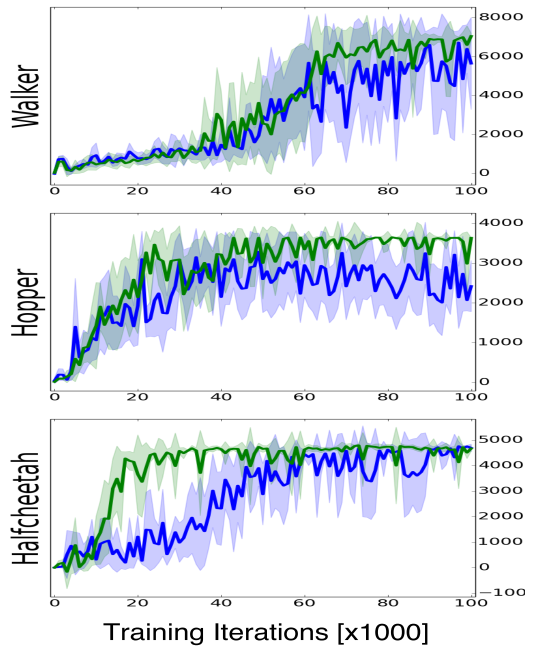
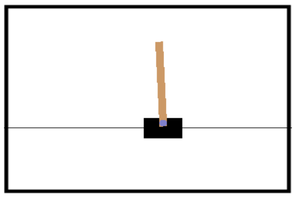


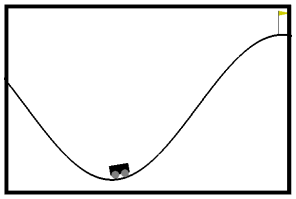
Figure 5. Performance comparison between a basic forward model (Blue), and the advanced forward model (Green).

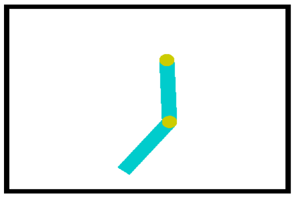
5. Discussion

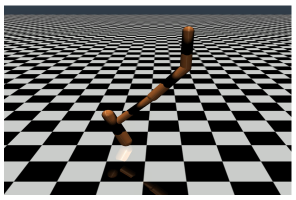
In this work, we presented a model-based algorithm for imitation learning. We showed how using a forward model enables to train policies using the exact gradient of the discriminator network. This way, the policy can imitate the expert's behavior, but also account for undesired deviations in the distributions of future states. The downside of this approach is the need to learn a forward model; a task that could prove difficult in some domains. An interesting line of future work would be to learn the system dynamics directly from raw images, as was done in Oh et al. (2015).

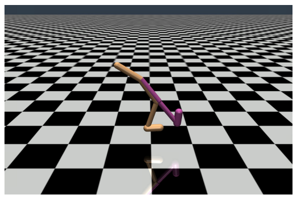
GANs algorithm violates a fundamental assumption made by all SL algorithms, which requires the data to be i.i.d. The problem arises because the discriminator network trains on a dynamic data distribution. For the training to succeed, the discriminator must continually adapt to the changes in the policy. In our context, the problem is emphasized even more since both the discriminator and the forward models are trained in a SL fashion using data that is sampled from a replay buffer $\mathcal{B}$ (Lin, 1993). A possible remedy is to *restart* the learning multiple times along the training period by resetting the learning rate (Loshchilov & Hutter, 2016). We tried this solution without significant success. However, we believe that further research in this direction is needed.

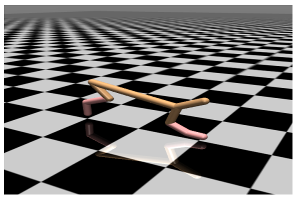

(a) Cartpole

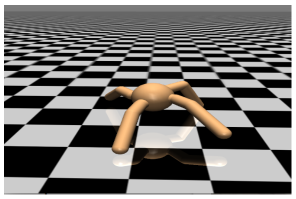

(b) MountainCar

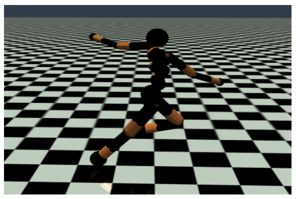

(c) Acrobot


(d) Hopper


(e) Walker


(f) Half-Cheetah


(g) Ant


(h) Humanoid

Task	Dataset size	Behavioral cloning	GAIL	MGAIL
Cartpole	1	72.02 ± 35.82	200.00 ± 0.00	200.00 ± 0.00
	4	169.18 ± 59.18	200.00 ± 0.00	200.00 ± 0.00
	7	188.60 ± 29.61	200.00 ± 0.00	200.00 ± 0.00
	10	177.19 ± 52.83	200.00 ± 0.00	200.00 ± 0.00
Mountain Car	1	-136.76 ± 34.44	-101.55 ± 10.32	-107.4 ± 10.89
	4	-133.25 ± 29.97	-101.35 ± 10.63	-100.23 ± 11.52
	7	-127.34 ± 29.15	-99.90 ± 7.97	-104.23 ± 14.31
	10	-123.14 ± 28.26	-100.83 ± 11.40	-99.25 ± 8.74
Acrobot	1	-130.60 ± 55.08	-77.26 ± 18.03	-85.65 ± 23.74
	4	-93.20 ± 35.58	-83.12 ± 23.31	-81.91 ± 17.41
	7	-96.92 ± 34.51	-82.56 ± 20.95	-80.74 ± 14.02
	10	-95.09 ± 33.33	-78.91 ± 15.76	-77.93 ± 14.78
Hopper	4	50.57 ± 0.95	3614.22 ± 7.17	3669.53 ± 6.09
	11	1025.84 ± 266.86	3615.00 ± 4.32	3649.98 ± 12.36
	18	1949.09 ± 500.61	3600.70 ± 4.24	3661.78 ± 11.52
	25	3383.96 ± 657.61	3560.85 ± 3.09	3673.41 ± 7.73
Walker	4	32.18 ± 1.25	4877.98 ± 2848.37	6916.34 ± 115.20
	11	5946.81 ± 1733.73	6850.27 ± 91.48	7197.63 ± 38.34
	18	1263.82 ± 1347.74	6964.68 ± 46.30	7128.87 ± 141.98
	25	1599.36 ± 1456.59	6832.01 ± 254.64	7070.45 ± 30.68
Half-Cheetah	4	-493.62 ± 246.58	4515.70 ± 549.49	4891.56 ± 654.43
	11	637.57 ± 1708.10	4280.65 ± 1119.93	4844.61 ± 138.78
	18	2705.01 ± 2273.00	4749.43 ± 149.04	4876.34 ± 85.74
	25	3718.58 ± 1856.22	4840.07 ± 95.36	4989.95 ± 351.14
Ant	4	1611.75 ± 359.54	3186.80 ± 903.57	4645.12 ± 179.29
	11	3065.59 ± 635.19	3306.67 ± 988.39	4657.92 ± 94.27
	18	2579.22 ± 1366.57	3033.87 ± 1460.96	4664.44 ± 183.11
	25	3235.73 ± 1186.38	4132.90 ± 878.67	4637.52 ± 45.66
Humanoid	80	1397.06 ± 1057.84	10200.73 ± 1324.47	10312.34 ± 388.54
	160	3655.14 ± 3714.28	10119.80 ± 1254.73	10428.39 ± 46.12
	240	5660.53 ± 3600.70	10361.94 ± 61.28	10470.94 ± 54.35

Table 1. Policy performance, boldface indicates better results, $\pm$ represents one standard deviation.

This research was supported in part by the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement 306638 (SUPREL) and the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

References

Cho, Kyunghyun, van Merrienboer, Bart, Gülçehre, Çaglar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

Dorigo, Marco and Colombetti, Marco. *Robot shaping: an experiment in behavior engineering*. MIT press, 1998.

Fu, Michael C. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

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

Gumbel, Emil Julius and Lieblein, Julius. Statistical theory of extreme values and some practical applications: a series of lectures. 1954.

Heess, Nicolas, Wayne, Gregory, Silver, David, Lillicrap, Tim, Erez, Tom, and Tassa, Yuval. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Ho, Jonathan and Ermon, Stefano. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kanamori, Takafumi and Shimodaira, Hidetoshi. Active learning algorithm using the maximum weighted log-likelihood estimator. *Journal of statistical planning and inference*, 116(1): 149–162, 2003.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lee, Honglak, Grosse, Roger, Ranganath, Rajesh, and Ng, Andrew Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pp. 609–616. ACM, 2009.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lin, Long-Ji. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

Loshchilov, Ilya and Hutter, Frank. Sgdr: Stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mathieu, Michael, Couprie, Camille, and LeCun, Yann. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

Mirza, Mehdi and Osindero, Simon. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Mnih, Andriy and Gregor, Karol. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Ng, Andrew Y, Russell, Stuart J, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.

Oh, Junhyuk, Guo, Xiaoxiao, Lee, Honglak, Lewis, Richard L, and Singh, Satinder. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pp. 2863–2871, 2015.

Pomerleau, Dean A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

Ranganath, Rajesh, Gerrish, Sean, and Blei, David M. Black box variational inference. In *AISTATS*, pp. 814–822, 2014.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Ross, Stéphane and Bagnell, Drew. Efficient reductions for imitation learning. In *AISTATS*, pp. 661–668, 2010.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, Drew. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, pp. 6, 2011.

Ruiz, Francisco R, AUEB, Michalis Titsias RC, and Blei, David. The generalized reparameterization gradient. In *Advances in Neural Information Processing Systems*, pp. 460–468, 2016.

Rusu, Andrei A, Colmenarejo, Sergio Gomez, Gulcehre, Caglar, Desjardins, Guillaume, Kirkpatrick, James, Pascanu, Razvan, Mnih, Volodymyr, Kavukcuoglu, Koray, and Hadsell, Raia. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

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

Shalev-Shwartz, Shai, Ben-Zrihem, Nir, Cohen, Aviad, and Shashua, Amnon. Long-term planning by short-term prediction. *arXiv preprint arXiv:1602.01580*, 2016.

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

Sutton, Richard Stuart. Temporal credit assignment in reinforcement learning. 1984.

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

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ziebart, Brian D, Maas, Andrew L, Bagnell, J Andrew, and Dey, Anind K. Maximum entropy inverse reinforcement learning. In *AAAI*, pp. 1433–1438, 2008.