Learning Goal-Conditioned Policies Offline with Self-Supervised Reward Shaping

Anonymous Author(s)
Affiliation
Address
email

Abstract: Developing agents that can execute multiple skills by learning from pre-collected datasets is an important problem in robotics, where online interaction with the environment is extremely time-consuming. Moreover, manually designing reward functions for every single desired skill is prohibitive. Prior works [1] targeted these challenges by learning goal-conditioned policies from offline datasets without manually specified rewards, through hindsight relabeling [2]. These methods suffer from the issue of sparsity of rewards, and fail at long-horizon tasks. In this work, we propose a novel self-supervised learning phase on the pre-collected dataset to understand the structure and the dynamics of the model, and shape a dense reward function for learning policies offline. We evaluate our method on two continuous control tasks, and show that our model is significantly better than existing related approaches [1, 2], especially on tasks that involve long-term planning.

Keywords: Offline Reinforcement Learning, Self-Supervised Learning, Goal-Conditioned RL

1 Introduction

While the goal of realizing general autonomous agents requires mastery of a large and diverse set of skills, achieving this by focusing on each skill individually with standard reinforcement learning (RL) frameworks is prohibitive. This is primarily due to the need for manually designed reward functions and environment interactions for each skill. Unsupervised RL has opened a way for learning agents that can execute diverse behaviours without supervision (*i.e.*, hand-crafted rewards), and then be further adapted to downstream tasks through few-shot or zero-shot generalization. However, learning policies with such methods is impractical with real robots as they require millions of interactions when trained online.

Recently, a line of study has emerged that uses pre-collected dataset of environment interactions and trains policies offline (*i.e.*, without additional interactions with the environment). More precisely, given a dataset of reward-free trajectories and a reward function designed to solve a specific task, the agent learns offline by relabeling the transitions in the dataset with the reward function. This setting is particularly relevant in robotics, where data collection is extremely time-consuming: disentangling data collection and policy learning in this context allows for faster policy iteration. However, it would require to design one specific reward function and to learn one policy per task that we want the agent to solve.

An important question to scale robot learning is therefore to find ways of learning multi-task policies from already collected datasets. Recent works [1, 3, 4], have targeted this problem from a goal-conditioned perspective: given a dataset of previously collected trajectories, the objective is to learn a goal-oriented agent that can reach any state in the dataset. The advantages of this formulation are two-fold: first, it makes it easy to interpret skills, and second it does not require any adaptation at

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

test time. Making this framework unsupervised requires to break free from hand-crafted rewards, as proposed by Chebotar et al. [1], where they learn goal-conditioned policies offline through hindsight relabeling [2]. However, their approach is subject to the pitfall of learning from sparse rewards, and can be inefficient in long-horizon tasks.

In this work, we present a self-supervised reward shaping method that enables building an offline dataset with dense rewards. To this end, we develop self-supervised learning phase with the pre-collected dataset that aims at learning the structure and dynamics of the environment before training the policy. During this phase, we: (i) train a Reachability Network [5] to estimate the local distance in the state space $\mathcal{S}$, then (ii) extract a set of representative states that covers $\mathcal{S}$, and finally (iii) build a graph on this set to approximate global distance in $\mathcal{S}$. We then use the graph to train goal-conditioned policy offline in two ways: to compute rewards through shortest path distance, and to create transitions of intermediate difficulty on the path to the goal.

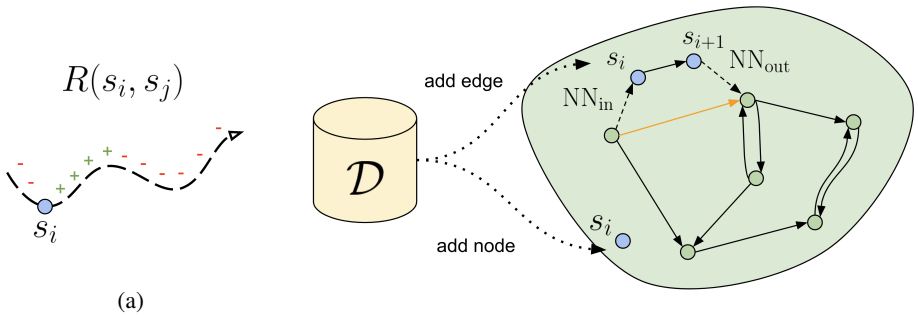
We evaluate our method on complex continuous control tasks, and compare it to previous state-of-the-art offline [1, 2] approaches. We show that our graph-based reward method learns good goal-conditioned policies by leveraging transitions from a dataset of past experience with neither any additional interactions with the environment nor manually designed rewards. Moreover, we show that, contrary to prior work [1] that uses datasets collected with a policy trained with supervised rewards, our method allows for learning goal-conditioned policies even from datasets of poor quality, *e.g.* containing trajectories sampled with a random policy. Our work is thus the first to enable learning goal-conditioned policies from offline datasets without any supervision, as it does not require any hand-crafted reward function at any stage: data collection, policy training and evaluation.

2 Related Work

Goal-conditioned RL In its original formulation, goal-conditioned reinforcement learning was tackled by several methods [6, 7, 2, 8]. The policy learning process is supervised in these works: the set of evaluation goals is available at train time as well as a shaped reward function that guides the agent to the goal. Several works propose solutions for generating goals automatically when training goal-conditioned policies, including self-play [9, 10, 11], and adversarial student-teacher policies [12]. In a recent line of research, some works [13, 14, 15, 16, 17, 18, 19, 20] focused on learning goal-conditioned policies in an unsupervised fashion. The objective is to train general agents that can reach any goal state in the environment without any supervision (reward, goal-reaching function) at train time. In particular, Mendonca et al. [19] trains a model-based agent that learns to discover novel goals with an explorer model, and reach them with an achiever policy via imagined rollouts.

Offline RL The data collection technique is an important aspect when studying the training of policies from pre-collected datasets. In this context, the first works assumed access to policies trained with task-specific rewards [21, 22]. More recently, methods proposed to leverage unsupervised exploration to collect datasets for offline RL [23, 24]. In particular, Yarats et al. [23] creates a dataset of pre-collected trajectories, ExoRL, on the DeepMind Control Suite [25] generated without any hand-crafted rewards. Similarly to URLB [26], ExoRL benchmarks a number of exploration algorithms Pathak et al. [27], Eysenbach et al. [28], Pathak et al. [29], Yarats et al. [30], and evaluates the performance of a policy trained on the corresponding offline datasets relabled with task-specific rewards.

Multi-task Offline RL Recent works proposed to learn multiple tasks from pre-collected datasets, starting with methods [31] the generates goals to improve the offline data collection process in a self-supervised way. This connection has also been studied in the supervised setting [3, 32] and to learn hierarchical policies [4]. In a setting closely relateed to our work, Actionable Models [1] considers the problem of learning goal-conditioned policies from offline datasets without interacting with the environment, and with no task-specific rewards. They employ goal-conditioned Q-learning with hindsight relabeling. As opposed to this work that relies on learning from sparse rewards, we propose to leverage a self-supervised training stage to densely shape rewards.



(a) (b)

Figure 1: Visualization of our dense reward shaping method. (a) shows how training labels are generated for training the RNet: given a state s_i, positive pairs are sampled in the same trajectory within a threshold τ_{reach}, and the rest of the trajectory forms negative pairs. (b) presents an overview of the graph building algorithm. Given a transition $(s_i, s_{i+1}) \in \mathcal{D}$, we add s_i as node if it is distant enough from existing nodes in the graph. Moreover, we add an edge in the graph between the corresponding nearest neighbors of s_i and s_{i+1}.

3 Preliminaries

Let $\mathcal{E} = (\mathcal{S}, \mathcal{A}, P, p_0, \gamma, T)$ define a reward-free Markov decision process (MDP), where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a state-transition probability function, $p_0 : \mathcal{S} \to \mathbb{R}_+$ is an initial state distribution, γ is a discount factor, and T is the task horizon. In the goal-conditioned setting, the objective is to learn a goal-conditioned policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ that maximizes the expectation of the cumulative return over the goal distribution, where $\mathcal{G}$ denotes the goal space. Here, we make the common assumption that states and goals are defined in the same form, *i.e.*, $\mathcal{G} \subset \mathcal{S}$.

We assume that we have access to a dataset $\mathcal{D}$ of pre-collected episodes generated by using any data collection algorithm in $\mathcal{E}$. Each episode is stored in $\mathcal{D}$ as a series of (s, a, s') tuples. In the general offline formulation introduced by Yarats et al. [23], the dataset $\mathcal{D}$ can be relabeled by evaluating any reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ at each tuple in $\mathcal{D}$, and adding the resulting tuple $(s, a, r(s, a), s')$ in the relabeled dataset $\mathcal{D}_r$. We can extend this protocol to the goal-oriented setting by considering a goal distribution $p_{\mathcal{G}}$ in the goal space, and any goal-conditioned reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. Given a tuple (s, a, s') in $\mathcal{D}$, we relabel it by sampling a goal $g \sim p_{\mathcal{G}}$, computing $r(s, a, g)$ and adding the resulting tuple $(s, a, g, r(s, a, g), s')$ in the relabeled dataset $\mathcal{D}_{r,p_{\mathcal{G}}}$. In the unsupervised framework, we assume that the evaluation goal space is unknown, and that we do not have access to any pre-defined reward function, nor any distance function in the state space.

Once the relabeled dataset $\mathcal{D}_{r,p_{\mathcal{G}}}$ is generated, we can learn a goal-conditioned policy by executing any offline RL algorithm. The algorithm runs completely offline, by sampling tuples from $\mathcal{D}_{r,p_{\mathcal{G}}}$ and without any interaction with the environment. The goal-conditioned policy is then evaluated online in $\mathcal{E}$ on a set of fixed evaluation goals that is not known during training.

4 Self-supervised Reward Shaping

We now describe our self-supervised reward shaping method for learning a goal-conditioned agent offline on a pre-collected dataset $\mathcal{D}$. Our method comprises three stages that we will describe below. In the first stage, we train a Reachability Network (RNet) [5] on the trajectories in $\mathcal{D}$ to predict whether two states are reachable from one another. The second stage consists in building a directed graph $\mathcal{M}$ whose nodes are a subset of states in $\mathcal{D}$, and edges connect reachable states. We employ the RNet as a criterion to avoid adding similar states to $\mathcal{M}$ so that its nodes cover the states in $\mathcal{D}$ uniformly. Subsequently, we determine the reach of all the nodes based on the transitions in $\mathcal{D}$ and the RNet, and connect them with directed edges. The final stage is training the goal-conditioned

policy on transitions and goals sampled from $\mathcal{D}$. It is trained with dense rewards computed as the sum of a global (based on the graph distance in $\mathcal{M}$) and local (based on the RNet) distance terms. The important aspect of our method is that the whole training only uses trajectories from the pre-collected dataset $\mathcal{D}$ without running a single action in the environment. We now describe each component in more detail.

4.1 Reachability network

In order to learn a good local distance between states in $\mathcal{D}$, we adopt an asymmetric version of the Reachability Network (RNet) [5]. The general idea of RNet is to approximate the distance between states in the environment by the average number of steps it takes for a random policy to go from one state to another. We adapted the the original formulation with two modifications: first, we use exploration trajectories from $\mathcal{D}$ instead of random trajectories and second, we leverage the temporal direction because a state can be reachable from another without the converse being true. Let $(s_1^a, ..., s_T^a)$ denote a trajectory in $\mathcal{D}$, where a is a trajectory index. We define a *reachability label* y_{ij}^{ab} for each pair of observations (s_i^a, s_j^b) by

$$y_{ij}^{ab} = \begin{cases} 1 & \text{if } a = b \text{ and } 0 \leq j - i \leq \tau_{\text{reach}}, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } 1 \leq i, j \leq T, \tag{1}$$

where the *reachability threshold* τ_{reach} is a hyperparameter. The reachability label is equal to 1 *iff* the states are in the same trajectory and the number of steps from s_i^a to s_j^b is below τ_{reach}. Note that $y_{ij}^{ab} \neq y_{ji}^{ab}$. We train a siamese neural network R, the RNet, to predict the reachability label y_{ij}^{ab} from a pair of observations (s_i^a, s_j^b) in $\mathcal{D}$. The RNet consists of an embedding network g, and a fully-connected network f to compare the embeddings, *i.e.*,

$$R(s_i^a, s_j^b) = \sigma\left[f(g(s_i^a), g(s_j^b))\right], \tag{2}$$

where σ is a sigmoid function. A higher R value indicates two states reachable easily with random walk, so they can be considered close in the environment. More precisely, R takes values in $(0, 1)$ and s' is reachable from s if $R(s, s') \geq 0.5$. RNet is trained in a self-supervised fashion, as the ground-truth labels needed to train the network are generated automatically.

4.2 Directed graph

In the next phase, we use trajectories in $\mathcal{D}$ to build a directed graph $\mathcal{M}$ to capture high-level dynamics of the environment. We want the nodes of $\mathcal{M}$ to evenly represent the states in $\mathcal{D}$. This is achieved by filtering the states in $\mathcal{D}$: a state is added to $\mathcal{M}$ only if it is distant enough from all the other nodes in $\mathcal{M}$. More precisely, a state $s \in \mathcal{D}$ is added to $\mathcal{M}$ if and only if

$$R(s, n) < 0.5 \text{ and } R(n, s) < 0.5, \quad \text{for all } n \in \mathcal{M}. \tag{3}$$

Note that we require both the directions to be novel. This filtering avoids redundancy by preventing similar states to be added to the memory. It also has a balancing effect because it limits the number of states that can be added from a certain area even if it is visited by the agent many times in $\mathcal{D}$.

Once the nodes are selected, we connect pairs that are reachable from one to another. To this end, we employ trajectories in $\mathcal{D}$ because they contain actual feasible transitions. Given a transition $s_i \to s_j$ in $\mathcal{D}$, we add edge $n_i \to n_j$ if s_i can be reached from node n_i and node n_j can be reached from s_j. This way, we have a chain $n_i \to s_i \to s_j \to n_j$ and can assume n_j is reachable from n_i. Concretely, we select node n_i to be the incoming nearest neighbor (NN_{in}) to s_i, and n_j to be the outgoing nearest neighbor (NN_{out}) from s_j, *i.e.*,

$$n_i = \text{NN}_{\text{in}}(s_i) = \operatorname*{argmax}_{n \in \mathcal{M}} R(n, s_i), \quad n_j = \text{NN}_{\text{out}}(s_j) = \operatorname*{argmax}_{n \in \mathcal{M}} R(s_j, n). \tag{4}$$

By performing this action over all the transitions in $\mathcal{D}$, we turn $\mathcal{M}$ into a directed graph where edges represent reachability from one node to another.

4.3 Distance function for policy training

RNet predicts the reachability between s_i and s_j so we can directly use it as a distance metric

$$d_l(s_i, s_j) = 1 - R(s_i, s_j), \quad \forall s_i, s_j \in \mathcal{S}. \tag{5}$$

However, this reachability metric is confined within a certain threshold, so there is no guarantee that the RNet predictions will have good global properties.

In contrast, the directed graph $\mathcal{M}$ captures high-level global dynamics of the environment. We can easily derive a distance function $d_{\mathcal{M}}(n_i, n_j)$ between any pair of nodes in $\mathcal{M}$ by computing the length of the shortest path in this graph, provided the graph is connected. In practice, we can use a trick to connect the graph if necessary, by adding an edge between the pair of nodes from different connected components with maximum RNet value. Moreover, we can extend this distance $d_{\mathcal{M}}$ to a global distance function d_g in the state space $\mathcal{S}$ by finding, for any pair s_i and s_j in $\mathcal{S}$ their corresponding nearest neighbors in the proper direction. More precisely,

$$d_g(s_i, s_j) = d_{\mathcal{M}}(\text{NN}_{\text{out}}(s_i), \text{NN}_{\text{in}}(s_j)), \quad \forall s_i, s_j \in \mathcal{S}. \tag{6}$$

The distance d_g between two states in the state space becomes the length of the shortest path between their respective closest nodes in the graph. This process propagates the good local properties of RNet to get a well-shaped distance function for states that are further away. Since d_g captures global distances while d_l captures local fine-grained distance, we use their combination as a final distance function: $\forall s_i, s_j \in \mathcal{S}, \quad d(s_i, s_j) = d_g(s_i, s_j) + d_l(s_i, s_j)$.

4.4 Policy training

The last phase of our method is training the goal-condition policy offline. Here, we create an offline replay buffer $\mathcal{B}$ that is filled with relabeled data. We randomly sample a transition (s_t, a_t, s_{t+1}) from $\mathcal{D}$ as well as a goal state g and relabel the transition with reward $r_t = -d(s_{t+1}, g)$. We then push the relabeled transition $(s_t, a_t, g, r_t, s_{t+1})$ to $\mathcal{B}$. In order to create a curriculum that artificially guides the agent towards the goal, we experimented with two different transition augmentation techniques described below.

Sub-goal augmentation. Let $(s_t, a_t, g, r_t, s_{t+1})$ denote a relabeled transition and $(n_0, ..., n_{P-1})$ the shortest path in the graph $\mathcal{M}$ between $n_0 = \text{NN}_{\text{out}}(s_t)$ and $n_{P-1} = \text{NN}_{\text{in}}(g)$. The augmentation technique consists in adding to the replay buffer every transition $(s_t, a_t, n_i, r_t^i, s_{t+1})$ for all $i \in \{0, P-1\}$, where $r_t^i = -d(s_{t+1}, n_i)$. In other words, given a transition (s_t, a_t, s_{t+1}) and a goal g from $\mathcal{D}$, we push to the replay buffer a set of relabeled transitions with all goals on the shortest path from s_t to g (and their corresponding rewards).

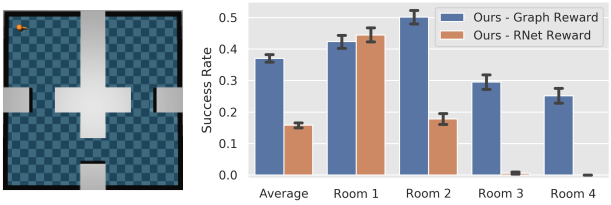
Edge augmentation. Similarly to the subgoal augmentation technique, we consider a relabeled transition $(s_t, a_t, g, r_t, s_{t+1})$ and the shortest path $(n_0, ..., n_{P-1})$ between the corresponding nearest neighbors of s_t and g. This time, we keep the same goal g for every augmented transition, but for every edge $(n_{i-1}, n_i), i \in \{1, P-1\}$, we add the relabeled transition $(s_t^i, a_t^i, g, r_t^i, s_{t+1}^i)$ to $\mathcal{B}$ where $(s_t^i, a_t^i, s_{t+1}^i) \in \mathcal{D}$, $\text{NN}_{\text{out}}(s_t^i) = n_{i-1}$, $\text{NN}_{\text{in}}(s_{t+1}^i) = n_i$ and $r_t^i = -d(s_t^i, g)$. Note that the existence of such a transition in $\mathcal{D}$ is guaranteed by construction: an edge is added to the graph from one node to another *iff* there exist a transition in $\mathcal{D}$ whose corresponding nearest neighbors are these two nodes (in the same order).

Once the replay buffer $\mathcal{B}$ is filled, the goal-conditioned policy can be trained using any off-policy algorithm. In our implementation, we chose Soft Actor-Critic [33].

5 Experiments

5.1 Environments & data collection

We perform experiments on two continuous control tasks with state-based inputs.



(a) UMaze

(b) RNet Rewards vs. Graph Rewards

Figure 2: (a) UMaze environment, and (b) Comparison of the performance of the goal-conditioned policy trained with RNet and Graph-based rewards on the UMaze

UMaze [34]. The first environment, shown in Figure 2a, is a two-dimensional U-shaped maze with continuous action space and a fixed initial position. We generate the training data for this environment by deploying a random policy with randomized start position in the maze. We collect 10k trajectories of length 1k. We evaluate the goal-conditioned agent trained offline on this task by giving a goal sampled at random in the environment to the agent and evaluating the final euclidean distance to the goal.

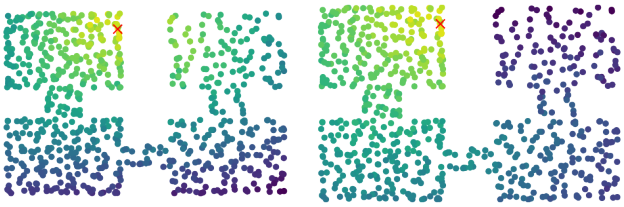
RoboYoga Walker [19]. Introduced by Mendonca et al. [19], the challenging RoboYoga benchmark is based on the Walker domain of the DeepMind Control Suite [25], and consists of 12 goals that correspond to body poses inspired from yoga (*e.g.* lying down, raising one leg or balancing). We consider the state-based version of the task, and use the task-agnostic dataset from Yarats et al. [23] generated with an unsupervised exploration policy. It contains 10k trajectories of length 1k obtained by deploying the "proto" [30] algorithm in the Walker domain. The success metric of the evaluation policy is assessed by the pose of the humanoid at the end of the episode.

5.2 Ablation & design choices

In this section, we explain the design choices of our graph-based reward shaping method. We first show that graph structure is necessary for long-term planning. Then, we explain the importance of the directness of the graph on tasks with asymmetric behaviours. Finally, we show the impact the transition augmentation techniques when labeling data for the goal-conditioned policy.

Necessity of graph-based rewards. An important component of our method is the construction of a graph $\mathcal{M}$ that allows for computing a distance with good global properties in the state space. In order to empirically validate this hypothesis, we performed a comparison between the goal-conditioned policy trained with RNet rewards (*i.e.*, by using the distance d_l from equation (5)) and the one trained with both distance terms as reward. We perform this experiment on the UMaze environment, and show results in Figure 2b. We see that the model trained with graph rewards outperforms the one trained with RNet rewards overall, particularly for faraway goals (ie. rooms 3 & 4). We also notice that the model trained with RNet rewards is slightly better for goals that are close from the initial position, which highlights the fact that RNet is good for estimating local distances. Figure 3 shows a qualitative visualization of the RNet and graph-based rewards. We see that RNet distance does not properly capture global distance, as it shows low values between states in the first and fourth rooms.

Importance of graph directness. In a second experiment, we investigate the importance of the asymmetry of the RNet and the directness of the graph. To do so, we implement an undirected version of our method where the RNet is symmetric and the graph is undirected. All other components of our method remain the same. We first look at some qualitative visualizations of the shortest path in the undirected and directed graphs in the RoboYoga task, as shown in Figure 4. In the undirected case, the humanoid defies the laws of gravity and will be encouraged to get on its head from the back, which might be extremely difficult, or even infeasible. In the directed case, the shortest path



(a) RNet distance (b) Graph distance

Figure 3: (a) Heatmap of rewards computed with the RNet (a) and graph (b) distances.

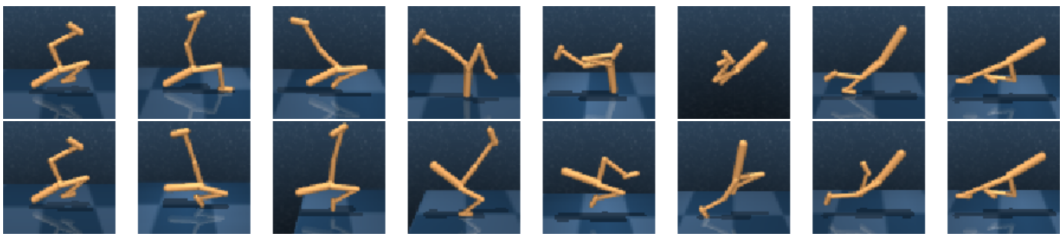


Figure 4: Shortest Path visualization on the RoboYoga Walker task for undirected (top) and directed (bottom) graphs.

fosters the agent to first get back on its legs, and then lean forward. In this exemple, the gravity makes the dynamics of the environment non-symmetric and non-fully reversible, which justifies the directed formulation described in our method.

Transition sampling strategy. As a final ablation study, we study the utility of the transition augmentation techniques described in subsection 4.4. We run experiments with the four possible variants of our method: (i) without any augmentation, (ii) with edge augmentation only, (iii) with subgoal augmentation only and (iv) with both augmentations. We execute this experiment on the RoboYoga task, and show results in Figure 6b. We see that both of the augmentation techniques improve the performance of the goal-conditioned, with subgoal augmentation showing the greater improvement. Moreover, we note that combining both augmentations improves the performance further. For the reminder of the experiments, we will always use both augmentation techniques.

5.3 Comparison to prior work

Baselines. We compare our method to prior works on unsupervised goal-conditioned policy learning. We perform an apples-to-apples comparison by implementing the baselines using the same learning framework than that of our method, and simply changing the reward relabeling process. We compare with the following baselines:

- **Hindsight Experience Replay [HER] [2]** A re-implementation of the standard unsupervised RL technique, adapted to the offline setting. More precisely, we relabel sub-trajectories from $\mathcal{D}$ with a sparse reward, which is equal to 1 only for the final transition of the sub-trajectory, and 0 everywhere else. Following Chebotar et al. [1], we also label sub-trajectories with goals sampled at random in $\mathcal{D}$ and zero reward.
- **HER [2] with random negative action** A variant of HER where, for a transition in $\mathcal{D}$ we sample an action uniformly at random in the action space and label it with zero reward. This helps leveraging the problem of over-estimation of the Q-values for unseen actions mentioned in Chebotar et al. [1].
- **Actionable Models [1]** A method based on goal-conditioned Q-learning with hindsight relabeling. We re-implemented the goal relabeling procedure that uses the Q-value at the

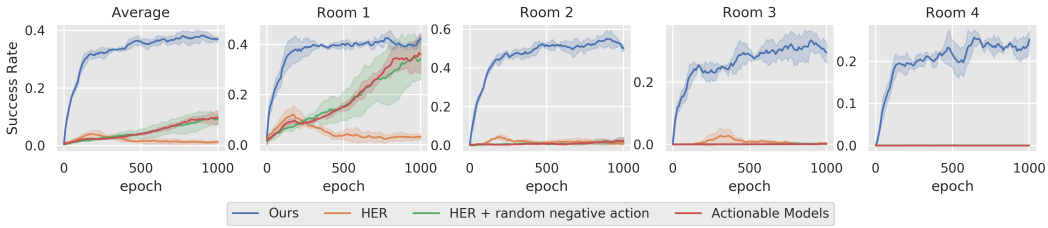
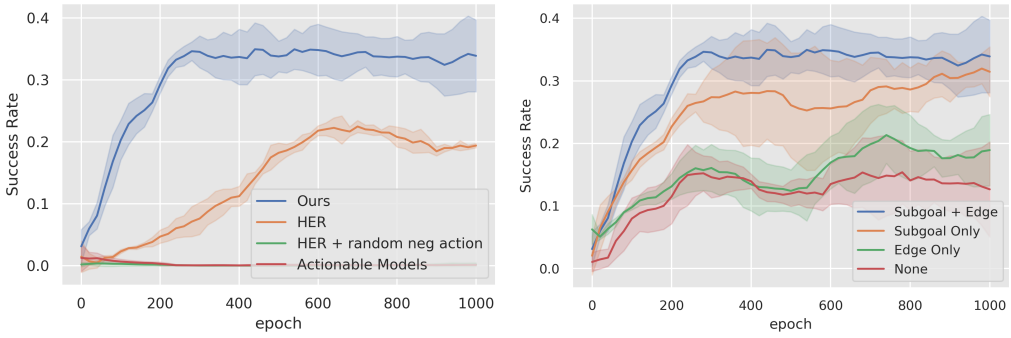


Figure 5: Performance on the UMaze-Task



(a) Comparison to baslines

(b) Impact of Transition Augmentation

Figure 6: Performance of on the RoboYoga Walker task

final state of sub-trajectories in $\mathcal{D}$ to enable goal chaining, as well as the negative action sampling trick.

Comparison on UMaze. We compare our method to the baselines on the UMaze task, and show results in Figure 5. We see that our model outperforms all baselines overall, and shows greater improvements on goals that are far from the initial position. Interestingly, we note that the Actionable model is able to reach goals in the first room only, and is unable to navigate towards goals from the second room. This confirms the idea that sparse rewards make it difficult for the policy to learn long-horizon tasks.

Comparison on RoboYoga Walker. In a second experiment, we compare our method to baselines on the RoboYoga task. These results are shown in Figure 6a. Here again, our method outperforms prior work by a large margin, and Actionable models does not make any significant improvement. The results broken down by goal are shown in the supplementary material. Overall the empirical results suggest that our dense reward shaping method allows for faster and more robust offline goal-conditioned policy training.

6 Conclusion: Summary and Limitations

We proposed a method for learning multi-task policies from pre-generated datasets in an offline and unsupervised fashion, *i.e.*, without requiring any additional interaction with the environment, nor manually designed rewards. Our method leverages a self-supervised stage that aims at learning the dynamics of the environment from the offline dataset, and that allows for shaping a dense reward function. The shaped reward function shows great improvement compared to prior works based on hindsight relabeling, especially on long-horizon tasks where dense rewards are crucial for good policy learning. We presented one form of self-supervised learning on the offline dataset in this work. This choice needs further evaluation by comparing different self-supervision strategies and evaluating their impact on the policy learning task. This constitutes an exciting future prospect.

References

[1] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. C. Julian, C. Finn, et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. In *International Conference on Machine Learning*, pages 1518–1528. PMLR, 2021.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[3] R. Yang, Y. Lu, W. Li, H. Sun, M. Fang, Y. Du, X. Li, L. Han, and C. Zhang. Rethinking goal-conditioned supervised learning and its connection to offline rl. *arXiv preprint arXiv:2202.04478*, 2022.

[4] J. Li, C. Tang, M. Tomizuka, and W. Zhan. Hierarchical planning through goal-conditioned offline reinforcement learning. *arXiv preprint arXiv:2205.11790*, 2022.

[5] N. Savinov, A. Raichuk, D. Vincent, R. Marinier, M. Pollefeys, T. Lillicrap, and S. Gelly. Episodic curiosity through reachability. In *International Conference on Learning Representations*, 2018.

[6] L. P. Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer, 1993.

[7] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[8] S. Nasiriany, V. Pong, S. Lin, and S. Levine. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

[10] S. Sukhbaatar, E. Denton, A. Szlam, and R. Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv preprint arXiv:1811.09083*, 2018.

[11] O. OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. de Oliveira Pinto, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. 2020.

[12] A. Campero, R. Raileanu, H. Kuttler, J. B. Tenenbaum, T. Rocktäschel, and E. Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2020.

[13] D. Warde-Farley, T. Van de Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih. Unsupervised control through non-parametric discriminative rewards. In *International Conference on Learning Representations*, 2018.

[14] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, 31:9191–9200, 2018.

[15] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[16] V. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, pages 7783–7792. PMLR, 2020.

[17] S. Venkattaramanujam, E. Crawford, T. Doan, and D. Precup. Self-supervised learning of distance functions for goal-conditioned reinforcement learning. *arXiv preprint arXiv:1907.02998*, 2019.

[18] K. Hartikainen, X. Geng, T. Haarnoja, and S. Levine. Dynamical distance learning for semi-supervised and unsupervised skill discovery. In *International Conference on Learning Representations*, 2019.

[19] R. Mendonca, O. Rybkin, K. Daniilidis, D. Hafner, and D. Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34, 2021.

[20] L. Mezghani, P. Bojanowski, K. Alahari, and S. Sukhbaatar. Walk the random walk: Learning to discover and reach goals without supervision. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

[21] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[22] C. Gulcehre, Z. Wang, A. Novikov, T. Le Paine, S. G. Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. 2020.

[23] D. Yarats, D. Brandfonbrener, H. Liu, M. Laskin, P. Abbeel, A. Lazaric, and L. Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.

[24] N. Lambert, M. Wulfmeier, W. Whitney, A. Byravan, M. Bloesch, V. Dasagi, T. Hertweck, and M. Riedmiller. The challenges of exploration for offline reinforcement learning. *arXiv preprint arXiv:2201.11861*, 2022.

[25] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[26] M. Laskin, D. Yarats, H. Liu, K. Lee, A. Zhan, K. Lu, C. Cang, L. Pinto, and P. Abbeel. Urlb: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*, 2021.

[27] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[28] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

[29] D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[30] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pages 11920–11931. PMLR, 2021.

[31] S. Endrawis, G. Leibovich, G. Jacob, G. Novik, and A. Tamar. Efficient self-supervised data collection for offline robot learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4650–4656. IEEE, 2021.

[32] Y. J. Ma, J. Yan, D. Jayaraman, and O. Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. *arXiv preprint arXiv:2206.03023*, 2022.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[34] Y. Kanagawa. https://github.com/kngwyu/mujoco-maze.