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

Lina Mezghani
Meta AI, Inria[*]
linamezghani@fb.com

Sainbayar Sukhbaatar
Meta AI

Piotr Bojanowski
Meta AI

Alessandro Lazaric
Meta AI

Karteek Alahari
Inria[*]

1 Implementation details

1.1 Self-supervised reward shaping

We provide pseudo-code for the two stages of our approach: the graph building process is shown in Algorithm 1, and the steps for filling the replay buffer are shown in Algorithm 2. Here we use the notation $R(s, \mathcal{M})$ as the maximum RNet value between s and all nodes in $\mathcal{M}$ *i.e.*,

$$R(s, \mathcal{M}) := \max_{m \in \mathcal{M}} R(s, m)$$

Algorithm 1 Building the directed graph $\mathcal{M}$

```
Input: pre-collected dataset $\mathcal{D}$, Reachability Network $R$
Initialize: $\mathcal{M} = \{\}$

/ * Build the set of nodes * /
for each state $s$ in $\mathcal{D}$ do
    if $R(s, \mathcal{M}) < 0.5$ and $R(\mathcal{M}, s) < 0.5$ then
        Update $\mathcal{M} := \mathcal{M} \cup \{s\}$
    end if
end for

/ * Build edges * /
for each transition $(s_t, s_{t+1})$ in $\mathcal{D}$ do
    Let $n_t := \text{NN}_{\text{in}}(s_t) = \text{argmax}_{n \in \mathcal{M}} R(n, s_t)$
    Let $n_{t+1} := \text{NN}_{\text{out}}(s_{t+1}) = \text{argmax}_{n \in \mathcal{M}} R(s_{t+1}, n)$
    Create directed edge from $n_t$ to $n_{t+1}$
end for
```

[*] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

Algorithm 2 Building replay buffer $\mathcal{B}$ for offline policy training

```
Input: pre-collected dataset $\mathcal{D}$, Reachability Network $R$, directed graph $\mathcal{M}$
Initialize: $\mathcal{B} = \{\}$

while $\mathcal{B}$ is not full do
    Sample a transition $(s_t, a_t, s_{t+1})$ at random in $\mathcal{D}$
    Sample a goal $g$ at random in $\mathcal{D}$

    Compute $d_l(s_{t+1}, g) := 1 - R(s_{t+1}, g)$
    Let $n_{t+1} := \text{NN}_{\text{out}}(s_{t+1}) = \text{argmax}_{n \in \mathcal{M}} R(s_{t+1}, n)$
    Let $n_g := \text{NN}_{\text{in}}(g) = \text{argmax}_{n \in \mathcal{M}} R(n, g)$
    Compute $d_g(s_{t+1}, g) := \text{ShortestPathLength}(n_{t+1}, n_g)$

    Compute reward $r_t := -(d_l(s_{t+1}, g) + d_g(s_{t+1}, g))$
    Relabel transition with goal $g$ and reward $r_t$, and
    Push $(s_t, a_t, g, r_t, s_{t+1})$ to $\mathcal{B}$
end while
```

1.2 Actionable Models baselines re-implementation

In this section, we provide details for our re-implementation of Actionable Models [1] and HER [2]. Since we are using the same optimization algorithm for offline policy training for these baselines and our method, the only difference lies in how the transitions in the pre-collected dataset $\mathcal{D}$ are relabeled to build the replay buffer $\mathcal{B}$.

In HER [2], the idea is to sample a trajectory and a goal g at random in $\mathcal{D}$, and to cut the trajectory at a step i. Each transition in the trajectory (until step i) is then relabelled twice: once with the goal g and reward 0 for all transitions, and once with goal s_i (the final state of the trajectory) and reward 0 for all transitions except the final one that gets a reward of 1. The pseudo-code for this method is shown in Figure 1.

Actionable Models [1] relies on a similar idea as in HER [2], but contains two additional steps to improve the method. The first step is a form of *goal chaining* and it consists in using the Q-value at the final state of the trajectory to compute the reward for the final transition. The second step aims at balancing the unseen action effect, in order to regularize the action space. In practice, it consists in sampling negative actions from the policy and label zero-reward transitions with these actions. The implementation of both tricks is shown in Figure 1.

For the implementation of the third baseline, HER + random negative action, the overall algorithm is the same as HER, except that we also generate transitions with negative actions, similar to Actionable Models. This time, the negative actions are not sample from the policy, but are generated uniformly at random from the action space.

Algorithm 3 HER

```
Input: dataset $\mathcal{D}$
Initialize: $\mathcal{B} = \{\}$
while $\mathcal{B}$ is not full do
    Sample trajectory $\tau \in \mathcal{D}$
    Sample goal $g \in \mathcal{D}$
    Randomly cut $\tau$ at step $i$
    for $j \in \{0, ..., i-2\}$ do
        $(s_j, a_j, g, 0, s_{j+1}) \rightarrow \mathcal{B}$
        $(s_j, a_j, s_i, 0, s_{j+1}) \rightarrow \mathcal{B}$
    end for
    $(s_{i-1}, a_{i-1}, g, 0, s_i) \rightarrow \mathcal{B}$
    $(s_{i-1}, a_{i-1}, s_i, 1, s_i) \rightarrow \mathcal{B}$
end while
```

Algorithm 4 Actionable Models

```
Input: dataset $\mathcal{D}$,
goal-conditioned critic network $Q$,
goal-conditioned policy $\pi$
Initialize: $\mathcal{B} = \{\}$
while $\mathcal{B}$ is not full do
    Sample trajectory $\tau \in \mathcal{D}$
    Sample goal $g \in \mathcal{D}$
    Randomly cut $\tau$ at step $i$
    for $j \in \{0, ..., i-2\}$ do
        $(s_j, a_j, g, 0, s_{j+1}) \rightarrow \mathcal{B}$
        $a_1 \sim \pi(s_j, g)$
        $(s_j, a_1, g, 0, s_{j+1}) \rightarrow \mathcal{B}$
        $(s_j, a_j, s_i, 0, s_{j+1}) \rightarrow \mathcal{B}$
        $a_2 \sim \pi(s_j, s_i)$
        $(s_j, a_2, s_i, 0, s_{j+1}) \rightarrow \mathcal{B}$
    end for
    $(s_{i-1}, a_{i-1}, g, Q(s_i, a_{i-1}, g), s_i) \rightarrow \mathcal{B}$
    $a_3 \sim \pi(s_{i-1}, g)$
    $(s_{i-1}, a_3, g, 0, s_i) \rightarrow \mathcal{B}$
    $(s_{i-1}, a_{i-1}, s_i, 1, s_i) \rightarrow \mathcal{B}$
    $a_4 \sim \pi(s_{i-1}, s_i)$
    $(s_{i-1}, a_4, s_i, 0, s_i) \rightarrow \mathcal{B}$
end while
```

Figure 1: Pseudo-code for replay buffer filling with HER [2] and Actionable Models [1] methods. We compare both implementations by showing in red modifications related to goal chaining, and in blue edits related to unseen action regularization.

1.3 Hyper-parameters

We first list the hyper-parameters for the self-supervised reward shaping phase in Table 1. Table 2 details the hyper-parameters for the offline policy training stage with SAC [3]. For the Actionable Models [1] and HER [2] baselines, we used the same parameters as in our approach, with the exception of some parameters specific to these methods, shown in Table 3.

These hyper-parameters were obtained by performing a random search for all the methods over several parameter values. All the experiments in this work were performed on 3 random seeds.

Common hyper-parameter	Value	
Task	UMaze	RoboYoga
Number of training pairs	5×10^5	5×10^5
Ratio of negatives	0.5	0.5
Ratio of negatives from same trajectory	0.5	0.5
Reachability threshold (τ_{reach})	5	2
Batch size	512	512
Learning rate	0.001	0.0003
Weight decay	0.00001	0.00001
Total number of training epochs	20	100
Capacity of the directed graph	1000	10000

Table 1: Hyper-parameters for reachability network training and directed graph construction.

Hyper-parameter	Value	
Task	UMaze	RoboYoga
Replay buffer capacity	10^6	10^6
Batch size	2048	2048
Discount (γ)	0.90	0.95
Number of updates per epoch	1000	1000
Total number of epochs	1000	1000
Target update interval	1	1
Soft update coefficient (τ)	0.005	0.005
SAC entropy parameter (α)	0.05	0.01
Optimizer	Adam	Adam
Learning rate	0.0003	0.0003
Action repeat	1	2
Reward scaling factor	0.1	0.5

Table 2: Hyper-parameters for offline policy learning with SAC [3] with our method.

Hyper-parameter	Value	
Task	UMaze	RoboYoga
Discount (γ)	0.99	0.99
SAC entropy parameter (α)	0.01	0.001
Learning rate	0.0001	0.0001
Reward scaling factor	1	10

Table 3: Hyper-parameters for offline policy learning with SAC [3] specific to Actionable Models [1] and HER [2] baselines.

1.4 Architecture details

Reachability Network [4] The RNet has a siamese architecture with two embedding heads (one for each observation of the pair) with tied weights, and a comparator network that compares both embeddings and returns a reachability score. For the UMaze task, we used an embedding head with 3 fully-connected layers with batch normalization and Tanh activations, with a hidden size of 64 and an embedding size of 16. For the Roboyoga Walker task, the embedding network has the same architecture, but we increased both the hidden and embedding sizes to 128. The comparator network is also a fully-connected network. It contains batch normalization and ReLU activations. The hidden size for the UMaze (respectively the RoboYoga Walker) task is set to 16 (resp. 128) and the number of layers is 2 (resp. 4).

Policy Network The goal-conditioned policy network takes as input both the observation and the goal, in separate heads with the same architecture but independent weights. These heads are implemented as 3-layer fully-connected networks with Tanh activations, hidden size of dimension 64, and 16 dimensions for the feature size. The output from both the heads is then concatenated and fed into a 2-layer fully-connected network of width 256. The critic network has the same architecture for both observation and goal heads, and is followed by 3 fully connected-layers of width 256.

2 Full results on RoboYoga Walker task

We show the comparison of our method against the aforementioned baselines on each of the 12 goals of the RoboYoga Walker task in Figure 2. These goals are illustrated in Figure 3. We see that our method masters most of the goals that do not require balancing (Lie Back & Front, Legs Up, Lunge), and succeeds quite well at more complicated goals like Side Angle, Lean Back and Bridge, but is unable to progress in complex goals like Head Stand or Arabesque.

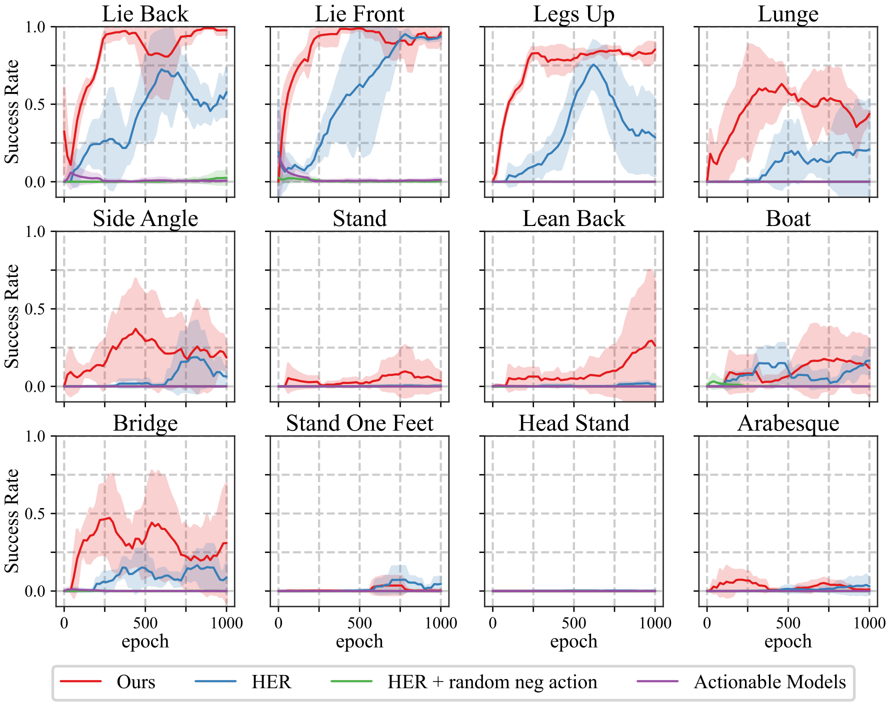
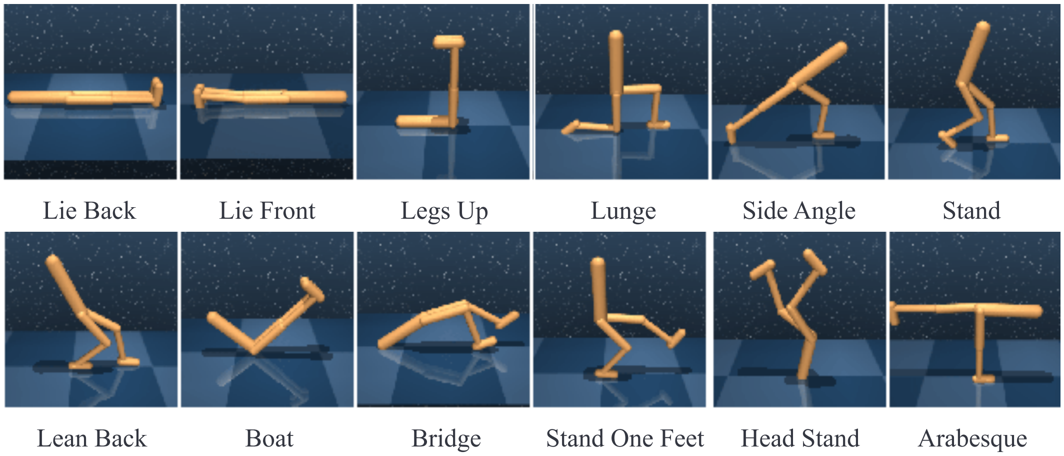


Figure 2: Performance on the RoboYoga Walker talk for each of the 12 goals.



Lie Back Lie Front Legs Up Lunge Side Angle Stand

Lean Back Boat Bridge Stand One Feet Head Stand Arabesque

Figure 3: Visualization of the 12 evaluation goals for the RoboYoga Walker task.

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] 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.

[4] 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.