# LEARNING EFFICIENT PLANNING-BASED REWARDS FOR IMITATION LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Imitation learning from limited demonstrations is challenging. Most inverse reinforcement learning (IRL) methods are unable to perform as good as the demonstrator, especially in high-dimensional environment, e.g, the Atari domain. To address this challenge, we propose a novel reward learning method, which streamlines a differential planning module with dynamics modeling. Our method learns useful planning computations with a meaningful reward function that focuses on the resulting region of an agent executing an action. Such planning-based reward function leads to policies that generalize well to new tasks. Empirical results with multiple network architectures and reward instances show that our method can outperform state-of-the-art IRL methods on multiple Atari games and continuous control tasks. Our method achieves performance that is averagely 1,139.1% of the demonstration.

## 1 INTRODUCTION

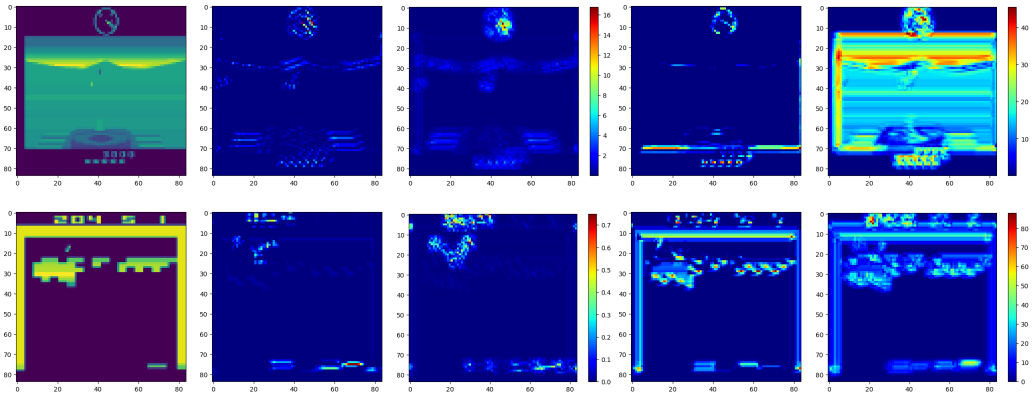
Imitation learning (IL) offers an alternative to reinforcement learning (RL) for training an agent, which mimics the demonstrations of an expert and avoids manually designed reward functions. Behavioral cloning (BC) (Pomerleau, 1991) is the simplest form of imitation learning, which learns a policy using supervised learning. More advanced methods, inverse reinforcement learning (IRL) (Ng & Russell, 2000; Abbeel & Ng, 2004) seeks to recover a reward function from the demonstrations and train an RL agent on the recovered reward function. In the maximum entropy variant of IRL, the aim is to find a reward function that makes the demonstrations appear near-optimal on the principle of maximum entropy (Ziebart et al., 2008; 2010; Boularias et al., 2011; Finn et al., 2016).

However, most state-of-the-art IRL methods fail to meet the performance of demonstrations in high-dimensional environments with limited demonstration data, e.g., a one-life demonstration in Atari game (Yu et al., 2020). This is due to the main goal of these IRL approaches is to recover a reward function that justifies the demonstrations only. The rewards recovered from limited demonstration data would be biased. Optimizing these rewards from an arbitrary initial policy results in inferior performance. Therefore, it is imperative to call for an effective reward function learning with less bias for achieving higher performance.

Differential planning modules learn to map observation to a planning computation for a task, and generates action predictions based on the resulting plan (Tamar et al., 2016; Nardelli et al., 2019; Zhang et al., 2020). For example, Tamar et al. (2016) proposed value iteration networks (VIN) that represents value iteration as a convolutional neural network (CNN). Meaningful reward and value maps have been learned along with the useful planning computation, which leads to policies that generalizes well to new tasks. Therefore, we believe that such module is potentially useful for achieving our goal of learning less based reward function for high-performance imitation learning. VIN and its many variants have shown good performance in the Grid World domain with simple transition dynamics. However, due to the inefficiency of summarizing transition dynamics in complex environments, VIN fails to scale up to Atari domain where the transition dynamics is more complicated.

To address this problem, we propose a novel method called variational planning-embedded reward learning (vPERL), which streamlines a differentiable planning module with transition dynamics modeling. Our method is composed of two submodules, one of which is based on the differential planning module for action back-tracing and the other is based on its inverse counterpart for transition dynamics modeling. We leverage a variational objective to jointly optimize the two submodules.

By explicitly modeling the transition dynamics, vPERL alleviates the difficulties of summarizing dynamics, and therefore makes it easier to learn meaningful reward and value maps as well as the useful planning computation in Atari domain. As shown in Figure 1, vPERL learns meaningful reward and value maps that focus on the resulting region of the agent executing an action, which is useful for high-performance imitation learning. However, directly applying VIN in Atari domain in the way of behavioral cloning (VIN/BC) as in the paper (Tamar et al., 2016) only learns reward and value maps that attend no specific region, which usually results in no avail.



(a) State. (b) vPERL Reward and Value Maps. (c) VIN/BC Reward and Value Maps.

Figure 1: Visualization of state, reward and value maps of vPERL and VIN/BC on Breakout game (the first row) and Battle Zone game (the second row).

Empirical results show that our method outperforms state-of-the-art IRL methods on multiple Atari games and continuous control tasks. Remarkably, our methods achieve performance that is up to 58 times of the demonstration. Moreover, the average performance improvement of our method is 1,139.1% of the demonstration over eight Atari games.

## 2 Background and Related Literature

**Markov Decision Process (MDP)**(Bellman, 1966) is a standard model for sequential decision making and planning. An MDP $M$ is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is the environment transition distribution, $R : \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor (Puterman, 2014). The expected discounted return or *value* of the policy $\pi$ is given by $V^\pi(s) = \mathbb{E}_\tau[\sum_{t=0} \gamma^t R(s_t, a_t)|s_0 = s]$, where $\tau = (s_0, a_0, s_1, a_1, \cdots)$ denotes the trajectory, in which the actions are selected according to $\pi$, $s_0 \sim \mathbb{T}_0(s_0)$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$. The goal in an MDP is to find the optimal policy $\pi^*$ that enables the agent to obtain high long-term rewards.

**Generative Adversarial Imitation Learning (GAIL)** (Ho & Ermon, 2016) extends IRL by integrating adversarial training technique for distribution matching (Goodfellow et al., 2014). Although GAIL benefits from adversarial training, it also inherits problems, such as instability in the training process. GAIL performs well in low-dimensional applications, e.g., MuJoCo. However, it does not scale well to high-dimensional scenarios, such as Atari games (Brown et al., 2019a). **Variational adversarial imitation learning (VAIL)** (Peng et al., 2019) improves on GAIL by compressing the information via variational information bottleneck. We have included both methods as comparisons to vPERL in our experiments.

**Generative Intrinsic Reward driven Imitation Learning (GIRIL)** (Yu et al., 2020) leverage generative model to learn generative intrinsic rewards for better exploration. As a result, GIRIL achieves better imitation learning performance than several IRL methods on limited demonstrations. However, the nature of the prediction error based intrinsic reward function reaches a performance plateau when handling stochastic dynamics (Pathak et al., 2017; Burda et al., 2019). The agent might get stuck at a region with stochasticity because the reward is always high. While our vPERL learns meaningful

planning-based reward function from demonstration data, and does not suffer from such problem as GIRIL. We have included GIRIL as a competitive baseline in our experiments.

**Differentiable planning modules (DPM)** perform end-to-end learning of planning computation, which leads to policies that generalize to new tasks. Value iteration networks (VIN) (Tamar et al., 2016) proposes to embed value iteration (VI) (Bellman, 1957) process with a recurrent convolutional network, and generalizes well in conventional navigation domains. Nardelli et al. (2019) proposes value propagation networks (VPN) generalize VIN for better sample complexity by employing value propagation (VProp). Recently, universal value iteration networks (UVIN) extends VIN to spatially variant MDPs (Zhang et al., 2020). Although VIN can be extended to irregular spatial graph by applying graph convolutional operator (Niu et al., 2018), most of the VIN variants still focus on solving the conventional navigation problems (Zhang et al., 2020).

In this paper, we extends differentiable planning module to learn efficient reward function for imitation learning on limited demonstration data. We dig more on leveraging the learned reward function that enables agent to generalize to complicated task; while previous related work of VIN focus more on the value function. Therefore, our work is complementary to the research of VIN and its variants. Note that any differentiable planning module can be embedded in our method. As a simple example, we utilize the basic VIN as a backbone to build our reward learning module.

## 3 Variational Planning-embedded Reward Learning

In this section, we introduce our solution, variational planning-embedded reward learning (vPERL). As illustrated in Figure 2, our reward learning module is composed of two submodules to accomplish planning-embedded action back-tracing and explicit forward transition dynamics modeling.

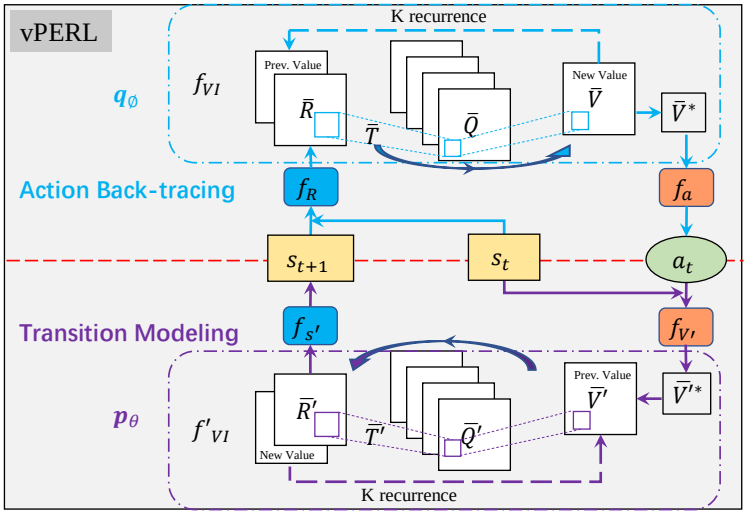



Figure 2: Illustration of the intrinsic reward inference procedure of the proposed vPERL model.

### 3.1 Action Back-tracing and Forward Dynamics Modelling in vPERL

**Planning-embedded action back-tracing.** Instead of directly applying VIN for policy learning (Tamar et al., 2016), we build our first submodule $q_\phi(a_t|s_t, s_{t+1})$ for action back-tracing. As illustrated in the top section of Figure 2, we first obtain the reward map $\overline{R} = f_R(s_t, s_{t+1})$ on an embedded MDP $\overline{M}$, where $f_R$ is a convolutional layer. A VI module $f_{VI}$ takes in the reward map $\overline{R}$, and effectively performs $K$ times of VI by recurrently applying a convolutional layer $\overline{Q}$ for $K$ times (Tamar et al., 2016). The $\overline{Q}$ layer is then max-pooled to obtain the next-iteration value $\overline{V}$. The right-directed circular arrow in a light-blue color denotes the direction of convolutions. Then, we simply obtain the action from the intermediate optimal value $\overline{V}^*$ by an action mapping function: $a_t = f_a(\overline{V}^*)$. On these terms, we build our planning-embedded action back-tracing

submodule, which is formally represented as $q_\phi(a_t|s_t, s_{t+1}) = f_a(f_{VI}(f_R(s_t, s_{t+1})))$. Since the convolutional kernel is incapable of summarizing the transition dynamics in a complex environment, directly training this submodule is still insufficient for learning efficient reward function and planning computation in an environment like Atari domain.

**Explicit transition dynamics modeling via inverse VI.** To address this, we further build upon another submodule $p_\theta(s_{t+1}|a_t, s_t)$ for explicit transition dynamics modeling. We build the submodule based on the inverse VI module, which is a NN architecture that mimics the process of inverse version of VI. The implementation of the inverse VI module is straightforward. We first map the action for the intermediate optima value in another embedded MDP $\overline{M'}$ by a value mapping function: $\overline{V'}^* = f_{V'}(s_t, a_t)$. Then, we apply the inverse VI module to obtain the reward map $\overline{R'}$. The inverse VI module $f'_{VI}$ takes in the intermediate value $\overline{V'}$ and recurrently apply a deconvolutional layer $\overline{Q'}$ for $K$ times on the value to obtain the reward map $\overline{R'}$. The left-directed circular arrow in a purple color denotes the direction of deconvolutions. To accomplish the transition, we map the obtained $\overline{R'}$ to the future state by: $s_{t+1} = f_{s'}(\overline{R'})$. The transition modeling is therefore presented as $p_\theta(s_{t+1}|a_t, s_t) = f_{s'}(f'_{VI}(f_{V'}(s_t, a_t)))$, which is a differentiable submodule, and can be trained simultaneously with the action back-tracing submodule.

**Variational solution to vPERL.** To handle both action back-tracing and forward dynamics modeling simultaneously, we introduce a variational solution to optimize the both submodules jointly. Our vPERL module is maximized with the following objective:

$$\begin{aligned}\mathcal{L}(s_t, s_{t+1}; \theta, \phi) = \mathbb{E}_{q_\phi(z|s_t, s_{t+1})}[\log p_\theta(s_{t+1}|z, s_t)] - \mathrm{KL}(q_\phi(z|s_t, s_{t+1}) \| p_\theta(z|s_t)) \\ - \alpha \mathrm{KL}(q_\phi(\hat{a}_t|s_t, s_{t+1}) \| \pi_E(a_t|s_t))]\end{aligned} \tag{1}$$

where $z$ is the latent variable, $\pi_E(a_t|s_t)$ is the expert policy distribution, $\hat{a}_t = \mathrm{Softmax}(z)$ is the transformed latent variable, $\alpha$ is a positive scaling weight. The first two terms on the RHS of Eq. (1) in the first line denote the evidence lower bound (ELBO) of the conditional VAE (Sohn et al., 2015). These two terms are critical for our reward learning module to perform planning-based action back-tracing and transition modeling. Additionally, we integrate the third term on the RHS of Eq. (1) in the second line to further boost the action back-tracing. The third term minimizes the KL divergence between the expert policy distribution $\pi_E(a_t|s_t)$ and the action distribution $q_\phi(\hat{a}_t|s_t, s_{t+1})$, where $\hat{a}_t = \mathrm{Softmax}(z)$ is transformed from the latent variable $z$. In this way, we train the forward state transition and action back-tracing simultaneously.

Note that the full objective in Eq. (1) is still a variational lower bound of the marginal likelihood $\log(p_\theta(s_{t+1}|s_t))$. Accordingly, it is reasonable to maximize this as an objective of our reward learning module. By optimizing the objective, we improve the forward state transition and action back-tracing. As a result, our reward learning module efficiently models the transition dynamics of the environment. During training, we use the latent variable $z$ as the intermediate action. After training, we will calculate the surrogate rewards from the learned reward map. As shown in Figure 1, our method learns meaningful reward map, which highlights the resulting region of an agent executing one action. To leverage such meaningful information, we calculate two types of rewards that both correspond to the highlighted informative region, i.e., $r_t = R_{\mathrm{Max}} = \max \overline{R}$ and $r_t = R_{\mathrm{Mean}} = \mathrm{mean}\overline{R}$, which uses the maximum and mean value of the reward map $\overline{R}$, respectively.

**Algorithm 1** Imitation learning via variational planning-embedded reward learning (vPERL).

1: **Input:** Expert demonstration data $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$.
2: Initialize policy $\pi$, and the dual planning networks.
3: **for** $e = 1, \cdots, E$ **do**
4: Sample a batch of demonstration $\tilde{\mathcal{D}} \sim \mathcal{D}$.
5: Train vPERL module on $\tilde{\mathcal{D}}$ to converge.
6: **end for**
7: **for** $i = 1, \cdots, \mathrm{MAXITER}$ **do**
8: Update policy via any policy gradient method, e.g., PPO on the learned surrogate reward $r_t$.
9: **end for**
10: **Output:** Policy $\pi$.

Algorithm 1 summarizes the full training procedure of imitation learning via vPERL. The process begins by training a vPERL module for $E$ epochs (steps 3-6). In each training epoch, we sample a

mini-batch demonstration data $\tilde{\mathcal{D}}$ with a mini-batch size of $B$ and maximize the objective in Eq. (1). Then in steps 7-9, we update the policy $\pi$ via any policy gradient method, e.g., PPO (Schulman et al., 2017), so as to optimize the policy $\pi$ with the learned surrogate reward function $r_t$.

# 4 EXPERIMENTS

## 4.1 ATARI GAMES

We first evaluate our proposed vPERL on one-life demonstration data for eight Atari games within OpenAI Gym (Brockman et al., 2016). To enable a fair comparison, we evaluate our method and all other baselines under the same standard setup, where we train an agent to play Atari games without the access to the true reward function (Ibarz et al., 2018; Brown et al., 2019a). The games and demonstration details are provided in Table 1.

Table 1: Statistics of Atari environments.

| | Demonstration Length | | # Lives |
|---|---|---|---|
| Game | One-life | Full-episode | available |
| Kung-Fu Master | 1,167 | 3,421 | 4 |
| Battle Zone | 260 | 1,738 | 5 |
| Centipede | 166 | 663 | 3 |
| Seaquest | 562 | 2,252 | 4 |
| Q*bert | 787 | 1,881 | 4 |
| Breakout | 1,577 | 2,301 | 5 |
| Beam Rider | 1,875 | 4,587 | 3 |
| Space Invaders | 697 | 750 | 3 |

A one-life demonstration only contains the states and actions performed by an expert player until they lose their life in a game for the first time (Yu et al., 2020). In contrast, one full-episode demonstration contains states and actions after the expert player loses all available lives in a game. Therefore, the one-life demonstration data is much more limited than one full-episode demonstration. We define three levels of performance: 1) basic one-life demonstration-level - gameplay up to one life lost (“one-life"), 2) expert-level - gameplay up to all-lives lost (“one full-episode"), and 3) beyond expert - “better-than-expert" performance. Our ultimate goal is to train an imitation agent that can achieve a better-than-expert performance with the demonstration data recorded up to the moment of losing their first life in the game.

**Demonstrations** To generate one-life demonstrations, we trained a PPO (Schulman et al., 2017) agent with the ground-truth reward for 10 million simulation steps. We used PPO implementation with the default hyper-parameters in the repository (Kostrikov, 2018). As Table 1 shows, the one-life demonstrations are all much shorter than the full-episode demonstrations, which make for extremely limited training data.

**Experimental Setup** Our first step was to train a reward learning module for each game on the one-life demonstration. We set $K = 10$ in vPERL for all of the Atari games. By default, we use a neural network architecture that keeps the size of reward map and value maps the same as that of the state, which is $84 \times 84$. We achieve this by using a convolutional kernel of size 3 for each convolutional layer, and applying padding. The corresponding method is called ‘vPERL-Large’. Additionally, to enable faster learning, we implement our method with another neural network architecture that reduces the size of reward map and value maps into $18 \times 18$. The corresponding method is called ‘vPERL-Small’. Both vPERL-Large and vPERL-Small are able to learn meaningful reward map as well as useful planning computation. Training was conducted with the Adam optimizer (Kingma & Ba, 2015) at a learning rate of 3e-5 and a mini-batch size of 32 for 50,000 epochs. In each training epoch, we sampled a mini-batch of data every four states. Detailed architectures of both our vPERL submodules are summarized in the Appendix.

To evaluate the quality of our learned reward, we trained a policy to maximize the inferred reward function via PPO. We set $\alpha = 100$ for training our reward learning module. We trained the PPO on the learned reward function for 50 million simulation steps to obtain our final policy. The PPO is trained with a learning rate of 2.5e-4, a discount factor of 0.99, a clipping threshold of 0.1, an entropy

coefficient of 0.01, a value function coefficient of 0.5, and a GAE parameter of 0.95 (Schulman et al., 2016). We compared imitation performance by our vPERL agent against VIN/BC, two state-of-the-art inverse reinforcement learning methods, GAIL (Ho & Ermon, 2016) and VAIL (Peng et al., 2019). More details of setup are outlined in the Appendix.

**Results** In Figure 3, we report the performance by normalizing the demonstration performance to 1. Figure 3 shows that vPERL achieves performance that is usually close or better than that of the expert demonstrator. The most impressive one is the Centipede game, our vPERL achieves performance that is around 60 times higher than the demonstration. GIRIL achieves the second best performance in Centipede, beating the demonstration by around 30 times. On the Qbert game, vPERL beats all other baselines by a large margin, achieving performance that is more than 15 times of the demonstration.

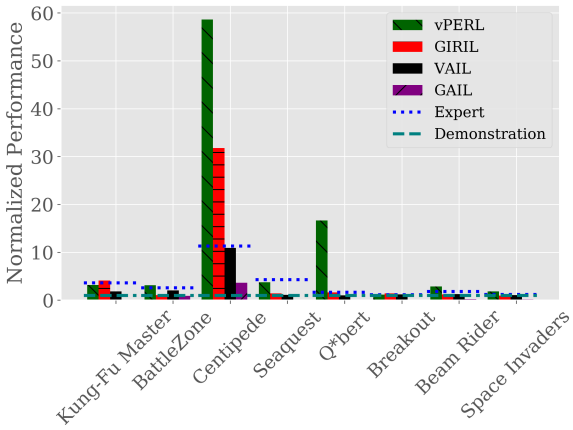


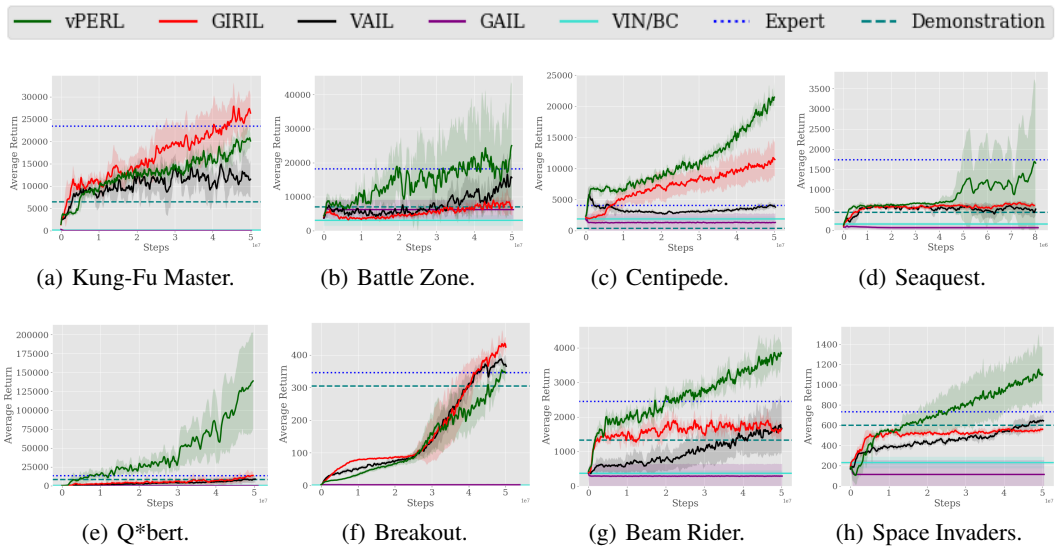

Figure 3: Performance improvement of vPERL and baselines.

Table 2: Average return of vPERL with two types network architectures (Large and Small) and two types of rewards ($R_{\text{Max}}$ and $R_{\text{Mean}}$), GIRIL (Yu et al., 2020), BC and state-of-the-art IRL algorithms GAIL (Ho & Ermon, 2016) and VAIL (Peng et al., 2019) with one-life demonstration data on four Atari games. The results shown are the mean performance over five random seeds with better-than-expert performance in bold. The last two rows show the average performance improvements of IL algorithms versus demonstration (Demo.) and expert performance with greater than 100% in bold.

| | Expert | Demo. | vPERL-Large | | vPERL-Small | | Imitation Learning Algorithms | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Game | Average | Average | $R_{\text{Max}}$ | $R_{\text{Mean}}$ | $R_{\text{Max}}$ | $R_{\text{Mean}}$ | GIRIL | VAIL | GAIL | VIN/BC |
| Kung-Fu Master | 23,434.8 | 6,500 | 20,700.0 | 18,900.0 | 20,800.0 | 20,518.3 | **26,742.6** | 11,751.2 | 1.3 | 190.4 |
| Battle Zone | 18,137.5 | 7,000.0 | **27,500.0** | 11000.0 | **47,200.0** | **21,899.1** | 7,372.8 | 14,222.7 | 6,200.0 | 3,372.0 |
| Centipede | 4,069.4 | 360.0 | **15,516.9** | **10,448.8** | **17,371.4** | **21,113.4** | **11,446.5** | 3,936.1 | 1,306.5 | 1,907.6 |
| Seaquest | 1,744.5 | 440.0 | 706.0 | 756.0 | 816.0 | 1,634.2 | 624.8 | 476.1 | 60.0 | 148.0 |
| Q*bert | 13,441.5 | 8,150.0 | **71,920.0** | **165,325.0** | **58,965.0** | **135,598.0** | 13,070.3 | 8,314.8 | 25.0 | 283.4 |
| Breakout | 346.4 | 305.0 | **374.7** | **420.0** | **422.3** | **348.8** | **430.3** | **370.6** | 2.0 | 0.4 |
| Beam Rider | 2,447.7 | 1,332.0 | **3,264.6** | **4,890.0** | **2,960.5** | **3,801.8** | 1,599.5 | 1,710.8 | 286.5 | 351.4 |
| Space Invaders | 734.1 | 600.0 | **915.2** | **1,135.0** | **1,325.5** | **1,106.5** | 556.6 | 646.2 | 113.9 | 227.4 |
| Average | **345.8%** | 100.0% | **823.1%** | **780.5%** | **913.8%** | **1,139.1%** | **544.0%** | **255.6%** | 63.3% | 85.3% |
| Improvements (%) | 100.0% | 47.8% | **194.9%** | **267.9%** | **210.1%** | **278.5%** | **103.9%** | 72.1% | 12.2% | 15.2% |

A detailed quantitative comparison of IL algorithms are listed in Table 2. We have evaluated four variants of vPERL with two types of network architectures (Large and Small) and surrogate rewards ($R_{\text{Max}}$ and $R_{\text{Mean}}$). Both vPERL-Large and vPERL-Small can learn meaningful reward and value maps as well as useful computation in Atari domain. In the Appendix B and C, we visualize the learned reward and value maps of vPERL-Large and vPERL-Small, respectively. With such meaningful rewards, the four variants of vPERL outperform the expert demonstrator in six out of eight Atari games. Remarkably, vPERL-Small with $R_{\text{Mean}}$ achieves an average performance that is 1,139.1% of the demonstration and 278.5% of the expert over the eight Atari games. Figure 3 shows the bar plot of normalized performance of out best variant against other baselines. The full version of bar plot comparison is illustrated in Appendix A.

Table 2 shows that VIN/BC is far from achieving demonstration-level performance, since it is unable to learn useful planning computation as well as the meaningful reward and value maps in Atari domain. GAIL fails to achieve good imitation learning performance. VAIL manages to exceed the expert performance in one game, i.e., Breakout. GIRIL performs better than previous methods, outperforming the expert demonstrator in three games. The results show that our vPERL agent outperforms the expert demonstrator by a large margin in six out of eight games.




(a) Kung-Fu Master. (b) Battle Zone. (c) Centipede. (d) Seaquest.

(e) Q*bert. (f) Breakout. (g) Beam Rider. (h) Space Invaders.

Figure 4: Average return vs. the number of simulation steps on Atari games. The solid lines show the mean performance over five random seeds. The shaded area represents the standard deviation from the mean. The blue dotted line denotes the average return of the expert. The area above the blue dotted line indicates performance beyond the expert.

Figure 4 shows the average performance of the expert, demonstration and IL methods with five random seeds. In Figure 4, we only show the learning curves of our best vPERL variant, i.e., vPERL-Small with $R_{\text{Mean}}$, and other baselines.

**Analysis:**

**vPERL vs. VIN/BC.** When applying VIN to complex video games, e.g. the Atari games, VIN fails to learn good planning computation due to its inefficiency of summarizing the complex dynamics. By explicitly modeling the transition dynamics, our vPERL generalizes the planning computation learning ability of VIN from a simple task to complicated video games. As shown in Figure 1, the learned reward and value maps of vPERL clearly focus on the resulting region of the action taking by the agent. This means that vPERL is able to learn good planing computation in Atari games; while VIN/BC cannot.

**vPERL vs. GAIL and VAIL.** Despite their distribution matching ability, GAIL and VAIL do not consider any mechanism to model environment dynamics. Our vPERL does not depend on distribution matching. Instead, vPERL learns useful reward functions by planning-embedded action back-tracing and explicitly modeling the transition dynamics of MDP in the environment. Under such design, variational training of the vPERL module learns an efficient surrogate planning-based reward function. Such planning-based reward function can attend meaningful resulting region of the agent taking one action. As a result, it leads to policy that can generalize to new domains. Moreover, our method does not suffer from the problem of adversarial training, and is potentially to outperform the expert instead of matching distribution of the limited demonstration data.

**vPERL vs. GIRIL.** Although GIRIL can outperform several IRL methods by applying generative intrinsic rewards, it has an potential limitation when handling stochastic dynamics. Our vPERL has better generalization ability than GIRIL by learning efficient planning-based reward function. As demonstrated in the above experiments, vPERL is generally better than GIRIL. There is another reason why vPERL can perform better than GIRIL. GIRIL utilizes the prediction-error based intrinsic reward function training agent. The images of prediction-error can be less informative if the prediction is accurate. Furthermore, GIRIL has few chances to capture informative regions of states as what we have achieved by vPERL. Therefore, vPERL can achieves better performance due to better ability of learning informative reward functions.

### 4.2 CONTINUOUS CONTROL TASKS

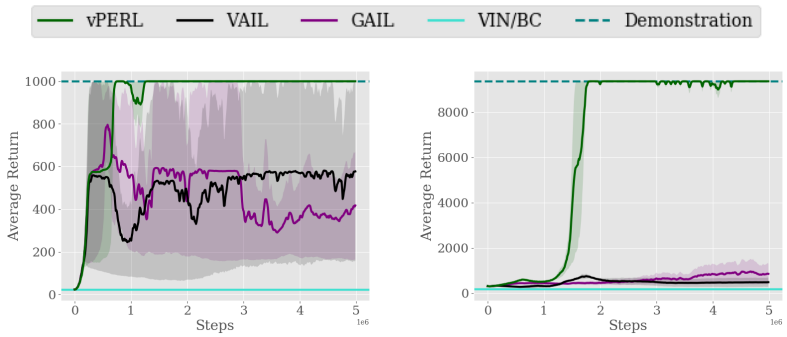
We also evaluated our method on continuous control tasks where the state space is low-dimensional and the action space is continuous. The continuous control tasks were from Pybullet[1] environment.

**Demonstrations** To generate demonstrations, we trained a Proximal Policy Optimization (PPO) agent with the ground-truth reward for 1 million simulation steps. We used the PPO implementation in the repository (Kostrikov, 2018) with the default hyper-parameters for continuous control tasks. In each task, we used one demonstration with a fixed length of 1,000 for evaluation.

**Results** Table 3 shows the detailed quantitative comparison of the demonstration and imitation methods. The results shown in the table were the mean performance over three random seeds. Under a fair comparison, our method vPERL achieves the best imitation performance in both continuous control tasks, i.e. Inverted Pendulum and Inverted Double Pendulum. A comparison of full learning curves can be found in Figure 5.

Table 3: Average return of vPERL, GIRIL, VIN/BC and state-of-the-arts inverse reinforcement learning algorithms GAIL (Ho & Ermon, 2016) and VAIL (Peng et al., 2019) with one demonstration data on continuous control tasks. The results shown are the mean performance over 3 random seeds with best imitation performance in bold.

| | Demonstration | vPERL | Imitation Learning Algorithms | | | |
|---|---|---|---|---|---|---|
| Task | Average | $R_{\text{Mean}}$ | GIRIL | VAIL | GAIL | VIN/BC |
| Inverted Pendulum | 1,000.0 | **1,000.0** | 990.2 | 520.1 | 572.2 | 21.0 |
| Inverted Double Pendulum | 9,355.1 | **9,357.8** | 9,164.9 | 539.8 | 519.6 | 181.1 |




(a) Inverted Pendulum. (b) Inverted Double Pendulum.

Figure 5: Average return vs. number of simulation steps on continuous control tasks.

Although GIRIL achieves performance that is close to the demonstration, the efficient planning-based reward function enables vPERL to perform even better. Other baselines are unable to reach the demonstration-level performance by learning from only one demonstration.

## 5 CONCLUSION

This paper presents a simple but efficient reward learning method, called variational planning-embedded reward learning (vPERL). By simultaneously training a planning-embedded action back-tracing module and a transition dynamics module, we can obtain meaningful reward function, useful planning computation and transition modeling. The efficient planning-based reward function is less biased and leads to policies that generalize well to new tasks, Therefore, our method avoid limiting its performance to the limited demonstration data. Empirical results show our method outperform state-of-the-art imitation learning methods on multiple Atari games by a large margin. An interesting topic for future investigation would be applying vPERL to hard exploration tasks with extremely sparse rewards.

[1] https://pybullet.org/

## REFERENCES


Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, pp. 1, 2004.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 182–189, 2011.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Daniel S Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, 2019a.

Daniel S Brown, Wonjoon Goo, and Scott Niekum. Ranking-based reward extrapolation without rankings. In *Conference on Robot Learning*, 2019b.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*, 2019.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pp. 49–58, 2016.

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.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. In *Advances in Neural Information Processing Systems*, pp. 8011–8023, 2018.

Animesh Karnewar. Pytorch implementations of variational discriminator bottleneck, 2018. URL `https://github.com/akanimax/Variational_Discriminator_Bottleneck`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms, 2018. URL `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`.

Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. In *International Conference on Learning Representations*, 2019.

Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, pp. 663–670, 2000.

Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C Smith, and Jelena Kovačević. Generalized value iteration networks: Life beyond lattices. In *AAAI Conference on Artificial Intelligence*, 2018.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pp. 2778–2787, 2017.

Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. In *International Conference on Learning Representations*, 2019.

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

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pp. 3483–3491, 2015.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Xingrui Yu, Yueming Lyu, and Ivor Tsang. Intrinsic reward driven imitation learning via generative model. In *International Conference on Machine Learning*, pp. 6672–6682, 2020.

Li Zhang, Xin Li, Sen Chen, Hongyu Zang, Jie Huang, and Mingzhong Wang. Universal value iteration networks: When spatially-invariant is not universal. In *AAAI Conference on Artificial Intelligence*, pp. 2154–2162, 2020.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.

Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling interaction via the principle of maximum causal entropy. In *International Conference on Machine Learning*, 2010.