ABSTRACT

LANKA, SAMEERA. Weighted Rewards for Hindsight Experience Replay. (Under the direction of Dr. Tianfu Wu).

Deep reinforcement learning(RL) for robotics currently faces a challenge in scalability due to poor sample efficiency of deep RL algorithms owing to sparse rewards in continuous control. Since most episodes encountered during training are likely to be unsuccessful (unless the algorithm is primed with domain-specific priors), an RL agent requires a restrictively large amount of data for effective exploration to determine which policies generate successful transitions. Hindsight experience replay(HER) enables an agent to learn from failure using the idea that an unsuccessful episode has achieved *some* goal. So assuming the achieved goal was specified as a target, the experienced episode, in hindsight, is considered successful.

In this text, we analyse how a naive deployment of HER could introduce an optimistic counterfactual bias and present a simple method which reduces this bias without any increase in computational complexity. We build upon HER by proposing a novel weighted reward mechanism which improves the sample-efficiency of HER by assigning a proportionally larger influence to rewards collected during hindsight replay and a smaller influence to rewards collected during the real episode. We show that our method, titled "ARCHER: Aggressive Rewards to Counter bias in Hindsight Experience Replay", achieves a higher success in lesser time across a range of reward functions and task complexities, proving that it is a beneficial algorithm to achieve good sample-efficiency.

Weighted Rewards for Hindsight Experience Replay

by
Sameera Lanka

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina
2018

APPROVED BY:

Dr. Edgar Lobaton

Dr. Xu Xu

Dr. Tianfu Wu
Chair of Advisory Committee

DEDICATION

To my parents, my sister, my mentors and myself.

BIOGRAPHY

Sameera Lanka is an M.S. student in the Department of Electrical and Computer Engineering at North Carolina State University. She pursued her Masters Thesis under the guidance of Dr. Tianfu Wu, where she conducted research in deep reinforcement learning and robotics. Prior to joining NCSU, Sameera graduated with a B.Tech from National Institute of Technology Karnataka, India.

ACKNOWLEDGEMENTS

I first and foremost extend my gratitude to my advisor, Dr. Tianfu Wu. The lessons I have learnt over the course of working with Dr. Wu have helped me become a better student, researcher and collaborator. In addition to his patience and support, Dr. Wu encouraged me to cultivate curiosity and think creatively. His emphasis on treating cognitive science and fundamental mathematics as the primary pillars of AI has given me a valuable respect for first principles which I will carry with me throughout my career. I thank Dr. Wu for the time, resources and knowledge he has imparted and I cannot overstate his role in the making of this thesis.

I thank my committee members Dr. Xu Xu and Dr. Edgar Lobaton for their generous support and time in supervising my thesis. I also convey my gratitude to Dr. Lobaton for his mentorship during my time at ARoS Laboratory. His support was a source of motivation for me to pursue research.

I thank my fellow labmates at the Visual Narrative Initiative for creating an optimistic and spirited environment filled with engaging ideas and an enthusiasm for learning.

I am grateful for the open-access to textbooks, online courses, research papers and software which has enabled me to learn and experiment with algorithms at the frontiers of current deep learning research.

Finally, I thank my parents and my sister for all the love and support and for always believing in me.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Deep reinforcement learning (RL) stands as a promising method for training robots to autonomously learn dynamic control policies [3, 4]. The key advantage of deep RL lies in its capacity to learn generalized policy representations in comparison to specialized domain and task specific hand-engineered policies. However, this advantage comes with a caveat that deep RL requires a restrictive amount of training data and computational time to learn optimal policies in high-dimensional, continuous state and continuous action domains, thus limiting its application in robotics.

Deep RL methods are commonly used in conjunction with a binary success/failure reward function. Such a reward mechanism allows for the agent to discover optimal policies intended for successful completion of the task, thus eliminating the need for expert reward specification or the danger of hacking inaccurate reward design [5, 6]. In high-dimensional continuous control, a characteristic property of most practical robots, a binary reward system presents a challenge when the task is complex and goal states are rarely encountered. In these sparse reward con-

ditions, an agent receives insufficient reinforcement signals to discriminate between successful and unsuccessful actions, and is consequently unable to converge to an effective policy to make progress at the task. The sparse-reward problem is particularly intensified when prolonged operation of a physical robot to explore and collect data for training is infeasible, dangerous or expensive. Thus, there exists a pressing demand to improve the sample efficiency of deep RL algorithms, to favorably transfer their theoretical promises into the real world.

If an agent was tasked to achieve a specified goal A but was unsuccessful, and instead terminated in a different state, traditional RL methods consider episode as a failure and do not gain any information from this episode. Hindsight Experience Replay(HER) [2], presents a solution to leverage the information gained from these failed episodes as follows - in the first run, the agent stores the transitions and rewards associated with the real episode pertaining to goal A. The algorithm then *replays* the episode by recomputing the rewards in hindsight, under the context of achieving goal B and thereby incorporates information from an unsuccessful episode to train its policy function.

In this thesis, we investigate further into HER and explain why a naive implementation of such a method would lead to a biased learning model, thereby achieving only a suboptimal increase in sample-efficiency. We challenge the central assumption that the same trajectory of states and actions can be replayed in hindsight, thereby creating real and hindsight experiences which impact the learning process *equivalently*, through their corresponding rewards. We also present a simple and effective solution to overcome the bias at no additional computational complexity. We regard our method as a step towards understanding the computation of counterfactual trajectories in reinforcement learning.

1.2 Thesis contributions and structure

The contributions of in thesis are as follows :-

1. We characterize the source and nature of bias in Hindsight Experience Replay.

2. We propose a weighted rewards mechanism to restrict the bias, thereby greatly increasing sample-efficiency of RL algorithms for continuous control.

3. We empirically demonstrate the success of our in simulation robotic control environments along with several ablation studies.

The thesis is organized according to the following structure :-

1. Chapter 2 provides a summary of concepts and algorithms from deep reinforcement learning which form the foundation for the contributions of this thesis.

2. Chapter 3 describes our weighted rewards mechanism along with experimental analyses and discussions.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning(RL) is a scientific discipline at the intersection of machine learning, optimal control, statistics, neuroscience and behavioral psychology, which pertains to the computational learning of the optimal actions to execute in a sequential decision process so as to maximize a numerical reward signal. An RL problem is mathematically formulated as a Markov Decision Process(MDP). An MDP consists of the following components -

1. State space $\mathcal{S}$ - Set of all possible states of the environment
2. Action space $\mathcal{A}$ - Set of all possible actions by the agent
3. Transition model $\mathcal{T}^{A}_{SS'}$ - Matrix of state transition probabilities,
   $\mathbb{P}[S_{t+1} = S' | S_t = S, A_t = A]$
4. Reward function $\mathcal{R}(S, A, S')$ - Scalar feedback signal

If the environment state is not directly accessible, we operate under the framework of Partially

Observable Markov Decision Processes (POMDPs), where the agent instead has access to an *observation*, which is a function of the latent state. However, for this thesis, we restrict ourselves to fully observable state spaces.

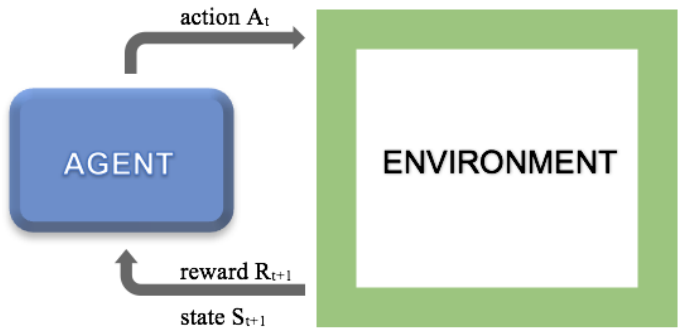


Figure 2.1: Schematic representation of a Markov Decision Process

In reinforcement learning, an agent interacts with an external environment with the purpose of achieving a *goal*. The agent learns through numerous independent interactions, which are termed as *episodes*. Every episode consists of a sequence of discrete time-steps $t = \{0, 1, 2, ...T\}$, where T is the terminal step of each episode. T assumes a finite value in terminating MDPs and is ∞ in the case of non-terminating MDPs.

At each time step t, the agent samples an action A_t from its *policy function*, a conditional probability distribution over actions given the current environment state S_t. The policy of the RL agent, denoted by $\pi(a|s)$ may a deterministic or stochastic probabilistic mapping from states to actions.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{2.1}$$

The action A_t selected by the agent drives the environment into its new state S_{t+1}, which the environment provides to the agent along with a reward R_t associated with this transition. Thus each time step yields an *experience tuple* of these four elements $\{S_t, A_t, R_t, S_{t+1}\}$.

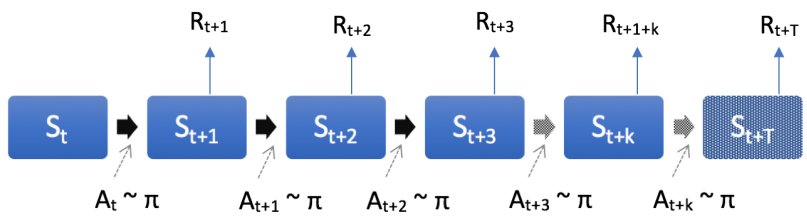


Figure 2.2: Schematic representation of an episode rollout

The rewards associated with a task are structured such that the sequence of actions which achieves the goal correspondingly generates the maximum possible total reward in each episode. Hence, the agent should take into consideration not only the immediate reward, but also the entire sequence of future rewards it receives along an episode. We thus state that the objective of an agent is to maximize its *return* at each time step, which is the cumulative sum of future rewards. To incorporate uncertainty about future rewards, to encourage expediency, and for mathematical convenience, future rewards are often multiplied by a *discount factor* $\gamma(0 \leq \gamma \leq 1)$. The discount factor regulates the present value of future rewards; a high discount factor lends little influence to later rewards while a low discount factor distributes a greater influence to future rewards. Therefore, the policy is optimized to have the agent select actions which maximize the total sum of discounted future rewards G_t -

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^k R_{t+1+k} + \ldots + \gamma^{T-1} R_{t+T} \tag{2.2}$$

$$= \sum_{k=0}^{T-1} \gamma^k R_{t+1+k} \tag{2.3}$$

Reinforcement learning methods are broadly categorized into two fields -
1) Value Function Optimization or 2) Policy Function Optimization

2.1.1 Value Function Optimization

Value-based reinforcement learning utilizes the construction of intermediate value functions to define a partial ordering over policies. The two stages of value function optimization are policy evaluation and policy improvement.

Policy Evaluation

We formally define *value functions* $v_\pi(s) : \mathcal{S} \to \mathbb{R}$ and $q_\pi(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, as relations which map to the expected return under a policy π. The state-value $v_\pi(s)$ is the expected return starting from any state s and following π. In other words, if an agent encounters a state s, the value of that state describes the return the agent can expect on average, if the agent chooses actions according to the policy π.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] && (2.4) \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k R_{t+1+k} \middle| S_t = s\right], \text{ for all } s \in \mathcal{S} && (2.5) \end{aligned}$$

Policy evaluation is the iterative process of computing the value functions for a given π. We can rewrite Eq. (2.5) as follows

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k R_{t+1+k} \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \sum_{k=1}^{T-1} \gamma^k R_{t+1+k} \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{T-1} \gamma^k R_{(t+1)+1+k} \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] && (2.6) \end{aligned}$$

Eq. (2.6) is known as the *Bellman expectation equation* for the state-value function. It expresses the value of the current state as the expectation over sum of the immediate reward and the discounted value of the next state. This representation helps us approach value estimation as a form of dynamic programming.

Similarly, we define the action-value $q_\pi(s, a)$ as the expected return from starting in state s, taking action a and thereafter following policy π

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] && (2.7) \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{T-1} \gamma^k R_{t+1+k} \middle| S_t = s, A_t = a \right], \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} && (2.8) \end{aligned}$$

The Bellman expectation equation for the action-value function is

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \right] \tag{2.9}$$

Policy Improvement

Policy improvement is the process of updating the current policy function in order to maximize a numerical performance metric. In this work, we utilize *greedy policy improvement*, where the new policy π' is constructed by choosing at each state actions which have the largest action-values by assigning non-zero probability to these actions, i.e.

$$\pi'(a|s) = \begin{cases} 1, \text{ if } a = \text{argmax}_a \, q_\pi(s,a) \\ 0, \text{ otherwise} \end{cases} \tag{2.10}$$

Sometimes a small probability, ϵ is assigned to non-maximal action to encourage exploration. This is known as ϵ-greedy policy improvement.

Policy Iteration

A policy is known as the *optimal policy* π^* if it follows that $v_{\pi^*} \geq v_\pi$, for every possible policy π. In other words, an optimal policy generates the greatest expected return in all states. There may be multiple optimal policies, in which case they share the same value functions, namely, the *optimal state-value function* $v_{\pi^*}(s)$ and *optimal action-value function* -

$$v_{\pi^*}(s) = \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S} \tag{2.11}$$

$$q_{\pi^*}(s,a) = \max_\pi q_\pi(s,a), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{2.12}$$

The *Bellman optimality equation* (2.13) equates the value of a state under the optimal policy to the action-value obtained from choosing the action which yields the maximum expected return in that state and hence connects the optimal state-value function and optimal action-value function as follows -

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} q_{\pi^*}(s,a) \tag{2.13}$$

$$= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma v_{\pi^*}(s_{t+1}) | S_t = s, A_t = a\right] \text{(from (2.9))} \tag{2.14}$$

Similarly, the Bellman optimality equation for action-value function can be written as

$$q_{\pi^*}(s,a) = \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \max_{a'} q_{\pi^*}(S_{t+1}, a') \middle| S_t = s, A_t = a\right] \tag{2.15}$$

Policy iteration is the iterative optimization of the policy function towards the optimal policy, by alternating between policy evaluation and policy improvement. The agent learns the optimal policy for a task through trial and error. At the start of training, the agent begins with an arbitrary initial policy π_0 and interacts with the environment. Episode trajectories are sampled from the environment dynamics and the sample rollouts are used to compute empirical expectations of the state and action value functions, in the policy evaluation stage. Next, the

agent maximizes its policy function across these values to give rise to a new policy π_1, during policy improvement. The agent recalculates the value functions for π_1 and this cycle continues - at each step evaluating policy π_k and then optimizing to get π_{k+1}. The cycle repeats until convergence. At convergence, the value function is consistent with the current policy and the policy corresponds to the maximally optimal policy over the current value function. Thus policy iteration converges to the optimal policy and the optimal value functions.

2.1.2 Policy Gradients

Policy gradient methods perform stochastic gradient-based optimization directly on a parameterized policy function to learn the optimal policy. The policy function, parameterized by weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ represented as

$$\pi(a|s, \boldsymbol{\theta}) = \mathbb{P}[A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}] \tag{2.16}$$

The objective function of these algorithms is a performance measure (such as state value of initial state, mean Q-value, Monte-Carlo return etc.) which is a function of the policy parameter $\eta(\boldsymbol{\theta})$. Policy gradient methods seek to maximize this objective function through a *gradient ascent* update on $\boldsymbol{\theta}$:-

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \nabla\eta(\boldsymbol{\theta}_t) \tag{2.17}$$

Let us assume the performance measure we wish to maximize is the Q-value function. So $\eta(\boldsymbol{\theta}) = q_{\pi_{\boldsymbol{\theta}}}(s, a)$. Using the *policy gradient theorem* we can express the gradient as -

$$
\begin{aligned}
\nabla\eta(\boldsymbol{\theta}_t) &= \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) \\
&= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta}) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t,A_t) \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta}) \right] \qquad (2.18)
\end{aligned}
$$

Therefore the policy gradient in Eq. (2.18) can be computed from sampled episodes as sample expectation of the expression on the right hand side and can be used to update the policy weights $\boldsymbol{\theta}$ using Eq. (2.17). The update increases the log probability of action A_t in proportion to the corresponding Q-value or the chosen performance objective.

2.2 Deep Deterministic Policy Gradient

Deep deterministic policy gradient(DDPG)[3] is an off-policy model-free reinforcement learning algorithm for deep Q-learning in continuous control environments. The DDPG architecture consists of two neural networks - *actor network* and *critic network*. The actor, parameterized by weights θ^μ, serves as a deterministic policy function denoted by $\mu(s|\theta^\mu) : \mathcal{S} \rightarrow \mathcal{A}$. It specifies a deterministic mapping from states to actions, i.e. the input to the actor network is a state and the output is the action to be executed for that state ; a = $\mu(s|\theta^\mu)$. The critic network, parameterized by weights θ^Q evaluates the action-value $Q(s,a|\theta^Q)$ of state-action pairs, which is used as the performance objective to be maximized by the actor.

Consider a transition consisting of the present state s_t, action sampled from the actor network $a_t = \mu(s_t)$, reward r_{t+1} and next state s_{t+1}. From Eq. (2.15), we construct the target

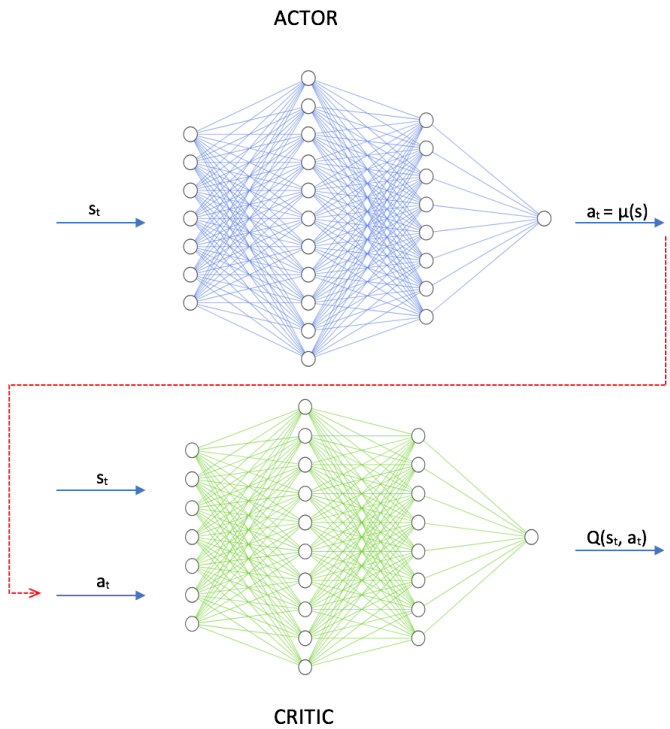


Figure 2.3: Actor-Critic architecture in DDPG

Q-values for the critic using the Bellman optimality criterion -

$$y_t = r_{t+1} + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \tag{2.19}$$

Hence the critic is trained by minimizing the mean-squared error loss $L(\theta^Q)$ between the predicted Q-value of the state-action pair and its corresponding target :-

$$L(\theta^Q) = \mathbb{E}\left[(Q(s_t, a_t|\theta^Q) - y_t)^2\right] \tag{2.20}$$

The actor network is optimized using policy gradients where the objective of the actor is to maximize the Q-value predicted by the critic. The derivative of the Q-value with respect to parameters of the actor is given by the *deterministic policy gradient*(DPG)[7] algorithm. The

actor parameters are updated using the following performance gradient :-

$$\nabla\eta(\theta^Q) = \mathbb{E}\left[\nabla_a Q(s,a|\theta^Q)\big|_{s=s_t,a=\mu(s_t)}\nabla_{\theta_\mu}\mu(s|\theta^\mu)\big|_{s=s_t}\right] \tag{2.21}$$

From Eq. (2.17) we get

$$\theta^Q_{t+1} = \theta^Q_t + \mathbb{E}\left[\nabla_a Q(s,a|\theta^Q)\big|_{s=s_t,a=\mu(s_t)}\nabla_{\theta_\mu}\mu(s|\theta^\mu)\big|_{s=s_t}\right] \tag{2.22}$$

To stabilize the training process of the actor and critic networks, the algorithm also makes use of *experience replay* and *target networks*, similar to their role in Deep Q-Networks [8].

Experience Replay

Experience replay[9] is a technique which allows an agent to learn from experiences collected from previous versions of its policy (learning from data generated under a different policy is termed as off-policy reinforcement learning). At every time step t, an *experience* tuple e_t is recorded comprising the current state, current action, next state and reward associated with this transition.

$$e_t = (s_t, a_t, r_t, s_{t+1}) \tag{2.23}$$

The experiences are appended to a *replay memory* $\mathcal{R}$ which forms the training data-set. Mini-batches are sampled from the replay buffer at random during for the actor and critic updates. The replay buffer is of finite size and once it is completely filled, old experiences are replaced by new data. Thus, the buffer gradually shifts to experiences consistent with the most recent policies. The advantages of using experience replay are:

1. Each experience may appear in multiple weight updates which leads to greater data

efficiency.

2. Uniform sampling breaks the temporal correlations between observed state-action pairs in a episode which allows the training data to be independent and identically distributed (i.i.d)

Target Networks

As the actor and critic networks undergo continuous optimization, the Q-learning target in the right hand side of Eq. (2.19) derived from these networks, also fluctuates constantly. This non-stationarity destabilizes the training process and inhibits convergence to the optimal policy. To provide stability, the target values in Eq. (2.19) are instead calculated from copies of the actor and critic network called *target actor* $\mu(a|s, \theta^{\mu'})$. The target networks are not optimized by gradient descent. Their weights are gradually adjusted towards the weights of real actor and critic networks through a soft weighted update with $\tau \ll 1$.

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{2.24}$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{2.25}$$

The DDPG algorithm is presented in its entirety in Fig. 2.4.

2.3 Hindsight Experience Replay

Consider the example of parents teaching their kids to play simple block stacking games. In the early stages, parents usually provide positive feedback even when kids do not build the correct configuration. By receiving such signals, kids stay encouraged and cultivate an exploratory approach, and over time refine their skill through memory and corrective feedback. This is the

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 2.4: DDPG Algorithm (Lillicrap, et al.)[1]

central intuition underlying the concept of Hindsight Experience Replay(HER) [2] which derives from the principle that an agent should learn from unsuccessful episodes, just as people are able to learn from failures and mistakes to make progress at a given skill or task.

HER presents a mathematical approach to extract useful reward which enables an RL agent to learn from its failures. Consider the state-space of the environment $\mathcal{S}$ and the space of goals for the task $\mathcal{G}$. We assume that for every state $s \in \mathcal{S}$, there exists some goal $g \in \mathcal{G}$ achieved in that state. This gives rise to the mapping $m : \mathcal{S} \rightarrow \mathcal{G}$ where $m(s) = g \,\forall\, s \in \mathcal{S}$, and $m(s)$ denotes the achieved goal in state s. This notion of goal-conditioned reinforcement learning is useful in transforming the unsuccessful episodes into transitions accompanying an alternate goal. In particular, we can consider any episode as successful under the condition that the goal of the episode was $m(s_T)$ where s_T is the concluding state reached in that episode.

To integrate this approach into traditional experience replay, the state input to the actor and

critic networks are appended with the desired goal for the episode. Let $s||g$ denote the state vector concatenated with the goal. If $\mu(\cdot|\theta^{\mu})$ denotes the policy function(actor) and $Q(\cdot|\theta^{Q})$ represents the Q-value function(critic) of the DDPG algorithm, the equations of these networks will be modified as

$$a_t = \mu(s_t||g\ |\theta^{\mu}) \tag{2.26}$$

$$Q_t = Q(s_t||g\ , a_t\ |\theta^{Q}) \tag{2.27}$$

The system of training DDPG using HER consists of two phases. In the first phase, we first sample a target goal g at the start of every episode and add the experience tuples $(s_t||g\ , a_t\ , s_{t+1}||g\ , r_t)$ to the replay buffer. This is the standard experience replay phase. Next, we execute hindsight experience replay where we pretend that the goal we intended to achieve was the goal corresponding to the state of the environment at the terminal step of the episode $g' = m(s_T)$. We modify the sequence of transitions generated during standard experience replay, by replacing the target goal g with the achieved goal g'. We supplement the replay buffer with hindsight experience tuples $(s_t||g'\ , a_t\ , s_{t+1}||g'\ , r'_t)$, where the reward is recomputed at each step in accord with the new goal.

Thus, the replay buffer is augmented with additional training data, increasing the sample-efficiency of DDPG. Furthermore, the successful transitions which accompany the hindsight experience replay mitigate the sparse reward problem and can help the agent learn policies for high-dimensional continuous control. The HER algorithm is presented in Fig. 2.5.

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 $G := \mathbb{S}(\textbf{current episode})$
    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 2.5: HER Algorithm (Andrychowicz, et al.)[2]

Chapter 3

Weighted Rewards for Hindsight Experience Replay

3.1 Hindsight Bias

Hindsight bias [10] is a widely-observed cognitive effect referring to the inflation in a person's predicted likelihood of the true outcome of an event, after the outcome is known. This phenomenon, also termed as "creeping determinism", is one of the most pervasive cognitive biases and routinely affects judgments in multiple domains, including medical diagnoses [11], criminal justice [12] and financial systems [13]. Upon inspection, we can characterize a mathematically analogous phenomenon of hindsight bias in the regular implementation of the Hindsight Experience Replay(HER) algorithm, which we shall refer to as vanilla-HER henceforth.

Compare the real experience tuple $(s_t||g, a_t, s_{t+1}||g, r_t)$ in Fig. 2.5 to the artificially constructed hindsight experience tuple $(s_t||g^h, a_t, s_{t+1}||g^h, r_t^h)$ (we have replaced g' with g^h to convey hindsight goals). This conversion of the a real experience to its corresponding hindsight experience makes the following unjustified assumption - *Given different inputs $s_t||g$ and $s_t||g^h$,*

the policy network $\mu(\cdot;\theta^{\mu})$, returns the same action, a_t. This assumption overestimates the probability assigned by the policy network to a_t, given the input $s_t||g^h$. If we actually execute the policy network with $s_t||g^h$ as input, it is unlikely to output a_t, making s_{t+1} also unlikely. Hence we observe a chain of compounding uncertainty along the hindsight episode.

Therefore, to more effectively use HER, we require to correct the hindsight bias induced by this overestimated probability. The intuitive check would be to generate hindsight experiences by using models capable of counterfactual reasoning, i.e. by asking the network *what if g^h was the actual goal*, instead of mere substitution of real experiences. However, this a critical limitation of deductive learning models [14] and remains a challenge for the future. In the following section, we present an alternative solution to this problem to which is simple and effective, and does not additional complexity.

3.2 Proposed Method: ARCHER

We propose a simple solution to offset the hindsight bias introduced by vanilla-HER. We make that case that a hindsight experience and a real experience cannot be treated in the same manner as real experiences are authentically generated by interacting with the environment, and hence their probability is unbiased. In contrast, to overcome hindsight bias, we need to match the true probability of the hindsight experiences to their biased probability. To do so, we nudge the current policy to be more consistent with the hindsight data in the replay buffer. Hence, to meet the overestimated hindsight likelihood of a_t for $s_t||g^h$, we utilize more aggressive hindsight rewards, so that a large positive reward given to a successful hindsight transition greatly increases the Q-value of the hindsight state-action pair, which indirectly drives an aggressive policy update towards choosing this maximizing action for the given hindsight state. Following from this idea, our method is titled "ARCHER: Aggressive Rewards to Counter bias in Hindsight Experience Replay".

We test our hypothesis by introducing two real-valued scalar multipliers, λ_r and λ_h, to distinguish between real rewards r_t and hindsight rewards r_t^h as follows:

$$r_t = \lambda_r \times r(s_t, a_t, g) \tag{3.1}$$

$$r_t^h = \lambda_h \times r(s_t, a_t, g^h) \tag{3.2}$$

where r($\cdot$) is the given reward function for the task. Vanilla HER is a special case with $\lambda_r = \lambda_h = 1$.

ARCHER framework requires that $r_t^h \geq r_t$. Hence using Eq. (3.1) and Eq. (3.2) we get,

$$\lambda_h \times r(s_t, a_t, g^h) > \lambda_r \times r(s_t, a_t, g) \tag{3.3}$$

Therefore in domains with positive reward functions, i.e. $r(\cdot) \geq 0$, ARCHER comprises weights with $\lambda^r < \lambda^h$. Conversely, in negative reward functions, i.e. $r(\cdot) \leq 0$, ARCHER comprises weights with $\lambda^r > \lambda^h$.

Using these weights, the target values for real experience and hindsight experience (Eq. **??**) can be rewritten as,

$$y_i = \lambda_r \cdot r_i + \gamma Q'(s_{i+1}||g, \mu'(s_{i+1}||g; \theta^{\mu'}); \theta^{Q'}), \tag{3.4}$$

$$y_i^h = \lambda_h \cdot r_i + \gamma Q'(s_{i+1}||g^h, \mu'(s_{i+1}||g^h; \theta^{\mu'}); \theta^{Q'}) \tag{3.5}$$

We present our complete algorithm below, in Algorithm 1.

Algorithm 1 Aggressive Rewards to Counter bias in Hindsight Experience Replay

Given:
- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DDPG
- a strategy $\mathbb{S}$ for sampling goals for replay
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$
- real reward weight λ_r, hindsight reward weight λ_h

Initialize $\mathbb{A}$
Initialize replay buffer R
for episode $= 1, M$ **do**
  Sample a goal g and initial state s_0
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavior policy from $\mathbb{A}$:
      $a_t \leftarrow \mu(s_t || g \,|\, \theta^{\mu}) + \mathcal{N}$ ▷ || denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  for $t = 0, T-1$ **do**
    $r_t = \lambda_r r(s_t, a_t, g)$
    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, $G = \mathbb{S}(\text{current episode})$
    for $g^h \in G$ **do**
      $r_t^h = \lambda_h r(s_t, a_t, g^h)$
      Store the transition $(s_t || g^h, a_t, r'_t, s_{t+1} || g^h)$ in R ▷ hindsight experience replay
  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

3.3 Experimental Setup

3.3.1 Network Architecture and Training

For our experimental setup, the actor and critic networks were designed with 2 fully connected hidden layers, consisting of 400 ReLU neurons in the first and 300 ReLU neurons in the second layer. The output layer of the actor networks used tanh activation. The layers of the networks were initialized uniformly from $\left[\frac{-1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$ where f was the fan-in of the layer. The final layers of the networks were initialized uniformly in the range $[-3 \times 10^{-3}, 3 \times 10^{3}]$. The discount factor γ

was 0.98. The weight τ for the weighted update of the target networks was 0.001. To encourage exploration, we added Ornstein-Uhlenbeck process noise with $\theta = 0.15$ and $\sigma = 0.2$ to the output of the actor. The added noise is multiplied by a factor ϵ, with an initial value of 0.1 with an exponential decay factor of 0.99, to gradually reduce exploration. The learning rates for the actor and critic were 10^{-4} and 10^{-3} respectively, and the networks were trained using Adam optimization. We used a replay buffer of size 10^5 from which minibatches of size 128 were uniformly sampled for training.

3.3.2 Reward Functions

We analyze the performance of ARCHER in comparison to vanilla HER, across 3 types of rewards for each task:

- **Binary -1/0 reward**: In this case, the agent receives a reward of -1 for every time-step in the episode where the goal is not achieved, and receives 0 when the agent it successful.

$$\begin{aligned} r(s_t, a_t, g) &= -[f_g(s_{t+1}) = 0] \\ &= \begin{cases} -1, & \text{if } m(s_{t+1}) \neq g \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.6}$$

  This reward function penalizes the agent for not achieving the goal at every time-step and therefore the agent is incentivized to learn a time-efficient optimal policy. The negative reward punishes the agent for executing unproductive actions. It is used in vanilla HER.

- **Binary 0/+1 reward**: In this case, the agent is awarded a value of 0 for every unsuccessful

time-step and is granted a reward of 1 when the goal is achieved.

$$\begin{aligned} r(s_t, a_t, g) &= [f_g(s_{t+1}) = 1] \\ &= \begin{cases} 0, \text{ if } m(s_{t+1}) \neq g \\ 1, \text{ otherwise} \end{cases} \end{aligned} \tag{3.7}$$

The positive rewards encourage the agent to learn the policy which allows it to collect most reward and encourages the successful actions taken by the agent.

- **Shaped reward**: In this case, the agent is provided with a continuous real-valued reward signal, in proportion to some metric representing how close to/far away from success the agent is. For our experiments, we have selected the negative of Frobenius norm of the difference vector between the achieved goal and actual goal. The Frobenius norm of a vector A is given as

$$\|A\|_F = \left(\sum_{i,j=1}^{n} |a_{ij}|^2 \right)^{1/2}$$

. Therefore, the shaped reward function is given by

$$r(s_t, a_t, g) = -\|m(s_t) - g\|_F \tag{3.8}$$

The agent receives a large negative reward for states far away from the goal state and rewards closer to 0 when the agent is close to the goal. Vanilla HER performs poorly in tasks with shaped reward.

3.3.3 Weights

For each of our experiments, we carefully selected the weights to gain insight into the relative reward optimization between hindsight and standard experience replay. The following set of weights helps us understand the impact on performance driven by (i) the ratio between hindsight

and real reward weights (ii) the magnitude of these weights relative to baseline HER.

- $\lambda_r = 1, \lambda_h = 1$: These weights are the same as vanilla-HER, which serves as the baseline standard for our tests.
- $\lambda_r, \lambda_h \in \{0.5, 1\}$: The magnitude of the smaller weight is half of the baseline weight. We have $\lambda_r = 0.5, \lambda_h = 1$ and $\lambda_r = 1, \lambda_h = 0.5$.
- $\lambda_r, \lambda_h \in \{1, 2\}$: The magnitude of the larger weight is twice the baseline weight. We have $\lambda_r = 2, \lambda_h = 1$ and $\lambda_r = 1, \lambda_h = 2$.
- $\lambda_r, \lambda_h \in \{0.5, 2\}$: The magnitude of the larger weight is twice the baseline weight and the magnitude of the smaller weight is half of the baseline weight. We have $\lambda_r = 2, \lambda_h = 0.5$ and $\lambda_r = 0.5, \lambda_h = 2$.

3.3.4 Goal Sampling

We test two strategies for choosing hindsight goals g^h as proposed in HER [2] -

- **Final**: Single hindsight goal corresponding to the goal reached by the final state of each episode.
- **Future**: k hindsight goals corresponding to k random states which come from the same episode as the transition being replayed and were observed after it (we use the best reported practice with $k = 4$).

3.3.5 Simulation Environments

We evaluate our method on the DeepMind (DM) Control Suite [15] simulation software. This library consists of a set of continuous control environments in Python, built on top of the

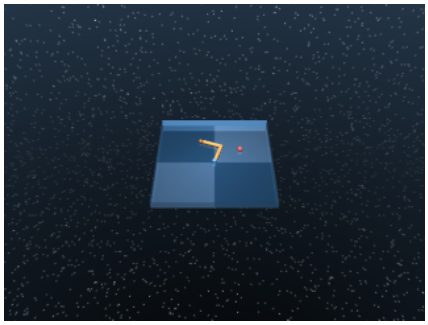
MuJoCo physics engine [16]. Each environment in the suite provides a physical task along with a well-defined continuous action space A, continuous state/observation space S, and intrinsic transition dynamics based on the physics engine. For our experiments, we program our own reward functions described in Sec. 3.3.2 to conduct ablation studies on ARCHER and verify its robustness, as detailed in the following sections. We tested our algorithm on the following two domains -

Reacher

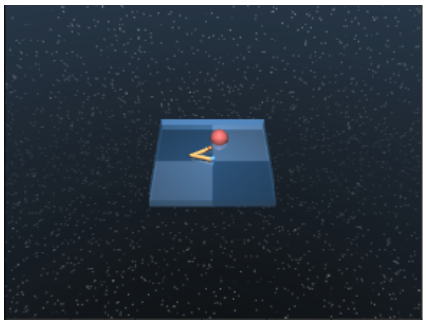
The Reacher environment (Fig. 3.1) includes a 2-DoF planar robot arm where the agent must place its end effector at a randomized target, indicated by the red sphere. In the dense reward condition (Fig. 3.1b), the target has a large size. The state input s_t is a 4-element vector with the first 2 elements containing the generalized positions and next 2 elements containing the generalized velocities of the two arm joints, as encoded by the Mujoco physics engine. s_t is concatenated with a real/hindsight goal g which specifies the 3-dimensional global coordinate of the target sphere/end effector respectively. The action a_t is a 2-dimensional vector where each element informs the relative displacement of each joint from its current position.

Finger

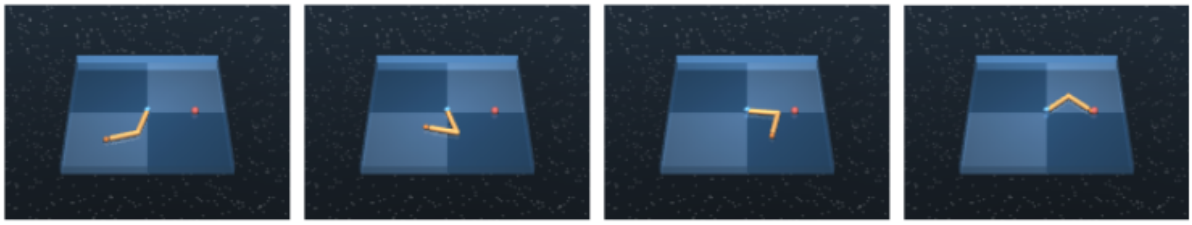
The finger environment (Fig. 3.2) is a multi-body arrangement where a planar 2-DoF robot arm has to flick a spinner resting on an unactuated hinge so as to place the red tip of the spinner on the target indicated by a red sphere. The arm must therefore learn a policy in a environment with discontinuous dynamics [17]. The state s_t is concatenation of a 4-dimensional position vector (generalized joint points as the first two elements and the relative (x, z) position of the spinner tip to its hinge as the next two elements), a 3-dimensional velocity vector of the 3 joints and followed by signals from the two touch-sensors on the top and bottom of the spinner. This



(a) Sparse Reward



(b) Dense Reward



(c) Example rollout in Reacher

Figure 3.1: Reacher Environment

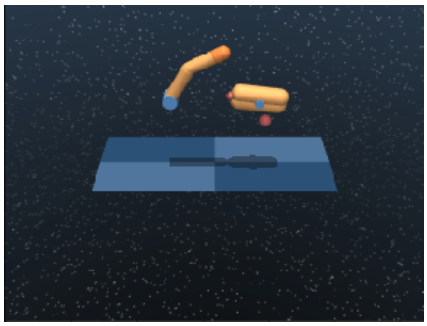
9-dimensional state is concatenated with a goal g determined by the position of target sphere relative to the hinge for real episodes, and the relative position of the spinner tip for hindsight episodes. The 2-dimensional action vector a_t specifies the relative displacement of the two arm joint from their current positions.

In both domains, the initial state and target locations are randomly initialized at the start of every episode.

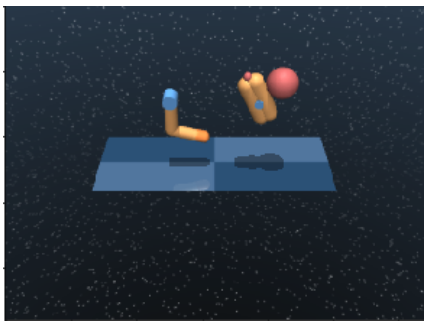
3.3.6 Network Architecture and Training

For our experimental setup, the actor and critic networks were designed with 2 fully connected hidden layers, consisting of 400 ReLU neurons in the first and 300 ReLU neurons in the second layer. The output layer of the actor networks used tanh activation.

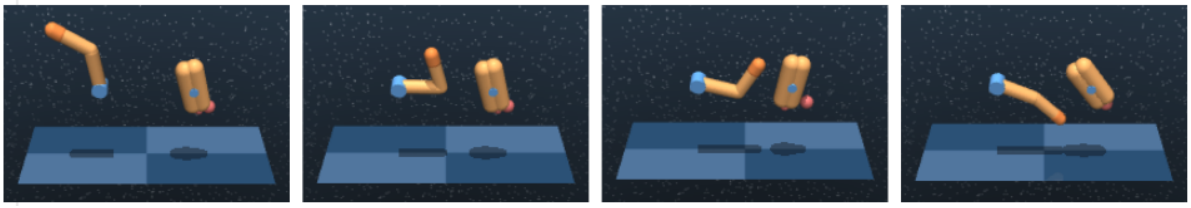
For our experimental setup, the actor and critic networks were designed with 2 fully con-



(a) Sparse Reward



(b) Dense Reward



(c) Example rollout in Finger

Figure 3.2: Finger Environment

nected hidden layers, consisting of 400 ReLU neurons in the first and 300 ReLU neurons in the second layer. The output layer of the actor networks used tanh activation. The layers of the networks were initialized uniformly from $\left[\frac{-1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$ where f was the fan-in of the layer. The final layers of the networks were initialized uniformly in the range $[-3 \times 10^{-3}, 3 \times 10^{3}]$. The discount factor γ was 0.98. The weight τ for the weighted update of the target networks was 0.001. To encourage exploration, we added Ornstein-Uhlenbeck process noise with $\theta = 0.15$ and $\sigma = 0.2$ to the output of the actor. The added noise is multiplied by a factor ϵ, with an initial value of 0.1 with an exponential decay factor of 0.99, to gradually reduce exploration. The learning rates for the actor and critic were 10^{-4} and 10^{-3} respectively, and the networks were trained using Adam optimization. We used a replay buffer of size 10^5 from which minibatches of size 128 were uniformly sampled for training. The networks were trained for 2000 cycles, where 1 cycle represents running the policy for 16 episodes followed by 40 steps of optimization.

3.4 Experiments and Results

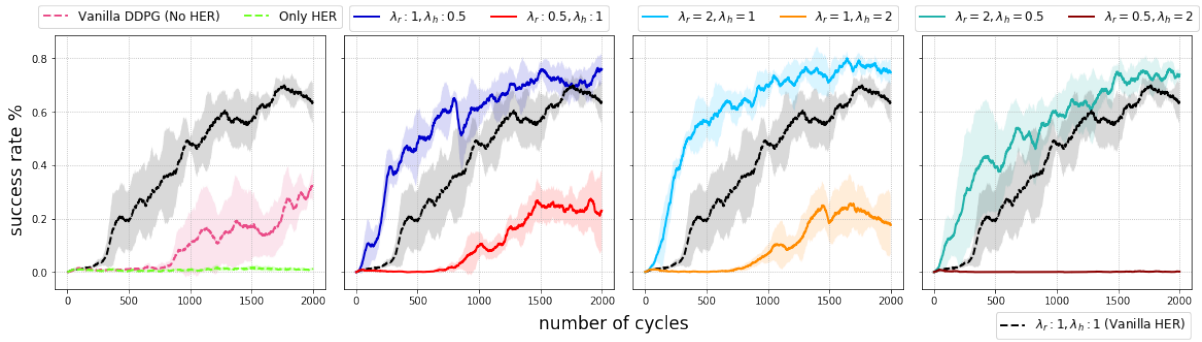
We evaluated our method by comparing the success rate of the learned policies against the required number of cycles to achieve that performance. An episode was considered successful if on final step of the episode, the end effector was placed at the target in the case of Reacher; or the spinner tip was aligned with the target in the case of Finger. Each episode consisted of 50 MuJoCo time-steps. The results presented in Fig. 3.3 and Fig. 3.4 show the performance curves of the actual actor networks of each agent, averaged over 5 random seeds and smoothed across the past 50 elements. The blue tinted plots depict HER with $\lambda^r > \lambda^h$ weights while the orange/red tinted plots depict HER with $\lambda^r < \lambda^h$ weights. Vanilla HER is shown with a perforated black plot.

I: Using Sparse Negative Rewards and the *Final* Goal Sampling Strategy

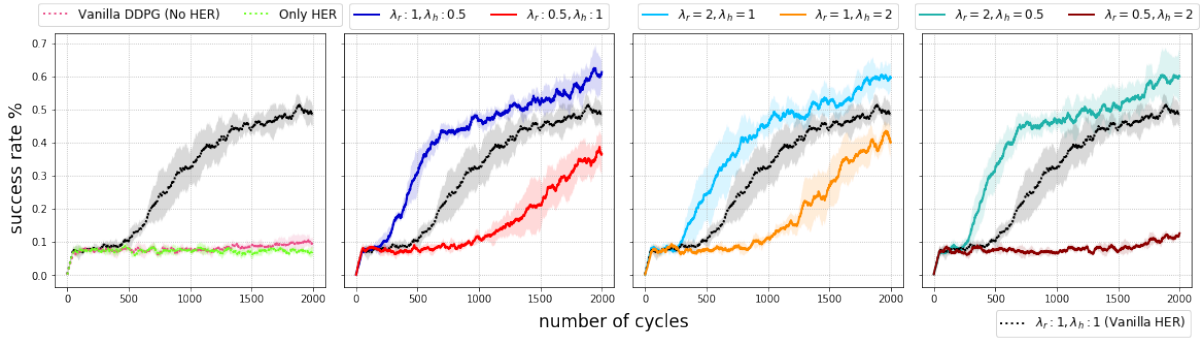
Fig. 3.3 displays the final performance plots in the Reacher and Finger domains with a binary negative -1/0 reward function (3.6). In both domains, we observe from the leftmost graph that vanilla HER greatly improves the performance of DDPG. We also see that learning exclusively from only real or only hindsight experiences decreases performance. This is because the agent now only observes one of successful or unsuccessful transitions, hampering the discriminative learning of good and bad actions. Hence we need a balanced mixture of real and hindsight experiences for competent learning.

The following 3 graphs show that the fastest performance is demonstrated in the blue curves where $\lambda^r > \lambda^h$, representing ARCHER. The hindsight bias is reduced as the weights ensure that the true expectation of values under hindsight experiences matches the optimistic estimate. We further experimented to exacerbate the discrepancy between the true hindsight probability and biased probability ($\lambda^r < \lambda^h$) to understand the effect of a large bias. From the corresponding red curves in each graph, we notice that performance is adversely affected and sample-efficiency

decreases. This confirms and validates our hypothesis of hindsight bias in HER.



(a) Reacher with negative reward function



(b) Finger with negative reward function

Figure 3.3: Policy performance in the Reacher and Finger environments with sparse binary negative rewards and the *final* sampling strategy for hindsight goals

II: Ablation Studies

Fig. 3.4 shows the results of the following ablation studies.

- **Robustness to reward design:** The first column shows the performance curves for the different algorithms when presented with a sparse binary positive 0/+1 reward function (3.7). The striking observation is that the orange tinted curves are above the baseline while the blue curves fall below. This result is consistent with ARCHER as when the reward function is positive, $\lambda^r < \lambda^h$ ensures that hindsight rewards are numerically greater. This graph shows

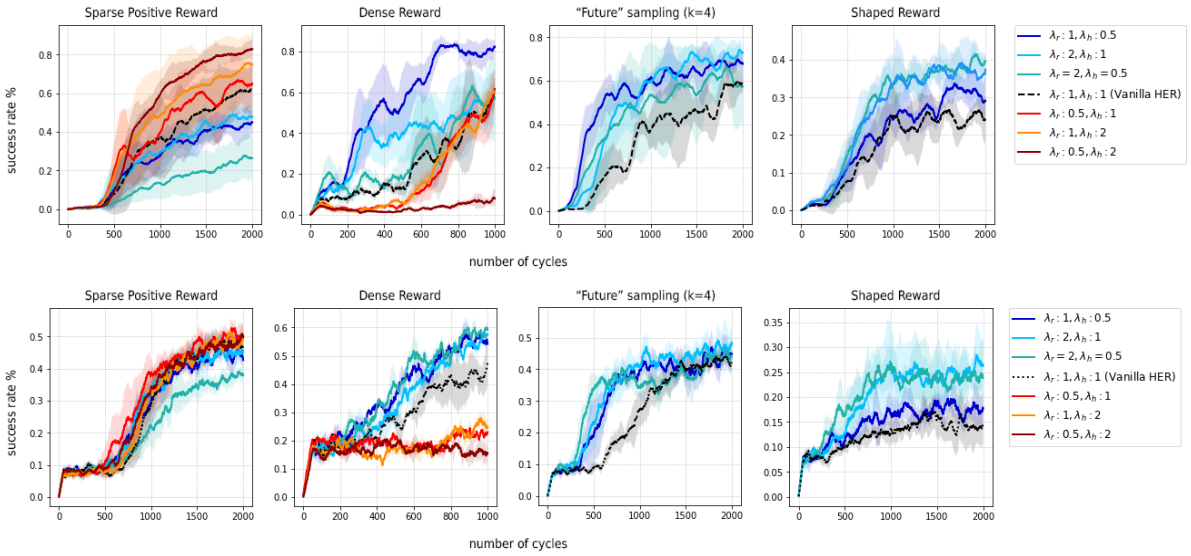


Figure 3.4: Ablation studies in the Reacher and Finger environments.

that ARCHER is not just a coincidence of implicit hyper-parameter tuning such as learning rate increase, but is robust to changes in reward sign. In the fourth column, we show the performance of the different algorithms when provided negative shaped reward (**??**). Shaped reward was mentioned as a weakness for vanilla HER in [2]. Here, we see that ARCHER outperforms the baseline even with shaped reward, although the final performance is lower than the other reward functions.

- **Robustness to reward density:** In the second column we illustrate the effectiveness of ARCHER in dense binary negative reward condition, where the size of the target sphere in both domains is magnified. As shown in the graph, the blue plots depicting ARCHER perform better than vanilla HER. In Reacher, the other curves eventually catch up to ARCHER but in the high-dimensional Finger domain, ARCHER maintains a noticeable lead. Hence the main benefit of ARCHER lies in high-dimensional, sparse reward domains.

- **Robustness to goal sampling:** In the third column, we plot the results for vanilla HER and HER with different weights when 4 "future" goals are sampled for hindsight replay with negative binary sparse reward. We observe that the blue ARCHER curves deliver the most

sample efficiency.

3.5 Conclusion and Future Work

In this thesis, we identify bias in hindsight experience replay and present a method ARCHER to counter the bias and increase sample-efficiency of deep RL algorithms, using numerically greater hindsight rewards. We also empirically verified our hypothesis of hindsight bias in vanilla HER by exhibiting that when DDPG is trained with rewards opposite to those specified by ARCHER, the bias is amplified and performance degrades. Using experiments from simulation environments, we demonstrated the increased sample efficiency derived from ARCHER in comparison to baseline vanilla HER. Our ablation studies prove ARCHER consistently outperforms vanilla HER across various reward functions and task complexities. Hence, ARCHER grants reliable sample efficiency for continuous control and robotics domains.

A few interesting directions emerge for further exploration. Some of our experiments reveal that ARCHER enjoys higher sample-efficiency only until a context-dependent number of samples, after which vanilla HER catches up to ARCHER. This effect makes intuitive sense as the high performance of ARCHER leads to the fast convergence of real and hindsight experiences, and diminished hindsight bias. Hence, a scheduled annealing of ARCHER remains of interest. Also, we specifically constructed a simple linear relation to derive a more informative hindsight reward function, however there may exist a more complex mapping between real and hindsight rewards and hence it may be advantageous to introduce a generative model to learn the latent mapping. Furthermore, measuring the performance of ARCHER on a real-world robot presents an important future direction.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[5] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping,"

[6] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[10] B. Fischhoff, "Hindsight is not equal to foresight: The effect of outcome knowledge on judgment under uncertainty.," *Journal of Experimental Psychology: Human perception and performance*, vol. 1, no. 3, p. 288, 1975.

[11] H. R. Arkes, D. Faust, T. J. Guilmette, and K. Hart, "Eliminating the hindsight bias.," *Journal of applied psychology*, vol. 73, no. 2, p. 305, 1988.

[12] J. D. Casper, K. Benedict, and J. L. Perry, "Juror decision making, attitudes, and the hindsight bias," *Law and Human Behavior*, vol. 13, no. 3, pp. 291–310, 1989.

[13] J. C. Anderson, D. J. Lowe, and P. M. Reckers, "Evaluation of auditor decisions: Hindsight bias effects and the expectation gap," *Journal of Economic Psychology*, vol. 14, no. 4, pp. 711–737, 1993.

[14] J. Pearl, "Theoretical impediments to machine learning with seven sparks from the causal revolution," *arXiv preprint arXiv:1801.04016*, 2018.

[15] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[16] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033, IEEE, 2012.

[17] Y. Tassa and E. Todorov, "Stochastic complementarity for local control of discontinuous dynamics," 2010.