Universiteit Leiden

# Master Computer Science

Hindsight Policy Gradient Interpolation


| | |
|---|---|
| Name: | Pavlos Skevofylax |
| Student ID: | s2440857 |
| Date: | 07/08/2021 |
| Specialisation: | Advanced Data Analytics |
| 1st supervisor: | Aske Plaat |
| 2nd supervisor: | Thomas Moerland |


Master's Thesis in Computer Science


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

**Abstract**

Reinforcement Learning (RL) with sparse rewards remains a major challenge. Hindsight Experience Replay (HER) is an enhancement for off-policy methods that addresses this issue by allowing the agent to learn from failed goal attempts by treating them as desired pseudo-goals. The goal of this thesis is to combine HER with on-policy algorithms. We consider the fact that off-policy RL methods are difficult to tune and tend to be unstable, while on-policy methods are easier to tune and more stable in order to build a hybrid HER method. We propose Proximal Policy Gradient Interpolation (PPGI), a hybrid RL algorithm that combines on-policy and off-policy updates. This allows us to introduce HER on the off-policy part of PPGI, achieving hindsight combined with on-policy updates. PPGI interpolates between the updates of two state of the art RL methods, Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradients (DDPG). We evaluate PPGI in 7 locomotion control tasks and hindsight PPGI in 1 simple and 2 complex goal-based navigational tasks. Our results show that PPGI can outperform PPO in one case and have a satisfactory performance in most locomotion tasks. In combination with HER, our method during interpolation outperforms the traditional off-policy HER method on the simple goal-based navigational task significantly. Regarding the 2 complex tasks, hindsight PPGI improves the performance of one while it deteriorates the performance of the other.

# Contents

# 1 Introduction

Reinforcement Learning (RL) is a sub-field of Machine Learning, where an agent interacts with an environment in order to achieve a certain goal through trial-and-error. We use the formal definition by Sutton and Barto (2018) where the RL problem is defined as a decision-making framework, in which the agent receives a state, performs an action, and the environment yields a learning signal (reward) and a new state in response. Each action is performed through a sequence of discrete time-steps. The goal of the agent is to learn a policy that maximizes the cumulative reward throughout the time-steps, by selecting appropriate actions.

In recent years, RL has been combined with Deep Learning (DL), forming the field of Deep Reinforcement Learning (DRL). In this field, (a part of) the solution is represented by deep neural networks. This improved the RL framework by allowing function approximators to learn from high dimensional state representations. The first big breakthrough that vastly popularized the field of DRL took place in 2013, where Mnih et al. (2013) combined convolutional neural networks with a variant of a popular RL algorithm called Q-learning (Watkins and Dayan, 1992). Their method, called Deep Q-Networks (DQN), learns a function approximator from raw pixels, making this the first RL algorithm that is able to fully solve some of the Atari games, using only the games' pixels as a state representation. Tesauro (1995) is an older work that combines neural networks with RL in a method called TD-Gammon, a program that managed to develop skills similar to the world champion of Backgammon in 1992. Unlike DQN, this method uses additional handcrafted heuristics in order to succeed, as well as a method called *self-play*. These two successful methods laid an important foundation on the field of RL, which led to the development of more DRL breakthroughs, that are able to learn complex games given no rules, such as Go, at a super-human level, beating world champions (Silver et al., 2016, 2017; Schrittwieser et al., 2020).

In parallel with these recent breakthroughs, DRL advancements also developed in simulated robotics. Robotic tasks usually use a continuous action space (e.g. degrees of an arm) in contrast with Atari and board games, where the action space is discrete (e.g. $(x, y)$ position on the Tic-Tac-Toe board). The number of actions in a continuous action space environment is technically infinite, therefore there are different RL approaches to tackle such tasks. A vast majority of these simulators include locomotion control, where the agent needs to control the angle of a robot's joints in its arms and legs in order to balance. Other tasks include the navigation of the agent to a certain point in the environment. The latter most commonly has a *sparse-reward* setting, because it only receives a positive reward once the agent reaches the goal destination. Tasks with a *sparse-reward* setting are more challenging because the only way for the agent to learn is if it accidentally reaches the goal destination and receives a learning signal. Since this is an unlikely scenario, there are many enhancements in RL that deal with sparse rewards (Matiisen et al., 2019; Pathak et al., 2017; Florensa et al., 2017; Held et al., 2018; Andrychowicz et al., 2017). Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) is a method for navigational robotics that addresses the sparse reward problem. It is inspired by the human cognition of learning from undesired outcomes. In this framework, when the agent reaches an undesired destination, the method treats it as if it was a desired one. This triggers the learning signal that teaches the agent how to reach that unwanted position. Upon convergence, the agent is able to navigate to any given position in the environment.

The presented thesis explores the capabilities of HER when combined with hybrid RL algorithms. The intuition behind our approach is to combine the stability of *on-policy* methods and

the sample efficiency of *off-policy* approaches with the success of HER. We start by reproducing our own variant of the hybrid RL algorithm Interpolated Policy Gradients (IPG) (Gu et al., 2017) which we call Proximal Policy Gradient Interpolation (PPGI). We test our method on 7 different locomotion environments and compare our results and findings with the ones from Gu et al. (2017). Afterwards, we apply HER on PPGI and test it on 3 navigational robotic tasks with a *sparse-reward* setting.

## 1.1 Research Goal

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) and its variants (Fang et al., 2019) have shown great potential in multi-goal based environments. It is usually combined with off-policy methods as it doesn't require additional mathematical operations to achieve good results in contrast with on-policy hindsight approaches. On the one hand, despite being sample efficient, off-policy methods are unstable and hard to tune. On the other hand, on-policy methods are easier to configure and more stable, but usually at the cost of sample efficiency. The goal of this research project is to effectively combine on-policy and off-policy methods into a new hybrid RL algorithm in order to achieve off-policy hindsight with the benefits of on-policy approaches. Previous work with hybrid RL algorithms already exist (Gu et al., 2016, 2017; Fakoor et al., 2020; Wang et al., 2016; O'Donoghue et al., 2016), but none of them combine their approaches with hindsight. We achieve this using techniques from Interpolated Policy Gradients (IPG) (Gu et al., 2017), a method that combines Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), as recommended by Plappert et al. (2018).

We will address the following research questions:

- Can the on-policy part of IPG be extended from TRPO to a more state-of-the-art algorithm like Proximal Policy Optimization (PPO) (Schulman et al., 2017)?
- How does the implementation of HER with a hybrid RL algorithm perform in comparison with the traditional HER method?

## 1.2 Approach

The thesis is separated into two parts. First we reproduce a variant of the IPG method with improved algorithmic versions, which we call Proximal Policy Gradient Interpolation (PPGI). The original IPG method is a hybrid algorithm which combines TRPO and DDPG. We instead use PPO, an improved version of TRPO, and DDPG. We apply this method on locomotion tasks with a *dense-reward* setting. Second, we implement the HER method on the off-policy part of the algorithm, apply it on navigational robotic tasks with a *sparse-reward* setting and evaluate its performance with and without interpolation.

## 1.3 Overview

Section 2 provides a general background for the field of Reinforcement Learning with a strong emphasis on the things that are important for this thesis, such as variance and bias trade-off, exploration, exploitation, importance sampling and goal-conditioned RL, which includes the sparse-reward problem and Universal Value Function Approximators (UVFA). Section 3 introduces the reader to deep learning and the architecture of Multi-layer Perceptrons (MLP).

Section 4 gives an overview of the field of Deep Reinforcement Learning (DRL) and introduces 4 DRL algorithms that laid the foundation for our PPGI method. In section 5 we present our method. We explain how we achieve interpolation with PPO and DDPG and how we combine PPGI with HER. In section 6 we show our experimental set up and results. Our set-up includes 2 parts, one that evaluates the overall PPGI method on 7 locomotion control tasks with a dense-reward setting and one that evaluates our method with HER on 3 goal-based environments with a sparse-reward setting. Section 7 provides related work prior to our method for hybrid RL algorithms, methods that address the sparse-reward problem and on-policy hindsight. In section 8 we discuss our results and in section 9 we conclude our work.

## 1.4 Results

The main hyper-parameter of our method is the interpolation, which controls the trade-off between off-policy and on-policy updates. We evaluate our PPGI method on 7 locomotion tasks with different interpolation parameters. In comparison to PPO, in 1 case interpolation improves the performance, in 2 cases it performs similar to PPO, in 2 cases it performs sub-optimal and in 2 cases it performs poorly. In addition, we try a control variate (CV) technique mentioned in the Appendix of Gu et al. (2017) that is used for variance reduction. We evaluate PPGI with and without the CV in two locomotion tasks and the results show that the performance deteriorates. Lastly, we try two different sampling methods for the off-policy part of PPGI in two locomotion tasks, sampling random transitions and sampling recent transitions. The performance in both tasks is very similar, with random sampling performing insignificantly better.

We evaluate our hindsight PPGI approach on 3 goal-based environments, 1 simple and 2 complex. In the case of the simple task, our hindsight PPGI method significantly outperforms the traditional HER method by combining the advantages of both on-policy and off-policy updates. The interpolation effect is very apparent because this simple task can be solved using PPO without any enhancements. In the case of 1 complex task, the interpolation effect is not obvious, but it still manages to improve the traditional HER method using a small amount of on-policy updates. For the final complex task, our method doesn't outperform the complete off-policy HER approach, which is an indicator that our method can reach its limit depending on the environment complexity.

# 2 Reinforcement Learning

The following section is based on Sutton and Barto (2018) and David Silver's RL lectures (David Silver, blog), except sub-section 2.6 which is based on Seungjae Ryan Lee's blog post (Seungjae Ryan Lee, blog), sub-section 2.11 which is based on Chris Yoon's blog post (Chris Yoon, blog) and sub-section 2.12. We cover basic Reinforcement Learning concepts that are crucial for understanding the methods for the research project presented in this thesis.

## 2.1 Markov Decision Process

We consider the standard RL formulation, in which the RL problem is defined as a Markov Decision Process (MDP). An MDP is a mathematical decision-making framework, in which a decision maker is being modeled to act. In RL, an MDP is a fully observable environment. It consists of a set of valid states $s \in S$, a set of valid actions $a \in A$, a reward function $\mathcal{R}$ and a transition probability $p(s'|s,a)$. The reward function gives feedback to the decision maker regarding how good an action is at a given a state. The transition probability describes the environment's dynamics, and tells us the probability of observing state $s'$ given state $s$ and action $a$. For this report, we consider the *finite-MDP*, in which every observable element in $\{S, A, R\}$ is finite.

## 2.2 Agent-Environment Interaction

In the RL framework, the agent interacts with an environment through episodes. An episode corresponds to a sequence of discrete time-steps *t*. For each time-step, the agent reads the environment state $s_t \in S$ and performs an action $a_t \in A$. The agent then receives a scalar reward $r_{t+1} \in \mathbb{R}$, where $r_{t+1} = \mathcal{R}(s_t, a_t)$ and a new state $s_{t+1}$ as a result of its action. This feedback loop generates a sequence of states, actions and rewards that form a trajectory $\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\}$, where $T$ denotes the final time-step (we also refer to $T$ as the time-step horizon). A trajectory represents the path of the agent from time-step $0$ to time-step $T$.

## 2.3 Policies

A policy is a mapping function that maps an observation from the state space $S$ to the action space $A$. A policy can be either *deterministic* $\pi(s)$, where $\pi : S \mapsto A$ or *stochastic* $\pi(a|s)$, where $\pi : S \times A \mapsto [0, 1]$. Deterministic policies output a single action for a state with a probability of 1. Stochastic policies are probability distributions over each possible discrete action $a \in A$ given a state $s$, such that: $\sum_{i=0}^{|A|} \pi(a_i|s) = 1$, where $|A|$ denotes the length of set $A$.

In RL, we want to model an optimal policy $\pi^*$ that maximizes the future cumulative reward $R_t$ from a current state $s_t$ at time-step $t$:

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

where $\gamma \in [0, 1]$ is a discount factor. The discount factor assigns an importance weight to future rewards throughout an episode. Most commonly $\gamma$ is close to the value of 1, making the agent strongly consider future rewards, but assign higher importance on immediate ones.

## 2.4 Value Functions

The majority of the RL algorithms include the estimation of state-value functions $V^\pi(s)$ and action-value functions $Q^\pi(s,a)$ with a current policy $\pi$. $V^\pi(s)$ measures the expected future return starting from a particular state $s$ and afterwards following the current policy $\pi$. The state-value function is formally given as:

$$V^\pi(s) = \mathbb{E}[R_t|S_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| S_t = s\right] \tag{2}$$

Similarly, $Q^\pi(s,a)$ measures the expected future return starting from a state $s$ and taking an action $a$. It is formally given as:

$$Q^\pi(s,a) = \mathbb{E}[R_t|S_t = s, A_t = a] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| S_t = s, A_t = a\right] \tag{3}$$

The value functions have a property in which they satisfy a recursive relationship, called "Bellman" equation. The Bellman equation for the state-value function is derived as follows:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R_t|S_t = s] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| S_t = s\right] \\ &= \mathbb{E}\left[r_{t+1} + \gamma\left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+2}\right) \middle| S_t = s\right] \\ &= \mathbb{E}\left[r_{t+1} + \gamma R_{t+1}|S_t = s\right] \\ &= \mathbb{E}\left[r_{t+1} + \gamma \mathbb{E}[R_{t+1}|S_{t+1}]|S_t = s\right] \\ &= \mathbb{E}\left[r_{t+1} + \gamma V^\pi(S_{t+1})|S_t = s\right] \end{aligned} \tag{4}$$

In Equation 4 we use the *law of iterated expectation* in order to arrive to the final expectation. The last expectation can be expanded to:

$$V^\pi(s) = \sum_{a\in A} \pi(a|s)\left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V^\pi(s')\right) \tag{5}$$

Similarly, the Bellman equation of the state-action value function can be written as:

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) \sum_{a'\in A} \pi(a'|s')Q^\pi(s',a') \tag{6}$$

Bellman equations form the problem of RL in a recursive manner. They tell us that the value of the current state is equal to the immediate reward added to the value of the following state.

The recursion starts from a state $s$ and goes through all possible successor states. It then backups the information to state $s$ averaged according to the model dynamics. Since MDPs

are stochastic, the environment dynamics $p$ control the uncertainty of the model. For example, in Fig. 1 we see that if we take action $a_1$ from state $s$, there is a $0.6$ chance to end up in state $s'_1$ and $0.4$ chance to end up in $s'_2$. Similarly, if we take action $a_2$, there is a $0.8$ chance to end up in $s'_3$ and $0.2$ to end up in $s'_4$. Due to the environment dynamics, the same action can yield different states.

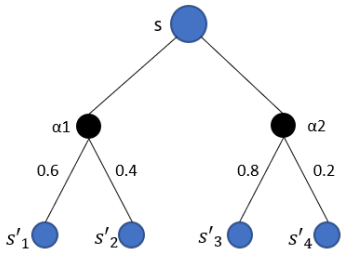



***Figure 1:*** *Showcase of the environment dynamics.*

For a policy $\pi$ to be better than a policy $\pi'$, its expected return for every state needs to be greater or equal than that of policy $\pi'$. Therefore, for an optimal policy $\pi^*$, we have:

$$V^{\pi*}(s) = \max_{\pi} V^{\pi}(s) \tag{7}$$

$$Q^{\pi*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{8}$$

With $V^{\pi*}$ we know the best expected return starting from a state. With $Q^{\pi*}$, we know the best expected return, and which action yields it. Therefore, an MDP is considered solved once we have an optimal action-value function. Thereafter, we can find an optimal policy $\pi^*$ by maximizing over $Q^{\pi*}$.

The Bellman equations for the optimal value functions can be written in a way that maximize the reward, rather than averaging it. Since the optimal policy that is being followed is greedy, it always chooses the action that maximizes the value. This form of Bellman equation is called *Bellman Optimality* equation. The Bellman Optimality equation for the state-value function is given as:

$$V^{\pi*}(s) = \max_{a} \mathcal{R}(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi*}(s') \tag{9}$$

Similarly, the Bellman Optimality equation for the action-value function is given as:

$$Q^{\pi*}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a'} Q^{\pi*}(s',a') \tag{10}$$

where $a'$ denotes the available actions that can be taken from a state $s'$.

The Bellman Optimality equations form a system that can be solved in order to find the optimal value functions. Eq. 9 and 10 can be solved if we know the model dynamics. For this thesis, we consider *model-free* RL methods, in which the model dynamics is unknown and we estimate the value functions using experience collected from the policy at the agent-environment interaction. Methods that attempt to estimate the model dynamics are called *model-based* methods, but they are beyond the scope of this thesis.

## 2.5 Model-free Methods

Model-free RL methods use sampling techniques in order to estimate the value functions. They assume that the transition probability of the model dynamics is not known. Estimating the value functions can be achieved using *Monte-Carlo (MC)* learning and *Temporal-Difference (TD)* learning.

MC methods estimate value functions from collected experience. Assuming the episodes can reach a terminal state within a finite time-step horizon $T$, we can estimate the state-value function for a state $s_t$ by sampling trajectories $\tau$ following our policy until we hit a terminal state $s_T$. We can use the sampled cumulative rewards $R_t$ to estimate the state-value function. This way, the expectation in Eq. 2 can be rewritten as:

$$V(s_t) = \frac{1}{c} \sum_{i=1}^{N} R_t^{(i)} \tag{11}$$

where $c = \sum_{i=1}^{N} \mathbb{1}\{s_t \in \tau^{(i)}\}$ is the number of trajectories that include $s_t$, $N$ is the number of sampled trajectories and $R_t^{(i)}$ denotes the total discounted reward starting from state $s_t$ at trajectory $i$. We can use Eq. 11 on every state that appears in the sampled episodes to find their corresponding estimated state-value approximation.

By the law of large numbers, the estimated value function can reach the expectation of the real one, given that $N \to \infty$. This method is the *first-visit* Monte-Carlo evaluation, where $c$ only counts the first state appearance in each episode. A similar MC method to the *first-visit* one is the *every-visit* Monte-Carlo evaluation, where $c$ counts every state appearance in a trajectory, rather than the first one. Another approach is the *incremental* MC method, which updates the value function on-line using the following update rule:

$$V(s_t) = V(s_t) + \alpha(R_t - V(s_t)) \tag{12}$$

where $\alpha$ is a step size parameter (learning rate). In this method, $V(s_t)$ is updated by getting closer to the error between the true observed cumulative reward and the estimation. We refer to the value inside the error calculation that comes before the subtraction of the value estimation as the *target* value.

TD learning methods learn from episode experience as well, with the advantage that they don't require an episode to reach a terminal state in order to estimate the value function. They estimate the state-value function online, as the episode progresses. In contrast with MC methods, in TD learning we can estimate the future cumulative reward, rather than observing it, making TD learning methods applicable to continuing (non-episodic) tasks. The simplest TD method, TD(0), updates the state-value function as follows:

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{13}$$

For every step taken in the environment, we calculate the *TD-error* inside the parentheses. This error measures the difference of the state-value function before and after taking a step. For the estimation of the reward after taking a step (*target* value), we use the immediate reward given by the environment after taking an action and estimate the rest of the future return starting from the following state $s_{t+1}$. For the estimation of the reward before taking a step, we use the state-value of the initial state $s_t$. The *TD-error* returns a direction in which the state-value

estimator can move in order to improve. The value function gets updated by moving towards the error with a step size of $\alpha$. The algorithm *SARSA* (Rummery and Niranjan, 1994) is an *on-policy* TD algorithm that extends the idea of TD(0) to estimate an action-value function. The action-value function updates as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{14}$$

where all actions $a$ are generated by the current policy. The algorithm *Q-learning* is an off-policy version of TD(0), where in the target value, the action that maximizes the action-value of the next state is chosen. Following is the update equation of *Q-learning*:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{15}$$

where $a_t$ is generated from the behavioural policy and $a$ is a greedy action that maximizes the action-value for state $s_{t+1}$ (see Section 2.8 for the distinction between off-policy and on-policy methods).

TD methods have laid the foundation for the majority of the state of the art RL algorithms. Those methods include Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017), both of which are implemented for this thesis.

TD(0) takes one step in the environment in order to get the following immediate reward. This method can be extended to n-steps. For n-step TD, Eq. 13 becomes:

$$V(s_t) = V(s_t) + \alpha(R_t^{(n)} - V(s_t)) \tag{16}$$

where $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$ is the *n-step target*. Depending on the problem we are trying to solve and the environment, the ideal value of $n$ may vary.

## 2.6 Variance and Bias Trade-off

Both variance and bias appear in RL methods when we want to estimate value functions and they both affect the estimators negatively. Methods that attempt to minimize both include IPG (Gu et al., 2017), P3O (Fakoor et al., 2020) among others. We use our own variant of IPG for this thesis in order to address this issue as well.

In Machine Learning, a high variance indicates *over-fitting*, meaning that the estimators are optimized heavily around the data which leads to bad generalization. A high bias indicates *under-fitting*, meaning that the estimations are poor due to the fact that the model failed to find enough patterns in the data.

In RL, high variance comes from MC methods. Due to the environment and policy stochasticity, sampled trajectories have noisy rewards, which leads to noisy value estimations. The advantage of MC methods is that they are completely unbiased. By definition, the bias is defined as the expectation of the estimator minus the true value. In MC methods, the expectation of the estimator is $R_t$ which is equal to the true value of $V^\pi$, therefore the bias is zero.

TD-learning approaches have lower variance due to the fact that only the immediate reward is considered for the update (i.e TD(0) in Eq. 13) rather than the whole trajectory. Because of bootstrapping, at the start of training the estimator is far from the true state-value, therefore

it's biased. As training progresses, the bias decreases. The *n-step* TD method in Eq. 16 can directly interpolate between variance and bias depending on the value of the steps. As $n$ becomes larger, we get closer to the MC method, therefore the bias decreases and the variance increases. As $n$ becomes smaller, we get closer to the TD(0) method, therefore the bias increases and the variance decreases.

## 2.7 Exploration and Exploitation

Both MC and TD methods follow the principle of the Dynamic Programming approach *Generalized Policy Iteration* (GPI). This method consists of computing the state-value function following a policy $\pi$ (policy-evaluation), and then act greedily by choosing actions that maximize the current state-value (policy-improvement). When policy-evaluation and policy-improvement interact in sequence, we can eventually find an optimal value function and policy.

Exploration refers to the process of introducing randomness (noise) during action selection in order to get new values for different actions. Exploitation refers to the process of selecting the best actions so far. In order to estimate an optimal action-value function $Q^*$ in the process of policy-evaluation, we need to maintain exploration for the policies, such that all actions have a probability to get chosen, not just the best ones. Exploration strategies can be implemented on both stochastic and deterministic policies. Stochastic policies maintain exploration through their stochastic probability distribution, where sampling from that distribution introduces some noise by default. Deterministic policies require external methods to achieve this. A very common strategy is the $\epsilon$-greedy. It assigns a probability of $\epsilon$ to non-greedy actions, and $1 - \epsilon$ to deterministic actions. A variant of this method starts by assigning a high value to $\epsilon$ which results to more exploration and as training progresses, the $\epsilon$ value slowly decreases, making the actions more probable to be greedy, resulting into exploitation.

## 2.8 On-policy and Off-policy methods

The field of RL includes two approaches: on-policy and off-policy methods. Each approach has its advantages and disadvantages.

*On-policy* methods update the same policy that is used to collect data and make decisions. They use stochastic policies, such that $\pi(a|s) > 0, \forall s \in S, \forall a \in A$. Stochastic policies by default are more exploratory in the beginning of their training, but become less random as training progresses.

*Off-policy* methods collect data using a *behavioural* policy $\beta$ and evaluate a *target* policy $\pi$ in order to estimate the value functions. Both policies are usually deterministic, with the exception that the *behavioural* policy uses an exploration strategy to make decisions.

*On-policy* approaches are *n-step* methods, therefore by definition they have high variance and low bias. While they are sample inefficient, they usually result into stable learning and are easy to implement. Off-policy methods are 1-step methods, where they update the action-value function by acting greedily on the next state (see Section 4.3), therefore they suffer from high bias. Most state-of-the-art off-policy methods reuse old data, which makes them sample efficient. Regardless, due to the high bias, off-policy methods are less stable in comparison to on-policy algorithms and are more difficult to implement.

In this thesis we present a hybrid RL method that combines both on-policy and off-policy updates in order to gain the advantages of both approaches (see Section 5.2).

## 2.9 Importance Sampling

In off-policy TD methods, the update equation estimates according to an action taken under a *behavioural* policy, but updates the value function of the *target* policy. This can lead to bad value estimations if the *behavioral* policy and the *target* policy differ by a large factor. Actions taken under $\beta$ need to be similar to actions that would have been taken under $\pi$ for successful estimations. A mechanism for dealing with this issue is the *Importance Sampling* (IS).

IS is a general technique in statistics where one can estimate the expected value of samples of some distribution, using samples from a different distribution. In Eq. 17 we derive the expected value of sampling from a distribution $p$ using distribution $q$.

$$\begin{aligned} \mathbb{E}_{X\sim p}[f(X)] &= \sum p(X)f(X) \\ &= \sum q(X)\frac{p(X)}{q(X)}f(X) \\ &= \mathbb{E}_{X\sim q}\left[\frac{p(X)}{q(X)}f(X)\right] \end{aligned} \tag{17}$$

We start by opening the expectation of distribution $p$, multiply and divide $q$ with $p$ and then form the expectation over $q$. This leaves us with the final expectation, where the value of $f(X)$ is multiplied with a ratio in order to match the expected value of distribution $p$ while sampling from $q$. This ratio is called the *importance sampling ratio*.

The same can be applied to RL policies. We can correct the estimation of taking an action under a different policy by weighting the TD target with the *importance sampling ratio* of the policies. To correct the off-policy version of TD updates in Eq.13, we can multiply the target in the brackets with the IS ratio. For the 1-step version of TD, Eq.13 becomes:

$$V(s_t) = V(s_t) + \alpha\left[\frac{\pi(a_t|s_t)}{\beta(a_t|s_t)}(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)\right] \tag{18}$$

State of the art off-policy algorithms, such as DQN (Mnih et al., 2013), DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018) don't require IS. They are successors of the *Q-learning* algorithm (Watkins and Dayan, 1992), which uses actions from both the *behavioural* and *target* policy in order to update its action-value function. Nevertheless, the majority of the state of the art *on-policy* algorithms, such as TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017), utilize IS in order to re-use the collected data for multiple policy updates, saving time and computational power as they become more sample efficient.

## 2.10 Policy Gradients

Policy gradient (PG) methods utilize parametarized models of a policy, with parameters $\theta$. Using gradient ascent, the parameters of the model change in order to maximize the objective function. The objective function is formed as the expected reward of the policy $\pi_\theta$

(Lilian Weng, blog):

$$J(\theta) = \mathbb{E}_{\pi_\theta}[V^\pi] = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a) \tag{19}$$

where $d^\pi(.)$ is the stationary distribution of the Markov chain (MC) under $\pi_\theta$. The stationary distribution $d^\pi(s) = P(s = s_t|s_0, \pi_\theta)$ belongs to the model dynamics, and it tells us the probability of reaching state $s_t$ given that we started from state $s_0$ and followed the policy with parameters $\theta$. We take the gradient of the objective function $\nabla_\theta J(\theta)$ in order to find towards which direction the parameters $\theta$ should move in order to maximize the objective.

We can get rid of the environment dynamics in Eq. 19 by applying the *likelihood-ratio* trick. Using the identity $\nabla f = f \nabla \log f$, we can formulate $\nabla J(\theta)$ as follows:

$$\begin{aligned} \nabla J(\theta) &= \sum_{s \in S} d^\pi(s) \sum_{a \in A} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \\ &= \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(s|a) Q^\pi(s,a) \\ &= \mathbb{E}_{\pi_\theta}[Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)] \end{aligned} \tag{20}$$

We start by opening the expectation in Eq. 19 and take the gradient with respect to $\theta$. In Eq. 20, we use the identity we mentioned which allows us to include two policies, one whose log probabilities is multiplied with the gradients and one unaffected. This allows us to form the expectation again, which gives us a model-free solution where we can calculate the derivatives of the objective function by sampling and observing rewards.

## 2.11 Actor-Critic Methods

*Actor-critic* methods combine policy and value function approximators. Depending on the algorithm, the *critic* is either a state-value function $V_w(s)$ or an action-value function $Q_w(s,a)$ with parameters $w$. The *actor* is a policy $\pi_\theta$ with parameters $\theta$. The policy parameters update towards the direction suggested by the critic the same way as the final expectation in Eq. 20, but instead of a real value function, an approximator is used. Actor-critic is a general framework which is widely adapted by many state-of-the-art RL algorithms.

Policy gradients by default (Eq. 20) introduce variance due to noisy rewards and value estimations. Many actor-critic approaches, such as Advantage Actor Critic (A2C, on-policy variant of Mnih et al. (2016)) utilize a *baseline* in order to reduce the variance. A *baseline* is subtracted from the cumulative reward or the value estimation which scales down the estimation, resulting into smaller policy gradient updates. The most common *baseline* used in on-policy actor-critic methods is a pamateterized state-value function. The result of the subtraction is the advantage function, denoted with $A(.,.)$. In this case, we update the policy parameters as follows:

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}[A(s,a) \nabla \log \pi_\theta(a|s)] \tag{21}$$

where $A(s,a) = Q_w(s,a) - V_u(s)$. We can avoid using two critics for the advantage function, by calculating the action-values using the state-value network. $Q(s_t, a_t)$ can be re-written as the immediate reward of taking action $a_t$ added to the state-value of the following state. The advantage function then becomes: $A(s_t, a_t) = r_t + V_u(s_{t+1}) - V_u(s_t)$. More recent on-policy actor-critic approaches such as TRPO (Schulman et al., 2015) and PPO (Schulman et al.,

2017) use an estimator for the advantage function. Other examples of actor-critic methods include off-policy approaches, such as DDPG (Lillicrap et al., 2015). DDPG is a 1-step method, therefore it doesn't utilize a *baseline* as it already has a low variance. Instead, it uses a critic that directly provides the gradient directions to the policy (see Eq. 29).

## 2.12 Goal-Conditioned Reinforcement Learning

The second part of our thesis focuses on goal-conditioned RL, where we combine our main hybrid RL method with an approach for goal-based environments called Hindsight Experience Replay (HER) Andrychowicz et al. (2017).

Goal-conditioned RL refers to the RL problem where the agent's task is to reach a goal state within a limited time-step horizon. Early work on this problem was done by Kaelbling (1993) and Kaelbling (1993). Both of these works replace the classic value functions in RL with a cost value function $D(s, a, g)$, that corresponds to the cost of reaching a goal state $g$ from a state $s$ after taking action $a$ and following the policy thereafter. Kaelbling (1993) uses an approach similar to Q-learning (Watkins and Dayan, 1992), where the action-value function is replaced with the state-to-goal cost function. In this case, $D(s, a, g)$ corresponds to the expected number of steps needed to reach state $g$ from state $s$ after taking an action $a$. In contrast with Q-learning, they update the value function by choosing actions that minimize it. Kaelbling (1993) is an extension of the former paper, where they construct a connectivity graph between all possible landmarks. Given a state $s$ and a goal-state $g$, the algorithm starts by finding the nearest landmark to $s$ and the nearest landmark to $g$. If those landmarks match, the policy performs the best local action $a$ to get to goal $g$, where $a = \min_a D(s, a, g)$. If the landmarks do not match, their method finds the second closest landmark to $s$ and $g$. This process repeats until the closest landmarks of $s$ and $g$ match.

For this thesis, we consider the same components of goal-conditioned RL as in most recent work in the field (Andrychowicz et al., 2017; Fang et al., 2019; Florensa et al., 2017; Held et al., 2018; Schaul et al., 2015). We denote the goal-space with $G$, which includes valid goals $g$ in $(x, y)$ or $(x, y, z)$ positions. We use the binary reward function in Eq. 22, where $\delta$ is a distance threshold and $d(.,.)$ denotes the distance between the goal $g$ and the agent's current position $p$.

$$\mathcal{R}(s, g, a) = \begin{cases} 0, & \text{if } d(p, g) \leq \delta \\ -1, & \text{otherwise} \end{cases} \tag{22}$$

A binary reward function indicates that the RL task is under a sparse-reward setting, where a learning signal is rarely achieved.

### 2.12.1 Sparse-Reward Problem

In RL, sparse reward settings are difficult to solve, due to the fact that most of the time the reward is negative, retrieving no learning signal for the agent to learn. In the goal-conditioned setting, the reward is positive only when the goal is reached, and negative otherwise (see Eq. 22). In complex goal-based environments, reaching a desired goal-state by accident is highly unlikely. Typical RL approaches can't work without enhancements. For this thesis, we combine Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), a method that addresses the sparse reward problem, with our PPGI approach.

### 2.12.2 Universal Value Function Approximators

Schaul et al. (2015) introduce the Universal Value Function Approximators (UVFAs). They suggest three approaches where the parametarized value functions can be trained such that they generalize among unseen goals as well, rather than only states. We use the more general and most commonly used approach, where the states $s \in S$ and goals $g \in G$ are concatenated together, forming higher dimensional observations, resulting into value functions $V(s, g)$ and $Q(s, g, a)$.

# 3 Deep Learning

The field of Deep Reinforcement Learning combines RL methods with Deep Learning approaches, leading to policies that are able to perform on complex environments and high dimensional observations. Our method combines the algorithms Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), both of which belong to the Deep RL field. In this section, we introduce a basic neural network architecture so the reader can better understand the following section regarding Deep RL.

Deep Learning (DL) methods model black-box differentiable functions in the form of multi-layer Neural Networks (NN). Neural Network architectures are considered deep when they have at least 3 layers. Deep Neural Networks (DNN) have achieved state-of-the-art results in the fields of image recognition (Krizhevsky et al., 2012), speech generation (Oord et al., 2016), image generation (Bau et al., 2020) among others. They are able to extract complex patterns and features from the data which leads to great performance.

## 3.1 Perceptron

Neural Networks, inspired by the functionality of the human brain, are used for complex ML tasks. They consist of artificial neurons which are represented by computational nodes. These nodes are mathematical operators that compute the weighted sum of the input features $x$, add a bias term and output the result. Each input feature is weighted in accordance to its corresponding weight $w$.

The simplest NN type is a single-node NN called *perceptron* (Rosenblatt, 1958), used for binary classification problems. Illustrated in Fig. 2, the perceptron takes a vector of values $x$ as input, calculates the weighted average of each input according to its weight $w$, adds a bias term $b$, passes the result to an activation function $f$ and outputs the result $o$, where $f : x \mapsto [-1, 1]$ (Eq. 24) and $o = f(\sum_i w_i x_i + b)$. The prediction can then be $y_1$ if $o \leq 0$ and $y_2$ if $o \geq 0$ . Once we have the prediction of the model, we calculate the error $e = d - y$, where $d$ is the true label of $x$ and $y$ is the predicted one. The weights can then be updated with: $w = w + \alpha e x$. This optimization process repeats multiple times until convergence to the optimal weights. The update rule for the perceptron is derived from the *Gradient Descent* (GD) algorithm (Goodfellow et al., 2016).

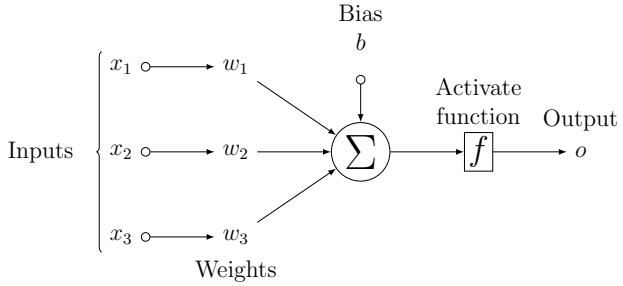



***Figure 2:*** *Illustration of the perceptron.*

## 3.2 Activation Functions

Neural networks are extended versions of previous linear ML models such as *logistic regression*. They perform more computations and utilize different activation functions in order to learn complex, non-linear patterns. The most common activation functions include: the Sigmoid (Eq. 23), the Hyperbolic Tangent Activation Function (Eq. 24) and Rectified Linear Unit (Eq. 25),

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{23}$$

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{24}$$

$$ReLU(z) = \max(0, z) \tag{25}$$

where $z$ is the weighted average of the network's input (see Section 3.1), $sigmoid : z \mapsto [0, 1]$, $tanh : z \mapsto [-1, 1]$ and $ReLU \mapsto \{0, z\}$. Depending on the task that needs to be solved, the appropriate activation function can be chosen.

## 3.3 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is the extension of the perceptron, where between the input and output there exists at least one hidden layer, performing more computations on the data. As shown in Fig. 3, the output of each layer is passed as an input to the neurons of the following layer, getting filtered through more activation functions and multiplied with more weights. Similar to the perceptron, an error function is used in order to update the weights after one forward pass. MLPs are commonly used to model RL policies and value functions due to their ability of extracting useful information from complex data.

# 4 Deep Reinforcement Learning

RL has its limitations when in comes to high dimensional state representations. In such cases, we need to model value functions and policies that are able to generalize among similar states. This can be achieved by combining deep learning with RL. This section includes two state-of-the-art Deep RL algorithms, PPO and DDPG (Schulman et al., 2017; Lillicrap et al., 2015), which we combine to create our own variant of the Interpolated Policy Gradients method

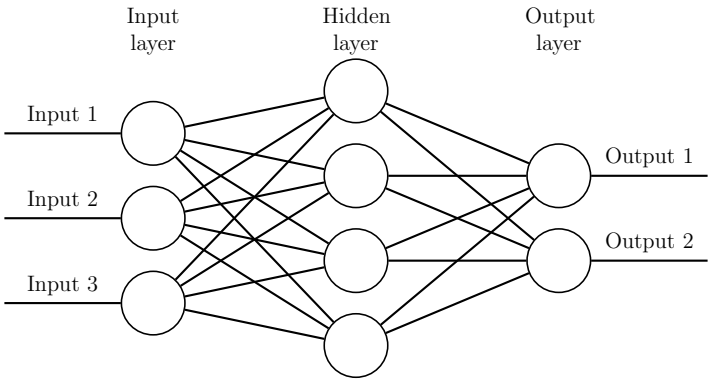



***Figure 3:*** *Illustration of a simple Multi-layer Perceptron with 1 hidden layer.*

(Gu et al., 2017) in Section 5.2, and their predecessors that laid the foundation for such advancements.

## 4.1 Function Approximation

Function approximation is a Machine Learning (ML) method in the field of Supervised Learning (SL), in which a function can be estimated using domain data. Neural networks are a type of function approximators. Given domain data (e.g. images of dogs, cats and their corresponding labels), we can train a neural network into a mapping function $f$, that maps an input image $i$ from the image space $I$ to a label $f : I \mapsto \{cat, dog\}$. Function approximators can generalize well with previously unseen observations, given that they are trained with enough data. Similar to SL, in RL we can model value functions and policy estimators with neural networks and optimize them.

## 4.2 Modeling Policies for Continuous Tasks

We consider stochastic policies for continuous action spaces (Sutton and Barto, 2018). A policy can be a model parametarized by $\theta$ that outputs the parameters of a Gaussian distribution, the mean ($\mu$) and the standard deviation ($\sigma$), such that:

$$\pi_\theta(a|s) = N(\mu_\theta(s), \sigma_\theta(s)) \tag{26}$$

Afterwards, an action can be sampled from that distribution given a state $s$, $a \sim N(\mu_\theta(s), \sigma_\theta(s))$. In this case, some exploration is already provided since the mean of the distribution acts as the deterministic action and the standard deviation acts as the exploration. Sampling from the distribution already provides noise given by $\sigma$.

For this thesis, we use a variant of a Gaussian policy, where only $\mu_\theta$ is parametarized and $\sigma$ is a constant. We model the policy using a 2-layer MLP with a Tanh activation function (Eq. 24).

## 4.3 Deep Q Network

Deep Q Network (DQN) (Mnih et al., 2013) is an extension of the RL algorithm called Q-learning (Watkins and Dayan, 1992). Q-learning is an off-policy TD learning method that

approximates the action-value function (Eq. 15). Q-learning is off-policy because it uses the greedy action of the next state to update its Q values, while SARSA (Eq. 14) uses the same policy to collect data and update.

Mnih et al. (2013) apply deep learning techniques with Q-learning that result to the Deep Q Network method (DQN). Specifically, they use down-sampled and cropped raw pixel representations of Atari games as state inputs. Since the state is an image representation, they model the action-value function with a convolutional neural network (Goodfellow et al., 2016). In addition, they use an *experience replay* buffer, first introduced by Lin (1992). Each transition $(s, a, r, s')$ is stored in the replay buffer, where $s'$ is the following state after $s$. During training, a mini-batch of transitions $\{s_j, a_j, r_j, s'_j\}_{j=0}^{N}$ of size $N$ is sampled at random. They optimize the action-value model by minimizing the mean squared TD-error using the sampled mini-batch:

$$L_w = \frac{1}{N} \sum_{j=0}^{N} (y_j - Q_w(s_j, a_j))^2 \tag{27}$$

where $y_j = r_j + \gamma \max_a Q_{w'}(s'_j, a)$. To increase stability, when calculating the TD target ($y_j$), a past snapshot $w'$ of the parameters of the network is used, also called *target* network. DQN was the first method to achieve decent performance in 5 Atari games and outperform humans in 2 Atari games.

## 4.4 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) is an off-policy actor-critic method that combines techniques used for DQN and Deterministic Policy Gradients (DPG) (Silver et al., 2014). DPG introduces deterministic policies to the actor-critic framework for both on-policy and off-policy in continuous action space tasks. For the off-policy part of DPG, rather than using a stochastic policy $\pi(a|s)$, they use a deterministic policy $\pi(s)$ and a stochastic behavioural policy $\beta(a|s)$. Starting from the policy improvement part in GPI, they update the policy's weights using gradient descent steps, rather than a greedy maximization over $Q$:

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha \mathbb{E}_{s \sim \rho^\beta} \left[ \nabla_\theta Q^{\pi^k}(s, \pi_\theta(s)) \right] \\ &= \theta_k + \alpha \mathbb{E}_{s \sim \rho^\beta} \left[ \nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi^k}(s, a)|_{a=\pi_\theta(s)} \right] \end{aligned} \tag{28}$$

where $\rho^\beta$ is the state distribution of the behavioural policy. The final expectation of Eq. 28 is derived by applying the *chain rule*. Following, they extend this method to the actor-critic framework by replacing the true action-value function $Q^\pi$ with an approximator $Q^w$ that gets optimized with Q-learning updates.

DDPG is a combination of the DPG method and the DQN algorithm, such that it works on continuous action spaces. It combines the deterministic off-policy actor-critic method of DPG with the replay buffer and target network of DQN. Both the actor and the critic models are represented by two separate MLPs. The actor is a deterministic policy that maps states to specific actions. To maintain exploration, data is collected using a behavioural policy $\beta$, where $\beta(s) = \pi(s) + \mathcal{N}$ and $\mathcal{N}$ is an exploration strategy. At each time-step, a transition $(s, a, r, s')$ is stored in the replay buffer $B$. Following, a mini-batch $\{s_j, a_j, r_j, s'_j\}_{j=0}^{N}$ of $N$ transitions is sampled from $B$ and the critic is updated by minimizing the loss in Eq. 27, where

$y_j = r_j + \gamma Q_{w'}(s'_j, \pi_{\theta'}(s'_j))$ and $\theta'$ denotes the target policy's parameters. The actor weights update similar to Eq. 28 as follows:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{j=0}^{N} \left[ \nabla_\theta \pi_\theta(s) \nabla_a Q_w(s,a)|_{s=s_j, a=\pi_\theta(s)} \right] \tag{29}$$

In contrast with DQN where the target network is just a snapshot of the previous parameters, in DDPG the target networks update as follows:

$$\begin{aligned} w' &= \varrho w' + (1-\varrho) w \\ \theta' &= \varrho \theta' + (1-\varrho) \theta \end{aligned} \tag{30}$$

where $\varrho \in [0, 1]$ is the polyak parameter.

## 4.5 Trust Region Policy Optimization

Trust Region Policy Optimization (Schulman et al., 2015) is the predecessor of PPO. It introduces an actor-critic method that uses *trust region* updates with policy gradients, rather than *line search* updates. *Gradient descent* falls into the *line search* updates category. Such methods start by finding the steepest direction that maximize the objective function and take a small step $\alpha$ toward that direction. *Trust region* methods set a step-size $\delta$ and create a circle region around the current parameters, with radius $\delta$. This circle (trust region) is treated as a sub-problem of the original objective function and is solved independently. Following, the optimal point within that trust region is located. This new local optimal point, acts as the center of the next trust region. This process repeats until convergence.

The intuition of *trust region* updates on on-policy methods is that we can get better policy updates by locating an optimal point within a region, rather than a step towards one direction. The latter can potentially perform a large step that ends up in an inefficient parameter space area and never recover. To approximate the original objective function locally, a quadratic model is used, which is obtained using the *Taylor* expansion up to the second derivative.

TRPO solves the following constrained problem per iteration:

$$\begin{aligned} \max_\theta \quad & \mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}, a\sim\pi_{\theta_k}} \left[ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \hat{A}_{\theta_k}(s,a) \right] \\ \text{s.t} \quad & \mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}} \left[ D_{KL}(\pi_{\theta_k}(.|s), \pi_\theta(.|s)) \right] \leq \delta \end{aligned} \tag{31}$$

where $D_{KL}$ is the KL-divergence, $\delta$ is the maximum radius of the region and $\hat{A}$ is the Generalized Advantage Estimation (GAE) (Schulman et al., 2015). The KL-divergence measures the difference between two policies, therefore the trust region includes policies that are within a divergence of $\delta$ from the current policy $\pi_{\theta_k}$.

GAE uses a value function estimator $V_u$ in order to estimate the advantage function in Eq. 21. After the policy update in Eq. 31, the value function updates using the *mean squared error* as follows:

$$L_u = \frac{1}{ET} \sum_{e=0}^{E} \sum_{t=0}^{T} (V_u(s_t) - \gamma^t R_t)^2 \tag{32}$$

where $E$ is the total number of trajectories (episodes) in a batch, $V_u(s_t)$ is the predicted return from time-step $t$ in a trajectory and $R_t$ is the observed return from a time-step $t$.

## 4.6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is the successor of TRPO. TRPO requires calculations of second derivatives to solve the constraint problem in Eq. 31 through *Taylor expansion*, which is computationally expensive and doesn't scale well as the network parameters become larger. PPO addresses this issue by adding soft constraints on the update method, leading into a first order derivative solution that can be solved with *line search* methods.

PPO with the *clipped objective* performs multiple policy updates per training iteration. Since the batch data is collected from the policy before the updates, the *importance sampling* ratio is used in order to correct the advantage estimation after each iteration. The objective function of PPO with the *clipped objective* is given as:

$$J^{CLIP}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_k}(s,a), \text{clip}(r(\theta), 1-e, 1+e)\hat{A}_{\theta_k}(s,a))] \tag{33}$$

where $r(\theta) = \pi_\theta(a|s)/\pi_{\theta_k}(a|s)$ is the *importance sampling ratio*. $r(\theta)\hat{A}_{\theta_k}$ is essentially the objective function of TRPO in Eq. 31. To avoid large policy updates, PPO uses a constraint on the IS ratio to guarantee that it stays within the interval $[1-e, 1+e]$ by clipping it if it exceeds $1+e$ or if it falls below $1-e$. Following, the objective function keeps the minimum value between the clipped objective and the normal one. This process has a similar effect as TRPO by avoiding large policy updates, which results into a method that can be optimized with *line-search* methods, making PPO more efficient than TRPO.

## 4.7 Interpolated Policy Gradients

Interpolated Policy Gradients (IPG) (Gu et al., 2017) is a hybrid RL algorithm that combines the TRPO and DDPG methods in order to interpolate between on-policy and off-policy updates. The intuition behind this method is to combine the advantages of both on-policy and off-policy approaches, such as the low variance of off-policy methods and the low bias of on-policy methods.

The following update rule is applied in IPG:

$$\nabla_\theta J(\theta) = (1-\nu)\mathbb{E}_{\rho^{\pi_\theta}}[\nabla_\theta \log \pi_\theta(a|s)\hat{A}_\theta(s,a)] + \nu\mathbb{E}_{\rho^\eta}[\nabla_\theta Q_w(s, \mu_\theta(s))] \tag{34}$$

where $\nu$ is the interpolation parameter and $\mu_\theta(.)$ is the mean of the Gaussian distribution that $\pi_\theta$ outputs for a state $s$. The parameter $\eta$ in the right hand side of Eq. 34 indicates the replay buffer sampling method for $Q_w$, which includes random or recent transitions.

To further reduce the variance of the estimator, a *control variate* (CV) (Ross, 2006) can be applied on the policy gradient term that has a similar effect as a baseline. With the CV, the update rule becomes:

$$\nabla_\theta J(\theta) = (1-\nu)\mathbb{E}_{\rho^{\pi_\theta}}[\nabla_\theta \log \pi_\theta(a|s)(\hat{A}_\theta(s,a) - A_w(s,a))] + \mathbb{E}_{\rho^\eta}[\nabla_\theta Q_w(s, \mu_\theta(s))] \tag{35}$$

where $A_w(s,a) = Q_w(s,a) - \bar{Q}_w(s,a)$ and $\bar{Q}_w(s,a)$ is the CV. The authors of IPG apply the CV that is used in the Q-Prop method (Gu et al., 2016), where $\bar{Q}_w(s,a)$ is the first-order Taylor expansion of the off-policy critic.

The idea behind the control variate from Gu et al. (2016) comes from the observation that an off-policy critic is biased with low variance, therefore it can be used as a type of CV to the policy gradient term. In a policy gradient setting with a Monte Carlo action-value estimator (i.e estimator of $Q^\pi$ in Eq. 20), only $A_w(s,a)$ can be used as the advantage function in Eq. 35, instead of the subtraction term in the parentheses (i.e $\bar{Q}_w(s,a)$ would act as a baseline). According to Gu et al. (2016), the reason the update rule takes the form in Eq. 35 is because we want to use an advantage function estimator, therefore the update rule needs to be written in terms of advantages.

IPG updates the actor using trust region updates, therefore the left hand side of Eq. 34 uses the constraint problem in Eq. 31. The off-policy critic $Q_w$ is trained the same way as DDPG.

# 5 Method

So far we have covered the preliminaries required for understanding the components of this research project. In this section we move to the methodology, where we introduce our contribution for the main topic of research. We cover our own variant of the IPG hybrid algorithm which we call PPGI and afterwards we cover our implementation of HER on the off-policy part of our approach.

## 5.1 Hindsight Experience Replay

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) is an experience replay enhancement for sparse-reward settings in goal-conditioned RL. Since it requires a replay buffer to work, HER is implemented with off-policy methods, such as DQN (Mnih et al., 2013) and DDPG (Lillicrap et al., 2015). This method is inspired by the human capability of learning from undesired outcomes. If an unwanted goal is reached within the environment, HER treats it as a desired one, yielding a positive reward. This populates the replay buffer with transitions of achieved goals and their corresponding rewards.

The augmentation part of the HER method is presented in Algorithm 1. Andrychowicz et al. (2017) use several strategies for achieved-goal selection. We use the strategy called *future*. After the episode simulation, each original transition is stored in the replay buffer, as well as $K$ duplicates of this transition with different future achieved goals and their corresponding rewards.

## 5.2 Proximal Policy Gradient Interpolation

Proximal Policy Gradient Interpolation (PPGI) is our own version of IPG. We replace the trust region updates in Eq. 34 with line search updates. Specifically, we use the PPO update rule to achieve this (see Section 4.6), which yields the following update rule for PPGI:

$$\begin{aligned}\nabla_\theta J(\theta) = (1-\nu)\mathbb{E}_{\rho^{\pi_\theta}}[\nabla_\theta \min(r(\theta)\hat{A}_{\theta_k}(s,a), \mathsf{clip}(r(\theta), 1-e, 1+e)\hat{A}_{\theta_k}(s,a))] \\ + \nu\mathbb{E}_{\rho^\eta}[\nabla_\theta Q_w(s, \mu_\theta(s))]\end{aligned} \tag{36}$$

In contrast with the IPG method where it uses the control variate from Gu et al. (2016), we use the CV mentioned in the Appendix of Gu et al. (2017). For our CV, given a Gaussian MLP

**Algorithm 1:** Hindsight Experience Replay with *future* strategy

**Input :** Policy $\pi_\theta$, buffer B
$E \mapsto$ total number of episodes
$T \mapsto$ time-step horizon
$K \mapsto$ Number of future goals to choose
**for** *0 to E* **do**
  Sample a goal $g$.
  $\tau = \emptyset \mapsto$ temporarily trajectory
  **for** *t=0 to T* **do**
    Sample action $a_t \sim \pi_\theta(a|s_t, g)$
    Execute $a_t$ and observe $s_{t+1}$, $r_t|g$.
    $\tau = \tau \cap (s_t, a_t, r_t|g, s_{t+1}, g)$
  **end**
  $B = B \cap \tau$
  **for** *t=0 to T* **do**
    **for** *k=0 to K* **do**
      Choose random achieved goal $g^{(k)}$ from $\tau$ that occurred after $t$.
      Calculate new reward $r_t|g^{(k)}$
      $B = B \cap (s_t, a_t, r_t|g^{(k)}, s_{t+1}, g^{(k)})$
    **end**
  **end**
**end**

policy, we have:

$$\bar{Q}_w(s, a) = \mathbb{E}_{e\sim N(0,1)}[Q_w(s, \mu_\theta(s) + e\sigma)] \tag{37}$$

where $\bar{Q}_w(s, a)$ is subtracted from the critic $Q_w(s, a)$ and the result is subtracted from the advantage function estimation on the left hand side of Eq. 36, the same way as Eq. 35. We choose this CV because it doesn't require the use of *Taylor expansion* in contrast with the CV in Gu et al. (2016), which is computationally expensive as it requires the calculation of second derivatives.

The whole process of PPGI is shown in Algorithm 2. We start by collecting a batch of data using the current policy and appending it in the replay buffer. After the data collection, we fit the off-policy critic for multiple training iterations using the replay buffer and update the target networks. Afterwards, we compute the GAE values using the batch data and the on-policy critic, and set these values as learning signals. In case of the CV usage, we instead set the learning signals by calculating the critic-based estimate, subtracting it from the GAE values and setting $\nu = 1$ for the off-policy update ($b$ value in line 16), the same way as the IPG method. During the policy update, we sample new transitions from the replay buffer according to $\eta$ for the off-policy part of the update and use the batch data for the on-policy part of the update. Finally, we fit the on-policy critic for multiple training iterations.

### 5.2.1 PPGI with $\nu = 1$

In the extreme cases of $\nu$, where $\nu \in \{0, 1\}$ we have either completely on-policy or completely off-policy updates. While $\nu = 0$ is the exact PPO method, $\nu = 1$ is not the exact DDPG. $\nu = 1$

**Algorithm 2:** Proximal Policy Gradient Interpolation

**Input:** $\nu, \eta, useCV$, where $useCV$ is a Boolean

Initialize:
Off-policy critic $Q_w$, target network $Q_{w'}$, on-policy critic $V_u$, stochastic policy $\pi_\theta$ (we denote the expected action of $\pi_\theta$ with $\mu_\theta$) and replay buffer $B = \emptyset$
*epochs*$\mapsto$ total epochs
$E\mapsto$ episodes per epoch
$T\mapsto$ time-step horizon
$N\mapsto$ off-policy batch size
$M \mapsto$ off-policy batch size for the policy update
**for** *0 to epochs* **do**
  Collect batch data $D = \{\tau_e\}_{e=0}^{E}$ where $\tau = \{s_t, a_t, r_t, s'_t\}_{t=0}^{T}$ using current policy $\pi_{\theta_k}$
  $B = B \cup D$
  Fit off-policy critic $Q_w$ using $B$:
    Sample $\{s_j, a_j, r_j, s'_j\}_{j=0}^{N} \sim B$
    Compute $y_j = r_j + \gamma Q_{w'}(s'_j, \mu_{\theta_k}(s'_j))$ and update $w$ according to Eq. 27.
  Update target $w' \leftarrow \varrho w' + (1-\varrho)w$, where $\varrho$ is the polyak parameter
  Compute $\hat{A}_{\theta_k}$ using $D$ and $V_u$.
  **if** *useCV* **then**
    Compute $\bar{Q}_w$ using Eq. 37
    Compute critic-based estimate $A_w = Q_w - \bar{Q}_w$
    Set learning signals $l = \hat{A}_{\theta_k} - A_w$, and $b = 1$
  **else**
    Set learning signals $l = \hat{A}_{\theta_k}$ and $b = \nu$
  **end**
  Sample $\{s_j, a_j, r_j, s'_j\}_{j=0}^{M}$ from $B$ according to $\eta$
  Update the policy parameters for multiple iterations using Eq. 36:

$$\nabla_\theta J(\theta) = \frac{(1-\nu)}{ET} \sum_{e=0}^{E} \sum_{t=0}^{T} \nabla_\theta \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} l,\ clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-e, 1+e\right) l\right) + \frac{b}{M} \sum_{j=1}^{M} \nabla_\theta Q_w(s_j, \mu_\theta(s_j))$$

  Fit on-policy critic $V_u$ using D according to Eq. 32.
**end**

is a variant of DDPG, where instead of a deterministic policy and an exploration strategy, we only use a stochastic policy. In addition, the original DDPG method updates the critic and the actor one time after each time-step is performed. In our variant, we first collect a batch of data, then update the critic for many iterations and afterwards update the actor for many iterations as well. An important hyperparameter in our setting is the number of iterations for fitting the off-policy critic. We asked Shixiang Gu, the main author of IPG (Gu et al., 2017) for the hyperparameter they use in their work. According to him, they fit the off-policy critic using the same number of iterations as the batch size. For example, if the batch data consists of $5000$ transitions, the off-policy critic gets fit with $5000$ iterations. We use the same logic for our PPGI method as well.

## 5.3 Hindsight PPGI

The last part of our thesis consists of implementing hindsight with our PPGI method. This can be achieved by two approaches. One approach is to evaluate a whole trajectory under a hindsight goal for the on-policy part of PPGI and correct the estimation using importance sampling (Rauber et al., 2017; Zhang et al., 2019). The second approach is to apply the exact HER method on the off-policy buffer of our PPGI method. While both approaches can be implemented on PPGI, we implement the latter and leave the implementation of the former for future work. Implementing HER on the off-policy part of our algorithm is simple and easier than implementing hindsight on the on-policy part. The reason behind this is that off-policy methods perform *1-step* backups, therefore we can simply assign hindsight goals in every transition since the data is not correlated to each other in a way that would affect the off-policy updates. Off-policy HER in PPGI allows us to fit the off-policy critic with hindsight transitions. Following, interpolation allows us to update the policy parameters towards the direction recommended by the hindsight critic, resulting into an off-policy HER method that can interpolate with on-policy updates. Our hindsight PPGI method combines the stability of on-policy algorithms which is a result of the *n-step* backups, the low variance of the off-policy estimators which is a result of the *1-step* backups (see Section 2.8) and the efficient use of data for goal-based tasks provided by the HER enhancement.

In contrast with vanilla DDPG, where the networks update after each time-step is performed, in HER (Andrychowicz et al., 2017) they update the networks after each *cycle* for $40$ iterations, where each cycle consists of $16$ episodes and each epoch consists of $50$ cycles. As we mentioned in Section 5.2.1, the authors of IPG update their off-policy critic using the same number of iterations as the batch size. This number is the same number of updates vanilla DDPG performs, but rather than updating after each time-step they perform the updates in an accumulated manner after each batch of data is collected. Using the same logic, we treat one batch collection as one *cycle*, therefore we use $40$ training iterations per batch collection for our off-policy hindsight critic to closely imitate the updates of the original HER method (Andrychowicz et al., 2017). Since our method partially utilizes PPO, we switched the number of cycles with the number of episodes, such that each cycle contains more data for PPO to train on. In the case of vanilla HER, each cycle contains $16$ episodes, which corresponds to a total of $16 \times 50 = 800$ time-steps (the time-step horizon of each episode is set to $50$). Instead of using $50$ cycles per epoch and $16$ episodes per cycle, we use $16$ cycles per epoch and $50$ episodes per cycle. This results into the PPO batch update to contain a total of $50 \times 50 = 2500$ transitions.

The whole process of hindsight PPGI is similar to that of PPGI in Algorithm 2 except that the off-policy buffer $B$ is populated with hindsight transitions, therefore the off-policy critic update in line 10 and the off-policy update in the right hand side of the equation in line 21 utilize hindsight transitions. Each hindsight goal $g'$ corresponds to the achieved position of the agent in a state during the episode simulation. Similar to Eq. 22, the hindsight reward is calculated as follows:

$$\mathcal{R}(s_t, g') = \begin{cases} 0, & \text{if } d(\psi(s_t), \psi(s')) \leq \delta \\ -1, & \text{otherwise} \end{cases} \tag{38}$$

where $g' = \psi(s')$, $s_t$ is the current state, $s'$ is a future state that occurred after time-step $t$ (chosen randomly), $d(.,.)$ denotes the Euclidean distance between two goals and $\psi(.)$ is a

function that projects an observation from the state-space $S$ to the goal-space $G$.

# 6 Experiments

## 6.1 Experimental Setup

In this section we provide experimental results that answer the following questions:

1. Does our PPGI method extend to more control tasks than those in IPG (Gu et al., 2017)?
2. Does our PPGI method outperform IPG?
3. What is the optimal interpolation parameter $\nu$ for dense reward settings?
4. Does the control variate improve PPGI?
5. What is the optimal interpolation parameter $\nu$ for PPGI with HER for sparse reward settings?
6. How does PPGI with HER perform in complex goal-based environments?

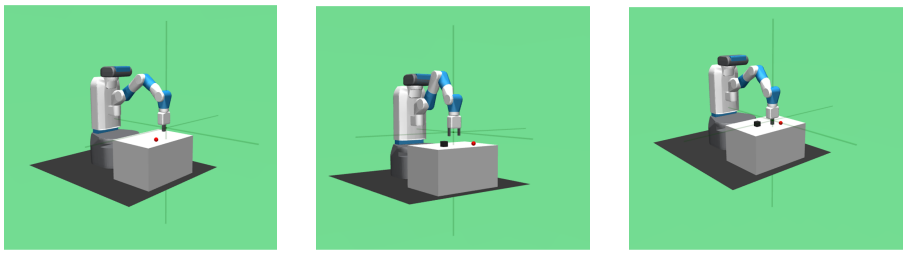
To answer the first question, we use 5 locomotion tasks from OpenAI gym (Brockman et al., 2016). The environments include Hopper, Walker2d, Reacher, InvertedPendulum and InvertedDoublePendulum (see Fig. 14 in Appendix). The agents in these environments are 3D models with multiple joints where the task is to achieve control and move in a forward direction. For the second question, we compare our findings with the results in Gu et al. (2017). To answer the third and fourth questions we use two of the most common environments, HalfCheetah and Ant. For the third question, we run a sweep of $\nu \in \{0.2, 0.4, 0.6, 0.8\}$ the same way as in Gu et al. (2017) to see if we can get similar results. To answer the fifth question, we use a simple goal-based environment called FetchReach. To answer the last question, we use two complex goal-based environments that were used in the original HER experiments in Andrychowicz et al. (2017), FetchPickAndPlace and FetchPush, and try different values of $\nu$ for each task to observe the effect of interpolation between hindsight off-policy updates and non-hindsight on-policy updates.

The tasks for the goal-based environments are the following:

- **FetchReach** (Fig. 4a): A fetch robot needs to navigate to a given goal position at $(x, y, z)$ coordinates.
- **FetchPickAndPlace** (Fig. 4b): A fetch robot needs to pick up a solid block and place it on a given goal position at $(x, y, z)$ coordinates.
- **FetchPush** (Fig. 4c): A fetch robot needs to push a solid block to a given goal position at $(x, y, z)$ coordinates.

## 6.2 Experimental Results for Dense Reward Settings

In this section we show our empirical results of our PPGI method on dense reward tasks.

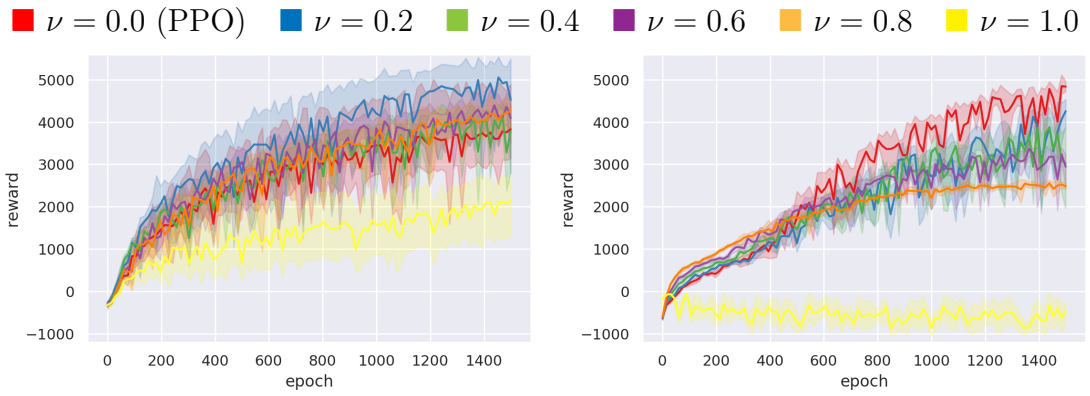


*(a) FetchReach* *(b) FetchPickAndPlace* *(c) FetchPush*

***Figure 4:*** *The environments in this figure are goal-based environments designed specifically Goal-conditioned RL. The red sphere in the figures represent the goal position that the Fetch robot needs to navigate to. These environments also belong to the OpenAI gym [5] framework.*

### 6.2.1 Optimal $\nu$ value

We test our method for the optimal interpolation parameter in two locomotion environments, where the agent is controlled by 2 legs with 4 actuated joints (HalfCheetah) and by 4 legs with 8 actuated joints (Ant). The task is to make the agent walk in a forward direction as fast as possible. We run a parameter sweep for $\nu \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. We see in Fig. 5a that we manage to improve PPO ($\nu = 0$) when $\nu = 0.2$. This matches the results of IPG where their optimal value for $\nu$ is $0.2$ for the HalfCheetah environment. For the special case of $\nu = 1$ (see Section 5.2.1), we see that in both environments in Fig. 5 it provides the worst performance. This contrasts the results in IPG, where their method for $\nu = 1$ outperforms both TRPO and vanilla DDPG in the Ant environment (IPG with $\nu = 1$ is similar to our DDPG variant in Section 5.2.1). This shows that trust region updates might be more crucial for the stability of the algorithm, rather than the clipping method of PPO. Another explanation might be that the hyper-parameter combination of the off-policy critic with our PPGI setting is not optimal for every task.

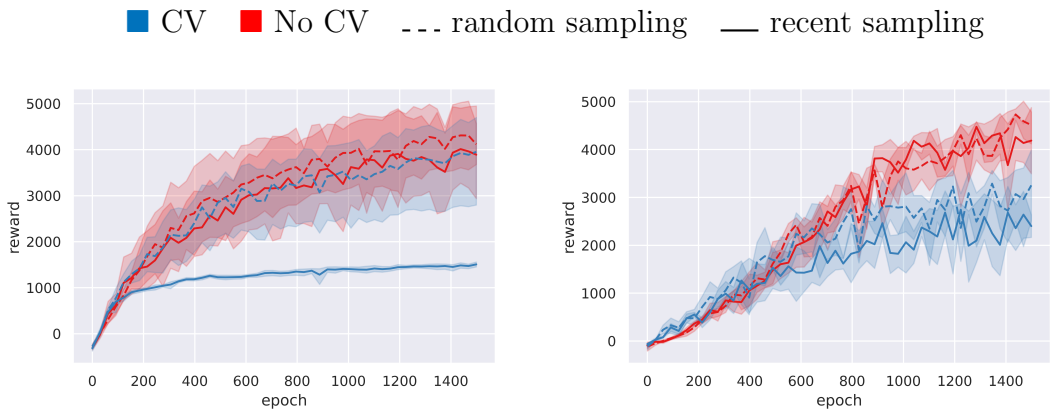



*(a) HalfCheetah* *(b) Ant*

***Figure 5:*** *Results for the $\nu$ variants. Fig. 5a displays the results for HalfCheetah and Fig. 5b the results for Ant. The y-axis indicates the total reward. The x-axis shows the epochs, where each epoch includes 4 episodes and each episode contains 1000 time-steps. Both plots are averaged over 5 seeds.*

### 6.2.2 Control Variate and $\eta$

We evaluate PPGI with and without the control variate in Eq. 37. In addition, we evaluate different update methods $\eta$ for the off-policy part of the policy update rule in Eq. 36. Our two settings of $\eta$ include updating either with random samples (dashed line in Fig. 6) or with recent samples (straight line in Fig. 6) from the replay buffer.




***(a)*** *HalfCheetah* ***(b)*** *Ant*

***Figure 6:*** *Effect of the control variate and sampling strategy $\eta$ on the HalfCheetah and Ant environments. For this experiment, $\nu$ is set to 0.2 and the off-policy batch size is the same as the on-policy one, which is 4000 samples (4 episodes of 1000 time-steps each). The results are averaged over 5 seeds.*

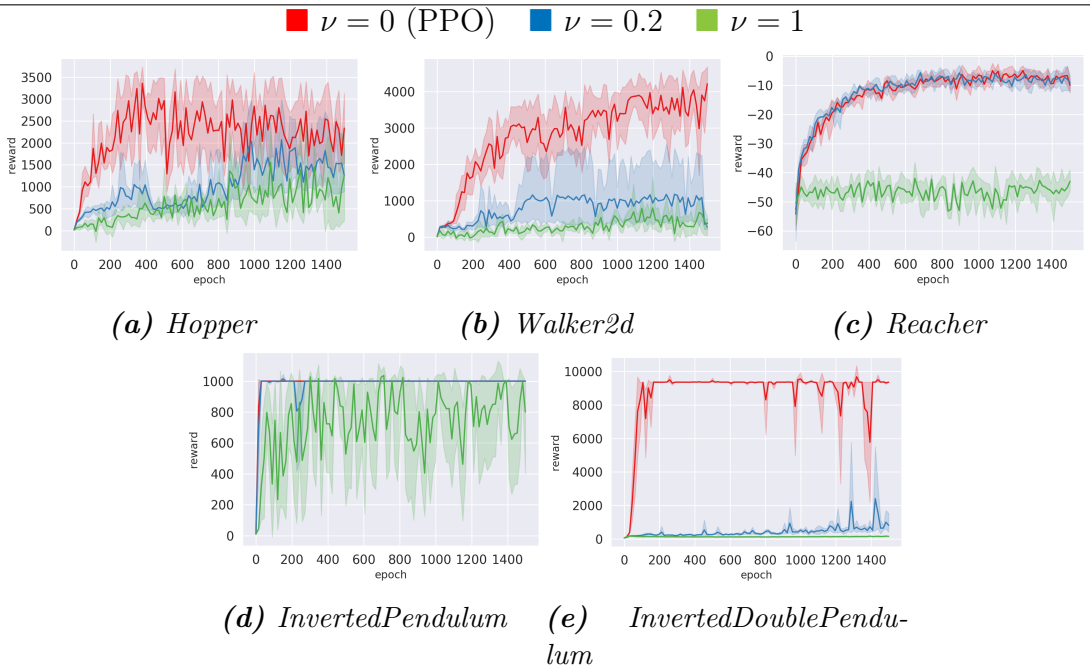
The GAE (Schulman et al., 2015) used by PPO already provides strong variance reduction for stochastic policies. In Fig. 6a and Fig. 6b we see that using the control variate as a baseline to the GAE estimator (lines 15, 16 in Algorithm 2) alters the learning signals in a way that deteriorates the performance of our method. In addition, we see that for either configuration of the CV, random transitions are superior than recent transitions. A possible explanation for this is that the policy over-fits due to the absence of experience diversity. Similar findings were mentioned in Mnih et al. (2013), where they show that breaking the correlation of the transitions by random sampling reduces the variance of the estimator.

### 6.2.3 Extension to More Control Tasks

To better evaluate our algorithm, we run it on 5 additional locomotion control tasks from Brockman et al. (2016). We compare the cases of completely off-policy ($\nu = 1$), completely on-policy ($\nu = 0$) and interpolation with $\nu = 0.2$, which is the best performing value of $\nu$ in Fig. 5a. In Fig. 7 we see that our method can be extended to some of the tasks. With some interpolation, it can perform sub-optimal in one case (Fig. 7a), perform similar to PPO in two cases (Fig. 7c, 7d) and perform poorly in two cases (Fig. 7b, 7e).

## 6.3 Experimental Results for Sparse Reward Settings

In this section we show results of the PPGI method combined with the HER method. We evaluate under sparse reward settings on 3 goal-based robotic tasks.




*(a) Hopper* *(b) Walker2d* *(c) Reacher*

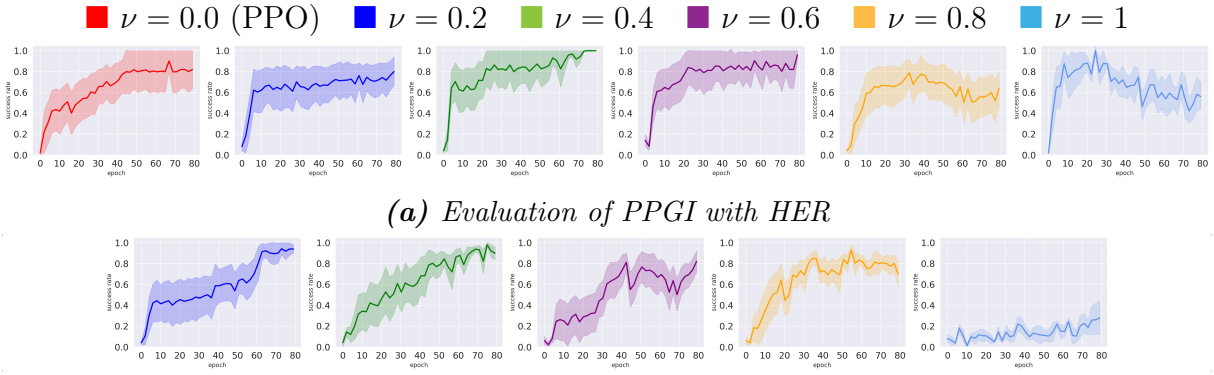
*(d) InvertedPendulum* *(e) InvertedDoublePendulum*

***Figure 7:*** *Results of PPGI on 5 locomotion tasks. The y-axis indicates the total reward and the x axis shows the epochs. Each epoch consists of 4 episodes and each episode contains 1000 time-steps. The results are averaged over 5 seeds.*

### 6.3.1 PPGI with Hindsight Interpolation

To evaluate our hindsight PPGI method, we measure its performance with different interpolation parameters on the FetchReach task. Since interpolation can improve the performance of PPO in some cases (Fig. 5a), we show results with and without HER, where we sample uniformly from the replay buffer. This gives us a better intuition on the effect of HER on PPGI. The results are shown in Fig. 8, where the top row (Fig. 8a) displays PPGI with hindsight and the bottom row (Fig. 8b) displays the results without hindsight. For $\nu = 1$ we see that PPGI without hindsight learns nothing while hindsight PPGI quickly picks up learning. In the case of PPO (where $\nu = 0$), HER has no effect as the contribution of the off-policy critic to the policy update is zero. Despite this, we see that the algorithm converges at around 40 epochs. On the other extreme case (where $\nu = 1$) we notice that it quickly reaches close to the maximum success rate ($\sim$ 8 epoch) but it never converges because as it keeps training the performance deteriorates. A possible explanation for this is that the off-policy critic starts over-fitting, therefore an earlier stopping criteria can be used to address this issue. In Fig. 8a, as we start increasing the interpolation parameter, we notice that our method combines the best out of the two extreme cases. When $\nu$ is close to complete PPO updates (where $\nu = 0.2$ and $\nu = 0.4$), we see that we get the early quick boost that $\nu = 1$ provides while still managing to converge without deteriorating the performance, which is an effect of our method when $\nu = 0$. As we get closer to completely off-policy updates (where $\nu = 0.8$), we notice that the convergence rate decreases, getting closer to not converging at all. The result with hindsight where $\nu = 0.6$ shows how efficient and robust PPO is. Despite the fact that the majority of the policy updates follow the off-policy critic, it still manages to keep the convergence property of PPO. We can conclude the same from the results without hindsight. In all of the graphs in Fig. 8b we notice that the performance in all of the interpolation cases is carried completely

by the PPO updates, even when $\nu = 0.8$, while $\nu = 1$ barely passes the success rate of $0.3$.



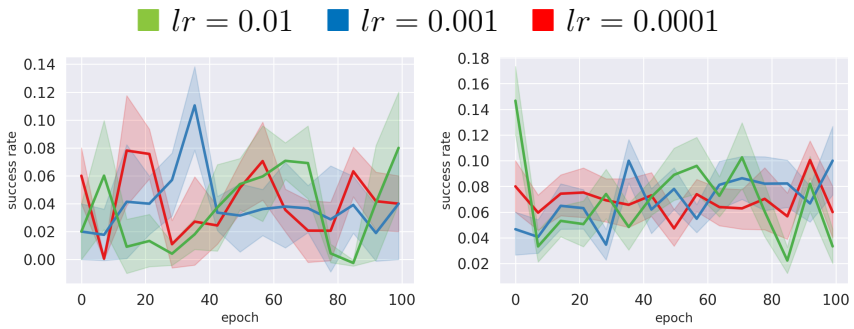

***(a)*** *Evaluation of PPGI with HER*

***(b)*** *Evaluation of PPGI without HER*

***Figure 8:*** *Results of PPGI when combined with HER (Fig. 8a) and without HER (Fig. 8b) on the FetchReach environment. The y-axis indicates the success rate and the x-axis the epochs. Each epoch consists of 10 cycles and each cycle consists of 50 episodes of 50 time-steps each. The off-policy batch size is set to 256. The results are averaged over 5 seeds.*

### 6.3.2 Hindsight PPGI on Complex Tasks and Trajectory Analysis

We evaluate our method on two complex goal-based environments. In contrast with our results in Fig. 8a, our method performs poorly in more complex tasks. Fig. 9 shows the results for $\nu = 1$ in the complex goal-based environments FetchPush (Fig. 4c) and FetchPickAndPlace (Fig. 4b) with different learning rates for the off-policy critic, an off-policy batch size of $256$ and $100$ epochs. In both environments, the average success rate is less than $0.12$ regardless of the learning rate value.

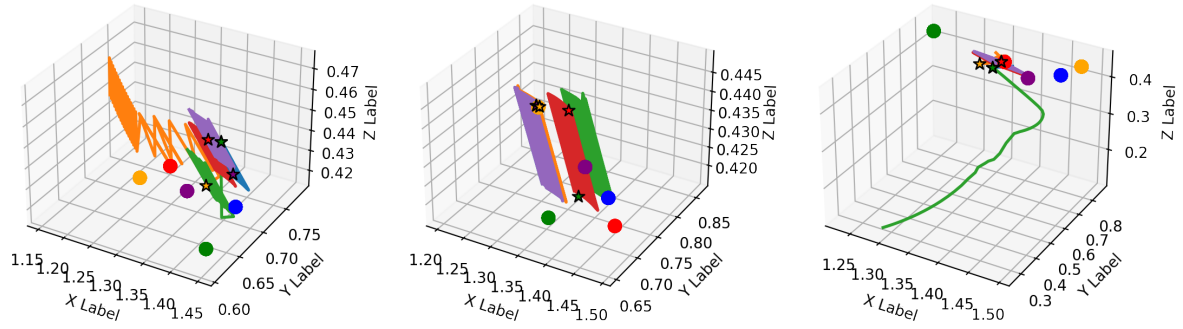



***(a)*** *FetchPickAndPlace* ***(b)*** *FetchPush*

***Figure 9:*** *Results of our hindsight PPGI method for $\nu = 1$ in the FetchPush and FetchPickAndPlace complex environments. Each epoch consists of 16 cycles and each cycle consists of 50 episodes of 50 time-steps each (1 epoch = 800 episodes = 40,000 time-steps). The off-policy batch size is 256 and the results are averaged over 5 seeds.*

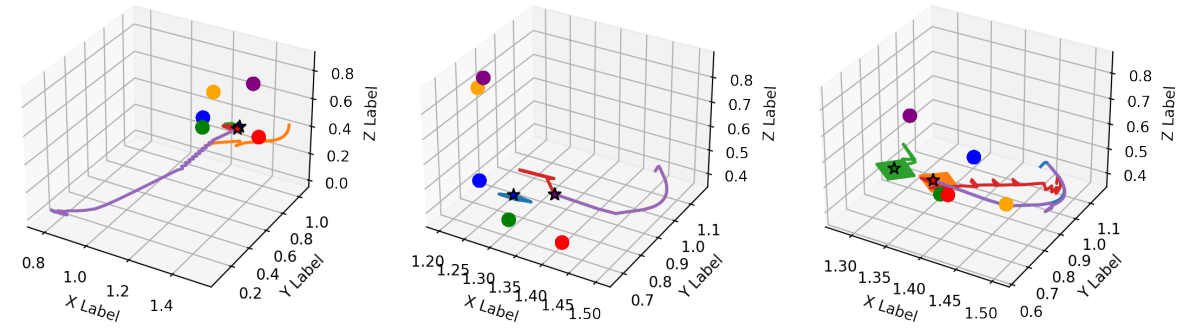
In this section, we analyze the trajectories of each complex environment in different parts during training, given the same hyper-parameter configuration that was used for the FetchReach environment in section 6.3.1, in order to get a better intuition about our method's poor performance.

***FetchPush Environment***




*(a) After 1400 updates* *(b) After 1500 updates* *(c) After 1600 updates*

***FetchPickAndPlace Environment***




*(d) After 1400 updates* *(e) After 1500 updates* *(f) After 1600 updates*

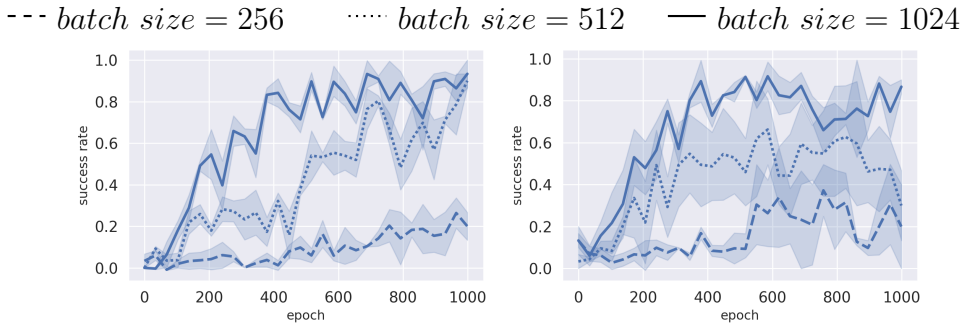
***Figure 10:*** *Results of our trajectory analysis, where the top row is the FetchPush environment and the botom row is the FetchPickAndPlace environment. Each color represents an episode run. The star marker represents the start position, the circle represents the desired goal of the episode and the line represents the trajectory. We plot the paths during evaluation using the mean output of our stochastic policy. The trajectories are based on the robot's grip position and not the achieved goal. For plotting, we use the default hyperparameter settings, which include a* $\sigma = 0.6$*, off-policy critic* $learning\_rate = 0.001$ *and* $batch\_size = 256$*.*

In Fig. 10 we show 5 sequential trajectories per plot every 100 policy updates, where each trajectory is plotted after 1 policy update. For example, Fig. 10a and Fig. 10d contain trajectories at updates $1401, 1402, 1403, 1404$ and $1405$. We plot the last 3 cases per environment. Each line corresponds to a trajectory, each circle corresponds to the desired goal for the line with the same color and the star symbol represents the start position of each trajectory. For visibility purposes, the $x, y, z$ limits of each graph are different such that it suits the trajectories that are displayed. For the FetchPush environment, we see in each case that the policy learns and performs similar paths, regardless of the start state distribution or the desired goals. In Fig. 10a and Fig. 10b we see that the fetch robot performs fast paced left-right or up-down moves, causing the paths to look like they have a rectangular shape. In Fig. 10c, Fig. 10d and Fig. 10e, some paths completely cover the others, due to the fact that they have a very similar trajectory. In Fig. 10f we see that in two cases, the paths have a rectangular shape, and

one case where the purple path completely overlays the blue path. Depending on the number of updates, the policy learns a couple of paths and keeps repeating them, regardless of the goal positions. This indicates over-fitting and is usually due to the lack of exploration. As we show in Appendix A.5, exploration doesn't seem to be the issue for the batch size of $256$. In addition we show that the learning rate of the off-policy critic and the standard deviation don't influence the performance. We run more experiments with different off-policy batch sizes and we manage to pick some learning with a success rate of $0.23$ for the FetchPush environment with an off-policy batch size of $1024$ and a learning rate of $0.001$. We also try an exploration strategy (SE in Table 2) for the batch size of $1024$ which shows some learning, but we manage to get a slightly better performance with a standard deviation of $0.4$ for our stochastic policy. This indicates that our approach requires more data in order to learn, in contrast with the original HER method (Andrychowicz et al., 2017) which requires 100 epochs to reach at least a sub-optimal performance. In the following section we explain why this is the case and show empirical results.

### 6.3.3 Hindsight PPGI on Complex Tasks with More Data

So far we have covered the complex tasks with a maximum of 100 epochs, which corresponds to a total of $4,000,000$ time-steps. In this section, we run experiments for 1000 epochs, which corresponds to $40,000,000$ total time-steps. Due to limited time, we choose a working hyper-parameter configuration (similar to the one from Section 6.3.1) and only run experiments with a different batch size, where we set $\nu = 1$ and the standard deviation at $0.4$.




***(a)*** *FetchPickAndPlace* ***(b)*** *FetchPush*

***Figure 11:*** *Results of our hindsight PPGI method for $\nu = 1$ in the FetchPush and FetchPickAndPlace complex environments with different batch size for the off-policy critic and more epochs. The results are averaged over 3 seeds. In contrast with Fig. 9, we see that the main hyperparameters that heavily influence the performance is the batch size and the number of epochs. The learning rate here is set to $0.001$.*

Our results in Figure 11 indicate that the main hyper-parameter that vastly influences the performance given more data, is the batch size. For both environments we see that the larger the batch size gets, the performance increases. With the batch size of $256$ (dashed line), FetchPush (Fig. 11b) starts learning at around $500$ epochs while FetchPickAndPlace starts learning at around $400$ epochs. In both cases the maximum success rate doesn't exceed $0.3$. For the batch size of $512$ (dotted line), we see a significant improvement in both environments, where the agent starts learning at around $100$ epochs. In Fig. 11b at $800$ epochs we see a similar effect to Fig. 8 with $\nu = 1$, where the performance starts getting worse as training progresses.

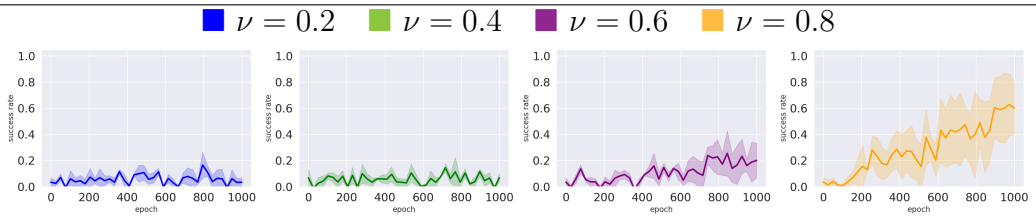
For the batch size of $1024$ we see in FetchPickAndPlace (Fig. 11a) that the learning starts at around $75$ epochs and for FetchPush (Fig. 11b) the learning starts at around 50 epochs. In both environments, the progress with the off-policy batch size of $1024$ is significantly better throughout the training process in contrast with the other batch size values. The original HER method (Andrychowicz et al., 2017) shows optimal results with a batch size of $256$, while our approach needs a larger batch size. The reason behind this has to do with our hindsight design choice. As we mentioned in Section 5.3, our method uses more transitions per cycle and less cycles per epoch in contrast with the original HER method (Andrychowicz et al., 2017). Therefore, a larger batch size is needed in order to match the performance of the original HER method since we update the networks less frequently and with more data gathered. Another possible solution is to train the off-policy critic for more iterations instead of using a larger batch size or a combination of both.

As we managed to make our hindsight DDPG variant from Section 5.2.1 work, in the following section we proceed and introduce interpolation with non-hindsight on-policy updates.

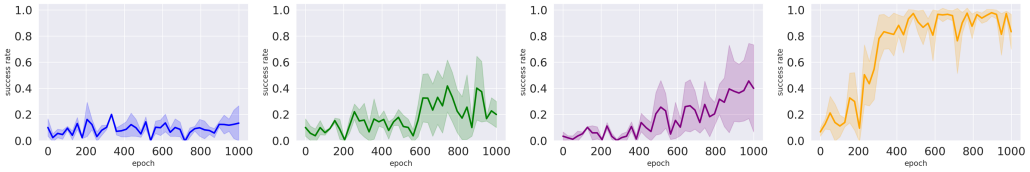
### 6.3.4 Hindsight Interpolation Effect on Complex Tasks

We believe that a larger batch size ($2048$ and possibly $4096$) can further improve the performance in Fig. 11. Since we want to examine the effect of interpolation and whether it can improve the traditional HER method, we use the batch size of 1024 for the following experiments, as optimal performance is beyond the scope of this thesis. The results of hindsight PPGI with interpolation are shown in Fig. 12. As expected, the closer the interpolation is to completely off-policy updates ($\nu = 1$) the better it performs. For the FetchPickAndPlace environment (Fig. 12a) we only see some learning occurring with $\nu = 0.6$ and $\nu = 0.8$, with the latter performing significantly better. In comparison with $\nu = 1$ and the off-policy batch size of $1024$ in Fig. 11a, interpolation doesn't seem to improve complete off-policy updates. For the FetchPush environment (Fig. 12b) we see some learning occurring for $\nu = 0.4$, $\nu = 0.6$ and $\nu = 0.8$, where $\nu = 0.8$ performs significantly better than the rest $\nu$ values. In contrast with the FetchPickAndPlace environment, in the case of FetchPush, interpolation with $\nu = 0.8$ improves the case of $\nu = 1$ with $1024$ off-policy batch size in Fig. 11b. With $\nu = 0.8$, the success rate is more stable, sustainable and after $500$ epochs it is very close to $1$, while the success rate in Fig. 11b is more unstable and only passes the success rate of $0.9$ in certain points during training without sustaining its performance. This matches the results of the FetchReach task in Fig. 8, where the inclusion of on-policy updates improved the stability and the sustainability of the success rate.

The results of different interpolation parameters for the complex tasks in Fig. 12 vastly differ from the results for the simpler FetchReach task in Fig. 8. While the positive effect of interpolation in the simpler task is obvious, it is not the case for the complex tasks. This shows that the difficulty of the environment plays a crucial role for the performance of our method. For a static interpolation approach (meaning that the interpolation parameter is the same during the whole training process), the positive effect is apparent only when PPO ($\nu = 0$) has a chance to learn the task. In complex environments with sparse reward settings, it is hard to pick up a learning signal that would trigger learning by accident. PPO's performance doesn't scale for complex sparse-reward setting environments as it does in dense reward environments. This indicates that while hindsight PPGI with interpolation can improve the traditional HER method in some cases (i.e Fig. 8 and Fig. 11b), it has a limit when it comes to more complex




***(a)*** *FetchPickAndPlace*




***(b)*** *FetchPush*

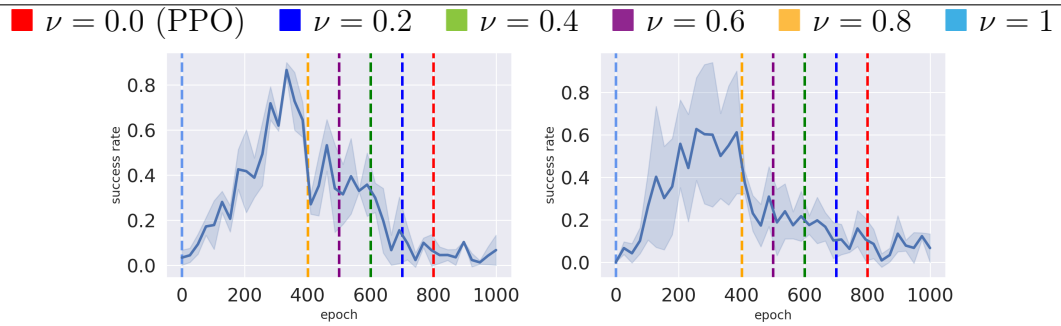
***Figure 12:*** *Results of PPGI when combined with HER for two complex environments. Each epoch consists of 16 cycles and each cycle consists of 50 episodes of 50 time-steps each. The off-policy batch size is set to* 1024*. The results are averaged over 3 seeds.*

tasks.

### 6.3.5 Hindsight Interpolation Annealing

In Sections 6.3.1 and 6.3.4 we showed our method's results with hindsight and different interpolation parameters that were static throughout the training process. As we discussed in Section 6.3.4, our method has its limitations when it comes to more complex environments. In this section, we evaluate our method with an annealing value of $\nu$ for the complex tasks. We want to see if annealing the interpolation can deteriorate our method's limitations by starting with complete off-policy updates and slowly decreasing it to complete on-policy (non-hindsight) updates. Our setting is the following: we train the agent with $\nu = 1$ until a satisfactory success rate is reached. Following, every 100 epochs, we start decreasing the value of $\nu$ by $0.2$ until we reach complete on-policy updates ($\nu = 0$), where training continues until termination. Specifically, according to Fig. 11, we believe that in both environments, a satisfactory success rate is reached at $400$ epochs, therefore that's when we start decreasing the interpolation parameter. At epoch $800$, $\nu$ reaches the value of 0, where it keeps training for $200$ more epochs.

The results of the interpolation annealing are shown in Fig. 13. For both environments, when annealing starts the performance of the agent vastly deteriorates. The lower the value of $\nu$ gets, the worse the performance becomes. This indicates that non-hindsight on-policy updates can't perform satisfactory in goal-based environments with sparse reward settings even with the help of an off-policy hindsight critic. Our method can perform optimal with a static value of $\nu$ throughout the training process, as in Fig. 8a and Fig. 12. While some interpolation can improve complete off-policy hindsight updates (as in Fig. 12b for $\nu = 0.8$), the change in the update rule during training seems to affect the learning process negatively even when the value of $\nu$ is close to complete off-policy updates ($\nu = 0.8$). Another explanation we have regarding this is that $100$ epochs of training during the interpolation change is not enough for our method to recover from its drop in performance. Regardless, more training iterations beats the purpose of this thesis, which is to improve complete off-policy hindsight updates by interpolation. A better distribution of annealing or even smaller change in the value of $\nu$




***(a)*** *FetchPickAndPlace* ***(b)*** *FetchPush*

***Figure 13:*** *Results of hindsight PPGI with annealing interpolation for two complex tasks. We start by complete off-policy hindsight with $\nu = 1$ for the first 400 epochs. Afterwards, we decrease the value of $\nu$ by 0.2 every 100 epochs. From epoch 800 to 1000 we have the PPO method ($\nu = 0$). We use the same configuration as in Fig. 12. The results are averaged over 3 seeds.*

throughout training could have better results, but we strongly believe that the performance in contrast with $\nu = 1$ would still get worse with the value of $\nu$ decreasing.

# 7 Related Work

## 7.1 Hybrid RL algorithms

Our work builds upon the work of QProp (Gu et al., 2016) and PGQL (O'Donoghue et al., 2016). PGQL is a similar method to IPG (Gu et al., 2017), where it combines policy gradient updates with Q-learning updates through an interpolation parameter $\nu$. In contrast with IPG, O'Donoghue et al. (2016) derive a method that can estimate the off-policy critic using the policy. QProp is the predecessor of IPG. Similar to its successor, QProp combines the stability of on-policy methods and the sample efficiency of off-policy methods. The main contribution of QProp is that it uses an action-value function that is trained off-policy as a control variate for Monte Carlo policy gradients. While IPG focuses more on interpolating between off-policy and on-policy updates, QProp focuses more on the control variate aspect. Overall they are similar methods and they both achieve high performance when they combine TRPO with DDPG and the QProp control variate.

ACER (Wang et al., 2016) is another hybrid method built on the A3C framework (Mnih et al., 2016), that addresses the problem of sample efficiency and application in both discrete and continuous action spaces. It uses the Retrace estimator (Munos et al., 2016) to estimate the off-policy critic, a clipped importance sampling weight to ensure bounded variance and trust region updates, specifically TRPO (Schulman et al., 2015). Policy-on Policy-off Policy Optimization (P3O) (Fakoor et al., 2020) considers the fact that previous work on hybrid approaches require a lot of tuning, such as the $\nu$ parameter in IPG and the clipping threshold in ACER. Their update method combines on-policy gradient, the off-policy gradient of ACER with the truncated importance weight and the KL divergence between the behavioural and target policy multiplied with a regularization coefficient. They manage to automate their main hyperparameters, the clipping threshold and the KL regularization coefficient by using a variant

of the Effective Sample Size (ESS) method derived by Kong (1992).

## 7.2 Addressing sparse rewards

Hindsight Experience Replay can be viewed as a form curriculum learning (CL) (Bengio et al., 2009). CL approaches address the sparse reward problem by creating easier sub-tasks which the agent can achieve at its current state of learning. Ideally, as training progresses, the sub-tasks come closer to the desired task. Recent methods, including HER, have managed to automate the process of generating sub-tasks. Florensa et al. (2017) propose a reversed curriculum technique that alters the start state distribution. They start by generating start states close to the desired goal and as the agent improves, they alter the start distribution such that it samples states further away from the desired goal. Held et al. (2018) use a variant of the Generative Adversarial Networks (GAN) (Goodfellow et al., 2014; Mao et al., 2017) that is trained to generate sub-tasks of appropriate difficulty for the agent. Curriculum-guided HER (Fang et al., 2019) improves HER by introducing a curriculum sampling technique that balances the trade-off between the proximity and the diversity of the achieved goals. Other methods achieve automatic curriculum through self-play. Sukhbaatar et al. (2017) use an agent with two different set of parameters and objectives, one that proposes a task of appropriate difficulty and another one that tries to achieve it. Through self-play and internal rewards, this method achieves efficient environment exploration and learns to achieve the target task faster. Liu et al. (2019) introduce another HER enhancement. They propose a multi-agent approach for single based environments that re-labels goals through a self-play competition in a way that encourages exploration.

## 7.3 On-policy hindsight approaches

To our knowledge, there are only two methods that combine hindsight with on-policy approaches, Hindsight Policy Gradients (HPG) (Rauber et al., 2017) and Hindsight TRPO (HTRPO) (Zhang et al., 2019). In contrast to our method, they apply hindsight on completely on-policy approaches. HPG introduces the idea that a sampled trajectory given a desired goal can be evaluated under an achieved goal using importance sampling. HTRPO extends the HPG method to TRPO, where they use a variant of the KL-divergence that further reduces the variance that comes from the evaluation of the trajectories under different tasks. They further improve their method by a technique called *hindsight goal filtering*, which chooses hindsight goals that are close to the original goal region. The advantage of our method over HPG and HTRPO is that we utilize the benefits of off-policy updates as well rather than just on-policy. With the success of HER with off-policy methods, we believe it is crucial to include off-policy hindsight updates for an optimal performance, since off-policy HER outperforms both HPG and HTRPO.

# 8 Discussion

In this work we described a method that's able to combine off-policy hindsight with the robustness of on-policy methods. We first tested our method without hindsight on 7 locomotion tasks with a dense reward setting in order to observe the performance of the interpolation between off-policy and on-policy updates. Our results showed that interpolation doesn't outperform complete on-policy updates as it performed sub-optimal in most cases and only outperformed

it in one case. We attempted to further reduce the variance of our method by using a control variate, but it deteriorated the performance. Afterwards, we introduced hindsight to the off-policy part of our method and tested it on 3 goal-based tasks with a sparse reward setting. On the simplest goal-based environment, we showed that interpolation improved both complete off-policy and on-policy cases by combining both of their advantages. Our on-policy part reaches a high success rate slowly and converges. Our off-policy part reaches a high success rate fast, but as training progresses the success rate deteriorates. During interpolation, we showed that our method can reach a high success rate fast, which is a feature of the off-policy part, and converge as training progresses which is a feature of the on-policy part. Finally, we evaluated our method on 2 complex goal-based environments with $100$ epochs and analyzed the agent's trajectories during different stages of training. The trajectories indicated that the policy learns specific paths and aims to reproduce them regardless of the start state distribution and the desired goal-position, which resulted into a poor performance. By using more epochs and a larger off-policy batch size for the complex tasks, we managed to achieve a good performance in our complete hindsight off-policy setting as well as improve it with some interpolation for the FetchPush environment. Moreover, we tried a different setting for the complex tasks, where we slowly annealed the interpolation parameter throughout the training process. We believed that doing so, the on-policy updates would benefit more from the hindsight off-policy critic and perform better. Our results showed that interpolation annealing doesn't perform optimal in contrast with a static interpolation throughout training, which improved the performance in two out of three goal-based environments.

The drawbacks of our approach is that it combines two methods, the hybrid RL algorithm and hindsight, both of which are hard to configure. HER is very sensitive to hyper-parameters and normalization (see Appendix A.4) and hindsight PPGI's performance highly depends on different interpolation parameters, depending on the task. As we showed in our results for the dense-reward setting, interpolation can either outperform, perform similar or perform worse than the complete on-policy case, depending on the environment and the interpolation parameter. In addition, in the sparse-reward setting, we achieved successful performance in 2 goal-based environments and poor performance in one using different hyper-parameter configurations. We also showed that our method can reach its limit depending on the environment complexity. To that extend, we believe that our approach can be limiting and unable to generalize among different tasks with the same or similar hyper-parameter configurations, especially the interpolation amount.

We left the implementation of hindsight on the on-policy part of our method for future work. This can be achieved by using techniques that were utilized in HPG or HTRPO (Rauber et al., 2017; Zhang et al., 2019). Other future directions include the automatic tuning during training for the interpolation parameter, the use TD3 techniques (Fujimoto et al., 2018) in order to improve the off-policy part of our method and non-random hindsight sampling methods, like curriculum guided HER (Fang et al., 2019) and prioritized experience replay (Schaul et al., 2015). Another direction we find interesting is how would our method perform with curriculum-learning approaches, such as reverse curriculum generation (Florensa et al., 2017) or automatic goal generation (Held et al., 2018). We are also curious about our method's performance in more complex tasks, such as Maze Ant. Liu et al. (2019) show sub-optimal results of the vanilla HER in such complex tasks. It would be interesting to see how our method would perform in comparison with their findings. Lastly, our method's components are orthogonal, therefore any on-policy algorithm can replace PPO, such as ACKTR (Wu et al., 2017) and

any off-policy algorithm that utilizes a replay buffer can replace our variant of DDPG, such as TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018).

# 9 Conclusion

We propose a method that introduces hindsight to on-policy approaches by interpolating between on-policy and off-policy updates using a method that we call PPGI. We first evaluate PPGI without hindsight on 7 locomotion tasks. While interpolation slightly deteriorates the performance in most cases, it allows us to use hindsight on the off-policy critic and evaluate it on goal-based environments. The off-policy critic is trained with hindsight data in order to partially direct the policy gradient updates. Empirically, we observe that on a simple goal-based environment, interpolation helps and manages to improve the performance by combining features of both on-policy and off-policy methods. In complex environments the effect of interpolation is not that apparent, but it can still improve the performance of complete hindsight off-policy updates in one case out of two. The limitation of our approach is that for different tasks, the optimal interpolation parameter is different, which indicates lack of generalization. Another limitation of our method is that interpolation doesn't improve the performance of complete off-policy hindsight when the task is highly complex. Regardless of our method's drawbacks, our results show that it is promising. To our knowledge, there is no other work that combines hindsight methods with hybrid RL algorithms and we believe that it is a promising path for future research.

# References


[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

[3] David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, and Antonio Torralba. Rewriting a deep generative model. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] Lisandro Dalcín, Rodrigo Paz, Mario Storti, and Jorge D'Elía. Mpi for python: Performance improvements and mpi-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008.

[7] Rasool Fakoor, Pratik Chaudhari, and Alexander J Smola. P3o: Policy-on policy-off policy optimization. In *Uncertainty in Artificial Intelligence*, pages 1017–1027. PMLR, 2020.

[8] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. 2019.

[9] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. 2019.

[10] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR, 2017.

[11] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[14] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine35. Qprop: Sample-efficient policy gradient with an off-policy critic. 2016.

[15] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. *arXiv preprint arXiv:1706.00387*, 2017.

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[17] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. 2018.

[18] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, volume 951, pages 167–173, 1993.

[19] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

[20] Augustine Kong. A note on importance sampling using standardized weights. *University of Chicago, Dept. of Statistics, Tech. Rep*, 348, 1992.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[23] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[24] Hao Liu, Alexander Trott, Richard Socher, and Caiming Xiong. Competitive experience replay. *arXiv preprint arXiv:1902.00528*, 2019.

[25] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[26] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9): 3732–3740, 2019.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[28] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[29] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*, 2016.

[30] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

[31] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[32] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[33] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[34] Paulo Rauber, Avinash Ummadisingu, Filipe Mutz, and Juergen Schmidhuber. Hindsight policy gradients. *arXiv preprint arXiv:1711.06006*, 2017.

[35] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[36] Sheldon M. Ross. *Simulation, Fourth Edition*. Academic Press, Inc., USA, 2006. ISBN 0125980639.

[37] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[38] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[39] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[40] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[41] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[42] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[45] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[46] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[47] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[48] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[49] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[50] Chris Yoon, blog. Chris Yoon understanding actor critic methods. https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f. Accessed: 2021-06-25.

[51] David Silver, blog. David Silver reinforcement learning lectures. https://www.davidsilver.uk/teaching/. Accessed: 2021-06-15.

[52] Lilian Weng, blog. Lilian Weng policy gadient algorithms. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html/. Accessed: 2021-07-09.

[53] Seungjae Ryan Lee, blog. Seungjae Ryan Lee bias-variance tradeoff in reinforcement learning. https://www.endtoend.ai/blog/bias-variance-tradeoff-in-reinforcement-learning/. Accessed: 2021-06-21.

[54] Tianhong Dai, github. Tianhong Dai hindsight-experience-replay. https://github.com/TianhongDai/hindsight-experience-replay. Accessed: 2021-06-25.

[55] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[56] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[57] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[58] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*, 2017.

[59] Hanbo Zhang, Site Bai, Xuguang Lan, David Hsu, and Nanning Zheng. Hindsight trust region policy optimization. *arXiv preprint arXiv:1907.12439*, 2019.

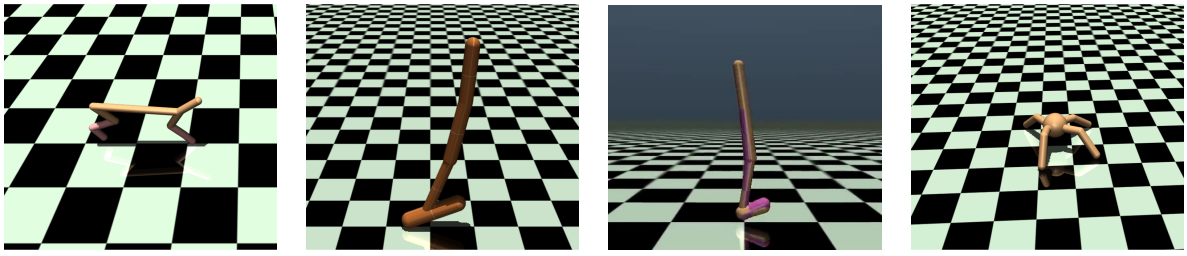
# A Appendix

## A.1 General information of our project

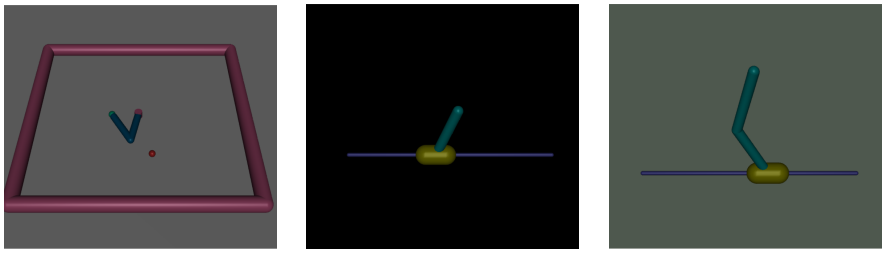
This project was recommended by a "request for research" paper from OpenAI [33]. For the code implementation, we built upon the PPO baseline of OpenAI's spinningup framework [1]. In addition, we used classes for the HER sampling and state normalization from Tianhong Dai's HER project [54]. Our code can be found in here: https://github.com/pavlosSkev/hindsight-pgi.

## A.2 Environments

In this section we give more details about the environments we used in our experiments. The task in Fig. 14a, Fig. 14b, Fig. 14c and Fig. 14d is to go in a forward direction as fast as possible. The task in Fig. 14e is to navigate to the red goal position and the task in Fig. 14f and Fig. 14g is to balance the pole in an upward direction. All of the environments have a continuous state and action space.



*(a) HalfCheetah* *(b) Hopper* *(c) Walker2d* *(d) Ant*



*(e) Reacher* *(f) InvertedPendulum* *(g) InvertedDoublePendulum*

***Figure 14:*** *This figure shows the locomotion environments we used for our experiments. They belong to the OpenAI gym framework [5] and they are implemented with the Mujoco physics engine [55].*

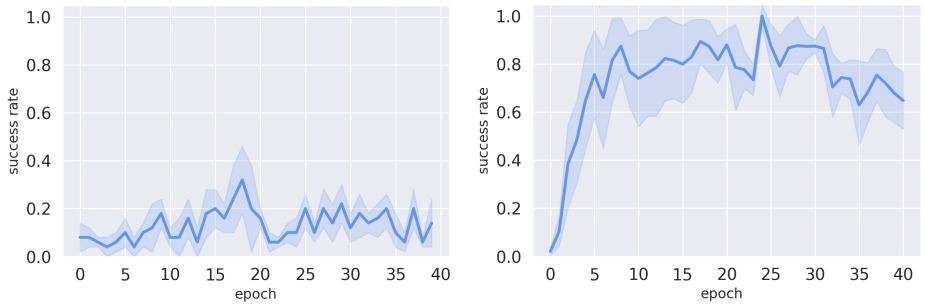
## A.3 Hyperparameters

Our Gaussian policy is a 2-layer neural network with a size of 64 and a Tanh activation function for the hidden-layer output. The on-policy critic is a 2-layer network with a size of 64 and a Tanh activation function. The off-policy critic is a 2-layer neural network with a size of 100 and a ReLU activation function. We use $\gamma = 0.99$, $80$ training iterations for the on-policy actor and critic and GAE with $\lambda = 0.95$. For the off-policy critic, we use $4000$ training iterations for the locomotion tasks and $80$ iterations for the goal-based tasks. We collect 4 episodes of 1000

time-steps each per batch data for the locomotion tasks and 50 episodes of 50 time-steps each for the goal-based tasks. The reason we use different updates for the off-policy critic depending on the task is because for the locomotion tasks we want to mimic the DDPG updates, where the models update after each time-step. For the goal-based hindsight tasks, we want to mimic DDPG with HER [2], where instead of updating after each time-step, they update for multiple iterations after collecting 16 episodes. In our case, we choose 50 episodes in contrast with [2] because PPO requires more data to perform optimal. In addition, we apply normalization only for the hindsight part of our thesis. All of our experiments are implemented with parallelization on 4 CPUs using the MPI framework [6].

## A.4 Effect of Normalization

The input scaling we use is similar to the original one used in the HER method [2]. We utilize a moving average and standard deviation in order to normalize the states such that they have a mean of 0 and standard deviation of 1. The effect of this normalization on our variant of DDPG with hindsight shows in Fig. 15. This is another indicator of how sensitive off-policy algorithms and HER can be. In our case, normalization makes the difference between learning and not learning at all.




***(a)*** *No Normalization* ***(b)*** *Moving Average Normalization*

***Figure 15:*** *Effect of normalization for the FetchReach environment under our hindsight approach when* $\nu = 1$.

## A.5 Hyper-parameter Sweep for Goal-based Tasks

We run a hyper-parameter sweep for the complex goal-based tasks. We try different learning rates for the off-policy critic, different standard deviations for sampling actions, a strong exploration strategy (SE in Tables 1, 2) and a different batch size for training the off-policy critic and for the off-policy part of the update in Eq. 36. For the SE setting we try to replicate the exploration used in the original HER paper in [2]. Their behavioural policy works as follows: with a probability of $80\%$, they use the deterministic action that's output by the actor network and add a small Gaussian noise to it with mean 0 and standard deviation of $0.5$. With a probability of $20\%$, they use a complete random action sampled from the space of valid actions of the environment. We use the same exploration settings but rather than adding noise to a deterministic action, we sample from our parametarized Gaussian distribution with a standard deviation of $0.6$.

Due to limited time and a mistake made when running the sweep, the experiments in both tables for the strong exploration (SE) with the batch size of $512$ are averaged over 3 seeds rather than 5. In addition, for both environments, all of the experiments with a batch size of $256$ are also averaged over 3 seeds instead of $5$.

| Learning rate | | | Standard Deviation | | | SE | Batch Size | | | Success rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.01 | 0.001 | 0.0001 | 0.4 | 0.6 | 0.8 | - | 256 | 512 | 1024 | - |
| ✓ | | | ✓ | | | | ✓ | | | 0.04 |
| ✓ | | | | ✓ | | | ✓ | | | 0.046 |
| ✓ | | | | | ✓ | | ✓ | | | 0.06 |
| ✓ | | | | | | ✓ | ✓ | | | 0.039 |
| | ✓ | | ✓ | | | | ✓ | | | 0.02 |
| | ✓ | | | ✓ | | | ✓ | | | 0.04 |
| | ✓ | | | | ✓ | | ✓ | | | 0.06 |
| | ✓ | | | | | ✓ | ✓ | | | 0.066 |
| | | ✓ | ✓ | | | | ✓ | | | 0.033 |
| | | ✓ | | ✓ | | | ✓ | | | 0.046 |
| | | ✓ | | | ✓ | | ✓ | | | 0.053 |
| | | ✓ | | | | ✓ | ✓ | | | 0.033 |
| ✓ | | | ✓ | | | | | ✓ | | 0.026 |
| ✓ | | | | ✓ | | | | ✓ | | 0.033 |
| ✓ | | | | | ✓ | | | ✓ | | 0.033 |
| ✓ | | | | | | ✓ | | ✓ | | 0.02 |
| | ✓ | | ✓ | | | | | ✓ | | 0.04 |
| | ✓ | | | ✓ | | | | ✓ | | 0.11 |
| | ✓ | | | | ✓ | | | ✓ | | 0.08 |
| | ✓ | | | | | ✓ | | ✓ | | 0.046 |
| | | ✓ | ✓ | | | | | ✓ | | 0.026 |
| | | ✓ | | ✓ | | | | ✓ | | 0.066 |
| | | ✓ | | | ✓ | | | ✓ | | 0.006 |
| | | ✓ | | | | ✓ | | ✓ | | 0.026 |
| ✓ | | | ✓ | | | | | | ✓ | 0.028 |
| ✓ | | | | ✓ | | | | | ✓ | 0.056 |
| ✓ | | | | | ✓ | | | | ✓ | 0.04 |
| ✓ | | | | | | ✓ | | | ✓ | 0.027 |
| | ✓ | | ✓ | | | | | | ✓ | 0.056 |
| | ✓ | | | ✓ | | | | | ✓ | 0.08 |
| | ✓ | | | | ✓ | | | | ✓ | 0.06 |
| | ✓ | | | | | ✓ | | | ✓ | 0.052 |
| | | ✓ | ✓ | | | | | | ✓ | 0.048 |
| | | ✓ | | ✓ | | | | | ✓ | 0.032 |
| | | ✓ | | | ✓ | | | | ✓ | 0.02 |
| | | ✓ | | | | ✓ | | | ✓ | 0.044 |

***Table 1:*** *Results of hyperparameter sweep for the FetchPickAndPlace environment [4b]. The results are averaged over 5 seeds, except the SE configurations for the batch size of 512, except the SE configurations for the batch size of 512 and all the configurations with a batch size of 256.*

| Learning rate | | | Standard Deviation | | | SE | Batch Size | | | Success rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.01 | 0.001 | 0.0001 | 0.4 | 0.6 | 0.8 | - | 256 | 512 | 1024 | - |
| ✓ | | | ✓ | | | | ✓ | | | 0.06 |
| ✓ | | | | ✓ | | | ✓ | | | 0.086 |
| ✓ | | | | | ✓ | | ✓ | | | 0.093 |
| ✓ | | | | | | ✓ | ✓ | | | 0.093 |
| | ✓ | | ✓ | | | | ✓ | | | 0.08 |
| | ✓ | | | ✓ | | | ✓ | | | 0.053 |
| | ✓ | | | | ✓ | | ✓ | | | 0.1 |
| | ✓ | | | | | ✓ | ✓ | | | 0.073 |
| | | ✓ | ✓ | | | | ✓ | | | 0.066 |
| | | ✓ | | ✓ | | | ✓ | | | 0.066 |
| | | ✓ | | | ✓ | | ✓ | | | 0.1 |
| | | ✓ | | | | ✓ | ✓ | | | 0.053 |
| ✓ | | | ✓ | | | | | ✓ | | 0.053 |
| ✓ | | | | ✓ | | | | ✓ | | 0.053 |
| ✓ | | | | | ✓ | | | ✓ | | 0.066 |
| ✓ | | | | | | ✓ | | ✓ | | 0.073 |
| | ✓ | | ✓ | | | | | ✓ | | 0.046 |
| | ✓ | | | ✓ | | | | ✓ | | 0.093 |
| | ✓ | | | | ✓ | | | ✓ | | 0.09 |
| | ✓ | | | | | ✓ | | ✓ | | 0.066 |
| | | ✓ | ✓ | | | | | ✓ | | 0.06 |
| | | ✓ | | ✓ | | | | ✓ | | 0.10 |
| | | ✓ | | | ✓ | | | ✓ | | 0.093 |
| | | ✓ | | | | ✓ | | ✓ | | 0.08 |
| ✓ | | | ✓ | | | | | | ✓ | 0.024 |
| ✓ | | | | ✓ | | | | | ✓ | 0.092 |
| ✓ | | | | | ✓ | | | | ✓ | 0.012 |
| ✓ | | | | | | ✓ | | | ✓ | 0.064 |
| | ✓ | | ✓ | | | | | | ✓ | **0.23** |
| | ✓ | | | ✓ | | | | | ✓ | 0.096 |
| | ✓ | | | | ✓ | | | | ✓ | 0.092 |
| | ✓ | | | | | ✓ | | | ✓ | **0.21** |
| | | ✓ | ✓ | | | | | | ✓ | 0.068 |
| | | ✓ | | ✓ | | | | | ✓ | 0.076 |
| | | ✓ | | | ✓ | | | | ✓ | 0.056 |
| | | ✓ | | | | ✓ | | | ✓ | 0.064 |

***Table 2:*** *Results of hyper-parameter sweep for the FetchPush environment [4c]. The results are averaged over 5 seeds, except the SE configurations for the batch size of 512 and all the configurations with a batch size of 256.*