Deep Reinforcement Learning for Sparse-Reward Manipulation Problems

Tejas Khot
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217
tkhot@andrew.cmu.edu

Sumit Kumar
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217
skumar2@andrew.cmu.edu

Abstract

Manipulation of objects using robots involves dealing with high degrees of freedom (DoF). For many tasks of interest, it is not trivial to specify by hand the exact analytical solutions. This calls for considering how manipulation can benefit from machine learning techniques. Posing these tasks in a deep reinforcement learning (RL) setup, we encounter the problem of there being very sparse rewards, available only on success. To this end, we present a technique that enables deep RL agents to learn from the failed examples and thereby explore the solution space more efficiently. Recognizing the limitations of applying this algorithm naively, we developed a new scheme which prioritizes the stored memories and selects them to optimize learning. We demonstrate with experiments that our prioritization based algorithm performs favourably compared to its standard variant.

1 Introduction

This project is aimed at solving sparse reward manipulation tasks using Deep Reinforcement Learning techniques. The robotic platforms considered in this project are shown in Figure 1 and listed below:

- Fetch Robot: It is a 7 DOF manipulator with parallel jaw grippers.
- ShadowHand Robot: This hand has 24 joints out of which 4 are under actuated, so effectively, it has 24 DOFs.

(a) Fetch Robot

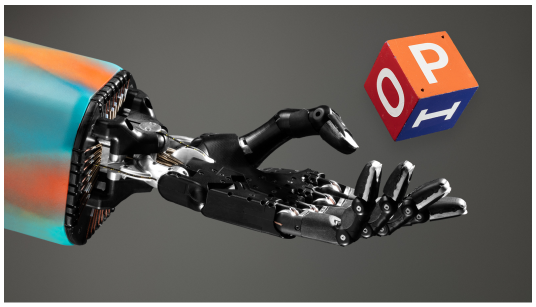
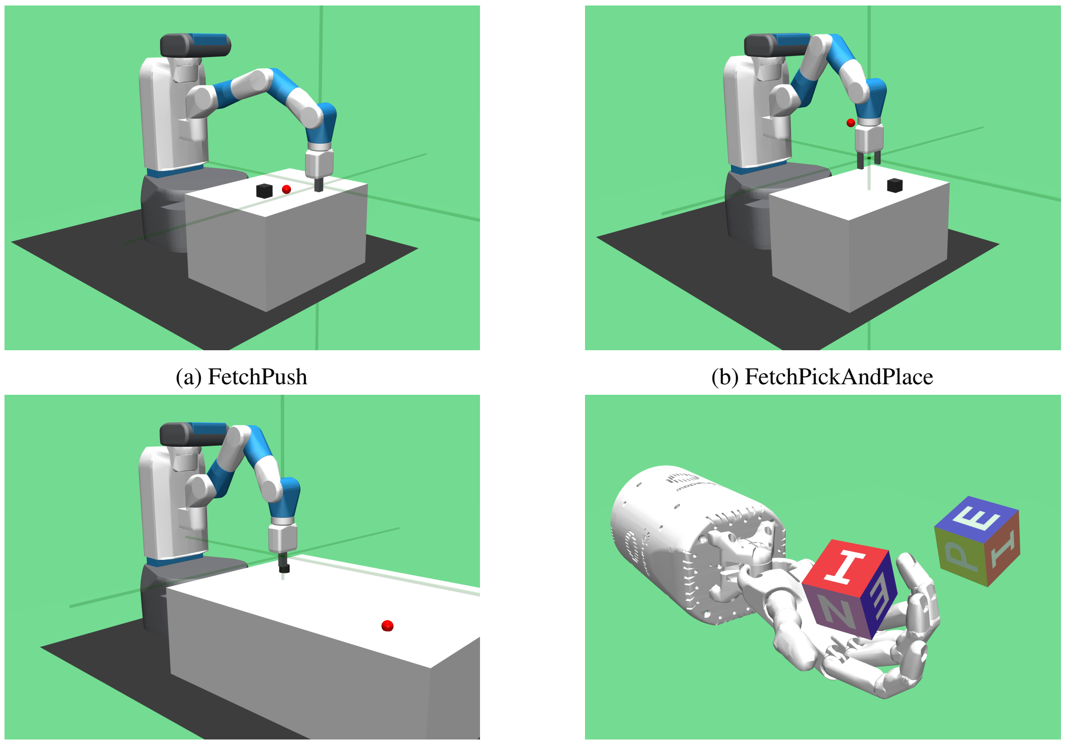

(b) ShadowHand Robot

Figure 1: Robotic platforms

Preprint. Work in progress.

We used OpenAI Gym's robotics environments (`https://gym.OpenAI.com/envs/#robotics`) for simulation and experiments. We considered four tasks in this work as shown in Figure 2. Each environment has a different task to be accomplished as described below:

- FetchPush: Move a box by pushing it until it reaches a desired goal position.
- FetchPickAndPlace: Pick up a box from table using grippers and move it to a desired goal above the table.
- FetchSlide: Slide the puck by pushing it until it reaches a desired goal position.
- HandManipulateBlock: Manipulate a block until it achieves a desired goal position and orientation.



(a) FetchPush (b) FetchPickAndPlace

(c) FetchSlide (d) HandManipulateBlock

Figure 2: Simulation Environments

2 Reinforcement Learning

Reinforcement Learning is a branch of machine learning that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. The overall framework is shown in Figure 3.

At timestep t, the agent observes the state S_t of the environment and takes an action A_t. The environment then gives a reward R_t to the agent and transitions to a new state S_{t+1}. The reward is a feedback to the agent via which it learns the optimal policy to complete the required task. The aim of the agent is to maximize the cumulative reward over a time horizon. We consider the case where the environment provides sparse rewards only, i.e., 0 on completing the task and -1 otherwise. For example, in the FetchPush task, the agent receives a reward of 0 only when it has successfully pushed the object to the desired position on the table.

The goal of the agent is to interact with the emulator by selecting actions in a way that maximises future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step, and define the future discounted *return* at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is

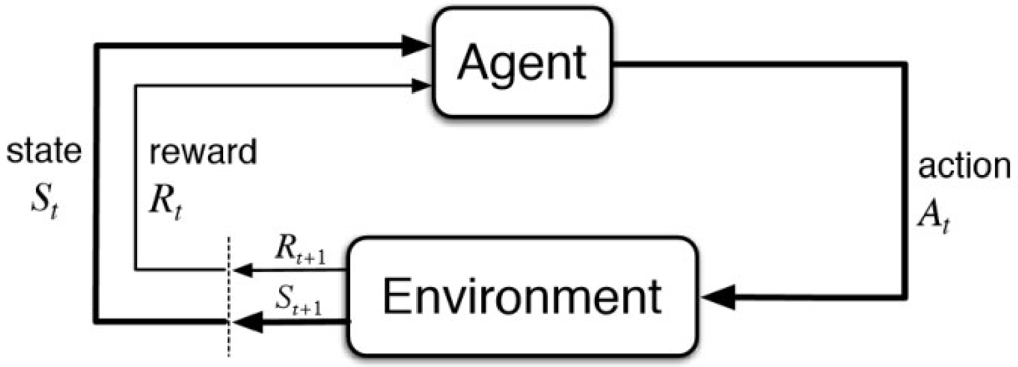


Figure 3: Reinforcement Learning

the time-step at which the episode terminates. We define the optimal action-value function $Q^*(s,a)$ as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a, $Q^*(s,a) = \max_\pi R_t|s_t = s, a_t = a, \pi$, where π is a policy mapping sequences to actions (or distributions over actions).

The optimal action-value function obeys an important identity known as the *Bellman equation*. This is based on the following intuition: if the optimal value $Q^*(s',a')$ of the sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximising the expected value of $r + \gamma Q^*(s',a')$,

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')\middle| s,a\right] \tag{1}$$

The Actor-Critic learning algorithm is used to represent the policy function independently of the value function. The policy function structure is known as the actor, and the value function structure is referred to as the critic. The actor produces an action given the current state of the environment, and the critic produces a TD (Temporal-Difference) error signal given the state and resultant reward. If the critic is estimating the action-value function $Q(s,a)$, it will also need the output of the actor. The output of the critic drives learning in both the actor and the critic. In Deep Reinforcement Learning, neural networks can be used to represent the actor and critic structures. Temporal difference (TD) learning is an approach to learning how to predict a quantity that depends on future values of a given signal. The TD error signal is excellent at compounding the variance introduced by your bad predictions over time. For the TD target y_i, the TD-error is given by,

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2}$$

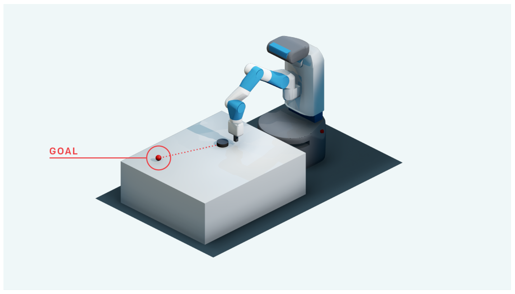
$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2) \tag{3}$$

The set of tasks considered in this work falls under the general category of multi-goal RL where the goal location or goal state is set randomly anywhere in the configuration space of the robot. For example, in the FetchPush task, the goal location can be anywhere on the table (see Figure 4a) and the Fetch robot is required to push the block there.

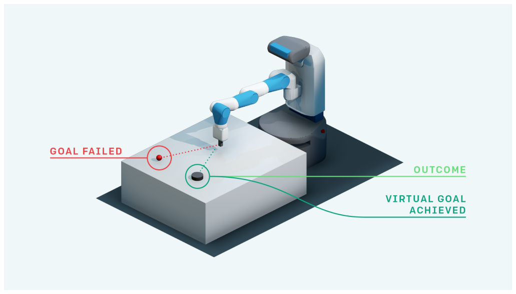
The agent gathers experience and the corresponding feedback from the environment and stores them in a replay buffer or memory. The transitions stored in the memory are sampled periodically and used to learn the policy $Q(s,a)$. Since, the environment provides only sparse rewards, the replay buffer comprises mostly of unsuccessful episodes or failure cases as in the beginning, the agent does not have any knowledge of how to complete the task and can only reach the goal state by chance or at random. Hence, there is very little positive signal or feedback for the agent to learn from. As a result, it is very difficult for the agent to master the task at hand due to lack of sufficient positive signal.

3 Learning from Failures

Andrychowicz et al. [1] proposed a framework called Hindsight Experience Replay (HER) which enables an agent to learn from its failures also. HER takes inspiration from human behavior: when attempting to reach a goal state, even though we were unsuccessful in reaching the specific goal, we have at least achieved a different one. So why not just pretend that we wanted to achieve this goal to begin with, instead of the one that we set out to achieve originally? By doing this substitution, the RL agent can obtain a learning signal since it has achieved some goal even if it wasn't the one set initially. By repeating this process, the agent eventually learns how to achieve any goal state in the entire configuration space.



(a) Task

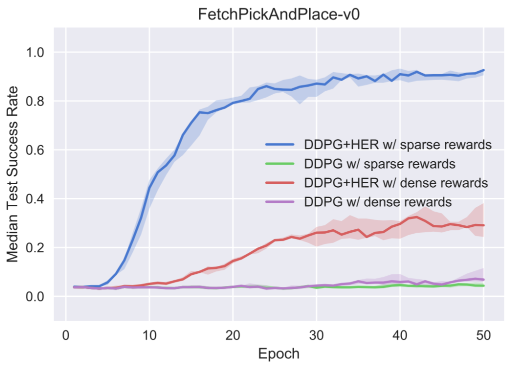


(b) Outcome

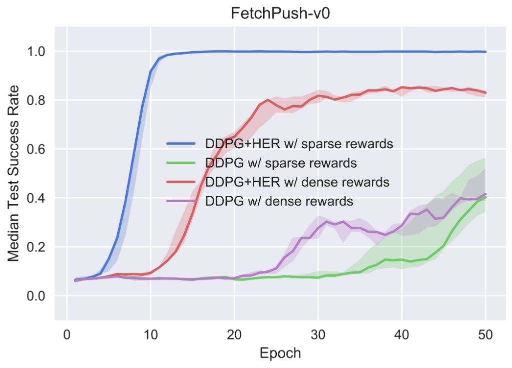
Figure 4: (a) The agent is required to slide the puck from its current location to the goal location. (b) Instead, the agent slides the puck to a different location. By treating this final location as the virtual goal, the agent gets positive learning signal and hence learns from its failures also.

HER has been shown to be a very efficient and powerful technique for solving multi-goal sparse reward RL tasks as shown in Figure 5. However, it has a few limitations:

- Uniform sampling: HER treats all experiences present in the replay memory as equal. In other words, it samples experiences uniformly from the buffer.
- Sample efficiency: It takes a long time to solve the task.



(a) FetchPickAndPlace



(b) FetchPush

Figure 5: The plots show the median test success rate against the number of training epochs. HER (shown in blue) is able to learn the policy required to solve the task whereas the simple RL (shown in green) fails to do that due to lack of positive learning signal from the environment. Image is taken from Andrychowicz et al. [1].

4 Prioritized Hindsight Experience Replay

In this work, we address the sample complexity and long training time problem of HER. We borrow the idea from the Prioritized Experience Replay framework of Schaul et al. [2] where they assign priorities to the transitions in the replay memory in proportion to model's temporal difference error

Algorithm 1 Prioritized Hindsight Experience Replay

```
1: Given:
       • an off-policy RL algorithm $\mathcal{A}$ (eg. DQN, DPPG, NAF, SDQN)
       • a strategy $\mathcal{S}$ for sampling goals for replay
       • a reward function $r : S \times A \times G \rightarrow \mathcal{R}$
2: Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$.
3: Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
4: Initialize neural networks
5: Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
6: for $t = 1$ to $T$ do
7:     Observe $S_t, R_t, \gamma_t$
8:     Store transition $(S_{t-1}, G, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
9:     Sample a set of additional goals for replay $\mathcal{G} := \mathcal{S}$(current episode)
10:    for $G' \in \mathcal{G}$ do
11:        $r' \leftarrow r(s_t, a_t, g')$
12:        Store transition $(S_{t-1}, G', A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$
13:    end for
14:    if $t \equiv 0 \mod K$ then
15:        for $j = 1$ to $k$ do
16:            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
17:            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
18:            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
19:            Update transition priority $p_j \leftarrow |\delta_j|$
20:            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
21:        end for
22:        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
23:        From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
24:    end if
25:    Choose action $A_t \sim \pi_\theta(S_t)$
26: end for
```

on them. Essentially, the transitions with high temporal difference error means the model's estimate on these samples are inaccurate and the model should focus more on them. This prioritized sampling from the buffer provides useful training signal to the agent. Formally, each transition i in the buffer is assigned a priority p_i as shown:

$$p_i = |\delta_i| + e \tag{4}$$

where, δ_i is the temporal difference error of the model and e is a small constant to ensure that all the experiences/transitions has non-zero probability of sampling. The probability of sampling is then given by :

$$P(i) = \frac{p_i^a}{\sum_k p_k^a} \tag{5}$$

where the summation is taken over all the samples in the buffer and a is a hyperparameter used to introduce some randomness in the sampling procedure. If $a = 0$, the sampling is uniform as all the transitions has an equal probability of being sampled. If $a = 1$, only experiences with high priorities are sampled. Once the transitions are sampled, they are assigned a weight $w(i)$ for importance sampling as shown:

$$w(i) = \left(\frac{1}{N}\frac{1}{P(i)}\right)^b \tag{6}$$

The hyperparameter b controls how much importance sampling affects learning. Normally, the value of b is annealed from 0 in the beginning to 1 towards the course of training as these weights are more important towards the end when the q values start to converge.

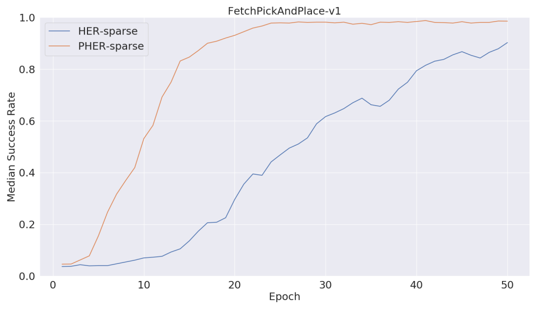
Note that in HER, we store the entire episode in the replay memory and while drawing a batch of samples for training the model, we substitute some of the them with virtual goal states reached in the episode. In other words, the replay buffer consists only of true transitions and the goal substitution is done at sampling or training time.

Our proposed framework Prioritized HER is composed of the following major steps:

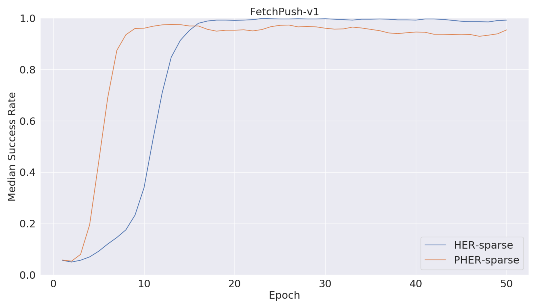
- We sample a batch of real transitions from the replay buffer as per Eq. 5.
- We replace a subset of these transitions with virtual goal or HER transitions to provide positive learning signal to the agent.
- We assign weights to the remaining real transitions as per Eq. 6. The weights of HER transitions are kept 1.
- Use the sampled batch to train the model.

The steps are detailed in Algorithm 1.

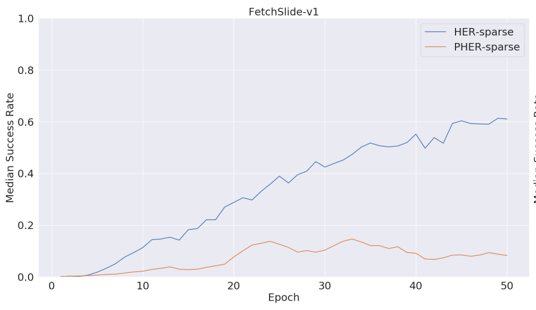
5 Experiments



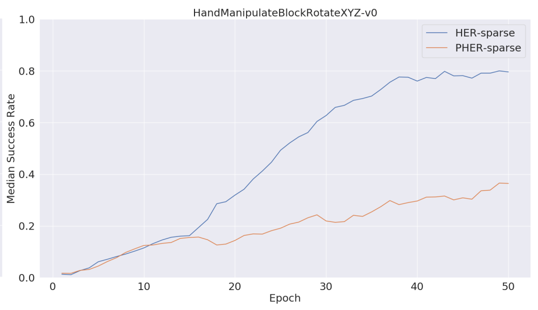
(a) FetchPickAndPlace



(b) FetchPush



(c) FetchSlide



(d) HandManipulateBlock

Figure 6: The plots show the median test success rate against the number of training epochs for HER (shown in blue) and PHER (shown in orange).

We implemented our proposed framework and compared it with the HER. Our implementation is based on OpenAI's version provided at `https://github.com/OpenAI/baselines`. The results for the all the 4 environments are shown in Figure 6.

For the FetchPickAndPlace task (see Figure 6a), we observe that our proposed PHER performs significantly better then HER as it is able to solve the task faster and also achieves better accuracy. For the FetchPush task (see Figure 6b), our model shows better sample efficiency but achieves lesser accuracy than HER at the end of training. In the other two tasks of sliding and block manipulation (see Figure 6c and 6d), PHER performs worse than HER. This happens because we have not assigned importance weights to HER transitions. As a result, when the weights of true transitions start declining, the model focuses almost entirely on these virtual experiences which is detrimental. A balance between real and virtual experiences is necessary to ensure smooth learning signal to the agent.

6 Conclusion

In this work, we proposed a novel framework Prioritized HER which combines HER and prioritized experience replay. Our model achieves better results on HER on some of the tasks however fails to do so on some others. This is because of non-weighting of HER transitions resulting creating instability in the training process. As future work, we will assign weights to HER transitions also in proportion to their temporal difference and analyze the behavior of the learned model on the same set of tasks.

References

[1] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[2] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).