Policy Continuation and Policy Evolution with Hindsight Inverse Dynamics

Hao Sun[1], Bo Dai[1], Zhizhong Li[1], Xiaotong Liu[2], Rui Xu[1], Dahua Lin[1], Bolei Zhou[1]
[1]The Chinese University of Hong Kong, [2]Peking University
sh018@ie.cuhk.edu.hk

Abstract

Solving goal-oriented tasks is an important but challenging problem in reinforcement learning (RL). For such tasks, the rewards are often sparse, making it difficult to learn a policy effectively. To tackle this difficulty, we propose a new approach called *Policy Continuation and and Policy Evolution with Hindsight Inverse Dynamics (PC&PEHID)*. This approach learns from Hindsight Inverse Dynamics based on Hindsight Experience Replay. This work also extends it to multi-step settings with Policy Continuation and Policy Evolution. The proposed method is general – it can work in isolation or be combined with other on-policy and off-policy algorithms. On challenging multi-goal tasks, PC&PEHID significantly improves the sample efficiency as well as the final performance.

1 Introduction

Imagine you are given the task of Tower of Hanoi with ten disks, what would you probably do to solve this complex problem? This game seems daunting at the first glance. However, through trials and errors, one may discover the key, that is, to recursively relocate the disks on the top of the stack from one pod to another, assisted by an intermediate one. In this case, you are actually learning skills from easier sub-tasks and those skills help you to learn more. This case exemplifies the procedure of self-imitated curriculum learning, namely recursively developing the skills of solving more complex problems.

Tower of Hanoi belongs to an important kind of challenging problems in Reinforcement Learning (RL), namely solving the goal-oriented tasks. In such tasks, rewards are usually very sparse. For example, in many goal-oriented tasks, a single binary reward is provided only when the task is completed [1, 2, 3]. Previous works attribute the difficulty in reward sparse problems to the low efficiency in experience collection [4], thus many approaches have been proposed to tackle this problem, including automatic goal generation [5], self-imitation learning [6] hierarchical reinforcement learning [7], curiosity driven methods [8, 9], curriculum learning [1, 10], and Hindsight Experience Replay (HER) [11]. Most of these works guide the agent by demonstrating on right choices based on sufficient exploration to improve learning efficiency. HER opens up a new way to learn more from failures but is limited, as it is only applicable when combined with off-policy algorithms[3].

In this paper we propose an approach of goal-oriented RL called Policy Continuation with Hindsight Inverse Dynamics (PCHID), which leverages the key idea of self-imitate learning. In contrast to HER, our method can work as an auxiliary module for both on-policy and off-policy algorithms, or as an isolated controller itself. Moreover, by learning to predict actions directly from back-propagation [12] through self-imitation, instead of temporal difference [13] or policy gradient [14, 15, 16, 17], the data efficiency is greatly improved.

The contributions of this work lie in three aspects: **(1)** We introduce the state-goal space partition for multi-goal RL and thereon define Policy Continuation (PC) as a new approach to such tasks.

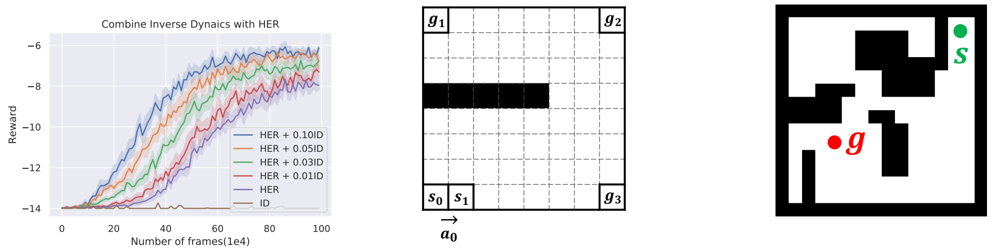

Figure 1: (a): Empirical results in bit-flipping problem. (b): An analogy of flat state space. (c): An example of the GridWorld domain, which is a non-flat case.

(2) We propose Hindsight Inverse Dynamics (HID), which extends the vanilla Inverse Dynamics method to the goal-oriented setting. **(3)** We further integrate PC and HID into PCHID, which can effectively leverage self-supervised learning to accelerate the process of reinforcement learning. Note that PCHID is a general method. Both on-policy and off-policy algorithms can benefit therefrom. We tested this method on challenging RL problems, where it achieves considerably higher sample efficiency.

2 Related Work

Hindsight Experience Replay Learning with sparse rewards in RL problems is always a leading challenge for the rewards are usually uneasy to reach with random explorations. Hindsight Experience Replay (HER) which relabels the failed rollouts as successful ones is proposed by Andrychowicz et al. [11] as a method to deal with such problem. The agent in HER receives a reward when reaching either the original goal or the relabeled goal in each episode by storing both original transition pairs s_t, g, a_t, r and relabeled transitions s_t, g', a_t, r' in the replay buffer.

Inverse Dynamics Given a state transition pair (s_t, a_t, s_{t+1}), the inverse dynamics [18] takes (s_t, s_{t+1}) as the input and outputs the corresponding action a_t. In previous works, inverse dynamics is always used to perform feature extraction [19, 9, 20] for policy network optimization. The actions stored in such transition pairs are always collected with a random policy so that it can barely be used to optimize the policy network directly. In our work, we use hindsight experience to revise the original transition pairs in inverse dynamics, and we call this approach Hindsight Inverse Dynamics. The details will be elucidated in the next section.

Auxiliary Task and Curiosity Driven Method Mirowski et al. [21] propose to jointly learn the goal-driven reinforcement learning problems with an unsupervised depth prediction task and a self-supervised loop closure classification task, achieving data efficiency and task performance improvement. But their method requires extra supervision like depth input.

Shelhamer et al. [20] introduce several self-supervised auxiliary tasks to perform feature extraction and adopt the learned features to reinforcement learning, improving the data efficiency and returns of end-to-end learning. Pathak et al. [19] propose to learn an intrinsic curiosity reward besides the normal extrinsic reward, formulated by prediction error of a visual feature space and improved the learning efficiency. Both of the approaches belong to self-supervision and utilize inverse dynamics during training. Although our method can be used as an auxiliary task and trained in self-supervised way, we improved the vanilla inverse dynamics with hindsight, which enables direct joint training of policy networks with temporal difference and self-supervised learning.

3 Method

3.1 Revisiting the Bit-Flipping Problem

Inspired by the self-imitated learning ability of humans, we aim to equip RL agents with a similar ability so that they could learn policies even when the original goal has not yet achieved. A straightforward way is to adopt the inverse dynamics. However, in most cases the actions stored in inverse dynamics are irrelevant to the goals. Specifically, transition pairs like $((s_t, g), (s_{t+1}, g), a_t)$

are saved to learn inverse dynamics of goal-oriented tasks, with the objective:

$$\phi = \arg\min_{\phi} \sum_{s_t, s_{t+1}, a_t} ||f_\phi((s_t, g), (s_{t+1}, g)) - a_t||^2, \tag{1}$$

where f_ϕ is a neural network parameterized by ϕ. While usually an agent is unaware of goals g in Eq.(1), we replace g with $g' = m(s_{t+1})$, which encodes the future state as a hindsight goal, so that an agent is knowing the hindsight goal when making decisions. *e.g.* a_t should be chosen if it wants to reach g' from a_t.

We incorporated the modified Eq.1 with HER in the bit-flipping problem [11], and obtained significant improment, as shown in Fig.1(a). We attribute this success to the flatness of the state space of the task. Fig.1(b) shows an intuitive analogy of such flatness, where it is not hard for the agent to extrapolate its policy to reach g_3 if it already knows how to reach s_1. Nevertheless, successes are not always within effortless reach. When challenging goals are encountered, such as reaching g_1 and g_2 in Fig.1(b) and navigation in the GridWorld (Fig.1(c), more sophisticated development of self-imitated learning is needed, which results in our proposed new approach, refered to as Policy Continuation with Hindsight Inverse Dynamics.

3.2 Perspective of Policy Continuation on Multi-Goal RL Task

Our approach is mainly based on policy continuation over sub-policies, which can be viewed as an emendation of the spontaneous extrapolation in the bit-flipping case. Given a policy π defined on the state space $\mathcal{S}_\pi$, if another policy Π defined on the state space $\mathcal{S}_\Pi$ satisfies 1) $\mathcal{S}_\pi \subset \mathcal{S}_\Pi$ and 2) $\Pi(s) = \pi(s)$ for any $s \in \mathcal{S}_\pi$, Π is called a *policy continuation* of π.

Upon this definition, we then introduce the concept of *k-step solvability*, where a state-goal pair (s, g) in a system with deterministic dynamics is called k-step solvable if reaching the goal g needs at least k steps under the optimal policy $\pi^\star$ starting from s. Here we follow HER to assume a mapping [11] $m : \mathcal{S} \to \mathcal{G}$ so that $r(s, m(s)) = 1$ for all s, which means the information of g is encoded in s. In the simplest case, m is an identical mapping.Consequently, we could divide the state-goal space according to their k-step solvability, as $\mathcal{S} \times \mathcal{G} = (\mathcal{S} \times \mathcal{G})_0 \cup (\mathcal{S} \times \mathcal{G})_1 \cup ... \cup (\mathcal{S} \times \mathcal{G})_T \cup (\mathcal{S} \times \mathcal{G})_U$, where T is a finite horizon we suppose the task should be solved within, and $(\mathcal{S} \times \mathcal{G})_U$ denotes unsolvable state-goal pairs, which is not our focus.

Following the partition over the state-goal spate, a curriculum way to obtain the optimal policy $\pi^\star$ over $(\mathcal{S} \times \mathcal{G}) \backslash (\mathcal{S} \times \mathcal{G})_U$ is readily at hand: starting from an optimal policy $\pi_0^\star$ on $(\mathcal{S} \times \mathcal{G})_0$, we recursively approximate $\pi^\star$ on $\bigcup_{j=0}^{i} (\mathcal{S} \times \mathcal{G})_j$ by expanding the domain of sub-state-goal space in policy continuation. In practice, we parameterize the policy function $\pi = f_\theta$ using neural networks. And optimization for $\pi = f_\theta$ is achieved by jointly self-supervised learning with data collected by Hindsight Inverse (HID).

3.3 Hindsight Inverse Dynamics

One step HID data can be collected easily. With n randomly rollout trajectories $\{(s_0, g), a_0, r_0, (s_1, g), a_1, ..., (s_T, g), a_T, r_T\}_i, i \in \{1, 2, ..., n\}$, we can use a modified inverse dynamics by substituting the original goal g with hindsight goal $g' = m(s_{t+1})$ for every s_t and result in $\{(s_0, m(s_1)), a_0, (s_1, m(s_2)), a_1, ..., (s_{T-1}, m(s_T)), a_{T-1}\}_i, i \in \{1, 2, ..., n\}$. We can then fit f_{θ_1} using SGD [22] by

$$\theta_1 = \arg\min_{\theta} \sum_{s_t, s_{t+1}, a_t} ||f_\theta((s_t, m(s_{t+1})), (s_{t+1}, m(s_{t+1}))) - a_t||^2. \tag{2}$$

When m is an identical mapping, the function f_{θ_1} is a good enough approximator for π_1^*. Otherwise, we should adapt equation (2) as $\theta_1 = \arg\min_\theta \sum_{s_t, s_{t+1}, a_t} ||f_\theta((s_t, m(s_{t+1})), m(s_{t+1})) - a_t||^2$, where the state information is omitted to leverage f_{θ_1} as a policy.

Once we have $f_{\theta_{k-1}}$, an approximator of π_{k-1}^*, k-step HID is ready to get. We can collect valid k-step HID data recursively by testing whether the k-step HID state-goal pairs indeed need k steps to solve. Fig.2 illustrates this process. The testing process is based on a function $\text{TEST}(\cdot)$.[1] A joint

[1]See Appendix A for the choice of TEST.

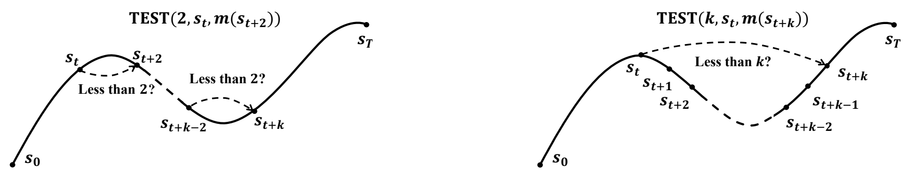

Figure 2: Test whether the transitions are 2-step (left) or k-step (right) solvable. The TEST function will return True if the transition $s_t \to s_{t+k}$ needs at least k steps.

training is utilized to ensure f_{θ_k} to be a policy continuation of $\pi_i^*, i \in \{1, ..., k\}$:

$$\theta_k = \arg\min_{\theta} \sum_{s_t^{(i)}, s_{t+i}^{(i)}, a_t^{(i)}, i \in \{1,...,k\}} ||f_\theta((s_t, m(s_{t+i})), (s_{t+i}, m(s_{t+i}))) - a_t||^2 \tag{3}$$

The combination of PC and with multi-step HID leads to our algorithm PCHID.[2] PCHID can work alone or as an auxiliary module with other RL algorithms.

3.4 Synchronous Improvement

In PCHID, the learning scheme is set to be curriculum, i.e., the agent must learn to master easy skills before learning complex ones. However, in general the efficiency of finding a transition sequence that is i-step solvable decreases as i increases. The size of buffer $\mathbb{B}_i$ is thus decreasing for $i = 1, 2, 3, ..., T$ and the learning of π_i might be restricted due to limited experiences. Besides, in continuous control tasks, the k-step solvability means the number of steps it should take from s to g, given the maximum permitted action value. And in practice the k-step solvability is an evolving concept that can gradually change as the learning goes. Specifically, at the beginning, an agent can only walk with small paces as it is learned from experiences collected by random movements. As the training continues, the agent is confident to move with larger paces, which may change the distribution of selected actions. Consequently, previous k-step solvable state goal pairs may be solved in less than k steps.

From the Ornstein-Uhlenbeck Process Perspective. At first, we study these issues in the perspective of the Ornstein-Uhlenbeck Process. For simplicity we consider 1-dimensional state-action space. An policy equipped with Gaussian noise in the action space $a \sim \mathcal{N}(\mu, \sigma^2)$ lead to a stochastic process in the state space. In the most simple case, the mapping between action space and the corresponding change in state space is an affine transformation, i.e., $\Delta s_t = s_{t+1} - s_t = \alpha a_t + \beta$. Without loss of generality, we have

$$\Delta s_t \sim \mathcal{N}(\epsilon(g - s_t), \sigma^2) \tag{4}$$

after normalization. The ϵ describes the correlations between actions and goal states. e.g., for random initialized policies, the actions are unaware of goal thus $\epsilon = 0$, and for optimal policies, the actions are goal-oriented thus $\epsilon = 1$. The learning process can be interpreted as maximizing ϵ, where better policies have larger ϵ. Under those notations,

$$\Delta s_t = \epsilon(g - s_t)\Delta t + \sigma \Delta W_t \tag{5}$$

where W_t is the Wiener Process. As Eq. (5) is exactly an Ornstein-Uhlenbeck (OU) Process, it has closed-form solutions:

$$s_t = s_0 e^{-\epsilon t} + g(1 - e^{-\epsilon t}) + \sigma \int_0^t e^{-\epsilon(t-s)} dW_s \tag{6}$$

and the expectation is

$$\mathbb{E}(s_t) - g = (s_0 - g)e^{-\epsilon t} \tag{7}$$

Intuitively, Eq. (7) shows that as ϵ increase during learning, it will take less time to reach the goal. More precisely, we are caring about the concept of First Hitting Time (FHT) of OU process, i.e., $\tau = \inf\{t > 0 : s_t = g | s_0\}$ [23]. Accordingly, the optimization of solving goal-oriented reward sparse tasks can be viewed as minimizing the FHT of OU process. From this perspective, any action that can reduce the FHT will lead to a better policy.

[2]See Appendix B for more details.

Inspired by such a perspective, and to tackle the efficiency bottleneck and further improve the performance of PCHID, we extend our method to a synchronous setting based on the evolving concept of k-step solvability. We refer to this updated approach as Policy Evolution with Hindsight Inverse Dynamics (PEHID). PEHID start the learning of π_i before the convergence of π_{i+1} by merging buffers $\{\mathbb{B}_i\}_{i=1}^{T}$ into one single buffer $\mathbb{B}$. And when increasing the buffer with new experiences, we will test an experience that is k-step solvable could be reproduced within k steps if we change the goal. We retain those experiences that are not reproducable as they contain new valuable skills for current policy to learn. Although such a synchronous learning scheme is not guaranteed to converge theoretically, in practice we empirically found it evolves to reach goals in less and less steps.

4 Experiments

As a policy $\pi(s, g)$ aims at reaching a state s' where $m(s') = g$, by intuition the difficulty of solving such a goal-oriented task depends on the complexity of m. In Sec.4.1 we start with a simple case where m is an identical mapping in the environment of GridWorld by showing the agent a fully observable map. Moreover, the GridWorld environment permits us to use prior knowledge to calculate the accuracy of any TEST function. We show that PCHID can work independently or augmented with the DQN in discrete action space setting, outperforming the DQN as well as the DQN augmented with HER. The GridWorld environment corresponds to the identical mapping case $\mathcal{G} = \mathcal{S}$. In Sec.4.2 we test our method on a continuous control problem, the OpenAI Fetch environment provided by Plappert et al. [3]. In FetchReach, PCHID outperforms PPO by achieving 100% successful rate in about 100 episodes. We further compare the sensitivity of PPO to reward values and the robustness PCHID owns. The state-goal mapping of FetchReach environment is $\mathcal{G} \subset \mathcal{S}$. We demonstrate PEHID in the other three environments and show its high learning efficiency.

4.1 GridWorld Navigation

We use the GridWorld navigation task in Value Iteration Networks (VIN) [24], in which the state information includes the position of the agent, and an image of the map of obstacles and goal position. In our experiments we use 16×16 domains, navigation in which is not an effortless task. Fig.1(c) shows an example of our domains. The action space is discrete and contains 8 actions leading the agent to its 8 neighbour positions respectively. A reward of 10 will be provided if the agent reaches the goal within 50 timesteps, otherwise the agent will receive a reward of -0.02. An action leading the agent to an obstacle will not be executed, thus the agent will stay where it is. In each episode, a new map will randomly selected start s and goal g points will be generated. We train our agent for 500 episodes in total so that the agent needs to learn to navigate within just 500 trials, which is much less than the number used in VIN [24].[3] Thus we can demonstrate the high data efficiency of PCHID by testing the learned agent on 1000 unseen maps. Our work follows VIN to use the rollout success rate as the evaluation metric.

Our empirical results are shown in Fig.3. Our method is compared with DQN, both of which are equipped with VIN as policy networks. We also apply HER to DQN but result in a little improvement. PC with 1-step HID, denoted by PCHID 1, achieves similar accuracy as DQN in much less episodes, and combining PC with 5-step HID, denoted by PCHID 5, and HER results in much more distinctive improvement.

4.2 OpenAI Fetch

In the Fetch environments, there are several tasks based on a 7-DoF Fetch robotics arm with a two-fingered parallel gripper. There are four tasks: FetchReach, FetchPush, FetchSlide and FetchPickAndPlace. In those tasks, the states include the Cartesian positions, linear velocity of the gripper, and position information as well as velocity information of an object if presented. The goal is presented as a 3-dimentional vector describing the target location of the object to be moved to. The agent will get a reward of 0 if the object is at the target location within a tolerance or -1 otherwise. Action is a continuous 4-dimentional vector with the first three of them controlling movement of the gripper and the last one controlling opening and closing of the gripper.

[3]Tarmar et al. train VIN through the imitation learning (IL) with ground-truth shortest paths between start and goal positions. Although both of our approaches are based on IL, we do not need ground-truth data

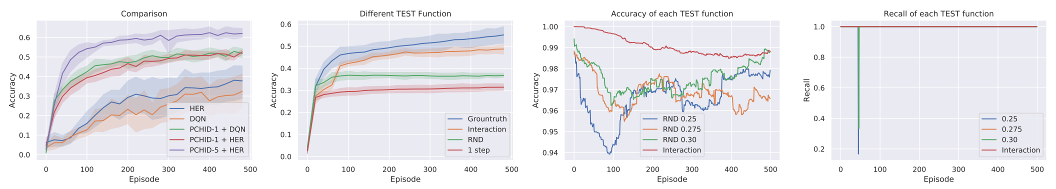

Figure 3: (a): The rollout success rate on test maps in 10 experiments with different random seeds. HER outperforms VIN, but the difference disappears when combined with PCHID. PCHID-1 and PCHID-5 represent 1-step and 5-step PCHID. (b): Performance of PCHID module alone with different TEST functions. The blue line is from ground truth testing results, the orange line and green line are Interaction and RND respectively, and the red line is the 1-step result as a baseline. (c)(d): Test accuracy and recall with Interaction and RND method under different threshold.

Table 1: The successful rate of different methods in the FetchPush, FetchSlide and FetchPickAndPlace environments (trained for 1.25M timesteps)

Method	FetchPush	FetchSlide	FetchPickAndPlace
PPO	0.00	0.00	0.00
DDPG	0.08	0.03	0.05
DDPG + HER	**1.00**	0.30	0.60
PEHID	0.95	**0.38**	**0.75**

FetchReach Here we demonstrate PCHID in the FetchReach task. We compare PCHID with PPO and HER based on PPO. Our work is the first to extend hindsight knowledge into on-policy algorithms [3]. Fig.4 shows our results. PCHID greatly improves the learning efficiency of PPO. Although HER is not designed for on-policy algorithms, our combination of PCHID and PPO-based HER results in the best performance.

FetchPush, FetchSlide, FetchPickAndPlace We evaluate PEHID in the FetchPush, FetchSlide and FetchPickAndPlace tasks. To demonstrate the high learning efficiency of PEHID, we compare the success rate of different method after 1.25M timesteps, which is amount to 13 epochs in the work of Plappert et. al [3]. Table 1 shows our results.

4.3 Combing PCHID with Other RL Algorithms

As PCHID only requires sufficient exploration in the environment to approximate optimal sub-policies progressively, it can be easily plugged into other RL algorithms, including both on-policy algorithms and off-policy algorithms. At this point, the PCHID module can be regarded as an extension of HER for off-policy algorithms.

We put forward three combination strategies and evaluate each of them on both GridWorld and FetchReach environment. 1.**Joint Training**: The first strategy for combining PCHID with normal RL algorithm is to adopt a shared policy between them. A shared network is trained through both

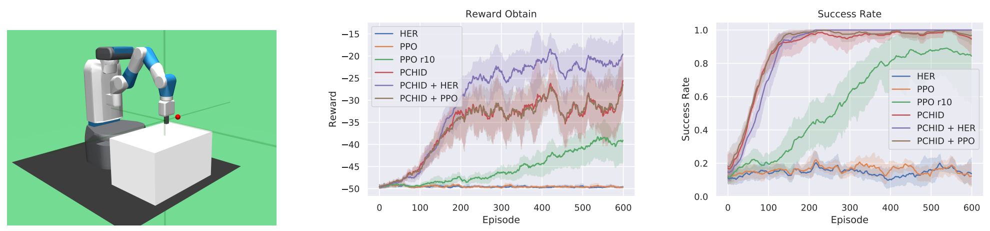

Figure 4: (a): The FetchReach environment. (b): The reward obtaining process of each method. In PPO r10 the reward of achieving the goal becomes 10 instead of 0 as default, and the reward is re-scaled to be comparable with other approaches. This is to show the sensitivity of PPO to reward value. By contrast, the performance of PCHID is unrelated to reward value. (c): The success rate of each method. Combining PPO with PCHID brings about little improvement over PCHID, but combining HER with PCHID improves the performance significantly.

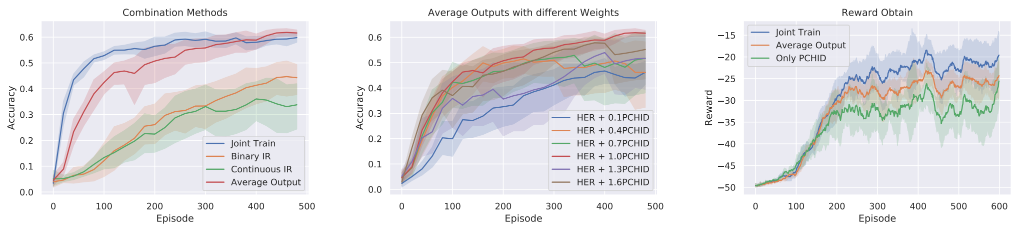

Figure 5: (a): Accuracy of GridWorld under different combination strategies. (b): Averaging outputs with different weights. (c): Obtained Reward of FetchReach under different strategies.

temporal difference learning in RL and self-supervised learning in PCHID. The PCHID module in joint training can be viewed as a regularizer. 2.**Averaging Outputs**: Another strategy for combination is to train two policy networks separately, with data collected in the same set of trajectories. When the action space is discrete, we can simply average the two output vectors of policy networks. When the action space is continuous, we can then average the two predicted action vectors and perform an interpolated action. 3.**Intrinsic Reward (IR)**: This approach is quite similar to the curiosity driven methods. Instead of using the inverse dynamics to define the curiosity, we use the prediction difference between PCHID module and RL agent as an intrinsic reward to motivate RL agent to act as PCHID. Maximizing the intrinsic reward helps the RL agent to avoid aimless explorations hence can speed up the learning process.

Fig.5 shows our results in GridWorld and FetchReach with different combination strategies. Joint training performs the best and it does not need hyper-parameter tuning. On the contrary, averaging outputs requires determining the weights and intrinsic reward requires adjusting its scale with regard to the external reward.

5 Conclusion

In this work we propose the Policy Continuation with Hindsight Inverse Dynamics (PCHID) to solve the goal-oriented reward sparse tasks from a new perspective. Our experiments show the PCHID is able to improve data efficiency remarkably in both discrete and continuous control tasks. Moreover, our method can be incorporated with both on-policy and off-policy RL algorithms flexibly. We then extend PCHID in synchronous settings and result in PEHID, in which the learning efficiency is further improved.

References

[1] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[2] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[3] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[4] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[5] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

[6] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *arXiv preprint arXiv:1806.05635*, 2018.

[7] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. 2018.

[8] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *IEEE Conference on Computer Vision & Pattern Recognition Workshops*, 2017.

[9] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[10] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.

[11] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5048–5058. Curran Associates, Inc., 2017.

[12] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

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

[14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[15] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[16] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

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

[18] Michael I Jordan and David E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354, 1992.

[19] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[20] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

[21] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, and Raia Hadsell. Learning to navigate in complex environments. 2016.

[22] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[23] Larbi Alili, Pierre Patie, and Jesper Lund Pedersen. Representations of the first hitting time density of an ornstein-uhlenbeck process. *Stochastic Models*, 21(4):967–980, 2005.

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

[25] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

A On the Selection of $\mathrm{TEST}(\cdot)$ Function

In Algorithm 1, a crucial step to extend the $(k-1)$-step sub policy to k-step sub policy is to test whether a k-step transition $s_t \to s_{t+k}$ in a trajectory is indeed a k-step solvable problem if we regard s_t as a start state s_0 and $m(s_{t+k})$ as a goal g. We propose two approaches and evaluate both in Sec.4.

Interaction A straightforward idea is to reset the environment to s_t and execute action a_t by policy π_{k-1}, followed by execution of $a_{t+1}, a_{t+2}, ...$, and record if it achieves the goal in less than k steps. We call this approach *Interaction* for it requires the environment to be resettable and interact with the environment. This approach can be portable when the transition dynamics is known or can be approximated without heavy computation expense.

Random Network Distillation (RND) Given a state as input, the RND [25] is proposed to provide exploration bonus by comparing the output difference between a fixed randomly initialized neural network N_A and another neural network N_B, which is trained to minimize the output difference between N_A and N_B with previous states. After training N_B with $1, 2, ..., k-1$ step transition pairs to minimize the output difference between N_A and N_B, since N_B has never seen k-step solvable transition pairs, these pairs will be differentiated for they lead to larger output differences.

B Algorithm1

Algorithm 1 PCHID Module

```
Require
  • a policy $\pi_b(s, g)$
  • a reward function $r(s, g) = 1$ if $g = m(s)$ else $0$
  • a buffer for PCHID $\mathbb{B} = \{\mathbb{B}_1, \mathbb{B}_2, ..., \mathbb{B}_{T-1}\}$
  • a list $\mathbb{K}$
Initialize $\pi_b(s, g)$, $\mathbb{B}$, $\mathbb{K} = [1]$
for episode $= 1, M$ do
  generate $s_0$, $g$ by the system
  for $t = 0, T-1$ do
    Select an action by the behavior policy $a_t = \pi_b(s_t, g)$
    Execute the action $a_t$ and get the next state $s_{t+1}$
    Store the transition $((s_t, g), a_t, (s_{t+1}, g))$ in a temporary episode buffer
  end for
  for $t = 0, T-1$ do
    for $k \in \mathbb{K}$ do
      calculate additional goal according to $s_{t+k}$ by $g' = m(s_{t+k})$
      if TEST($k, s_t, g'$) = True then
        Store $(s_t, g', a_t)$ in $\mathbb{B}_k$
      end if
    end for
  end for
  Sample a minibatch $B$ from buffer $\mathbb{B}$
  Optimize behavior policy $\pi_b(s_t, g')$ to predict $a_t$ by supervised learning
  if Converge then
    Add $\max(\mathbb{K}) + 1$ in $\mathbb{K}$
  end if
end for
```

C Algorithm2

Algorithm 2 PEHID Module

```
Require
  • a policy $\pi_b(s, g)$
  • a reward function $r(s, g) = 1$ if $g = m(s)$ else $0$
  • a buffer for PEHID $\mathbb{B}$
  • a list $\mathbb{K} = [1, 2, ..., K]$
Initialize $\pi_b(s, g)$, $\mathbb{B}$
for episode $= 1, M$ do
  generate $s_0$, $g$ by the system
  for $t = 0, T - 1$ do
    Select an action by the behavior policy $a_t = \pi_b(s_t, g)$
    Execute the action $a_t$ and get the next state $s_{t+1}$
    Store the transition $((s_t, g), a_t, (s_{t+1}, g))$ in a temporary episode buffer
  end for
  for $t = 0, T - 1$ do
    for $k \in \mathbb{K}$ do
      calculate additional goal according to $s_{t+k}$ by $g' = m(s_{t+k})$
      if TEST($k, s_t, g'$) = True then
        Store $(s_t, g', a_t)$ in $\mathbb{B}$
      end if
    end for
  end for
  Sample a minibatch $B$ from buffer $\mathbb{B}$
  Optimize behavior policy $\pi_b(s_t, g')$ to predict $a_t$ by supervised learning
end for
```