Summary of the paper: Hindsight Experience Replay

Ignacio Monardes

February 16, 2026

Reference

Authors: Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba
Title: *Hindsight Experience Replay*
Conference/Journal: 30° NeurIPS
Link: `https://arxiv.org/pdf/1707.01495`
Team: OpenAI

1 Motivation

Current problem

The main objectives in tasks such as robotics is to move blocks because it is very difficult to achieve the goals, generating sparse rewards. These trajectories may be seen as useless because all of them lead to a 0 reward and doesn't help to train the model at all. At the time of this paper, the solution was to create handmade rewards with a curriculum to lead the agent to achieve new goals, but it used complicated formulas and it is not generizable to many environments.

Relevance of the problem

The problem is relevant because we could use these sparse trajectories to learn how to get to some destinations, it is not the final goal, but it helps the model understand how to interact and to achieve things with something that could be generalized.

Hole in the literature / previous limitations

The hole in the literature is the lack of a generalized algorithm that uses the sparse trajectories to learn useful things about the behaviour. This could produce a more efficient learning.

2 Idea

Main Idea

Humans can learn from undesired outcomes by doing it, and this helps our training to achieve what we really want to do. This makes the use of trajectories more sample-efficient, and helps

with sparse rewards, because we will learn how to achieve many final states. To do this we have to use an experience buffer because we are going to "change" the states and the rewards of the trajectories and learn by this.

Context

With DQN we have a replay buffer which we use to train our neural network that approximates Q-values. We train DQN by using this loss:

$$\mathcal{L} = \mathbb{E}\left[(Q(s_t, a_t) - y_t)^2\right], \quad y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$$

with many trajectories from the replay buffer in the form (s_t, a_t, r_t, s_{t+1}).

DDPG is similar to DQN, but for continuous actions. Here, our behavior policy is: $\pi_b(s) = \pi(s) + N(0, 1)$. We have two neural networks: an *actor* which is a target policy π and a *critic* Q^π that says how good are the actions of the actor. The critic is trained like DQN, but with $y_t = r_t + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1}))$.

The actor is trained with this loss:

$$\mathcal{L}_a = -\mathbb{E}_s\left[Q^\pi(s_{t+1}, \pi(s_{t+1}))\right]$$

We could also use many goals to achieve it, so for that we can just add the goal as part of the state: $\hat{s}_t := (s_t, g)$.

Innovation

The innovation consists of learning how to recreate many outcomes by learning from the trajectories of our replay buffer. We can make this by just getting the final state $g := s_T$ and adding the states $\hat{s}_t := (s_t, g)$ in the buffer for the whole trajectory.

It works as an impicit *curriculum*.

3 Methodology

More formally, we have some requisites:

- For every goal $g \in \mathcal{G}$ we have to know if the current state matches or not the goal g. In other words:
$$\forall g \in \mathcal{G}\ \exists f_g : \mathcal{S} \to \{0, 1\}.$$
- For every state $s \in \mathcal{S}$ we can easily find a goal g which is satisfied by this state. In mathematical words: exists $m : \mathcal{S} \to \mathcal{G}$ such that
$$\forall s \in \mathcal{S}\ f_{m(s)}(s) = 1.$$
This could be done if we just use $\mathcal{S} = \mathcal{G}$ and $f_g(s) = \mathbb{1}_{[s=g]}$.

Modeling

Algorithm 1: HER pseudocode

```
agent = make_offline_agent()
replay = make_replay()
env = make_env()
goal_sampler = make_goal_sampler()

for episode in range(M):
    state = env.reset()
    g = goal_sampler.single_sample()
    states = [state]
    actions, rewards = [], []
    for t in range(T):
        a_t = agent.behaviour_policy(s_t, g)
        state, reward = env.step(a_t)
        states.append(state)
        actions.append(a_t)
        rewards.append(reward)

    for t in range(T):
        r_t = r(states[t], actions[t], g)
        replay.add([(states[t], g), actions[t], r_t, (states[t+1], g)])
        G = goal_sampler.multiple_sample()
        for g_prime in G:
            r_prime = r(states[t], actions[t], g_prime)
            replay.add(
                [(states[t], g_prime), actions[t], r_prime, (states[t+1],
                    g_prime)]
            )

    for t in range(N):
        minibatch = replay.minibatch()
        agent.update(minibatch)
```

Architecture

It was used with simple MLPs for the policies with ReLU activations. Training is performed using DDPG with ADAM optimizer.

4 Experiments

Environments and datasets

It was trained on a 7-DOF robotic arms in a simulated environment.

There are three different classes of tasks:

- Pushing: push a block in front of the robot to a target location. The fingers are locked. The learned behaviour is a mixture of pushing and rolling.

- Sliding: A puck (disk) is placed on a slippery table and the robot must hit the puck to slide to the target.
- Pick-and-place: Fingers are not locked in. The arm must pick a box and move it to a target in the air.

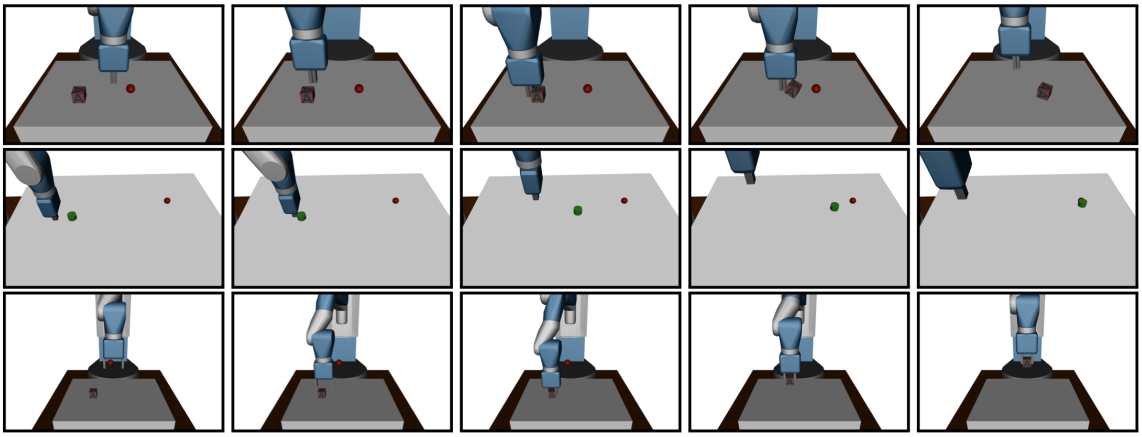

Figure 2: Different tasks: *pushing* (top row), *sliding* (middle row) and *pick-and-place* (bottom row). The red ball denotes the goal position.

There are two approaches for the target. One could just use the reward just as the indicatrix of the final state or maybe a shaped reward as the measure of how near am I to the final state. One example is:

$$r(s, g, a) := \lambda |g - s_{obj}|^p - |g - s_{obj}|^p$$

with $\lambda \in \{0, 1\}$, $p \in \{1, 2\}$ are hyperparameters.

The result is that geometrical rewards are worse than the binary final state because they tend to stimulate to be near but not to complete the mission.

We could also try different strategies for adding more experiences to the buffer. The default strategy will be *final* which consists in only adding the final state.

$$multiple\_sample(s_0, s_1, \ldots, s_T) = \{m(s_T)\}.$$

Other possible options are the following strategies:

- *future*: replay with k random states from the current episode but with only states that will be in the future.

$$multiple\_sample(s_0, s_1, \ldots, s_t) = \{m(s_{t'_1}), m(s_{t'_2}), \ldots, m(s_{t'_k})\}, \quad t'_i > t \; \forall i$$

- *episode*: replay with k random states from the current episode.

$$multiple\_sample(s_0, s_1, \ldots, s_t) = \{m(s_{t'_1}), m(s_{t'_2}), \ldots, m(s_{t'_k})\}$$

- *random*: replay with k random states encountered so far in the whole training procedure.

Assumptions

The states are the physical data such as: angles, velocities of the robot joints and positions, velocities and rotations of the objects.

The goals are every position in a 3D plane. The function that decides if the state accomplished the goal is $f_g(s) = \mathbb{1}_{\|g - s_{obj}\| \leq \varepsilon}$. The mapping is just $m(s) = s_{obj}$.

The reward is just $r(s, a, g) := -\mathbb{1}_{f_g(s')=0}$ with s' the next state of the environment.

For observations we could use just the position of the gripper the relative position of the object and target to the gripper and the distance between the fingers. Q also receives the linear velocity of the gripper and fingers and relative velocities of the object.

The gripper rotation is fixed to not be used. Three dimensions specify the desired relative gripper position at the next timestep (MuJoCo constraints). The last dimension specifies the desired distance between two fingers.

Baselines

The baselines are: DDPG, DDPG + count based exploration such as:

5 Results

HER outperforms DDPG variants that couldn't solve the tasks.

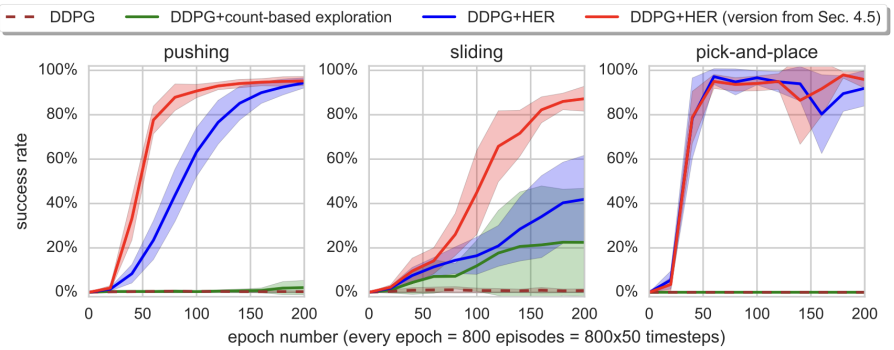

Figure 3: Learning curves for multi-goal setup. An episode is considered successful if the distance between the object and the goal at the end of the episode is less than 7cm for pushing and pick-and-place and less than 20cm for sliding. The results are averaged across 5 random seeds and shaded areas represent one standard deviation. The red curves correspond to the `future` strategy with $k = 4$ from Sec. 4.5 while the blue one corresponds to the `final` strategy.

HER could solve problems even if the real task was only one and not multiple.

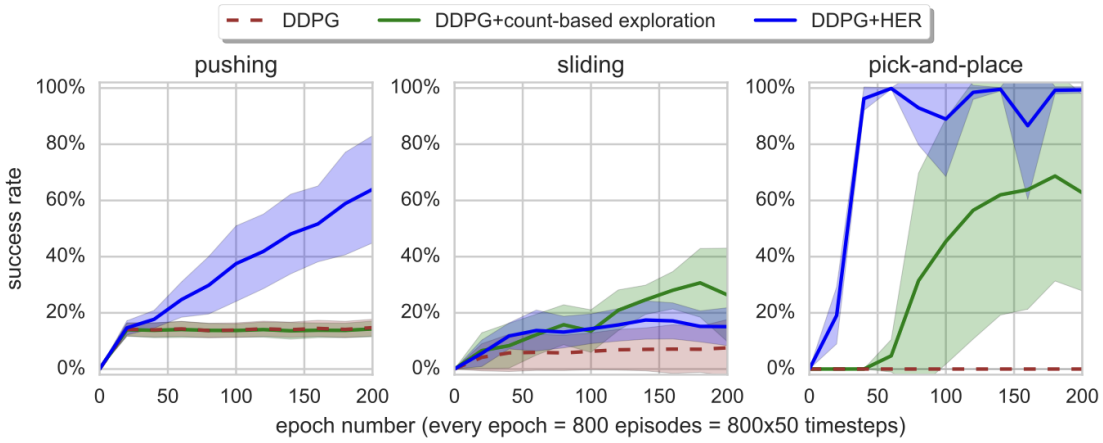

Figure 4: Learning curves for the single-goal case.

HER with *future* and *final* strategies tends to achieve better performance than *random*, *episode* and *no HER*.

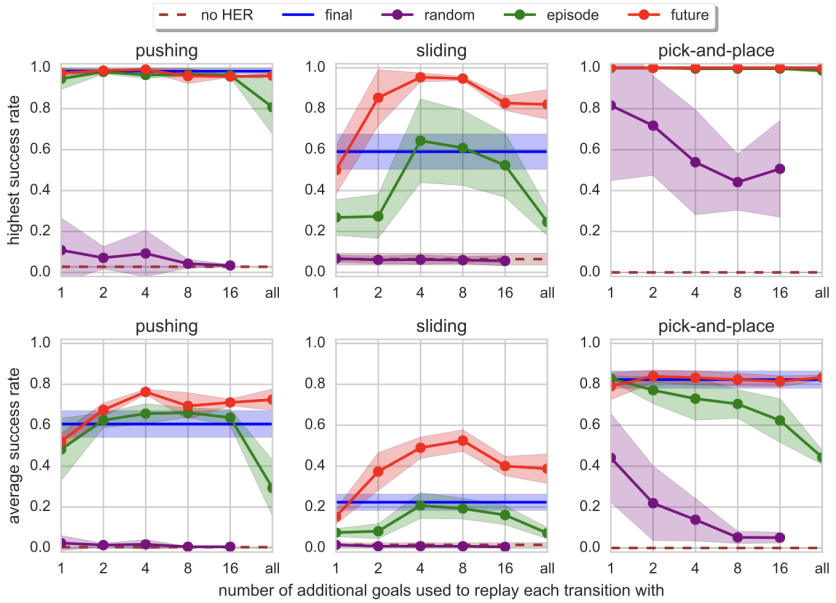

Figure 6: Ablation study of different strategies for choosing additional goals for replay. The top row shows the highest (across the training epochs) test performance and the bottom row shows the average test performance across all training epochs. On the right top plot the curves for `final`, `episode` and `future` coincide as all these strategies achieve perfect performance on this task.

Deployment in real robot

With a trained *future* strategy and without finetuning the robot achieved 2/5 goals. The position of the box was get from a trained CNN using a camera images. After retraining the policy with gaussian noise to the observations the success increased to 5/5.

Good and bad performance in different scenarios

It seems that the *future* strategy with $k = 4$ performs better than the other approaches. For some reason the *final* strategy is not better than the future but it is used as default in the paper.

The algorithm works to learn intermediate goals to achieve a higher goal as an implicit curriculum.

6 Critical Discussion

Strength

It works because it encourages learning acrros multiple goals in a sparse environment.

It is demonstrated with the figures that this methodology helps to achieve the final goal.

Weakness

I don't know if it could encourage to go to undesirable states for other goals that are undesired.

Questionable assumptions

What it is not clear

It is mentioned that HER with geometrical rewards was worse than binary goal but there is no figure comparing both of these approaches.

7 Conclusions

Personal opinion

Is it a good contribution?

Absolutely. It is easy to implement and helps a lot to achieve not only one goal but multiple at the same time. The algorithm also works when we have just one final goal.

Would I use it on my investigation?

Yes. I could combine it with dreamer.