GENERATIVE PREDECESSOR MODELS FOR SAMPLE-EFFICIENT IMITATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Generative Predecessor Models for Imitation Learning (GPRIL), a novel imitation learning algorithm that matches the state-action distribution to the distribution observed in expert demonstrations, using generative models to reason probabilistically about alternative histories of demonstrated states. We show that this approach allows an agent to learn robust policies using only a small number of expert demonstrations and self-supervised interactions with the environment. We derive this approach from first principles and compare it empirically to a state-of-the-art imitation learning method, showing that it outperforms or matches its performance on two simulated robot manipulation tasks and demonstrate significantly higher sample efficiency by applying the algorithm on a real robot.

1 INTRODUCTION

Training or programming agents to act intelligently in unstructured and sequential environments is a difficult and central challenge in the field of artificial intelligence. Imitation learning provides an avenue to tackle this challenge by allowing agents to learn from human teachers, which constitutes a natural way for experts to describe the desired behavior and provides an efficient learning signal for the agent. It is thus no surprise that imitation learning has enabled great successes on robotic (Chernova and Thomaz, 2014) as well as software domains (e.g. Aytar et al. (2018)). Yet, key challenges in the field are diverse and include questions such as how to learn from observations alone (e.g. Aytar et al. (2018)), learning the correspondence between the expert’s demonstrations and the agent’s observations (e.g. Sermanet et al. (2017)) as well as the question of how to integrate imitation learning with other approaches such as reinforcement learning (e.g. Vecerik et al. (2017)). However, at the core of the imitation learning problems lies the challenge of utilizing a given set of demonstrations to match the expert’s behavior as closely as possible. In this paper, we approach this problem considering the setting where the set of expert demonstrations is given up-front and the dynamics of the environment can only be observed through interaction.

In principle, the imitation learning problem could be seen as a supervised learning problem, where the demonstrations are used to learn a mapping from observed states to actions. This solution approach is known as behavioral cloning. However, it has long been known that the sequential structure of the task admits more effective solutions. In particular, the assumptions made in supervised learning are restrictive and don’t allow the agent to reason about the effect of its actions on it’s future inputs. As a result, errors and deviations from demonstrated behavior tend to accumulate over time as small mistakes lead the agent to parts of the observation space that the expert has not explored (Ross and Bagnell, 2010). In this work, we propose a novel imitation learning algorithm, Generative Predecessor Models for Imitation Learning (GPRIL), based on a simple core insight: Augmenting the training set with state-action pairs that are likely to eventually lead the agent to states demonstrated by the expert is an effective way to train corrective behavior and to prevent accumulating errors.

Recent advances in generative modeling, such as Goodfellow et al. (2014); Kingma and Welling (2013); Van Den Oord et al. (2016b,a); Papamakarios et al. (2017); Dinh et al. (2016), have shown great promise at modeling complex distributions and can be used to reason probabilistically about such state-action pairs. Specifically, we propose to utilize Masked Autoregressive Flows (Papamakarios et al., 2017) to model long-term predecessor distributions, i.e. distributions over state-action pairs which are conditioned on a state that the agent will see at some point in the future.
In this work, we train such a predecessor model by utilizing self-supervised interactions with the environment and propose an algorithm that iterates over the following steps:

1. Interact with the environment and observe state, action as well as a target state. To encode long-term corrective behavior, these states should be multiple steps apart.
2. Train a conditional generative model to produce samples like the observed state-action pair when conditioned on the observed target state.
3. Train the agent in a supervised way, augmenting the training set using data drawn from the model conditioned on the demonstrations. The additional training data shows the agent how to reach demonstrated states, enabling it to recover after deviating from expert behavior.

In the above, we laid out the sketch of an algorithm that intuitively learns to reason about the states it will observe in the future. In section 3 we derive this algorithm from first principles as a maximum likelihood approach to matching the state-action distribution of the agent to the expert’s distribution. In section 4, we compare our approach to a state-of-the-art imitation learning method (Ho and Ermon, 2016) and show that it matches or outperforms this baseline on our domains while being significantly more sample efficient. Furthermore, we show that our approach can be used to learn effectively using demonstrated states alone which allows for a wider variety of methods that the expert can use to record demonstrations. Together these properties are sufficient to allow GPRIL to be applied in real-world settings, which we demonstrate in section 4.3. To our knowledge this is the first instance of dynamic, contact-rich and adaptive behavior being taught solely using the kinesthetic-teaching interface of a collaborative robot, without resorting to tele-operation, auxiliary reward signals, or manual task-decomposition.

2 BACKGROUND

2.1 MARKOV DECISION PROCESSES WITHOUT REWARDS

As is usual, we model the problem as a Markov decision process without reward. That is, given state and action sets \( \mathcal{S}, \mathcal{A} \), the agent is observing states \( s \in \mathcal{S} \) and taking actions \( a \in \mathcal{A} \). In this work, we use \( s \) and \( a \) to refer to states and actions observed during self-supervision and \( \mathcal{S} \) and \( \mathcal{A} \) to refer to target and demonstration states and actions. We furthermore use superscripts \( s^{(i)} \) and \( \pi^{(i)} \) to refer to specific instances, e.g. specific demonstrated state-action pairs, and subscripts, e.g. \( s_t, a_t \), to indicate temporal sequences. The observed transitions are guided by the Markovian dynamics of the environment and the probability of transitioning from state \( s \) to state \( s' \) by taking action \( a \) is denoted as \( p(s_{t+1} = s' | s_t = s, a_t = a) \). The agent’s behavior is defined by a stationary parametric policy \( \pi_\theta(a|s) \) while the expert’s behavior is modeled by a stationary distribution \( \pi^* (s | a) \). We denote as \( d_{t}^s (s) \) the probability of observing state \( s \) at time-step \( t \) when following policy \( \pi \). Under the usual ergodicity assumptions, each such policy induces a unique stationary distribution of observed states \( d_t^\pi (s) = \lim_{t \to \infty} d_t^s (s) \) as well as a stationary joint state-action distribution \( \rho^\pi (s, a) = \pi(a|s) d_t^\pi (s) \). Furthermore, we use \( q_{t}^\pi \) to refer to the dynamics of the time reversed Markov chain induced by a particular policy \( \pi \) at time-step \( t \)

\[
q_{t}^\pi (s_t = s, a_t = a | s_{t+1} = s') = d_{t+1}(s')^{-1} d_{t}(s) \pi(a_t|s_t) p(s_{t+1} = s' | s_t = s, a_t = a) \tag{1}
\]

and define \( q^\pi (s_t = s, a_t = a | s_{t+1} = s') = \lim_{t \to \infty} q_{t}^\pi (s_t = s, a_t = a | s_{t+1} = s') \). For the purposes of this work, we handle the episodic case with clear termination conditions by adding artificial transitions from terminal states to initial states. This creates a modified, ergodic MDP with identical state-distribution and allows us to assume arbitrarily large \( t \) such that \( q_{t}^\pi = q^\pi \). Finally, we extend this notation to multi-step transitions by writing \( q^\pi (s_t = s, a_t = a | s_{t+j} = s') \).

2.2 IMITATION LEARNING

In this work, we are considering two settings of imitation learning. In the first setting, the agent is given a set of observed states \( \mathcal{S}^{(1)}, \mathcal{S}^{(2)}, \ldots, \mathcal{S}^{(N)} \) and observed corresponding actions \( \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N)} \) as expert demonstrations. The goal in this setting is to learn a policy \( \pi_\theta \) that matches the expert’s behavior as closely as possible. In the second setting, the agent is given the states observed by the expert but is not aware of the actions the expert has taken. Recent years have
seen heightened interest in a related setting where to goal is to track expert state trajectories (Zhu et al., 2018; Peng et al., 2018; Pathak et al., 2018). These approaches do not learn general policies that can adapt to unseen situations. A straightforward approach, usually referred to as behavioral cloning (BC), trains a policy in the first setting by treating the task as a supervised learning problem (e.g. Pomerleau (1989)). However, as outlined in section 1, predictions made by \( \pi_{\theta}(a|s) \) influence future observations thus violating a key assumption of supervised learning which states that inputs are drawn from an i.i.d. distribution. This has formally been analyzed by Ross et al. who introduce a family of algorithms (e.g. Ross and Bagnell (2010); Ross et al. (2011)) that provably avoid this issue. However, these approaches require the expert to continuously provide demonstrations and thus are not applicable when the set of demonstrations is assumed to be fixed. A popular avenue of research that considers this setting is inverse reinforcement learning (IRL). Inverse reinforcement learning (Ng and Russell, 2000) aims to learn a reward function for which \( \pi^* \) is optimal and thus captures the intent of the expert. The arguably most successful approach to IRL aims to match the state-action distribution to that of the demonstrations (Ziebart et al., 2008) with recent approaches extending these ideas to the model free case (Boularias et al., 2011; Finn et al., 2016; Fu et al., 2018).

However, inverse reinforcement learning is indirect and ill-defined as many reward functions induce the same behavior. Recently, methods have been proposed that aim to match state-action distributions directly and achieve state-of-the-art result without learning a reward function first. Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016) uses an adversarial objective, training a discriminator to identify demonstrations and using TRPO to train a policy that fools the discriminator. While GAIL is able to achieve impressive results, the adversarial objective can make the learning procedure unstable and unpredictable. This is especially true when parts of the state-space are not under the agent’s control, yielding a setting resembling a conditional GAN (Mirza and Osindero, 2014) which are prone to issues such mode collapse (Odena et al., 2017). We compare our approach with GAIL in section 4. State Aware Imitation Learning (SAIL) (Schroefl and Isbell, 2017) is an alternative method that aims to learn the gradient of the state-action distribution using a temporal-difference update rule. This approach is able to avoid instabilities prompted by the adversarial learning rule but is only applicable to policies with a small number of parameters where learning a representation of the gradient is feasible. In this work, we follow a gradient descent approach similar to SAIL but estimate the gradient without representing it explicitly by a neural network. These methods can also be used to learn from states alone by matching state distributions. While this does not necessarily induce a unique policy, including transition information such as velocities can allow the agent to learn solely from expert state trajectories nonetheless.

### 2.3 Generative Models

Recent years have seen great advances in deep generative models. A variety of approaches such as Generative Adversarial Networks (Goodfellow et al., 2014), Variational Auto Encoders (Kingma and Welling, 2013), autoregressive networks (e.g. Germain et al. (2015); Van Den Oord et al. (2016b,a) and normalizing flows (Dinh et al., 2016) have been proposed which enable us to learn complex distributions and efficiently generate samples. In this work, we use generative models to model the distribution of long-term predecessor state-action pairs. While the approach we propose is model agnostic, we choose to model this distribution using masked autoregressive flows (Papamakarios et al., 2017) (MAF). MAFs are trained using a maximum likelihood objective which allows for a stable and straight-forward training procedure. Autoregressive models are capable of representing complex distributions \( p(x); x \in \mathbb{R}^n \) by factoring the distribution \( p(x) = p_1(x_1) \prod_{i=1}^{N-1} p_{i+1}(x_{i+1}|x_1,\ldots,x_i) \) and learning a model for each \( p_i \). In this paper, we model each \( x_i \) to be distributed by \( x_i \sim \mathcal{N}(\mu_i,\sigma_i) \) where each \( \mu_i \) and \( \sigma_i \) is a function of \( x_{1:i}\). Masked autoencoders (Germain et al., 2015) provide a straight-forward approach to parameter sharing and allow representing these functions using a single network. MAFs stack multiple autoregressive models with different orderings and thus avoid the strong inductive bias imposed by the order of variables. Using the reparameterization trick, the autoregressive model can be seen as a deterministic and invertible transformation of a random variable: \( x_i = f_i(z_i) := \mu_i + \sigma_i z_i; z_i \sim \mathcal{N}(0,1) \). The change of variable formula then allows us to calculate the density of \( x_i \):

\[
\log p(x_i) = \log p_N(f_i^{-1}(x_i)) + \log \det(J(f_i^{-1}(x_i)))
\]

Where the autoregressive nature of \( f \) allows for tractable computation of the second term. MAFs chain multiple such transformations to derive highly expressive explicit density models able to model complex dynamics between target states and long-term predecessor state-action pairs.
3 GPRIL

In section 1, we provided an intuitive framework for using predecessor models to augment our training set and achieve robust imitation learning from few samples. In this section, we will derive this algorithm based on state-action distribution matching. To this end, we first derive the gradient of the logarithmic state distribution based on samples from a long-term predecessor distribution that we will define below. In section 3.2, we describe how to train a generative model of the predecessor distribution which will allow us to evaluate this gradient. Ascending on this gradient evaluated we will define below. In section 3.2, we describe how to train a generative model of the predecessor distribution which will allow us to evaluate this gradient. Ascending on this gradient evaluated at demonstrated states leads the agent to stick to those states and thus provides a corrective measure (Schroecker and Isbell, 2017). Furthermore, reproducing the states of the expert can be sufficient to achieve the correct behavior if the state-space is chosen appropriately as we will show in section 4. We will show how to use this gradient to match state-action-distributions in section 3.3.

3.1 REASONING OVER CHANGES TO THE STATIONARY STATE DISTRIBUTION

Here, we will show that the samples drawn from a long-term predecessor distribution conditioned on π enable us to estimate the gradient of the logarithmic state distribution ∇θ log dπ(π) and, later, to match the agent’s state-action-distribution to that of the expert. To achieve this goal, we can utilize the fact that the stationary state distribution of a policy can be defined recursively in terms of the state distribution at the previous time step, similar to Morimura et al. (2010):

\[
d^{π}(π) = \int d^{π}(s)π(π)(s|s_{t+1} = π, s_t = s, a_t = a)ds, a. \tag{3}
\]

Taking the derivative shows that this notion extends to the gradient as well as its logarithm:

\[
∇_θ d^{π}(π) = \int q^{π}(s, a)p(s_{t+1} = π|s_t = s, a_t = a) (∇_θ log d^{π}(s) + ∇_θ log π(π|s_t, s_{t+1} = π))ds, a \tag{4}
\]

\[
∇_θ log d^{π}(π) = \lim_{T→∞} \frac{1}{T} \sum_{j=0}^{T} q^{π}(s_t = s, a_t = a|s_{t+j+1} = π)∇_θ log π(π|s_t)ds, a \tag{5}
\]

The recursive nature of this gradient then allows us to unroll the gradient indefinitely. However, this process is cumbersome and will be left for appendix A. We arrive at the following equality:

\[
∇_θ log d^{π}(π) = \lim_{T→∞} \frac{1}{T} \sum_{j=0}^{T} q^{π}(s_t = s, a_t = a|s_{t+j+1} = π)∇_θ log π(π|s_t)ds, a \tag{6}
\]

The derivation of our approach now rests on two key insights: First, in ergodic Markov chains, such as the ones considered in our setting, decisions that are made at time t affect the probability of seeing state π at time t + j more strongly if j is small. In the limit, as j → ∞, the expectation of the gradient ∇_θ log π(π|s_t) remains finite and the decision at time t only adds variance to the gradient estimate. Introducing a discount factor γ similar to common practice in reinforcement learning.
(Sutton and Barto, 1998) places more emphasis on decisions that are closer in time and can thus greatly reduce variance. Second, by introducing a discount factor, the effective time-horizon is now finite. This allows us to replace the sum over all states and actions in each trajectory with a scaled expectation over state-action pairs. Formally, we can write this as follows and arrive at our main result:

\[ \nabla_\theta \log d^{\pi_\theta}(\pi) \approx \int \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}(s_t = s, a_t = a|s_{t+j+1} = \pi) \nabla_\theta \log \pi_\theta(a|s) ds, a \]

where \( B_\pi \) corresponds to the long-term predecessor distribution modeling the distribution of states and actions that, under the current policy \( \pi_\theta \), will eventually lead to the given target state \( \pi \):

\[ B_\pi(s, a|\pi) := (1 - \gamma) \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}(s_t = s, a_t = a|s_{t+j+1} = \pi) \]

### 3.2 Long-term Generative Predecessor Models

In the previous section, we derived the gradient of the logarithm of the stationary state distribution as approximately proportional to the expected gradient of the log policy, evaluated at samples obtained from the long-term predecessor distribution \( B_\pi \). In this work, we propose to train a model \( B_\pi \) to represent \( B_\pi \) and use its samples to estimate \( \nabla_\theta \log d^{\pi_\theta}(\pi) \). However, rather than unrolling a time-reversed Markov model in time, which is prone to accumulated errors, we propose to use a generative model to directly generate jump predictions. We have furthermore found that imposing a sensible order on autoregressive models achieves good results and thus propose to use two conditional MAFs (Papamakarios et al., 2017) \( B_{\omega_0}, B_{\omega_1} \) in a factored representation:

\[ B_{\omega_0}(s, a|\pi) := B_{\omega_0}(s|\pi) B_{\omega_1}(a|s, \pi). \]

To train this model, we collect training data using self-supervised roll-outs: We sample states, actions and target-states where the separation in time between the state and target-state is selected randomly based on the geometric distribution parameterized by \( \gamma \) as a training set for \( B_{\omega_0} \).

Training data for \( B_{\omega_0}, B_{\omega_1} \) are obtained by executing the current policy to obtain a sequence \( s_0, a_0, s_1, a_1, \ldots \), which we store in a replay buffer. In practice, we store data from multiple iterations in this buffer in order to decrease the variance of the gradient. While our algorithm does not explicitly account for off-policy samples, we found empirically that a short replay buffer does not degrade final performance while significantly improving sample efficiency. To obtain a training sample, we first pick \( s = s_0 \) and \( a = a_1 \) for a random \( t \). We now select a future state \( \pi = s_{t+j+1} \) from that sequence. For any particular \( s_{t+j+1} \) we now have \( s, a \sim q^{\pi_\theta}(s_t = s, a_t = a|s_{t+j+1} = \pi) \). Note that in the episodic case, we can add transitions from terminal to initial states and pick \( t \) to be arbitrarily large such that the approximate equality becomes exact (as outlined in section 2.1). In non-episodic domains, we find the approximation error to be small for most \( t \). Finally, we choose \( j \) at random according to a geometric distribution \( j \sim Geom(1 - \gamma) \) and have a training triple \( s, a, \pi \) that can be used to train \( B_{\omega_0} \) and \( B_{\omega_1} \) as it obeys

\[ s, a \sim (1 - \gamma) \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}(s_t = s, a_t = a|s_{t+j+1} = \pi) = B^{\pi_\theta}(s, a|\pi). \]

### 3.3 Matching state-action distributions with GPRIL

State-action distribution matching has been a promising approach to sample-efficient and robust imitation learning (see section 2.2). While each policy induces a unique distribution of states and behavioral cloning would therefore be sufficient in the limit, it is sub-optimal the case of limited data. Matching the joint-distribution directly ensures that we minimize discrepancies between everything we observed from the expert and the behavior the agent exhibits. In this work, we propose a maximum likelihood based approach, ascending on the estimated gradient of the joint distribution:

\[ \nabla_\theta \log \rho^{\pi}(\pi, \pi) = \nabla_\theta \log \pi_\theta(\pi|\pi) + \nabla_\theta \log d^{\pi_\theta}(\pi). \]
Under review as a conference paper at ICLR 2019

Figure 1: a) Depiction of the clip-insertion task. b) Median final success rate and interquartile range out of 100 roll-outs over 8 seeds. GPRIL achieves the highest success rate followed by GAIL. c) Median final success rate and IQR on clip insertion comparing sample efficiency. GPRIL is able to solve the task using several orders of magnitude fewer environment interactions. d) Comparison on clip insertion trained on states alone. Learning from states alone only slightly affects performance.

where $\nabla_{\theta} \log \pi_{\theta}(a|\pi)$ can be computed directly by taking the gradient of the policy using the demonstrated state-action pairs and $\nabla_{\theta} \log d^{\pi_{\theta}}(\pi)$ can be evaluated up to a constant factor using samples drawn from $B^{\pi_{\theta}}(\cdot, \cdot | s)$ according to equation 7. We introduce scaling factors $\beta_{\pi}$ and $\beta_{d}$ to account for this constant factor as well as to allow for finer control, interpolating between matching states only ($\beta_{\pi} = 0$) and behavioral cloning ($\beta_{d} = 0$). We arrive at the following estimate of the gradient:

$$\nabla_{\theta} \log \rho^{\pi_{\theta}}(s, a) \approx \beta_{\pi} \log \pi_{\theta}(a|s) + \beta_{d} E_{s,a \sim B^{\pi_{\theta}}(\cdot, \cdot | s)} \left[ \nabla_{\theta} \log \pi_{\theta}(a|s) \right].$$

(12)

This gives rise to the full-algorithm: We fill the replay buffer by asynchronously collecting experience using the current policy. Simultaneously, we repeatedly draw samples from the replay buffer to update the predecessor models and use expert samples in combination with an equal number of artificial samples to update the policy. This procedure is described fully in algorithm 1.

4 EXPERIMENTS

To evaluate our approach, we use a range of robotic insertion tasks similar to the domains introduced by Vecerik et al. (2017) but without access to a reward signal or, in some cases, expert actions. We choose these domains both for their practical use, and because they highlight challenges faced when applying imitation learning to the real world. Specifically, collecting experience using a robot arm is costly and demands efficient use of both demonstrations and autonomously gathered data. Furthermore, insertion tasks typically require complex searching behavior, particularly when the socket position is uncertain, and cannot be solved by open-loop tracking of a given demonstration trajectory. Insertion therefore poses a challenge for existing imitation learning and, especially, kinesthetic teaching approaches. We first compare against state-of-the-art imitation learning methods on a simulated clip insertion task, then explore the case of extremely sparse demonstrations on a simulated peg insertion task and finally, demonstrate real-world applicability on its physical counterpart.

4.1 CLIP INSERTION

In the first task, a simulated robot arm has to insert an elastic clip into a plug which requires the robot to first flex the clip in order to be able to insert it (see figure 1a). In real-world insertion tasks the
We report final results after convergence and can see that both GAIL and GPRIL outperform behavioral cloning, indicating that generalizing over state-action trajectories requires fewer demonstrations than generalizing over actions alone. Furthermore, we observe a higher success rate using GPRIL and find that policies trained using GPRIL are more likely to retry insertion if the robot slides the clip past the insertion point. To compare sample efficiency of GPRIL to GAIL, we limit the rate at which the asynchronous actor is collecting data. While sample efficiency of GAIL could be increased by decreasing batch size or increasing various learning rates, we found that this can lead to unstable learning performance while reducing the amount of samples required by only a small amount. As can be seen in figure 1c, GPRIL requires several orders of magnitudes fewer environment interactions to learn this task. Finally, we evaluate the case where the expert’s actions are positive, handling terminal transitions explicitly can induce a bias towards longer episodes. This is beneficial in the domains used by Ho and Ermon but harmful in domains such as ours where the task terminates when the robot leaves the work-space, reaches the goal, or after 50 seconds.

For comparative evaluation, we train a policy network to predict sufficient statistics of a multivariate normal distribution with independent dimensions over target velocities and train it using GPRIL, GAIL as well as behavioral cloning. We record expert demonstrations using tele-operation and normalize observations based on the recorded demonstrations. We then train GPRIL using a single asynchronous simulation and compare against the open source implementation of GAIL\(^1\) for which we use 16 parallel simulations. We select the best hyper parameters found on a grid around the hyperparameters used by Ho and Ermon (2016) but lower the batch size to 256 as it increases the learning speed and accounts for the significantly slower simulation of the task. We furthermore enable bootstrapping regardless of whether or not the episode terminated. As all discriminator rewards are positive, handling terminal transitions explicitly can induce a bias towards longer episodes. This is beneficial in the domains used by Ho and Ermon but harmful in domains such as ours where the task terminates on success. A detailed list of hyper-parameters can be found in appendix B.

\(^1\)https://github.com/openai/baselines/tree/master/baselines/gail
are unknown. Since the state-space includes information about joint velocities as well as positions, we find that matching the state-distribution is sufficient to solve the task. GPRIL can achieve this by setting $\beta_d = 1$ and $\beta_\pi = 0$. As can be seen in figure 1d, performance deteriorates only marginally with a similar difference in performance between both methods.

4.2 Peg insertion with partial demonstrations

The second task is a simulated version of the peg-insertion task depicted in figure 2a. In this task, the robot has to insert the peg into the hole which is again mounted on a pan-tilt that randomly assumes pan and tilt angles varying by 0.4 and 0.1 respectively. Hyperparameters are largely identical and we report minor differences in appendix B. Observation and action space are identical with the exception of the omission of torques from the observation space as they are not necessary to solve this task. We use this task to evaluate the performance of GAIL and GPRIL when learning from only a very limited set of demonstrated states. To this end, we compare three different scenarios in which the demonstrations are sparsified to varying degrees: In the first case, the agent has access to the full state-trajectories of the expert, in the second only every tenth state is available and in the third the agent sees only the final state of each of the 25 trajectories. Being able to learn from only partial demonstrations is a useful benchmark for the effectiveness of imitation learning methods but can also provide a convenient way of providing demonstrations and can free the agent to find more optimal trajectories between states (see for example Akgun et al. (2012); Schroecker et al. (2016)).

As can be seen in figures 2d and 2e, GPRIL achieves similar final success rates in all three scenarios while being able to learn a significantly faster insertion policy when learning from final states alone. We find that in the first two scenarios, this holds for GAIL as well as can been in figure 2b while in the third case, GAIL becomes highly unstable and the resulting performance can vary wildly, leading to a low average success rate. We hypothesize that these instabilities are a result of the discriminator overfitting to the very small amount of negative samples in its training data.

4.3 Peg insertion on a physical system

In previous sections we demonstrated sample-efficiency that indicates applicability of GPRIL to real-world physical systems. To test this, we evaluate our approach on two variations of the physical peg-insertion task depicted in figure 2a involving a physical pan-tilt unit which is fixed in one scenario and has pan and tilt angles varying by 0.1 and 0.02 radians in the second scenario. For each scenario we provide 20 demonstrations using kinesthetic teaching, which constitutes a natural way of recording demonstrations but provides state-trajectories only (Chernova and Thomaz, 2014). Hyper-parameters are altered from section 4.2 to trade off a small margin of accuracy for higher learning speeds and are reported in appendix B. Note, however, that tuning hyper-parameters precisely is very difficult on a physical system. As can be seen in figure 2f, GPRIL is able to learn a successful insertion policy that generalizes to unseen insertion angles using just a few hours of environment interactions\(^2\). We report best-seed performance as we observe a high amount of variability due to factors outside the agent’s control, such as the pan-tilt unit not reporting accurate information after physical contact with the robot. However, we wish to point out that the increased difficulty due to less predictable control is also likely to introduce additional variance that could be reduced further with careful design of exploration noise and other hyper-parameters. We furthermore provide a video of the training procedure and final policy to highlight the efficiency of our method\(^3\).

5 Conclusion

We introduced GPRIL, a novel algorithm for imitation learning which uses generative models to model multi-step predecessor distributions and to perform state-action distribution matching. We show that the algorithm compares favorably with state-of-the-art imitation learning methods, achieving higher or equivalent performance while requiring several orders of magnitude fewer environment samples. Important to stability and sample-efficiency of GPRIL are sufficient to enable experiments on a real robot, which we demonstrated on a peg-insertion task with a variable-position socket.

\(^2\)Total time to collect and train on 2000 roll-outs was 18.5 and 16.5 hours on the fixed and changing versions of the task respectively. However, GPRIL converged to good policies significantly sooner.

\(^3\)https://youtu.be/Dm0OCNujEmE
REFERENCES

Akgun, B., M. Cakmak, J. W. Yoo, and A. L. Thomaz

Aytar, Y., T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas

Boularias, A., J. Koerber, and J. Peters

Chernova, S. and A. L. Thomaz

Dinh, L., J. Sohl-Dickstein, and S. Bengio

Finn, C., S. Levine, and P. Abbeel

Fu, J., K. Luo, and S. Levine

Germain, M., K. Gregor, I. Murray, and H. Larochelle

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio

Ho, J. and S. Ermon

Kingma, D. P. and M. Welling

Mirza, M. and S. Osindero

Morimura, T., E. Uchibe, J. Yoshimoto, J. Peters, and K. Doya

Ng, A. and S. Russell

Odena, A., C. Olah, and J. Shlens

Papamakarios, G., I. Murray, and T. Pavlakou

Peng, X. B., P. Abbeel, S. Levine, and M. van de Panne  

Pomerleau, D. A.  

Ross, S. and D. Bagnell  

Ross, S., G. Gordon, and D. Bagnell  

Schroecker, Y., H. Ben Amor, and A. Thomaz  

Schroecker, Y. and C. L. Isbell  

Sermanet, P., C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine  

Sutton, R. S. and A. G. Barto  


Van Den Oord, A., N. Kalchbrenner, and K. Kavukcuoglu  


Ziebart, B. D., A. L. Maas, J. A. Bagnell, and A. K. Dey  
### A Derivation of Equation 6

Here, we derive equation 6 which unrolls the recursive definition of $\nabla_\theta \log d^{\pi_\theta}(s)$ and rewrites it such that it can be replaced by an expectation over states and actions along trajectories leading to the state $s$. To do so, we first unroll said definition:

$$\nabla_\theta \log d^{\pi_\theta}(s) = \lim_{T \to \infty} \int \left( \prod_{j=0}^{T-2} q^{\pi_\theta}(s_{t+j}, a_{t+j} | s_{t+j+1}) \right) \left( \nabla_\theta \log \pi_\theta(a|s) \right) ds,a$$

Note that $\lim_{T \to \infty} q^{\pi_\theta}(s_t = s, a_t = a|s_{t+T} = \pi) = \rho^{\pi_\theta}(s,a)$ due to Markov chain mixing and, therefore, the second term of the above sum reduces to 0 as

$$\int d^{\pi_\theta}(s) \pi(a|s) \nabla_\theta \log d^{\pi_\theta}(s) ds,a = 0. \quad (14)$$

By pulling out the sum, we can now marginalize out most variables and shift indices to arrive at the desired conclusion:

$$\nabla_\theta \log d^{\pi_\theta}(s) = \lim_{T \to \infty} \int \left( \prod_{j=0}^{T-2} q^{\pi_\theta}(s_{t+j}, a_{t+j} | s_{t+j+1}) \right) \nabla_\theta \log \pi_\theta(a|s) ds,a$$

$$= \lim_{T \to \infty} \sum_{j=0}^{T} \int q^{\pi_\theta}(s_t = s, a_t = a|s_{t+T+1-j} = \pi) \nabla_\theta \log \pi_\theta(a|s) ds,a \quad (15)$$

$$= \lim_{T \to \infty} \sum_{j=0}^{T} \int q^{\pi_\theta}(s_t = s, a_t = a|s_{t+T+1-j} = \pi) \nabla_\theta \log \pi_\theta(a|s) ds,a$$
## B Hyperparameters

### General Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total iterations</td>
<td>5e6</td>
</tr>
<tr>
<td>Batch size</td>
<td>256</td>
</tr>
<tr>
<td>Hidden layers</td>
<td>300, 200</td>
</tr>
<tr>
<td>Learning rate</td>
<td>10^4</td>
</tr>
<tr>
<td>σ bounds</td>
<td>(0.01, 0.1)</td>
</tr>
<tr>
<td>Network sizes</td>
<td>10000, 50000</td>
</tr>
<tr>
<td>L2-regularization</td>
<td>10^2</td>
</tr>
<tr>
<td>Gradient clip (L_2) norm</td>
<td>100</td>
</tr>
</tbody>
</table>

### Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>256</td>
</tr>
<tr>
<td>Hidden layers</td>
<td>500, 500</td>
</tr>
<tr>
<td>Learning rate</td>
<td>2 · 10^{-5}</td>
</tr>
<tr>
<td>Burnin</td>
<td>50000 iterations</td>
</tr>
<tr>
<td>L2-regularization</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>min(σ_i)</td>
<td>0.1</td>
</tr>
<tr>
<td>Policy πθ</td>
<td>300, 200</td>
</tr>
</tbody>
</table>

### Real Robot Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>0.9</td>
</tr>
<tr>
<td>Replay memory size</td>
<td>10000, 50000</td>
</tr>
<tr>
<td>Network sizes</td>
<td>10000, 5000</td>
</tr>
<tr>
<td>L2-regularization</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>min(σ_i)</td>
<td>0.1</td>
</tr>
<tr>
<td>Gradient clip (L_2) norm</td>
<td>100</td>
</tr>
</tbody>
</table>

### General Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total iterations</td>
<td>1e4</td>
</tr>
<tr>
<td>#Processes</td>
<td>16</td>
</tr>
<tr>
<td>Batch size</td>
<td>16 · 256</td>
</tr>
<tr>
<td>Actor steps/iteration</td>
<td>3</td>
</tr>
<tr>
<td>Discriminator steps/iteration</td>
<td>1</td>
</tr>
<tr>
<td>γ</td>
<td>0.995</td>
</tr>
<tr>
<td>Hidden layers</td>
<td>300, 200</td>
</tr>
<tr>
<td>KL step size</td>
<td>0.01</td>
</tr>
<tr>
<td>σ bounds</td>
<td>(0.01, 0.1)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>10^3</td>
</tr>
</tbody>
</table>

### Discriminator Parameters

<table>
<thead>
<tr>
<th>Layer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden layers</td>
<td>150, 100</td>
</tr>
<tr>
<td>Learning rate</td>
<td>10^3</td>
</tr>
<tr>
<td>Entropy regularization</td>
<td>1</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
</tbody>
</table>

### Critic Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden layers</td>
<td>300, 200</td>
</tr>
<tr>
<td>Learning rate</td>
<td>5 · 10^{-4}</td>
</tr>
</tbody>
</table>

### GAIL Parameters for clip insertion

- **GPRL**: Total iterations = 2e6, Batch size = 256, γ = 0.9, Replay memory size = 50000, N_B = 15000, N_π = 5000, B_s and B_a, Stacked autoencoders = 2, Hidden layers = 500, 500, Optimizer = Adam, Learning rate = 2 · 10^{-5}, Burnin = 50000 iterations, L2-regularization = 10^{-2}, min(σ_i) = 0.1, Gradient clip (L_2) norm = 100, Policy πθ = 300, 200, Optimizer = Adam, Learning rate = 10^4, σ bounds = (0.01, 0.1).

- **GAIL**: Total iterations = 1e4, #Processes = 16, Batch size = 16 · 256, Actor steps/iteration = 3, Discriminator steps/iteration = 1, γ = 0.995, Hidden layers = 300, 200, KL step size = 0.01, σ bounds = (0.01, 0.1), Optimizer = Adam, Learning rate = 10^3, Entropy regularization = 1, Optimizer = Adam, Learning rate = 5 · 10^{-4}.

### BC Parameters for clip insertion

- **Simulation**: Batch size = 256, γ = 0.9, Replay memory size = 10000, N_B = 10000, N_π = 5000, B_s and B_a, Stacked autoencoders = 2, Hidden layers = 500, 500, Optimizer = Adam, Learning rate = 10^{-3} · 10^{-5}, Burnin = 0, L2-regularization = 10^{-2} · 10^{-4}, min(σ_i) = 0.1, 0.01, Gradient clip (L_2) norm = 100, Policy πθ = 300, 200, Optimizer = Adam, Learning rate = 10^{-4}, σ bounds = (0.01, 0.1).

- **Real Robot**: Batch size = 0.7, Replay memory size = 50000, N_B = 50000, N_π = 10000, B_s and B_a, Stacked autoencoders = 2, Hidden layers = 500, 500, Optimizer = Adam, Learning rate = 3 · 10^{-5}, Burnin = 0, L2-regularization = 10^{-2} · 10^{-4}, min(σ_i) = 0.1, 0.01, Gradient clip (L_2) norm = 100, Policy πθ = 300, 200, Optimizer = Adam, Learning rate = 10^{-4}, σ bounds = (0.01, 0.1).

### GPRIL Parameters for peg insertion

- **GPRL**: Total iterations = 2e6, Batch size = 256, γ = 0.9, Replay memory size = 50000, N_B = 15000, N_π = 5000, B_s and B_a, Stacked autoencoders = 2, Hidden layers = 500, 500, Optimizer = Adam, Learning rate = 2 · 10^{-5}, Burnin = 50000 iterations, L2-regularization = 10^{-2}, min(σ_i) = 0.1, Gradient clip (L_2) norm = 100, Policy πθ = 300, 200, Optimizer = Adam, Learning rate = 10^{-4}, σ bounds = (0.01, 0.1).

- **GAIL**: Total iterations = 1e4, #Processes = 16, Batch size = 16 · 256, Actor steps/iteration = 3, Discriminator steps/iteration = 1, γ = 0.995, Hidden layers = 300, 200, KL step size = 0.01, σ bounds = (0.01, 0.1), Optimizer = Adam, Learning rate = 10^3, Entropy regularization = 1, Optimizer = Adam, Learning rate = 5 · 10^{-4}.

- **BC**: Batch size = 256, γ = 0.9, Replay memory size = 10000, N_B = 10000, N_π = 5000, B_s and B_a, Stacked autoencoders = 2, Hidden layers = 500, 500, Optimizer = Adam, Learning rate = 10^{-3} · 10^{-5}, Burnin = 0, L2-regularization = 10^{-2} · 10^{-4}, min(σ_i) = 0.1, 0.01, Gradient clip (L_2) norm = 100, Policy πθ = 300, 200, Optimizer = Adam, Learning rate = 10^{-4}, σ bounds = (0.01, 0.1).