Adding Priortized Experience to Hindsight Experience Replay

Saumya Mehta
Indiana University
mehtasau@iu.edu

Abstract

Experience allows reinforcement learning agents to remember past experiences and learn from them. For mujoco based environments, it is not feasible to take random actions in the environment and hope to achieve decent results. Even with an off-policy algorithm like Deep Deterministic Policy Gradient(Lillicrap et al., 2015),the Reinforcement Learning agent won't be able to learn environment dynamics effectively as the goal is always fixed and the reward is sparse. I use Hindsight Experience Replay(HER)(Marcin Andrychowicz, 2018) which helps the Reinforcement Learning agent learn effectively with sparse rewards. HER allows the Reinforcement Learning agent to learn from previous experiences in a sample efficient manner which makes learning with sparse rewards feasible. I further experiment with combining HER with Prioritized Experience Replay(PER)(Schaul et al., 2015) which adds a prior over collected sample transitions. This allows replaying important transitions more often and hence leads to more effective learning. I show the experiment results on two mujoco environments, (FetchSlide) and(FetchPickAndPlace).

1 Introduction

Reinforcement Learning combined with Neural Networks has been successful in solving many tasks but for some tasks like robotics, there is a need for reward engineering (reward shaping). This can be an expensive process and might not be feasible to every environment. Hindsight Experience Replay alleviates this issue by considering multiple goals instead of a single goal. It treats the state at the end of a trajectory as a valid goal state. This allows the agent to learn from sparse rewards by learning about the environment dynamics from the sampled trajectories.
With Hindsight Experience replay, the transitions are still sampled uniformly from the replay buffer. Prioritized Experience replay applies a prior over the samples in the replay buffer such that more important transitions are sampled more often. This allows for more efficient learning.

2 Background

2.1 Reinforcement Learning

Reinforcement learning is a formalism for an agent interacting with the environment. The environment is assumed to be fully observable and it consists of a state space S, action space A, reward function $r : SXA- > R$. We use a discount factor γ to discount future rewards. Wr try to find an optimal policy Q^* which maximises the expected future reward given a state, action pair. Q^* is given by the following equation:

$$Q^* = E_{s\sim p(.|s,a)}[r(s,a) + \gamma max_{a' \in A} Q^*(s^{'}, a^{'})]$$

2.2 Deep Deterministic Policy Gradients(DDPG)

Deep Deterministic Policy Gradient(Lillicrap et al., 2015) is a model free RL algorithm for continuous action spaces. In DDPG, we have a *target policy*(actor network) $\pi : S- > A$ and an action-value function approximator(critic network) $Q : SXA- > R$. Critic is responsible to calculate Q^π from π generated by actor.
The episodes are generated by adding a noise term to the target policy (here we experiment with Gaussian and Ornstein–Uhlenbeck(Wikipedia contributors, 2022) noise terms). Actor model is trained with mini-batch gradient descent with the loss term as $-E_s[Q(s, \pi(s))]$; where s is sampled from the replay buffer.

2.3 Universal Value Function Approximators(UVFA)

Universal Value Function Approximators(Schaul et al.) is an extension of DQN(Mnih et al., 2015) where there are more than one goals to achieve. The episodes are stored as pairs of state and goal and policy now includes goal along with the current state $\pi : SXG- > A$ which returns a reward at timestamp t. So, the Q function now depends on the state-action pair as well as the goal $Q^{\pi}(s_t, a_t, g) = E[R_t|s_t, a_t, g]$

2.4 Temporal Difference Error(TD Error)

Temporal difference Error is the measure of expected learning progress. This gives more priority to samples with higher TD error and so, such transitions are replayed more often. This can cause a lack of sample diversity which can be addressed by applying a stochastic prioritization and the bias can be corrected by using importance sampling as implied in Prioritized Experience Replay(Schaul et al., 2015).

3 Algorithms

3.1 Hindsight Experience Replay

With Hindsight Experience Replay, the trick is the consider that we want the agent to learn multiple goals instead of a single goal.

3.1.1 Multi goal RL

We can achieve this by using an approach analogous to *Universal Function Approximators*(Schaul et al.). Instead of just storing the states, we can store state->goal pair when we record the transitions. So, the policies are a function of state as well as goal. We assume that there is a mapping for state and goal and new goals can be sampled from this mapping.

3.1.2 Hindsight Generation

After an episode ends(we reach end of a trajectory with length T), we store every transition of the form $s_t- > s_{t+1}$ not only with the original goal but also with the intermediate goals achieved along the trajectory. To decide how much Hindsight Experience I need, I use a factor k which decides the ratio of HER replay vs standard replay. The probability of selecting HER replay can be given by $P_{future} = 1 - \frac{1}{(1+k)}$.

Algorithm 1 Hindsight Experience Replay (HER)

Given:
- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = $1, M$ **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

Figure 1: Hindsight Experience Replay Algorithm

3.2 Prioritized Experience replay

Prioritized Experience replay applies a prior on the standard experience replay buffer so that the trajectories that lead to a higher learning progress are preferred

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset, \Delta = 0, p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Figure 2: Prioritized Experience Replay Algorithm

3.2.1 Applying Prioritization to replay buffer

TD error for a transition is a good measure of how surprising or unexpected a transition is which can measure the expected learning progress. However just using TD error greedily can cause initial transitions with very low TD error to never occur. It can also be very sensitive to noise when the reward are stochastic. Finally, it may be prone to over-fitting as the initial high TD error transitions will be replayed very frequently.
So, following the implementation for *Prioritized Experience Replay*(Schaul et al., 2015), I use a stochastic prioritization which isolated between pure greedy prioritization and uniform random sam-

pling. Thus, the probability of sampling a transition i becomes: $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

4 Architecture Details

There are 2 different kinds of networks used here namely, actor network and critic network. The actor network generates a target policy which is then used as an input for the critic network. The critic network evaluates the quality of policy generated by the actor. Both actor and critic network use adam optimizer with learning rate of 0.001.

4.1 Actor Network

The actor network consists of 3 Fully Connected Layers each activated using ReLu activation function. The default number of units in input and hidden layers is 256. This is followed by an action selection unit which is a fully connected layer with dimensions of $(units\ in\ 3rd\ Fully\ connected\ layer\ X\ number\ of\ actions)$ .This output action selection layer is activated using a tanh activation function.

4.2 Critic Network

The actor network consists of 3 Fully Connected Layers each activated using ReLu activation function. The default number of units in input and hidden layers is 256. This is followed by an Q value output unit which is a fully connected layer with dimensions of $(units\ in\ 3rd\ Fully\ connected\ layer X\ 1)$.

5 Experiments

In section 5.1, I introduce the multi-goal RL environments used in the experiments. In seciton 5.2, I present the findings from my experiment with including Prioritized experience replay with Hindsight Experience replay. In section 5.3, I present the performance differences for a setup with same random seed across environment, numpy and torch vs having individual seeds for each one of them.

5.1 Environment

The environment I used for my experiments is the *7-DOF Fetch Robotics*(FetchPickAndPlace),(FetchSlide) environment from OpenAi. Using OpenAi gym(gym), I create two different environments for two different tasks under consideration:

a. *Sliding:* In this task, a puck is place on a long slipperly table. There is a goal position on the table which is outside the reach of the robot's arm. The goal for the agent is to learn to hit the puck with an amount of force such that it reaches the goal position with some small degree of error.

b. *Pick-and-palce:* In this task, a box is placed in front of the robot. The goal location can either be on the table in front of the robot or in the air. The goal for the agent is to learn how to pick the object and place it on the goal location with some small degree of error.

The environments consist of a state space, an action space, goals, rewards and actions. The agent observes a state and takes an action in that state. Based on the action, it reaches another state/goal state and recieves some rewards. The terms are defined below:

State space: The state space of the system is represented by Mujoco physics engine. It consists of angles, velocities and positions of robot joints and rotation and velocities (angular and linear) of objects.

Action space: The action space is 4 dimensional. First three dimensions represent desired relative position of gripper in the next timestamp. The last dimension represents the distance between two fingers of the gripper.

Rewards: Rewards are sparse and binary. I use reward of -1 when the goal is reached and reward of 0 if the goal is reached with certain tolerance.

5.2 Does PER improve performance over HER?

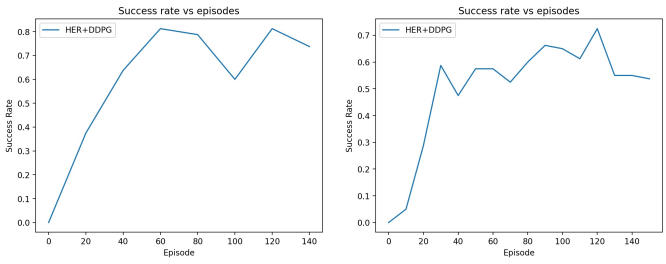

Figure 3: Performance on FetchSlide-v1 environment for a.) Prioritized Experience Replay combined with Hindsight Experience Replay and b.) Hindsight Experience Replay

As we can see from the plots in Figure 3 and Figure 4, PER improves performance over HER in

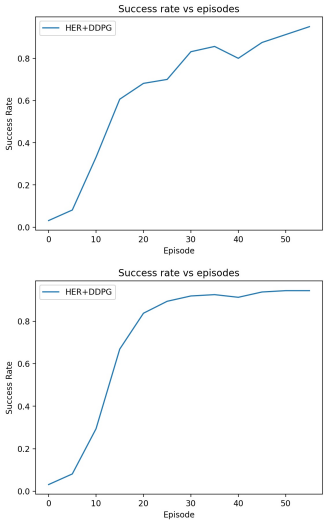

Figure 4: Performance on FetchPickAndPlace-v1 environment for a.) Prioritized Experience Replay combined with Hindsight Experience Replay and b.) Hindsight Experience Replay

both tasks.

5.3 Does setting individual seeds result in a better performance?

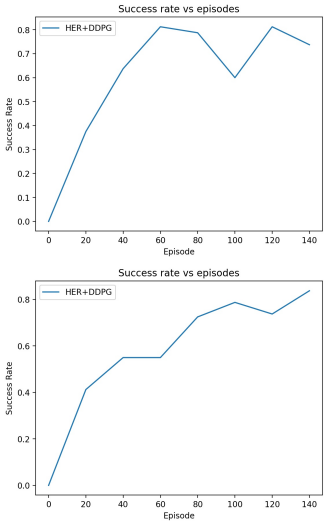

Figure 5: Performance on FetchSlide-v1 environment with PER and HER for a.)Same seed for environment and numpy b.) Different seeds for environment and numpy

From the plots above(Figure 5), we can see that setting individual seeds may not improve performance over setting the same seed. This needs further experimentation with different seed setting.Table1 shows that the average success rate during testing for same seed setting is higher than individual seed setting.
The results from my experiments on Hindsight Experience Replay match the results from the original paper(Marcin Andrychowicz, 2018) for both environments and exceed the results from the paper when I use Prioritized Experience Replay.

5.4 Test results

The model was evaluated for 50 episodes with 50 timestamps per episodes. Table 1 shows the evaluation resuls for different environment, algorithm and seed settings

Environment	**Algorithm**	**Seed**	**average success rate**
FetchSlide-v1	HER	same	31.5%
FetchSlide-v1	HER+PER	same	**42.6%**
FetchSlide-v1	HER+PER	distinct	40%
FetchPickAndPlace-v1	HER	same	77.6%
FetchPickAndPlace-v1	HER+PER	same	**77.8**%

Table 1: Results for Sliding and pushing task for different algorithms and seed settings.

5.5 Demo and code

The demos for the experiments are hosted in the Mujoco Experiments Playlist. The code for this project can be accessed form the github repository.

6 Experiment Setup

6.1 Agent Training

The negative of the loss from the critic network is used as a loss function for the actor network. I also add mean squared value of action to the actor loss. During training, the observation and goal are first clipped in a range of $[-200, 200]$ and then the agent selects a noisy action from the target policy generated by the actor. If the agent uses Prioritized Experience replay, the agent takes another action based on the next observation and the current goal and gets the Q values from the target critic. The agent also calculates Q values for the previous action. The TD error is calculated between the current and next Q values the transition is store in a replay buffer and the stochastic prioritization(3.2.1) is applied to the

samples in the replay buffer using the TD error. The hindsight buffer is generated from the samples which is then used to generate trajectories. I train on 200 epochs. Each epoch runs for 50 cycles and each cycle has 2 roll-outs with each roll-out being 50 timestamps long. The Q-value from the target network is clamped by a factor of γ in the range: $[-\frac{1}{(1-\gamma)}, 0]$.

6.2 Agent Evaluation

During evaluation, the agent evaluates the policy generated by the trained actor model by sampling actions from the policy based on the state, goal pair. The evaluation runs for 50 epochs each of which is 50 timestamps in length.

6.3 Action selection

With probability of 20%, Gaussian noise is added to the actions generated from the actor network. Then, I choose between noisy and random actions by using a binomial distribution which has a 30% chance of selecting random actions.

6.4 Network Updates

The target networks are updated using a polyak average(Polyak and Juditsky, 1992). The target networks' parameters contain 95% of their own parameters and 5% of the parameters of actor/critic networks.

7 Conclusion

I tried matching results from Hindsight Experience Replay(Marcin Andrychowicz, 2018) paper on both FetchSlide-v1 and FetchPickAndPlace-v1 environments and showed that Prioritized Experience Replay(Schaul et al., 2015) can improve the performance for Hindsight Experience Replay on both environments

These experiments demonstrate that Hindsight Experience Replay allows the robot arm to learn push and pick-and-place tasks successfully using only sparse reward and thus saving a lot of time and effort over manual reward engineering.

8 Future work

Combine HER with on policy algorithms like PPO(Schulman et al., 2017) and PETS(Chua et al., 2018) to see if learning uncertainity in the mujoco environment can help achieve better results in push and pick-and-place algorithms

References

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models.

OpenAi FetchPickAndPlace. Fetchpickandplace environment version 1.

OpenAi FetchSlide. Fetchslide environment version 1.

OpenAi gym. Openai gym documentation.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning.

Alex Ray Jonas Schneider Rachel Fong Peter Welinder Bob McGrew Josh Tobin Pieter Abbeel Wojciech Zaremba Marcin Andrychowicz, Filip Wolski. 2018. Hindsight experience replay.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

OpenAi. Openai baselines.

OpenAi. 2018. Ingredients for robotics research.

B. T. Polyak and A. B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

Wikipedia contributors. 2022. Ornstein–uhlenbeck process — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Ornstein%E2%80%93Uhlenbeck_process&oldid=1077542907. [Online; accessed 29-April-2022].