Improving Sample Efficiency in Hindsight Policy Gradient Using Actor-Critic Architectures

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics

presented by
Avinash Ummadisingu

under the supervision of
Prof. Dr. Jürgen Schmidhuber
co-supervised by
Dr. Paulo Rauber

September 2018

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Avinash Ummadisingu
Lugano, 10 September 2018

Abstract

A reinforcement learning agent that needs to pursue different goals across episodes requires a goal-conditional policy. This setting of learning goal-conditional policies has received significant attention in the past because of their ability to generalize behaviour to goals not-yet encountered. Sparse-reward environments, in particular, are extremely challenging to learn with the agent only receiving a non-zero reward when a goal has been achieved and a zero otherwise. Goal-conditional policies find heavy use in robotics and in the context of goal-based curricula and hierarchical reinforcement learning. It is, therefore, crucial to address the fact that learning in this setting often requires an inordinate number of trails before the agent is able to receive a usable reward signal to begin learning. One way to address this problem is to exploit the information about the degree to which an arbitrary goal has been achieved while trying to achieve another, thereby allowing an agent to learn from its failures as well as its successes. This capability of hindsight, which was recently introduced to goal-conditional policies, appears vital for sample efficient learning. This thesis discusses prior work that proposes and uses hindsight in goal-conditional reinforcement learning. It then presents our work in introducing it to an entire class of highly successful algorithms based on policy gradients as well as novel attempts to improve them using actor-critic architectures. We present experiments carried out on a diverse set of environments which show that hindsight is imperative to learn goal-conditional policies in a sparse-reward setting and demonstrate that our proposed methods reveals a striking improvement in sample efficiency that is further enhanced by the use of actor-critic architectures.

Acknowledgements

I would like to express my deep gratitude to Prof. Jürgen Schmidhuber for giving me the opportunity to undertake my thesis work with him. I am also extremely thankful to Dr. Paulo Rauber for his patience and steadfast guidance throughout my thesis and for the many invaluable lessons I learnt from him during our collaboration. Finally, I would like to thank my family and friends, without whose constant support this work would not be possible.

Contents

Figures

Tables

Chapter 1

Introduction

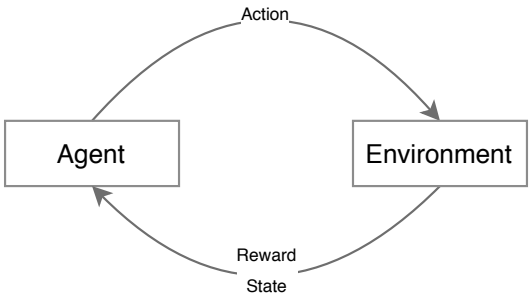

Figure 1.1. Reinforcement Learning framework overview as a sequential decision making task

The goal of *Reinforcement Learning (RL)* is to find an optimal behaviour strategy for an agent in a sequential decision task. At each discrete time-step, an agent interacts with the environment by performing an action, which may alter the environment in some way, as illustrated in Figure 1.1. From the environment, the agent then perceives a new state observation and a scalar reinforcement signal called the reward (indicating success or failure). An RL agent aims to find the optimal sequence of actions based on its observations in order to maximize the total reward received through a learning process of trial-and-error. It is through this process of trial and error that the agent learns to solve the problem of *credit assignment* where it assigns credit (or blame) of a reward to an action or a set of actions.

Reinforcement Learning is both a problem framework and the study of algorithms to solve them. Its general formulation makes few assumptions and can thus be used for a wide variety of problems in social sciences and engineering [Sutton and Barto, 1998] from robotic control [Kober et al., 2013] to automated trading systems [Moody and Wu, 1997], to managing power systems for servers [Tesauro et al., 2008].

Reinforcement Learning in an *episodic* setting focuses on tasks with a notion of terminal states. Upon reaching a terminal state, an agent resets to one of the possible start states and the environment acts independently of all previous interactions. This naturally breaks the in-

teraction between an agent and environment into subsequences called *episodes*. Traditional Reinforcement Learning focusses on learning a *policy* that maps the agent's states to actions in order to pursue some common goal across episodes. If, however, the aim of the agent is to pursue different *goals* across episodes, then a *goal-conditional policy* could instead be learnt which takes into account not only the current state but also the goal that needs to be pursued. We call this form of the reinforcement learning task as *Goal-Conditional Reinforcement Learning*.

Reinforcement Learning is a step more realistic than the more widely studied supervised learning. Given some *training set* $\{(x_i, y_i)\}_{i=1}^{N}$ of input-output pairs, the supervised learning task aims to learn a function that approximately maps inputs x to the (possibly noisy) outputs y. The aim in learning such a mapping of the relationships between the inputs and the outputs is that when given a separate *test set* of inputs which are drawn from the same distribution as the training set, the algorithm would be able to label these points correctly. In its most basic form, the supervised learning problem is solved by choosing some expressive function class (such as a Neural Network), setting up a loss function between the output of the function and the desired output, and then modifying the function's parameters to minimize the loss through a training procedure such as gradient descent. Supervised learning algorithms must possess a strong ability to *generalize* their predictions to new, previously unseen data, drawn from the same distribution as the training set. It is this ability of generalization that makes it strongly suited to be used in Reinforcement Learning generalizing behaviour to previously unseen states based on those it has already encountered. Neural Networks have a long history of being successful supervised learners driving significant improvement in many fields of Machine Learning including Reinforcement Learning [Schmidhuber, 2014].

The combination of Reinforcement Learning and Neural Networks has well-documented successes in the past. Tesauro's TD-Gammon [Tesauro, 1994] is one such example in which an RL agent using a Neural Network learnt to play backgammon at a world-class level through self-play. The powerful combination has also found application in robotics [Lin, 1992b], industrial manufacturing and other combinatorial search problems. The recent resurgence of Neural Networks and the ability to train much larger, deeper networks on GPUs has fuelled rapid advancement in the capabilities of RL systems with the combination often referred to as *Deep Reinforcement Learning*. Deep Neural Networks have proven themselves as powerful function approximators and representation learners. They have elevated RL's capabilities and allowed them to scale to tasks previously considered intractable. A particularly influential instance of this is the work by [Mnih et al., Dec 2013], where a Deep Convolutional Neural Network was used in combination with classical Reinforcement Learning methods to surpass human-level performance in a number of Atari 2600 games, learning directly from the video input and successfully circumventing the curse of dimensionality. Hybrid Deep-RL methods have also been used in conjunction with Monte-Carlo Tree Search, a traditional heuristic search algorithm, to defeat the world champion in the ancient Chinese game of Go [Silver et al., 2016, 2017]. An explosion of interest in Deep Reinforcement learning led to it being applied to myriad tasks including, but not limited to game playing [Mnih et al., Dec 2013; Tesauro, 1994; Silver et al., 2016], robotic controllers [Peng et al., 2017; Levine et al., 2018], large-scale architecture searches to optimize neural networks [Zoph and Le, 2016], simulations for optimization of prosthetics [Pilarski et al., 2011; Seth et al., 2018], etc.

A recurring problem that arises in machine learning is the excessive time taken to train a

system capable of some intelligent behaviour. Two interesting aspects of this are the *computational complexity* and the *sample complexity*. Computational complexity is a measure of how much computation is required in order to successfully learn the behaviour required. For example, deep neural networks have shown remarkable gains from building larger, more complex networks requiring a significant amount of computational resources.[1] Ongoing research in the field is attempting to speed up the execution of such networks by building specialized hardware or optimizing low-level operations [Sze et al., 2017]. Sample complexity or sample efficiency, on the other hand, relates to the number of samples that an algorithm needs in order to successfully learn the function or solve the task well. In RL, it may take the form of the number of episodes the agent needs to interact with the environment before it is able to perform well consistently. The problem of high sample complexity is often the limiting factor in a number of real-world applications where the observations or the action spaces may be high dimensional, large, continuous and over an extended temporal horizon. This phenomenon strongly relates to the *curse of dimensionality* [Bellman, 2015] where the number of samples required to estimate an arbitrary function with the same level of accuracy grows exponentially with respect to the dimensionality of the function. This problem makes the task of credit assignment difficult, resulting in the need for an inordinate number of episodes before the agent is able to perform satisfactorily. Throughout this thesis, the terms: *episode*, *trial* and *trajectory* are used interchangeably, but they convey the same meaning.

Another challenge faced during the design of a reinforcement learning task is that of designing and implementing an appropriate reward function. This reward function is typically the only extrinsic signal the agent receives about the progress it is making in solving the task and is a way for the task designer to modulate how the agent ought to behave. A well-designed reward function could very quickly guide an agent towards the intended behaviour and expedite the process of credit-assignment. However, in many real-world applications, designing of such informative reward functions is practically difficult and could still lead to unintended, suboptimal behaviour. The easiest form of reward function to implement is that of *sparse rewards* where the agent gets a positive reward (such as $+1$) for successfully finishing a task and nothing (0) for failure. Although very easy to implement, a sparse reward setting may be an extremely complex setting for credit assignment because the agent may achieve the reward through a large sequence of decisions and needs to determine which one of them was responsible for its eventual success or failure.

Learning goal-conditional behaviour has received significant attention in machine learning and robotics, especially because a goal-conditional policy may generalize desirable behaviour to goals unseen by the agent [Schmidhuber and Huber, 1990; Da Silva et al., 2012; Foster and Dayan, 2002; Schaul et al., 2015; Kaelbling, 1993; Kupcsik et al., 2013; Deisenroth et al., 2014; Zhu et al., 2017; Kober et al., 2012; Ghosh et al., 2018; Mankowitz et al., 2018; Pathak et al., 2018]. In many real-world problems, it is impractical for an agent to have experienced every possible combination of observations and goals. Therefore, the ability to generalize behaviour from those combinations that the agent has seen is highly crucial for successful learning. Goal-conditional policies strongly promote this by generalizing not only over states, as in the traditional reinforcement learning setting but also over goals. In this way, learning to reach goals that are within the agent's capabilities may aid its ability to reach ones that are much more difficult.

[1] https://blog.openai.com/ai-and-compute/

This idea of developing goal-based curricula to facilitate learning has attracted considerable interest [Fabisch and Metzen, 2014; Florensa et al., 2017; Sukhbaatar et al., 2018; Srivastava et al., 2013; Florensa et al., 2018]. Goal-conditional policies are also frequently encountered in the context of *hierarchical reinforcement learning* [Schmidhuber, 1991; Ring, 1991; Bakker and Schmidhuber, 2004; Wiering and Schmidhuber, 1998a; Dietterich, 2000; Dayan and Hinton, 1993; Barto and Mahadevan, 2003] where they may be used to plan using subgoals [Oh et al., 2017; Vezhnevets et al., 2017; Kulkarni et al., 2016; Levy et al., 2017], allowing for abstraction of details for low-level policies.

The methods presented in this thesis aim at improving sample complexity in goal-conditional policies in a sparse reward setting where the agent only receives a non-zero reward when the goal has been achieved and zero otherwise. This is a particularly challenging combination because not only is it a sparse-reward setting but it is also goal-conditional where the agent will not receive a reward unless the goal has been achieved thereby having no feedback to learn from until it does. An uninformed agent at the beginning of training must, through a sequence of random actions, achieve the goal required of it during that episode before any usable feedback is generated. The probability of performing such a sequence of actions completely at random may be extremely slim in environments with large, high dimensional goal spaces or a large number of actions. Consider, for example, a task involving a robotic arm that needs to grasp an object and move it to a specific goal location on a table. It receives no reward until purely by chance, it completes an incredibly complex sequence of actions that led to it successfully grasping the object and moving to the target location. In addition to this, the sparse rewards make the problem of learning from this one outcome challenging and would require many more such trials before usable patterns are identified and learning can begin, thereby leading to excessively high sample complexity.

A method designed to address the above-mentioned problem in goal-conditional reinforcement learning with sparse reward is that of *hindsight* [Karkus et al., 2016; Andrychowicz et al., 2017; Rauber et al., 2017]. Hindsight equips an agent to learn about arbitrary goals that were achieved when trying to achieve the original one. The key idea behind this being, the agent should learn not only from successful episodes but also from failures where it achieved goals other than what was intended. In learning about the degree to which it was able to achieve arbitrary goals, it leverages the generalization capabilities of goal-conditional policies to learn to solve more complex goals.

This thesis initially discusses our work on *Hindsight Policy Gradient (HPG)* presented in [Rauber et al., 2017], where we address this problem of high sample complexity in goal-conditional reinforcement learning by introducing the idea of hindsight to policy gradient methods [Williams, 1986, 1992; Sutton et al., 1999a], generalizing this idea to a successful class of reinforcement learning algorithms [Peters and Schaal, 2008; Duan et al., 2016]. We show how the estimator we propose leads to a remarkable increase in sample efficiency across a diverse selection of sparse reward environments.

It then covers novel work done to improve Hindsight Policy Gradient methods using Actor-Critic Architectures [Konda and Tsitsiklis, 2000] resulting in a new method we call "***Hindsight Actor-Critic (HAC)***" which attains improvement in sample complexity in a subset of the environments tested.

Finally, it introduces another extension that further improves the Hindsight Actor-Critic algorithm discussed by equipping the critic with the capacity of hindsight leading to another novel algorithm which we call "***Hindsight Actor Hindsight Critic (HAHC)***". HAHC showed striking improvement in sample complexity over hindsight policy gradients across all environments tested.

The organization of the rest of the thesis is as given below:

- Chapter 2 (Background) provides the relevant background information for all subsequent chapters and a review of the main classes of algorithms that exist in the Reinforcement Learning literature as well as some of their extensions to a goal-conditional reinforcement learning. It also provides a brief overview of Deep Neural Networks as function approximators in the context of Reinforcement Learning.
- Chapter 3 (Hindsight) discusses in detail the idea of hindsight as suggested by the title. It contains an introduction to off-policy reinforcement learning systems and our work in the introduction of hindsight to policy gradient methods as well as efforts to improve them using actor-critic architectures.
- Chapter 4 (Experiments) presents environments we use as platforms to test the performance of the methods discussed in this thesis as well as provide details on the experimental methodology used in the evaluation. It then presents the empirical results and relevant discussions of all the methods discussed in this thesis.
- Chapter 5 (Conclusion and Future Work) briefly summarizes the work and proposes avenues for future work.

Chapter 2

Background

Reinforcement Learning methods are broadly classified as either Model-Based or Model-Free depending on whether they choose to explicitly model the dynamics of the environment or not. However, there exist methods which combine aspects of both in order to build RL systems. Model-Free methods, since they do not model the dynamics of the environment, are preferred in cases where the environment is too complex to reliably model or for highly stochastic environments where there are a large number of possible realizations of the future. Model-based methods, however, may allow the use of planning algorithms using the model of the environment to derive a policy rather than real interaction with the environment thereby requiring a fewer number of episodes to learn a reasonable policy. This, however, is only true if a reliable model of the environment is already known or can be approximated easily. Further discussion in this thesis is restricted to Model-Free methods. We also focus on the *episodic* or terminating case where the agent interacts with the environment for a fixed number of time-steps before the episode terminates due to some criterion.

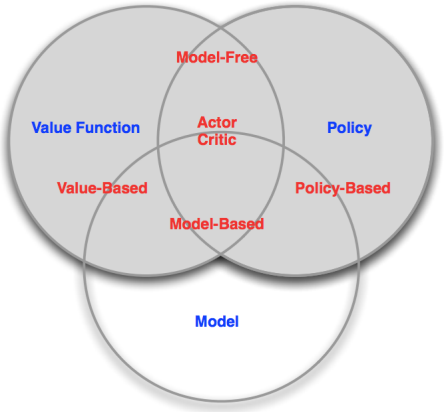

Figure 2.1. Reinforcement Learning taxonomy. Adapted from [Silver, 2015]

The taxonomy of RL algorithms and the intersections between them is shown in Figure 2.1. Note that some methods like random search have been omitted from the taxonomy described. Model-Free RL is broadly categorized into Policy Optimization methods and Value

Function based (Dynamic Programming) methods both of which will be discussed in the following sections. Throughout this thesis, we denote random variables by upper case letters and assignments to these variables by the corresponding lower case letters.

2.1 Dynamic Programming Methods

This class of RL algorithms approaches the solution using Modified Policy Iteration [Puterman and Shin, 1978], an algorithm from Approximate Dynamic Programming (ADP). They typically focus on learning *value functions*, which predict the expected sum of rewards the agent is going to receive under a policy and then acting greedily with respect to it to improve the policy.

2.1.1 Markov Decision Processes

Reinforcement Learning often models the interaction between the agent and the environment in the form of a *Makov Decision Process (MDP)*, a sequential decision-making framework. The theory behind solving these MDPs has been thoroughly developed [Puterman, 1994; Bertsekas and Tsitsiklis, 1995] and *Dynamic Programming (DP)* techniques are highly applicable to solving them. The MDP is typically defined by the following components:

- S (state space): a set of states of the environment
- A (action space): a set of actions the agent has to choose between to perform
- $p(s_{t+1} \mid s_t, a_t)$ (Transition probability distribution): For each state $s_t \in S$ and $a_t \in A$, the transitional distribution (or state-transition function) gives us the probability of transitioning to state s_{t+1} given the agent was in state s_t and performed an action a_t. Note that the transition probability distribution is conditionally independent of previous states and action. In other words, the state transitions of the MDP satisfy the Markov property (hence the name of a Markov Decision Process)
- $r(s_t) \in \mathbb{R}$ (reward): reward signal received when state s_t is achieved. Depending on the application, the reward function is sometimes also formulated as $r(s_t, a_t)$ or $r(s_t, a_t, s_{t+1})$

Return

The agent aims to find a behavioural policy π which maps states to actions. In this work, we only consider stochastic policies which are conditional distributions $\pi(a_t \mid s_t) = p(a_t \mid s_t)$. Methods that aim to learn deterministic policies where $a = \pi(s)$ have been used in literature. Let η_t be the sum of rewards or *return* which is given by

$$\eta_t = r(s_t) + r(s_{t+1}) + \cdots + r(s_{T-1}) = r_t + r_{t+1} + \cdots r_{T-1} = \sum_{t'=t}^{T-1} r_{t'}, \tag{2.1}$$

where t is the time index in an *episodic* environment which terminates at time step $T-1$.

For continuing (non-terminating) tasks, the discount factor $\gamma \in (0, 1]$ is introduced to discount future rewards, in order to allow for a finite horizon to optimize over and to prioritize

either short or long-term rewards. Let η_t be the *discounted* sum of rewards or *discounted return* which is given by

$$\eta_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}. \tag{2.2}$$

To avoid clutter, we discuss the episodic case of the return without the discount factor as shown in Equation 2.1 from here onwards.

Value Functions

In value-based methods, the agent needs some measure of how "good" it is to be in a state or to pick an action when in a state in order to learn to pick actions that lead to a better outcome. Value functions provide just such a measure through the *state-value* function or the *action-value* function. Both of them build upon the return defined in the previous section and are defined as the expected sum of long-term (perhaps, discounted) future reward.

Let $V_t^{\boldsymbol{\theta}}(s)$ be the state-value function, which is the expected sum of rewards for starting from the state s at time t and picking actions according to the policy parametrized by $\boldsymbol{\theta}$. Formally, for an un-discounted episodic task with T time-steps, the state-value function is defined as

$$V_t^{\boldsymbol{\theta}}(s) = \mathbb{E}_{\boldsymbol{\theta}}\left[\eta_t \mid S_t = s, \boldsymbol{\theta}\right] = \mathbb{E}_{\boldsymbol{\theta}}\left[\sum_{t'=t}^{T-1} r_{t'} \mid S_t = s, \boldsymbol{\theta}\right]. \tag{2.3}$$

Similarly, the action-value function, also known as a *Q-function*, can be defined as the expected sum of rewards starting from the state s at time t and picking an action a, and thereafter following the policy parametrized by $\boldsymbol{\theta}$. Concretely, for a non-discounted episodic setting, the action-value function $Q_t^{\boldsymbol{\theta}}(s,a)$ is defined as

$$Q_t^{\boldsymbol{\theta}}(s,a) = \mathbb{E}_{\boldsymbol{\theta}}\left[\eta_t \mid S_t = s, A_t = a, \boldsymbol{\theta}\right] = \mathbb{E}_{\boldsymbol{\theta}}\left[\sum_{t'=t}^{T-1} r_{t'} \mid S_t = s, A_t = a, \boldsymbol{\theta}\right]. \tag{2.4}$$

From their definitions, we can see that the two value functions are related to each other as

$$V_t^{\boldsymbol{\theta}}(s) = \mathbb{E}_{a \sim p(\cdot|s,\boldsymbol{\theta})}\left[Q_t^{\boldsymbol{\theta}}(s,a)\right]. \tag{2.5}$$

2.1.2 Monte-Carlo Methods

Monte-Carlo methods work on the idea of *Generalized Policy Iteration (GPI)* [Sutton and Barto, 1998], a special case of the Modified Policy Iteration. The GPI is an iterative process composed of two steps as shown in Figure 2.2. The first step tries to estimate how good a policy is by building a value function that is consistent with the current policy, known as the *policy evaluation* step. In the second step, known as the *policy improvement* step, the policy is improved by acting greedily with respect to the current value function. This process is repeated until each step has converged to the optimal value function and the optimal policy respectively.

Monte-Carlo methods perform a number of rollouts by executing the current policy over a sequence of episodes. Monte-Carlo methods are straightforward to implement but have been reported to require a large number of iterations for convergence, typically due to high variance

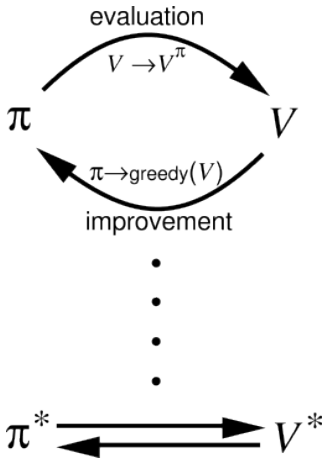

Figure 2.2. Generalized Policy Iteration. The two stages have the Value and Policy Functions interact until convergence. Adapted from [Sutton and Barto, 1998]

in value estimation [Sutton, 1988; Sutton and Barto, 1998]. Recent results [Amiranashvili et al., 2018], however, suggest that this might not be true in a deep reinforcement learning setting with finite-horizons. The update equation for an every-visit Monte-Carlo estimate of the value function $\hat{V}_t^{\boldsymbol{\theta}}(s_t)$ with learning rate α and the (optionally discounted) expected return η_t is given by

$$\hat{V}_t^{\boldsymbol{\theta}}(s_t) \leftarrow \hat{V}_t^{\boldsymbol{\theta}}(s_t) + \alpha[\eta_t - \hat{V}_t^{\boldsymbol{\theta}}(s_t)]. \tag{2.6}$$

2.1.3 Temporal-Difference Learning

Temporal Difference (TD) methods [Sutton, 1984] also build on top of GPI but differ from the Monte-Carlo methods in the way the value function is estimated. Instead of using the return as a target, TD methods incorporate ideas from Dynamic Programming by *bootstrapping* and performing updates based on current estimates by using the recursive definition of the form

$$V_t^{\boldsymbol{\theta}}(s) = \mathbb{E}_{\boldsymbol{\theta}}\left[\sum_{t'=t}^{T-1} r_{t'} \mid S_t = s, \boldsymbol{\theta}\right] = \mathbb{E}_{\boldsymbol{\theta}}\left[r_t + \sum_{t'=t+1}^{T-1} r_{t'} \mid S_t = s, \boldsymbol{\theta}\right] \tag{2.7}$$

$$= \mathbb{E}_{\boldsymbol{\theta}}\left[r_t + V_t^{\boldsymbol{\theta}}(S_{t+1}) \mid S_t = s, \boldsymbol{\theta}\right]. \tag{2.8}$$

By exploiting this recursive definition, we can update the value function through a temporal-difference error by updating the current estimate of the value function towards the new estimate by using the reward acquired at that time step. This form of update reduces the variance in the estimation of the value function but introduces a bias depending on the initial values of the value functions [Kearns and Singh, 2000; Sutton, 1988]. Although typically initialized with zeros or a small random value, initializing values with large numbers may make use of this bias in the form of *optimistic initialization* encouraging exploration and causing the agent to visit states it has not done so before [Brafman and Tennenholtz, 2002].

The update equation for the simplest one-step Temporal-Difference algorithm, TD(0), for the value function $V_t^{\boldsymbol{\theta}}(s_t)$ of the state s_t, one-step reward $r(s_{t+1})$ and the next state s_{t+1} with

the learning rate α is given by

$$V_t^{\theta}(S_t) \leftarrow V_t^{\theta}(S_t) + \alpha[r(S_{t+1}) + V_t^{\theta}(S_{t+1}) - V_t^{\theta}(S_t)], \tag{2.9}$$

where the term $r(S_{t+1}) + V_t^{\theta}(S_{t+1})$ is known as the *TD-target* and the difference between it and the current value function estimate is known as the *TD-error*. This one-step TD(0) formulation can be easily extended to multiple, arbitrary steps (also known as n-step TD methods) thereby allowing for faster propagation of values across a number of time-steps with a single update.

2.2 Function Approximation

A natural way to store a value function is in a tabular form indexed by the possible combinations of the input. This approach, although simple, is very limiting in environments with a large or continuous state space. With increasing dimensionality of the state space, most states might never be encountered by the agent despite a large amount of data collected. This requires the agent to generalize its behaviour to these states, based on those it has already observed. This kind of generalization of behaviour over states is not possible in a tabular setting which might even be computationally infeasible to store with a limited amount of memory. These problems often limit the applicability of tabular methods to many machine learning techniques, Reinforcement Learning included.

In order to overcome this limitation, function approximators are used to store the value function. A function approximator can be thought of as a mapping from states to value functions parameterized by some parameters θ thus giving $V(s;\theta)$. Since the number of parameters of a function approximator is smaller than the number of states in the state-space, an update to one part of the space also affects the value of nearby states similarly; a desirable property that aids in the generalization of the value function. Similar principles apply when function approximation is applied to represent other such functions, not just value functions.

A number of function approximators have been used in the past in Reinforcement Learning, often paired with feature encoding schemes such as tile coding and radial basis functions [Sutton and Barto, 1998]. Recently, the use of neural networks has become much more widespread due to their capability to learn complex, nonlinear representations of the value functions as opposed to methods like tile coding. Their ability of being powerful representation learners allows for learning with large, high-dimensional inputs such as input video frames [Mnih et al., Dec 2013].

Despite their power, they are typically not as efficient as online learners and generally operate in batch learning mode and require a number of alterations to be made before they can be retrofitted to reinforcement learning setting. Although convergence guarantees typically do not apply to reinforcement learning methods when using function approximation (as they do with the tabular case), they are a necessity for solving complex, real-world tasks and have shown promising empirical performance and stability.

2.2.1 Artificial Neural Networks

Artificial neural networks are loosely modelled on how the brain works. In their most basic form, feedforward neural networks consist of a set of neurons, arranged in layers with each neuron connecting to every single neuron in the previous layer. A neuron receives real-valued outputs as its input and producing a real-valued output itself.

In more concrete terms, each neuron i takes a vector of inputs x, calculates their weighted sum with the weights w_i and adds to it a bias term b_i. It is then passed through an *activation function* f, to produce the output. The complete equation for a neuron can be written as

$$y_i = f(\sum_j x_j w_{i,j} + b_i), \tag{2.10}$$

where x_j is the input corresponding to the j-th neuron in the previous layer. Each of these inputs have their own weight $w_{i,j}$ and a bias term. The activation function f introduces non-linearity to the output of the neuron thus allowing the neural network to learn more complex, non-linear representations from training data. Frequently used activation functions include:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic Tangent: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectified Linear (ReLU): $f(x) = max(0, x)$

The training of a neural network is made possible through the backpropagation algorithm, which allows for the calculation of the gradient needed in the weight updates. An optimization method such as *gradient descent* is used to then adjust the weights of the network by optimizing for a loss function that measures the disparity between the network's predictions and the target.

In recent years, the *stochastic gradient descent* has been a popular choice with momentum based methods to accelerate training speed. Adaptive step-size methods like AdaGrad [Duchi et al., 2011] and Adam [Kingma and Ba, 2014] have also been widely adopted due to their ability to adaptively adjust step-size for individual parameters and empirically achieve a higher speed of convergence.

2.3 Policy Optimization Methods

Value Function based methods aim to learn the state/action-value function and then use it to derive a behaviour policy. Policy Gradient methods instead aim to directly learn the policy using a parameterized policy function $\pi(a \mid s, \boldsymbol{\theta}) = p(A_t = a \mid S_t = s, \boldsymbol{\Theta_t} = \boldsymbol{\theta})$ where $\boldsymbol{\theta} \in \mathbb{R}^d$ represents the parameters of the policy function.

Policy Gradient methods aim to learn the policy by estimating the gradient of a performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameters $\boldsymbol{\theta}$. The aim is to maximize the performance measure which we accomplish by *approximate gradient ascent* with the update rule taking the form

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \overline{\nabla J(\boldsymbol{\theta_t})}, \tag{2.11}$$

where $\overline{\nabla J(\boldsymbol{\theta})}$ is some stochastic approximation of the gradient of the performance measure with respect to the policy parameters.

For discrete actions, we may represent the policy using an exponential softmax distribution given by

$$p(a \mid s, \boldsymbol{\theta}) = \frac{e^{f(s,a,\boldsymbol{\theta})}}{\sum_b e^{f(s,b,\boldsymbol{\theta})}}, \tag{2.12}$$

where $f(s, a, \boldsymbol{\theta})$ is the output of a parametric function approximator such as a deep neural network or as simple as a linear weighted sum between input features and parameters $\boldsymbol{\theta}^T \phi(s, a)$.

The advantage to using the exponential softmax distribution (and consequently, a continuous policy parameterization) is that the action probabilities change smoothly as a function of the learnt parameter θ. In a value-based method with an ϵ-greedy policy, a small change in action-values can make it highly biased to pick the maximum action leading to a drastic change in trajectories. Policy gradient methods with the softmax formulation, on the other hand, allow for a much smoother shift in policies when a small update is made to the policy parameters.

For continuous actions, a simple and common choice is to represent the policy using a linear-mean gaussian distribution given by

$$p(a \mid s, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{a - f(s, \boldsymbol{\theta})}{\sigma}\right)^2\right). \tag{2.13}$$

A linear-mean Gaussian parameterized policy is in no way the only option for continuous actions and a number of other continuous parameterizations such as Mixture of Gaussians [Agostini and Celaya Llover, 2010] have been proposed in literature.

Equation 2.11 has provided for a generic way to update the policy parameters such that we are always improving according to the performance metric chosen. There exist a number of estimation methods in literature, the most prominent of which include methods of *finite-difference*, *likelihood ratio policy gradient* [Glynn, 1987] and *natural policy gradient* [Kakade, 2002]. Amongst them, likelihood ratio methods have yielded the most real-world results in robotics, in addition to offering some appealing properties of stability and fast theoretical convergence speeds [Glynn, 1987].

2.3.1 Likelihood Ratio Policy Gradient

Let $\boldsymbol{\tau} = s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T$ denote a trajectory. We assume that the probability $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ of encountering a trajectory $\boldsymbol{\tau}$ given a policy parameterized by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is given by

$$p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, \boldsymbol{\theta}) p(s_{t+1} \mid s_t, a_t). \tag{2.14}$$

We choose the expected return $\eta(\boldsymbol{\theta})$ of a policy parameterized by $\boldsymbol{\theta}$ be the performance measure we want to directly optimize. This is given by

$$J(\boldsymbol{\theta}) = \eta(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(S_t) \mid \boldsymbol{\theta}\right] = \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t) = \sum_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) r(\tau), \tag{2.15}$$

where $r(\tau) = \sum_{t=1}^{T} r(s_t)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$. We use the standard identity from differential calculus, $\nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau) / p_\theta(\tau)$ (also known as the *likelihood-ratio trick*), in the above equation and get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{\tau} p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) = \mathbb{E}_{\boldsymbol{\tau}} \left[\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \, r(\boldsymbol{\tau}) \right]. \tag{2.16}$$

Note that applying the log on both sides of Equation 2.14 results in

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \log p(s_1) + \sum_{t=1}^{T-1} \log p(a_t \mid s_t, \boldsymbol{\theta}) + \sum_{t=1}^{T-1} \log p(s_{t+1} \mid s_t, a_t). \tag{2.17}$$

Using the value of $\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ obtained above, we see that only the middle term $p(a_t \mid s_t, \boldsymbol{\theta})$ depends on the policy parameters. The derivative of the other two terms becomes zero and using that value in $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ gives us

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}} \left[\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \, r(\boldsymbol{\tau}) \right] = \mathbb{E}_{\boldsymbol{\tau}} \left[\sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) \sum_{t'=1}^{T} r(s_{t'}) \right]. \tag{2.18}$$

Using the REINFORCE algorithm [Williams, 1992], we approximate the above estimate by performing N rollouts and taking a sample average of them as a gradient estimate as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) \sum_{t'=1}^{T} r(s_{t'}). \tag{2.19}$$

This gives us the update equation for the REINFORCE Policy Gradient. From Equation 2.19 and 2.11 we can see that the policy gradient update is quite interpretable. It updates the parameters $\boldsymbol{\theta}$ in such a way that it increases the probability of picking actions that lead to a high return and decreases the probability of choosing ones that yield a lower return.

2.3.2 Variance Reduction Methods for REINFORCE

REINFORCE provides a consistent, unbiased estimate of the gradient. However, like most Monte-Carlo methods, REINFORCE suffers from high variance in its estimates, particularly in stochastic environments with large time horizons or action/state spaces. Simply put, because of a large number of possible futures, each with potentially very different magnitudes of rewards obtained, the sample returns obtained could be highly variable thus leading to very noisy parameter updates.

This high variance increases the number of samples required in order to get a good estimate and can be a limiting factor in real-world applications like robotics where real-world experience is expensive to collect. There are a number of well-established methods of variance reduction techniques for policy gradient methods some of which work without biasing the estimate [Weaver and Tao, 2001; Williams, 1992] while others add bias, thus leading to a bias-variance trade-off [Gu et al., 2017].

One common approach to reduce variance in the REINFORCE Policy Gradient estimator is to use the principle of *causality* where a policy at time t' cannot affect the reward at a time t

when $t < t'$. We can, therefore, replace the sum of reward term in Equation 2.19 to only sum rewards from the current time step t as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) \sum_{t'=t}^{T} r(s_{t'}). \tag{2.20}$$

This reduces the variance of the estimator while still allowing for an unbiased, consistent estimate of the gradient. Another common approach to reduce variance in Monte-Carlo methods is through the use of *control variates*. *Baselines* are one popular form of control variates used in Policy Gradient methods. A baseline could be any function or random variable that is subtracted from the sample returns as a baseline/reference, thereby potentially reducing the magnitude of returns and hence, the estimator's variance. The formulation of REINFORCE with a baseline function b is shown below:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) (\sum_{t'=t}^{T} r(s_{t'}) - b) \tag{2.21}$$

An appealing property of baselines is that as long as they are independent of the selected action, the estimator still remains unbiased ($\nabla_{\boldsymbol{\theta}} \mathbb{E}[b] = 0$), in spite of drastically reducing the variance of the estimate. There are a number of popular choices for the baseline including *constant* baselines, *average reward* baselines, *state-value function* baselines, etc. Baseline choices such as the average reward also offer an intuitive interpretation of the update rule by making actions that lead to higher-than-average return more likely in the future while making actions leading to lower-than-average returns less likely.

2.4 Actor-Critic Methods

All the model-free methods discussed in the previous sections fall into two categories:

1. *Actor-only* methods work with a parameterized family of policies and encompass the various policy gradient methods discussed in Section 2.3. A potential drawback of these methods, as discussed, is their high variance which could greatly affect their convergence speeds. Furthermore, these methods tend to be *on-policy* and a new trajectory needs to be rolled out after every update to the parameters, making limited use of past experience or data collected from other sources such as human demonstrations or a different behaviour policy.

2. *Critic-only* methods rely heavily on the estimation of value functions and aim to learn an approximate solution to the Bellman equation which would lead to an optimal policy. This category of methods typically has a much lower variance than their actor-only counterparts. However, critic-only methods typically learn deterministic policies and are not as directly applicable to continuous action spaces as policy gradient methods [Silver et al., 2014] which adversely impacts their applicability to tasks like robotics and motor control. The ability to learn stochastic policies is highly favourable in situations such as state aliasing (or perceptual aliasing) where partial observability causes agents in two locations to experience the same input but have different optimal actions [Whitehead and Ballard, 1991]. Formulations to address this problem of uncertain state information such as the *Partially Observable Markov Decision Process (POMDP)* have been introduced

to reinforcement learning [Kaelbling et al., 1998] but see limited applicability due to high computational demands.

Actor-Critic methods aim to combine the strengths of both the above-described methods. The parameterized actor brings forth all its advantages while using the critic for low-variance estimation of its performance through which it can optimize its behaviour. This lower variance estimate helps in faster learning and stability, although initially trading for a high bias estimate when the critic's values are inaccurate early in the training process.

Intuitively, in environments with a high degree of stochasticity, there are many possible future outcomes, each with potentially very different returns. By using sample estimates as we do in actor-only methods like REINFORCE, the gradient estimate becomes high variance. By learning a value function, the critic learns the *expected* return over all possible futures from the current state thus offering a significantly lower variance estimate.

Referring to the REINFORCE with baseline formulation in 2.21, we consider the case where the baseline is an estimate of the value function $\hat{V}_t^{\boldsymbol{\theta}}(S)$ thereby giving

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) \left[\sum_{t'=t}^{T} r(s_{t'}) - \hat{V}_{t'}^{\boldsymbol{\theta}}(s_{t'}) \right]. \tag{2.22}$$

The subtraction of value function baseline term from the empirical return aims to reduce the variance of the estimator. Despite the normalizing effect of the baseline, the empirical return could still be a source of high variance for reasons mentioned above. To further decrease variance, we can replace the empirical return by the value function estimate as follows

$$\sum_{t=1}^{T} r(s_t) \approx r(s_t) + \hat{V}_t^{\boldsymbol{\theta}}(s_{t+1}). \tag{2.23}$$

By substituting Eq.2.23 in Eq.2.22, we obtain

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) (r(s_t) + \hat{V}_t^{\boldsymbol{\theta}}(s_{t+1}) - \hat{V}_t^{\boldsymbol{\theta}}(s)) \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(a_t \mid s_t, \boldsymbol{\theta}) \hat{A}_t^{\boldsymbol{\theta}}(s_t). \end{aligned} \tag{2.24}$$

$\hat{A}_t^{\boldsymbol{\theta}}(s_t)$ is called the **Advantage Function** typically defined as $\hat{A}_t^{\boldsymbol{\theta}}(s_t, a_t) = \hat{Q}_t^{\boldsymbol{\theta}}(s_t, a_t) - \hat{V}_t^{\boldsymbol{\theta}}(s_t)$. This actor-critic formulation of the update rule is often called the **advantage actor-critic**. Actor-critic methods can make use of value functions which are already a popular choice as a baseline function and offer improved stability at no additional computational cost. These value functions may be fit using either the Monte-Carlo methods (discussed in Section 2.1.2) or Temporal Difference methods (discussed in Section 2.1.3).

2.5 Off-Policy Reinforcement Learning

Methods discussed so far such as policy gradient methods and value-function based methods are *On-Policy* learning methods in the sense that they only learn from experience generated by

picking actions from the same policy that's being improved or evaluated. In contrast, in *Off-Policy* learning methods, a policy known as the *behaviour policy* is used to pick actions while another, potentially unrelated policy, typically called the *target policy*, is being evaluated or improved.

Off-policy methods are more powerful and general, with a number of advantages such as being able to re-use past experience, learning from expert demonstrations, using experience collected by other policies, learning about the optimal policy when following a behavioural policy, etc. On-Policy methods may, in fact, be seen as a special case of off-policy learning methods where the behaviour policy and the target policy are the same. Off-policy methods, despite being more general are typically considerably more difficult to train and typically have higher variance and are initially slower to converge [Van Seijen et al., 2009; Sutton and Barto, 1998].

2.5.1 Temporal Difference Methods

SARSA: On-Policy TD Control

In order to perform control using value-function based methods, learning an action-value function rather than a state-value function is important. In particular, we must estimate the value of $Q_t^{\boldsymbol{\theta}}(s,a)$ for the current behaviour policy parametrized by $\boldsymbol{\theta}$ for all $s \in S$ and $a \in A$ at time t. We can re-use the methods of temporal-difference specified in Section 2.1.3 to define a TD(0) style update for the action-value function analogous to Eq. 2.9

$$Q_t^{\boldsymbol{\theta}}(S_t,A_t) \leftarrow Q_t^{\boldsymbol{\theta}}(S_t,A_t) + \alpha\left[R_{t+1} + Q_t^{\boldsymbol{\theta}}(S_{t+1},A_{t+1}) - Q_t^{\boldsymbol{\theta}}(S_t,A_t)\right]. \tag{2.25}$$

This update can be done after every transition from a non-terminal state S_t. This update rule makes use of every element of the quintuple $(S_t,A_t,R_{t+1},S_{t+1},A_{t+1})$ thus giving rise to its name, *SARSA* [Rummery, 1995].

Q-Learning: Off-Policy TD Control

Q-Learning [Watkins, 1989] is an off-policy TD control algorithm which aims to approximate the action-value function of the greedy policy with respect to $Q_t^{\boldsymbol{\theta}}$. Its update rule is given as

$$Q_t^{\boldsymbol{\theta}}(S_t,A_t) \leftarrow Q_t^{\boldsymbol{\theta}}(S_t,A_t) + \alpha\left[R_{t+1} + \max_a Q_t^{\boldsymbol{\theta}}(S_{t+1},a) - Q_t^{\boldsymbol{\theta}}(S_t,A_t)\right]. \tag{2.26}$$

When compared with the SARSA update (Eq. 2.25), it is immediately apparent that the choice of action in $Q_t^{\boldsymbol{\theta}}(S_{t+1},A_{t+1})$ has been replaced with the action a that maximizes $Q_t^{\boldsymbol{\theta}}(S_{t+1},a)$. It is this replacement that allows the algorithm to be off-policy and learn about actions not generated by the behaviour policy. By always picking the greedy action, Q-Learning steps towards approximating the optimal value function and has a number of early convergence proofs and optimality guarantees [Watkins and Dayan, 1992; Tsitsiklis, 1994] under a tabular setting as long as the behaviour policy continues to visit every state-action pair.

2.5.2 Monte-Carlo Methods

Off-policy Monte-Carlo methods typically use *importance sampling* [Bishop, 2013], a general technique for estimating expected values under a target distribution while samples are drawn

from another (often called a proposal distribution in literature). In a reinforcement learning setting, we typically use importance sampling to do off-policy learning by weighing the return obtained by the relative probabilities of the trajectory occurring under the target policy and the behaviour policy. This ratio is referred to as the *importance sampling ratio*.

From Eq. 2.14, we know that the probability of a trajectory $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ is given by

$$p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, \boldsymbol{\theta}) p(s_{t+1} \mid s_t, a_t). \tag{2.27}$$

Assuming that we had another behaviour policy parameterized by $\boldsymbol{\beta}$, the relative probability of a trajectory $\boldsymbol{\tau}$ under the target and the behaviour policies is given by

$$\rho_{1:T-1} = \frac{p(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{p(\boldsymbol{\tau} \mid \boldsymbol{\beta})} = \frac{p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, \boldsymbol{\theta}) p(s_{t+1} \mid s_t, a_t)}{p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, \boldsymbol{\beta}) p(s_{t+1} \mid s_t, a_t)} = \prod_{t=1}^{T-1} \frac{p(a_t \mid s_t, \boldsymbol{\theta})}{p(a_t \mid s_t, \boldsymbol{\beta})}. \tag{2.28}$$

We assume the setting of off-policy policy evaluation where we try to estimate the value function of a target policy ($V_t^{\boldsymbol{\theta}}$) with actions being drawn from another behavioural policy $\boldsymbol{\beta}$ and hence, so are the returns η_t. These returns have the wrong expectation, and hence cannot be directly averaged to obtain $V_t^{\boldsymbol{\theta}}$ and give $V_t^{\boldsymbol{\beta}}$ instead. To compensate for this, we weigh the returns with the importance weights $\rho_{t:T-1}$ thus allowing us to estimate $V_t^{\boldsymbol{\theta}}(s)$ as

$$V_t^{\boldsymbol{\theta}}(s) = \mathbb{E}_{\boldsymbol{\theta}} \left[\eta_t | S_t = s \right] = \mathbb{E}_{\boldsymbol{\beta}} \left[\rho_{t:T-1} \eta_t \mid S_t = s \right]. \tag{2.29}$$

Intuitively, if a trajectory τ (and consequently, its return) is more probable under the behaviour policy than the target policy, it will be generated a lot more often and hence, it should be given a smaller weight. Similarly, if τ is more probable under the target policy than the behaviour policy, we increase its weight to simulate it occurring more often. This re-weighting is precisely what the importance weights take care of by looking at the relative likelihoods of the trajectory occurring under both the policies.

In a Monte-Carlo setting where we have to estimate this expectation from a sample, let M be the number of episodes that contain the state s. Let t_m be the first time when $S_t = s$ in the m^{th} episode. The *first-visit ordinary importance sampling estimator* of $V_t^{\boldsymbol{\theta}}(s)$ is given by

$$V_t^{\boldsymbol{\theta}}(s) \approx \frac{1}{M} \sum_{i=1}^{M} \left[\prod_{t'=t_m+1}^{T} \frac{p(a_{t'}^{(i)} \mid s_{t'}^{(i)}, \boldsymbol{\theta})}{p(a_{t'}^{(i)} \mid s_{t'}^{(i)}, \boldsymbol{\beta})} \sum_{k=t_m+1}^{T} r(s_k^{(i)}) \right]. \tag{2.30}$$

Ordinary importance sampling methods are *consistent*, meaning that they converge with probability 1 to the real expectation as the number of samples goes to infinity. They are also *unbiased*, meaning that their expected value after any number of examples is also the real expectation. However, these estimates have unbounded variance in general [Mahmood et al., 2014; Andradóttir et al., 1995]. As actions get increasingly unlikely according to the behaviour policy, the importance weights can grow fairly large leading to high variance. A popular solution used to alleviate this problem is through *weighted importance sampling* methods which are consistent but biased [Precup et al., 2000] in addition to being typically faster and more stable than the unweighted version. Weighted importance sampling methods, as the name suggests, do a weighted average by normalizing by the sum of likelihood ratios. This, although

introducing bias, allows for much smoother estimates and shows dramatically lower variance in practice and is often strongly preferred over ordinary importance sampling methods. The first-visit weighted importance sampling estimate of $V_t^{\boldsymbol{\theta}}(s)$ is given by

$$V_t^{\boldsymbol{\theta}}(s) \approx \frac{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} r(s_k^{(i)}) \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\beta})}\right]}{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\beta})}\right]}. \tag{2.31}$$

2.5.3 Goal-Conditional Value Functions

As with the typical undiscounted episodic reinforcement learning setting discussed in Section 2.1.1, we consider an agent that interacts with its environment in a sequence of episodes each of a total of T time-steps. At the beginning of an episode, the agent receives a goal $g \in G$ to be achieved during the episode. At every time-step t, the agent observes a state $s_t \in S_t$, receives a reward $r(s_t, g) \in \mathbb{R}$ and chooses an action $a_t \in A_t$ according to its policy. The reward $r(s_t, g)$ now also takes into account the goal we are trying to achieve. We note that most of the results presented in this section and the next have been adapted from our work in [Rauber et al., 2017].

We assume that the state-transition probability (also known as the transition model) denoted by $p(s_{t+1} \mid s_t, a_t)$ is independent of the goal pursued by the agent, which we denote by $S_{t+1} \perp\!\!\!\perp G \mid S_t, A_t$.

Using the above definitions, it is straightforward to define the goal-conditional Q-Function $Q_t^{\boldsymbol{\theta}}(s, a, g)$ and Value Function $V_t^{\boldsymbol{\theta}}(s, g)$ of a policy parameterized by $\boldsymbol{\theta}$ as below:

$$Q_t^{\boldsymbol{\theta}}(s, a, g) = \mathbb{E}\left[\sum_{t'=t+1}^{T} r(S_{t'}, g) \mid S_t = s, A_t = a, g, \boldsymbol{\theta}\right]$$

$$V_t^{\boldsymbol{\theta}}(s, g) = \mathbb{E}\left[Q^{\boldsymbol{\theta}}(s, A_t, g) \mid S_t = s, g, \boldsymbol{\theta}\right]$$

In contrast to the Markov decision process, this formulation allows the probability of state transition given an action to change across timesteps within an episode, a property that is implicit to policy gradient methods.

2.5.4 Goal-Conditional Policy Gradient

The derivation for the goal-conditional policy gradient methods follows results previously obtained in Section 2.3.1.

Let $\boldsymbol{\tau} = s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T$ denote a trajectory. We assume that the probability $p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta})$ of trajectory $\boldsymbol{\tau}$, given goal g and a policy parameterized by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is given by

$$p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) = p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, g, \boldsymbol{\theta}) p(s_{t+1} \mid s_t, a_t). \tag{2.32}$$

Assuming that $G \perp\!\!\!\perp \boldsymbol{\Theta}$, the expected return $\eta(\boldsymbol{\theta})$ of a policy parameterized by $\boldsymbol{\theta}$ is given by

$$\eta(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(S_t, G) \mid \boldsymbol{\theta}\right] = \sum_{g} p(g) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g). \tag{2.33}$$

The gradient of the expected returns with respect to theta $\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta})$ is given by

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \sum_{g} p(g) \sum_{\boldsymbol{\tau}} \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g). \tag{2.34}$$

The likelihood-ratio trick allows rewriting the previous equation as

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \sum_{g} p(g) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g). \tag{2.35}$$

Note that

$$\log p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) = \log p(s_1) + \sum_{t=1}^{T-1} \log p(a_t \mid s_t, g, \boldsymbol{\theta}) + \sum_{t=1}^{T-1} \log p(s_{t+1} \mid s_t, a_t). \tag{2.36}$$

Therefore,

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \sum_{g} p(g) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \left[\sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \right] \left[\sum_{t=1}^{T} r(s_t, g) \right]. \tag{2.37}$$

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \sum_{g} p(g \mid \boldsymbol{\theta}) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g) \sum_{t'=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(a_{t'} \mid s_{t'}, g, \boldsymbol{\theta}). \tag{2.38}$$

The previous equation can be rewritten as

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{t'=1}^{T-1} \mathbb{E}\left[r(S_t, G) \nabla_{\boldsymbol{\theta}} \log p(A_{t'} \mid S_{t'}, G, \boldsymbol{\theta}) \mid \boldsymbol{\theta} \right]. \tag{2.39}$$

Let c denote an expectation inside Eq. 2.39 for $t' \geq t$. In that case, $A_{t'} \perp\!\!\!\perp S_t \mid S_{t'}, G, \boldsymbol{\Theta}$, therefore:

$$c = \sum_{s_t} \sum_{s_{t'}} \sum_{g} \sum_{a_{t'}} p(a_{t'} \mid s_{t'}, g, \boldsymbol{\theta}) p(s_t, s_{t'}, g \mid \boldsymbol{\theta}) r(s_t, g) \nabla_{\boldsymbol{\theta}} \log p(a_{t'} \mid s_{t'}, g, \boldsymbol{\theta}). \tag{2.40}$$

Reversing the likelihood-ratio trick,

$$c = \sum_{s_t} \sum_{s_{t'}} \sum_{g} p(s_t, s_{t'}, g \mid \boldsymbol{\theta}) r(s_t, g) \nabla_{\boldsymbol{\theta}} \sum_{a_{t'}} p(a_{t'} \mid s_{t'}, g, \boldsymbol{\theta}) = 0. \tag{2.41}$$

Therefore, the terms where $t' \geq t$ can be dismissed from Eq. 2.39, leading to

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(S_t, G) \sum_{t'=1}^{t-1} \nabla_{\boldsymbol{\theta}} \log p(A_{t'} \mid S_{t'}, G, \boldsymbol{\theta}) \mid \boldsymbol{\theta} \right]. \tag{2.42}$$

The previous equation can be conveniently rewritten as

$$\nabla_{\boldsymbol{\theta}}\eta(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(A_t \mid S_t, G, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} r(S_{t'}, G) \mid \boldsymbol{\theta} \right]. \tag{2.43}$$

This gives us the update rule to the *Goal-Conditional Policy Gradient (GCPG)* estimator.

As with Policy Gradient methods in Section 2.3.2, we define a real-valued baseline function $b_t^\theta(s_t, g)$ which is independent of the action selected. We then subtract it from the return in Eq. 2.43 to obtain

$$\nabla_{\boldsymbol{\theta}} \eta(\boldsymbol{\theta}) = \sum_g p(g) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g, \boldsymbol{\theta}) \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \left[\left[\sum_{t'=t+1}^{T} r(s_{t'}, g) \right] - b_t^{\boldsymbol{\theta}}(s_t, g) \right]. \tag{2.44}$$

The above estimate remains an unbiased estimate of the gradient because the baseline term $\sum_{t=1}^{T-1} \mathbb{E}\left[\frac{\partial}{\partial \theta_j} \log p(A_t \mid S_t, G, \boldsymbol{\theta}) b_t^{\boldsymbol{\theta}}(S_t, G) \mid \boldsymbol{\theta} \right] = 0$.

Equation 2.44 gives us the update rule to the *Goal-Conditional Policy Gradient with Baseline (GCPG+B)* estimator with the choice of a baseline function $b_t^{\boldsymbol{\theta}}(S_t, G)$ being a approximation to time-dependent state-value function $V_t^{\boldsymbol{\theta}}(s, g)$ defined in Equation **??**.

Chapter 3

Hindsight

In a typical sparse-reward goal-oriented reinforcement learning task, the agent is rewarded for reaching the goal state and receives no usable feedback when it fails to achieve the goal. This results in a number of wasteful trials until an agent finally achieves a goal and receives a reward, only then obtains any real feedback to learn from. The regular way to combat such a problem is through *reward shaping* where reward signals are hand-tuned to provide more frequent and informative feedback. This, however, often requires thorough domain knowledge and a task conducive to reward shaping. However, reward shaping sometimes leads to biased and often suboptimal behaviour [Ng et al., 1999] as well as cases of "reward hacking"[1] where the agent learns to exploit a poorly designed reward function to maximize its reward whilst operating outside the expected behaviour. Developing methods that are capable of learning in these sparse reward scenarios is therefore highly pertinent.

Solving goal-oriented tasks is of great practical importance in a number of scenarios which are plagued by high sample-complexity caused in part due to the wastefulness of trials in a sparse reward scenario. Methods that aid exploration [Rückstieß et al., 2008; Sehnke et al., 2008; Wiering and Schmidhuber, 1998b; Bellemare et al., 2016; Houthooft et al., 2016] in reinforcement learning aim to address this problem through a more systematic or informed exploration of the state space. However, in environments with long horizons and sparse rewards, exploration might not be enough to solve the task at hand. Consider the simple task of *bit-flipping* where a string of k bits is given to the agent initialized with $\mathbf{0}$ and the goal is to reach a given target configuration of bits. The agent has k actions that flip the value of the corresponding bit. For a string of length $k = 50$, there are $2^{50} = 1.1258999e + 15$ (or over a quadrillion) possible goals and for *each possible goal*, the agent needs to search through over a quadrillion of its own possible states before it reaches one that fetches it the reward. Under such a sparse reward setting, the chances of an agent employing methods of randomized exploration and still seeing enough rewarding states are infinitesimally small. The use of count-based exploration methods is also infeasible due to the size of the state/goal space, thus severely limiting the benefits obtained of such methods [Andrychowicz et al., 2017]. A strategy to ensure learning is possible in such an environment is to learn from the episodes where the agent failed to reach the goal as well as during those where it succeeds. Such capability is introduced to the system through methods of *hindsight*.

[1] https://blog.openai.com/faulty-reward-functions/

The ability of an agent to learn and exploit information from reaching arbitrary goals while trying to achieve another is called hindsight. This capacity was introduced by [Andrychowicz et al., 2017] to off-policy reinforcement learning algorithms that rely on experience replay [Lin, 1992a]. Hindsight was introduced to policy gradient methods by [Rauber et al., 2017], generalizing the idea to a broad class of successful algorithms. In earlier work, Karkus et al. [2016] introduced hindsight to policy search based on Bayesian optimization [Metzen et al., 2015].

In order to illustrate the concept of hindsight, consider a typical goal-conditional reinforcement learning agent in the *four rooms* maze environment as shown in Figure 3.1. At the start of an episode, the agent begins in the state shown in green and must reach the goal state marked red. In the process of trying to achieve the goal, the agent performs a sequence of actions and reaches the room to the bottom. In a typical sparse-reward scenario, this would yield no useful feedback. During hindsight, we can re-use potential goal states we have already encountered and learn about them as if they were the intended goals. We could use this otherwise wasted trajectory and learn how to reach one or more of the goals shown in the bottom row of figure 3.1. This approach not only learns to solve the hindsight goals but help exploit generalization capabilities of goal-conditional policies [Schaul et al., 2015; Sutton et al., 2011].

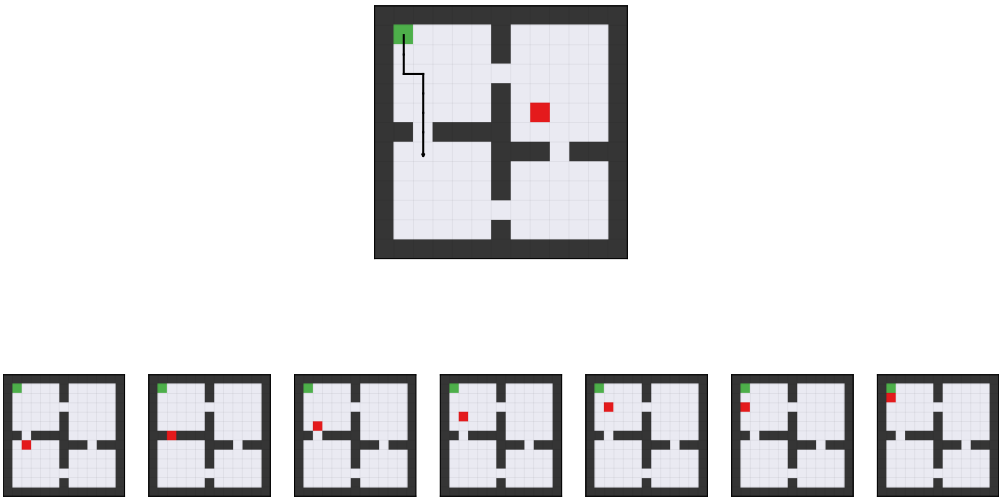

Figure 3.1. Top Row: Four Rooms maze with original intended goal (red) and path followed. Bottom Row: Hindsight with possible goal choices

This chapter first discusses prior work by [Andrychowicz et al., 2017] introducing the concept of hindsight to off-policy reinforcement learning algorithms that use experience replay. It then presents work done by [Rauber et al., 2017] where we introduce hindsight to policy gradient methods thus equipping a whole new class of algorithms with this powerful capability. The sections that follow describe attempts to further improve sample complexity of hindsight policy gradient using actor-critic architectures resulting in two novel methods. The experimental results presented in the next chapter further validate the idea of hindsight and demonstrate empirically that the striking improvement in sample complexity obtained by hindsight policy gradient methods and the further enhancement gained by using methods introduced in this chapter.

3.1 Hindsight Experience Replay (HER)

In *Hindsight Experience Replay (HER)*, [Andrychowicz et al., 2017] introduced the capability of hindsight to off-policy reinforcement learning algorithms that rely on experience replay [Lin, 1992a]. In addition to the state transitions that are acquired through interaction with the environment and added to the replay buffer, we also sample additional goals that were achieved in the trajectory and add transitions with these sampled goals.

Formally, after experiencing some episode $\tau = s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T$, every transition is stored in the replay buffer as the tuple $(s_t, a_t, r_t, s_{t+1}, g)$. We also store additional copies of the transitions with goals $g' \in \mathbb{S}(\tau)$ with updated rewards r'_t evaluated as $r(s_t, g')$. This implicitly states that we need access to the reward function to evaluate whether a goal has been achieved in a state. Another assumption made by this process is that the state-transition function is independent of the goal being pursued.

The operation $\mathbb{S}(\tau)$ represents a sampling strategy used to select one or more of the goals achieved in the trajectory τ as additional goals for HER. The authors discuss a number of possible sampling strategies including:

- *final* - Only goals achieved in the final state of the environment is picked as an active goal.
- *episode* - Choose k random goals achieved in the episode the transition tuple was drawn from.
- *future* - Choose k random goals achieved in the episode after the transition.
- *random* - Choose k random goals encountered in the entire training period.

From the various goal sampling strategies considered, *future* was picked as the best performing one and used in conjunction with *Deep Q Learning (DQN)* [Mnih et al., Dec 2013] for discrete actions and *Deep Deterministic Policy Gradient (DDPG)* [Lillicrap et al., 2015] for a continuous action space tasks. Experiments were performed on a variant of the bit-flipping environment (Section 4.1.1) and a number of novel continuous control robotic tasks with a simulation of a Fetch Robotics hand (see Section 4.1.3 for a variant of one of these environments).

Agents trained with Hindsight Experience Replay showed overwhelmingly better results across all tasks considered when compared to vanilla versions of DQN or DDPG and versions of them augmented with a count based exploration method. This proves the ability of the agent to learn about achieving arbitrary goals and relying on the generalization capability of the general value function (or goal-conditional value functions) [Schaul et al., 2015; Sutton et al., 2011] in order to learn how to solve a complex task.

3.2 Hindsight Policy Gradient (HPG)

In our method *Hindsight Policy Gradient (HPG)* [Rauber et al., 2017], we introduce hindsight to policy gradient methods. As with previous methods of hindsight [Andrychowicz et al., 2017], we make the assumption that we have access to $r(s, g)$ for every state and goal thereby allowing us to evaluate a trajectory obtained while trying to achieve the original goal g' with an alternative goal g. The sections below present results from [Rauber et al., 2017]. We refer the reader to the paper for their corresponding proofs. The corresponding experimental section in the next chapter (4.3) discuss their empirical evaluation across a broad range of environments.

Hindsight Policy Gradient Estimators

Hindsight is introduced to policy gradient methods through the application of importance sampling (Section 2.5.2) that allows the use of off-policy information to update our current policy. This is done by multiplying the reward obtained by a ratio of how likely a trajectory was if we had pursued the alternate goal g instead of the original goal g' in Equation 2.14. This leads to the formulation of the *Every-decision hindsight policy gradient* given below

Formally, for an arbitrary (original) goal g', the gradient $\nabla\eta(\boldsymbol{\theta})$ of the expected return with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\eta(\boldsymbol{\theta}) = \sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} \left[\prod_{k=1}^{T-1} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] r(s_{t'}, g). \tag{3.1}$$

The estimator defined above considers the complete returns without breaking them down into rewards. On the other hand, the *per-decision importance sampling estimator* presented below is able to estimate the value at every time-step and could potentially be more efficient and easier to implement as it works on a per-step (or per-decision) basis. This is done by removing the dependence of the weight on reward r_t on future time-steps. This is analogous to the important result applied to action-value functions by [Precup et al., 2000].

For an arbitrary (original) goal g', the gradient $\nabla\eta(\boldsymbol{\theta})$ of the expected return with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\eta(\boldsymbol{\theta}) = \sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} \left[\prod_{k=1}^{t'-1} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] r(s_{t'}, g). \tag{3.2}$$

Hindsight Policy Gradient with Baseline

As with policy gradient methods, we can introduce a baseline function to the above hindsight policy gradient formulation and further decrease the variance of the estimator without affecting its bias.

For every g', $t, \boldsymbol{\theta}$, and associated real-valued (baseline) function $b_t^{\boldsymbol{\theta}}$,

$$\sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \left[\prod_{k=1}^{t} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] b_t^{\boldsymbol{\theta}}(s_t, g) = \mathbf{0}. \tag{3.3}$$

Although not optimal, we follow convention and let the baseline function $b_t^{\boldsymbol{\theta}}$ be an approximation to the undiscounted state-value function $\hat{V}_t^{\boldsymbol{\theta}}$. [Rauber et al., 2017] also presents the optimal constant baseline that minimizes the (element-wise) variance of the estimator.

For every g', $t, \boldsymbol{\theta}$, and associated real-valued (baseline) function $b_t^{\boldsymbol{\theta}}$, the gradient $\nabla\eta(\boldsymbol{\theta})$ of the expected return with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\eta(\boldsymbol{\theta}) = \sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) z, \tag{3.4}$$

where

$$z = \left[\sum_{t'=t+1}^{T} \left[\prod_{k=1}^{t'-1} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] r(s_{t'}, g) \right] - \left[\prod_{k=1}^{t} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] \hat{V}_t^{\boldsymbol{\theta}}. \tag{3.5}$$

Weighted Per-Decision Hindsight Policy Gradient

Preliminary experiments found unweighed estimators (Eq. 3.2) sometimes led to unstable learning, potentially due to high variance in gradient estimation. As discussed in Section 2.5.2, weighted importance sampling is often preferred to ordinary importance sampling methods in an effort to trade increased bias for reduced variance. We apply the same to the per-decision hindsight policy gradient estimator to obtain the *weighted per-decision hindsight policy gradient estimator* given by

$$\sum_{i=1}^{N} \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(A_t^{(i)} \mid S_t^{(i)}, G^{(i)} = g, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} \frac{\left[\prod_{k=1}^{t'-1} \frac{p(A_k^{(i)} \mid S_k^{(i)}, G^{(i)}=g, \boldsymbol{\theta})}{p(A_k^{(i)} \mid S_k^{(i)}, G^{(i)}, \boldsymbol{\theta})} \right] r(S_{t'}^{(i)}, g)}{\sum_{j=1}^{N} \left[\prod_{k=1}^{t'-1} \frac{p(A_k^{(j)} \mid S_k^{(j)}, G^{(j)}=g, \boldsymbol{\theta})}{p(A_k^{(j)} \mid S_k^{(j)}, G^{(j)}, \boldsymbol{\theta})} \right]}, \tag{3.6}$$

The above estimator is a consistent but biased estimator of the gradient $\nabla\eta(\boldsymbol{\theta})$ for the expected return. A proof of the above and a result for a consistency-preserving *weighted baseline* is presented in [Rauber et al., 2017].

We emphasize that the set $\mathcal{G}^{(i)} = \{g \in G \mid \text{exists a } t \text{ such that } r(s_t^{(i)}, g) \neq 0\}$ correspond to goals in the i^{th} episode that have a non-zero expectation over goals in the above defined estimators. This set, which we call ***active goals***, often corresponds directly to states visited during the episode in many sparse-reward environments. This allows for the computation of the exact expectation when the goal distribution is known, failing which it may need to be approximated.

3.3 Hindsight Actor-Critic (HAC)

Following a progression similar to that of vanilla policy gradient methods in Section 2.4, we may combine the power of the actor-only hindsight policy gradient and a critic-only method like a state-value function or action-value function with the critic providing a lower-variance performance signal than the empirical return thus leading to potentially more stable training.

This chapter first describes the *action-value formulation* and *advantage formulation* of the hindsight policy gradient estimator that was presented in [Rauber et al., 2017]. Two potential ways to train the critic's value functions are discussed along with the bias/variance consequence of such a choice. The corresponding experimental section in the next chapter (4.4) shows their empirical evaluation.

The *action-value formulation* of the hindsight policy gradient (presented in [Rauber et al., 2017]) is reproduced below. For an arbitrary goal g', the gradient $\nabla\eta(\boldsymbol{\theta})$ of the expected return with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\eta(\boldsymbol{\theta}) = \sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \left[\prod_{k=1}^{t} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] Q_t^{\boldsymbol{\theta}}(s_t, a_t, g). \tag{3.7}$$

Similarly, the *advantage formulation* of the hindsight policy gradient has also been introduced and is shown below. For an arbitrary (original) goal g', the gradient $\nabla\eta(\boldsymbol{\theta})$ of the expected return with respect to $\boldsymbol{\theta}$ is given by

$$\nabla\eta(\boldsymbol{\theta}) = \sum_{\tau} p(\tau \mid g', \boldsymbol{\theta}) \sum_{g} p(g) \sum_{t=1}^{T-1} \nabla \log p(a_t \mid s_t, g, \boldsymbol{\theta}) \left[\prod_{k=1}^{t} \frac{p(a_k \mid s_k, g, \boldsymbol{\theta})}{p(a_k \mid s_k, g', \boldsymbol{\theta})} \right] A_t^{\boldsymbol{\theta}}(s_t, a_t, g). \tag{3.8}$$

It can be observed that these formulations are analogous to those of the advantage actor-critic formulation of the vanilla policy gradient methods discussed in Section 2.4. The advantage $A_t^{\boldsymbol{\theta}}(s_t, a_t, g)$ in the above formulation can be approximated under a goal g by using the state transitions collected while pursuing another goal as shown below [Rauber et al., 2017]. For every t and $\boldsymbol{\theta}$, the advantage function $A_t^{\boldsymbol{\theta}}$ is given by

$$A_t^{\boldsymbol{\theta}}(s, a, g) = \mathbb{E}\big[r(S_{t+1}, g) + V_{t+1}^{\boldsymbol{\theta}}(S_{t+1}, g) - V_t^{\boldsymbol{\theta}}(s, g) \mid S_t = s, A_t = a \big]. \tag{3.9}$$

This advantage actor-critic formulation, which is henceforth called the *Hindsight Actor-Critic (HAC)*, updates the parameters of the actor using the immediate reward and the value function of the critic as seen in Equation 3.9. From the preliminaries discussed in Section 2.1, there are two ways to do policy evaluation and learn a state-value function $V_t^{\boldsymbol{\theta}}(s, g)$ that estimates the expected return of a policy parametrized by parameters $\boldsymbol{\theta}$ in a state s while trying to achieve a goal g at time t.

The state-value function may be learnt through methods of Monte-Carlo (discussed in 2.1.2) where the value function is learnt through regression against the empirical return. A critic learnt through such methods is henceforth called the *Monte-Carlo critic*. State-Value functions may also be learnt through methods of Temporal-Difference learning (discussed in Section 2.1.3) where the value-function is bootstrapped to recursively re-use values of other states to accelerate training, henceforth called the *Bootstrapped critic*.

The choice of which type of critic to use further presents a bias-variance tradeoff. Monte-Carlo methods are high variance but unbiased thus allowing the Monte-Carlo critic to provide unbiased gradients but is typically slower to learn due to its variance. Methods of bootstrapping have very low variance but come at the cost of increased bias, especially during the initial stages of training. The initial values of the value function may have a very strongly biased influence on the feedback signal a bootstrapped critic provides to the actor. Despite this, bootstrapped critics are the typical choice in actor-critic settings with the bias often empirically aiding learning [Sutton and Barto, 1998]).

3.4 Hindsight Actor Hindsight Critic (HAHC)

From the formulation of the actor-critic method, it is immediately apparent that its performance requires both actor and critic to function well. A badly fit critic would not be capable of providing the right gradients to the actor and would cause the system to destabilize. Empirical evidence obtained from the Hindsight Policy Gradient estimator with a value function baseline in Section 4.3.1 shows instability in some environments. This problem also recurs in the case of the Hindsight Actor-Critic estimators on the same environments, as seen in Section 4.4. This has been hypothesized as being due to overestimation by a badly fitted value function causing good actions or trajectories to be penalized.

Under the assumption that the function approximator that we choose to represent the value function is expressive enough, typical ways to tackle a badly fitted function would be to either provide it with more training data or use better learning algorithms for faster training. Applying the former to the HAC estimator could take the form of running multiple steps of policy evaluation where only the critic is trained after one training step for the policy. This is, however, extremely wasteful in a setting where minimization of sample complexity is the aim. The use of better algorithms would be, for example, through the use of eligibility traces or multi-step Temporal-Difference learning methods for a bootstrapped critic [Sutton and Barto, 1998]) or improved learning algorithms or better optimizers.

Another way to address such a problem would be through the application of the concept of hindsight to the policy evaluation step that fits the critic, thereby evaluating not only the values of the original goals the policy was trying to achieve, but also the active goals achieved in hindsight through the use of off-policy policy evaluation techniques,as discussed in the section below. We call this novel class of algorithms *Hindsight Actor Hindsight Critic (HAHC)*.

There exist two main methods to do off-policy policy evaluation. The first is through the use of importance sampling methods as discussed in Section 2.5.2 in the context of re-using data collected off-policy from a behaviour policy to estimate the value function of a separate target policy. The second uses the recursive definition between the state-value function and an action-value function and can be computed through the Bellman equation.

Off-Policy Policy Evaluation using Importance Sampling

Weighted per-decision importance sampling has been shown to be highly effective in imbuing policy gradient estimators with hindsight as evidenced by Hindsight Policy Gradient in Section 4.3. Based on their efficacy, we chose to apply the same per-decision weighted importance sampling algorithm to allow a Monte-Carlo estimator of the state-value function $V_t^{\boldsymbol{\theta}}(s)$ to learn from off-policy information (as discussed in Section 2.5.2). We begin by recalling the formulation of the per-decision weighted importance sampling weighted estimator of the value function from Eq.2.31 which gives an estimate of the value function $V^{\boldsymbol{\theta}}(s)$ of a policy parametrized by $\boldsymbol{\theta}$ where trajectories were sampled from a separate behaviour policy parameterized by $\boldsymbol{\beta}$

$$V_t^{\boldsymbol{\theta}}(s) \approx \frac{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} r(s_k^{(i)}) \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\beta})}\right]}{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},\boldsymbol{\beta})}\right]}.$$

With a focus on goal-conditional policies, we replace the likelihood ratio of the trajectory under the two policies with the likelihood ratio of the trajectory had an alternate goal g been pursued instead of the original goal g', just as done with the formulation of the hindsight gradient estimator in Equation 3.2. This would thereby replace the above with a goal-conditional value function (Section 2.5.3) estimating the value function under hindsight goals as shown below

$$V_t^{\boldsymbol{\theta}}(s,g) \approx \frac{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} r(s_k^{(i)}) \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},g,\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},g',\boldsymbol{\theta})}\right]}{\sum_{i=1}^{N}\left[\sum_{k=t_m+1}^{T} \prod_{t'=t_m+1}^{k-1} \frac{p(a_{t'}^{(i)}|s_{t'}^{(i)},g,\boldsymbol{\theta})}{p(a_{t'}^{(i)}|s_{t'}^{(i)},g',\boldsymbol{\theta})}\right]}. \tag{3.10}$$

This would allow the critic to fit its value function using off-policy data during hindsight, just as done with the actor. Being analogous with the hindsight actor-critic formulation proposed in the previous section, we call this new formulation the *Hindsight Actor Hindsight Critic with Monte-Carlo Critic (HAHC_MCCRITIC)*.

Off-Policy Policy Evaluation using Temporal-Difference methods

The traditional SARSA method (Section 2.5.1) is an on-policy algorithm used to update the action-value function based on the sampled action and state. We recall from Eq. 2.25 that its update rule is of the form

$$Q_t^{\boldsymbol{\theta}}(s_t,a_t) \leftarrow Q_t^{\boldsymbol{\theta}}(s_t,a_t) + \alpha\big[r(s_{t+1}) + Q_t^{\boldsymbol{\theta}}(s_{t+1},a_{t+1}) - Q_t^{\boldsymbol{\theta}}(s_t,a_t)\big].$$

Since SARSA is a purely on-policy, the use of a_{t+1} introduces variance when the policy being estimated is stochastic which may slow convergence. A variant of SARSA was proposed in [Sutton and Barto, 1998] designed to reduce variance in the estimate. The proposed *Expected SARSA* update rule replaces the sample action a_{t+1} with an expectation over actions available at s_{t+1} as shown below

$$Q_t^{\boldsymbol{\theta}}(s_t,a_t) \leftarrow Q_t^{\boldsymbol{\theta}}(s_t,a_t) + \alpha\left[r(s_{t+1}) + \sum_a p(a \mid s_{t+1},\boldsymbol{\theta}) Q_t^{\boldsymbol{\theta}}(s_{t+1},a) - Q_t^{\boldsymbol{\theta}}(s_t,a_t)\right]. \tag{3.11}$$

Expected SARSA has a number of properties that make it a significant improvement over SARSA such as its lower variance [Van Seijen et al., 2009]. Although Expected SARSA has been predominantly studied as an on-policy method in the past, recent work [De Asis et al., 2017] has shown that Expected SARSA can be used directly for off-policy estimation without the need for any importance sampling correction and generalizes easily to multi-step updates. Therefore, it serves as an ideal way to do off-policy policy estimation using methods of temporal-difference.

Since the focus is on goal-conditional policies, we use results stated in Section 2.5.3 and rewrite the Expected SARSA update (Eq. 3.11) to be conditioned on a goal g that is trying to achieve as shown below

$$Q_t^{\boldsymbol{\theta}}(s_t,a_t,g) \leftarrow Q_t^{\boldsymbol{\theta}}(s_t,a_t,g) + \alpha\left[r(s_{t+1},g) + \sum_a p(a \mid s_{t+1},g,\boldsymbol{\theta}) Q_t^{\boldsymbol{\theta}}(s_{t+1},a,g) - Q_t^{\boldsymbol{\theta}}(s_t,a_t,g)\right]. \tag{3.12}$$

Since the Expected SARSA update rule allows for off-policy updates, we are free to evaluate the actor for any arbitrary goal g even if the experience was collected while trying another original goal g'. Therefore, this will allow the critic to fit its value function using off-policy data during hindsight, in tandem with the actor. Although we are estimating an action-value function $Q_t^{\boldsymbol{\theta}}(s_t, a_t, g)$, a state-value function $V_t^{\boldsymbol{\theta}}(s, g)$ can be obtained by taking an expectation over the actions drawn from the policy parametrized by $\boldsymbol{\theta}$ as shown below.

$$V_t^{\boldsymbol{\theta}}(s, g) = \sum_a p(a \mid s, g, \boldsymbol{\theta}) Q_t^{\boldsymbol{\theta}}(s, a, g) \tag{3.13}$$

Equation 3.13 allows us to convert the action-value into a state-value thus not requiring any changes to the update rule of the advantage formulation of the hindsight policy gradient. Being analogous to methods presented in the previous section, we call this new formulation the *Hindsight Actor Hindsight Critic with a Bootstrapped Critic (HAHC_BCRITIC)*. As with HAHC_MCCRITIC, not only does the actor use hindsight in the form of hindsight policy gradient but also does the critic which does off-policy policy evaluation, thereby evaluating the policy on hindsight goals.

Chapter 4

Experiments

This chapter discusses the empirical evaluation of the estimators discussed in this thesis. It first begins by introducing the environments that are used as a testbed for evaluating goal-conditional policies. It then provides an overview of the experimental protocol used in the evaluation of these policies including metrics and modalities tested as well as procedures for hyperparameter search and conclusive experiments. It then presents the experimental results of the Hindsight Policy Gradient (3.2), the Hindsight Actor-Critic (3.3) and the Hindsight Actor Hindsight Critic (3.4) estimators that were discussed in previous chapter.

4.1 Environments

Games and simulations serve as ideal testbeds for reinforcement learning systems due to a number of appealing properties. Environments may be chosen based on a number of properties the system needs to be tested against such as partial observability, stochasticity in environment dynamics, episodic/continuing tasks, dimensionality and continuity of observations, dimensionality and continuity of actions, length of episodes, presence of optimal substructure, sparse/dense rewards, goal-conditionality, etc. Games also allow for very quickly obtaining the thousands or millions of episodes often needed for successful learning depending on the complexity of the task.

Environments suited to the evaluation of goal-conditional policies are not well established in literature. Popular benchmarks like ATARI [Bellemare et al., 2013]) or Physics based environments with continuous control (MuJoCo, Bullet, etc.) do not typically provide specific goals to achieve nor the ability to evaluate when such a goal has been achieved. In order to evaluate the methods presented in [Rauber et al., 2017], we created/modified existing environments to satisfy a number of criteria:

1. An episode should terminate when the goal is reached or some maximum number of time-steps have elapsed.
2. A goal $g \in G$ should be provided by the environment at the starting of the episode.
3. There is access to the reward function $r(s_t, g)$ for some arbitrary $s_t \in S_t$ and $g \in G$.
4. A non-zero reward is given only when the current goal g has been achieved.

Apart from satisfying the above requirements, the environments were picked to differ in stochasticity, observation dimensionality, size and continuity, number of actions and relationship between goals and actions. In all the environments presented below, the agent receives the remaining number of time-steps plus one reward for reaching a goal state which also terminates the environment and receives a reward of 0 in any other case.

4.1.1 Bit Flipping

The *bit flipping* environment is similar to the one introduced by [Andrychowicz et al., 2017], where the agent starts in a state $\mathbf{0}$ represented by k bits of zeros and must reach a randomly chosen configuration of bits in a maximum of $k+1$ time steps. There are k actions, each of which flips the bit at the corresponding location. Despite its apparent simplicity, large values of k are practically unsolvable by standard RL systems even when augmented with exploration methods. The problem is not necessarily one of exploration, but one of a restrictively large search space to explore. For example, for $k = 30$, there are over a billion possible configurations reachable in $t+1$ time steps and structured exploration methods to check every possible configuration for every possible goal g are unfeasible.

The typical solution to such a problem would be to use a shaped reward to guide the current configuration to the goal with a more meaningful reward signal. Although this holds for this environment, it is difficult and sometimes impossible to do for more complex environments. Results from [Andrychowicz et al., 2017] and [Rauber et al., 2017] have shown that systems without hindsight fail to solve the task satisfactorily for $k > 15$. In order to demonstrate this, we show results for $k = 8$ and $k = 16$.

4.1.2 Grid world environments

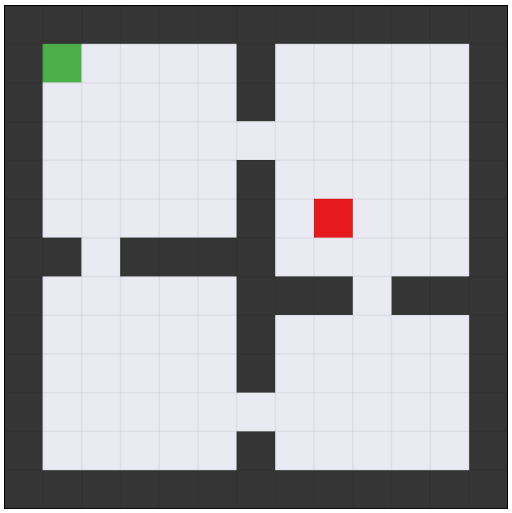

Figure 4.1. Four Rooms Environment

The *grid world* environments are modifications to the classical grid world environments standard in Reinforcement Learning literature. The agent starts every episode in a (potentially) random $(row, column)$ location in a 11×11 grid and must reach a randomly chosen goal location in a maximum of 32 time steps. Four actions allow the agent to move one step in the cardinal directions as long as there isn't a wall blocking its destination.

In the *empty room* environment, the agent starts the episode at the upper-left of the grid and there are no walls. In the *four rooms* environment [Sutton et al., 1999b]), the agent starts in one of the four corners of the grid (see Figure 4.1). There are walls which partition the grid into four rooms, such that each is linked to two others through openings in the walls (doors). Stochasticity is also introduced by replacing the selected action with a random one with a probability of 0.2.

4.1.3 FetchPush (Robotics)

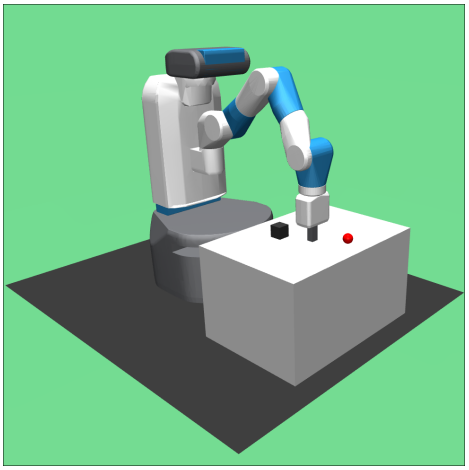

Figure 4.2. FetchPush

The *FetchPush* environment is a variant of the environment released by [Plappert et al., 2018] and first reported by [Andrychowicz et al., 2017] using Hindsight Experience Replay with Deep Deterministic Policy Gradient [Lillicrap et al., 2015]. This environment provides a simulation of a 7 degree-of-freedom Fetch Robotics arm with a two finger parallel gripper. The arm is placed in front of a table (see Figure 4.2) upon which rests a randomly placed block that must be pushed to a specific goal location on the table (visualized as a red sphere). This task is of practical interest in robotics and is challenging to do without reward shaping since, under the sparse reward scenario, the probability of performing a complex enough maneuver to position the arm near the block and push/pull it towards the goal by chance is vanishingly small.

In contrast to the original environment with continuous actions, the action space is limited to increasing or decreasing the velocity in one of two orthogonal directions along the table by ± 0.1 or ± 1, leading to a total of 8 actions. An observation is represented by a 28-dimensional real vector that contains the following information: positions of the gripper and block; rotational and positional velocities of the gripper and block; relative position of the block with respect to

the gripper; state of the gripper; and current desired velocity of the gripper along each direction. The goal is a triple (x, y, z) representing the target position the block has to be moved to as shown by the red sphere in Figure 4.2. An episode lasts a maximum of 50 time steps or until the goal has been achieved.

4.1.4 Ms. Pac-man (ATARI)

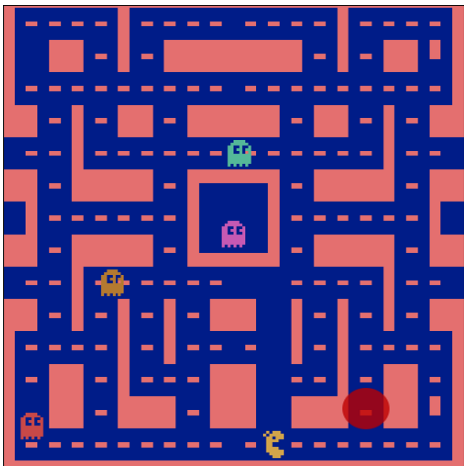

Figure 4.3. Ms. Pac-man

The *Ms. Pac-man* environment is a variant of the classic game as part of the *Arcade Learning Environment* [Bellemare et al., 2013])- a set of ATARI 2600 games popularly used to test algorithms against complex, high dimensional video input and a wide variety of environment dynamics. The Ms. Pac-man environment (see Figure 4.3) is more challenging than the previous grid world environments because of its high dimensional input as well as the presence of enemies that must be avoided or will cause the episode to prematurely end on collision.

The original environment was modified to make it suited for goal-conditional policies. The game's frame is partitioned into a 14×19 grid and goals locations are randomly chosen, non-initial (x, y) tuples on this grid. Observations are the input frames of the game as shown in the Figure 4.3 after some preprocessing as described below. Actions include those that move the agent in the four cardinal directions for 13 game ticks. An episode ends when the agent reaches a goal or is captured by a ghost, or reaches the maximum allowable time-steps of 28.

4.2 Experimental Protocol

This section provides an overview of the experimental protocol used to evaluate goal-conditional policies. We discuss the metrics used for performance evaluation of the policies as well as the sensitivity of the hyperparameters. We also discuss which hyperparameters were tuned through a grid search as our procedure for picking the best performing agent that was representative of the method's performance for conclusive experiments. We note that this experimental protocol is the same as that discussed in [Rauber et al., 2017].

4.2.1 Evaluation Metric

With the goal of measuring performance and sample complexity of goal-conditional policies, we choose to evaluate performance with *learning curves* and *average performance* scores which are obtained as follows. After collecting a small batch number of trajectories and goals, the agent undergoes a *training step* which enables one step of gradient ascent. After a number of such training steps, the agent undergoes an *evaluation step*. During evaluation, the agent interacts with the environment for a number of episodes selecting actions greedily (selecting actions with the highest probability) according to its policy. A learning curve is then plotted showing the average return obtained during each evaluation step, averaged across multiple runs (where each run is an independent learning procedure with a different random seed). The curves are also plotted with a 95% bootstrapped confidence interval. The average performance is given by the average return across evaluation steps, averaged across runs.

There exist a number of other possible evaluation metrics such as that of *success rate* used in [Andrychowicz et al., 2017], which measures the percentage of episodes the agent was able to achieve its goal within some collection of training steps. However, this only measures *if* the agent solved the task and is not a measure of *how well* it learnt to solve it, which, in reality, is what the agent is trying to optimize. We believe the average performance scores and learning curves are more indicative of such progress.

In addition to the learning curves and average performance scores, which give us an idea about the agent's learning performance, we also assess the difficulty in finding a good set of hyperparameters as an indicator of the method's stability. We do this through *hyperparameter sensitivity plots* which presents the average performance of every set of parameters (sorted in decreasing order of performance) obtained during the hyperparameter search procedure described below. Ideally, for a given environment, the average performance should decrease gradually and by as small a magnitude as possible as the hyperparameter combinations get from best to worst. This would indicate that the method is not as sensitive to the exact setting and it is easier to get reasonable and stable performance out of it.

4.2.2 Hyperparameter Search

For each environment and method, there exist a number of tunable hyper-parameters which may have a significant impact on the performance of the agent. These include decisions about the network architecture (network type, number of layers, number of neurons in each layer, activation functions, weight initialization, optimizer, learning rates, etc.) and the algorithm itself (number of trajectories to collect before an update, choice of baseline, etc.). An exhaustive search of every one of these combinations is computationally expensive.

We chose to fix some of these values (such as the network architecture, optimizer, initialization, etc.), choosing reasonable values depending on the environments and use grid search to select hyperparameters for each estimator according to their average performance score (after the corresponding standard deviation across runs is subtracted, as suggested by [Duan et al., 2016]). *Definitive results* are obtained by using the best hyperparameters found for each estimator in additional runs.

4.2.3 Experimental settings

In every experiment, the policy is represented by a feedforward neural network with a softmax output layer. The input to such a policy is a pair composed of state and goal. A value function is represented by a feedforward neural network with a single (linear) output neuron if approximating a state-value function or equal to the number of actions if approximating an action-value function. The input to a value function is a triple composed of state, goal and time step. Parameters for both policies and value functions are updated using Adam [Kingma and Ba, 2014]). Network architectures are fixed depending on the environments and are further elaborated on below.

Bit flipping environments and grid world environments.

Both policy and baseline networks have two hidden layers, each with 256 hyperbolic tangent units. Every weight is initially drawn from a Gaussian distribution with mean 0 and standard deviation 0.01 (and redrawn if far from the mean by two standard deviations), and every bias is initially zero.

Ms. Pac-man environment.

The policy network is represented by a convolutional neural network. The network architecture is given by a convolutional layer with 32 filters (8×8, stride 4); convolutional layer with 64 filters (4×4, stride 2); convolutional layer with 64 filters (3×3, stride 1); and three fully-connected layers, each with 256 units. Every unit uses a hyperbolic tangent activation function. Every weight is initially set using variance scaling [Glorot and Bengio, 2010]), and every bias is initially zero.

A sequence of images was obtained from the Arcade Learning Environment and is preprocessed in a fashion similar to that done in [Mnih et al., 2015]. Individually, for each colour channel, an element-wise maximum operation is employed between two consecutive images to reduce rendering artefacts. Such a $210 \times 160 \times 3$ preprocessed image is converted to grayscale, cropped, and rescaled into an 84×84 image x_t. A sequence of images $x_{t-12}, x_{t-8}, x_{t-4}, x_t$ obtained in this way is *stacked* into an $84 \times 84 \times 4$ image, which is an input to the policy network (recall that each action is repeated for 13 game ticks). The goal information is concatenated with the *flattened* output of the last convolutional layer.

FetchPush environment.

The policy network has three hidden layers, each with 256 hyperbolic tangent units. Every weight is initially set using variance scaling [Glorot and Bengio, 2010], and every bias is initially zero.

Tables 4.1 and 4.2 document the experimental settings. The number of runs, training batches and batches between evaluations is reported separately for hyperparameter search and definitive runs. The number of training batches is adapted according to how soon each estimator leads to apparent convergence. Note that it is very difficult to establish this setting before hyperparameter search. The number of batches between evaluations is adapted so that there are 100 evaluation steps in total. Other settings include the sets of policy and baseline learning

rates under consideration for hyperparameter search, and the number of active goals subsampled per episode. In Tables 4.1 and 4.2, $\mathscr{R}_1 = \{\alpha \times 10^{-k} \mid \alpha \in \{1, 5\} \text{ and } k \in \{2, 3, 4, 5\}\}$ and $\mathscr{R}_2 = \{\beta \times 10^{-5} \mid \beta \in \{1, 2.5, 5, 7.5, 10\}\}$.

As mentioned in Section 4.2.2, the definitive runs use the best combination of hyperparameters (learning rates) found for each estimator. Every setting was chosen during preliminary experiments to ensure that the best result for each estimator is representative. In particular, the best performing learning rates rarely lie on the extrema of the corresponding search range. In the single case where the best performing learning rate found by hyperparameter search for a goal-conditional policy gradient estimator was such an extreme value (FetchPush, for a small batch size), evaluating one additional learning rate lead to decreased average performance.

Table 4.1. Experimental settings for the bit flipping and grid world environments

	Bit flipping (8 bits)		Bit flipping (16 bits)	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
Runs (definitive)	20	20	20	20
Training batches (definitive)	5000	1400	15000	1000
Batches between evaluations (definitive)	50	14	150	10
Runs (search)	10	10	10	10
Training batches (search)	4000	1400	4000	1000
Batches between evaluations (search)	40	14	40	10
Policy learning rates	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$
Baseline learning rates	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$
Episodes per evaluation	256	256	256	256
Maximum active goals per episode	∞	∞	∞	∞

	Empty room		Four rooms	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
Runs (definitive)	20	20	20	20
Training batches (definitive)	2200	200	10000	1700
Batches between evaluations (definitive)	22	2	100	17
Runs (search)	10	10	10	10
Training batches (search)	2500	800	10000	3500
Batches between evaluations (search)	25	8	100	35
Policy learning rates	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$
Baseline learning rates	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$	$\mathscr{R}_1$
Episodes per evaluation	256	256	256	256
Maximum active goals per episode	∞	∞	∞	∞

Table 4.2. Experimental settings for the Ms. Pac-man and FetchPush environments

	Ms. Pac-man		FetchPush	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
Runs (definitive)	10	10	10	10
Training batches (definitive)	40000	12500	40000	12500
Batches between evaluations (definitive)	400	125	400	125
Runs (search)	5	5	5	5
Training batches (search)	40000	12000	40000	15000
Batches between evaluations (search)	800	120	800	300
Policy learning rates	$\mathscr{R}_2$	$\mathscr{R}_2$	$\mathscr{R}_2$	$\mathscr{R}_2$
Episodes per evaluation	240	240	512	512
Maximum active goals per episode	∞	3	∞	3

4.3 Experimental Results: Hindsight Policy Gradient (HPG)

In this section, we present the empirical performance of the weighted per-decision hindsight policy gradient estimator presented in Section 3.2. We investigate their performance with and without a approximate state-value function baseline (which we henceforth abbreviate as *HPG* (Eq.3.6) and *HPG+B* (Eq. 3.4)). In order to evaluate how the addition of hindsight effects performance, we compare the estimators with *Goal-Conditional Policy Gradient* estimators, again with and without a state-value function baseline (which we abbreviate as *GCPG* (Eq. 2.43) and *GCPG+B* (Eq. 2.44)).

We note that the *HPG* estimator does not precisely correspond to Equation 3.6. The original estimator requires a constant number of time steps T which, in the environments we consider, would require the agent to act *after* an episode has terminated. As discussed in Section 3.2, if the goal distribution is known exactly, it is feasible to compute exactly the expectation over goals in Equation 3.6. However, we sometimes choose to subsample the set of active goals per episode. Furthermore, when including a baseline that approximates the value function, we again consider only active goals, which results in an inconsistent estimator (*HPG+B*)

Despite the above deviations from the formulation, we see from the experimental section below that these compromised estimators lead to remarkable sample efficiency. We evaluate the *HPG, HPG+B, GCPG* and *GCPG+B* estimators in a small batch ($batch\_size = 2$) and a medium batch ($batch\_size = 16$) setting on all the environments described in Section 4.1. The evaluation protocol used is as discussed in 4.2 with a first stage of hyper-parameter search followed by a set of *conclusive experiments* which we present below.

4.3.1 Learning curves (batch size 2)

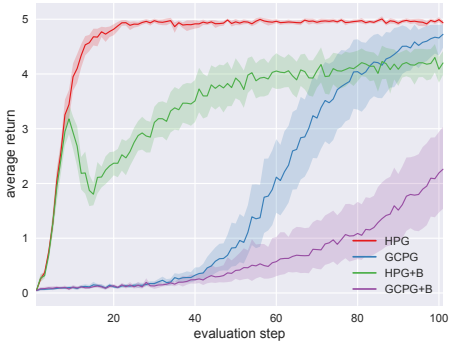

Figure 4.4. Bit flipping ($k = 8$).

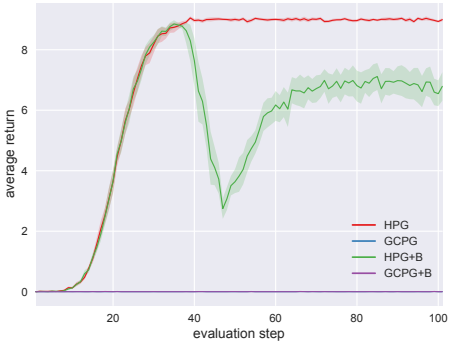

Figure 4.5. Bit flipping ($k = 16$).

Figure 4.6. Empty room.

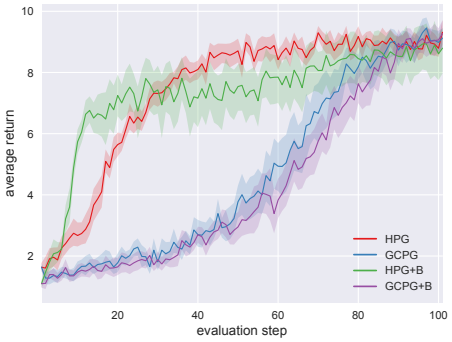

Figure 4.7. Four rooms.

Figure 4.8. Ms. Pac-man.

Figure 4.9. FetchPush.

4.3.2 Learning curves (batch size 16)

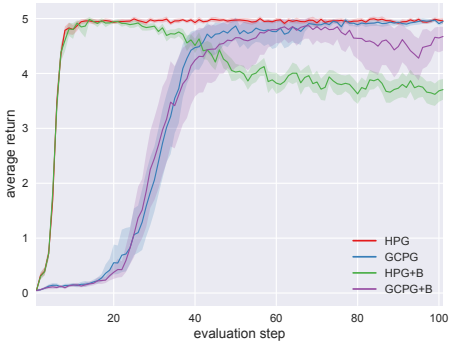

Figure 4.10. Bit flipping ($k = 8$).

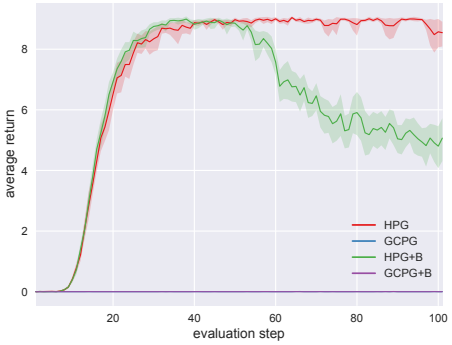

Figure 4.11. Bit flipping ($k = 16$).

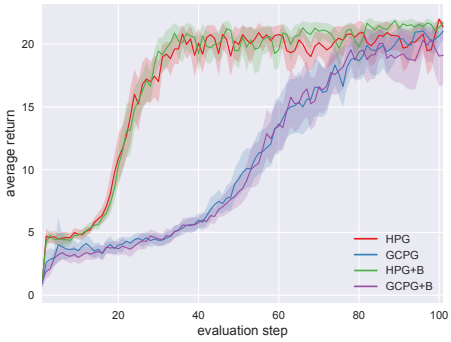

Figure 4.12. Empty room.

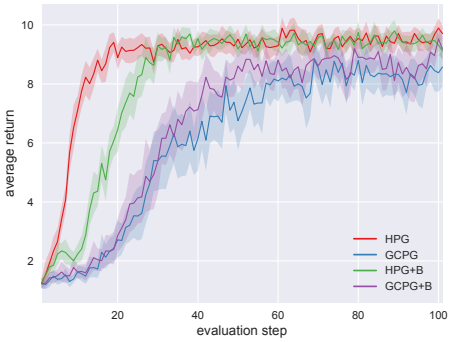

Figure 4.13. Four rooms.

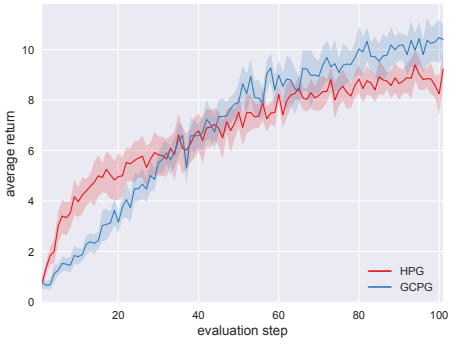

Figure 4.14. Ms. Pac-man.

Figure 4.15. FetchPush.

4.3.3 Hyperparameter sensitivity plots (batch size 2)

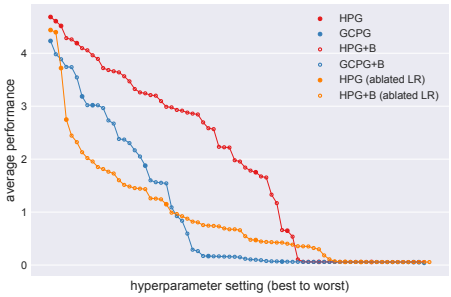

Figure 4.16. Bit flipping ($k = 8$).

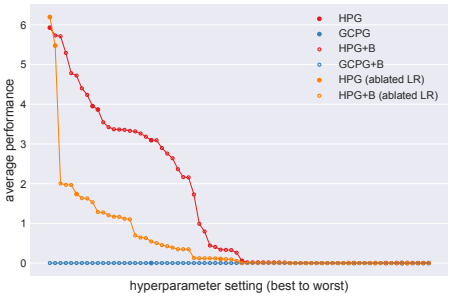

Figure 4.17. Bit flipping ($k = 16$).

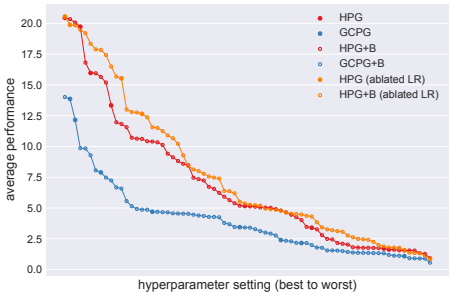

Figure 4.18. Empty room.

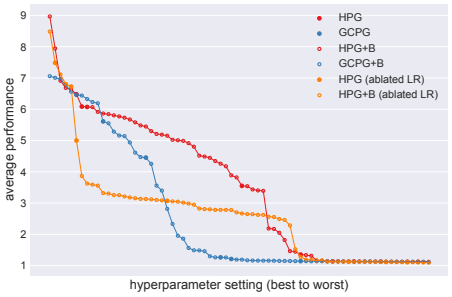

Figure 4.19. Four rooms.

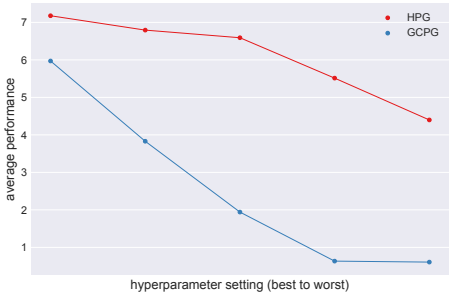

Figure 4.20. Ms. Pac-man.

Figure 4.21. FetchPush.

4.3.4 Hyperparameter sensitivity plots (batch size 16)

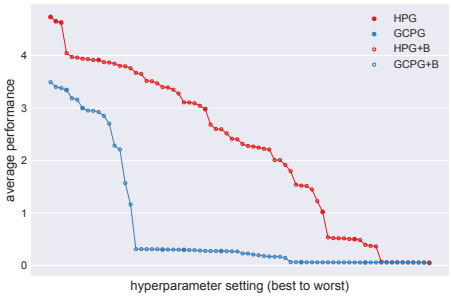

Figure 4.22. Bit flipping ($k = 8$).

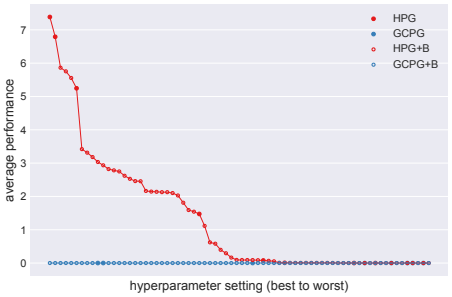

Figure 4.23. Bit flipping ($k = 16$).

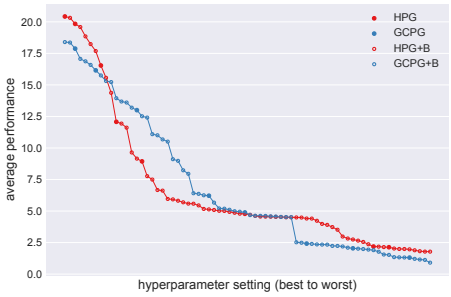

Figure 4.24. Empty room.

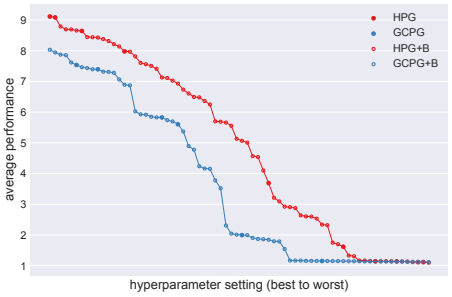

Figure 4.25. Four rooms.

Figure 4.26. Ms. Pac-man.

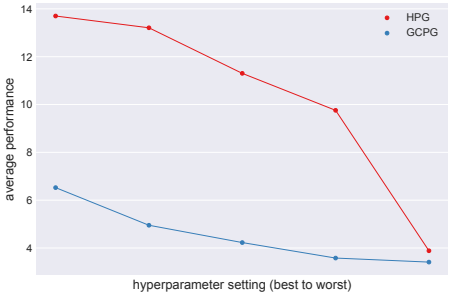

Figure 4.27. FetchPush.

Average performance results

Table 4.3 presents average performance results for every combination of environment and batch size.

Table 4.3. HPG: Definitive average performance results

	Bit flipping (8 bits)		Bit flipping (16 bits)	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
HPG	**4.60 ± 0.06**	**4.72 ± 0.02**	**7.11 ± 0.12**	**7.39 ± 0.24**
GCPG	1.81 ± 0.61	3.44 ± 0.30	0.00 ± 0.00	0.00 ± 0.00
HPG+B	3.40 ± 0.46	4.04 ± 0.10	5.35 ± 0.40	6.09 ± 0.29
GCPG+B	0.64 ± 0.58	3.31 ± 0.58	0.00 ± 0.00	0.00 ± 0.00

	Empty room		Four rooms	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
HPG	**20.22 ± 0.37**	16.83 ± 0.84	**7.38 ± 0.16**	**8.75 ± 0.12**
GCPG	12.54 ± 1.01	10.96 ± 1.24	4.64 ± 0.57	6.12 ± 0.54
HPG+B	19.90 ± 0.29	**17.12 ± 0.44**	7.28 ± 1.28	8.08 ± 0.18
GCPG+B	12.69 ± 1.16	10.68 ± 1.36	4.26 ± 0.55	6.61 ± 0.49

	Ms. Pac-man		FetchPush	
	Batch size 2	Batch size 16	Batch size 2	Batch size 16
HPG	**6.58 ± 1.96**	6.80 ± 0.64	**6.10 ± 0.34**	**13.15 ± 0.40**
GCPG	5.29 ± 1.67	**6.92 ± 0.58**	3.48 ± 0.15	4.42 ± 0.28

4.3.5 Discussion

Bit flipping environments

Figure 4.10 presents the learning curve for $k = 8$ for the medium batch setting. We see that both the goal-conditional policy gradient estimators with and without a value function baseline (*GCPG* and *GCPG+B*, respectively) learn a satisfactory policy and are comparable in performance. However, we see that the hindsight policy gradient estimators with and without state-value function baselines (*HPG* and *HPG+B*, respectively) achieve the same maximum level of performance almost 400 batches earlier than its GCPG counterparts. The HPG+B estimator (in green) degrades as it is trained further. We attribute this stability to two potential factors-the fact that the HPG+B estimator focusses solely on active goals and not the actual distribution over all goals and that the value function baseline is poorly fitted and overestimates the potential return of a $(state, goal)$ pair. The latter would, in effect lead to wrongly penalizing trajectories that in reality lead to good outcomes.

We observe similar behaviour in the small batch setting in Figure 4.4. The HPG estimator remains the most stable while the GCPG and GCPG+B estimators have not yet fully reached their level of maximum performance. We also observe that the HPG+B estimator, like before, experienced a destabilizing effect. It recovers but is unable to reach the maximum performance.

The learning curves for $k = 16$ for the medium batch setting are presented in Figure 4.11. Clearly, neither GCPG nor GCPG+B is able to learn policies better than picking an action at random. This is a direct result of being unable to reach a target configuration and obtain intermediate rewards often enough to learn any useful behaviour. Like its $k = 8$ counterpart, we observe a similar destabilizing effect for HPG+B. Figure 4.5 mirrors similar outcomes for a small batch setting.

As seen in the above settings, the HPG estimator shows remarkable improvement in sample efficiency over GCPG as confirmed by the table of average performance in Table 4.3. We also see from the hyperparameter sensitivity graphs (Figures 4.16,4.22) for $k = 8$ under the small and medium batch settings, the HPG estimator's performance for non-optimal hyperparameter combinations degrades much more slowly than GCPG's. This is an additional indicator of HPG's stability and suggests that HPG is less sensitive to the choice of hyperparameters than the other estimators.

Grid world environments

Figure 4.12 shows the learning curve for the empty room environment for the medium batch setting. We again see that all the estimators achieve satisfactory policies with both the HPG and HPG+B estimators being similar in performance and improving upon the GCPG estimators by atleast 200 batches. We also note that the HPG+B estimator does not show any signs of instability as with the bit flipping environments and in-fact, seem to aid in performance with the HPG+B estimator performing better than the HPG estimator (see Table 4.3 for $batch\_size = 16$). Similar behaviour is encountered for the small batch setting in Figure 4.6.

We observe similar behaviour in the four rooms environment as seen in Figure 4.13. Although none of the agents was able to achieve satisfactory policy, it is still clear that the HPG and HPG+B estimators perform better than GCPG and GCPG+B and are able to improve sample efficiency and peak in performance much quicker. Similar behaviour is seen for the small batch setting in Figure 4.7.

Ms. Pac-man environment

Figure 4.14 shows the learning curve for the Ms. Pac-man environment for the medium batch setting. GCPG+B and HPG+B are excluded from this comparison because of the significant computational cost of systematic hyperparameter search for these large environments. Although we see that HPG obtains a better policy early on in training, the GCPG estimator eventually overtakes it in performance. However, for a medium batch setting, we only choose 3 active goals per episode (out of the potential 28). Although this subsampling is necessary for computational reasons, it appears to significantly handicap the estimator in this case. This hypothesis is evidenced by the fact that for a small batch setting where all the active goals are used, the HPG estimator outperforms GCPG as seen in Figure 4.8 or the average performance in Table 4.3.

FetchPush environment

Figure 4.15 shows the learning curve for the medium batch setting for the modified FetchPush environment described in Section 4.1. As before, we exclude the GCPG+B and HPG+B estimators from this comparison. HPG obtains a good policy early on in the training process while the GCPG estimator appears to be slowly rising in performance. The stark difference in learning curves yet again shows how the HPG estimator is significantly more sample efficient than the GCPG estimator. It is an additional point of interest that like the Ms. Pac-man environment, the FetchPush environment also sub-sampled 3 active goals per episode (out of potentially 50) for the medium batch setting. Similar results are obtained for the small batch setting as seen in Figure 4.9.

4.3.6 Ablation Study

The section on Hyperparameter Stability for the small batch setting (Section 4.3.3) also documents an ablation study performed where the importance weights are removed from HPG (shown in orange). We notice that although the best parameter sets do indeed perform comparably to our estimators, they are extremely sensitive and very quickly drop in performance for configurations that are not close to ideal. This would seem to suggest that some useful behaviour could, indeed be learnt but would require significant effort in finding a perfect set of parameters that perform well. This study confirms that the correction offered by the importance sampling is indeed well-founded and useful.

4.4 Experimental Results: Hindsight Actor-Critic (HAC)

The section below provides an empirical evaluation of the Hindsight Actor-Critic algorithm introduced in Section 3.3. HAC combines the power of an actor (hindsight policy gradient) with that of a critic (a state-value function, in this formulation) which provides low-variance feedback about the performance of the actor instead of the potentially more noisy and high-variance empirical returns used in actor-only methods.

We evaluate its performance in a small batch setting ($batch\_size = 2$) for the bit-flipping environments and grid-world environments and a comparison of their performance against that of the hindsight policy gradient estimator with and without a value function baseline (HPG and HPG+B respectively). The GCPG and GCPG+B estimators are omitted for the sake of readability in this section. A combined result of all the methods covered in the thesis is included in the appendix (Section A.1). To further study the bias/variance trade-off between a critic trained with TD (bootstrapping) and Monte-Carlo, we implement and evaluate both the *Hindsight Actor-Critic with a Bootstrapped Critic (HAC_BCRITIC)* and the *Hindsight Actor-Critic with a Monte-Carlo Critic (HAC_MCCRITIC)*.

Experiments were carried out in accordance with the experimental protocol (Section 4.2) and the results are presented in the section below. It is to be noted that like with hindsight policy gradients, some hyperparameters that were selected for conclusive experiments were at the edge of their respective search ranges. Extra experiments were carried out with extended ranges and they did not yield further improvement.

4.4.1 Learning curves (batch size 2)

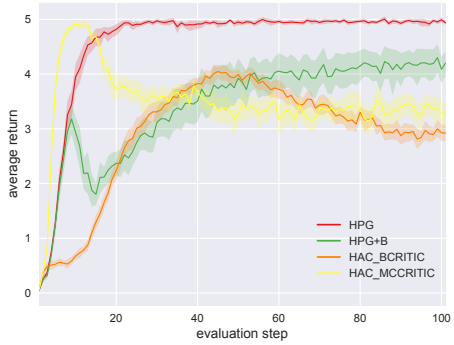

Figure 4.28. Bit flipping ($k = 8$).

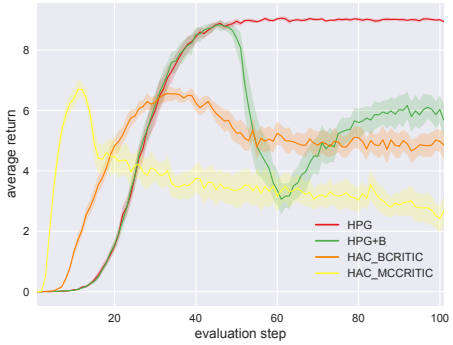

Figure 4.29. Bit flipping ($k = 16$).

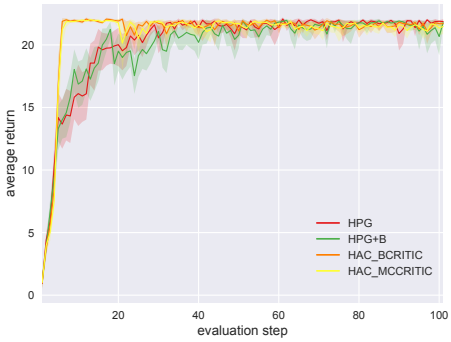

Figure 4.30. Empty room.

Figure 4.31. Four rooms.

4.4.2 Hyperparameter sensitivity plots (batch size 2)

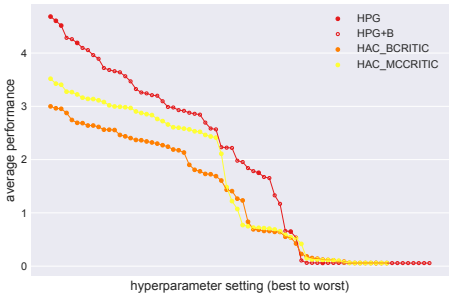

Figure 4.32. Bit flipping ($k = 8$).

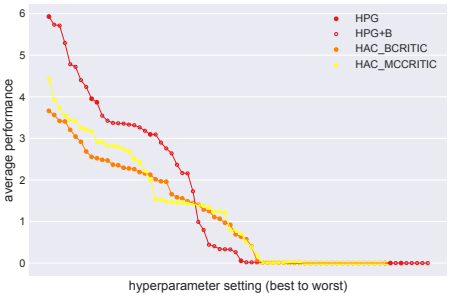

Figure 4.33. Bit flipping ($k = 16$).

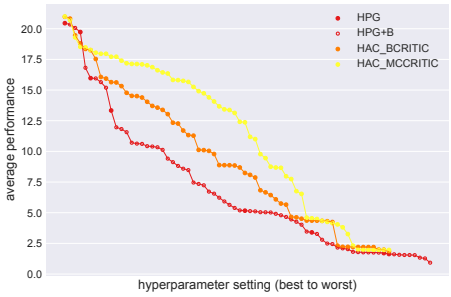

Figure 4.34. Empty room.

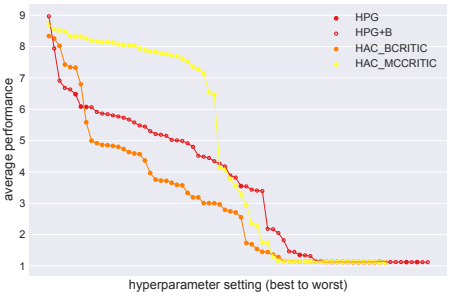

Figure 4.35. Four rooms.

4.4.3 Average performance results

Table 4.4 presents average performance results for every combination of environment and batch size.

Table 4.4. HAC: Definitive average performance results

	Bit flipping (8 bits)	Bit flipping (16 bits)
	Batch size 2	Batch size 2
HPG	**4.60 ± 0.06**	**7.11 ± 0.12**
GCPG	1.81 ± 0.61	0.00 ± 0.00
HPG+B	3.40 ± 0.46	5.35 ± 0.40
GCPG+B	0.64 ± 0.58	0.00 ± 0.00
HAC_MCCRITIC	3.48 ± 0.16	3.56 ± 0.62
HAC_BCRITIC	2.90 ± 0.15	4.65 ± 0.39

	Empty room	Four rooms
	Batch size 2	Batch size 2
HPG	20.22 ± 0.37	7.38 ± 0.16
GCPG	12.54 ± 1.01	4.64 ± 0.57
HPG+B	19.90 ± 0.29	7.28 ± 1.28
GCPG+B	12.69 ± 1.16	4.26 ± 0.55
HAC_MCCRITIC	20.83 ± 0.16	**8.12 ± 0.26**
HAC_BCRITIC	**20.92 ± 0.15**	8.10 ± 0.39

4.4.4 Discussion

Bit flipping environments

Figure 4.28 presents the learning curve for $k = 8$ for the small batch setting. We see that instability of the form seen in the HPG+B estimator occurs for both HAC estimators. It is of interest that the Monte-Carlo critic (in yellow) appears to initially learn significantly more quickly than HPG before the instability occurs. We also see from the average performance table (Section 4.4) that the HAC_MCCRITIC outperforms the HPG+B estimator on the account of its faster initial learning phase despite its divergence. It is also interesting to note that the Monte-Carlo critic outperforms the bootstrapped critic which grows much slower in performance but still suffers the instability.

As with the discussion of the HPG+B estimator's performance on the bit flipping environments presented in the previous chapter, we reiterate our belief that this is caused by a poorly fitted state-value function which results in poor estimation of performance. Similar results can be observed in the learning curve for $k = 16$ for the small batch setting in Figure 4.29. Here, both HAC estimators begin learning much quicker before they destabilize.

Grid world environments

Figure 4.30 shows the learning curve for the empty room environment for the small batch setting. It can be noticed that both the HAC estimators achieve a satisfactory policy at a much faster rate than the HPG or HPG+B estimators and stay stable as learning continues. We also note the variance in performance is much lesser in comparison throughout the length of training. Although they achieve similar performance, the HAC with a bootstrapped critic performs slightly better than with the Monte-Carlo critic as noted in the average performance table in Section 4.4.

A similar trend is noticed for the learning curve for the four rooms environment for a small batch setting shown in Figure 4.31. Both variants of the HAC estimators learn a reasonable policy (although still not optimal) much quicker than that of HPG or HPG+B thus further improving their sample efficiency. The HAC_BCRITIC estimator achieves maximum performance very early on in the training procedure and then degrades as the training continues. HAC_MCCRITIC, on the other hand, shows much more steady behaviour and continues to gradually rise in performance as the training continues and achieves the best result on this environment as seen in the average performance table (4.4).

The hyperparameter sensitivity graphs further support the efficacy of these methods for the grid world environments. We hypothesize that this is due to the value functions of the maze environments being less complex to approximate than those of bit flipping environments. Figure 4.34 presents the hyperparameter sensitivity results for the empty room environment and shows that the HAC estimators as more resilient to hyperparameter settings than HPG with HAC_MCCRITIC showing the most stability. The hyperparameter sensitivity of the four rooms environment on the other hand (Figure 4.35), shows the HAC_MCCRITIC very quickly dropping in performance and is more sensitive to parameter changes than HPG. The HAC_MCCRITIC, on the other hand, remains the most stable among them.

4.5 Experimental Results: Hindsight Actor Hindsight Critic (HAHC)

The section below provides an empirical evaluation of the Hindsight Actor Hindsight Critic algorithm introduced in Section 3.4. Like HAC, HAHC is an Actor-Critic method that uses a value function to provide low-variance performance feedback to an actor (hindsight policy gradient) with the aim of providing faster, more stable learning thereby further decreasing sample complexity. However, HAHC uses a value function critic trained with an off-policy policy evaluation method that allows it to be augmented with the capability of hindsight and enable it to incorporate additional information about values of arbitrary goals g that were achieved when attempting to achieve the original goal g'.

We evaluate its performance in a small batch setting ($batch\_size = 2$) for the bit-flipping environments and grid-world environments and compare their performance against that of the hindsight policy gradient estimator with and without a value function baseline (HPG and HPG+B respectively). The GCPG and GCPG+B estimators are omitted like before. Experiments were carried out in accordance with the experimental protocol (Section 4.2) and the results are presented in the section below. As with HAC, we implement and evaluate critics trained with both methods of TD (bootstrapping) and Monte-Carlo. The Hindsight Actor Hindsight Critic with Monte-Carlo Critic (HAHC_MCCRITIC) uses per-decision weighted importance sampling to correct for the off-policy updates whereas the Hindsight Actor Hindsight Critic with a Bootstrapped Critic (HAHC_BCRITIC) performs off-policy TD updates through a form of Expected SARSA.

Importantly, we note that our implementation does not fully capture the true learning capabilities of this formulation. As previously mentioned, the critic is capable of off-policy policy estimation and may strongly benefit from the use of Hindsight Experience Replay [Andrychowicz et al., 2017] thereby allowing the critic to perform policy evaluation of the actor on arbitrarily off-policy data. For reasons of simplicity and computational complexity, we restrict off-policy evaluation to the *active-goals* we use for updating the critic during hindsight for the results presented in this thesis.

We note here a significant deviation from the hyperparameter settings listed in Section 4.2.3. During the grid search to find an optimal set of hyperparameters, we noted that a number of combinations, particularly for the grid world environments had their critic learning rate lie on the edge of the search space and required significant expansion to lower values in order to find the best hyperparameter set. Particularly, the search range $\mathscr{R}_3 = \{\alpha \times 10^{-k} \mid \alpha \in \{1, 5\}$ and $k \in \{2, 3, 4, 5, 6, 7\}\}$ was used for the critic's learning rate. Since these extra parameters are included in the parameter sensitivity graphs (4.5.2), it results in an inordinate number of points being present for these estimators.

4.5.1 Learning curves (batch size 2)

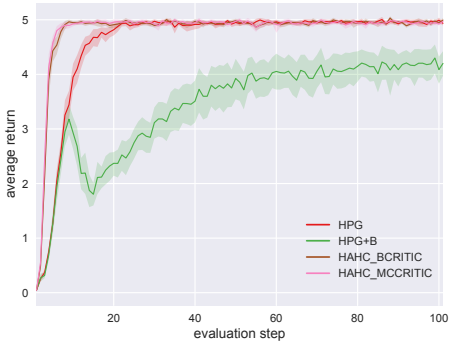

Figure 4.36. Bit flipping ($k = 8$).

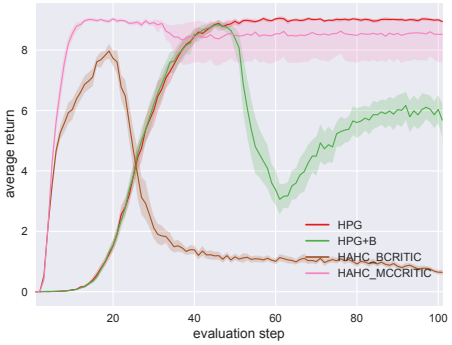

Figure 4.37. Bit flipping ($k = 16$).

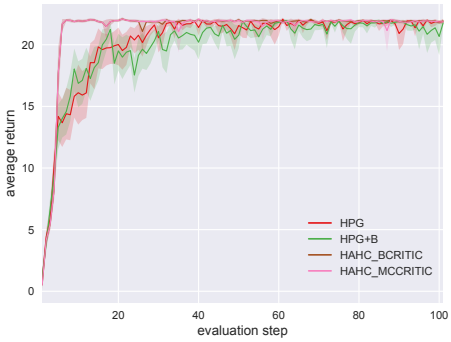

Figure 4.38. Empty room.

Figure 4.39. Four rooms.

4.5.2 Hyperparameter sensitivity plots (batch size 2)

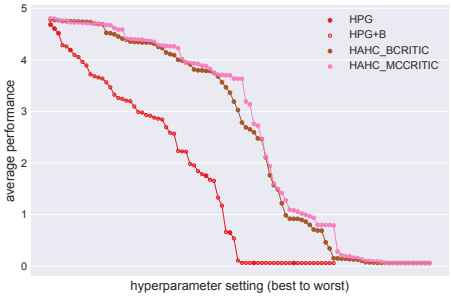

Figure 4.40. Bit flipping ($k = 8$).

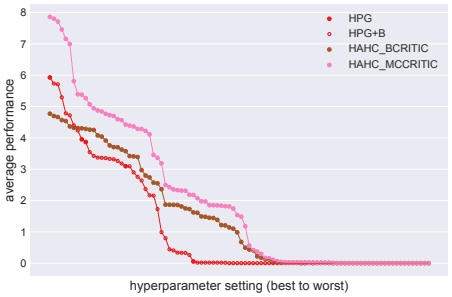

Figure 4.41. Bit flipping ($k = 16$).

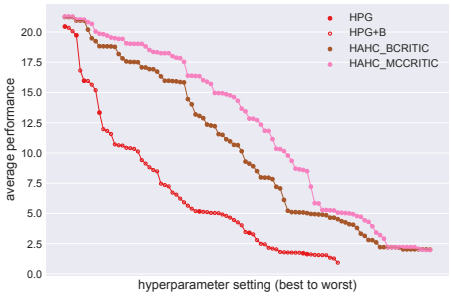

Figure 4.42. Empty room.

Figure 4.43. Four rooms.

4.5.3 Average performance results

Table 4.5 presents average performance results for every combination of environment and batch size.

Table 4.5. HAHC: Definitive average performance results

	Bit flipping (8 bits)	Bit flipping (16 bits)
	Batch size 2	Batch size 2
HPG	4.60 ± 0.06	7.11 ± 0.12
GCPG	1.81 ± 0.61	0.00 ± 0.00
HPG+B	3.40 ± 0.46	5.35 ± 0.40
GCPG+B	0.64 ± 0.58	0.00 ± 0.00
HAHC_MCCRITIC	$\mathbf{4.82 \pm 0.02}$	$\mathbf{8.19 \pm 1.20}$
HAHC_BCRITIC	4.80 ± 0.02	2.27 ± 0.23

	Empty room	Four rooms
	Batch size 2	Batch size 2
HPG	20.22 ± 0.37	7.38 ± 0.16
GCPG	12.54 ± 1.01	4.64 ± 0.57
HPG+B	19.90 ± 0.29	7.28 ± 1.28
GCPG+B	12.69 ± 1.16	4.26 ± 0.55
HAHC_MCCRITIC	21.16 ± 0.11	8.99 ± 0.07
HAHC_BCRITIC	$\mathbf{21.18 \pm 0.07}$	$\mathbf{9.04 \pm 0.07}$

4.5.4 Discussion

Bit flipping environments

Figure 4.36 presents the learning curve for $k = 8$ for the small batch setting. We see that both HAHC estimators show excellent performance and learn a reasonable policy within just a few training batches. Unlike previous estimators that included value functions (HPG+B, HAC), neither of the two estimators show any instability on further training and are able to outperform HPG.

In the learning curve for $k = 16$ for the small batch setting seen in Figure 4.37, we notice that the HAHC_BCRITIC quickly diverges similarly to what was observed with previous methods. The HAHC_MCCRITIC, on the other hand, is able to make progress and learn a reasonable policy much faster than HPG. We note the slight destabilizing effect it faces around the 35th evaluation step after which it appears to be unable to fully recover. Despite this, it achieves the highest average performance results on this environment as shown in Table 4.5 showing a remarkable improvement in sample efficiency for the bit flipping tasks.

We also note that the hyperparameter sensitivity graphs (Section 4.5.2) show both HAHC methods being significantly more stable than HPG across all environments considered. The hyperparameter sensitivity graph for the $k = 16$ bit flipping environment shows HAHC_MCCRITIC (in pink), with a significant number of parameters that outperform HPG showing very high stability, although a portion of this effect may be attributed to the HAHC having more hyperparameter combinations due to its extended search.

Grid world environments

Figure 4.38 shows the learning curve for the empty room environment for the small batch setting. It can be seen that both estimators attain excellent policies within just a few training batches, showing significant improvement in performance. We also note the variance in the learning curve is much less than that of HPG and HPG+B.

Similar results can be observed in Figure 4.39 for the four rooms environment for the small batch setting. The HAHC estimators once again very quickly obtain reasonable policies within a few training batches thereby drastically improving over HPG and HPG+B. From the average performance table (4.5), we see that the bootstrapped critic (HAHC_BCRITIC) slightly outperforms the Monte-Carlo critic (HAHC_MCCRITIC) and once again, show a marked improvement in sample efficiency for both the gridworld tasks.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have explored the idea of hindsight and its application to the setting of reinforcement learning with different goals that need to be pursued across episodes under the setting of sparse rewards. We discuss why this setting is particularly challenging, even with the application of exploration based methods and why the introduction of hindsight, fuelled by the generalization properties of goal-conditional policies form a highly effective strategy for efficient learning.

We first provide the relevant background on reinforcement learning and its extension to learning goal-conditional policies, with a brief aside on deep learning, which form the core components of the work presented in the thesis. We then discuss why existing methods in the literature fail to address the problem of learning in this setting and why hindsight would help ameliorate the excessively high sample complexity faced by existing learning algorithms.

We mention prior work that introduced the concept of hindsight to off-policy reinforcement learning algorithms that rely on experience replay [Andrychowicz et al., 2017] and policy search based on Bayesian optimization [Karkus et al., 2016]. We discuss some of the work presented in [Rauber et al., 2017], where we introduce hindsight to policy gradient methods and generalize it to a broad class of algorithms. We show that hindsight is highly beneficial to sparse-reward environments that require learning of a goal-conditional policy and that our proposed estimator shows remarkable sample efficiency across a wide variety of environments and on occasion, even being the difference between learning a policy that picks actions at random and learning the optimal behaviour policy.

Following the success of applying actor-critic architectures to improve policy gradient methods, we build on the advantage formulation of the Hindsight Policy Gradient introduced in [Rauber et al., 2017] and propose a new estimator we call the Hindsight Actor-Critic (HAC) which uses a trained state-value function in the form of a critic to provide low-variance performance feedback to the actor. We show further improvement in sample efficiency of these estimators in some environments and posit that these estimators work well when both the actor and the critic are able to perform well. Working towards this hypothesis, we introduce

another actor-critic based method where we equip the critic with the capability of hindsight and allow it to train the value-function for an actor for arbitrary goals achieved when trying to achieve another. We call this novel method the Hindsight Actor Hindsight Critic (HAHC) estimator and demonstrate its striking improvement in sample efficiency over Hindsight Policy Gradient methods across all environments tested.

One of the main drawbacks of the Hindsight Policy Gradient method (as discussed in [Rauber et al., 2017]) is that of its computational cost, which is directly related to the number of active goals in a batch. This issue also rings true for the Hindsight Actor-Critic and more so for the Hindsight Actor Hindsight Critic where the critic also incurs additional training cost for every additional active goal in a batch. This issue may be mitigated by sub-sampling active goals, which generally leads to inconsistent estimators. Fortunately, experimental results show that this is a viable alternative for Hindsight Policy Gradient estimators. An unprincipled focus on active goals while fitting value functions has been hypothesized as the reason for unstable learning in Hindsight Policy Gradient estimators using baselines as well as the actor-critic variants we propose. Further investigation of this phenomenon is required.

5.2 Future Work

There are many possibilities for future work for both HPG and HAHC, besides integrating them into systems that rely on goal conditional policies: assessing whether they can successfully circumvent catastrophic forgetting during curriculum learning of goal-conditional policies, approximating the reward function to reduce the supervision required, evaluating the methods proposed in a dense reward setting, analysing the variance of the estimators empirically, studying the effect of active goal sub-sampling, evaluating the techniques proposed on continuous action spaces and comparing with techniques based on hindsight experience replay.

While initial results attained by the HAHC are promising, they still require extensive testing in more challenging environments such as Ms. Pac-man and the FetchPush environment and for larger batch sizes under much harsher sub-sampling of active goals. The effects of training HAHC critics using data that is more off-policy such as with Hindsight Experience Replay is also a question open for future work. Further improvements that are standard in Actor-Critic or Reinforcement Learning literature such as the use of n-step updates for both the actor and critic [Mnih et al., 2016], addition of entropy regularization terms to the loss function [Mnih et al., 2016], sharing of weights of convolutional layers between the actor and the critic are all independent to our method and can be applied.

A.1 Unified Results of HPG, HAC and HAHC for a small batch setting

The section below provides an empirical evaluation of all the methods discussed in this thesis for a small batch setting ($batch\_size = 2$) for the bit-flipping environments and grid-world environments.

Learning curves (batch size 2)

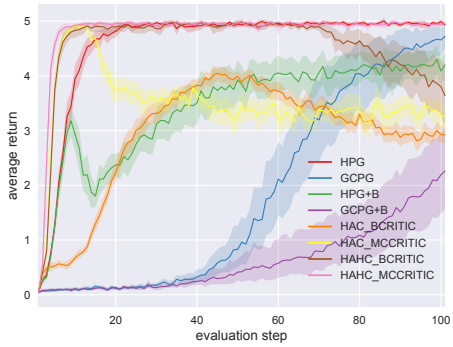

Figure 1. Bit flipping ($k = 8$).

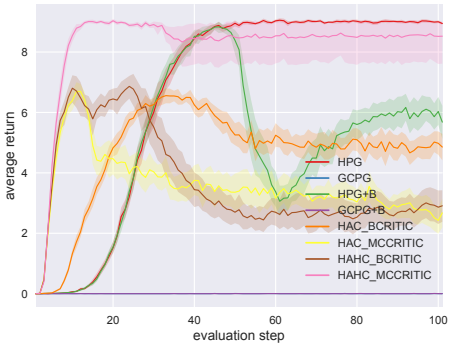

Figure 2. Bit flipping ($k = 16$).

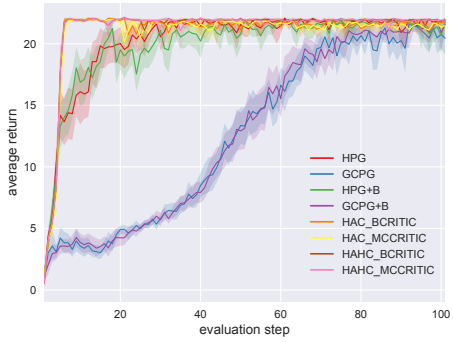

Figure 3. Empty room.

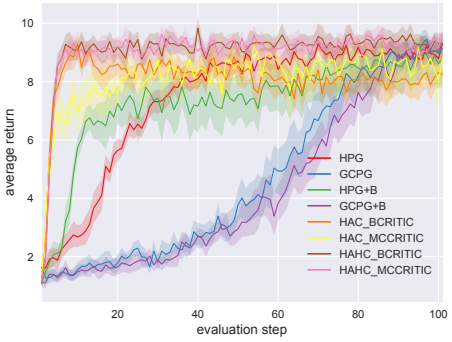

Figure 4. Four rooms.

Hyperparameter sensitivity plots (batch size 2)

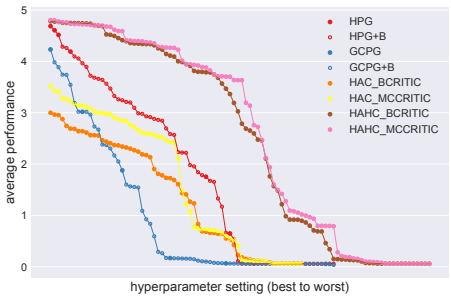

Figure 5. Bit flipping ($k = 8$).

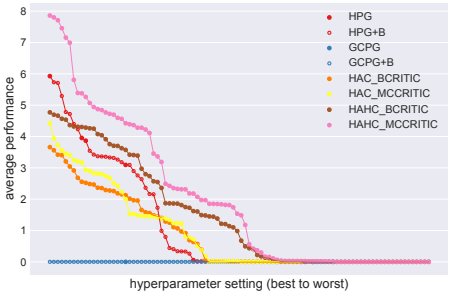

Figure 6. Bit flipping ($k = 16$).

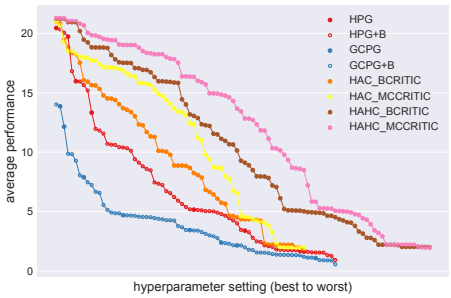

Figure 7. Empty room.

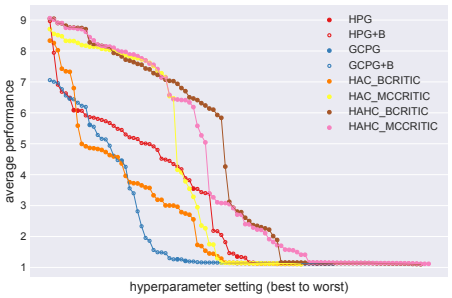

Figure 8. Four rooms.

Average performance results

Table 1 presents average performance results for every combination of environment and batch size.

Table 1. Unified: Definitive average performance results

	Bit flipping (8 bits)	Bit flipping (16 bits)
	Batch size 2	Batch size 2
HPG	4.60 ± 0.06	7.11 ± 0.12
GCPG	1.81 ± 0.61	0.00 ± 0.00
HPG+B	3.40 ± 0.46	5.35 ± 0.40
GCPG+B	0.64 ± 0.58	0.00 ± 0.00
HAC_MCCRITIC	3.48 ± 0.16	3.56 ± 0.62
HAC_BCRITIC	2.90 ± 0.15	4.65 ± 0.39
HAHC_MCCRITIC	$\mathbf{4.82 \pm 0.02}$	$\mathbf{8.19 \pm 1.20}$
HAHC_BCRITIC	4.80 ± 0.02	2.27 ± 0.23

	Empty room	Four rooms
	Batch size 2	Batch size 2
HPG	20.22 ± 0.37	7.38 ± 0.16
GCPG	12.54 ± 1.01	4.64 ± 0.57
HPG+B	19.90 ± 0.29	7.28 ± 1.28
GCPG+B	12.69 ± 1.16	4.26 ± 0.55
HAC_MCCRITIC	20.83 ± 0.16	8.12 ± 0.26
HAC_BCRITIC	20.92 ± 0.15	8.10 ± 0.39
HAHC_MCCRITIC	21.16 ± 0.11	8.99 ± 0.07
HAHC_BCRITIC	$\mathbf{21.18 \pm 0.07}$	$\mathbf{9.04 \pm 0.07}$

Bibliography

Alejandro Gabriel Agostini and Enric Celaya Llover. Reinforcement learning with a gaussian mixture model. In *2010 International Joint Conference on Neural Networks*, pages 3485–3492, 2010.

Artemij Amiranashvili, Alexey Dosovitskiy, Vladlen Koltun, and Thomas Brox. Analyzing the role of temporal differencing in deep reinforcement learning. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=HyiAuyb0b`.

Sigrún Andradóttir, Daniel P Heyman, and Teunis J Ott. On the choice of alternative measures in importance sampling with markov chains. *Operations research*, 43(3):509–519, 1995.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In F. Groen et al., editor, *Proc. 8th Conference on Intelligent Autonomous Systems IAS-8*, pages 438–445, Amsterdam, NL, 2004. IOS Press.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE Publ. Piscataway, NJ, 1995.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, 2013. ISBN 9788132209065.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

B. C. Da Silva, G. Konidaris, and A. G. Barto. Learning parameterized skills. In *Proceedings of International Conference of Machine Learning*, 2012.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Kristopher De Asis, J Fernando Hernandez-Garcia, G Zacharias Holland, and Richard S Sutton. Multi-step reinforcement learning: A unifying algorithm. *arXiv preprint arXiv:1703.01327*, 2017.

M. P. Deisenroth, P. Englert, J. Peters, and D. Fox. Multi-task policy search for robotics. In *IEEE International Conference on Robotics and Automation, 2014*, pages 3876–3881, 2014.

T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of International Conference on Machine Learning*, pages 1329–1338, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

A. Fabisch and J. H. Metzen. Active contextual policy search. *The Journal of Machine Learning Research*, 15(1):3371–3399, 2014.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 482–495, 13–15 Nov 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pages 1514–1523, 2018.

David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49 (2-3):325–346, 2002.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *International Conference on Learning Representations*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Peter W Glynn. Likelilood ratio gradient estimation: an overview. In *Proceedings of the 19th conference on Winter simulation*, pages 366–375. ACM, 1987.

Shixiang Gu, Tim Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3846–3855, 2017.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

Leslie Pack Kaelbling. Learning to achieve goals. In *IN PROC. OF IJCAI-93*, pages 1094–1098. Morgan Kaufmann, 1993.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

Peter Karkus, Andras Kupcsik, David Hsu, and Wee Sun Lee. Factored contextual policy search with bayesian optimization. *arXiv preprint arXiv:1612.01746*, 2016.

Michael J Kearns and Satinder P Singh. Bias-variance error bounds for temporal difference updates. In *COLT*, pages 142–147. Citeseer, 2000.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pages 1401–1407, 2013.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948*, 2017.

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.

L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3/4):69–97, 1992a.

Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992b. UMI Order No. GAX93-22750.

A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

Daniel J Mankowitz, Augustin Žídek, André Barreto, Dan Horgan, Matteo Hessel, John Quan, Junhyuk Oh, Hado van Hasselt, David Silver, and Tom Schaul. Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*, 2018.

Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. Bayesian optimization for contextual policy search. In *Proceedings of the Second Machine Learning in Planning and Control of Robot Motion Workshop., Hamburg*, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. Technical Report arXiv:1312.5602 [cs.LG], Deepmind Technologies, Dec 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

J. Moody and Lizhong Wu. Optimization of trading systems and portfolios. In *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pages 300–307, March 1997. doi: 10.1109/CIFER.1997.618952.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, 06–11 Aug 2017.

Deepak Pathak, Parsa Mahmoudieh, Michael Luo, Pulkit Agrawal, Dian Chen, Fred Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Patrick M Pilarski, Michael R Dawson, Thomas Degris, Farbod Fahimi, Jason P Carey, and Richard S Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–7. IEEE, 2011.

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.

Doina Precup, Richard S Sutton, and Satinder P Singh. Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*, pages 759–766, 2000.

Martin L Puterman. Markov decision processes. j. *Wiley and Sons*, 1994.

Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

Paulo Rauber, Filipe Mutz, and Juergen Schmidhuber. Hindsight policy gradients. *arXiv preprint arXiv:1711.06006*, 2017.

M. B. Ring. Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In *Machine Learning: Proceedings of the Eighth International Workshop*, pages 343–347. Morgan Kaufmann, 1991.

T. Rückstieß, M. Felder, and J. Schmidhuber. State-Dependent Exploration for policy gradient methods. In W. Daelemans et al., editor, *European Conference on Machine Learning (ECML) and Principles and Practice of Knowledge Discovery in Databases 2008, Part II, LNAI 5212*, pages 234–249, 2008.

Gavin Adrian Rummery. *Problem solving with reinforcement learning*. PhD thesis, University of Cambridge Ph. D. dissertation, 1995.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *Proceedings of the International Conference on Machine Learning*, pages 1312–1320, 2015.

J. Schmidhuber. Learning to generate sub-goals for action sequences. In *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland, 1991.

J. Schmidhuber. Deep learning in neural networks: An overview. Technical Report IDSIA-03-14 / arXiv:1404.7828v1 [cs.NE], The Swiss AI Lab IDSIA, 2014.

J. Schmidhuber and R. Huber. *Learning to Generate Focus Trajectories for Attentive Vision*. Institut für Informatik, 1990.

F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Policy gradients with parameter-based exploration for control. In *Proceedings of the International Conference on Artificial Neural Networks ICANN*, 2008.

Ajay Seth, Jennifer L Hicks, Thomas K Uchida, Ayman Habib, Christopher L Dembia, James J Dunne, Carmichael F Ong, Matthew S DeMers, Apoorva Rajagopal, Matthew Millard, et al. Opensim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLoS computational biology*, 14(7):e1006223, 2018.

David Silver. Introduction to Reinforcement Learning (Slides), 2015. URL `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf`.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

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, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

R. K. Srivastava, B. R. Steunebrink, and J. Schmidhuber. First experiments with PowerPlay. *Neural Networks*, 41(0):130 – 136, 2013. Special Issue on Autonomous Learning.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999a.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999b.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1497–1504, 2008.

John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16 (3):185–202, 1994.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 177–184. IEEE, 2009.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3540–3549, 06–11 Aug 2017.

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

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 538–545, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1. URL `http://dl.acm.org/citation.cfm?id=2074022.2074088`.

Steven D Whitehead and Dana H Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1998a.

M. A. Wiering and J. Schmidhuber. Efficient model-based exploration. In J. A. Meyer and S. W. Wilson, editors, *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 6*, pages 223–228. MIT Press/Bradford Books, 1998b.

R. J. Williams. Reinforcement-learning in connectionist networks: A mathematical analysis. Technical Report 8605, Institute for Cognitive Science, University of California, San Diego, 1986.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pages 3357–3364, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.