HIPPO - Hindsight in Proximal Policy Optimization

Gagan Khandate, Pratyus Pati

Columbia University, New York, USA
gk2496@columbia.edu, pp2636@columbia.edu

Abstract

Goal based environments require a goal-conditional policy. It is helpful in generalizing to new goals and also critical for curriculum learning. Hindsight Experience Replay (HER) generates samples by substituting the original goal with the achieved goal. HER, applicable only to off-policy methods like DDPG, has been shown to significantly improve the sample efficiency and importantly enables learning on sparse and binary reward environments. Recent state of the art algorithms like PPO have attractive stability traits. In this work we propose an approach to introduce hindsight to PPO which we call HIPPO and show that it improves sample efficiency for learning on dense reward environments and enables learning with sparse rewards.

1. Background

In this section we introduce reinforcement learning formalism used in the paper as well as RL algorithms we use in our experiments.

1.1. Reinforcement Learning

Consider the RL problem with state space S, action space A, the reward function r, the policy $\pi(s|a)$ where $s \in S$, $a \in A$. We will use the following standard definitions (Sutton & Barto, 1998) of the expected return R_t, state value function Q_π, the value function V_π and the advantage function A_π.

$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

$V_\pi(s,a) = E_\pi [R_t|s_t = s]$

$Q_\pi(s,a) = E_\pi [R_t|s_t = s, a_t = a]$

$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s,a)$

1.2. Off-policy vs. On-policy

On-policy methods attempt to evaluate or improve the policy that is used to make decisions. For example, SARSA is an on-policy learner, as it uses only the Q-value of the action taken by the policy at the next step to estimate the current Q-value. In on-policy control methods the policy is generally soft, meaning that $\pi(s, a) > 0 \ \forall \ s \in S, a \in A$

In off-policy methods the functions of evaluating a policy and control are separated. The policy used to generate behavior, called the behavior policy, may in fact be unrelated to the policy that is used and improved, called the estimation policy. An advantage of this separation is that the estimation policy may be deterministic, while the behavior policy can continue to sample all possible actions for exploration. An off-policy learner learns the value of the optimal policy independently of the agent's actions. For example, Q-learning is an off-policy learner, as it needs to know the Q-values of all possible actions at the next step to estimate the Q-value of the current state (Sutton & Barto, 1998).

1.3. Actor Critic Methods

Actor-critic methods (Konda & Tsitsiklis, 2000) employ the use of an actor which generates action based on a particular policy, and a critic which evaluates the actor's policy. Learning involves improving both the actor and the critic based on results generated by the other. The actor learns to take better actions based on the action values provided by the critic, whereas the critic updates its parameters based on the TD error computed after taking that action. The TD error provides an estimation of the advantage function of taking a particular action at a given state, and leads to better stability.

Learning in actor-critic methods is always on-policy, as both the actor and the critic learn from the current policy being followed by the actor (Sutton & Barto, 1998).

1.4. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (Schulman et al., 2017) is a recently proposed family of policy gradient methods (Sutton et al., 2000) which alternate between sampling data through interaction with the environment, and optimizing an objective function using stochastic gradient ascent (Ruder, 2016). This is unlike the standard policy gradient methods where one gradient update per data sample is performed. The objective function enables multiple epochs of minibatch updates and retains some of the properties of Trust Region Policy Optimization (Schulman et al., 2015a) in which the policy updates are constrained to lie within a trust region. Policy gradient methods compute an estimate of the policy gradient using the gradient in stochastic gradient ascent fashion. The most common estimator has the form

$$\hat{g} = \hat{E}_t [\nabla_\theta \log \pi_\theta (a_t | s_t) \hat{A}_t] \quad (1)$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at timestep t.

It is common to make use of auto-differentiation software (Abadi et al., 2015) in implementations, in which case the gradient estimator $\hat{g}$ can be obtained by differentiating the objective

$$L^{PG} (\theta) = \hat{E}_t [\log \pi_\theta (a_t | s_t) \hat{A}_t] \quad (2)$$

Using loss L^{PG} to perform multiple steps of optimization using the same trajectory leads to destructively large policy updates causing the policy to diverge.
In the Trust Region Policy Optimization the goal is to maximize a ”surrogate” objective (3) under the constraint (4).

$$\text{maximize}_{\theta} \ \hat{E}_t \left[\frac{\pi_\theta (a_t \mid s_t)}{\pi_{\theta old} (a_t \mid s_t)} \hat{A}_t \right] \quad (3)$$

$$\text{subject to } \hat{E}_t [KL [\pi_\theta (\cdot , s_t) \| \pi_{\theta old} (\cdot , s_t)]] \quad (4)$$

where Θ_{old} is the policy parameters before the update and *KL[p,q]* is the Kullback-Liebler divergence (Kullback & Leibler, 1951) between the distributions *p* and *q*. Hence, (4) constrains the size of the policy update.

Solving the constraint optimization problem in computationally complex and PPO addresses it indirectly through a Clipped Surrogate Objective. Let $r_t(\theta)$ denote the action probability ratio between the new and old policy

$$r_t(\theta) = \frac{\pi_\theta (a_t \mid s_t)}{\pi_{\theta old} (a_t \mid s_t)} \quad (5)$$

The Conservative Policy Iteration (Kakade & Langford, 2002) objective given by (3) can be written as

$$L^{CPI} = \hat{E}_t [r_t (\theta) \hat{A}_t] \quad (6)$$

Unconstrained maximization of L^{CPI} results in very large policy updates. PPO uses a Clipped Surrogate Objective which penalizes the changes to policy that move $r_t(\theta)$ away from 1.

$$L^{CLIP} = \hat{E}_t [\min (r_t (\theta) \hat{A}_t , \text{clip} (r_t (\theta), \varepsilon) \hat{A}_t] \quad (7)$$

where

$$\text{clip} (r_t (\theta), \varepsilon) := \begin{cases} 1 - \varepsilon & \text{if } r_t (\theta) < 1 - \varepsilon \\ r_t (\theta) & \text{if } 1 - \varepsilon \leq r_t (\theta) \leq 1 + \varepsilon \\ 1 + \varepsilon & \text{if } r_t (\theta) > 1 + \varepsilon \end{cases}$$

For using value function approximation, (7) is appended with $L^{VF} = (V_\theta(s_t) - V_t^{targ})^2$. An entropy loss is also included for exploration, $S[\pi_\theta](s_t)$. The final objective function sums to

$$L_t (\theta) = \hat{E}_t [L_t^{CLIP} (\theta) - c_i L_t^{VF} + c_2 S [\pi_\theta] (s_t)] \quad (8)$$

The advantage function is estimated by Generalized Advantage Estimate (Schulman et al., 2015b),

$$\hat{A}_t = \sum_{k=0}^{T-t+1} (\gamma\lambda)^k(r_t + \gamma V(s_{t+1}) - V(s_t)) \quad (9)$$

Algorithm 1 PPO, Actor-Critic Style, (Schulman et al.,2017)

for *iteration = 1, 2, ...* ***do***

  for *actor = 1, 2, ..., N* do

    Run policy $\pi_{\theta old}$ in environment for *T* timesteps

    for *t = 1, ..., T* **do**

      $p_t := \pi_{\theta old}(a_t|s_t)$
      $v_t := V_{\theta old}(s_t)$
      $t := r(s_t,a_t)$
      Save (s_t,a_t,r_t,p_t,v_t)

    end
    Compute advantage estimates $(\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_T)^{actor}$

  end
  Optimize surrogate *L* wrt θ, with *K* epochs and a mini-batch size $M \leq NT$
  Update parameters, $\theta_{old} \leftarrow \theta$

end

PPO has been shown to perform well for continuous control problems with dense rewards, such as those involving a humanoid robot tasked with running, steering or other forms of locomotion.

It outperforms other algorithms such as A2C, A3C and TRPO for such continuous control tasks. However, its performance in these tasks is contingent on careful reward shaping. PPO hasn't been shown to perform well under conditions with sparse rewards.(Schulman et al., 2017)

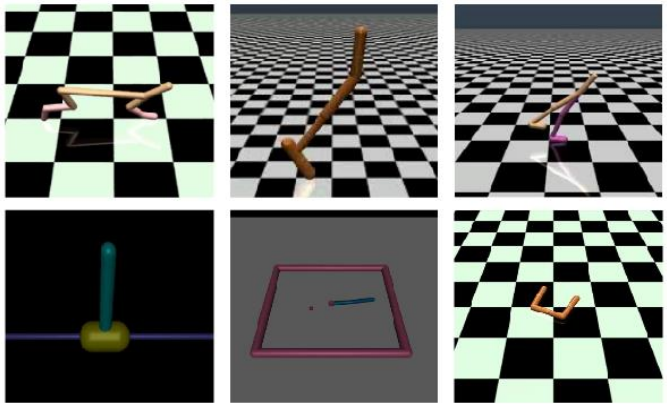

Figure 1. Environments where PPO has been shown to perform well as compared to other existing algorithms. Clockwise from top left: HalfCheetah-v1, Hopper-v1, Walker2d-v1, Swimmer-v1, Reacher-v1, InvertedDoublePendulum-v1

1.5. Goal-based Environments

Goal-based environments are characterized by the presence of a desirable state or configuration associated with higher reward as compared to all other states. The objective of a reinforcement learning agent in such environments, is then to reach the specified goal states in the fewest number of steps to maximize the rewards gained from the episode. Hence, the agent must be aware of the goal to take actions accordingly. Goal-conditional policies are therefore used, wherein, the probabilities of actions depend not just on the current state, but on the desired goal as well.

Multi-goal environments include multiple goals that the agent can achieve. In such environments, a predicate $f_g : S \rightarrow \{0, 1\}$ determines whether a particular state corresponds to a particular goal state g. Additionally, a mapping function $m : S \rightarrow G$ can be defined, which maps each state to a goal, such that $f_{m(s)}(s) = 1$. This is not always possible, but holds true for most practical environments.

1.6. Universal Value Function Approximators

Universal Value Function Approximators (Andrychowicz et al., 2017) are an extension of value function approximators for multi-goal environments. In single-goal environments, the agent's policy is trained to reach that specific goal, and the reward function $r : S \times A \rightarrow R$ depends only on the action and the state. In multi-goal environments however, there is a separate reward function $r_g : S \times A \rightarrow R$ for each goal $g \in G$. As a result, the policy is modified to consider the desired goal while taking actions, such that $\pi : S \times G \rightarrow A$. The action-value function is modified accordingly from $Q_\pi(s_t,a_t)$ to $Q_\pi(s_t,a_t,g)$. The value function is modified accordingly from $V_\pi(s_t)$ to $V_\pi(s_t, g)$.

1.7. Hindsight Experience Replay (HER)

Hindsight Experience Replay refers to the re-interpretation of episodes generated from a policy unsuccessfully trying to achieve a desired goal, as episodes where the policy was successfully able to achieve an actual goal. More formally, if the policy generated a trajectory $s_1, s_2, ...s_T$ while trying to reach goal g, and $m(s_T) = g$, we can re-interpret this as the policy

successfully reaching the desired state when g = m(s_T), and use that as an as an additional sample for training the policy. In this example, g is called the desired goal, and m(s_T) is the achieved goal. Algorithm 2 shows how HER is used with an on-policy algorithm such as DDPG (Lillicrap et al., 2015) in the original paper (Andrychowicz et al., 2017). Since HER reinterprets existing episodes to generate new virtual samples, it effectively increases the sample efficiency of the learning task.

HER+DDPG is shown to work well for multi-goal environments with sparse, binary rewards such as MuJoCo-based FetchPush, FetchSlide and FetchPickAndPlace. In comparison, DDPG without HER wasn't able to complete these tasks at all. (Andrychowicz et al., 2017)

Algorithm 2 Hindsight Experience Replay

Initialize off-policy algorithm $\mathbb{A}$
Initialize Replay Buffer R

for *episode = 1, 2, ... M* **do**

  Sample goal g and initial state s_0
  for *t = 1, 2, ..., T - 1* **do**

    $a_t \sim \pi_b(s_t||g)$
    Execute a_t and observe s_{t+1}

  end
  for *t = 1, 2, ..., T - 1* **do**

    $r_t := r(s_t, a_t, g)$
    Store $(s_t||g, a_t, r_t, s_{t+1}||g')$ in R
    Sample additional goals g from current episode
    for $g' \in G$ **do**

      $r' := r(s_t, a_t, g')$
      Store $(s_t||g', a_t, s_{t+1}||g')$ in R

    end

  end
  for *t = 1, 2, ..., N* **do**

    Sample mini-batch B from R
    Optimize $\mathbb{A}$ using B

  end

end

2. Hindsight in Proximal Policy Optimization

As discussed in the Sec (1.7), HER is crucial to learning from sparse rewards and improving sample efficiency, but this has been applied only to off-policy methods such as DQN, DDPG and others. Furthermore an optimization based on-policy method such as PPO has only been shown to work with dense - shaped rewards. We introduce hindsight to PPO through a technique we call hindsight experience justification.

Hindsight experience justification relies on the insight that the trajectory obtained by following a stochastic policy for the original goal is still a valid trajectory under the hindsight goal. The difference is in the probabilities of actions taken and rewards obtained. We use this to create a vicarious trajectory that is used as additional information in policy optimization.

Consider a trajectory defined given by tuples $(s_t, a_t, r_t)\ \forall\ t = 1, \ldots, T$ which are obtained by following a goal conditional policy $\pi(a|s, g)$ where g is the original goal. A vicarious trajectory is obtained by reliving the actions but with the hindsight goal, g'. The probability of action at under this new trajectory would change, $\pi(a_t|s_t, g) \rightarrow \pi(a_t|s_t, g')$. This re-computation of action probabilities which is a form of importance sampling (Rauber et al., 2017) is the primary difference between our approach and HER for off-policy methods. For actor-critic methods on goal-based environments the value is obtained from a universal value function approximator that is conditioned on the goal. The state values in the new trajectory change, $V(s_t, g) \rightarrow V(s_t, g')$. This approach is strictly for soft policies which are common for on-policy methods. A soft-policy requires that $\pi(s, a) > 0\ \forall\ s \in S, a \in A$.

Algorithm 3 introduces a new approach called HIPPO, that uses hindsight experience justification to generate a new vicarious trajectory that is used alongside the original trajectory to optimize the policy by iterating over this batch of data over a number of epochs. HIPPO enables learning from sparse rewards and also improves sample efficiency in case of dense rewards. The results of this approach on FetchReach-v1 are presented in the results section.

The hindsight goal can be obtained in multiple ways. The naive approach would be to use the goal mapped to by the terminal state of the trajectory, $g' := s_T$. This fails when the trajectory given is suboptimal. Hence, the hindsight goal has to be in the near future (Andrychowicz et al., 2017), which can be achieved by frequently updating the hindsight goal. HIPPO introduces a hindsight timestep horizon parameter, h that essentially controls how frequently the hindsight goal is updated within the trajectory.

Algorithm 3 HIPPO

for *iteration = 1, 2, …* **do**

  for *actor = 1, 2, … N* **do**

    Run policy $\pi_{\theta \text{old}}$ in environment for T timesteps
    for *t = 1, …, T* **do**

      $p_t := \pi_{\theta \text{old}}(a_t|s_t, g)$
      $v_t := V_{\theta \text{old}}(s_t, g)$
      $r_t := r(s_t, a_t, g)$
      $R_{actor}(t) := (s_t, a_t, r_t, p_t, v_t)$

    end
    Compute advantage estimates $(\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_T)^{\text{actor}}$
    Generate hindsight from *R*
    for *t = T, T-1, …, 1* **do**

      $h \leq T$, hindsight timestep horizon
      if $t = kh, k \in \mathbb{Z}^+$ or $t = T$ **then**

        Update hindsight goal, $g' := m(s_t)$

      end
      $p_t := \pi_{\theta \text{old}}(a_t|s_t, g')$
      $v_t := V_{\theta \text{old}}(s_t, g')$
      $r_t := r(s_t, a_t, g')$
      $H(t) := (s_t, a_t, r_t, p_t, v_t)$

    end
    Compute advantage estimates
    $(\hat{A}_1^h, \ldots, \hat{A}_T^h)^{\text{actor}}$ from *H*

  end
  Optimize surrogate *L* wrt θ, with *K* epochs and a mini-batch of $R||H$ of size $M \leq 2NT$
  Update parameters, $\theta_{old} \leftarrow \theta$

end

HIPPO can be extended to generate new trajectories from trajectories obtained by following older policies. This is shown in Algorithm 4. A new parameter that determines how many trajectories from the immediate past are to be used for generating new trajectories through hindsight experience justification. This parameter is denoted by n is called hindsight iterations horizon. This parameter also affects the number of samples available at the optimization step. Also, trajectories from only the immediate past are considered as the far older trajectories may not be valuable as the current policy may have gone far from the trajectory responsible for generating the old trajectory. This can result in near 0 values for $\pi(a|s)$ which may in-turn cause numerical issues.

Algorithm 4 HIPPO, Extended Replay

Initialize buffer R
for *iteration = 1, 2, …* **do**
  for *actor = 1, 2, … N* **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    for $t = 1, \ldots, T$ **do**
      $p_t := \pi_{\theta_{\text{old}}}(a_t|s_t, g)$
      $v_t := V_{\theta_{\text{old}}}(s_t, g)$
      $r_t := r(s_t, a_t, g)$
      $R_{actor}^{iteration}(t) := (s_t, a_t, r_t, p_t, v_t)$
    end
    Compute advantage estimates $(\hat{A}_1^0, \hat{A}_2^0, \ldots, \hat{A}_T^0)^{\text{actor}}$
  end
  Let n be hindsight iterations horizon
  for $i = \max(iteration - n, 1), \ldots, iteration$ **do**
    for *actor = 1, 2, …, N* **do**
      Generate hindsight from $R_{actor}^{iteration}$
      Initialize buffer H
      for $t = T, T\text{-}1, \ldots, 1$ **do**
        Let $h \leq T$ be hindsight timesteps horizon
        if $t = kh, k \in \mathbb{Z}^+$ or $t = T$ **then**
          Update hindsight goal, $g' := m(s_t)$
        end

				$p_t := \pi_{\theta old}(a_t \mid s_t, g')$ $v_t := V_{\theta old}(s_t, g')$ $r_t := r(s_t, a_t, g')$ $H_i(t) := (s_t, a_t, r_t, p_t, v_t)$
			end Compute advantage estimates $(\hat{A}_1^h, \ldots, \hat{A}_T^h)^{actor}$ from H_i	
		end		
	end Optimize surrogate L wrt θ, with K epochs and a mini-batch of $R \mid\mid H$ of size $M \leq (n+1)NT$ Update parameters, $\theta_{old} \leftarrow \theta$			
end				

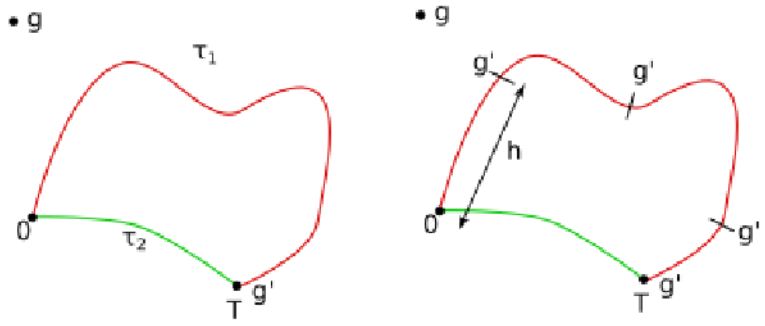

Figure 2. Illustration of generating hindsight goals by reinterpreting episodes. Trajectory τ_1 is suboptimal and using g′ as shown on the left is the naive approach which is fine for close to optimal trajectory like τ_2. To improve the performance the hindsight goal is frequently updated as shown on the right. A shorter trajectory has a higher likelihood of being optimal. The hindsight goal is updated every h timesteps.

2.1. Environment

We are using OpenAI Gym's MuJoCo-based Robotics environments (Brockman et al., 2016), (Todorov et al., 2012) for testing. Specifically, we use the FetchReach-v1 environment. The goal of this environment is to learn a policy that drives the end-effector of the Fetch arm to goal position in 3D. When the reward type is set to dense, the rewards obtained are simply the negative of the distance of the end-effector from the goal position. When the reward type is sparse, the agent receives a reward of 0 if the goal position is reached within a tolerance and -1 otherwise.

3. Results & Discussion

We evaluated HIPPO on the FetchReach-v1 as described in the previous section. The approach we followed was to first arrive at the optimal parameters for PPO and then evaluate HIPPO at the same parameters so that there is fair comparison. The results shown are averaged over multiple seeds for about 6 runs each. The episode length of the FetchReach-v1 environment is 50 steps. The parameters we used for PPO are as follows.

- number of time steps, T : 2048
- discount factor, γ : 0.99
- GAE parameter, λ : 0.95
- clip range, ϵ : 0.2
- entropy coefficient, c : 0.0
- minibatch size m : 32
- number of epochs, K : 10
- learning rate, α : 3.0 × 10-4
- number of actors, N : 1

The learning curves for PPO and HIPPO for both dense and sparse rewards are shown in Figure (4). The plots show the mean episode reward against the number of time steps of interaction with the environment. In case of dense rewards, it can be clearly seen that the HIPPO learns faster and also achieves better rewards. HIPPO achieves the same performance as PPO approximately 3 to 4 times faster and exceeds the performance thereon. Also, the learning curve exhibits lesser variance with HIPPO as compared to PPO.

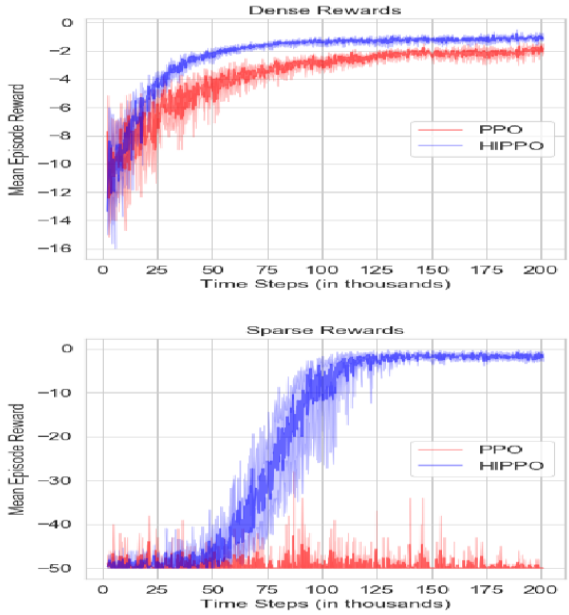

Figure 3. Comparison of mean episode rewards using the FetchReach-v1 environment. Data was aggregated from multiple runs of the experiment for each algorithm and reward-type pair. The shaded colors indicate the range of variance observed. The minimum possible reward from an episode is -50 as there are 50 steps in an episode, with -1 being the least possible reward obtainable from each step.

The learning curves for sparse rewards cases are much more interesting. Clearly, from the flat curve, PPO is unable to learn whereas HIPPO is consistently able to learn from sparse rewards. Although the learning is slow compared to the dense case, it is acceptable given that the sparse rewards are much more challenging and learning to achieve the goal is an accomplishment in itself.

Apart from the FetchReach-v1 we also tried learning a policy through PPO or HIPPO for another environment in OpenAI Robotics environments called HandReach-v0. In this task, the goal is to make the high dof fingered hand to reach a goal configuration. Both PPO and HIPPO failed to learn in this environment in both dense and sparse rewards even, for a large number of iterations (5M). We believe that a simple feed forward policy is not sufficient for such tasks and also that due to the complexity, HIPPO with Extended Replay might be more appropriate for this task. The results for this are not available as the implementation for HIPPO with Extended Replay is yet to be completed.

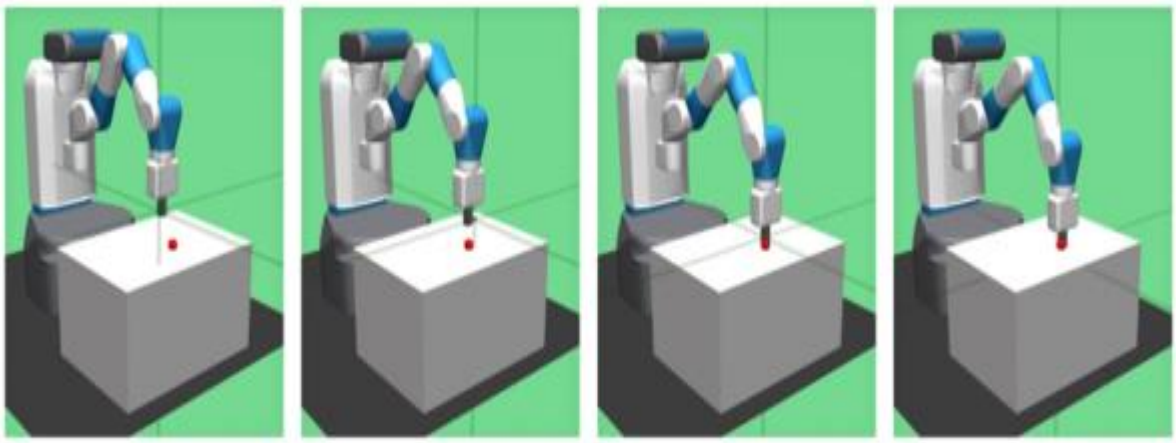

Figure 4. Frames from testing a trained robotic arm in the FetchReach-v1 environment. Sparse rewards are rewarded when the robotic end-effector is within a certain proximity of the red ball. Dense rewards are defined as the distance between the end-effector and the red ball.

4. Conclusion

In this work we take ideas from Hindsight Experience Replay and apply it to a on-policy method - PPO. We use a technique we call hindsight experience justification to generate new vicarious trajectories and use them alongside the actual trajectories for obtaining the optimal policy. HIPPO is a first cut implementation of this approach and it performs significantly better than PPO in case of dense rewards and more importantly enables learning from sparse rewards just as in case of off-policy HER.

We also propose a full-fledged version - HIPPO, Extended Replay that we hope will be able to learn in more complex environments.

5. Acknowledgement

We would like to thank Prof. Chong Li[1] for providing us with strong foundations in reinforcement learning and Prof. Lei Zhang[2] for bringing us up to speed on advanced topics in deep reinforcement learning. We would also like to thank OpenAI implementations of all baseline algorithms in DeepRL (Dhariwal et al., 2017) and numerous environments on which to learn on. Also, thanks to the creators of stable baselines (Hill et al., 2018) repository for easy to understand re implementation of OpenAI baselines. We would also like to thank Prof. Matei Ciocarlie[3] for providing the opportunity to be part of his research group which indirectly helped us in coming up with this project topic.

References

[1] Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dandelion, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon,Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, **2015**. URL https://www.tensorflow.org/ Software available from tensorflow.org

[2] Andrychowicz, Marcin, Wolski, Filip, Ray, Alex, Schneider, Jonas, Fong, Rachel, Welinder, Peter, McGrew, Bob, Tobin, Josh, Abbeel, Pieter, and Zaremba, Wojciech. Hindsight experience replay. CoRR, abs/1707.01495, **2017**. URL http://arxiv.org/abs/1707.01495

[3] Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. CoRR, abs/1606.01540, **2016**. URL http://arxiv.org/abs/1606.01540

[4] Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, Wu, Yuhuai, and Zhokhov, Peter. Openai baselines. https://github.com/openai/baselines **2017**.

[5] Hill, Ashley, Raffin, Antonin, Ernestus, Maximilian, Traore, Rene, Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai. Stable baselines. https://github.com/hill-a/stable-baseline , **2018**

[6] Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In Proceedings of the Nineteenth International Conference on Machine Learning, ICML '02, pp. 267-274, San Francisco, CA, USA, **2002**. Morgan

Kaufmann Publishers Inc. ISBN 1-55860-873-7. URL http://dl.acm.org/citation.cfm?id=645531.656005

[7] Konda, Vijay R. and Tsitsiklis, John N. Actor-critic algorithms. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), Advances in Neural Information Processing Systems 12, pp. 1008-1014. MIT Press, **2000**. URL http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf

[8] Kullback, S. and Leibler, R. A. On information and sufficiency. Ann. Math. Statist., 22(1):79-86, 1951. Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. CoRR, abs/1509.02971, **2015**. URL http://arxiv.org/abs/1509.02971

[9] Rauber, Paulo, Mutz, Filipe, and Schmidhuber, Jürgen. Hindsight policy gradients. CoRR, abs/1711.06006, **2017**. URL http://arxiv.org/abs/1711.06006

[10] Ruder, Sebastian. An overview of gradient descent optimization algorithms. CoRR, abs/1609.04747, **2016**. URL http://arxiv.org/abs/1609.04747

[11] Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I., and Abbeel, Pieter. Trust region policy optimization. CoRR, abs/1502.05477, **2015**a. URL http://arxiv.org/abs/1502.05477

[12] Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael I., and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. CoRR, abs/1506.02438, **2015**b. URL http://arxiv.org/abs/1506.02438

[13] Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. CoRR, abs/1707.06347, **2017**. URL http://arxiv.org/abs/1707.06347

[14] Sutton, Richard S. and Barto, Andrew G. Reinforcement learning - an introduction. Adaptive computation and machine learning. MIT Press, **1998**. ISBN 0262193981. URL http://www.worldcat.org/oclc/37293240

[15] Sutton, Richard S, McAllester, David A., Singh, Satinder P., and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), Advances in Neural Information Processing Systems 12, pp. 1057-1063. MIT Press, **2000**. URL http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning.pdf

[16] Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. InIROS, pp. 5026-5033. IEEE, **2012**. ISBN 978-1-4673-1737-5. URL http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12